INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Manifold Constraints: A Paradigm for Connectionist Computation

By

Daryl Herbert Graf, B.Sc., M.C.S.

A thesis submitted
to the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science

Carleton University

Ottawa Ontario

March 1998

© Copyright

1998, Daryl Herbert Graf

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-32335-8

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

**Manifold Constraints: A Paradigm for
Connectionist Computation**

submitted by
Daryl Herbert Graf, B.Sc., M.C.S.

in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Director, School of Computer Science

Thesis Supervisor

External Examiner

Carleton University

March 17, 1998

Manifold Constraints: A Paradigm for Connectionist Computation

Copyright 1998

by

Daryl Herbert Graf

For my wife.

Pauline Edith Graf.

with love.

Acknowledgements

First and foremost. I would like to thank my advisor Professor B. John Oommen without whom this thesis would not be. John brings to his graduate students more than just depth of technical expertise, clarity of thought, and astonishing organizational capability. Just as importantly, he offers genuine interest, dedication and warmth — qualities that make research a rewarding and worthwhile human experience. I also owe John a debt of gratitude for his indefatigable efforts to secure funding for me and to *make things happen*. Thank you for everything John. You are an inspiration.

I would also like to thank Professor Evangelos Kranakis and the School of Computer Science for their patience and generous financial support. More than once Professor Kranakis has made it possible for me to complete this thesis.

Thanks also to the faculty of the School of Computer Science. In particular, I would like to say a special thanks to Professor Franz Oppacher (who taught me AI and how to repel down a cliff face), Professor Frank Dehne and Professor Frank Fiala for many rewarding insights and lots of much needed encouragement during this degree.

To the school staff, and especially Marlene Wilson, Rosemary Carter, and Barbara Coleman, I want to say a special thank you. I cannot count the times that you have been there for me.

Special thanks are due to The Object People, most especially to John Pugh and Paul White for their unlimited patience and financial and moral support. I would also like to thank the National Science and Engineering Research Council for financial assistance.

Abstract

In our thesis we develop techniques that permit a general class of continuous-time neural networks to learn a partition of activation space into surfaces (manifolds) on which the network dynamics must evolve. These surfaces determine both the controllability and observability characteristics of the network. We call these surfaces *manifold constraints*.

We present manifold constraints as a new paradigm for understanding what a neural network can and should learn in order to perform a particular class of computations. Our goal is to provide the theorist and network designer with another powerful tool, that can be used in conjunction with the idea of energy landscapes, to enrich the run-time behaviour of continuous-time neural networks. Manifold constraints are particularly well suited to problem domains in which differentiable manifolds can be identified with a measure of cost, quality, or importance.

In the first part of the thesis, we explore the problem of adaptively learning observability and controllability manifolds for a very general class of continuous-time neural networks that we call Grossberg Normal Form (GNF) nets. We discover that, in general, learning algorithms do NOT exist for robust learning of controllability manifolds in the presence of internal noise. We find that several important special case networks do permit robust learning algorithms, however, the class of learnable manifolds is very limited in each of these cases. In addition, we show that these results have a dual counterpart for the problem of learning observability manifolds.

In the second part of the thesis, we show how to modify the GNF neuron equations with compensating terms that constrain the network dynamics to the desired activation space manifolds. We also examine the more general problem of learning a rich class of nonlinear manifold constraints. We introduce a novel technique called Principal Manifold Approximation (PMA) and develop a connectionist

realization of this method. We then show how PMA learning can be applied to create a novel class of holonomic networks which obey the learned manifold constraints. We conclude with a discussion of how holonomic networks can be applied to the problem of learning type constraints on variable bindings in connectionist representations of structured knowledge.

Contents

List of Figures	xiii
1 Introduction	1
1.1 The Ubiquity of Manifold Constraints	2
1.1.1 Constraint Manifolds Occur in Many Domains	3
1.1.2 Constraint Manifolds as Computational Abstractions	4
1.1.3 Constraint Manifolds and Time-Varying Inputs	9
1.1.4 Constraint Manifolds for Dynamical Models of Cognition	10
1.2 Overview of the Dissertation	10
1.3 Contributions of the Thesis	12
1.4 What this Dissertation Does Not Describe	15
1.5 On Notation	17
2 Grossberg Normal Form Networks	18
2.1 Connectionism and Control	18
2.1.1 Implications of the Control Theoretic Viewpoint	19
2.2 The GNF Family of Neural Networks	21
2.2.1 Grossberg Normal Form	21
2.2.2 A Sufficient Condition for Stability in GNF Systems	28
2.2.3 Special Cases of the GNF Family	29
2.3 Chapter Conclusions	39
3 Nonlinear Control Theory	40
3.1 An Introduction to Control	40
3.1.1 A Brief History	41
3.2 The State Space View of Control	43
3.2.1 Control = Injecting Inputs to Overcome Drift Dynamics	44
3.3 Reachability Manifolds and Controllability	45

3.3.1	Local Reachability Analysis	46
3.3.2	Global Reachability Analysis	52
3.4	Observability and the Equivalence of Initial Values	54
3.4.1	Local Observability Analysis	54
3.4.2	Global Observability Analysis	55
3.4.3	Interactions Between Reachability and Observability Manifolds	56
3.5	The Relative Degree Normal Form	56
3.5.1	Vector Relative Degree	58
3.5.2	Finding the Coordinate Transformation	60
3.5.3	Deriving the Normal Form	61
3.5.4	What Does the Normal Form Mean?	62
3.6	Feedback Control of Dynamics	63
3.6.1	Output Zeroing Dynamics	66
3.6.2	Exact Linearization of System Dynamics	67
3.6.3	Asymptotic Stabilization	68
3.6.4	Asymptotic Tracking	70
3.6.5	Decoupling Feedback	72
3.7	Implications of the Theory for GNF Networks	73
3.7.1	The Implications of Controllability for GNF Networks	74
3.7.2	Implications of Observability for GNF Networks	75
3.7.3	Implications of Feedback for GNF Networks	75
3.8	Chapter Conclusions	79
4	The Reachability Properties of GNF Networks	80
4.1	Contributions of the Chapter	82
4.2	The Approach to the Analysis	85
4.2.1	Grossberg Normal Form (GNF) Networks	85
4.2.2	The Relationship of Reachability to Manifold Constraints . . .	85
4.2.3	Absolutely Holonomic Networks	86
4.2.4	Reachability Analysis	87
4.3	Reachability Results for Open GNF Networks	88
4.3.1	Supporting Definitions and Lemmas	90
4.3.2	Results for General Open GNF Networks	111
4.3.3	General GNF Networks are Generically Nonholonomic	116
4.4	Results for Classical Neural Network Models	118
4.4.1	Shunting Inhibition Networks	118
4.4.2	Bidirectional Associative Memories, Hopfield Networks, BSB Networks, and the McCulloch-Pitts Model	134
4.5	Reachability Results for Closed GNF Networks	139

4.5.1	Supporting Definitions and Lemmas	140
4.5.2	Results for Closed GNF Networks	142
4.6	Chapter Conclusions	151
5	The Observability Properties of GNF Networks	153
5.1	Contributions of the Chapter	154
5.2	GNF Observability is Dual to GNF Reachability	155
5.3	Observability of Open GNF Networks with Output Summation Fields	156
5.4	Observability of Closed GNF Networks with Output Summation Fields	165
5.5	Chapter Conclusions	168
6	Enforcing Manifold Constraints by Additive Compensation	169
6.1	Contributions of the Chapter	171
6.2	Modularity of Manifold Constraints	172
6.3	Modular Linearizing Transforms	174
6.4	Modular Invariance	176
6.4.1	Modular Reachability	178
6.4.2	Enforcing Modular Invariance with Additive Compensation . .	180
6.4.3	Enforcing Manifold Constraints by Additive Compensation . .	186
6.5	Adaptive Learning of Linear Reachability Constraints	190
6.5.1	A Compensated Continuous-Time Feed Forward Network . . .	191
6.5.2	A Law for Simultaneous Learning of Fixed Points and Reach- ability Constraints	194
6.6	Chapter Conclusions	196
7	Principal Manifold Approximation	198
7.1	Contributions of the Chapter	199
7.2	The Manifold Approximation Problem	200
7.2.1	Manifold Approximation is Underconstrained	201
7.3	Principal Manifold Approximation	211
7.3.1	Vector Representations for PMA	211
7.3.2	Local Approximation by Principal Components Analysis	215
7.3.3	Global Approximation by Regularized Functions	221
7.3.4	Summary of Principal Manifold Approximation	235
7.4	Chapter Conclusions	236
8	Connectionist Principal Manifold Approximation	238
8.1	Contributions of the Chapter	239
8.2	Connectionist Architectures for PMA	241
8.2.1	Connectionist PMA Learning	243

8.2.2	Oja-Style PCA Algorithms	244
8.2.3	Derivation of Batch PMA Learning Rule	246
8.3	Holonomic Networks	257
8.4	A Potential Application of Holonomic Networks: Connectionist Types	260
8.4.1	Representing Symbolic Structures	260
8.4.2	Connectionist Types	265
8.5	Application of PMA to Nonholonomic Networks	269
8.5.1	Adaptive Compensation for Nonlinear Constraints	272
8.6	Chapter Conclusions	275
9	Conclusions and Directions for Future Research	277
9.1	Directions For Future Work	280
9.1.1	Empirical Studies	280
9.1.2	Reducing the Number of Neurons	281
9.1.3	Other Families of Mappings	281
9.1.4	Complete Characterization of Regularized RBF Inverse	282
9.1.5	Discrete-Time BR-PMA	282
9.1.6	Transporting Manifold Constraints Between Networks	282
9.1.7	Learning Multiple Environmental Foliations by Competitive PMA	283
9.2	Manifold Constraints for Continuous-time Pattern Recognition	284
A	A Philosophy and History of Connectionism	288
A.1	The Origins of Connectionism	288
A.2	A Brief History of Connectionist Models	290
A.3	A Generic Connectionist Architecture	297
A.3.1	Neuroscience Basics	297
A.3.2	A Simple Mathematical Model	299
A.3.3	Learning	304
B	Differential Geometry Review	307
B.1	Functions	308
B.1.1	Smooth Functions	308
B.1.2	Analytic Functions	308
B.2	The Jacobian	308
B.2.1	Inverse Function Theorem	308
B.2.2	Rank Theorem	309
B.3	What is a Manifold?	309
B.4	Tangent Spaces	310
B.4.1	The Problem With Calculus on General Manifolds	310

B.4.2	The Solution to the Problem	310
B.5	Tangent Vectors	311
B.6	Tangent Space	311
B.6.1	The Basis Vector Fields of a Tangent Space	311
B.7	Numeric Calculations	312
B.8	Manifold Mappings and Tangent Space Mappings	314
B.8.1	Manifold Embeddings	314
B.9	Vector Fields	315
B.9.1	Ordinary Differential Equations on Nonlinear Manifolds	315
B.9.2	Lie Derivatives	315
B.9.3	Lie Brackets	315
B.10	Distributions	316
B.10.1	Involutive Distributions	317
B.11	Completely Integrable Distributions and Frobenius' Theorem	318
B.11.1	Invariant Distributions	318
B.12	Covector Fields and Codistributions	319
C	VISICON—A Simulation Library for the Visualization of Controlled Neurocomputers	321
C.1	Motivation	322
C.2	Design Philosophy	322
C.2.1	VISICON is Object Oriented	322
C.2.2	VISICON Strives for Numerical Accuracy	325
C.2.3	VISICON is Portable	327
C.2.4	Error Handling in VISICON	327
C.3	The VISICON Class Abstraction	328
C.3.1	Scalar Data Types	329
C.3.2	Class Vector<Scalar_Type>	331
C.3.3	Class Matrix<Scalar_Type>	335
C.3.4	Class Vector_Field	344
C.3.5	Class Distribution	358
C.3.6	Class Manifold	362
C.3.7	Class Integrator	366
C.3.8	Class GNF_Net	372
C.3.9	Class Plotter	378
C.3.10	The Utility Classes	379
	Bibliography	380

List of Figures

1.1	The Energy Landscape Abstraction of Computation	6
1.2	The Manifold Constraint Abstraction of Computation	7
2.1	A System of Interconnected GNF Networks	24
2.2	Output Summation Field Network	27
3.1	Lie Brackets Are Not Linear Combinations of the Input Fields	48
3.2	Δ_* Locally Foliates the Plant's State Space	49
3.3	System Dynamics as Drifting 'Slice'	51
3.4	Observability Manifolds Partition Reachability Manifolds	57
3.5	Two Module Feedback Architecture for Learning Dynamics Consistant with a Constraint Manifold	76
3.6	Imposing Multiple Dynamics on a Manifold	78
4.1	Length d Open GNF Network Multi-Lie bracket Decomposition Tree.	99
4.2	Multi-Lie Bracket Leaf Derivation Tree for $\mathcal{L}_{\mathbf{g}_{p_4}} \mathcal{L}_{\mathbf{g}_{p_3}} \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1}$	103
4.3	Evaluation Tree for Expansion of $\mathbf{g}_{p_1}^{(r_1)} \odot \mathbf{g}_{p_2}^{(r_2)} \odot \cdots \mathbf{g}_{p_d}^{(r_d)}$	109
6.1	Dynamics of Modular Invariance	179
7.1	Learning System for Manifold Approximation Problem	201
7.2	The Environmental Foliation - Three leaves (manifolds) from a 2- dimensional foliation of $\mathbb{R}^3$ and six behavioural events. Notice that the events are constrained to evolve on only one manifold - they can- not "hop" from one to another. They may intersect themselves and each other.	202
7.3	The Dynamic Region $\mathcal{U}$ - $\mathcal{U}$ is the region in which the events occur. The foliation should be accurate here. $\mathcal{U}$ is uniformly sampled.	203

7.4	Sparse Sampling Ambiguity - (a) Do the sparse samples in region R tell us the foliation is...(b) 1-dimensional or (c) 2-dimensional?	206
7.5	Approximating The Tangent Spaces on a Covering of $\mathcal{U}$	208
7.6	Gaussian Noise in the Velocity Estimates	210
8.1	Constraint Linearizing Network	242
8.2	Regularized PMA Network	242
8.3	Regularized PMA Network	245
8.4	Holonomic Network Architecture	258
9.1	Competitive PMA Network	283
A.1	Typical Connectionist Network Architectures	300
A.2	Sigmoidal Output Function	302

Chapter 1

Introduction

Connectionist computation holds a certain fascination for the artificial intelligence community. It exhibits computational characteristics that are attractive, and in some ways complementary to those of more traditional forms of AI. Connectionism is also interesting in its own right because it offers an alternative way of looking at computation that leads to novel operators. How best to combine connectionist and symbolic operations in hybrid AI systems is currently an actively debated topic of research. Indeed, theoretical results suggest that connectionist networks are powerful enough to act as stand-alone Turing-complete computers.

Despite high levels of interest and suggestive theoretical results, the types of computations that connectionist networks typically perform are quite limited. This is in part due to the lack of a comprehensive theory of connectionist-style computation. In turn, this has hindered the development of a set of intuitively appealing computational abstractions to help designers crystallize what a network learns, and how this knowledge can be used at run-time. We believe that, to exploit the full capabilities of connectionist systems, we must improve our theoretical understanding and use our knowledge to develop better conceptual tools.

It is the goal of this thesis to contribute to the *theory* of connectionist computation, and to suggest a new computational abstraction that we call *manifold*

constraints. This abstraction encourages the designer to think of computational constraints as nonlinear surfaces (manifolds) in activation space on which the dynamics of a connectionist system is constrained to evolve. In the next section, we will see that these types of constraints are naturally present in many domains of interest. We view manifold constraints as one more useful conceptual tool in the network designer’s toolbox. It is our hope that this tool set will one day become as rich as that available for conventional digital computers.

In particular, our goal is to describe a system that retains many of the desirable characteristics of traditional connectionist networks while being able to learn and enforce constraint manifolds. We are not concerned with biological plausibility here, but rather with exploring how control concepts can broaden connectionist computational capabilities. Indeed, the systems that we develop might be better classified as ‘hybrid connectionist/control systems’, while they simultaneously retain the desirable characteristics of conventional connectionist networks. Along the way, we hope to introduce the reader to useful analytical tools from the theory of differential geometric state space control.

We view our work as foundational in that it is, to our knowledge, the first effort to develop learning algorithms and architectures specifically for learning foliations of constraint manifolds. The methods that we develop under the manifold constraint abstraction should be seen as an initial pass at the problem that will benefit from additional development and empirical study.

1.1 The Ubiquity of Manifold Constraints

Before we embark on our study, we shall discuss why we have chosen this particular computational abstraction. Indeed, in this brief sub-section, we offer some motivation for viewing sub-manifolds of a network’s activation space as useful computational constraints.

1.1.1 Constraint Manifolds Occur in Many Domains

Manifolds naturally occur in diverse problem domains where they play an important role in constraining the behaviours of the computational system. Whenever a problem domain includes a set of objects that can be given a metric topology¹ there are invariably submanifolds of this space that contain objects with properties of special interest.

An important example occurs when physical laws place constraints on a part of the domain that is to be controlled or observed by the computational system. In this case, the set of legal configurations of the domain often corresponds to a differentiable manifold. This manifold must be modeled by the computational system. As an example, consider the problem of planning a sequence of legal state changes (in any applicable problem) that take the domain from an initial configuration to a desired configuration. From the computational viewpoint this corresponds to generating a trajectory through configuration space that is confined to a constraint manifold. Robotics offers an important real-world domain that exhibits this type of structure when planning collision-free movements of multi-joint manipulators [Mni96], [ML94], controlling compliant robotic hands [LC93], guiding wheeled autonomous vehicles [PK94], [KP92], and maintaining a pattern of gait in walking machines [eZMP96], [BOU94]. The dual problem of recognizing that a sequence consists of legal configurations is equivalent to checking that the sequence is confined to a constraint manifold. We consider the tasks of learning a model manifold from observations of domain behaviours, and applying the model to trajectory generation as the subjects of this dissertation.

In other domains the goals of the computational system impose smooth functions of cost or importance over a vector space of measured variables. In these cases a set of measurements that have a particular value (e.g. zero cost) define one or more manifolds. Process control domains are typical examples in which a function

¹More generally, it is enough that subsets of objects exist which have a countable basis, the Hausdorff property and an atlas (see Appendix B).

measures degree of risk [Fro90]. A chemical process controller must ensure that the plant does not wander out of a low risk safety zone which is a constraint manifold. Another source of manifolds occurs in domains where measurement introduces a mixture of dependent and independent variables. In this case, the implicit function theorem (refer to Appendix B) tells us that the kernel of a function defined on n variables, where $n - d$ variables are dependent, is a d -dimensional manifold (also see [Doo90]). When the cost functions are not explicitly available to the computational system it may be necessary to infer associated manifolds directly from observations of the dynamic environmental trajectory. The methods that we develop in the sequel can be applied to this task.

Occasionally, a domain has an accompanying theory that implies the existence of distinguished submanifolds of a domain space. The computational objectives of the system can sometimes be described in terms of confinement within these manifolds. Many important domains make use of the state space theory of control to describe computational objectives in this way. Examples include flight control in aircraft, spacecraft stabilization, noninteractive control of complex machinery [Bat94], target tracking [You93], disturbance decoupling for noisy communication systems, and many others (see [Isi89] and [NvdS90]). Indeed, we believe that it is highly advantageous to explicitly model these manifolds, and we provide techniques to do so in this dissertation.

1.1.2 Constraint Manifolds as Computational Abstractions

A *computational abstraction* is a method of description that permits us to talk about how a computation proceeds, and what is computed, in terms that hide the architectural complexity of the machine. A good abstraction has intuitive appeal and promotes visualization of the essential structure of the computation. Furthermore, there is a formal connection between the abstract level of description and the low level architectural details of the network that permits a sound implementation of

the computation from a description given in terms of the abstraction. Although it is beneficial, and often the only way to proceed, it is not enough to merely have a heuristic argument that the computation will proceed as described by an abstraction. Since abstractions are at a higher level of description than the network architecture, there is usually more than one way to implement them. Alternatively, the same implementation may be described by more than one abstraction, i.e. from more than one point of view.

At the lowest level of description a connectionist computation must be described in terms of individual synaptic weights and their effect on the transfer functions of particular neurons. While this level of description is as fundamental as the bit patterns that encode a symbolic computer program, it is equally difficult for humans to manipulate and reason about. Relatively few abstractions have been developed for connectionist computation, most of these remain at a fairly low level of description.

One of the most successful connectionist abstractions is the energy landscape. In this abstraction we consider a ‘hilly’ surface defined over a vector space (that is determined by the domain). Figure 1.1 illustrates such a landscape on $\mathbb{R}^2$. The vector space models by the activation space of the network. The state of the computation is represented by the vector directly below the point at which a ball sits on the landscape. Given an initial vector, the network performs its run-time computation by rolling the ball down the landscape in the direction of steepest descent. When the ball comes to rest at the locally lowest point on the landscape, the corresponding vector is the output value of the computation. Hopfield networks, Bidirectional Associative Memories, Brain-State-in-a-Box networks, Boltzmann Machines, and other models have all been designed to support this type of computational abstraction. Hopfield and Tank have devised networks to solve associative memory retrieval [Hop84], find heuristic solutions to the Traveling salesman problem [HN90], and problems involving VLSI layout using this abstraction. Model theoretic deduction for evidential reasoning has been described in terms of energy landscapes by

Figure 1.1: The Energy Landscape Abstraction of Computation

Derthik [Der91] and Pinkas [Pin94]. Jagota has employed energy landscapes to accomplish representation and error correcting recall of structured knowledge [Jag94]. Various algorithms in computational geometry are easily described in terms of energy landscapes (see [Val92]). Parsing [WP85] and lexical ambiguity resolution [Mii93] are also naturally described in this way.

Just as assembly languages allow the programmer to describe a computation in terms of control structures (e.g. loops, conditionals, subroutines, etc.), connectionist abstractions (such as energy landscapes) permit us to raise the level of description

Figure 1.2: The Manifold Constraint Abstraction of Computation

to that of geometric structures that ‘guide’ or ‘shape’ the run-time dynamics of the network. It seems reasonable to expect that a better understanding of abstractions at this level will facilitate building even higher level, and more powerful, methods of description. Like energy landscapes, constraint manifolds are a type of computational abstraction that facilitate thinking about what a connectionist network is learning and how that knowledge is used at run-time. The structure that is learned is a sub-manifold of a vector space which imposes a control constraint on the run-time computation by confining computed trajectories to the vectors in the manifold. By imposing different manifolds at different times, and combining the manifolds in different ways, the computational dynamics can be shaped to varying degrees of precision (see Figure 1.2).

The geometric nature of energy landscapes and constraint manifolds gives them a visual appeal that aids intuition. Lest the reader underestimate the importance of this point, consider that Hopfield’s introduction of the energy landscape abstraction² [Hop84] was one of the main reasons for the remarkable resurgence of

²Hopfield borrowed the notion of energy landscapes from the mathematical study of Lyapunov

interest that connectionism has enjoyed after an equally long period of neglect³. All it took was an appealing abstraction to demystify connectionist computation and facilitate its application in engineering and AI. We introduce constraint manifolds as another visually appealing geometric abstraction.

More importantly, geometric structures give the network designer a well understood language that can be used to map particular classes of problems onto the low-level mechanics of a connectionist net. For example, energy landscapes are described by energy functions. These functions are parameterized by the weights of a symmetric recurrent network and have the network activation space as their domain. A constraint satisfaction problem is mapped onto the network by assigning output variables to the network neurons, and defining energy function terms that are minimized by correct solutions. The constraints on the problem appear in these terms as the values of synaptic weights. At run-time the network neurons update the activation state in such a way that it follows the gradient of the energy landscape, and ‘settle’ into a solution that minimizes the terms in the corresponding energy function.

While the energy landscape abstraction permits the designer to think in terms of constraint satisfaction, constraint manifolds naturally support the notion of maintaining a *computational invariant*. By a computational invariant we mean some aspect of a computation, usually representing a coherent unit of data, that must be maintained by part of the network despite the influence of other parts of the system. For example, in Chapter 8 we shall show how the problem of enforcing type invariants on connectionist variable bindings can be described in terms of constraint manifolds. Indeed, explicit representation of structured data in connectionist memories (as represented by conjunctive coding schemes such as Smolensky’s tensor product representations [Smo91], and convolution-based memories [Pla94]) leads naturally to differentiable manifolds that represent relations on part of the structure.

³See [RM86a] for a good account of the historical ups and downs of connectionist computation.

ture. Maintaining the truth of these relations (another computational invariant) at run-time corresponds to enforcing a manifold constraint.

1.1.3 Constraint Manifolds and Time-Varying Inputs

As we have already suggested in Section 1.1.1, many domains have time varying components that may appear as inputs to a computation. Time varying inputs are also of considerable interest when connectionist networks are modular components of larger systems, and they must react to the dynamic behaviours of other modules that they are connected to. Computational abstractions such as the energy landscape paradigm are not well suited to these domains, because driving inputs dynamically change the shape of the landscape. Although many connectionist models (usually based on recurrent network architectures) have recently been developed to perform computations on time series, they often do not have an accompanying abstraction to which they are related to in a formal manner.

Due to the above, developing abstractions to describe how the network computation is affected by changing inputs is an important research goal. As we shall see in Chapter 4 constraint manifolds are ideal candidate abstractions. This is the case because the very general class of networks that we study in this dissertation belong to a class of autonomous nonlinear dynamical systems that can only be driven to reach states that lie on a differentiable sub-manifold of the state space. This fundamental result tells us that constraints on a network's behaviour that are invariant to changing inputs are indeed exactly manifold constraints. This is a point that we discuss at length in Chapters 3 and 4. The fact that this class of networks includes many classical connectionist models means that the manifold constraint abstraction is of general interest. Clearly, our techniques for learning and enforcing manifold constraints will have utility when the network inputs are time-varying.

1.1.4 Constraint Manifolds for Dynamical Models of Cognition

Recently, the cognitive science community has become interested in modeling cognitive processes as dynamical systems⁴. As Jaeger [Jae97] and MacLennan [Mac94] have argued, we need to develop concepts from the theory of dynamical systems that are continuous analogs of symbolic data because certain aspects of cognition clearly have both a dynamical and symbolic component. In Jaeger's words these phenomena need to be *intrinsically isolatable*, i.e. they need to have identifiable boundaries, and *intrinsically classifiable*, i.e. they need to have enough structure to permit classification. Since trajectories confined to constraint manifolds have clearly defined boundaries in state space, they are intrinsically identifiable dynamical phenomena. In addition, we shall see in Chapter 8 that there are simple mechanisms that permit their classification, and therefore they meet the condition for intrinsic classifiability. Since they possess such powerful properties, it is our thesis that manifold constraints can play an important role in bridging the gap between continuous dynamics and symbols.

1.2 Overview of the Dissertation

The dissertation is divided into two parts: (i) a control theoretic analysis that shows that the problem of learning manifold constraints cannot be solved in practice for unaugmented neural networks, and (ii) a development of hybrid connectionist/control systems and learning methods that do solve the problem.

The first part begins with Chapter 2, where we introduce the reader to the main conceptual and mathematical tools that form the background of our study⁵. This includes a description of the Grossberg Normal Form (GNF) nets, a family of

⁴For example, Busemeyer and Townsend [BT95] have argued that decision making is more accurately modeled by evolution equations than by algorithms, and Kelso [Kel95] has shown that certain properties of finger movements are predicted by a simple class of differential equation.

⁵In addition, we have provided Appendix A which surveys the history and basic principles of connectionism for the novice.

continuous-time connectionist models that we have selected to motivate our work. Chapter 3 gives a survey of the relevant control theoretic ideas and tools that we use throughout the subsequent chapters, and a description of where our work fits into the literature on connectionism and control.

In Chapters 4 and 5 we ask two questions that are central to an understanding of the dynamics of any nonlinear, input-driven system. The first, called the controllability problem, asks: "Which states can the system attain for a general class of input signals?" The second, the dual problem of observability, asks: "How much of the internal system dynamics is visible to an external observer, and how much is hidden?" In Chapter 4 we explore the problem of adaptively learning controllability manifolds for GNF nets. We discover that, in general, learning algorithms do NOT exist for robust learning of controllability manifolds. We also find that several important special case networks do permit robust learning algorithms; however, the class of learnable manifolds is very limited in each of these cases. In Chapter 5 we show that these controllability results have a dual counterpart for the problem of learning observability manifolds.

Chapter 6 opens the second part of the dissertation. Our purpose here is to examine how control techniques can be used to augment GNF nets in a way that permits successful learning of controllability and observability manifolds. We begin by showing how to modify the neuron equations with compensating terms that constrain the network dynamics to the desired activation space manifolds. (Our approach is illustrated by developing an algorithm that permits simultaneous learning of *linear* manifold partitions and energy landscape equilibria in a continuous-time associative memory.)

In the next two chapters, we conduct a study of the more general problem of learning a rich class of *nonlinear* manifold constraints. We introduce a novel technique called Principal Manifold Approximation (PMA) in Chapter 7 and develop a connectionist realization of this method in Chapter 8. We then show how PMA learning can be applied to the linear compensator of Chapter 6 to enforce nonlin-

ear controllability constraints in GNF networks and we develop a novel class of continuous-time recurrent neural networks, called Holonomic networks, that have dynamics which are naturally confined to an adaptively learned set of manifold constraints. We conclude with a discussion of how these networks can be applied to the problem of learning type constraints on variable bindings in connectionist representations of structured knowledge.

The final chapter summarizes possible extensions to the approach and other opportunities for future research. It then wraps up the dissertation with a summary of results and general conclusions.

1.3 Contributions of the Thesis

In all brevity, we would like to highlight the salient contributions of this thesis. Each of these results have led to one or more potential publications, and references to these are also included below. Indeed, the details of these results and their relevance to the existing ‘state-of-the-art’ will be found in the corresponding chapters.

In Chapter 4 we perform a study of the control properties of the general class of autonomous, continuous-time Grossberg Normal Form (GNF) networks that we introduce in Chapter 2. This family includes many classical run-time architectures. The traditional control theoretic view is that restrictions on controllability are an annoyance to the designer, and are to be avoided if possible. Therefore, the usual approach is to show conditions under which full controllability is guaranteed. However, since the context of our study is to impose specific limitations on the reachable set of states, our intentions and results are diametrically opposed to the few studies of reachability in connectionist systems that have been published⁶.

We succeed in establishing how the feedback weights and input connection

⁶For example, see [Son97] for a recent discussion of conditions that guarantee full controllability in a simpler class of continuous-time networks.

weights are related to a set of vector fields that span a reachability distribution. In particular, we have given a detailed characterization of the reachability distribution for seven related families of GNF networks. This characterization is given in terms of a set of vector fields that span the reachability distribution.

Our results also clearly expose the different contributions that are made by the adaptive and non-adaptive terms of the neuron equations to the controllability properties of the networks. We conclude that the non-adaptive terms have a significant role to play in shaping controllability, and that the network designer can make use of our results to make a sensible choice of these terms.

Besides providing results for the general family of GNF networks we also provide results for Bidirectional Associative Memory, Hopfield/Tank models, continuous-time back propagation networks, the Brain-State-in-a-Box model and McCulloch-Pitts networks. For each of these models we provide results for three different mixtures of synaptic valences. To our knowledge, this is the first study of network reachability to include the effects of differing mixtures of neuron valences.

As a result of our analysis we are able to characterize each family of networks as either holonomic or non-holonomic. The main conclusion that we are able to draw from our analysis is that practical learning algorithms do not exist for the nonholonomic families of GNF networks. We have shown that the holonomic families of GNF networks do support robust manifold constraints; however, our results indicate that the classes of constraint manifolds that can be enforced in these networks are so limited as to be of little use in most application domains.

In Chapter 5 we establish a ‘syntactic’ duality between the equations that define the reachability manifolds of GNF networks, and equations that partition the reachability manifold into classes of states that are computationally equivalent to an observer of the network output dynamics.

In Chapter 6 we characterize the form of an additive compensator that can enforce a manifold constraint in the subdynamics of any subset of neurons. The compensator requires that the equations of neural dynamics be modified. We illustrate

our results by showing how a version of Kohonen's novelty filter can be incorporated into the weight-update equations for a continuous time back propagation network to simultaneously learn associative memories and linear manifold constraints.

Chapter 7 describes the first method for explicitly learning a local partition of activation space into general nonlinear manifold constraints from sparse, noisy velocity estimates — Principal Manifold Approximation (PMA). This algorithm can therefore learn any finite number of non-overlapping nonlinear constraint manifolds from observations of a dynamic environment. We describe a novel method for modeling locally completely integrable distributions that is particularly well suited for implementation in connectionist networks, because the distribution can be represented by a vector of weights or activation values. Manifold constraints can therefore be stored as long-term control information, or can be loaded into short-term memory by associative recall. Therefore, we may also regard PMA as offering an entirely novel form of run-time connectionist control knowledge. PMA is also easily implemented as a non-connectionist, numerical algorithm, and we provide an example of such an algorithm to aid intuition.

Our approach is to use local tangent space dynamics to drive the global weight space dynamics. This technique is of interest in its own right, since it permits us to generalize theoretical results for the local dynamics to the global setting. We prove that the only stable fixed points of the BR-PMA algorithm are correct learning solutions, and therefore, our algorithm does not suffer from the problem of local minima. Furthermore, it is guaranteed to produce a good estimate of the foliation in a local neighbourhood of each observed domain state. We also show that convergence will 'almost certainly' occur from any initial weight vector. Finally, we show that the BR-PMA algorithm has the desirable property that any degree of accuracy can be achieved if enough hidden neurons are added to the network and we are willing to observe the environment long enough.

We describe a modified feed-forward shared-weight RBF network architecture, called a PMA network, that supports BR-PMA learning. PMA nets are a

connectionist module that may be incorporated into fully connectionist or hybrid connectionist/control systems to support manifold constraint learning.

We also describe a new class of holonomic connectionist architectures that support manifold constraint learning. The recurrent output layer consists of continuous-time neurons with decay rates determined by linear combinations of the inputs and the outputs of a PMA network. We refer to these as holonomic nets. We also describe a hybrid connectionist/control system that employs a PMA network to extend the compensator of Chapter 6 to include nonlinear manifolds. The evolution equations of this system is non-local; however, its close relationship with GNF networks permits it to retain many of the desirable properties of these connectionist systems.

1.4 What this Dissertation Does Not Describe

Our interests are of a theoretical nature, and, although we develop learning methods and architectures for acquiring and enforcing manifold constraints, we do not describe empirical simulations of our learning methods. Time has not permitted us to provide such studies in sufficient detail, and so we leave these for a future time or to others who are so inclined.

On the other hand, we wish to emphasize that the algorithms developed in this thesis are easily implemented on digital computers. That is to say, we do not develop results which are purely abstract and without practical application in a computational setting. In fact, we have already developed a comprehensive library of C++ classes, called VISICON, that is designed specifically for the task of simulating connectionist/control systems and modeling a rich set of differential geometric structures that underlie the theory of nonlinear state-space control. This dissertation contains a detailed description of VISICON. However, to avoid distractions from the main focus of the thesis we have included it in an appendix (Appendix C). A paper [Gra98n], which describes VISICON, is also being compiled. VISICON provides a complete infrastructure for direct implementation of our results.

We have chosen to study connectionist networks that have continuous-time dynamics rather than the more frequently used discrete-time models. This is partly because continuous-time systems are, in a sense, a more general class of dynamical system than discrete-time systems [Coo94], but it is also because continuous-time connectionist models are less frequently studied and piqued our curiosity. Another consideration that motivated our choice was the availability of a richer theory of control for continuous-time nonlinear systems than their discrete-time counterparts. As we shall point out in Chapter 9, the continuous-time learning algorithms that we develop have straightforward discrete-time implementations. However, even in this case we assume that the network run-time dynamics are continuous-time.

Unlike almost all studies of recurrent networks, we are not interested in learning a specific network dynamics, i.e. a particular congruence that produces trajectories or equilibria corresponding to the goals of a problem domain. Instead, we develop methods to learn general constraints that can be satisfied by an entire family of dynamics. This is analogous to the way that a single declarative constraint does not specify a particular computational path that a PROLOG program will take. In a sense, ours is a search for tools to ‘sculpt’ the general shape of the network dynamics without specifying detailed trajectories.

We would like to make it clear from the outset, however, that our approach is entirely compatible with the goal of learning to generate particular trajectories that conform to a manifold constraint. Indeed, we indicate how these two goals can be combined in Chapter 3, and we describe in detail an algorithm that performs both types of learning in Chapter 6. As we shall see in Chapter 3, there are good theoretical reasons to separate the learning of general manifold constraints from the learning of detailed dynamics.

1.5 On Notation

Throughout this dissertation we will follow the mathematical notation most commonly used in the nonlinear control literature. In particular, we follow the conventions established by Isidori [Isi89], and by Nijmeijer and van der Schaft [NvdS90], in their influential textbooks.

Subscripted lower case Roman letters will be used to indicate scalar coefficients or elements of vectors, e.g. x_i . Scalar functions will be appropriately parameterized, e.g. $f(\mathbf{x})$. Occasionally, we will use Greek symbols to highlight scalars that have a special purpose to play in the discussion. Bold font symbols indicate a vector value (e.g. $\mathbf{x}$) or a vector field (e.g. $\mathbf{f}(\mathbf{x})$). The parameterization of a vector field might be dropped when it is clear from context that the object is a field of vectors. We will occasionally use subscripts to distinguish between different vectors or vector fields.

Constant matrices (e.g. $\mathbf{M}$) and function matrices (e.g. $\mathbf{M}(\mathbf{x})$) are represented by bold upper case Roman letters. Bold upper case Greek letters represent differential geometric distributions, e.g. $\Delta(\mathbf{x})$ or $\beta(\mathbf{x})$. The parameter will usually be dropped when we are referring to the entire distribution as a mathematical object rather than to the subspace that is its value at a particular point.

Derivation of $\mathbf{x}$ with respect to time is indicated both by $\frac{d}{dt}\mathbf{x}$ and by the dot notation $\dot{\mathbf{x}}$. A symbol to take special note of is $\odot$, which stands for the entry-wise (Hadamard) vector product. Repeated Hadamard products are represented by a variation of the usual Pi notation, e.g. $\odot_{i=1}^n$.

Chapter 2

Grossberg Normal Form Networks

The purpose of this chapter is to introduce the reader to the family of continuous-time neural networks that motivate our work — Grossberg Normal Form (GNF) networks. We direct the novice reader first to Appendix A for a brief discussion of the history and basic concepts of connectionist computation, and then to one of the many excellent reviews of connectionist learning algorithms such as [Gol96] or [RM86a]. After describing the GNF network model in Section 2.2.1, we show how this family generalizes many of the classical continuous-time connectionist run-time architectures in Section 2.2.3.

2.1 Connectionism and Control

During the 1960's, while the AI community investigated issues of adaptation and nonlinear dynamics, essentially the same issues were being investigated by the control systems community. The mathematical approach taken by the control theorists was rather different from that taken by the connectionists because they were motivated by the control of complex dynamics rather than classification or rea-

soning¹. Still, with the clear view of hindsight we can now see that connectionist research would have benefited from an early application of the powerful, general purpose theorems developed for system control. Given connectionism's historical origin in the control community, it is ironic that the two cultures had become so closed to each other.

To a great extent, this schism still exists today. Control theorists are certainly more aware of connectionist models. In fact, connectionism is currently a trendy subject in the control journals. But many control theorists still view connectionist models as black-box parameter identification techniques, rather than an end unto themselves. They are not motivated to apply control techniques to make connectionist systems better general purpose computers, or better models of human reasoning. Similarly, most researchers in the AI community, even those working on subsymbolic systems, are not aware of even basic results from control theory. It is our desire that this thesis help bridge the gap by illustrating how control techniques can be a useful addition to the connectionist toolbox.

2.1.1 Implications of the Control Theoretic Viewpoint

As we will see in Section A.3, connectionist networks are a special family of nonlinear dynamical systems. Computations performed by a network are interpretations of its run-time dynamics. Inputs to the computation are encoded in the initial state of the system and/or in external input signals that drive the dynamics. When we say that the network learns, we mean that it modifies the parameters that control its dynamical behaviour such that, at run-time, the dynamics are constrained to produce useful computations.

One of the most important open problems facing the study of connectionist systems is how to realize the computational power of which they are theoretically capable. We will take the stance that greater computational power implies a greater

¹Of all the connectionists, Stephen Grossberg's work is probably closest in spirit to that of the control theorists.

degree of control over a network's run-time dynamics. We must develop systems that are capable of performing more than one class of run-time computation. To achieve this, control must be exerted both at run-time and at training time. At run-time, the network's dynamics must be controlled by computational goals that may change in response to external inputs and the current system state. During training, important constraints that determine the details of this run-time control must be derived from observations of external inputs and the network's internal state. In both cases we require a theory that explains how the computational dynamics are determined, and how they can be controlled. Control theorists have given us exactly this type of theory. Ironically, the importance of controlling connectionist run-time dynamics is not recognized by the control community, and the availability of tools to achieve this is not recognized by the connectionists.

Control theory is concerned with how to modify the dynamics of physical systems in order to make them behave in useful ways. It is also concerned with understanding limitations on the degree to which the system under study can be controlled. The power of the theory stems from its broad applicability to many classes of systems, including the Grossberg Normal Form (GNF) networks whose computational properties we wish to study. From the control theoretic viewpoint, GNF networks simply represent a specialized class of systems whose dynamics are amenable to control. A recent trend in control theory is to take advantage of the specialized characteristics of important classes of nonlinear systems. We see our thesis as an extension of this trend.

Our primary goal is to understand to what degree we can exert control over GNF network dynamics. To this end, we will perform an analysis of the controllability and observability properties of GNF systems. The goal of this study will be to show how the network weights determine the degree to which the network dynamics can be influenced by external inputs. Our intent is threefold. First, we want to show that useful computational constraints can be described in terms of controllability and observability distributions. Second, we shall try to demonstrate how a training

algorithm can, in principle, shape these constraints by adjusting the network weights. Finally, we shall show how these learned constraints can be made operational in novel classes of networks and hybrid connectionist/control systems closely related to the GNF family of networks.

Before we consider the application of control theory to connectionist computation, we will present a review of basic connectionist concepts in the next section. This will be followed by a description of GNF networks in Section 2.2, and an overview of basic notions from control in Chapter 3.

2.2 The GNF Family of Neural Networks

In this section we describe an important *family* of connectionist networks that we study in the thesis, namely the Grossberg Normal Form (GNF) family of networks. Our choice of this continuous-time family of networks has been motivated primarily by our intended analytical toolset. More results are available for the control of nonlinear continuous-time systems than the discrete-time variety. The GNF family is a good choice also because it generalizes many classic connectionist run-time architectures. In Chapters 4 and 5 we will contribute to the general understanding of the run-time controllability of these networks, and relate our findings to the classical models, some of which are summarized below.

2.2.1 Grossberg Normal Form

Many different phenomena can be described in terms of a population of agents that interact by changing the rate of reproduction of other agents. These phenomena include chemical network kinetics, the population dynamics of ecological systems, and the operation of the immune system. Despite the diversity of these domains, the mathematical models that govern them are very similar. They all fall under the banner of *generalized Lotka-Volterra equations*.

Stephen Grossberg, an early pioneer in continuous time modeling of connectionist systems, has studied [Gro67, Gro68, Gro73, Gro76, Gro87a, Gro87b] a class of recurrent neural networks with neuron dynamics given by the following equation

$$\dot{x}_i = a_i(x_i) \left[b_i(x_i) - \sum_{j=1}^n \mathbf{H}_{ij} d_j(x_j) - \sum_{k=1}^m \mathbf{J}_{ik} u_k \right]. \quad (2.2.1)$$

We shall study a slightly modified form of Grossberg's model. Our generalization is given by an n -dimensional system of ordinary differential equations of the form

$$\begin{aligned} \dot{x}_i = a_i(x_i) & \left(b_i(x_i) + \sum_{j=1}^n \left(s_{j,i}^+(x_i) H_{i,j}^+ - s_{j,i}^-(x_i) H_{i,j}^- \right) d_j(x_j) \right. \\ & \left. - \sum_{k=1}^m \left(s_{(k+n),i}^+(x_i) J_{i,k}^+ - s_{(k+n),i}^-(x_i) J_{i,k}^- \right) u_k \right). \end{aligned} \quad (2.2.2)$$

Equation (2.2.2) determines the time derivative of the i^{th} neuron's activation state, in a network of n neurons. The three terms inside the outer-most parentheses represent mechanisms that drive the level of excitation of the neuron at time t . Term $a_i(x_i)$ models metabolic nonlinearities that modulate the i^{th} neuron's overall level of excitation of neuron i . Function $b_i(x_i)$ is a damping term that determines the rate at which the activation level moves toward its 'rest' state when unperturbed by external input signals. The second term — parameterized by the n excitatory intra-network weights $H_{i,j}^+$ and n inhibitory intra-network weights $H_{i,j}^-$ that connect the output, $d_j(x_j)$, of the j^{th} neuron to the i^{th} neuron — represents excitation due to signals received by neuron i from all of the neurons in the network. The third term represents the degree of stimulation received by the cell from sources outside the network (e.g. from sensory cells, or another network in a modular system). It is parameterized by m excitatory input weights $J_{i,k}^+$ and m inhibitory input weights $J_{i,k}^-$ that connect the k^{th} external input signal, from external neuron u_k , to the i^{th} neuron. The n by n matrices $\mathbf{H}^+$, and $\mathbf{H}^-$, and the n by m matrices $\mathbf{J}^+$, and $\mathbf{J}^-$, correspond to the excitatory and inhibitory synaptic weights associated

with the recurrent intra-network connections and connections from external sources, respectively. In the sequel, the j^{th} column vector of a matrix $\mathbf{M}$ is denoted $\mathbf{M}_j$, and the element at row i , column j is given by $M_{i,j}$. Vector $\mathbf{H}_i^\pm$ corresponds to the synaptic fan-out of neuron i , and $\mathbf{J}_i^\pm$ represents the connections that the i^{th} input (i.e. sensory neuron) makes with the network. For the purposes of this chapter, we will assume that the weights are not changing. The synaptic gain terms $s_{j,i}^+(x_i)$ and $s_{j,i}^-(x_i)$, $1 \leq i \leq n + m$, represent nonlinearities that modify the synaptic weighting of the signal received from neuron j by neuron i . These gains model the nonadaptive peculiarities of a particular synaptic connection, as opposed to the synaptic weights which may be modified by learning.

The neurons obeying equation (2.2.2) can, in general, make both excitatory and inhibitory connections with other neurons. We refer to these as *heterosynaptic* neurons. In the event that neuron i is constrained to have only excitatory ($\mathbf{H}_i^- = \mathbf{0}$ and $\mathbf{J}_i^- = \mathbf{0}$) or inhibitory ($\mathbf{H}_i^+ = \mathbf{0}$ and $\mathbf{J}_i^+ = \mathbf{0}$) connections, but not both, we refer to the neuron as *homosynaptic*. These terms may also be applied to a network that contains neurons of that type. In the special case of a homosynaptic network composed of neurons with a single valency, we say that the network is *monosynaptic*. To our knowledge, very little attention has been paid to the theoretical implications of different mixtures of neurons.

In general, we are interested in constructing computational systems that consist of more than one network obeying equation (2.2.2). Examples abound in both the neural network literature, and the hybrid systems literature, in which several connectionist modules have to connect to one another as shown in Figure 2.1. Indeed, it is well known that nature has also employed a degree of modularization of cortical and subcortical structures, which at a first order of approximation, can also be modeled in this way.

The following terms will help us distinguish between connections within a network and connections between modules.

Figure 2.1: A System of Interconnected GNF Networks

Definition 2.1

The connections that occur between neurons in a network will be referred to as intra-network connections. The functions $d_j(x_j)$, $1 \leq j \leq n$, that appear in equation (2.2.2) are intended to model the nonlinear thresholding characteristics of individual neurons. We will refer to these as the neural output functions.

In the neural network literature it is standard to assume that the output of the network as a whole consists of the neural output functions of a fixed subset of neurons called the *network output set*. We will generalize this notion and assume that a smooth time invariant mapping is applied to transform the entire activation state of an n neuron network into p output values. i.e.

$$\mathbf{h} : \mathbb{R}^n \mapsto \mathbb{R}^p.$$

where $\mathbb{R}^n$ is the activation space of the network, and $\mathbb{R}^p$ is the *output space* of the network. The notion of the output set is subsumed by this mapping.

Definition 2.2

The vector field $\mathbf{h}(\mathbf{x})$ will be referred to as the network output field, and its elementary functions $h_i(\mathbf{x})$, $1 \leq i \leq p$, will be called the inter-network output channels, or simply output channels. Since $\mathbf{h}$ is not necessarily injective, the activation state of a network is not directly available to other modules.

The family of output fields that we consider is more general than the one usually considered in the classical neural network literature — where each output channel is usually taken to be a sigmoidal function of a single neuron from a fixed size output set. The class of output functions that we study includes the sigmoids, but also includes many other nonlinear, noninvertible mappings. The class of output fields that we choose to study is defined as follows:

Definition 2.3

An axial summation field is a vector field of the form

$$\mathbf{h}(\mathbf{x}) = \sum_{j=1}^n \tau_j(x_j). \quad (2.2.3)$$

where $\tau_j(x_j) \in C^\infty$. The family of axial summation fields is denoted $\mathcal{F}_{sum}$.

Intuitively, each elementary function in an axial summation field can be decomposed into a sum of n functions. Each of these is driven by the activation value of exactly one neuron. This class is general enough to cover a large family of nonlinear and possibly nondiffeomorphic mappings and, most importantly, it includes the classic ‘sigmoid’ functions that are ubiquitous in the neural network literature. Most neural network models adopt some type of monotonic sigmoid function as their output nonlinearity and a subset of these are typically designated as output channels² of the network. Popular choices of sigmoid are

$$h_i(x_i) = \frac{1}{1 + e^{-\theta_i x_i}}, \quad \theta \in \mathbb{R} \text{ and}$$

$$h_i(x_i) = \arctan(x_i).$$

Output fields composed of such smooth monotonic functions³ are clearly members of $\mathcal{F}_{sum}$ since they can be written in the form

$$\mathbf{h}(\mathbf{x}) = \sum_{j=1}^n h_j(x_j) \mathbf{e}_j.$$

We may naturally visualize an output summation field as a network of specialized neurons that perform smooth, nonlinear and possibly noninvertible time-invariant mappings of the activation values in the output set of a GNF network. Such a network is illustrated in Figure 2.2. Although, this is in keeping with a connectionist viewpoint, other interpretations of the physical meaning of an output summation field are also possible.

²i.e. the ‘output set’.

³Although the discontinuous Heavyside function is sometimes used to represent a sigmoid, it can be approximated arbitrarily accurately by its continuous cousins.

Figure 2.2: Output Summation Field Network

We shall refer to a computational module that consists of a network with neurons obeying equation (2.2.2) and an output field of the form (2.2.3) as a *Grossberg Normal Form (GNF) network*. The generality of this type of system makes it a good candidate for detailed study. We will say that a connectionist system is in *Grossberg normal form*, or is a *GNF system*, or that it is of the *GNF family* of connectionist systems if its run-time dynamics can be expressed as a special case of the above equations. In this thesis we will concern ourselves primarily with the study of GNF systems and systems which are essentially modifications of the GNF model.

2.2.2 A Sufficient Condition for Stability in GNF Systems

Cohen and Grossberg [CG83] produced an early result on sufficient conditions for the stability of homogenous GNF systems. As we shall see below, this result generalizes many other stability results. We will also be briefly concerned with system stability in this thesis, but our emphasis will be on stabilization of heterogenous GNF systems through state feedback control. For completeness we state the theorem of Cohen and Grossberg below, without proof.

Theorem 2.1 (Cohen-Grossberg)

Given a homogenous GNF system of the form

$$\dot{x}_i = a_i(x_i) \left[b_i(x_i) - \sum_{j=1}^n c_{ij} d_j(x_j) \right], \text{ for all } 1 \leq i \leq n \quad (2.2.4)$$

where

$$c_{ij} = c_{ji} \quad (2.2.5)$$

$$a_i(x_i) \geq 0 \quad (2.2.6)$$

$$\frac{d}{dx_j} d_j(x_j) \geq 0 \quad (2.2.7)$$

then there exists a global Liapunov function

$$E = \sum_{i=1}^n \int^{x_i} b_i(\xi_i) \frac{d}{dt} d_i(\xi_i) d\xi_i + \frac{1}{2} \sum_{j=1}^n \sum_{k=1}^n c_{jk} d_j(x_j) d_k(x_k) \quad (2.2.8)$$

such that

$$\frac{d}{dt}E = - \sum_{i=1}^n a_i(x_i) \frac{d}{dt}d_i(x_i) \left[b_i(x_i) - \sum_{j=1}^n c_{ij}d_j(x_j) \right]^2 \quad (2.2.9)$$

where $\frac{d}{dt}E < 0$.

Proof: The proof can be found in [CG83]. ■

In other words, a GNF system trajectory will always evolve toward an equilibrium state, regardless of its initial conditions, as long as the weights between each pair of neurons are equal (i.e. the weight matrix is symmetric), and some reasonable conditions are assumed for the excitation functions a_i and b_i .

This result is important to many connectionist models which store memories as equilibria states. In these systems, the memory recall or pattern classification computation corresponds to the evolution of the system toward an attracting equilibria. Grossberg has shown that GNF systems may have infinitely many equilibria under the right conditions.

2.2.3 Special Cases of the GNF Family

Grossberg has claimed that many well-studied ‘classical’ neural networks can be posed as special cases of equation (2.2.1). Therefore, results that apply to the general form of the equations can be related to more specialized systems of interest to the connectionist community. Although Grossberg has alluded to these results, we are unaware of a formal proof of these claims.

In this section we shall formalize this claim by demonstrating that several notable connectionist models are special cases of the general GNF family described above. We do so in the interest of continuity and to assist future researchers interested in analysing special cases of GNF networks. Each specialized model can be put into Grossberg normal form by a simple change of variables. We concentrate on the run-time dynamics in this section, and do not present the various learning algorithms associated with these models. Full accounts can be found in the references.

The reader's attention is drawn to the severely limited run-time dynamics of each of these models. We believe that control techniques can enable more sophisticated types of behaviours than simple convergence to a fixed point.

2.2.3.1 Shunting Inhibition Networks

Biological neurons cannot achieve membrane potentials of arbitrarily large magnitude. The range of activation values is limited by the activity of chemical pumps, and other metabolic processes that are summarized by the Hodgkin-Huxley membrane equation. Shunting inhibition networks are designed to mimic the limited operational range of biological neurons.

The shunting inhibition equations are of the form:

$$\begin{aligned} \dot{x}_i = & -A_i x_i \\ & + (B_i - x_i) \left[\sum_{j=1}^n f_j(x_j) C_{ij} Z_{ij}^- + \sum_{k=1}^m g_k(I_k) D_{ik} W_{ik}^+ \right] \\ & - (x_i + E_i) \left[\sum_{j=1}^n h_j(x_j) F_{ij} Z_{ij}^- + \sum_{k=1}^m i_k(I_k) G_{ik} W_{ik}^- \right]. \end{aligned} \quad (2.2.10)$$

The passive decay term $-A_i x_i$ indicates that the neurons are stable, in the sense that their zero-input dynamics cause the activation value to decay to zero at the scalar rate $-A_i$. The output functions of excitatory and inhibitory cells are given by f_j and h_j respectively. The term $\sum_{j=1}^n f_j(x_j) C_{ij} Z_{ij}^+$ represents the excitatory influence that cell i receives from all the other cells. The modifiable weight of the synapse connecting an excitatory cell j to a cell i is given by Z_{ij}^+ , and C_{ij} represents a fixed gain associated with this synapse. Similarly, term $\sum_{k=1}^m g_k(I_k) D_{ik} W_{ik}^+$ describes the excitatory influence that external sensory input I_k has on cell i . The function g_k represents a distortion of the input signal introduced by the receiving sensory cell, and W_{ik}^+ and D_{ik} correspond to the modifiable and constant gain components of the synapse that connects the sensory cell to the i^{th} neuron. In order to ensure

that these excitatory influences cannot produce arbitrarily large activation values. they are multiplied by the shunting term $(B_i - x_i)$. As long as the initial activation value $x_i(0) < B_i$, the activation value cannot exceed B_i , since the excitation term is multiplied by 0 at $x_i = B_i$, thereby allowing the remaining decay terms to drag the activation value back toward 0. A similar description applies to the inhibitory influence term $(x_i + E_i) \left[\sum_{j=1}^n h_j(x_j) F_{ij} Z_{ij}^- + \sum_{k=1}^m i_k(I_k) G_{ik} W_{ik}^- \right]$, which ensures that the activation value never falls below E_i . The result is that the operational range of the neurons is $[E_i, B_i]$.

Theorem 2.2

The Shunting Inhibition Network belongs to the class of GNF neural networks.

Proof: To put a shunting inhibition network into Grossberg normal form, we make the change of variables $y_i = x_i + E_i$. We have

$$\begin{aligned}
 \dot{y}_i &= \dot{x}_i \\
 &= -A_i(y_i - E_i) \\
 &\quad + (B_i - (y_i - E_i)) \left[\sum_{j=1}^n f_j(y_i - E_i) C_{ij} Z_{ij}^+ + \sum_{k=1}^m g_k(I_k) D_{ik} W_{ik}^+ \right] \\
 &\quad - y_i \left[\sum_{j=1}^n h_j(y_i - E_i) F_{ij} Z_{ij}^- + \sum_{k=1}^m i_k(I_k) G_{ik} W_{ik}^- \right] \\
 &= a_i(y_i) \left[b_i(y_i) + \sum_{j=1}^n \left(s_j(y_j) \mathbf{H}_{ij}^+ - \mathbf{H}_{ij}^- \right) d_j(y_j) \right] \tag{2.2.11}
 \end{aligned}$$

$$+ \sum_{k=1}^m \left(s_k(y_k) \mathbf{J}_{ik}^+ - \mathbf{J}_{ik}^- \right) u_k \tag{2.2.12}$$

where

$$a_i(y_i) = y_i, \quad b_i(y_i) = \frac{A_i E_i}{y_i} - A_i, \quad s_i(y_i) = \frac{1}{y_i} (B_i - y_i + E_i)$$

$$\mathbf{H}_{ij}^+ = C_{ij} Z_{ij}^+, \quad \mathbf{H}_{ik}^- = F_{ik} Z_{ik}^-, \quad \mathbf{J}_{ij}^+ = D_{ij} W_{ij}^+, \quad \mathbf{J}_{ik}^- = G_{ik} W_{ik}^-$$

This last equation is clearly in Grossberg normal form, and the theorem is proved. ■

2.2.3.2 Hopfield Networks

Many connectionist models have been developed as memories for the associative storage and retrieval of patterns [ASRJ77, Koh77, CG83, Hop84, AHS85, Kos88]. Unlike conventional computer memories, where a pattern is retrieved by specifying the exact address where it is stored, associative memories allow an incomplete or noisy *key pattern* to recall an associated *response pattern*. This is referred to as *content addressability*. If the key and response patterns are the same, we call the memory *autoassociative*, and we refer to the retrieval process as *pattern completion* since the corrupted pattern is converted into its uncorrupted (i.e. completed) form. The more general form of associative memory is referred to as *heteroassociative*. These memories find application as pattern classifiers, time series generators, and signal filters. They are also used to solve difficult optimization problems, and, of course, as associative memories in AI applications.

In 1984, Hopfield [Hop84] published a network model that is generally credited with rekindling interest in connectionist systems. Hopfield's model was originally presented as an autoassociative memory, but it has also found application as a good heuristic technique for solving NP-hard optimization problems.

In the case of the Hopfield network, we have the following activation and output functions

$$\dot{x}_i = -x_i + \sum_{j=1}^n S(x_j)M_{ij} + I_i \quad (2.2.13)$$

$$S(z) = \frac{1}{1 + e^{-Tz}} \quad (2.2.14)$$

and the energy function

$$\begin{aligned}
 E = & - \sum_{j=1}^n \int^{x_j} \xi_j \frac{d}{dt} S(\xi_j) d\xi_j \\
 & - \sum_{j=1}^n \sum_{k=1}^n \mathbf{M}_{j,k} S(x_j) S(x_k) - \sum_{j=1}^n S(x_j) I_j
 \end{aligned} \tag{2.2.15}$$

where

$$\frac{d}{dt} E = - \sum_{j=1}^n \frac{d}{dt} S(x_j) \dot{x}_j^2 \leq 0. \tag{2.2.16}$$

Theorem 2.3

The Hopfield network as a member of the GNF class of neural networks.

Proof: The result trivially follows if we make the following substitutions

$$\begin{aligned}
 a_i(x_i) &= 1. \\
 b_i(x_i) &= -x_i. \\
 \mathbf{H}_{i,j} &= -\mathbf{M}_{i,j}. \\
 \mathbf{J} &= \mathbf{I}. \\
 d_j(x_j) &= S(x_j). \\
 u_i &= I_i.
 \end{aligned}$$

■

2.2.3.3 Bidirectional Associative Memories (BAMs)

Kosko's [Kos88] bidirectional associative memory (BAM) is a very general type of heteroassociative memory which consists of two layers of neurons that are fully interconnected by symmetric weights. In fact, the Hopfield network is a special case of the BAM in which the neurons in the two layers are identified to produce a single layer. Connections in the BAM only occur between layers, and not within

a layer. The symmetry property means that the weights on connections between the i^{th} and j^{th} neurons in different layers are equal ($w_{ij} = w_{ji}$). $\mathbf{M}$ represents the symmetric matrix of weights, and the evolution equations for the continuous-time version of this model are divided into a -layer equations, and b -layer equations as follows. The activation state equations are given by

$$\dot{a}_i = -a_i + \sum_{j=1}^{n_b} S(b_j) \mathbf{M}_{ij} + I_i \quad (2.2.17)$$

$$\dot{b}_j = -b_j + \sum_{i=1}^{n_a} S(a_i) \mathbf{M}_{ji} + J_j \quad (2.2.18)$$

and the output equations for both layers of neurons are given by any function S which is a bounded monotone-increasing function. A typical output function is given by the sigmoid

$$S(z) = \frac{1}{1 + e^{-Tz}}. \quad (2.2.19)$$

Clearly, the properties of the output function and the requirement for a symmetric $\mathbf{M}$ are motivated by the Cohen-Grossberg theorem. In this case, stability is required to ensure that the BAM always converges to a stored memory. The energy function and its derivative are given by

$$\begin{aligned} E = & - \sum_{j=1}^{n_a} \int^{a_j} \frac{d}{dt} S(\xi_j) d\xi_j - \sum_{j=1}^{n_a} \sum_{k=1}^{n_b} \mathbf{M}_{jk} S(a_j) S(b_k) \\ & - \sum_{j=1}^{n_a} S(a_j) I_j + \sum_{k=1}^{n_b} \int^{b_k} \frac{d}{dt} S(\xi_k) d\xi_k \end{aligned} \quad (2.2.20)$$

where

$$\frac{d}{dt} E = - \sum_{j=1}^{n_a} \frac{d}{dt} S(a_j) \dot{a}_j^2 - \sum_{k=1}^{n_b} \frac{d}{dt} S(b_k) \dot{b}_k^2 \leq 0. \quad (2.2.21)$$

Thus, we see that the system will minimize the energy function and therefore converge to a stable fixed point.

The weight update dynamics are designed to ensure that the recallable patterns correspond to system equilibria. The basin of attraction of a combined ab -activation state hopefully includes the subspace of b states with a constant a pattern. If this is the case, the network will evolve toward the complete ab pattern when the a pattern is presented to the network.

Theorem 2.4

The Bidirectional Associative Memory neural network belongs to the class of GNF neural networks.

Proof: The result trivially follows if we observe that by defining

$$\mathbf{x} = \begin{pmatrix} a_1 \\ \vdots \\ a_{n_a} \\ b_1 \\ \vdots \\ b_{n_b} \end{pmatrix}, \quad \text{where } a_i(x_i) = 1, \quad b_i(x_i) = -x_i.$$

the parameters of the GNF equation are as follows:

$$\begin{aligned} \mathbf{H}_{ij} &= -\mathbf{M}_{ij}, \\ \mathbf{J} &= \mathbf{I}, \\ d_j(x_j) &= S(x_j), \\ u_i &= I_i. \end{aligned}$$

The above assignment can be seen to render the BAM in the same form as the canonical GNF system (2.2.1). ■

2.2.3.4 Boltzmann Machines

Another popular connectionist model was proposed in 1985 by Ackley, Hinton, and Sejnowski [AHS85]. This model is actually a combination of Hopfield's

network and the technique of simulated annealing. Whereas the Hopfield network uses a deterministic algorithm to store memories as point attractors, the Boltzmann machine uses a stochastic algorithm based on simulated annealing to position its energy wells. The basic form of the evolution equations is the same as the Hopfield network, and so it too fits into the GNF family of networks as was demonstrated earlier.

2.2.3.5 Brain-State-In-A-Box (BSB)

This is an early associative memory model that was applied to pattern classification by its inventor Anderson and his colleagues Silverstein, Ritz and Jones [ASRJ77]. The Brain-State-In-A-Box is a homogenous, saturating network in which classes are associated with the corners of an n -dimensional hypercube. A pattern to be classified is encoded as the initial state of the network, and the classification is performed by allowing the system trajectory to saturate to one of the 2^n corners, thereby assigning a classification.

The network is fully interconnected and the weights are assumed to be symmetric. The discrete time state evolution equations of the BSB are given by

$$x_i(t+1) = S\left(\sum_{j=1}^n \mathbf{B}_{ij}x_j(t)\right) \quad (2.2.22)$$

and equivalent continuous time equations are of the form

$$\dot{x}_i = -x_i + S\left(\sum_{j=1}^n \mathbf{B}_{ij}x_j\right). \quad (2.2.23)$$

In both cases the output function S is given by

$$S(z) = \max\{F, \min\{-F, z\}\}. \quad (2.2.24)$$

Theorem 2.5

The Brain-State-in-a-Box network is a member of the GNF family of neural networks.

Proof: These continuous time equations can be put into GNF form by the change of coordinates. $y_i = \sum_{j=1}^n B_{ij}x_j$. as follows

$$\begin{aligned}
 \dot{y}_i &= \sum_{j=1}^n \mathbf{B}_{ij} \dot{x}_j \\
 &= - \sum_{j=1}^n \mathbf{B}_{ij} x_j + \sum_{k=1}^n \mathbf{B}_{ik} S\left(\sum_{q=1}^n \mathbf{B}_{iq} x_q\right) \\
 &= -y_i + \sum_{j=1}^n \mathbf{B}_{ij} S(y_j) \\
 &= a_i(y_i) \left[b_i(y_i) - \sum_{j=1}^n \mathbf{H}_{ij} d_j(y_j) \right]
 \end{aligned} \tag{2.2.25}$$

where

$$a_i(y_i) = 1, \quad b_i(y_i) = -y_i, \quad \mathbf{H}_{ij} = -\mathbf{B}_{ij}, \quad d_j(y_j) = S(y_j).$$

Hence the result. ■

Applying these substitutions to equation (2.2.8) we see that the energy function for the continuous BSB model is given by

$$E = - \sum_{j=1}^n \int^{y_j} \xi_j \frac{d}{dt} S(\xi_j) d\xi_j \tag{2.2.26}$$

$$- \frac{1}{2} \sum_{j=1}^n \sum_{k=1}^n \mathbf{B}_{jk} S(y_j) S(y_k) - \sum_{j=1}^n S(x_j) I_j \tag{2.2.27}$$

where

$$\frac{d}{dt} E = - \sum_{j=1}^n \frac{d}{dt} S(y_j) \left[y_i - \sum_{k=1}^n \mathbf{B}_{jk} S(y_k) \right]^2 \leq 0. \tag{2.2.28}$$

2.2.3.6 The McCulloch-Pitts Model

The discrete time version of this early network has the form

$$x_i(t+1) = S_i\left(\sum_{j=1}^n \mathbf{A}_{ij} x_j\right) \tag{2.2.29}$$

where

$$S_i(z) = \text{sgn}(z - \mathbf{B}_i). \quad (2.2.30)$$

The equivalent continuous time equation is of the form

$$\dot{x}_i = -x_i + S_i\left(\sum_{j=1}^n \mathbf{B}_{ij}x_j\right). \quad (2.2.31)$$

Comparing this equation to that of the BSB model we see that the same change of coordinates. $y_i = \sum_{j=1}^n \mathbf{B}_{ij}x_j$. puts the McCulloch-Pitts model into the Grossberg normal form.

2.2.3.7 Mikhailov's Network, and the Volterra-Lotka Population Model

Mikhailov [MP91] has investigated the properties of attractors in simplified GNF systems. In particular. Mikhailov studies GNF systems in which

$$\begin{aligned} a_i(x_i) &> 1 \\ b_i(x_i) &= B_i \\ B_i &< \mathbf{W}_{ij}d_j(x_j) \\ a_j(0) &= d_j(0) = 0. \end{aligned}$$

A very closely related model is the classic version of the Volterra-Lotka model of population dynamics. This population model can be expressed as the GNF system

$$\begin{aligned} \dot{x}_i &= A_i x_i \left[1 - S\left(\sum_{j=1}^n \mathbf{B}_{ij} \dot{x}_j\right) \right] \\ &= a_i(x_i) \left[b_i(x_i) - \sum_{j=1}^n \mathbf{W}_{ij} d_j(x_j) \right] \end{aligned} \quad (2.2.32)$$

where

$$a_i(x_i) = A_i x_i, \quad b_i(x_i) = 1, \quad \mathbf{W}_{ij} = -\mathbf{B}_{ij}, \quad d_j(x_j) = x_j.$$

2.3 Chapter Conclusions

In this chapter we have introduced a very general family of continuous-time connectionist systems, called the Grossberg Normal Form (GNF) networks, which are important to the remainder of the thesis. We choose to study GNF networks because they generalize many classical run-time architectures, and they are amenable to analysis by control theoretic techniques. Our purpose is to find GNF weight update algorithms that can learn manifold constraints from observations of an evolving domain, without supervision. Furthermore, these constraints must be operational in that they restrict the activation space dynamics of a GNF network to the learned manifold.

Chapter 3

Nonlinear Control Theory

The purpose of this chapter is to introduce the reader to the main conceptual and mathematical tools from the theory of differential geometric state-space control that form the basis of our study. After setting the stage (in Section 3.1.1) with a *brief* history of control, we introduce the reader in Sections 3.2 through 3.6 to relevant control theoretic ideas and tools that will be used in the sequel. Our particular viewpoint is from the theory of differential geometric¹ state-space control rather than frequency-based control. Finally, we examine some of the implications of the theory for our study of manifold constraints in Section 3.7.

3.1 An Introduction to Control

Control theory is a wide ranging discipline that has a long and distinguished history. It is so encompassing that it has come to have many different meanings to the people that work within its boundaries. To give a comprehensive accounting of its various subdisciplines would require several volumes, but, for the purposes of our thesis, we can sum up our view of control theory with one question: “Given an

¹The reader is directed to Appendix B for a brief review of differential geometry. A comprehensive survey of the area can be found in the classic book of Boothby [Boo86].

arbitrarily complex system that has some inputs and outputs, and an approximate mathematical model of its internal workings, how do we adjust the inputs to make the system behave in some desired way?" An equivalent way of saying this is, "How can I change the dynamics of a dynamical system to suit my purpose?"

The general mathematical techniques that have been developed to answer these questions are remarkable in that they can be applied to a broad range of physical systems. The theory lends itself to anything that can be mathematically modeled, from interplanetary spacecraft to the central pattern generator of a crayfish. Consequently, these can be made to behave as desired, within the physical limitations of the system. Part of the theory is concerned with understanding these limitations. The other part provides us with techniques, often employing some form of feedback loop, that change the dynamics of the system in desirable ways.

For historical reasons, the system to be controlled is almost universally referred to as the *plant*. Any additional machinery that is introduced to change the plant's dynamics is referred to as the *controller*. It is often a matter of taste as to where one draws the line between the plant and the controller, because the combination of the two becomes a new dynamical system (plant) in its own right, which is subject to control.

Our intent in this chapter is to give the reader a brief and informal (without proof) review of the more important nonlinear control theoretic concepts and techniques. A detailed accounting of the control theoretic techniques described below can be found in the classic books of Isidori [Isi89], and Nijmeijer and van der Schaft [NvdS90].

3.1.1 A Brief History

The development of control theory began in the mid 1800s with Maxwell's analysis of the stabilization of the speed of a mechanical engine by feedback governors. Maxwell demonstrated that the behaviour of a dynamical system in the vicinity of an

equilibrium was governed by the locations of the eigenvalues of a linearization of the system. By the turn of the century, introduction of Fourier analysis-based techniques for studying dynamical phenomena firmly rooted this theory in the frequency domain. By 1932 Nyquist and others had ushered in a new era of frequency-based feedback techniques for the control of a system's dynamics. Transfer functions, relating the connection between inputs and outputs in the frequency domain, had become an important tool by the 1950s.

Around the same time, Kalman popularized the state space view of control as an alternative to frequency-response techniques. This approach, based on pioneering work by Poincare and Lyapunov, expressed the system model as a set of dynamical variables characterized in terms of the trajectory of a point in an n -dimensional space. When n is finite the system is referred to as a lumped parameter system, otherwise it is called a continuous parameter system. Rather than attempting to model a direct connection between the inputs and outputs, the state space approach separates the relationship involving the inputs and state dynamics from the corresponding one relating the state dynamics and outputs. In a sense, the state trajectory represents a memory of the system's past behaviour. Although methods were developed to move between the frequency-response and state space representation of a system, the state space viewpoint rapidly became dominant in the 1960s. By the early 1970s, research emphasis had shifted to optimal control-finding solutions to control problems that are low cost relative to a performance function.

Up until this time, models of a system's dynamics were almost exclusively linear. The theory of nonlinear functions was not well understood, and linear approximations to the true dynamics were a pragmatic necessity. Largely in response to this situation, techniques were developed to adaptively refine the underlying linear model to track the true dynamics. The mathematical tools that were most often applied to linear state space control were based on an intuitively appealing geometric interpretation of linear algebra. It was not until the end of the 1970s that a coherent theory began to emerge for the control of nonlinear system dynamics. The 1980s saw

rapid development of a state-space oriented view of nonlinear control based on the theory of differential geometry. Many linear techniques were generalized and applied to nonlinear systems.

The 1990s have seen a greater emphasis on nonlinear control. The stabilization of nonlinear systems in conjunction with other control objectives has been an area of active research. Adaptive versions of nonlinear control schemes have also seen recent advances. A cross fertilization between bifurcation theory, the theory of dynamical systems, and nonlinear control theory is also yielding interesting new insights. Finally, more recently, attention has been paid to the special physical structure of some classes of nonlinear control systems. It is in this spirit that we intend to apply the theory of control to the family of continuous time GNF networks.

3.2 The State Space View of Control

Throughout this thesis we will consider autonomous, multivariable, lumped parameter nonlinear plants of the form

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) + \sum_{j=1}^m \mathbf{g}_j(\mathbf{x}) u_j \\ &= \mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x}) \mathbf{u}\end{aligned}\tag{3.2.1}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x})\tag{3.2.2}$$

that we call *Isidori-style systems*². The input vector $\mathbf{u}$ is m -dimensional, the state space vector $\mathbf{x}$ is n -dimensional, and the output vector $\mathbf{y}$ is p -dimensional. With no loss of generality, we will assume³ throughout the remainder of the thesis that $p \leq m$, and n can be of any magnitude, but that it is typically assumed that $n \geq p$. Equation (3.2.1) describes the velocity of the state trajectory $\Phi_T^{\mathbf{x}}(\mathbf{x}_0(t_0), \mathbf{u})$ for some interval

²After Alberto Isidori who has summarized the control literature pertaining to such systems in his influential textbook [Isi89].

³Assuming $p \leq n$ simplifies the analysis. We can always make $p \leq m$ by adding extra inputs that are not actually connected to the system and which therefore do not affect the dynamics.

of time $[t_0, t_0 + T]$ starting at initial state $\mathbf{x}_0(t_0)$ under the influence of the input $\mathbf{u}$. The state space in which the equations evolve can be any n -dimensional differentiable manifold $\mathcal{M}$, and the vector fields $\mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})$ are defined over the tangent space $T_{\mathbf{x}}\mathcal{M}$ (see Appendix B for a definition of tangent spaces). The vector field $\mathbf{f}(\mathbf{x})$ represents a model of the internal workings of the plant before input control has been applied. We will refer to this part of the model as the passive *drift dynamics*, since the system state will drift through state space under control of this vector field until a nonzero input signal is injected into the system. The input signal $\mathbf{u}$ is a piecewise continuous function that is used to offset the drift dynamics by a linear combination of the *input fields* $\mathbf{g}_j(\mathbf{x})$, $1 \leq j \leq m$. These input fields represent constraints on the mechanisms that transmit the input signals to the physical plant.

Linear plants are special cases of equations (3.2.1) and (3.2.1). In the linear setting the state space is the flat, open manifold $\mathbb{R}^n$, and the vector fields are given by the linear transformations $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $\mathbf{G}(\mathbf{x}) = \mathbf{B}$ and $\mathbf{h}(\mathbf{x}) = \mathbf{C}\mathbf{x}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are appropriately sized real matrices.

A large class of physical models can be expressed as Isidori-style systems.

3.2.1 Control = Injecting Inputs to Overcome Drift Dynamics

The traditional goal of control theory is to overcome the drift dynamics of the plant by supplying appropriate inputs that ‘fight against it’. This is done by finding an input equation $\dot{\mathbf{u}}$, called a *controller*, that cancels the drift dynamics and produces some other desired dynamics. It is useful to think of the controller as a system that steers the plant’s state through state space. If $\mathbf{f}(\mathbf{x})$ is an invertible mapping, its inverse is often incorporated into the equations of the controller to cancel the drift dynamics. We will see in Section 3.6 that the controller can be a feedback function that incorporates the state variable $\mathbf{x}$ or the output variable $\mathbf{y}$.

3.3 Reachability Manifolds and Controllability

Given a control system of the form (3.2.1), we wish to characterize the region of state space that can be reached from an initial state value under influence of the driving inputs. In other words, our question is, “Which states can I possibly steer the system into from a given starting point?”. The vector field $\mathbf{f}(\mathbf{x})$ causes the system state to passively drift along a solution trajectory $\Phi_t^{\mathbf{f}}$ if no inputs are applied (i.e. $\mathbf{u} = 0$). In order to move the system to a desired state value, an input $\mathbf{u}$ must be applied to produce a linear combination of vector fields $\hat{\mathbf{f}}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \sum_{j=1}^m \mathbf{g}_j(\mathbf{x})$ that causes the plant to move along a new trajectory $\Phi_t^{\hat{\mathbf{f}}}$ that intersects the target state.

In the nonlinear setting, *reachability analysis* is typically performed in two steps. First, the analysis is performed locally, in an open set around the initial state. This allows us to make an important simplifying assumption - that distributions relevant to the analysis are nonsingular. Local analysis is always possible and yields valuable information about the reachable states. However, it only applies in the local neighbourhood of the selected initial value, and must be repeated for other points of interest outside of this neighbourhood. Next, we attempt to extend the results of the local analysis to the entire state space. Unfortunately, this global analysis can be prohibitively complex. In practice, it is only applied to plants that can be shown to satisfy regularity conditions that make the analysis analytically tractable.

In both the local and global analyses our goal is to find a special distribution $\Delta(\mathbf{x})$ that foliates the space into reachability manifolds. Every initial state is contained in one such manifold, and the set of states that can be reached from this point will be an open subset of the manifold. In the local analysis, whereas $\Delta(\mathbf{x})$ can be different for each initial value, the global analysis produces a (possibly nonsingular) $\Delta(\mathbf{x})$ that applies to all initial values.

Definition 3.1

We say that a system is controllable or that it has the property of complete control-

lability if, for any two state values $\mathbf{x}_1$, and $\mathbf{x}_2$, there exists a finite time T and an admissible input function $u : [0, T] \mapsto U$ such that $\Phi_t^f(\mathbf{x}_1, \mathbf{u}) = \mathbf{x}_2$.

In other words, a controllable plant is one in which any initial value can be steered to any desired state in finite time by an appropriate driving input. From the viewpoint of control system design complete controllability is a desirable property and much research effort has been devoted to characterizing the conditions that give a plant this property. This is because it guarantees that the system designer can find (at least locally) a control signal that can produce essentially any desired dynamics in the plant. We shall see however, that our goals are actually to remove the property of complete controllability.

3.3.1 Local Reachability Analysis

The issue of reachability is well understood for linear control systems. Therefore it may appear that, given a nonlinear plant, we can simply apply the linear theory to a linearization of the plant around an initial value. It turns out though, that nonlinear plants are sometimes controllable even when their linearizations are not. This happens in the nonlinear case because the directions in which the plant can be steered depend not only on the vector fields $\mathbf{f}(\mathbf{x})$, and $\mathbf{g}_j(\mathbf{x})$, but also on the lie brackets of these vector fields. This is most easily seen by considering the Taylor expansion of a simple plant that has no drift dynamics, such as

$$\dot{\mathbf{x}} = \mathbf{g}_1(\mathbf{x})u_1 + \mathbf{g}_2(\mathbf{x})u_2.$$

One might expect the inputs to push the plant only in directions spanned by the vector fields $\mathbf{g}_1(\mathbf{x})$ and $\mathbf{g}_2(\mathbf{x})$. However, given input function

$$\mathbf{u}(t) = \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & 0 \leq t < \tau \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \tau \leq t < 2\tau \\ \begin{bmatrix} -1 \\ 0 \end{bmatrix} & 2\tau \leq t < 3\tau \\ \begin{bmatrix} 0 \\ -1 \end{bmatrix} & 3\tau \leq t < 4\tau \end{cases}$$

the Taylor expansion for this plant is

$$\mathbf{x}(t) = \mathbf{x}(0) + h^2 [\mathbf{g}_1, \mathbf{g}_2](\mathbf{x}_0) + O(h^3).$$

and we see that we can (at least approximately) steer the plant from $\mathbf{x}_0$ into the direction given by the Lie bracket $[\mathbf{g}_1, \mathbf{g}_2]$: a vector field that can be orthogonal to $\text{span}\{\mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})\}$ (see Figure 3.1). In fact, in general we can move in directions given by multibrackets of the form $[\tau_i, [\tau_j, \dots [\tau_k, \tau_l]]]$ where $\tau_i, \tau_j, \dots, \tau_k, \tau_l \in \mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})$.

The spanning set of these multibrackets turns out to be characterized by the smallest nonsingular, involutive distribution that contains the distribution $\mathbf{G}(\mathbf{x}) = \text{span}\{\mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})\}$, and is invariant under the vector fields $\mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})$. We write this distribution as $\Delta_*(\mathbf{x}) = \langle \mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x}) | \mathbf{G}(\mathbf{x}) \rangle$. $\Delta_*(\mathbf{x})$ can be

Figure 3.1: Lie Brackets Are Not Linear Combinations of the Input Fields

shown to be unique, and it can be computed by application of the recursive algorithm

$$\begin{aligned}\Delta_0 &= \text{span}\{g_1, \dots, g_m\} \\ \Delta_n &= \Delta_{n-1} + [f, \Delta_{n-1}] + \sum_{i=1}^m [g_i, \Delta_{n-1}]\end{aligned}\tag{3.3.1}$$

Recurrence (3.3.1) generates a nondecreasing sequence of nested distributions which converge to Δ_* . As long as each Δ_i is nonsingular, there will exist a k such that $\Delta_k = \Delta_{k+1}$, and $\Delta_* = \Delta_k$. Notice that, by construction, the final distribution is involutive, and since $[f, \Delta_*] \in \Delta_*$ and $[g_j, \Delta_*] \in \Delta_*$, $1 \leq j \leq m$, we see that it is also invariant to the vector fields $f, g_1, \dots, g_m$. By induction we see that Δ_* is clearly minimal on the sequence of distributions generated by the recurrence.

Δ_* locally foliates the state space into integral submanifolds which the plant's state must evolve on (see Figure 3.2). The inputs can push the plant around the manifold that contains the initial point x_0 , but the state can never leave this manifold. We will refer to these manifolds as *reachability manifolds*. In the event that the distribution is n -dimensional, the only manifold in the foliation is the entire state space, and the plant is therefore controllable.

There is an interesting and useful relationship between

$$\Delta_*(x) = \langle f(x), g_1(x), \dots, g_m(x) | G(x) \rangle$$

Figure 3.2: Δ_* Locally Foliates the Plant's State Space

and the distribution

$$\Delta_R(\mathbf{x}) = \langle \mathbf{f}(\mathbf{x}) . \mathbf{g}_1(\mathbf{x}) . \dots . \mathbf{g}_m(\mathbf{x}) | \text{span} \{ \mathbf{f}(\mathbf{x}) \} + \mathbf{G}(\mathbf{x}) \rangle .$$

Suppose that the dimensionality of Δ_R is r . It is possible to show that $\Delta_* + \text{span} \{ \mathbf{f} \}$ is a subset of Δ_R , and its dimensionality d is either r or $r-1$. The following geometric interpretation of this relationship can greatly aid the intuition when thinking about the forced dynamics of a plant that is not completely controllable. Since Δ_* is completely integrable (i.e. nonsingular and involutive) and it is invariant to all the plant's vector fields, we can find a change of coordinates, $\mathbf{z} = \Phi(\mathbf{x})$, which decomposes the plant dynamics into two pieces. In the new coordinates the form of the plant will be

$$\begin{aligned} \dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{z}_2) + \sum_{i=1}^m \mathbf{g}_{1,i}(\mathbf{z}_1, \mathbf{z}_2) u_i \\ \dot{\mathbf{z}}_2 &= \mathbf{f}_2(\mathbf{z}_2) \end{aligned}$$

$\mathbf{f}_2$ only depends on $\mathbf{z}_2$ because of the invariance of Δ_* under $\mathbf{f}$. This would be true of the $\mathbf{g}_i$ as well, except that, because they are completely contained in Δ_* , they are not dependent on dynamics outside of this distribution (i.e. $\mathbf{z}_2$) at all.

The above property can be extended further. Since Δ_* and $\mathbf{f}$ are both entirely contained in Δ_R , the above decomposition must occur within an r -dimensional subspace of the new coordinates, and, outside of this subspace, $\mathbf{f}$ will not depend on these r coordinates. The result is that we can further decompose the plant as follows

$$\begin{aligned} \dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) + \sum_{i=1}^m \mathbf{g}_{1,i}(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) u_i \\ \dot{\mathbf{z}}_2 &= \mathbf{f}_2(\mathbf{z}_2, \mathbf{z}_3) \\ \dot{\mathbf{z}}_3 &= 0 \end{aligned}$$

where $\mathbf{z}_1$ is d -dimensional, $\mathbf{z}_2$ is at most 1-dimensional, and $\mathbf{z}_3$ is $(n-r)$ -dimensional. In the event that $d = r$, dynamics $\mathbf{z}_2$ does not occur at all.

Figure 3.3: System Dynamics as Drifting 'Slice'

This decomposition has the following geometric meaning: there is always a change of coordinates⁴ which allows us to picture the system state evolving on a flat d -dimensional manifold which, in turn, is either stationary in state space if $d = r$, or drifts along a single dimension if $r - d = 1$ according to the drift dynamics $f_2(z_2, z_3)$ (see Figure 3.3). The dynamics within this slice of the state space (i.e. z_1) are completely controllable through the inputs, but the drift of the slice itself along dimension z_2 cannot be influenced by the inputs. This picture of a drifting slice characterizes the restriction in behavioural repertoire that is imposed by lack of controllability. The set of reachable states is determined by the drifting trajectories of the slice $\dot{z}_1 = 0$, and if the drift of the slice never intersects a state value, then that value is not reachable.

⁴This is because the distribution is controlled invariant as explained in Section B.11 of Appendix B.

3.3.2 Global Reachability Analysis

The local decomposition given above is dependent on the fact that Δ_* is locally nonsingular within a neighborhood of the initial value. Nonsingularity guarantees that Δ_* is completely integrable and therefore produces a well behaved foliation of the neighborhood into d -dimensional slices. Note that complete integrability is a local property of distributions.

This assumption of nonsingularity can be removed, and a global partition of the state space can be found, if Δ_* satisfies a global version of complete integrability called the *maximal integral submanifolds* (MIS) property. This property makes the foliation induced by a distribution well-behaved on a global scale. An *integral submanifold* $\mathcal{S}$ of a distribution Δ has a tangent space $T_{\mathbf{p}}\mathcal{S}$ equal to Δ at every point $\mathbf{p} \in \mathcal{S}$. A *maximal integral submanifold* $\mathcal{M}$ of Δ is a connected integral submanifold of Δ which is maximal in the sense that every other integral submanifold of Δ that contains $\mathcal{M}$ is equal to $\mathcal{M}$. Therefore, any two maximal integral submanifolds passing through a state space point $\mathbf{p}$ must be the same maximal integral submanifold. An *MIS distribution* Δ is one that partitions the entire state space into a set of non-intersecting maximal integral submanifolds. Notice that the submanifolds in the partition do not have to have the same dimensionality, and so Δ can be nonsingular. The important thing is that an MIS distribution induces a globally well-behaved partition of the space into manifolds.

There is a theorem by Sussman that describes the necessary and sufficient conditions on a distribution that give it the MIS property. This theorem states that, Δ is MIS iff for every $\tau \in \Delta$ and for every pair $(t, p) \in \Delta_R \times \mathcal{M}$ (where $\mathcal{M}$ is the state space manifold) such that the flow Φ_t^τ of τ is defined, the differential Φ_{t*}^τ at $\mathbf{p}$ maps the subspace $\Delta(\mathbf{p})$ into the subspace $\Delta(\Phi_t^\tau(\mathbf{p}, 0))$. While conceptually simple, this condition is very difficult to test, because it requires that one find a solution to the flows Φ_t^τ associated with all the vector fields $\tau \in \Delta$. In practice, to avoid this requirement, we look for classes of distributions that are guaranteed to

have the MIS property if they also have some other property present which is easy to test. Fortunately, there are some important classes of distributions that exhibit this behaviour. For example, a Locally Finitely Generated (LFG) distribution (see Appendix B) is MIS if it is involutive. Similarly, nonsingular distributions are MIS iff they are involutive, and analytic distributions on a real analytic manifold are MIS iff they are involutive. Involutivity is usually easy to test. For the purposes of this thesis we will be interested in distributions that are LFG.

We now define Δ_C the global version of the distribution Δ_* . First we define the Control Lie Algebra C , the smallest subalgebra of $V(\mathcal{M})$ which contains the vector fields $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$. C is defined as the span of the set of vector fields composed of multibrackets of the form $[\tau_i, [\tau_j, \dots [\tau_k, \tau_l]]]$ where $\tau_i, \tau_j, \dots, \tau_k, \tau_l \in \{\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m\}$. We can use C to define the distribution $\Delta_C = \text{span}\{\tau : \tau \in C\}$. In order to ensure that Δ_C has the MIS property, we will also make the reasonable assumption in this thesis that Δ_C is LFG. Since it is also involutive (it is the span of a subalgebra which is by definition involutive), it is therefore MIS.

In general, Δ_C is a subset of Δ_* at every point $\mathbf{p} \in \mathbb{R}^n$. However, when $\mathbf{p}$ is a regular point of Δ_C (see Appendix B for a definition of regular points), then

$$\Delta_C = \Delta_* + \text{span}\{\mathbf{f}\} = \Delta_R$$

in the neighborhood of $\mathbf{p}$. When Δ_C turns out to be nonsingular, this equality will hold on the entire state space, and all the maximal integral manifolds that partition the space will have the same dimension (same as Δ_R). On the other hand, if Δ_C is singular, then the maximal integral submanifolds will have different dimensionalities, but will correspond to the locally determined foliations at each regular point.

It can be further shown that the reachable points in the global analysis will be an open set of the maximal integral submanifold that passes through the initial state value.

3.4 Observability and the Equivalence of Initial Values

Recall that in the plant described by equation (3.2.1) the state dynamics are modified by output functions. This transformation produces $\mathbf{y}$, a corrupted version of the state that possibly ‘hides’ certain components of the state dynamics behind the kernel of the output transformation (3.2.2). In this section we ask the question, “For which initial values will the output dynamics be equivalent?” The following definitions formalize this equivalence.

Definition 3.2

We say that two states $\mathbf{x}_1$, and $\mathbf{x}_2$ are indistinguishable under input $\mathbf{u}(t)$ if and only if the output $\mathbf{h}(\mathbf{x}(t))$ of the system is the same for $t \geq 0$ when the system trajectory is initialized at either state. Otherwise we say that they are distinguishable.

It happens that this question is mathematically dual to that for reachability, and consequently the development of the theory is very similar. In analogy with the definition for complete controllability, we make the following definition:

Definition 3.3

We say that a system is observable or that it has the property of complete observability if, for any two state values $\mathbf{x}_1$, and $\mathbf{x}_2$, these states can be distinguished under piecewise constant inputs.

3.4.1 Local Observability Analysis

Since we want to investigate the effect of the output function kernel $\mathbf{H}(\mathbf{x}) = \text{span}\{\partial \mathbf{h}_1(\mathbf{x}) \dots \partial \mathbf{h}_p(\mathbf{x})\}^\perp$ on the state space dynamics, it is perhaps not surprising that the mathematical entity equivalent to Δ_* is a co-distribution. It is characterized as the smallest, involutive co-distribution

$$\Omega_*(\mathbf{x}) = \langle \mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x}) | \mathbf{H}(\mathbf{x}) \rangle$$

that contains the co-distribution $\mathbf{H}(\mathbf{x})$, and is invariant under the vector fields $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$. Ω_* is unique, and it can be computed by application of a recursive

algorithm that is dual to recursion (3.3.1). namely

$$\begin{aligned}\Omega_0 &= \mathbf{H}(\mathbf{x}) \\ \Omega_n &= \Omega_{n-1} + \mathcal{L}_{\mathbf{f}}\Omega_{n-1} + \sum_{i=1}^m \mathcal{L}_{\mathbf{g}_i}\Delta_{n-1}.\end{aligned}\quad (3.4.1)$$

As in the case of reachability, this recurrence converges on Ω_* .

Having found Ω_* we turn our attention to its dual distribution $\Delta_Q = \Omega_*^\perp = \langle \mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x}) | H(\mathbf{x}) \rangle^\perp$. Δ_Q is the biggest distribution invariant under $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$ and contained in the kernel of the output transformation. Vector fields contained in Δ_Q will therefore not influence the output $\mathbf{y}$. Since Δ_Q is nonsingular and involutive and therefore completely integrable, it also induces a local foliation of the state space just like Δ_* (refer to Figure 3.2).

Just as we did for Δ_* can find a nonlinear coordinate transformation for Δ_Q which decomposes the system. In these coordinates the plant has the form

$$\begin{aligned}\dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{z}_2) + \sum_{i=1}^m \mathbf{g}_{1,i}(\mathbf{z}_1, \mathbf{z}_2) u_i \\ \dot{\mathbf{z}}_2 &= \mathbf{f}_2(\mathbf{z}_2) + \sum_{i=1}^m \mathbf{g}_{2,i}(\mathbf{z}_2) u_i \\ \mathbf{y} &= \mathbf{h}_2(\mathbf{z}_2)\end{aligned}$$

From this decomposition we see that only the $(n-d)$ -dimensional $\mathbf{z}_2$ dynamics influence the output. The d -dimensional $\mathbf{z}_1$ dynamics are ‘hidden’ behind the output kernel. This means that Δ_Q partitions the space into d -dimensional manifolds in which the last $n-d$ coordinates are constant under the above transformation. Any state within such a manifold produces identical dynamical behaviour. In other words, Δ_Q produces a foliation of state space into equivalence classes of initial values.

3.4.2 Global Observability Analysis

The development of the global theory of observability is parallel to that of global reachability. We consider the observation space $\mathcal{O}$ which is the smallest

subspace of C^∞ which contains the functions $h_1, \dots, h_p$, and is closed under differentiation along the vector fields $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$. With this subalgebra we define the global version of Ω_* as $\Omega_O = \text{span} \{ \partial \lambda : \lambda \in O \}$. The rest of the development is similar to that for global controllability.

3.4.3 Interactions Between Reachability and Observability Manifolds

To better understand the relationship between reachability and observability manifolds consider Figure 3.4. Manifold $\mathcal{R}$ is a reachability manifold Figure 3.4(a), and manifold $\mathcal{O}$ is an observability manifold. The intersection $\mathcal{S} \subset \mathcal{R}$ between these manifolds in Figure 3.4(b) represents a set of initial plant states that will produce identical output dynamics $\mathbf{y}(t)$ for the same input signal $\mathbf{u}(t)$.

On a local region where the reachability and observability foliations are both defined, their intersection partitions the reachability manifolds into sets of points that are equivalent. These sets are not necessarily submanifolds of the reachability manifolds since they may not be deformable to a subset of $\mathbb{R}^d$ at some points. For example, the dashed intersection of the two manifolds in Figure 3.4(c) is not a manifold because it bifurcates. However, except for pathological cases, these intersection sets are often manifolds except in the local region of a ‘small’ number of points.

3.5 The Relative Degree Normal Form

An important research trend emerged in the nonlinear control literature during the 1980’s. Its purpose was to find special state space coordinates in which particular aspects of a system’s dynamical structure are made transparent. When expressed in these coordinates, the resulting normal form makes it possible to clearly ‘see’ the connection between driving inputs, state dynamics, and system outputs. Putting a system into an appropriate normal form can significantly simplify analysis. In fact, placing the system into normal form is prerequisite to almost all nonlinear

Figure 3.4: Observability Manifolds Partition Reachability Manifolds

feedback techniques.

We will be concerned with two closely related normal forms in this section. The first is actually a special case of the second. The purpose of these normal forms is to clarify the connection between the inputs and the output dynamics. Intuitively, the change of coordinates ‘straightens out’ the distortion of state space caused by the system output function, as much as this is possible given that the output function kernel ‘hides’ some components of the state space dynamics. The motivation for this normal form originated with the need to find a transformation in which the dynamics of a nonlinear system become as linear as possible. The resulting normal form modularizes the state space dynamics into subspaces that have dynamics associated with a single output. These modules are either entirely decoupled, or are coupled in a structured and easily understood way. The normal form is particularly useful for understanding the effect of feedback on the output dynamics.

The normal form of a connectionist network can help clarify its dynamical structure. This is particularly true for networks that incorporate models of diffusive excitation.

3.5.1 Vector Relative Degree

Some systems have a local property called vector relative degree which greatly simplifies their analysis. Although this property is not generally encountered, it is quite reasonable to expect connectionist systems to have it, since each output function of the network depends on the activation value of only one neuron, and these output functions are generally monotonic. Furthermore, it is often possible to use dynamic compensation to give a system vector relative degree [Isi89, Section 7.5].

Relative degree is the number of times an output channel must be differentiated before some input explicitly influences it. This concept is easiest to understand in the context of a single input-single output system. We say that the system has relative degree r at a point $\mathbf{x}_0$ if

1. $\mathcal{L}_{\mathbf{g}}\mathcal{L}_{\mathbf{f}}^k h(\mathbf{x}) = 0, k < (r - 1)$, for all $\mathbf{x}$ in an open neighborhood of $\mathbf{x}_0$, and
2. $\mathcal{L}_{\mathbf{g}}\mathcal{L}_{\mathbf{f}}^{r-1} h(\mathbf{x}_0) \neq 0$.

The set of points where we can define vector relative degree will be an open and dense subset of state space (ideally the whole state space). The meaning of relative degree is made clear by taking the derivative of the output of a system with $r > 1$ as follows

$$\begin{aligned} y(t) &= h(\mathbf{x}(t)) = h(\mathbf{x}_0) \\ y^1(t) &= \frac{\partial h}{\partial \mathbf{x}} \frac{d\mathbf{x}}{dt} = \frac{\partial h}{\partial \mathbf{x}} \dot{\mathbf{x}}(t) = \frac{\partial h}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x}(t)) + \mathbf{g}(\mathbf{x}(t)) u(t)] \\ &= \mathcal{L}_{\mathbf{f}} h(\mathbf{x}(t)) + \mathcal{L}_{\mathbf{g}} h(\mathbf{x}(t)) u(t) = \mathcal{L}_{\mathbf{f}} h(\mathbf{x}(t)). \end{aligned}$$

Since $r > 1$, we have $\mathcal{L}_{\mathbf{g}} h(\mathbf{x}(t)) = 0$, and so the input $u(t)$ has no influence on the output. Continuing to differentiate the output gives us

$$\begin{aligned} y^2(t) &= \frac{\partial \mathcal{L}_{\mathbf{f}} h}{\partial \mathbf{x}} \frac{d\mathbf{x}}{dt} = \frac{\partial \mathcal{L}_{\mathbf{f}} h}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x}(t)) + \mathbf{g}(\mathbf{x}(t)) u(t)] \\ &= \mathcal{L}_{\mathbf{f}}^2 h(\mathbf{x}(t)) + \mathcal{L}_{\mathbf{g}} \mathcal{L}_{\mathbf{f}}^2 h(\mathbf{x}(t)) u(t) = \mathcal{L}_{\mathbf{f}}^2 h(\mathbf{x}(t)) \\ &\vdots \end{aligned} \tag{3.5.1}$$

$$y^k(t) = \mathcal{L}_{\mathbf{f}}^k h(\mathbf{x}(t)) \quad \text{for } k < r. \quad \text{and,}$$

$$y^r(t) = \mathcal{L}_{\mathbf{f}}^r h(\mathbf{x}(t)) + \mathcal{L}_{\mathbf{g}} \mathcal{L}_{\mathbf{f}}^{r-1} h(\mathbf{x}(t)) u(t). \tag{3.5.2}$$

So we see that the relative degree r is equal to the number of times one has to differentiate the output $y(t)$ in order to have the value $u(t)$ of the input explicitly appear.

The multivariable generalization of this property is called *vector relative degree*. The idea is to form a matrix where each row represents an output channel h_i , and each column represents the term $\mathcal{L}_{\mathbf{g}_j} \mathcal{L}_{\mathbf{f}_i}^{r_i-1} h_i \mathbf{x}$ for input signal u_j . $r_i - 1$ is the number of times that h_i needs to be differentiated before at least one input signal directly influences it. So we say that a system has vector relative degree $\{r_1, \dots, r_p\}$ when

1. $\mathcal{L}_{\mathbf{g}_j} \mathcal{L}_f^k h_i \mathbf{x} = 0, 1 \leq i, j \leq m$, for all $\mathbf{x}$ in the neighborhood of $\mathbf{x}_0$.

$$2. \mathbf{A}(\mathbf{x}) = \begin{bmatrix} \mathcal{L}_{\mathbf{g}_1} \mathcal{L}_f^{r_1-1} h_1(x) & \mathcal{L}_{\mathbf{g}_m} \mathcal{L}_f^{r_1-1} h_1(x) \\ \mathcal{L}_{\mathbf{g}_1} \mathcal{L}_f^{r_2-1} h_2(x) & \mathcal{L}_{\mathbf{g}_m} \mathcal{L}_f^{r_2-1} h_2(x) \\ \dots & \vdots & \dots \\ \mathcal{L}_{\mathbf{g}_1} \mathcal{L}_f^{r_p-1} h_p(x) & \mathcal{L}_{\mathbf{g}_m} \mathcal{L}_f^{r_p-1} h_p(x) \end{bmatrix}, \text{ and}$$

3. $\mathbf{A}(\mathbf{x})$ has rank p , where $p \leq m$.

$\mathbf{A}(\mathbf{x})$ is often called the *decoupling matrix*, for reasons that will be made clear in Section 3.6.5. Condition 3 is a crucial aspect of vector relative degree that makes it possible to simplify the analysis of these structures, and treat the system as if it were composed of p single input-single output systems. It is also a very restrictive condition. It can be shown that $\sum_{j=1}^m pr_j \leq n$.

3.5.2 Finding the Coordinate Transformation

Our goal now is to find a coordinate transformation that puts a system with vector relative degree into a useful normal form. A coordinate transformation must be an n -dimensional diffeomorphic mapping in order to be invertible. This means that it must have a Jacobian that is nonsingular. As it happens, one can show that the row vectors (i.e. covectors)

$$d\mathcal{L}_f^k h_i(\mathbf{x}), \quad 1 \leq k \leq (r_i - 1) \quad (3.5.3)$$

are all linearly independent for a system with vector relative degree $\{r_1, \dots, r_p\}$. Therefore, the functions

$$\phi_{r_i}^i(\mathbf{x}) = \mathcal{L}_f^k h_i(x), \quad 1 \leq i \leq p, 1 \leq k \leq (r_i - 1)$$

form the first $r = \sum_{j=1}^m pr_j$ elements of a coordinate transformation

$$\Phi(x) = \left[\phi_1^1(\mathbf{x}) \quad \dots \quad \phi_{r_1}^1(\mathbf{x}) \quad \dots \quad \phi_{r_1}^{p_1}(\mathbf{x}) \quad \phi_{r_1+1}^1(\mathbf{x}) \quad \dots \quad \phi_{r_1+1}^1(\mathbf{x}) \right]^T$$

All that remains is to find $n - r$ functions that have covectors

$$\partial\phi_{r-1}^1(\mathbf{x}) \dots \partial\phi_{r+1}^1(\mathbf{x})$$

that are linearly independent from the covectors in (3.5.3). It can be shown that it is always possible to find these functions.

3.5.3 Deriving the Normal Form

We can now derive the normal form itself. Since $\mathbf{z} = \Phi(x)$ is a manifold mapping, the Jacobian is its differential $\Phi_*(\mathbf{x}) = \partial\Phi(x)$. As shown in Section B.8, the differential maps the tangent spaces of the domain manifold into the tangent spaces of the range manifold. In order to express the system evolution in the new coordinates we must use the differential to transform the time varying vector field $\dot{\mathbf{x}}(t)$ from $T_{\mathbf{x}}\mathcal{M}$ to $T\Phi(\mathcal{M})\mathbf{z} = \Phi(x)$. In other words, we need to compute $\dot{\mathbf{z}} = \partial\Phi(x) \dot{\mathbf{x}}$. In order to elucidate the general form of $\dot{\mathbf{z}}$, we can make a direct calculation of the derivative of $\Phi(\mathbf{x}(t))$ with respect to time. Referring to (3.5.2), we see that for each output function $\phi_i^1 = h_i(\mathbf{x}(t))$, $1 \leq i \leq p$, we have

$$\begin{aligned} \frac{d\phi_1^1}{dt} &= \phi_2^1 \\ &\vdots \\ \frac{d\phi_{r_i-1}^1}{dt} &= \phi_{r_i-1}^1 \\ \frac{d\phi_{r_i}^1}{dt} &= \mathcal{L}_f^{r_i} h_i(\mathbf{x}(t)) + \sum_{j=1}^m \mathcal{L}_{\mathbf{g}_j} \mathcal{L}_f^{r_i-1} h_i(\mathbf{x}(t)) u_j(t). \end{aligned}$$

So the normal form for $\mathbf{z} = \begin{bmatrix} \xi^1 & \dots & \xi^p & \eta \end{bmatrix}^T$ becomes

$$\begin{aligned} \dot{\xi}_1^i &= \xi_2^i \\ &\vdots \\ \dot{\xi}_{r_i-1}^i &= \xi_{r_i}^i \\ \dot{\xi}_{r_i}^i &= \mathbf{b}_i(\xi, \eta) + \sum_{j=1}^m \mathbf{a}_{i,j}(\xi, \eta) u_j \\ \mathbf{y}_i &= \xi_1^i \end{aligned}$$

where $\mathbf{b}_i(\xi, \eta) = \mathcal{L}_f^{r_i} h_i(\Phi^{-1}(\xi, \eta))$, and $\mathbf{a}_{i,j}(\xi, \eta) = \mathcal{L}_{\mathbf{g}_j} \mathcal{L}_f^{r_i-1} h_i(\Phi^{-1}(\xi, \eta))$.

In the event that $r = n$, the normal form is composed of exactly p such modules, and η is 0-dimensional. In the event that $r < n$ we had to find $n - r$ additional functions for the coordinate transformation, there will be an additional $(n - r)$ -dimensional dynamical component

$$\dot{\eta} = \mathbf{q}(\xi, \eta) + \sum_{j=1}^m \mathbf{p}_j(\xi, \eta) u_j.$$

It can be shown that if the span is involutive, the additional functions can always be selected so that

$$\mathcal{L}_{g_j} \phi_i(x) = 0.$$

for $(r + 1) \leq i \leq n$, and $1 \leq j \leq m$. This produces a slightly simpler normal form since the $\mathbf{p}_j(\xi, \eta) = 0$ when this is true.

3.5.4 What Does the Normal Form Mean?

We shall now consider the physical interpretation of the subspace ξ^i . We see from this remarkable normal form that the dynamics of output channel y_i is determined by the dynamics within subspace $\text{span}\{\xi^i\}$. Within this distribution the inputs drive the dynamics of one of the axes. The dynamics of the other axes

represent the outputs of a chain of linear integrators driven by the value of this first axis. The dynamics of the last axis in this series corresponds to the output dynamics. This modular form is repeated for each output channel. As we will see in Section 3.6, we can take advantage of this simple modular structure to design a number of useful feedback control systems.

The interpretation of the subspace $\text{span}\{\eta\}$ is just as clear. If the kernel of $\mathbf{h}$ is nonempty (it is a singular transformation), then $r < n$ and η represents the output zeroing submanifold of the system. As long as the system state is forced to remain within this submanifold, the output of the system will be 0. To see this, consider setting the initial value of the system to $\mathbf{z}_0 = \begin{bmatrix} \mathbf{0} & \eta \end{bmatrix}^T$. Clearly, each module will output 0 at this point, giving us $\mathbf{h}(\mathbf{x}_0) = 0$. Now, if we apply an appropriate input function to counteract the drift dynamics we can keep the system state in this submanifold. An appropriate input is described in Section 3.6.1.

3.6 Feedback Control of Dynamics

As we saw in Section 3.2.1, the goal of control theory is to find an input equation $\hat{\mathbf{u}}(t)$, called a controller, that makes a plant behave in a desired way. This behaviour is actually the dynamics of a new plant formed from the combination of the old plant and the controller, i.e. $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{j=1}^m \mathbf{g}_j(\mathbf{x}) \hat{\mathbf{u}}_j$. We refer the desired behaviour as the *control objective*. The controller achieves the new dynamics by linearly combining the input vector fields $\mathbf{g}_1, \dots, \mathbf{g}_m$ to form a new vector field which is an element of the distribution $\Delta_G(\mathbf{x}) = \text{Im}(\mathbf{G}(\mathbf{x})) = \text{span}\{\mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})\}$. This new vector field $\mathbf{G}(\mathbf{x}) \hat{\mathbf{u}}$ offsets the drift dynamics $\mathbf{f}(\mathbf{x})$ to produce the desired dynamics $\dot{\mathbf{x}}$.

Over the years, several control objectives have repeatedly proven useful during the design of a wide variety of systems. Much of control theory is devoted to finding controllers that satisfy these reusable objectives, as well as providing general techniques that can be used to satisfy custom control objectives. Even from the

earliest days of control, it was apparent that controllers must often be functions of the plant state or output. When the state or output is fed back to the function that determines the next input we say that the plant is driven by a *feedback controller*. The reason for this is intuitively appealing: in order to know what input to inject into the system at a given point in time, the controller must know what the current state of the system is in order to know which drift dynamics vector has to be offset. We call the original, uncontrolled system equation the *open loop* system equation, and the equation incorporating a feedback controller the *closed loop* system equation.

Ideally, the system state variable is available for feedback into the controller equation. When this is possible the controller equation is particularly simple. In practice, though, the state variable is sometimes only a mathematical abstraction of the true internal workings of the physical plant. In these cases, the output, a corrupted form of the state variable, is fed into the controller instead, and the state is reconstructed from it. Because of the loss of information incurred by the output function, only a partial reconstruction of the state may be possible, limiting the amount of control that can be exerted over the system dynamics.

Two types of feedback controller have been studied extensively. The first, called a *static feedback controller*, is of the form

$$\hat{\mathbf{u}} = \alpha(x) + \beta(x) \mathbf{v}$$

where $\alpha(x)$ is a mapping of state space to input space. In the event that the closed system has q inputs, the q -dimensional function of time $\mathbf{v}(t)$ represents the new input vector, and $\beta(x)$ is a matrix with q vector field columns that performs the same role as $\mathbf{G}(\mathbf{x})$ in the open system. The closed system equations form a new system

$$\dot{\mathbf{x}} = \hat{\mathbf{f}}(\mathbf{x}) + \hat{\mathbf{G}}(\mathbf{x}) \mathbf{v},$$

where

$$\begin{aligned} \hat{\mathbf{f}}(\mathbf{x}) &= \mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x}) \alpha(x), & \text{and} \\ \hat{\mathbf{G}}(\mathbf{x}) &= \mathbf{G}(\mathbf{x}) \beta(x). \end{aligned}$$

Static feedback produces a new drift dynamics by offsetting the old drift dynamics with a linear combination of the original input vector fields. The mappings $\alpha(x)$ and $\beta(x)$ produce appropriate linear combinations for each point of the state space.

The other type of feedback equation is called a *dynamic feedback controller*. In this controller we introduce a second, r -dimensional dynamical system that is driven by the original system state. The state of this new system is fed into the controller equation along with the state of the original system. The controller is therefore described by two equations

$$\begin{aligned}\dot{\mathbf{w}} &= \gamma(\mathbf{x}, \mathbf{w}), \\ \dot{\mathbf{u}} &= \alpha(\mathbf{x}, \mathbf{w}) + \beta(\mathbf{x}, \mathbf{w}) \mathbf{v}.\end{aligned}$$

Dynamic feedback controllers are a generalization of static feedback controllers. We may think of the controller as expanding the state space of the original open system into an $(n+r)$ -dimensional system whose state is given by the vector $\begin{bmatrix} \mathbf{x} \\ \mathbf{w} \end{bmatrix}$ and whose state space is the cartesian product of the two state spaces. The new expanded system is referred to as an *exosystem*. From this viewpoint, the controller is just a static feedback controller for the exosystem. The difference, then, between a dynamic and static feedback controller, is that a dynamic feedback controller expands the dynamical repertoire of the original system⁵. The state of the system $\mathbf{w}$ can act as a summary of the history of $\mathbf{x}$. The vector field that offsets the drift dynamics of the system is not only dependent on the current state of $\mathbf{x}$, but it is also sensitive to this history. Dynamic feedback controllers are therefore, in some sense, more powerful than static feedback controllers. However, the associated theory is also more complex, so there are more results for static feedback controllers.

In the following sections we present background theory for feedback controllers. To simplify the discussion, the plants described below are assumed to be in vector relative degree normal form as described in Section 3.5.1.

⁵This is often referred to as *dynamic compensation*.

3.6.1 Output Zeroing Dynamics

The system output function $\mathbf{y} = \mathbf{h}(\mathbf{x})$ maps the state space manifold to an output manifold. In this section we consider a static feedback controller that restricts reachability of the system state to the submanifold of states that are mapped to $\mathbf{y} = \mathbf{0}$.

As we saw in Section 3.5.1, a coordinate transformation

$$\mathbf{z} = \Phi(x) = \begin{bmatrix} \xi^1 & \dots & \xi^p & \eta \end{bmatrix}^T$$

can be found in which a vector relative degree system has normal form

$$\begin{aligned} \dot{\xi}_1^i &= \xi_2^i \\ &\vdots \\ \dot{\xi}_{r_i-1}^i &= \xi_{r_i}^i \\ \dot{\xi}_{r_i}^i &= \mathbf{b}_i(\xi, \eta) + \sum_{j=1}^m \mathbf{a}_{i,j}(\xi, \eta) u_j \\ \dot{\eta} &= \mathbf{q}(\xi, \eta) + \sum_{j=1}^m \mathbf{p}_j(\xi, \eta) u_j \\ \mathbf{y} &= \begin{bmatrix} \xi_1^1 & \xi_1^2 & \dots & \xi_1^p \end{bmatrix}^T \end{aligned} \tag{3.6.1}$$

It is clear that forcing the outputs $\mathbf{y}_i = \mathbf{i}_{\leq}^1 \mathbf{0}$ requires that

$$\mathbf{y}_i = \mathbf{i}_{\leq}^{r_i} \mathbf{b}_i(\mathbf{0}, \eta) + \sum_{j=1}^m \mathbf{a}_{i,j}(\mathbf{0}, \eta) u_j = 0, \quad \text{for } 1 \leq i \leq p.$$

We can rewrite this requirement in vector notation as

$$\begin{aligned} &\mathbf{b}(\mathbf{0}, \eta) + \mathbf{A}(\mathbf{0}, \eta) \mathbf{u} \\ &= \begin{bmatrix} \mathbf{b}_1(\mathbf{0}, \eta) \\ \vdots \\ \mathbf{b}_p(\mathbf{0}, \eta) \end{bmatrix} + \begin{bmatrix} \mathbf{a}_{1,1}(\mathbf{0}, \eta) & \dots & \mathbf{a}_{1,m}(\mathbf{0}, \eta) \\ \vdots & \dots & \vdots \\ \mathbf{a}_{p,1}(\mathbf{0}, \eta) & \dots & \mathbf{a}_{p,m}(\mathbf{0}, \eta) \end{bmatrix} \mathbf{u} = \mathbf{0}. \end{aligned}$$

Recall, from Section 3.5.1, that the decoupling matrix $\mathbf{A}(\mathbf{z})$ is of rank p , and therefore we can always solve for the input function $\mathbf{u}(t)$ in the neighbourhood of the initial value $\mathbf{z}_0 = \begin{bmatrix} \mathbf{0} & \eta \end{bmatrix}^T$.

A static state controller for the output zeroing problem is given by⁶

$$\dot{\mathbf{u}} = -\mathbf{A}^-(\mathbf{0}, \eta) \mathbf{b}(\mathbf{0}, \eta).$$

Under this controller the system evolution equation will sum to a vector field that is tangent to a submanifold of the manifold of states that zero the output.

3.6.2 Exact Linearization of System Dynamics

In normal form coordinates, the dynamics of nonlinear systems can sometimes be made linear and controllable by a static state feedback controller. It can be shown that this is possible iff

$$r = \sum_{j=1}^p r_j = n.$$

In this case $\mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} \xi^1 & \dots & \xi^p \end{bmatrix}^T$ and we need not consider the dynamics η . Linearizing the dynamics of a system is often desirable because it allows us to apply results from linear control theory to further alter the dynamics of the system. This gives us a larger toolbox of control techniques to draw on. For example, we can always find a controller that assigns *any* desired eigenvalues to the dynamics of a linear and controllable system. As we will see, the feedback controllers described in subsequent chapters all employ variations on the linearizing controller.

Assuming that the condition $r = n$ is satisfied, if we select the controller

$$\dot{\mathbf{u}} = \mathbf{A}^+(\xi) [-\mathbf{b}(\xi) + \mathbf{v}]$$

(using the vector notation developed in the previous section), the controlled system

⁶ $\mathbf{M}^+$ is the Moore-Penrose pseudo-inverse of the matrix $\mathbf{M}$. In the event that $p = m$, this reduces to the usual matrix inverse.

has the simple form

$$\begin{aligned}
 \dot{\xi}_1^i &= \xi_2^i \\
 &\vdots \\
 \dot{\xi}_{r_i-1}^i &= \xi_{r_i}^i \\
 \dot{\xi}_{r_i}^i &= j_i \quad \text{for } 1 \leq i \leq p.
 \end{aligned} \tag{3.6.2}$$

We may express the system in vector form as the linear system

$$\begin{aligned}
 \dot{\mathbf{z}} &= \text{diag} \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_p \end{bmatrix} \mathbf{z} + \text{diag} \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_p \end{bmatrix} \mathbf{v} \\
 &= \mathbf{A} \mathbf{z} + \mathbf{B} \mathbf{v}
 \end{aligned}$$

where

$$\mathbf{A}_i = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & & 1 \\ 0 & 0 & 0 & & 0 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

It is well known from linear control theory that a linear system in this form, the *Brunowsky canonical form*, is controllable.

In the event that $r < n$, the dynamics cannot be entirely linearized. However, we can still partially linearize it, and in this case the equations above still hold, but we also have the equation

$$\begin{aligned}
 \dot{\eta} &= \mathbf{q}(\xi, \eta) + \mathbf{P}(\xi, \eta) \dot{\mathbf{u}} \\
 &= \mathbf{q}(\xi, \eta) + \mathbf{P}(\xi, \eta) \mathbf{A}^\top(\xi, \eta) [-\mathbf{b}_0(\xi, \eta) + \mathbf{v}]
 \end{aligned} \tag{3.6.3}$$

3.6.3 Asymptotic Stabilization

It is no exaggeration to say that control theory owes its existence to the need to stabilize systems through feedback. The earliest motivation for a theory of control

came from the need to understand how mechanical governors stabilize the operation of steam engines through feedback. In the more abstract setting, any system that operates in the real world must keep its operational parameters within a 'safe' range. Chemical plants must not blow up, internal combustion engines must not increase their speed until they seize through overheating, and neurons in the brain must not all saturate or oscillate uncontrollably. In other words, the state trajectories must remain near an equilibrium state⁷ somewhere in the safe operating range of the state space. If the drift dynamics of a system is unstable, i.e. it produces trajectories that escape from this safe region, a feedback controller can be designed that compensates for the drift and keeps the state near a desirable 'set point'.

There are many different definitions of stability in the control literature. These differ in the degree of attractiveness to the equilibrium point. One of the strongest and most useful types of stability is the concept of asymptotic stability. A system with this property will tend toward the equilibrium state in the limit, and will guarantee that for bounded input signals the outputs will also be bounded.

In this section we see how a static feedback controller can change the dynamics of a nonlinear system to make the system locally asymptotically stable at a point $\mathbf{x}_0$. *Asymptotic stability* means that, when there are no inputs to a system (i.e. $\mathbf{u} = \mathbf{0}$), the drift dynamics will eventually return the system to the equilibrium state $\mathbf{z}_0 = [\xi_0 \quad \eta_0]^T$. This technique is general enough to handle systems that have linearizations with eigenvalues on the imaginary axis.

We begin with the linearizing controller described in the previous section, satisfying

$$\mathbf{A}^+(\xi, \eta) [-\mathbf{b}(\xi, \eta) + \mathbf{v}].$$

The difference is that we allow $r < n$. In this case, using the theory of linear control, we may choose a linear controller $\mathbf{v} = \mathbf{F}\xi$ which stabilizes the matrix $\mathbf{A} + \mathbf{B}\mathbf{F}$ at ξ_0 .

⁷In general, the equilibria might be stable orbits rather than single fixed points. However, as we shall see in Section 3.6.4 the problem of finding a stable orbit can be rephrased in terms of stabilization around a fixed point.

This will asymptotically stabilize the subdynamics ξ at ξ_0 . Notice that, in the closed system, each ξ_j^i is only influenced by $\mathbf{v}_i$, not η . So the stability of ξ is independent of η . However, in order for the entire system state $\mathbf{z} = \begin{bmatrix} \xi & \eta \end{bmatrix}$ to be asymptotically stabilized, subdynamics η must also be asymptotically stable at η_0 . Given that this is true, η will return to its equilibrium η_0 whenever $\mathbf{v} = \mathbf{0}$, as desired. Notice that we can choose a matrix $\mathbf{F}$ that has eigenvalues that return the ξ component of the state to ξ_0 as quickly as desired.

In summary then, as long as the η dynamics are asymptotically stable at $\mathbf{z}_0$, a stabilizing controller exists which is of the form

$$\mathbf{A}^-(\xi, \eta) [-\mathbf{b}(\xi, \eta) + \mathbf{F}\xi].$$

3.6.4 Asymptotic Tracking

An important control objective is to steer the system output along a desired trajectory given by a p -dimensional function $\mathbf{y}^R(t)$. In other words, we want to make the system behave in a repeatable way. A feedback controller that provides just the right input sequence to force the outputs to track the function $\mathbf{y}^R(t)$, despite the natural drift dynamics of the system, is referred to as a *tracking controller*.

A more realistic control objective should include robustness to intermittent noise, either in the initial value of the system, the state space trajectory, or in the controller itself. If this noise happens to push the system output off the target trajectory $\mathbf{y}^R(t)$, the controller should recover by returning to the trajectory as quickly as possible. This is reminiscent of asymptotic stability, where the system returns to an equilibrium value when disturbed. We will refer to this more robust control objective as *asymptotic tracking*.

Asymptotic tracking controllers are particularly easy to design for vector relative degree systems in normal form. We can base them on the linearizing controller described in Section 3.6.2. Recall that the output signal y_i of a closed system

controlled by a linearizing tracker is governed by the subspace dynamics:

$$y_i = \mathbf{i}^1$$

where

$$\begin{aligned} \dot{\xi}_1^i &= \xi_2^i \\ &\vdots \\ \dot{\xi}_{r_i-1}^i &= \xi_{r_i}^i \\ \dot{\xi}_{r_i}^i &= \mathbf{j}, \quad \text{for } 1 \leq i \leq p. \end{aligned} \tag{3.6.4}$$

Our goal now is to make $y_i = y_i^R(t)$. If we choose the controller input

$$v_i = y_i^{Rr_i} - \sum_{j=1}^{r_i} c_{j-1} \left(z_j - y_i^{Rj-1} \right).$$

then y_i will asymptotically track y_i^R as desired if we select an appropriate set of real valued coefficients $c_0 \dots c_{r_i-1}$. To see this, consider defining an error function $e_i = y_i - y_i^R$. We may re-express v_i as

$$v_i = y_i^{Rr_i} - \sum_{j=1}^{r_i} c_{j-1} e_i^{j-1}.$$

Now we have

$$\begin{aligned} \dot{z}_{r_i} &= y_i^{r_i} \\ &= y_i^{Rr_i} - \sum_{j=1}^{r_i} c_{j-1} e_i^{j-1} \\ \Leftrightarrow y_i^{r_i} - y_i^{Rr_i} &= - \sum_{j=1}^{r_i} c_{j-1} e_i^{j-1} \\ \Leftrightarrow \sum_{j=0}^{r_i} c_j e_i^j, \quad \text{where } c_{r_i} &= 1. \end{aligned} \tag{3.6.5}$$

From this last equation (3.6.5) we see that the error function $e_i(t)$ satisfies an order r_i differential equation whose roots are controlled by the values of the coefficients c_i , $0 \leq i \leq r_i - 1$. By selecting coefficients that make the roots negative, the error function $e_i(t)$ is made to converge to 0 with arbitrarily fast exponential decay. Since the output $y_i = y_i^R$ when $e_i(t) = 0$, the effect of the input is to make the output of the system 'track' the target output. Clearly, this process can be repeated for each output channel.

A system controlled by the tracking controller described above will only exhibit internal stability⁸ if the dynamics of the vector η are stable. For completeness sake, we should also mention that other types of controllers can achieve internal stability and asymptotic tracking when this dynamics is unstable.

3.6.5 Decoupling Feedback

Given a system with m inputs and p outputs, it is sometimes useful to use feedback to reduce the system (from the input-output point of view) to an aggregate of p independent single-input single-output channels. We say that such a feedback *decouples* the system. A system is said to be *block decoupled* when the input and output sets are each partitioned into q subsets, and each input subset influences exactly one subset of outputs.

The main result for decoupling controllers is the fact that a system can be decoupled iff it has vector relative degree. In fact, the linearizing controller with inputs described in Section 3.6.2 also decouples a nonlinear system. In this case $\mathbf{v}$ is p -dimensional. Since the system is linearized, we see that exactly one controller input v_i influences each output $y_i = \xi_i^1$. In the event that $r < n$, the dynamics ξ does not influence any of the output channels, and so it may be ignored. The system is therefore decoupled as well as linearized by this controller.

⁸By *internal stability* we mean that all state variables remain bounded for all $t > 0$. If the subdynamics η is unstable, then the state variables in this subspace of tangent space will increase without bound.

In the case of block decoupling, we can add block diagonal matrices $\mathbf{C} = \text{diag} [\mathbf{C}_1 \ \cdots \ \mathbf{C}_q]$ and $\mathbf{D} = \text{diag} [\mathbf{D}_1 \ \cdots \ \mathbf{D}_q]$ to the controller, where $\mathbf{C}_i$ and $\mathbf{D}_i$ are ρ_i by χ_i matrices. Given the controller

$$\mathbf{A}^-(\xi, \eta) [-\mathbf{b}(\xi, \eta) + \mathbf{C}\xi].$$

we see that the block of outputs from are influenced by the block of inputs from . The coefficients of $\mathbf{C}$ and $\mathbf{D}$ determine the dynamics of each of the q linear, decoupled subsystems.

Notice that we have complete control over the dynamics of the decoupled subsystems. This means that we can guarantee the internal stability of each subsystem.

3.7 Implications of the Theory for GNF Networks

Recall the form of a GNF network from Chapter 2. The dynamics of an n -dimensional network are determined by a system of n ordinary differential equations of the form

$$\begin{aligned} \dot{x}_i &= a_i(x_i) \left(b_i(x_i) + \sum_{j=1}^n \left(s_{j,i}^-(x_i) H_{i,j}^+ - s_{j,i}^-(x_i) H_{i,j}^- \right) d_j(x_j) \right. \\ &\quad \left. - \sum_{k=1}^m \left(s_{(k+n),i}^-(x_i) J_{i,k}^- - s_{(k+n),i}^-(x_i) J_{i,k}^- \right) u_k \right) \\ &= a_i(x_i) \left[b_i(x_i) + \sum_{j=1}^n \left(s_{j,i}^+(x_i) H_{i,j}^+ - s_{j,i}^-(x_i) H_{i,j}^- \right) d_j(x_j) \right] \\ &\quad + \sum_{k=1}^m \left(a_i(x_i) s_{(k+n),i}^-(x_i) J_{i,k}^- - a_i(x_i) s_{(k+n),i}^+(x_i) J_{i,k}^+ \right) u_k. \end{aligned} \quad (3.7.1)$$

Alternatively, we may express the network in vector notation as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{k=1}^m \mathbf{g}_k(\mathbf{x}) u_k, \quad (3.7.2)$$

where

$$\mathbf{f}(\mathbf{x}) = a_i(x_i) \left[b_i(x_i) + \sum_{j=1}^n \left(s_{j,i}^+(x_i) H_{i,j}^+ - s_{j,i}^-(x_i) H_{i,j}^- \right) d_j(x_j) \right]. \quad (3.7.3)$$

$$\mathbf{g}_k(\mathbf{x}) = \left(a_i(x_i) s_{(k+n),i}^-(x_i) J_{i,k}^- - a_i(x_i) s_{(k+n),i}^+(x_i) J_{i,k}^+ \right). \quad (3.7.4)$$

But equation (3.7.2) is in the same form as equation (3.2.1), and we may therefore conclude that GNF networks are a special type of Isidori-style nonlinear control system. Fortunately then, all of the theory introduced in this chapter is directly applicable to an understanding of the control properties of this general family of continuous-time connectionist networks.

3.7.1 The Implications of Controllability for GNF Networks

Our primary goal in this thesis, as stated in the introduction, is to develop a weight adaptation algorithm which will permit a connectionist system to learn and apply constraint manifolds. As we have already said, although our work is motivated by an interest in GNF networks, we are also interested in exploring hybrid architectures in which we use control notions to modify the basic architecture of a GNF network in useful ways. Nevertheless, a sensible place to start is with an analysis of the control properties of GNF networks.

A constraint manifold is a connected region of activation space that confines the dynamics of a GNF network. If the state of the system is initially an element of the manifold, then all subsequent states are as well. This viewpoint is consistent with the definition of a reachability manifold given in Section 3.3.1. There we saw that the reachability distribution Δ_* induces a foliation of state space into reachability manifolds. Clearly then, a very direct way for a GNF network to learn a constraint manifold is to have it adaptively modify the equations of motion so that the shape of any one leaf of the reachability foliation passes through the desired manifold. In the more general case we may want to approximate the entire state space partition

on some region of interest⁹. In either case it is clear that a reachability analysis is necessary to clearly relate the network weights (that parameterize the vector fields in equations (3.7.3) and (3.7.4)) to the reachability distribution Δ_* . This will be the subject of Chapter 4.

3.7.2 Implications of Observability for GNF Networks

As we have seen in Section 3.4.3 the observability manifolds partition constraint manifolds into sets of states that produce the same observable output trajectory. From a computational viewpoint this means that the constraint manifold is divided into equivalence classes, where two states are computationally equivalent if they belong to the same class, i.e. they produce the same computational behaviour from the viewpoint of an observer that is monitoring the network outputs. To characterize the states that are computationally equivalent in this fashion, we need to perform an observability analysis. This is the subject of Chapter 5.

3.7.3 Implications of Feedback for GNF Networks

In this thesis we are not interested in learning a particular network dynamics. Rather, we develop methods to learn general manifold constraints that can be satisfied by an entire family of dynamics. However, in order to fulfill a particular computational objective, we will also need to select a particular run-time dynamics that is consistent with the constraint. We need to identify an approach for combining the dual goals of learning a constraint manifold and learning a particular run-time dynamics on this constraint manifold.

One approach is to try to combine both goals into a single learning algorithm. This is certainly possible in special cases. Indeed, in Chapter 6 we will describe such an algorithm for the special case of learning linear manifold constraints. An alternative approach is to find an architecture that permits us to combine com-

⁹We study this general version of the problem in Chapter 7.

Figure 3.5: Two Module Feedback Architecture for Learning Dynamics Consistent with a Constraint Manifold

computational modules that are each responsible for one learning task. The advantages of a modular approach to design are well known. Modules permit each learning algorithm to be simpler and easier to analyze than an algorithm that performs both tasks simultaneously. Also, separate modules can be implemented with very different technologies if the interface between them is well defined. Should better algorithms for one of the two learning goals become available at a future time, the appropriate module can be updated with little or no change to the other.

The dual module feedback architecture of Figure 3.5 takes advantage of network output feedback to impose a particular dynamics on the constraint manifolds. As we have seen, feedback modifies the dynamical characteristics of a plant by

steering it with an appropriate choice of input signal. It is a noninvasive technique in that the dynamics of the closed system are achieved without making destructive changes to the plant. The plant module implements the algorithm that learns a set of network vector fields that confine the reachability space of the plant to the desired constraint manifold. The feedback controller module is responsible for learning to generate an appropriate input signal for each plant module output that directs the plant dynamics in the desired direction.

Clearly, the controller cannot generate plant trajectories that violate the manifold constraint that the plant has learned. A feedback controller cannot increase the set of points that can be reached. It can only modify the equations of motion on the reachable set (and possibly make the closed system reachability manifold smaller than that of the open system). Therefore, this architecture is consistent with the manifold constraint abstraction.

Additionally, the studies of Levin and Narendra [LN93], [LN96] and others [MKK97, PL95] suggest that recurrent networks make for particularly effective controllers of nonlinear plants. Therefore, there is both theoretical and empirical justification to expect neural implementations of the controller module to be able to learn to generate the control signals that will give the closed system the desired dynamics on the constraint manifold. In the event that the internal weight matrices of the plant module are available for inspection, and the plant has vector relative degree, its dynamics can be linearized and a conventional linear controller can be designed for the control module.

As we see in Figure 3.6, the same system can exhibit many different dynamics on the same manifold by selecting between the control signals of a bank of controllers, each able to impose its own dynamical goal. We will not elaborate further on this arrangement except to say that it is generally consistent with the 'system of experts' models that have been proposed to combine the knowledge acquired by multiple networks.

Figure 3.6: Imposing Multiple Dynamics on a Manifold

3.8 Chapter Conclusions

In this chapter we have provided an historical perspective on control and a comprehensive survey of control techniques from the differential geometric point of view. We have seen that GNF networks are amenable to analysis by these techniques. We have also seen that our motivation is unusual in that the goals of this thesis are diametrically opposed to those of other control theoretic studies of connectionist systems. Our purpose is to find GNF weight update algorithms that can learn manifold constraints from observations of an evolving domain, without supervision. Furthermore, these constraints must be operational in that they restrict the activation space dynamics of a GNF network to the learned manifold. Rather than attempting to find architectural conditions that maximize the set of achievable activation states (as those other studies do) we seek an unsupervised adaptive process that can place very specific *limitations* on this set. Consequently, our study and the methods we develop are entirely novel.

Having put our concerns and adopted tools in perspective, we are now ready to begin our analysis of GNF networks and search for learning algorithms that are consistent with the manifold constraint perspective.

Chapter 4

The Reachability Properties of GNF Networks

Consider a connectionist network that is driven by time-varying input signals. If our intent is to understand and control the dynamical behaviour of this network, then one of the most fundamental questions we can ask is: “To what degree can we exert control over the network’s activation state through the inputs?” In the context of systems theory, this question is referred to as the controllability (or state reachability) problem. Another important question, called the observability problem, is: “To what extent are the internal state dynamics of a network reflected in the observable output dynamics?” In this chapter and the next we answer these questions from the viewpoint of differential geometric state-space control.

The answer to the controllability question is directly relevant to the problem of developing learning methods for the manifold constraint paradigm. In this chapter we investigate how to adjust the network weights in order to confine the dynamics of the network to a differentiable submanifold of the network’s activation space, even when the dynamics are being driven by a time-varying input signal. Finding a solution to this problem is a matter of understanding how the network weights

parameterize the mathematical structures that connect the inputs to the network dynamics. In fact, we will go further and clarify the relative contributions of both the weights and the non-adaptive¹ terms of the neuron equations to these structures.

Surprisingly, we shall discover that it is not possible to reliably learn manifold constraints for the general class of GNF networks if there is any degree of uncertainty in the values of the weights². On the other hand, it is possible to find weights that reliably represent manifold constraints for several interesting special case GNF networks. However, the classes of constraints that can be learned in these cases are rather limited. We conclude that the family of GNF networks must be modified to permit robust learning of general manifold constraints. How to perform these enhancements is the subject of the second part of the dissertation.

We begin the chapter in Section 4.2 by reviewing the control theoretic tools that are relevant to a study of controllability. The analysis itself is divided along two organizational axes. The first of these distinguishes between networks that are fully connected and those that are not. We make this distinction for both analytical convenience and clarity of exposition. In Section 4.2.4 we study the latter case where the network has ‘open’ connections, i.e. where recurrent connections are treated as if they carry external input signals. This assumption greatly simplifies the analysis and makes it easier to state our final results. A more general analysis of ‘closed’ networks is the subject of Section 4.5. Closed networks have normal (recurrent) connections that propagate the activation state of the network between neurons. As it happens, it is possible to express our results for closed networks in terms of the results for open networks. Once again, this permits a considerable simplification of the analysis, and clarifies how the inter-neuron connections affect the reachability properties of the closed network.

The second axis along which we organize our study divides Sections 4.2.4

¹By non-adaptive terms, we refer to the functions that appear in the GNF network equations that are not parameterized by synaptic weights.

²As might be due, for example, to internal noise in the synapses, or discretization of the stored weight values.

and 4.5 into analyses of GNF networks that have varying degrees of specialization. Our study of open networks in Section 4.3.1 begins with essential definitions and theorems that are relevant to all such networks. Then, we divide our reachability results into those for general GNF networks in Section 4.3.2 and those for various ‘classical’ network models in Section 4.4. Each of these sections is further subdivided into studies of networks that have different mixtures of synaptic valences. The study of closed networks is organized along similar lines. Section 4.5.1 introduces definitions and ancillary theorems, and Section 4.5.2 presents the main results with a description of how the results for the various specialized open networks are extended to the closed network case.

Finally, Section 4.6 wraps up with general discussion and chapter conclusions.

4.1 Contributions of the Chapter

In this chapter we begin a study of the control properties of the general class of autonomous, continuous-time Grossberg Normal Form (GNF) networks introduced in Chapter 2. We succeed in establishing how the feedback weights and input connection weights are related to a set of vector-fields that span a reachability distribution. This distribution defines the tangent bundle of a manifold on which the network dynamics are constrained to evolve from the initial state. This set of reachable states corresponds exactly to a manifold constraint that is imposed by the particular vector of synaptic weights.

To our knowledge, this is the first extensive analysis of the controllability properties of such a general class of continuous-time networks from the perspective of differential geometric state-space control. It is also the first analysis that seeks to impose manifold constraints on the dynamics of the network. The traditional control theoretic view is that restrictions on controllability are an annoyance to the designer, and are to be avoided if possible. Therefore, the usual approach is to

show conditions under which full controllability³ is guaranteed. As we have seen in Appendix A, however, the question of what a connectionist system can compute, and how it achieves this, may be phrased, at least in part, in terms of *limitations* on the state values that can be reached from any given state. Therefore, full controllability is the condition that we wish to avoid, and our intentions and results are diametrically opposed to the few studies of reachability in connectionist systems that have been published [Son97, LN93, LN96].

In particular, we have given a detailed characterization of the reachability distribution for seven related families of GNF networks. This characterization is given in terms of a set of vector fields that span the reachability distribution. First we provide results for the general family of GNF networks. This study examines in turn the effect of three different mixtures of neurons on reachability. We study the case where the network consists of neurons that are heterosynaptic, i.e. able to make both inhibitory and excitatory connections. This is the most general family of GNF networks. We also study the biologically realistic family of networks in which neurons are homosynaptic. In this case each neuron makes connections of only one synaptic valence. Finally, we demonstrate the difference in reachability that occurs when all of the neurons in a network are of the same valence. Such a network is referred to as monosynaptic. To our knowledge, this is the first study of the effects of differing mixtures of neuron valences that has been performed in the context of network reachability.

In addition to the studies of general GNF networks, we also provide results for specialized families of GNF networks that represent ‘classical’ continuous-time network models that are of historical and on-going interest. The first model that we study is the family of shunting inhibition networks. These networks exhibit biologically realistic properties such as contrast enhancement by competitive feedback. Results are again provided for heterosynaptic, homosynaptic and monosynaptic versions of this family of networks. In addition, we analyze a family of GNF networks

³The ability to reach any state from any other state, i.e. the lack of any manifold constraints.

that has been used as the run-time model for a variety of classical connectionist models. These include the Bidirectional Associative Memory, Hopfield/Tank models, continuous-time back propagation networks, the Brain-State-in-a-Box model and McCulloch-Pitts networks.

As a result of our analysis we are able to characterize each family of networks as either holonomic or non-holonomic. The main conclusion that we are able to draw from our analysis is that practical learning algorithms do not exist for the nonholonomic families of GNF networks. We find that the weight vectors that are needed to implement non-trivial manifold constraints are located on thin manifolds of weight space. Consequently, they are not robust to noise and require perfect convergence from a learning algorithm - a requirement that is impossible to achieve in practice. We have shown that the holonomic families of GNF networks do support robust manifold constraints; however, our results indicate that the classes of constraint manifolds that can be enforced in these networks are so limited as to be of little use in most application domains.

The relevance of controllability to the manifold constraint paradigm is not our only motivation for the analysis in this chapter. Answers to questions of controllability and observability also contribute to a deeper understanding of the fundamental forces that shape and bound the behaviours of a connectionist network. In finding these answers, we have made a foundational contribution that provides a basis for further research.

The results presented in this chapter are currently being compiled into two papers [Gra98g, Gra98b] which summarize the results for open and closed GNF networks, and summarize the implications of these results for the manifold constraint paradigm and the selection of the non-adaptive terms.

4.2 The Approach to the Analysis

Before we initiate the study of the reachability analysis, to assist the reader we *briefly* reiterate the properties of neural networks in the GNF family and discuss how the reachability analysis is to be performed.

4.2.1 Grossberg Normal Form (GNF) Networks

As shown in Section 2.2, Grossberg [CG83] has stated that many interesting connectionist models are special cases of a particular type of competitive Lotka-Volterra equation, which we formally proved in Section 2.2. These models include shunting inhibition networks, bi-directional associative memories, Hopfield networks, BSB networks, and Boltzmann machines, among others. In this chapter we are interested in the reachability properties of a generalization of Grossberg's network given by an n -dimensional system of ordinary differential equations of the form

$$\begin{aligned} \dot{x}_i = a_i(x_i) \left(b_i(x_i) + \sum_{j=1}^n \left(s_{j,i}^+(x_i) H_{i,j}^- - s_{j,i}^-(x_i) H_{i,j}^+ \right) d_j(x_j) \right. \\ \left. - \sum_{k=1}^m \left(s_{(k+n),i}^+(x_i) J_{i,k}^+ - s_{(k+n),i}^-(x_i) J_{i,k}^- \right) u_k \right). \end{aligned} \quad (4.2.1)$$

4.2.2 The Relationship of Reachability to Manifold Constraints

In Appendix A we show that a run-time computation performed by a GNF network is actually an interpretation of the trajectory of the network's activation vector, $\mathbf{x}$, as it evolves from its initial value $\mathbf{x}_0$ through state space. How the trajectory is interpreted depends on the application domain. It is clear that the computation performed for a given input function $\mathbf{u}$ depends on the weight vectors that parameterize the evolution equations. In this sense the weights are analogous to a conventional 'stored program', however, it is very difficult to give a meaningful description of this program directly in terms of the weights. We need higher level computational abstractions that can be related back to the method of interpretation, but the relevant

question is: what form should these take for GNF networks?

As we shall see in Section 4.2.4, the network trajectories are constrained to lie on a special submanifold of the state space that includes $\mathbf{x}_0$ and is defined locally on a neighbourhood of $\mathbf{x}_0$. The shape of this submanifold is, in turn, governed by a differential geometric distribution spanned by vector fields from the evolution equations (4.2.1). These vector fields are parameterized by the synaptic weights. This indirect relationship between the weights and computational trajectories suggests that the submanifolds and their associated distributions are an appropriate higher level abstraction. Control theory provides various methods to constrain the shape of these submanifolds and determine the dynamics of the network trajectories on them. We will now consider a characterization of these submanifolds for a class of nonlinear systems that includes the GNF networks.

4.2.3 Absolutely Holonomic Networks

When physical laws place constraints on the instantaneous velocity of a mechanical system, the constraints are called *Pfaffian* [ML94]. We will also use this term to describe the neuron equations of a GNF network since they place constraints on the instantaneous velocity of a network's dynamics. In the literature, integrable Pfaffian constraints that restrict movement to configurations on a sub-manifold of constraint space are called *holonomic* constraints⁴ [LC93]. Constraints that are not integrable, but which also restrict movement to a sub-manifold, are called *partially holonomic* [LC93]. For convenience, we will lump both categories together under the term 'holonomic'. All other types of constraints are *nonholonomic*. Clearly nonholonomy is equivalent to complete controllability as defined in Section 3.3. Since the neuron equations of a GNF network are Pfaffian, we will also apply the terms holonomic and nonholonomic to networks that have weight assignments that produce the corresponding property in the neural equations.

⁴The word holonomic is Greek for 'all together lawful' referring to the pleasant property of integrability that these constraints possess.

Ultimately, our goal in this chapter is to determine how the weights in a GNF network shape the constraint manifolds that are induced by the reachability distribution Δ . We hope to discover that, at the very least, the property of holonomy is locally generic on the space of weights. If this is not true, the slightest deviation from the subset of holonomic weights, e.g. due to noise or finite convergence time, could make the system nonholonomic, thereby destroying the learned constraint manifold. In other words, robust learning of manifold constraints is not possible without generic holonomy on the weights. We will say that a class of networks that does not have generic holonomy on the selection of weight matrices is *effectively nonholonomic*, and that all other networks are *absolutely holonomic*. Clearly, our interest lies in classes of networks that are absolutely holonomic. In the sequel we will use the terms nonholonomic and holonomic to mean effectively nonholonomic and absolutely holonomic respectively.

4.2.4 Reachability Analysis

Consider an autonomous, nonlinear system with dynamics described by the vector evolution equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{j=1}^m \mathbf{g}_j(\mathbf{x}) u_j, \quad (4.2.2)$$

where $\mathbf{f}(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_m(\mathbf{x})$, are smooth vector fields defined over the system state space and $\mathbf{u}$ is an m -dimensional vector of external inputs.

We wish to characterize the region of state space that can be reached from an initial state $\mathbf{x}_0$ under influence of the driving input vector $\mathbf{u}$. This space can be viewed as a useful computational constraint on the set of trajectories that such a system can evolve on when influenced by the inputs.

It is well known to nonlinear control theorists [Isi89] that the reachable region for system (4.2.2), given initial value $\mathbf{x}_0$, is a submanifold with tangent space

equal to the smallest nonsingular, involutive distribution

$$\Delta_*(\mathbf{x}) = \langle \mathbf{f}(\mathbf{x}) \cdot \mathbf{g}_1(\mathbf{x}) \dots \mathbf{g}_m(\mathbf{x}) | \mathbf{G}(\mathbf{x}) \rangle$$

that contains the distribution $\mathbf{G}(\mathbf{x}) = \text{span} \{ \mathbf{g}_1(\mathbf{x}) \dots \mathbf{g}_m(\mathbf{x}) \}$, and is invariant under the vector fields $\mathbf{f}(\mathbf{x}) \cdot \mathbf{g}_1(\mathbf{x}) \dots \mathbf{g}_m(\mathbf{x})$. This distribution is uniquely defined on a local neighbourhood of $\mathbf{x}_0$, and it can be computed by the following recursion

$$\Delta_0 = \text{span} \{ \mathbf{g}_1, \dots, \mathbf{g}_m \} \quad (4.2.3)$$

$$\begin{aligned} \Delta_k &= \Delta_{k-1} + \text{span} \{ [\mathbf{f}, \tau] : \tau \in \Delta_{k-1} \} \\ &\quad + \sum_{i=1}^m \text{span} \{ [\mathbf{g}_i, \tau] : \tau \in \Delta_{k-1} \}. \end{aligned} \quad (4.2.4)$$

Δ_* locally foliates the state space into integral submanifolds on which the system state must evolve. The inputs can push the system around the submanifold that contains the initial point, but the state can never leave this manifold.

4.3 Reachability Results for Open GNF Networks

In our analysis of an ‘open’ GNF system, we shall treat the recurrent connections between neurons as external inputs. To this end, we augment the m -dimensional input vector field u with n additional feedback signals. Of course, every neuron in the network receives these signals because they actually represent the recurrent intra-network connections between neurons. An open network analysis is more general than a ‘closed’ network analysis, because all $n + m$ input signals are allowed to vary freely. This is in contrast to the closed network analysis in which the outputs of the network neurons are fed back to each neuron through the first n inputs. The result is that the first n input signals are constrained to be the special functions $u_j = d_j(x_j)$, $1 \leq j \leq n$. Restricting the set of legal input functions this way means that the reachability manifold for a closed network must be a subset of the open system reachability manifold.

In many applications the open network results are all that is required, as they describe state space bounds which a closed system is guaranteed not to violate. Open network results also have the advantage of being easier to state and work with than their closed network counterparts, and, in practice, they may be of more importance because of this. Finally, as we shall see in Section 4.5, it is much easier to analyze closed GNF networks if we first perform an open network analysis.

Before we begin, we must find a form for our GNF networks that is better suited to analysis. Observe that the general GNF network equation (4.2.1) is a special case of the nonlinear system equation (4.2.2). The following definition shows this correspondence.

Definition 4.1

We define the open GNF network vector fields $\mathbf{g}_0(\mathbf{x}), \mathbf{g}_1(\mathbf{x}), \dots, \mathbf{g}_{n+m}(\mathbf{x})$ as follows

$$\begin{aligned} g_{0,i} &= f_i(\mathbf{x}) \\ &= a_i(x_i) b_i(x_i), \end{aligned} \tag{4.3.1}$$

for $1 \leq i \leq n$.

$$g_{j,i} = a_i(x_i) \left(s_{j,i}^+(x_i) H_{i,j}^- - s_{j,i}^-(x_i) H_{i,j}^+ \right), \tag{4.3.2}$$

for $1 \leq i, j \leq n$.

$$g_{n+k,i} = a_i(x_i) \left(s_{(n+k),i}^+(x_i) J_{i,k}^- - s_{(n+k),i}^-(x_i) J_{i,k}^+ \right), \tag{4.3.3}$$

for $1 \leq i \leq n$, and $1 \leq k \leq m$.

Equations (4.3.2) and (4.3.3) have similar forms because they represent the modulation of neural inputs u_1 to $u_{(n+m)}$ by synaptic connections. They differ from equation (4.3.1), however, because the 0^{th} field represents the inherent decay of the neuron activations to their rest state — an intrinsic property of neurons that does not depend on parameterization by synaptic weights. Monitoring one *special* GNF network field considerably complicates the analysis. Therefore, we shall introduce the following notation that permits us to treat all GNF network vector fields in an egalitarian manner.

Definition 4.2

The open ‘faux’ neuron with index position 0 is a notational device which provides a constant output signal 0 to every neuron in a GNF network. We consider the activation decay terms $b_i(x_i)$, $1 \leq i \leq n$, to be synaptic gains on inputs from the faux neuron. We will consider this neuron to be homosynaptic with excitatory connection weights. The following notation will be used unless otherwise stated — for $1 \leq i \leq n$

$$\begin{aligned} s_{0,i}^-(x_i) &= b_i(x_i), \quad s_{0,i}^+(x_i) = 0, \\ H_{i,0}^+ &= 1, \quad H_{i,0}^- = 0, \\ g_{0,i} &= a_i(x_i) \left(s_{0,i}^+(x_i) H_{i,0}^+ - s_{0,i}^-(x_i) H_{i,0}^- \right). \end{aligned} \quad (4.3.4)$$

4.3.1 Supporting Definitions and Lemmas

In this section, we introduce definitions and lemmas that characterize different aspects of the vector fields that span the reachability distribution for GNF networks. These provide the framework for proving our main results, which are stated in Section 4.3.2 and Section 4.4.

Our basic approach is to apply the recursive algorithm (4.2.4) to the GNF network vector fields. At each step k this recurrence produces a set Δ_k spanned by vector fields based on the $n + m + 1$ fields $\mathbf{g}_j(\mathbf{y})$, $0 \leq j \leq n + m$, determined by the system of n neuron equations. The local foliation of the state space into controllability spaces is determined by the final distribution $\Delta_* = \Delta_k = \Delta_{k_*+j}$, $j > 0$. Our ultimate goal is to describe the spanning set of Δ_* with respect to the synaptic weights, clarifying the contribution of the nonadaptive components of the neuron equations as well. To that end we now introduce the following definitions and lemmas.

In the sequel we will need to index repeated Lie derivatives and multi-Lie brackets with combinations of integer indices. The following definitions provide notation for this task.

Definition 4.3

A multi-index, π , of length⁵ $|\pi| \geq 1$, is an ordered sequence of integers denoted $\pi_1 / \cdots / \pi_{|\pi|}$.

Definition 4.4

π^i , is the position set for index i in multi-index π . It is the set of positions that have index value i , i.e.

$$\pi^i = \{j : \pi_j = i\}.$$

The inverse position set for index i consists of those positions in π that do not have value i .

$$\overline{\pi^i} = \{j : \pi_j \neq i\}.$$

Definition 4.5

Given a multi-index π containing indices for a set of $|\pi|$ homosynaptic neurons, we define, q , the homosynaptic valence pattern for π to be a multi-index of length $|\pi|$, such that for $1 \leq j \leq |\pi|$

$$q_j = \begin{cases} 0 & \text{if neuron } \pi_j \text{ is excitatory} \\ 1 & \text{if neuron } \pi_j \text{ is inhibitory.} \end{cases}$$

Definition 4.6

$\Omega_k^{(p,q)}$, the complete multi-index set, is the set of $(q-p+1)^k$ multi-indices π of length $\leq k$ that can be made from combinations, with replacement, of the indices in the integer interval $[p, q]$.

Definition 4.7

$\Pi_k^{(p,q)}$, the k^{th} restricted multi-index set, is the set of multi-indices of length k that can be made from the $\binom{q-p+1}{k}$ unique combinations (with replacement) of indices in the integer interval $[p, q]$. The internal ordering of indices within each $\pi \in \Pi_k^{(p,q)}$ is given by $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_k$. Clearly, $\Pi_k^{(p,q)} \subseteq \Omega_k^{(p,q)}$.

⁵ We will use the operator $|\xi|$ to return the size of its argument ξ . If ξ is a set, then its cardinality is returned. If ξ is a multi-index, then its length is returned. The meaning of this operator will always be clear from context.

Definition 4.8

$\hat{\Pi}_k^{(p,q)}$, the multi-index set on indices $\{p, p+1, \dots, q-1, q\}$, is the set of $\sum_{j=1}^k \binom{q-p+j}{j}$ multi-indices of length $\leq k$.

$$\bigcup_{j=1}^k \Pi_j^{(p,q)}.$$

Next, we continue to introduce notation that allows us to hide differences between the GNF network vector fields. The following definitions introduce vector fields that distinguish between the nonadaptive and adaptive sources of synaptic modulation of input signals. They describe the mechanisms that determine the effect that an input signal has on the reachability properties of the network dynamics. The notation allows us to treat modulation of all input signals in a uniform manner.

Definition 4.9

The synaptic gain on the j^{th} input signal is a vector field that captures the total nonadaptive amplification of the signal by each neuron in the network. The excitatory synaptic gain on the j^{th} signal is given by

$$\gamma_j^0(\mathbf{x}) = \begin{bmatrix} a_1(x_1) s_{j,1}^+(x_1) \\ \vdots \\ a_n(x_n) s_{j,n}^+(x_n) \end{bmatrix}$$

and the inhibitory synaptic gain is given by

$$\gamma_j^1(\mathbf{x}) = \begin{bmatrix} a_1(x_1) s_{j,1}^-(x_1) \\ \vdots \\ a_n(x_n) s_{j,n}^-(x_n) \end{bmatrix}$$

Definition 4.10

The open network weight vector that represents the adaptive weighting of the j^{th}

input signal at synaptic connections of valence s is given by

$$\begin{aligned} \mathbf{W}_j^s &= \begin{cases} \mathbf{H}_0^+ & s = 0 \\ \mathbf{H}_0^- & s = 1. \end{cases} \\ \mathbf{W}_j^s &= \begin{cases} \mathbf{H}_j^+ & s = 0 \\ \mathbf{H}_j^- & s = 1. \end{cases} & 1 \leq j \leq n. \\ \mathbf{W}_j^s &= \begin{cases} \mathbf{J}_j^+ & s = 0 \\ \mathbf{J}_j^- & s = 1. \end{cases} & n+1 \leq j \leq n+m. \end{aligned}$$

where $s = 0$ represents excitatory connections, and $s = 1$ represents inhibitory connections⁶.

Having determined a formulation and notation for our GNF networks that is amenable to analysis, we may now begin the task of deriving the reachability distribution for variations on this class of network. In our first lemma, we establish that the reachability distribution for a GNF network is spanned by a particular set of multi-Lie brackets of the network vector fields $\{\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n+m}\}$. In the interest of notational simplicity, we will regard a single network vector field as a multi-Lie bracket of length one.

Lemma 4.1

The d -dimensional reachability distribution Δ_ of an open GNF network with n neurons and m inputs is of the form*

$$\begin{aligned} \Delta_* &= \text{span} \left\{ \mu_\pi : \pi \in \hat{\Pi}_d^{(0, n+m)} \right\}, \quad \text{where} \\ \mu_\pi &= \begin{cases} \mathbf{g}_{\pi_1} & \text{if } |\pi| = 1 \\ \left[\mathbf{g}_{\pi_1}, \left[\mathbf{g}_{\pi_2}, \dots, \left[\mathbf{g}_{\pi_{|\pi|-1}}, \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] & \text{if } |\pi| > 1. \end{cases} \end{aligned}$$

⁶In the interest of simplicity, we request of the reader permission to marginally abuse the notation. Rather than using a multi-index position to index a weight vector, e.g. $\mathbf{W}_{\pi_j}^s$ or $\mathbf{W}_{p_j}^s$, we will use the position index instead, as in $\mathbf{W}_j^s$. The intended multi-index will always be clear from context.

Proof: Let Δ_{k-1} and Δ_k be nonsingular distributions in the sequence generated by recursion (4.2.4). In addition, let

$$\Delta_{k-1} = \text{span} \left\{ \theta_1^{k-1} \dots \theta_{z_{k-1}}^{k-1} \right\}. \quad (4.3.5)$$

First, we establish that the spanning set of Δ_k has the general form

$$\begin{aligned} \Delta_k = \left\{ \theta_1^{k-1} \dots \theta_{z_{k-1}}^{k-1} \cdot \left[\mathbf{g}_0 \cdot \theta_1^{k-1} \right] \dots \left[\mathbf{g}_0 \cdot \theta_{z_{k-1}}^{k-1} \right] \dots \right. \\ \left. \dots \left[\mathbf{g}_{n+m} \cdot \theta_1^{k-1} \right] \dots \left[\mathbf{g}_{n+m} \cdot \theta_{z_{k-1}}^{k-1} \right] \right\}. \end{aligned} \quad (4.3.6)$$

From the form of the recurrence (4.2.4), it is immediate that Δ_{k-1} is invariant under the vector fields $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n+m}$, and therefore

$$\left[\mathbf{g}_i \cdot \theta_j^{k-1} \right] \in \Delta_{k-1}, \quad 0 \leq i \leq n+m, \quad 1 \leq j \leq z_{k-1}. \quad (4.3.7)$$

Now, observing that any vector field $\tau \in \Delta_{k-1}$ can be written in the form $\tau(\mathbf{y}) = \sum_{j=1}^{z_{k-1}} c_j(y) \theta_j^{k-1}$, where $c_j \in C^\infty$, we have:

$$[\mathbf{g}_i, \tau] = \left[\mathbf{g}_i \cdot \sum_{j=1}^{z_{k-1}} c_j(y) \theta_j^{k-1} \right] = \sum_{j=1}^{z_{k-1}} c_j \left[\mathbf{g}_i \cdot \theta_j^{k-1} \right] + \sum_{j=1}^{z_{k-1}} (\mathcal{L}_{\mathbf{g}_i} c_j) \theta_j^{k-1} \quad (4.3.8)$$

for $0 \leq i \leq n+m$. Since c_j and $(\mathcal{L}_{\mathbf{g}_i} c_j)$ are scalar values, we know from (4.3.5) and (4.3.7) that the terms of (4.3.8) are elements of Δ_{k-1} , and therefore $[\mathbf{g}_i, \tau] \in \Delta_{k-1}$ for all $\tau \in \Delta_{k-1}$. Consequently, for $0 \leq i \leq n+m$,

$$\text{span} \left\{ \left[\mathbf{g}_i \cdot \theta_1^{k-1} \right] \dots \left[\mathbf{g}_i \cdot \theta_{z_{k-1}}^{k-1} \right] \right\} \subseteq \text{span} \{ [\mathbf{g}_i, \tau] : \tau \in \Delta_{k-1} \}. \quad (4.3.9)$$

If we add Δ_{k-1} to both sides of (4.3.9) we obtain

$$\begin{aligned} \Delta_{k-1} + \text{span} \{ [\mathbf{g}_i, \tau] : \tau \in \Delta_{k-1} \} \\ = \text{span} \left\{ \theta_1^{k-1} \dots \theta_{z_{k-1}}^{k-1} \cdot \left[\mathbf{g}_i \cdot \theta_1^{k-1} \right] \dots \left[\mathbf{g}_i \cdot \theta_{z_{k-1}}^{k-1} \right] \right\}. \end{aligned} \quad (4.3.10)$$

From (4.3.10) we see that, at step k the recurrence (4.2.4) will generate the distribution

$$\begin{aligned}\Delta_k &= \Delta_{k-1} + \sum_{i=0}^{n+m} [\mathbf{g}_i, \Delta_{k-1}] \\ &= \text{span} \left\{ \theta_1^{k-1}, \dots, \theta_{z_{k-1}}^{k-1}, [\mathbf{g}_0, \theta_1^{k-1}], \dots, [\mathbf{g}_0, \theta_{z_{k-1}}^{k-1}], \dots \right. \\ &\quad \left. \dots [\mathbf{g}_{n+m}, \theta_1^{k-1}], \dots, [\mathbf{g}_{n+m}, \theta_{z_{k-1}}^{k-1}] \right\}.\end{aligned}$$

ane (4.3.6) is proven.

Next, we prove Lemma 4.1 by showing that equation (4.3.6), the general form of the spanning vectors of Δ_* , implies that $\Delta_* = \text{span} \left\{ \mu_\pi : \pi \in \Omega_d^{(0, n+m)} \right\}$. We begin by observing that a multi-index $\pi_a / \dots / \pi_b / \pi_c$ can be formed by preserving the order of subscripts that appear in a multi-Lie bracket $[\tau_a, \dots [\tau_b, \tau_c]]$. The proof proceeds by induction on k . At $k = 1$, Δ_0 is spanned by $\theta_j^0 = \mathbf{g}_j$ for $0 \leq j \leq n + m$. It is immediate that the multi-indices corresponding to these multi-Lie brackets form the set of all length one multi-indices $\Omega_1^{(0, n+m)} = \{0, 1, \dots, n + m\}$. For the inductive step, we assume that for $k - 1$, the multi-Lie brackets θ_j^{k-1} , $1 \leq j \leq z_{k-1}$, for some integer z_{k-1} , span Δ_{k-1} . We assume that the multi-index set for these multi-Lie brackets is $\Omega_k^{(0, n+m)}$. Equation (4.3.11) adds multi-Lie brackets of length $k + 1$ to the spanning set of Δ_{k-1} , by taking the Lie bracket of every network vector field with all the multi-Lie brackets spanning Δ_{k-1} . Every multi-index in the spanning set of Δ_{k-1} appears in Δ_k with every index from 0 to $n + m$ prepended. Clearly then, the corresponding multi-index set is $\Omega_{k+1}^{(0, n+m)}$. As we have seen in Section 3.3.1, the recursion stops at $k = k_*$. For a d -dimensional Δ_* , d represents an upper bound on the value of k_* because we know that $\Delta_* = \Delta_k$ for all $k \geq k_*$. By induction, we therefore have Δ_* spanned by multi-Lie brackets with corresponding multi-index set $\Omega_d^{(0, n+m)}$ as claimed in the lemma. ■

Having determined the general form of the vector fields that span the reachability distribution of a GNF network, we now introduce two technical lemmas that will prove useful in establishing the spanning set in terms of a summation of Lie

derivatives.

Lemma 4.2

A multi-Lie bracket may be written as the difference of Lie derivatives

$$[\tau_d, [\tau_{d-1}, \dots, [\tau_2, \tau_1]]] = \mathcal{L}_{\tau_d} [\tau_{d-1}, \dots, [\tau_2, \tau_1]] - \mathcal{L}_{[\tau_{d-1}, \dots, [\tau_2, \tau_1]]} \tau_d$$

Proof: Letting $\psi = [\tau_{d-1}, \dots, [\tau_2, \tau_1]]$, we have.

$$\begin{aligned} [\tau_d, [\tau_{d-1}, \dots, [\tau_2, \tau_1]]] &= [\tau_d, \psi] \\ &= \frac{\partial \psi}{\partial \mathbf{y}} \tau - \frac{\partial \tau}{\partial \mathbf{y}} \psi \\ &= \langle \partial \psi, \tau \rangle - \langle \partial \tau, \psi \rangle \\ &= \mathcal{L}_\tau \psi - \mathcal{L}_\psi \tau \\ &= \mathcal{L}_{\tau_d} [\tau_{d-1}, \dots, [\tau_2, \tau_1]] - \mathcal{L}_{[\tau_{d-1}, \dots, [\tau_2, \tau_1]]} \tau_d. \end{aligned}$$

as claimed. ■

Lemma 4.3

A Lie bracket $\mathcal{L}_{\gamma} \tau_d$, with respect to a multi-Lie bracket $\gamma = [\tau_{d-1}, \dots, [\tau_2, \tau_1]]$, may be written as a difference of Lie derivatives

$$\mathcal{L}_{[\tau_{d-1}, \dots, [\tau_2, \tau_1]]} \tau_d = \mathcal{L}_{\tau_{d-1}} \mathcal{L}_{[\tau_{d-2}, \dots, [\tau_2, \tau_1]]} \tau_d - \mathcal{L}_{[\tau_{d-2}, \dots, [\tau_2, \tau_1]]} \mathcal{L}_{\tau_{d-1}} \tau_d.$$

Proof: Letting $\psi = [\tau_{d-2}, \dots, [\tau_2, \tau_1]]$, we have.

$$\begin{aligned} \mathcal{L}_{[\tau_{d-1}, \psi]} \tau_d &= \langle \partial \tau_d, [\tau_{d-1}, \psi] \rangle \\ &= \left\langle \frac{\partial \tau_d}{\partial \mathbf{y}}, \partial \psi \tau_{d-1} - \partial \tau_{d-1} \psi \right\rangle \\ &= \left\langle \frac{\partial \tau_d}{\partial \mathbf{y}}, \partial \psi \tau_{d-1} \right\rangle - \left\langle \frac{\partial \tau_d}{\partial \mathbf{y}}, \partial \tau_{d-1} \psi \right\rangle \\ &= \sum_{k=1}^n \frac{\partial \tau_d}{\partial y_k} \sum_{l=1}^n \tau_{d-1,l} \frac{\partial \psi_k}{\partial y_l} - \sum_{k=1}^n \frac{\partial \tau_d}{\partial y_k} \sum_{l=1}^n \psi_l \frac{\partial \tau_{d-1,k}}{\partial y_l}. \end{aligned} \tag{4.3.11}$$

By adding

$$\sum_{l=1}^n \sum_{k=1}^n \tau_{d-1,l} \frac{\partial^2 \tau_d}{\partial y_k \partial y_l} \psi_k - \sum_{l=1}^n \sum_{k=1}^n \psi_l \frac{\partial^2 \tau}{\partial y_k \partial y_l} \tau_{d-1,k} = 0$$

to equation (4.3.11) we obtain

$$\begin{aligned} \mathcal{L}_{[\tau_{d-1}, \psi]} \tau_d &= \sum_{l=1}^n \tau_{d-1,l} \sum_{k=1}^n \left(\frac{\partial}{\partial y_l} \frac{\partial \tau_d}{\partial y_k} \psi_k + \frac{\partial \tau_d}{\partial y_k} \frac{\partial \psi_k}{\partial y_l} \right) \\ &\quad - \sum_{l=1}^n \psi_l \sum_{k=1}^n \left(\frac{\partial}{\partial y_l} \frac{\partial \tau_d}{\partial y_k} \tau_{d-1,k} + \frac{\partial \tau_d}{\partial y_k} \frac{\partial \tau_{d-1,k}}{\partial y_l} \right) \\ &= \left\langle \partial \left(\sum_{k=1}^n \frac{\partial \tau}{\partial y_k} \psi_k \right), \tau_{d-1} \right\rangle - \left\langle \partial \left(\sum_{k=1}^n \frac{\partial \tau}{\partial y_k} \tau_{d-1,k} \right), \psi \right\rangle \\ &= \langle \partial \langle \partial \tau_d, \psi \rangle, \tau_{d-1} \rangle - \langle \partial \langle \partial \tau_d, \tau_{d-1} \rangle, \psi \rangle \\ &= \mathcal{L}_{\tau_{d-1}} \mathcal{L}_{\psi} \tau_d - \mathcal{L}_{\psi} \mathcal{L}_{\tau_{d-1}} \tau_d \end{aligned} \tag{4.3.12}$$

which, after substitution of ψ , yields the desired result. ■

We now introduce a lemma that allows us to re-express the spanning multi-Lie brackets in terms of a recurrence of repeating Lie derivatives. The recurrence allows us to visualize the multi-Lie brackets as a sum of the leaves of a binary tree. This will prove useful in clarifying the relationship between the network weight vectors and Δ_* .

Lemma 4.4

Each multi-Lie bracket $\left[\tau_{\pi_1}, \left[\tau_{\pi_2}, \dots, \left[\tau_{\pi_{|\pi|-1}}, \tau_{\pi_{|\pi|}} \right] \right] \right]$, $\pi \in \Omega_d^{(0, n+m)}$ is the sum of repeating Lie derivatives of the form

$$\mu_\pi = \sum_{p \in \Phi(d)} \sigma(p) \mathcal{L}_{\tau_{p_1}} \mathcal{L}_{\tau_{p_2}} \cdots \mathcal{L}_{\tau_{p_{|\pi|-1}}} \tau_{p_{|\pi|}},$$

where $\tau_i \in \{\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n+m}\}$, and $p = p_1/p_2/\dots/p_{|\pi|}$ is a member of the set $\Phi(|\pi|)$

defined recursively as⁷

$$\Phi(i) = \{\pi_i / \Phi(i-1)\} \cup \{\Phi(i-1) / \pi_i\}. \quad (4.3.13)$$

$$\Phi(1) = \{\pi_1\}. \quad (4.3.14)$$

Proof: Observe that Lemma 4.2 allows us to decompose a length $|p|$ multi-Lie bracket into the sum of two terms which both contain the same multi-Lie bracket of length $|p| - 1$. Lemma 4.2 can be reapplied to the left term, and Lemma 4.3 to the right, to further decompose the sum. Application of the lemmas yields terms in which the multi-Lie brackets are one vector field shorter. In this way, the two lemmas can be recursively applied to the sum to expand all of its terms until they are all of length one.

Figure 4.1 illustrates the recurrence as a binary tree. The nodes within level i ($1 \leq i \leq d$) of the tree represent terms that sum to the multi-Lie bracket μ_π at the root of the tree. Note that a multi-index $p_a / \cdots / p_b / p_c$ can be formed from the ordered subscripts that appear in a repeated Lie derivative $\mathcal{L}_{\tau_a} \cdots \mathcal{L}_{\tau_b} \tau_c$. If, in place of a subscripted vector field τ_i , the Lie derivative contains a Lie bracket $[\mathbf{a}, \mathbf{b}]$, we may simply place the entire bracket into the multi-index (e.g. $p_{|p|} / \cdots / p_3 / [\mathbf{a}, \mathbf{b}] / p_1$, if $i = 2$). In this way, we may represent each node generated by the recurrence with its corresponding multi-index (as shown in the figure). Each node contains a multi-Lie bracket of length (tree-depth - node-level + 1), because the decompositions in both lemmas always produce terms that have multi-Lie brackets that are one vector field smaller than the original term. This fact also implies that the height of the tree is equal to the length d of the root multi-Lie bracket.

⁷In the recursion $\Phi(n)$, we assume that, given an integer w and set z ,

$$w/z = \bigcup_{e \in z} \{w/e\}, \quad \text{and}$$

$$z/w = \bigcup_{e \in z} \{e/w\}.$$

where multi-index

$$i_1 / \cdots / i_{j-1} / [a, b] / i_{j+1} / \cdots / i_d$$

corresponds to repeated Lie derivative

$$\mathcal{L}_{\tau_{i_1}} \cdots \mathcal{L}_{\tau_{i_{j-1}}} \mathcal{L}_{[a, b]} \mathcal{L}_{\tau_{i_{j+1}}} \cdots \mathcal{L}_{\tau_{i_{d-1}}} \tau_d$$

Figure 4.1: Length d Open GNF Network Multi-Lie bracket Decomposition Tree.

Referring to Lemma 4.2, we see that a multi-Lie bracket with multi-index $\cdots / [\tau_i, [\tau_{i-1}, \mathbf{b}]]$ produces a left term with multi-index $\cdots / i / [\tau_{i-1}, \mathbf{b}]$. Similarly, Lemma 4.3 decomposes the Lie derivative with multi-index $\cdots / [\tau_i, [\tau_{i-1}, \mathbf{b}]] / \cdots$ into a left term with multi-index $\cdots / i / [\tau_{i-1}, \mathbf{b}] / \cdots$. Consequently, a left branch in the tree always inserts an index equal to the parent term's level number immediately to the left of the multi-Lie bracket in the child multi-index, and a right branch inserts the level number immediately to the right of the inner multi-Lie bracket. The inner multi-Lie bracket has a multi-index π with index π_1 one less than the index of the bracket in the parent term. From these considerations, we define the recursive set $\Phi(n)$ that mimics the construction of the leaf node multi-indices, where n represents the number of levels in the tree.

With regard to the sign of the node terms, it is clear from Lemmas 4.2 and 4.3 that a left branch always preserves the sign of the parent term, while a right branch term always inverts the parent term's sign. Therefore, the sign of a node is positive if the number of right branches leading to that node is even, and negative otherwise. Figure 4.1 records left branches with a $+$ sign, and right branches with a $-$ sign. We represent the sign of a node with multi-index p as $\sigma(p)$.

From the repeated application of Lemmas 4.2 and 4.3, we see that the recursion terminates at leaf nodes of the form $\sigma(p) \mathcal{L}_{\tau_1} \mathcal{L}_{\tau_2} \cdots \mathcal{L}_{\tau_k} \tau_l$, where $p = i/j/\cdots/k$ is one of the 2^{d-1} valid permutations of $\pi_d/(\pi_{d-1})/\cdots/\pi_1$ that occur at the leaves, and $\sigma(p)$ is the sign of the leaf term. The multi-Lie bracket at the root is equal to the sum of the level d leaf terms, and so we have

$$[\tau_{\pi_d}, [\tau_{\pi_{d-1}}, \cdots [\tau_{\pi_2}, \tau_{\pi_1}]]] = \sum_{p \in \Phi(d)} \sigma(p) \mathcal{L}_{\tau_{p_1}} \mathcal{L}_{\tau_{p_2}} \cdots \mathcal{L}_{\tau_{p_{d-1}}} \tau_{p_d},$$

which, after generalizing the length d to $|p| = |\pi|$, proves the lemma. ■

The next definition and lemma tell us that we may express a GNF multi-Lie bracket leaf as a sum of Hadamard products derived from repeated differentiation of the network vector fields. We examine a heterosynaptic network.

Definition 4.11

The j^{th} GNF network derivative field of order r is derived from the j^{th} GNF network vector field by taking the r^{th} partial derivative of each of its $1 \leq i \leq n$ elementary functions with respect to the i^{th} coordinate. i.e.

$$\mathbf{g}_i^{(r)} = \begin{bmatrix} \frac{\partial^r g_{i,1}}{\partial y_1^r} \\ \vdots \\ \frac{\partial^r g_{i,n}}{\partial y_n^r} \end{bmatrix}.$$

Lemma 4.5

Given a heterosynaptic GNF network of n neurons with m inputs, a repeating Lie derivative of network vector fields $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n-m}$, with multi-index p , may be expressed as the sum

$$\mathcal{L}_{\mathbf{g}_{p|p|}} \mathcal{L}_{\mathbf{g}_{p|p|-1}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} = \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|p|-1}=1}^{|p|-1} \bigodot_{j=1}^{|p|} \mathbf{g}_{p_j}^{(\omega(p,i,j))}, \quad (4.3.15)$$

where⁸

$$i = i_1 / i_2 / \cdots / i_{|p|-1},$$

$$\omega(p, i, j) = \begin{cases} 0 & j = |p| \\ \sum_{k=j}^{|p|-1} \delta_j(i_k) & j < |p|. \end{cases}$$

Proof: Expanding the repeating Lie derivative as inner products we have

$$\begin{aligned} \mathcal{L}_{\mathbf{g}_{p|p|}} \mathcal{L}_{\mathbf{g}_{p|p|-1}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} &= \left\langle \partial \mathcal{L}_{\mathbf{g}_{p|p|-1}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} \cdot \mathbf{g}_{p|p|} \right\rangle \\ &= \left\langle \partial \left\langle \partial \mathcal{L}_{\mathbf{g}_{p|p|-2}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} \cdot \mathbf{g}_{p|p|-1} \right\rangle \cdot \mathbf{g}_{p|p|} \right\rangle \\ &\vdots \\ &= \left\langle \partial \left\langle \cdots \partial \left\langle \partial \mathbf{g}_{p_1} \cdot \mathbf{g}_{p_2} \right\rangle \cdots \cdot \mathbf{g}_{p|p|-1} \right\rangle \cdot \mathbf{g}_{p|p|} \right\rangle. \end{aligned}$$

⁸ δ is the Dirac delta function, $\delta_a(b) = \begin{cases} 0 & \text{if } a \neq b \\ 1 & \text{if } a = b. \end{cases}$

Observe that, because the j^{th} element of $\mathbf{g}_i(\mathbf{y})$ is a function of the j^{th} coordinate $g_{i,j}(y_j)$ for all $1 \leq j \leq n$, the Jacobian $\frac{\partial \mathbf{g}_i(\mathbf{y})}{\partial \mathbf{y}}$ is a diagonal matrix. Therefore, the Lie derivative

$$\begin{aligned}
 \mathcal{L}_{\mathbf{g}_{p_2}(\mathbf{y})} \mathbf{g}_{p_1}(\mathbf{y}) &= \langle \partial \mathbf{g}_{p_1}(\mathbf{y}) \cdot \mathbf{g}_{p_2}(\mathbf{y}) \rangle \\
 &= \begin{bmatrix} \frac{\partial g_{p_1,1}(y_1)}{\partial y_1} & & 0 \\ & \ddots & \\ 0 & & \frac{\partial g_{p_1,n}(y_n)}{\partial y_n} \end{bmatrix} \mathbf{g}_{p_2}(\mathbf{y}) \\
 &= \begin{bmatrix} \frac{\partial g_{p_1,1}(y_1)}{\partial y_1} g_{p_2,1}(y_1) \\ \vdots \\ \frac{\partial g_{p_1,n}(y_n)}{\partial y_n} g_{p_2,n}(y_n) \end{bmatrix}
 \end{aligned} \tag{4.3.16}$$

is again a vector field with j^{th} element a function of the j^{th} coordinate, and diagonal Jacobian. Expressing the next repeated Lie derivative, and making use of (4.3.16), we have

$$\begin{aligned}
 \mathcal{L}_{\mathbf{g}_{p_3}(\mathbf{y})} \mathcal{L}_{\mathbf{g}_{p_2}(\mathbf{y})} \mathbf{g}_{p_1}(\mathbf{y}) &= \langle \partial \langle \partial \mathbf{g}_{p_1}(\mathbf{y}) \cdot \mathbf{g}_{p_2}(\mathbf{y}) \rangle \cdot \mathbf{g}_{p_3}(\mathbf{y}) \rangle \\
 &= \begin{bmatrix} \frac{\partial}{\partial y_1} \left(\frac{\partial g_{p_1,1}(y_1)}{\partial y_1} g_{p_2,1}(y_1) \right) \\ \vdots \\ \frac{\partial}{\partial y_n} \left(\frac{\partial g_{p_1,n}(y_n)}{\partial y_n} g_{p_2,n}(y_n) \right) \end{bmatrix}
 \end{aligned}$$

and so on recursively, so that, after $|p| - 1$ such contractions, we have

$$\begin{aligned}
 &\mathcal{L}_{\mathbf{g}_{p_{|p|}}} \mathcal{L}_{\mathbf{g}_{p_{|p|-1}}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} \\
 &= \langle \partial \langle \cdots \partial \langle \partial \mathbf{g}_{p_1} \cdot \mathbf{g}_{p_2} \rangle \cdots \cdot \mathbf{g}_{p_{|p|-1}} \rangle \cdot \mathbf{g}_{p_{|p|}} \rangle \\
 &= \begin{bmatrix} \left(\frac{\partial}{\partial y_1} \left(\cdots \frac{\partial}{\partial y_1} \left(\frac{\partial g_{p_1,1}(y_1)}{\partial y_1} g_{p_2,1}(y_1) \right) g_{p_3,1}(y_1) \cdots \right) g_{p_{|p|},1}(y_1) \right) \\ \vdots \\ \left(\frac{\partial}{\partial y_n} \left(\cdots \frac{\partial}{\partial y_n} \left(\frac{\partial g_{p_1,n}(y_n)}{\partial y_n} g_{p_2,n}(y_n) \right) g_{p_3,n}(y_n) \cdots \right) g_{p_{|p|},n}(y_n) \right) \end{bmatrix}
 \end{aligned} \tag{4.3.17}$$

Now, expanding this vector field from the inner most set of parentheses outward by applying the usual Leibnitz' rule of differentiation to the parenthesized

Figure 4.2: Multi-Lie Bracket Leaf Derivation Tree for $\mathcal{L}_{g_{p_4}} \mathcal{L}_{g_{p_3}} \mathcal{L}_{g_{p_2}} g_{p_1}$

expression, we can produce an evaluation tree (illustrated for the case $|p| = 4$ in Figure 4.2). Each level of the tree represents a sum of terms (the nodes in that level) that is equal to the root term. The first level is a trivial sum containing only the root term itself. The children of a given node represent the terms in the sum that results from application of the Leibnitz rule to the innermost derivative in the parent term. For example, in Figure 4.2 the innermost derivative of the root term in level one is $\frac{\partial}{\partial y_i} \left(\frac{\partial g_{p_1,i}}{\partial y_i} g_{p_2,i} \right)$. Expanding this derivative, the root node expands to the sum $\frac{\partial}{\partial y_i} \left(\frac{\partial^2 g_{p_1,i}}{\partial y_i^2} g_{p_2,i} g_{p_3,i} \right) g_{p_4,i} + \frac{\partial}{\partial y_i} \left(\frac{\partial g_{p_1,i}}{\partial y_i} \frac{\partial g_{p_2,i}}{\partial y_i} g_{p_3,i} \right) g_{p_4,i}$, and these terms occur as nodes in the sum at level two.

By induction on tree level i , we see that the innermost derivative of a term at level i is a product of $i + 1$ elementary functions. This is clearly true at level 1, proving the base case. For the inductive step, we make the assumption that it is true at level $i - 1$, and observe from the form of (4.3.17) that each derivative is multiplied by a function. Using the inductive assumption, expanding a level $i - 1$ term produces a sum of i terms, each of which is then multiplied by the function $g_{p_{i+1}}$. Therefore, the terms at level i are products of $i + 1$ functions as claimed. The induction terminates at the level $|p| - 1$ leaf terms where only simple derivatives remain.

Furthermore, if the root term contains $|p|$ GNF vector fields, then the expansion tree must have $(|p| - 1)!$ leaf terms. This is so because every node at level $1 \leq i < (|p| - 1)$ produces $i + 1$ children ($i < (|p| - 1)$ because there are $|p| - 2$ derivatives of products in the root term given by (4.3.17)).

Notice that, from the induction above, we know that the terms in the leaf summation are products of elementary functions of the GNF vector fields

$$g_{p_1,i}^{(r_1)} g_{p_2,i}^{(r_2)} \cdots g_{p_{|p|},i}^{(r_d)} = \prod_{j=1}^{|p|} g_{p_j,i}^{(r_j)}.$$

We can produce a closed form for the sum of leaf nodes as follows. We introduce the variable $i_2 \in [1, 2]$ to count the two branches beneath the nodes in level one. When

$i_2 = 1$, the degree of differentiation r_1 of $g_{p_1,i}$ must be increased by one to account for the derivative to the left branch, and if $i_2 = 2$ the degree of differentiation r_2 of $g_{p_2,i}$ must be increased by one to account for the right branch. Now, for each value of i_2 , we want an additional variable $i_3 \in [1, 3]$ to account for the three branches below the nodes in level two. When $i_3 = j$, the degree of differentiation r_j of $g_{p_j,i}$ must be increased by one to account for the derivative on the j^{th} function on the j^{th} branch under a node in level two. For a root term with $|p|$ functions, we will require $|p| - 2$ such variables for the $|p| - 2$ nonleaf levels of the tree, and an additional variable $i_1 = 1$ to account for the fact that there is already a derivative of $g_{p_1,i}$ in the root term. The sum of the $(|p| - 1)!$ leaf terms is generated by the repeated sum over these variables

$$\sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|p|-1}=1}^{|p|-1} \prod_{j=1}^{|p|} g_{p_j,i}^{(r_j)}. \quad (4.3.18)$$

As we saw above, the function $g_{p;p_i}$ always has degree of differentiation 0 in every level $|p| - 1$ leaf term. For $j < |p|$, the j^{th} function $g_{p_j,i}^{(r_j)}$ in a leaf term has degree of differentiation r_j that depends on the variable assignments for that term. Clearly, we need to sum the number of variables that have values equal to the index j . Observing that only variables i_k with $k \geq j$ can be assigned the value j , we have

$$r_j = \sum_{k=j}^{|p|-1} \delta_j(i_k), \quad j < |p|. \quad (4.3.19)$$

Combining (4.3.18) and (4.3.19), and recognizing that the indices of the vector fields are given by the multi-index $p = p_1 / \cdots / p_{|p|}$, in vector field notation we have

$$\mathcal{L}_{g_{p_{|p|}}} \mathcal{L}_{g_{p_{|p|-1}}} \cdots \mathcal{L}_{g_{p_2}} g_{p_1} = \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|p|-1}=1}^{|p|-1} \bigodot_{j=1}^{|p|-1} g_{p_j}^{(\sum_{k=j}^{|p|-1} \delta_j(i_k))} \odot g_{p_{|p|}}^{(0)},$$

as claimed in the lemma. ■

The next two lemmas enable us to express repeated Lie derivatives of network vector fields directly in terms of the synaptic weights. The first lemma gives the

form of the derivative of a GNF network vector field in terms of a Hadamard product of network weight vectors, and the second relates a Hadamard product of derivatives of GNF vector fields to the network weights and non-adaptive components of the neuron equations.

Lemma 4.6

Given a heterogenous GNF network of n neurons with m inputs, the j^{th} GNF derivative field of order r , $0 \leq j \leq n + m$, can be expressed as

$$\mathbf{g}_y^{(r)}(=) \gamma_j^{0(r)}(\mathbf{y}) \odot \mathbf{W}_j^0 - \gamma_j^{1(r)}(\mathbf{y}) \odot \mathbf{W}_j^1 \quad (4.3.20)$$

where

$$\gamma_j^{0(r)}(\mathbf{y}) = \begin{bmatrix} \frac{\partial^r}{\partial y_1^r} \left(a_1(y_1) s_{j,1}^+(y_1) \right) \\ \vdots \\ \frac{\partial^r}{\partial y_n^r} \left(a_n(y_n) s_{j,n}^+(y_n) \right) \end{bmatrix},$$

$$\gamma_j^{1(r)}(\mathbf{y}) = \begin{bmatrix} \frac{\partial^r}{\partial y_1^r} \left(a_1(y_1) s_{j,1}^-(y_1) \right) \\ \vdots \\ \frac{\partial^r}{\partial y_n^r} \left(a_n(y_n) s_{j,n}^-(y_n) \right) \end{bmatrix}.$$

Proof: Recall from Definition 4.11 that the i^{th} elementary function of the j^{th} GNF derivative field of order r is $\frac{\partial^r g_{j,i}(y_i)}{\partial y_i^r}$. Taking repeated derivatives of $g_{j,i}(y_i)$ for $1 \leq i \leq n$ and $0 \leq j \leq n + m$, we have

$$\begin{aligned} g_{j,i}(y_i) &= a_i(y_i) s_{j,i}^+(y_i) W_{j,i}^0 - a_i(y_i) s_{j,i}^-(y_i) W_{j,i}^1 \\ &= \gamma_{j,i}^{0(0)}(y_i) W_{j,i}^0 - \gamma_{j,i}^{1(0)}(y_i) W_{j,i}^1. \end{aligned}$$

Taking repeated derivatives of this field we have:

$$\begin{aligned}
\frac{\partial g_{j,i}(y_i)}{\partial y_i} &= \frac{\partial \gamma_{j,i}^{0(0)}(y_i)}{\partial y_i} W_{j,i}^0 + \gamma_{j,i}^{0(0)}(y_i) \frac{\partial W_{j,i}^0}{\partial y_i} - \frac{\partial \gamma_{j,i}^{1(0)}(y_i)}{\partial y_i} W_{j,i}^1 - \gamma_{j,i}^{1(0)}(y_i) \frac{\partial W_{j,i}^1}{\partial y_i} \\
&= \gamma_{j,i}^{0(1)}(y_i) W_{j,i}^0 - \gamma_{j,i}^{1(1)}(y_i) W_{j,i}^1 \\
\frac{\partial^2 g_{j,i}(y_i)}{\partial y_i^2} &= \frac{\partial \gamma_{j,i}^{0(1)}(y_i)}{\partial y_i} W_{j,i}^0 + \gamma_{j,i}^{0(1)}(y_i) \frac{\partial W_{j,i}^0}{\partial y_i} - \frac{\partial \gamma_{j,i}^{1(1)}(y_i)}{\partial y_i} W_{j,i}^1 - \gamma_{j,i}^{1(1)}(y_i) \frac{\partial W_{j,i}^1}{\partial y_i} \\
&= \gamma_{j,i}^{0(2)}(y_i) W_{j,i}^0 - \gamma_{j,i}^{1(2)}(y_i) W_{j,i}^1 \\
&\vdots \\
\frac{\partial^r g_{j,i}(y_i)}{\partial y_i^r} &= \gamma_{j,i}^{0(r)}(y_i) W_{j,i}^0 - \gamma_{j,i}^{1(r)}(y_i) W_{j,i}^1
\end{aligned}$$

where $\gamma_{j,i}^{0(r)}(y_i) = \frac{\partial^r a_i(y_i) s_{j,i}^+(y_i)}{\partial y_i^r}$, and $\gamma_{j,i}^{1(r)}(y_i) = \frac{\partial^r a_i(y_i) s_{j,i}^-(y_i)}{\partial y_i^r}$. This last expression proves the claim. ■

Lemma 4.7

Given a heterosynaptic GNF network of n neurons with m inputs, a Hadamard product of GNF derivative fields can be expressed as the sum

$$\mathbf{g}_{p_1}^{(r_1)} \odot \mathbf{g}_{p_2}^{(r_2)} \odot \cdots \odot \mathbf{g}_{p_d}^{(r_d)} = \sum_{i_1=0}^1 \cdots \sum_{i_d=0}^1 \eta(i) \bigodot_{j=1}^d \left(\gamma_j^{i_j(r_j)} \odot \mathbf{W}_j^{i_j} \right).$$

where

$$i = i_1/i_2/\cdots/i_d,$$

$$\eta(i) = \prod_{j=1}^d (1 - 2i_j).$$

All other definitions are as in Lemma 4.6.

Proof: As we have seen in Lemma 4.6, the k^{th} elementary function of the r_j degree derivative of a GNF network vector field

$$\begin{aligned}
\mathbf{g}_j^{(r)}(\mathbf{y}) &= \gamma_j^{0(r_j)}(\mathbf{y}) \odot \mathbf{W}_j^0 - \gamma_j^{1(r_j)}(\mathbf{y}) \odot \mathbf{W}_j^1 \\
&= T_j^0 - T_j^1
\end{aligned} \tag{4.3.21}$$

always consists of two terms. The left term T_j^0 and right term T_j^1 relate the respective contributions of the excitatory and inhibitory weights (that fan out from neuron j) to the derivative field.

Expansion of the Hadamard product $\mathbf{g}_{p_1}^{(r_1)} \odot \mathbf{g}_{p_2}^{(r_2)} \odot \cdots \odot \mathbf{g}_{p_d}^{(r_d)}$ can be visualized as a binary tree, as shown in Figure 4.3. We start with the empty expression as the root node. At each level $i > 1$, we multiply each term in the previous level by the sum $T_{i-1}^0 - T_{i-1}^1$. The left and right branches under a node represent the terms produced by multiplication with T_{i-1}^0 and T_{i-1}^1 respectively. The leaf terms at level $d + 1$ are therefore a product of d terms. We can express the sum of the leaf terms in closed form as follows. Let the variables $i_j \in \{0, 1\}$, $1 \leq j \leq d$, respectively represent the branch taken ($0 = \text{LEFT}$, and $1 = \text{RIGHT}$) at level j . Then the 2^d possible variable assignments act as addresses that trace the combinations of branches that lead from the root node to a corresponding leaf term in the leaf level summation. The following repeated summation over the branch variables produces the leaf level summation:

$$\sum_{i_1=0}^1 \cdots \sum_{i_d=0}^1 \bigodot_{j=1}^d T_j^{i_j}.$$

Next, we substitute $\gamma_j^{0(r_j)}(\mathbf{y}) \odot \mathbf{W}_j^0$ for T_j^0 and $-\gamma_j^{1(r_j)}(\mathbf{y}) \odot \mathbf{W}_j^1$ for T_j^1 . Since the T_j^1 are negative for all j , those leaf terms that have an even number of indices with value 1 will produce a positive product, and those that have an odd number will produce a negative product. To capture this sign dependence on the variable assignments, we introduce the function

$$\eta(i) = \prod_{j=1}^{|i|} (1 - 2i_j) = \begin{cases} -1 & \text{odd number of } i_j \text{ are } 1 \\ +1 & \text{otherwise} \end{cases}$$

Combining these observations we have

$$\sum_{i_1=0}^1 \cdots \sum_{i_d=0}^1 \bigodot_{j=1}^d \eta(i) \gamma_j^{i_j(r_j)}(\mathbf{y}) \odot \mathbf{W}_j^{i_j}.$$

Figure 4.3: Evaluation Tree for Expansion of $\mathfrak{g}_{p_1}^{(r_1)} \odot \mathfrak{g}_{p_2}^{(r_2)} \odot \dots \odot \mathfrak{g}_{p_d}^{(r_d)}$

which proves the result. ■

In the event that the neurons in the network have the biologically realistic property that they only make one type of synaptic connection (i.e. they are homosynaptic), the Hadamard product described in the previous lemma is considerably simplified. This is captured by the following corollary.

Corollary 4.1

Given a homosynaptic GNF network of n neurons with m inputs, a Hadamard product of network derivative fields can be expressed as the sum

$$\mathbf{g}_{p_1}^{(r_1)} \odot \mathbf{g}_{p_2}^{(r_2)} \odot \dots \odot \mathbf{g}_{p_d}^{(r_d)} = \prod_{j=1}^d (1 - 2q_j) \left(\bigodot_{j=1}^d \gamma_j^{q_j(r_j)} \right) \odot \left(\bigodot_{j=1}^d \mathbf{W}_j^{q_j} \right).$$

The pattern of neuron valences is given by the multi-index

$$q = q_1 / \dots / q_d, \quad \text{where}$$

$$q_j = \begin{cases} 0 & \text{if neuron } j \text{ is excitatory.} \\ 1 & \text{if neuron } j \text{ is inhibitory} \end{cases} \quad 1 \leq j \leq d.$$

All other definitions are as in Lemma 4.7.

Proof: Recall from Definition 4.2 that the faux neuron with index 0 may be regarded as homosynaptic and excitatory. Therefore, we may handle all indices $0 \leq j \leq n + m$ uniformly.

If q_j represents the valence of the j^{th} neuron, then $1 - q_j$ represents the opposite valence. It must be the case that $\mathbf{W}_j^{1-q_j} = \mathbf{0}$, otherwise the neuron would make connections of opposite valence contradicting homosynaptivity. Since $\mathbf{0} \odot \mathbf{f} = \mathbf{f} \odot \mathbf{0} = \mathbf{0}$, any Hadamard product $\bigodot_{j=1}^d \mathbf{W}_j^{i_j}$ involving an index $i_j \neq q_j$ will be zero. Using the commutativity of the Hadamard product, we see that, if $i_1 / \dots / i_d \neq q_1 / \dots / q_d$, then

$$\bigodot_{j=1}^d \left(\gamma_j^{i_j(r_j)} \odot \mathbf{W}_j^{i_j} \right) = \left(\bigodot_{j=1}^d \gamma_j^{i_j(r_j)} \right) \odot \left(\bigodot_{j=1}^d \mathbf{W}_j^{i_j} \right) = \mathbf{0}.$$

We may conclude that the only nonzero term in the sum of Lemma 4.7 is

$$\left(\bigodot_{j=1}^d \gamma_j^{q_j(r_j)} \right) \odot \left(\bigodot_{j=1}^d \mathbf{w}_j^{q_j} \right).$$

as claimed. ■

4.3.2 Results for General Open GNF Networks

In this section we employ the results of the previous section to characterize the spanning set of the distribution Δ_* for general GNF networks (see (4.2.1)) with various mixtures of neurons.

We begin with the simplest type of GNF network — one composed entirely of neurons of the same valence. In this case each multi-Lie bracket in the span of Δ_* is a vector modified by a diagonal transformation that depends on the nonadaptive (metabolic) characteristics of the neurons. The vectors that are transformed are constant Hadamard products of the weight vectors. Each transformation and vector depends on a multi-index associated with the spanning field.

Theorem 4.1

An open monosynaptic GNF network, of n neurons with m inputs and valency $s \in \{0, 1\}$, has a d -dimensional reachability distribution

$$\Delta_* = \text{span} \left\{ \mathbf{D}_\pi(\mathbf{y}) \mathbf{v}_\pi : \pi \in \Omega_d^{(0, n+m)} \right\}.$$

where

$$1. \mathbf{D}_\pi(\mathbf{y}) =$$

$$\sum_{p \in \Phi(|\pi|)} \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \text{diag} \left[\sigma(p) \left(\bigodot_{j \in \pi^0} \gamma_j^0(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \odot \left(\bigodot_{j \in \pi^0} \gamma_j^s(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \right]$$

$$2. \mathbf{v}_\pi = \bigodot_{j \in \overline{\pi^0}} \mathbf{W}_j^s.$$

Proof: Given a multi-index p , a Hadamard product of synaptic gain derivatives, for a homosynaptic GNF network with valency pattern q , can be written in the form

$$\bigodot_{j=1}^{|\pi|} \gamma_j^{q_j(r_j)} = \left(\bigodot_{j \in \pi^0} \gamma_0^{0(r_j)}(\mathbf{y}) \right) \odot \left(\bigodot_{j \in \overline{\pi^0}} \gamma_j^{q_j(r_j)}(\mathbf{y}) \right). \quad (4.3.22)$$

This follows from Definition 4.4, where we see that for every $p_j = 0$ (i.e. $j \in \pi^0$) $q_j = 0$. The commutativity of the Hadamard product allows us to reorder the product such that all of the faux neuron synaptic gains occur first, as in equation (4.3.22). Similarly, we may reorder the Hadamard product of weight vectors as follows

$$\begin{aligned} \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^s &= \left(\bigodot_{j \in \pi^0} \mathbf{W}_0^0 \right) \odot \left(\bigodot_{j \in \overline{\pi^0}} \mathbf{W}_j^s \right) \\ &= \bigodot_{j \in \overline{\pi^0}} \mathbf{W}_j^s. \end{aligned} \quad (4.3.23)$$

The faux neuron weights do not affect the product because they are, by Definition 4.2, all equal to $+1$. Since the neurons in a monosynaptic network are homosynaptic, we may apply equations (4.3.22) and (4.3.23) to Corollary 4.1 and obtain, for a Hadamard product of monosynaptic GNF network derivative fields,

$$\begin{aligned} \mathbf{g}_{p_1}^{(r_1)} \odot \mathbf{g}_{p_2}^{(r_2)} \odot \dots \odot \mathbf{g}_{p_{|\pi|}}^{(r_{|\pi|})} &= \\ \prod_{j=1}^{|\pi|} (1 - 2s) \left(\bigodot_{j \in \pi^0} \gamma_0^{0(r_j)}(\mathbf{y}) \right) \odot \left(\bigodot_{j \in \overline{\pi^0}} \gamma_j^{q_j(r_j)}(\mathbf{y}) \right) \odot \left(\bigodot_{j \in \overline{\pi^0}} \mathbf{W}_j^s \right). \end{aligned} \quad (4.3.24)$$

This follows from the simple observation that, if s represents the valency of the network synapses ($s = 0$ if excitatory, $s = 1$ if inhibitory), then $1 - s$ is of the opposite valency. Therefore, the only non-zero Hadamard product is the one with non-faux neuron weight vectors of valency s , as shown in equation (4.3.24).

Substituting this equation into the sums of Lemma 4.5. combining with Lemma 4.4 and Lemma 4.1. and rearranging terms. we see that

$$\begin{aligned} & \left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdots \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] = \\ & \sum_{p \in \Phi(|\pi|)} \prod_{j=1}^{|\pi|} (1 - 2s) \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \sigma(p) \\ & \left(\bigodot_{j \in \pi^0} \gamma_0^{0(\omega(p,i,j))}(\mathbf{y}) \right) \odot \left(\bigodot_{j \in \pi^0} \gamma_j^{q_j(\omega(p,i,j))}(\mathbf{y}) \right) \odot \left(\bigodot_{j \in \pi^0} \mathbf{w}_j^s \right) \end{aligned} \quad (4.3.25)$$

where $\omega(p, i, j)$ is given in Lemma 4.5. The multiplying factor,

$$\prod_{j=1}^{|\pi|} (1 - 2s) = \pm 1.$$

may be dropped since it does not affect the distribution that is spanned by vector field (4.3.25).

Since a Hadamard product $\mathbf{a} \odot \mathbf{b}$ may also be written $\text{diag}[\mathbf{a}] \mathbf{b}$, we can make the appropriate assignment to $\mathbf{D}_\pi(\mathbf{y})$, and also assign to $\mathbf{v}_\pi$, thereby proving the lemma. ■

All of the evidence suggests that real biological networks consist of different classes of neurons which make either exclusively excitatory, or exclusively inhibitory connections with other cells. How do the reachability properties of such networks compare to monosynaptic nets? If we consider such a homosynaptic network, we find that a spanning field has the same general form as that for monosynaptic networks, except that the transformation matrix now includes a sum over a subset of permutations of the multi-index, allowing for more complicated reachability manifolds. However, as we shall see in Theorem 4.3, the homosynaptic spanning set is considerably simpler than the general heterosynaptic case.

Theorem 4.2

An open homosynaptic GNF network, of n neurons with m inputs, has a d -dimensional

reachability distribution

$$\Delta_{\bullet} = \text{span} \left\{ \mathbf{D}_{\pi}(\mathbf{y}) \mathbf{v}_{\pi} : \pi \in \Omega_d^{(0,n+m)} \right\}.$$

where

1. $q = q_1 / \cdots / q_{|\pi|}$ is the homosynaptic valence pattern for multi-index $p \in \Phi(|\pi|)$.

2. $\mathbf{D}_{\pi,q}(\mathbf{y}) =$

$$\sum_{p \in \Phi(|\pi|)} \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \text{diag} \left[\sigma(p) \left(\bigodot_{j \in \pi^0} \gamma_j^0(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \odot \left(\bigodot_{j \in \pi^{\bar{0}}} \gamma_j^{q_j}(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \right].$$

3. $\mathbf{v}_{\pi} = \bigodot_{j \in \pi^{\bar{0}}} \mathbf{W}_j^{q_j}$.

Proof: The proof is identical to that of Theorem 4.1 (the details of which are omitted to avoid repetition), except that the valence of the neurons is given by the homosynaptic valence patterns for π . Corollary 4.1 may be applied as in the previous proof. The scalar product

$$\prod_{j=1}^{|\pi|} (1 - 2q_j) = \pm 1.$$

is the same for all valence patterns of $p \in \Phi(|\pi|)$, because they are all permutations of the valence pattern for π , and have the same number of 0's and 1's. ■

Now, if we further generalize the network to the GNF form expressed in equation (4.2.1) we see that the form of the diagonal transformations that appear in the multi-Lie bracket summation are dependent on the multi-index of each of the $(d-1)!$ repeated Lie derivatives in the multi-Lie bracket sum.

Theorem 4.3

An open heterosynaptic GNF network, of n neurons with m inputs, has a d -dimensional reachability distribution

$$\Delta_{\bullet} = \text{span} \left\{ \sum_{q_1=0}^1 \cdots \sum_{q_{|\pi|-1}=0}^1 \sum_{q_{|\pi|=0}}^1 \mathbf{D}_{\pi,q}(\mathbf{y}) \mathbf{v}_{\pi,q} : \pi \in \Omega_d^{(0,n+m)} \right\}.$$

where

1. q is the multi-index containing the summation variables $q_1 / \cdots / q_{|\pi|}$.

2. $\mathbf{D}_{\pi}(\mathbf{y}) =$

$$\sum_{p \in \Phi(|\pi|)} \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \text{diag} \left[\sigma(p) \eta(q) \left(\bigodot_{j \in \pi^0} \gamma_j^0(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \odot \left(\bigodot_{j \in \pi^0} \gamma_j^{q_j}(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \epsilon_j(i_k))} \right) \right].$$

where

$$\eta(q) = \prod_{j=1}^{|q|} (1 - 2q_j).$$

3. $\mathbf{v}_{\pi} = \bigodot_{j \in \pi^0} \mathbf{W}_j^s$.

All other definitions are as in Theorem 4.2.

Proof: Once again, the proof is a straight forward consequence of the lemmas of Section 4.3.1. From Lemma 4.1 we know that a GNF system has a reachability distribution $\Delta_{\bullet}$ that is spanned by multi-Lie brackets. Lemma 4.4 tells us that these spanning vectors may be expressed as a summation over $2^{|\pi|}$ repeated Lie derivatives with multi-indices in the set $\Phi(|\pi|)$. Lemma 4.5 allows us to expand the terms in this sum into a sum of $(|\pi| - 1)!$ Hadamard products. Combining these results we

have

$$\begin{aligned} & \left[\mathbf{g}_{\pi_1}, \left[\mathbf{g}_{\pi_2}, \dots, \left[\mathbf{g}_{\pi_{|\pi|-1}}, \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] = \\ & \sum_{q_1=0}^1 \cdots \sum_{q_{|\pi|-1}=0}^1 \sum_{p \in \Phi(|\pi|)} \sum_{t_1=1}^1 \sum_{t_2=1}^2 \cdots \sum_{t_{|\pi|-1}=1}^{|\pi|-1} \\ & \sigma(|\pi|) \eta(q_1 \cdots q_{|\pi|-1}) \left(\bigodot_{j=1}^{|\pi|} \gamma_j^{\sum_{k=j}^{|\pi|-1} \delta_j(t_k)} \right) \odot \left(\bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j} \right). \end{aligned}$$

Letting

$$\mathbf{D}_{\pi,q}(\mathbf{y}) = \text{diag} \left[\sum_{t_1=1}^1 \sum_{t_2=1}^2 \cdots \sum_{t_{|\pi|-1}=1}^{|\pi|-1} \bigodot_{r=1}^{|\pi|} \gamma_r^{\sum_{k=j}^{|\pi|-1} \delta_j(t_k)} \right],$$

and $\mathbf{v}_{p,q} = \bigodot_{j=1}^{|\pi|} \mathbf{W}_{p_j}^{q_j}$, we obtain the multi-Lie bracket summation form stated in the declaration of the theorem. ■

4.3.3 General GNF Networks are Generically Nonholonomic

All along during the previous discussions we have alluded to the fact that, in general, GNF networks are effectively nonholonomic. We shall now conclude this section by showing that Theorems 4.1 to 4.3 imply that this is true.

Theorem 4.4

An open GNF network with monosynaptic, homosynaptic or heterosynaptic valences is effectively nonholonomic.

Proof: It is sufficient to show that open GNF networks, with any of the three types of valences described in the theorem, are *locally* non-holonomic on the state space, because the local result also implies the global case.

In Theorems 4.1, 4.2 and 4.3 we see that the spanning vectors have the form

$$\mathbf{D}(\mathbf{x}) \mathbf{v},$$

where $\mathbf{D}(\mathbf{x})$ is a diagonal function matrix, and the $\mathbf{v}$ are constant vectors. We are not concerned with the nonadaptive matrix, but rather with the degree to which the adaptive vectors can assure holonomy when $\mathbf{D}(\mathbf{x})$ is nonsingular.

Local holonomy requires that for an open region on which $\mathbf{D}(\mathbf{x})$ is constant dimensional, the subspace V spanned by the vectors $\mathbf{v} \in \Omega_d^{(0,n+m)}$ is a proper subset of $\mathbb{R}^n$. If this is not the case, then the adaptive components of the GNF network cannot be used to determine the dimensionality of the reachability distribution Δ , and a learning law cannot be found that ensures generic holonomy.

To see that $V \subset \mathbb{R}^n$ is *not* a generic property of the space of GNF weight matrices, it is sufficient to consider the space spanned by Hadamard products of a single column vector of these matrices, i.e. $\hat{\mathbf{v}} \in \mathbb{R}^n$. Sufficiency follows from the fact that the spanning vectors given in Theorems 4.1, 4.2 and 4.3 are all Hadamard products.

Let $\hat{\mathbf{v}}^r = \bigodot_{i=1}^r \hat{\mathbf{v}}$. We know from the definition of $\Omega_d^{(0,n+m)}$ that the spanning set will contain the vectors $\hat{\mathbf{v}}, \hat{\mathbf{v}}^1, \dots, \hat{\mathbf{v}}^n$. In order for the network to be holonomic, it must be the case that

$$\hat{\mathbf{v}}^n = \sum_{j=1}^{n-1} c_j \hat{\mathbf{v}}^j, \quad \text{for all } \hat{\mathbf{v}} \in \mathcal{U},$$

where $\mathcal{U}$ is an open region of $\mathbb{R}^n$. However, this implies that the coefficients must satisfy the set of linear equations

$$c_1 \hat{\mathbf{v}}_i + c_2 \hat{\mathbf{v}}_i^2 + \dots + c_{n-1} \hat{\mathbf{v}}_i^{n-1} - \hat{\mathbf{v}}_i^n = 0, \quad 1 \leq i \leq n,$$

on $\mathcal{U}$. Since these polynomial equations are only satisfied for a finite set of points $\hat{\mathbf{v}} \in \mathbb{R}^n$, generic holonomy fails because any infinitesimally small change in the weight matrix will make $V = \mathbb{R}^n$. This concludes the proof of Theorem 4.4. ■

Note, that this theorem states that *in the general case* holonomy cannot be assured, however, as we shall see in Section 4.4, this does not rule out the existence of special classes of networks that place constraints on the adaptive and nonadaptive terms of the neuron equations that guarantee holonomy.

4.4 Results for Classical Neural Network Models

In Chapter 2 we saw that the GNF family of networks generalizes the runtime architecture of many important classical neural network models. Having derived reachability results for the general case, we are now ready to explicitly examine these special cases. Once again our goal is to characterize the reachability distributions of these special case GNF networks in terms of the network weights. We begin by relating the multi-Lie brackets in Lemma 4.1 to the network weights for *shunting inhibition networks*.

4.4.1 Shunting Inhibition Networks

Recall from section Section 2.2.3.1 that these networks are of the form

$$\begin{aligned} \dot{x}_i = & -A_i x_i \\ & + (B_i - x_i) \left[\sum_{j=1}^n f_j(x_j) C_{j,i} Z_{j,i}^+ + \sum_{k=1}^m g_k(I_k) D_{k,i} W_{k,i}^0 \right] \\ & - (x_i + E_i) \left[\sum_{j=1}^n h_j(x_j) F_{j,i} Z_{j,i}^- + \sum_{k=1}^m i_k(I_k) G_{k,i} W_{k,i}^1 \right]. \end{aligned} \quad (4.4.1)$$

and can be put into GNF form by the change of variables $y_i = x_i + E_i$ as follows:

$$\begin{aligned} \dot{y}_i = & a_i(y_i) \left[b_i(y_i) + \sum_{j=1}^n \left(s_j^+(y_i) H_{j,i}^+ - s_j^-(y_i) H_{j,i}^- \right) d_j(y_j) \right. \\ & \left. + \sum_{k=1}^m \left(s_k^+(y_i) J_{k,i}^+ - s_k^-(y_i) J_{k,i}^- \right) u_k \right]. \end{aligned}$$

where we have for $1 \leq i \leq n$.

$$a_i(y_i) = y_i, \quad b_i(y_i) = \frac{A_i E_i}{y_i} - A_i. \quad (4.4.2)$$

$$s_{j,i}^+(y_i) = \frac{1}{y_i} (B_i - y_i + E_i), \quad s_{j,i}^-(y_i) = 1. \quad (4.4.3)$$

$$H_{j,i}^+ = C_{j,i} Z_{j,i}^+, \quad H_{j,i}^- = F_{j,i} Z_{j,i}^-. \quad (4.4.4)$$

$$J_{j,i}^+ = D_{j,i} W_{j,i}^0, \quad J_{j,i}^- = G_{j,i} W_{j,i}^1. \quad (4.4.5)$$

Notice that all of the excitatory synaptic gains are equal, as are all the inhibitory gains. This permits us to simplify our notation slightly in this section. We will refer to all the s_j^+ as s , and we will not refer to the s_j^1 which disappear in the Hadamard product with inhibitory weight vectors. The substitutions (4.3.1) through (4.3.3), in turn, allow us to express shunting inhibition networks in Isidori form (4.2.2).

The following lemma states that, under very reasonable conditions, we can always find an appropriate H_0^- and H_0^+ for any shunting inhibition network.

Lemma 4.8

The weight assignments

$$H_{0,i}^+ = \frac{A_i E_i}{E_i + B_i}, \quad \text{and} \quad H_{0,i}^- = A_i - \frac{A_i B_i}{E_i + B_i}, \quad 1 \leq i \leq n.$$

1. yield a valid heterosynaptic faux neuron if $E_i \neq -B_i$ for $1 \leq i \leq n$.
2. yield a valid excitatory homosynaptic faux neuron if $E_i \neq B_i$ and $B_i = 0$ for $1 \leq i \leq n$, and
3. yield a valid inhibitory homosynaptic faux neuron if $B_i \neq E_i$ and $E_i = 0$ for $1 \leq i \leq n$.

Proof:

$$\begin{aligned}
\mathbf{f}(\mathbf{y}) &= \mathbf{g}_0(\mathbf{y}) \\
&= \mathbf{a}(\mathbf{y}) \odot \mathbf{b}(\mathbf{y}) \\
&= \mathbf{a}(\mathbf{y}) \odot \mathbf{s}(\mathbf{y}) \odot \mathbf{H}_0^- - \mathbf{a}(\mathbf{y}) \odot \mathbf{H}_0^- \\
\Rightarrow \mathbf{s}(\mathbf{y}) \odot \mathbf{H}_0^- - \mathbf{H}_0^- &= \mathbf{b}(\mathbf{y}) \\
\Rightarrow \left(\frac{B_i - y_i + E_i}{y_i} \right) H_{0,i}^- - H_{0,i}^- &= \frac{A_i E_i}{y_i}, \quad 1 \leq i \leq n \\
\Rightarrow H_{0,i}^+ (E_i + B_i) &= A_i E_i, \quad \text{and} \\
H_{0,i}^+ + H_{0,i}^- &= A_i, \\
\Rightarrow H_{0,i}^+ &= \frac{A_i E_i}{E_i + B_i}, \quad \text{and} \\
H_{0,i}^- &= \frac{A_i B_i}{E_i + B_i}, \quad 1 \leq i \leq n.
\end{aligned}$$

and the result follows. ■

So we see from the preceding lemma that values $\mathbf{H}_0^+$ and $\mathbf{H}_0^-$ can always be found to make the network vector field $\mathbf{g}_0$ compatible with any type of network we wish to study. The condition $E_i \neq -B_i$, $1 \leq i \leq n$, is reasonable since, given appropriate initial conditions, the operating range of a shunting inhibition network of this form is (see Section 2.2.3.1)

$$0 < y_i < E_i + B_i, \quad 1 \leq i \leq n.$$

Furthermore, we are always free to choose $E_i = 0$, because we can compensate by increasing the size of B_i to compensate for the decrease in range, and vice versa. In this section we will assume that the faux neuron is heterosynaptic in a heterosynaptic network, excitatory in a general homosynaptic network, and the same valence as the other neurons in a monosynaptic network. Now that all neuron indices can be treated uniformly, we can redefine the synaptic gains for shunting inhibition networks as follows:

Definition 4.12

The synaptic gains for a *shunting inhibition network* are the same for all input signals. Specifically, the excitatory synaptic gain on the j^{th} signal is given by

$$\gamma^0(\mathbf{y}) = \begin{bmatrix} a_1(y_1) s_1(y_1) \\ \vdots \\ a_n(y_n) s_n(y_n) \end{bmatrix}$$

and the inhibitory synaptic gain is given by

$$\gamma^1(\mathbf{y}) = \begin{bmatrix} a_1(y_1) \\ \vdots \\ a_n(y_n) \end{bmatrix}$$

Having found a network form that is appropriate for controllability analysis, we may begin characterizing the controllability distribution Δ_* for variations of the shunting inhibition network. The following three technical lemmas allow us to significantly simplify the form of the general GNF transformations $\mathbf{D}_{\pi,q}(\mathbf{y})$ for shunting inhibition networks. The first lemma is straight forward, but the second two state rather surprising, and counterintuitive, simplifications that follow from it.

Lemma 4.9

For a shunting inhibition network of n neurons.

$$\begin{aligned} \gamma^{1(1)}(\mathbf{y}) = -\gamma^{0(1)}(\mathbf{y}) &= \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} && \text{and} \\ \gamma^{1(r)}(\mathbf{y}) = \gamma^{0(r)}(\mathbf{y}) &= \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, && r \geq 2. \end{aligned}$$

Proof: Applying substitutions (4.4.2) to the definition of γ^0 and γ^1 in Lemma 4.6. we have

$$\begin{aligned}\gamma_i^0(\mathbf{y}) &= a_i(\mathbf{y}) s_i(\mathbf{y}) = B_i - y_i + E_i \quad \text{and} \\ \gamma_i^1(\mathbf{y}) &= a_i(\mathbf{y}) = y_i.\end{aligned}$$

Taking derivatives with respect to y_i , for elements $1 \leq i \leq n$, immediately yields the desired result. ■

Lemma 4.10

Given a heterosynaptic shunting inhibition network of n neurons with m inputs, a repeating Lie derivative of network vector fields $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n+m}$, with multi-index p , may be expressed as the single Hadamard product

$$\mathcal{L}_{\mathbf{g}_{p_1 \pi_1}} \mathcal{L}_{\mathbf{g}_{p_1 \pi_1 - 1}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} = \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_1 \pi_1 - 1}^{(1)} \odot \mathbf{g}_{p_1 \pi_1}^0.$$

Proof: Recall from Lemma 4.5 that the general form of a repeated Lie derivative is a sum of Hadamard products of GNF field derivatives. We now show that there is exactly one term in this summation that is non-zero.

First we must show that any Hadamard product from the Lie derivative sum in Lemma 4.5 is zero if it contains even one derivative field with degree ≥ 2 . Applying the results of Lemma 4.9 to the Hadamard product of GNF field derivatives described in Lemma 4.6, we obtain

$$\mathbf{g}_j^{(r)}(\mathbf{y}) = \begin{cases} \gamma^{0(r)}(\mathbf{y}) \odot \mathbf{W}_j^0 - \gamma^{1(r)}(\mathbf{y}) \odot \mathbf{W}_j^1 & \text{if } r = 0 \\ -\mathbf{W}_j^0 - \mathbf{W}_j^1 & \text{if } r = 1 \\ 0 & \text{if } r > 1. \end{cases} \quad (4.4.6)$$

Therefore, a derivative field with degree $r \geq 2$ is zero, and any Hadamard product involving it will reduce to the zero vector, because $\mathbf{f} \odot \mathbf{0} = \mathbf{0} \odot \mathbf{f} = \mathbf{0}$.

Next we prove that exactly one term from the Lie derivative sum in Lemma 4.5 has derivative fields with degree < 2 , and therefore is non-zero. The proof proceeds

by induction on the tree described in Theorem 4.5 (see Figure 4.2). For the basis case at level 1 we have a single term $\mathbf{g}_{p_1}^{(1)} \odot \mathbf{g}_{p_2}^{(0)}$ with derivative degree less than 2. For the inductive step we assume that there is only one term with derivative degree less than 2 on level i . Clearly, all the children of a level i term with derivative degree ≥ 2 will also have derivatives with degree ≥ 2 , because each branch adds a degree to one of the vector fields in the child term. We also know from the proof of Theorem 4.5 that every new child term at level $i + 1$ contains a vector field $\mathbf{g}_{p_i}^{(0)}$ with derivative degree equal to zero (always the right-most field in each term of Figure 4.2). Therefore, the branch of the assumed low degree term in level i will have a branch that increments the degree of this zero degree field, and the resulting term at level $i + 1$ has degree < 2 . The induction therefore succeeds, and the claim is proven.

To complete the proof, we must show that the surviving term in the Lie derivative summation has the form claimed above. From the previous induction, we know that the surviving term at level i increments the field that has degree zero in its parent term, and adds a new field of degree zero. Clearly then (see Figure 4.2 for a quick verification), the degree of differentiation of the GNF derivative field that is not inherited from the parent node is zero, and all the others (accumulated from parent terms) are one. We conclude that the non-zero term is of the form

$$\mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|\pi|-1}}^{(1)} \odot \mathbf{g}_{p_{|\pi|}}^0, \quad (4.4.7)$$

as claimed. This completes the proof. ■

Lemma 4.11

Given a heterosynaptic shunting inhibition network of n neurons with m inputs, the Hadamard product of derivative fields $\mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|\pi|-1}}^{(1)} \odot \mathbf{g}_{p_{|\pi|}}^0$ can be expressed as

$$\left(\gamma^1 \odot \mathbf{W}_{|\pi|}^1 - \gamma^0 \odot \mathbf{W}_{|\pi|}^0 \right) \odot \left(\sum_{i_1=0}^1 \sum_{i_2=0}^1 \cdots \sum_{i_{|\pi|-1}=0}^1 \bigodot_{j=1}^{|\pi|-1} \mathbf{W}_j^{i_j} \right).$$

Proof: Employing the result of Lemma 4.7, we may rearrange the general form for a

Hadamard product of GNF derivative fields to obtain

$$\begin{aligned} \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|\pi|-1}}^{(1)} \odot \mathbf{g}_{p_{|\pi|}}^0 = \\ \sum_{i_1=0}^1 \cdots \sum_{i_{|\pi|}=0}^1 \left(\prod_{j=1}^{|\pi|} (1 - 2i_j) \right) \left(\bigodot_{k=1}^{|\pi|-1} \gamma^{i_k(1)} \right) \odot \gamma^{i_{|\pi|}(0)} \odot \left(\bigodot_{l=1}^{|\pi|} \mathbf{w}_l^{i_l} \right). \end{aligned} \quad (4.4.8)$$

We may apply the derivatives of Lemma 4.9 to simplify equation (4.4.8). Observing that $\gamma^{i_k(1)} = 2i_k - 1$, we have

$$\begin{aligned} \prod_{j=1}^{|\pi|} (1 - 2i_j) \left(\bigodot_{k=1}^{|\pi|-1} \gamma^{i_k(1)} \right) &= \prod_{j=1}^{|\pi|} (1 - 2i_j) \prod_{k=1}^{|\pi|-1} (2i_k - 1) \\ &= (1 - 2i_{|\pi|}) \prod_{j=1}^{|\pi|-1} (1 - 2i_j) (2i_j - 1) \\ &= (1 - 2i_{|\pi|}) (-1) \\ &= (2i_{|\pi|} - 1). \end{aligned} \quad (4.4.9)$$

for any assignment to the variables $i_k \in \{0, 1\}$, $1 \leq i \leq d$.

Finally, we prove the claim by substituting multiplier (4.4.9) into equation (4.4.8). Thus,

$$\begin{aligned} \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|\pi|-1}}^{(1)} \odot \mathbf{g}_{p_{|\pi|}}^0 &= \sum_{i_1=0}^1 \cdots \sum_{i_{|\pi|}=0}^1 (2i_{|\pi|} - 1) \gamma^{i_{|\pi|}(0)} \odot \left(\bigodot_{j=1}^{|\pi|} \mathbf{w}_j^{i_j} \right) \\ &= \left(\sum_{i_{|\pi|}=0}^1 (2i_{|\pi|} - 1) \gamma^{i_{|\pi|}(0)} \odot \mathbf{w}_{|\pi|}^{i_{|\pi|}} \right) \odot \left(\sum_{i_1=0}^1 \cdots \sum_{i_{|\pi|-1}=0}^1 \bigodot_{j=1}^{|\pi|-1} \mathbf{w}_j^{i_j} \right) \\ &= \left(\gamma^1 \odot \mathbf{w}_{|\pi|}^1 - \gamma^0 \odot \mathbf{w}_{|\pi|}^0 \right) \odot \left(\sum_{i_1=0}^1 \cdots \sum_{i_{|\pi|-1}=0}^1 \bigodot_{j=1}^{|\pi|-1} \mathbf{w}_j^{i_j} \right), \end{aligned}$$

which is the desired result. ■

Lemma 4.12

Given a heterosynaptic shunting inhibition network of n neurons with m inputs, a spanning multi-Lie bracket with multi-index $|\pi|$ of length ≥ 2 (see Lemma 4.1) can be written

$$\begin{aligned} & \left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdots \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] = \\ & \mathcal{L}_{\mathbf{g}_{\pi_2}} \mathcal{L}_{\mathbf{g}_{\pi_1}} \mathcal{L}_{\mathbf{g}_{\pi_3}} \cdots \mathcal{L}_{\mathbf{g}_{\pi_{|\pi|-1}}} \mathbf{g}_{\pi_{|\pi|}} - \mathcal{L}_{\mathbf{g}_{\pi_1}} \mathcal{L}_{\mathbf{g}_{\pi_2}} \mathcal{L}_{\mathbf{g}_{\pi_3}} \cdots \mathcal{L}_{\mathbf{g}_{\pi_{|\pi|-1}}} \mathbf{g}_{\pi_{|\pi|}}. \end{aligned} \quad (4.4.10)$$

where the order of the first two indices in the second term is reversed in the first. The order of the remaining indices is the same in both terms.

Proof: Let $\pi \in \hat{\Pi}_d^{(0,n-m)}$ be the multi-index for the spanning multi-Lie bracket in the lemma. The proof proceeds by induction on the binary tree that is used in the proof of Lemma 4.4 (see Figure 4.1). Recall that the multi-Lie bracket at the root of the tree is equal to the sum of the terms (nodes) in any given level of the tree. In particular it is equal to the sum of the leaf terms. We show that all but two of the leaf terms cancel each other and, therefore, the sum reduces to a difference of two terms as claimed.

We begin by showing that any two leaf terms with multi-indices $p^1, p^2 \in \Phi(|\pi|)$, such that $p^1_{|\pi|} = p^2_{|\pi|}$, are equal. To this end, observe that the leaf terms in the tree are of the form

$$\begin{aligned} & \mathcal{L}_{\mathbf{g}_{p^1_{|\pi|}}} \mathcal{L}_{\mathbf{g}_{p^1_{|\pi|-1}}} \cdots \mathcal{L}_{\mathbf{g}_{p^1_2}} \mathbf{g}_{p^1_1} = \\ & \left(\gamma^1 \odot \mathbf{W}_{|\pi|}^1 - \gamma^0 \odot \mathbf{W}_{|\pi|}^0 \right) \odot \left(\sum_{i_1=0}^1 \sum_{i_2=0}^1 \cdots \sum_{i_{|\pi|-1}=0}^1 \bigodot_{j=1}^{|\pi|-1} \mathbf{W}_j^{i_j} \right). \end{aligned} \quad (4.4.11)$$

where $p \in \Phi(|\pi|)$ (apply Lemma 4.11 to Lemma 4.10). Inspection of Lemma 4.4.11 reveals that the Hadamard products of weight vectors, in the right hand side of the equality, are the same for all leaves of the tree. This is so because the set of leaf multi-indices, $\Phi(|\pi|)$, consists of permutations of the indices in π (see the

proof of Lemma 4.4). Therefore, every leaf term has exactly the same indices in its Hadamard products. Of course, the leaves do differ, but only by the term $(\gamma^1 \odot \mathbf{W}_{|\pi|}^1 - \gamma^0 \odot \mathbf{W}_{|\pi|}^0)$, which is a function of the vector field $\mathbf{g}_{p_{|\pi|}}$, and is independent of all other vector fields in the leaf. Therefore, we may conclude that any two leaf terms with multi-indices $p^1, p^2 \in \Phi(|\pi|)$, such that $p_{|\pi|}^1 = p_{|\pi|}^2$, are equal as claimed.

Next, we show by induction that all but two leaf terms occur in pairs which are equal but of opposite sign, and therefore cancel. The induction proceeds on level i of the tree. We will make three inductive assumptions about level i :

1. Only one term in a level has a multi-Lie bracket in its highest index position⁹, p_i .
2. There is exactly one pair of terms that have different values for index $p_{|\pi|}$, and they inherit from the same term.
3. The right term multi-index has descending indices $p = i/i - 1/\dots/1$, and the left term multi-indices is the same except for reversal of positions p_i and p_{i-1} .

We choose level $i = 2$ as the base case. Assumption 1 is true because the root term has one multi-Lie bracket index, and its left and right children respectively insert scalar index $\pi_{|\pi|}$ to the left and right of this multi-Lie bracket, reducing the highest index. Only the right child still has a multi-Lie bracket in its highest term in level 2. Assumption 2 is also satisfied, because there is only one pair of terms in level two, both children inherit from the root, and their highest indices are different. The fact that there are only two indices in the level two terms, and that they are reversed in the two terms, satisfies Assumption 3. The reader should verify these statements by referring to Figure 4.1.

Now we choose an arbitrary level i . Select any term that does not have a multi-Lie bracket in its highest index position. Both children in level $i + 1$ will

⁹See the proof of Lemma 4.4 for a description of the use of multi-Lie brackets as indices in the decomposition tree.

have this property because new indices are always inserted immediately to the left and right of the multi-Lie bracket index. Therefore the scalar index in the highest position of the parent must also occur in the highest position of both children. If we pick the only level i term that, by assumption, does have a multi-Lie bracket in its highest index position, then we see that its right child in level $i + 1$ shares this property, but its left child does not. Therefore, Assumption 1 is satisfied in level $i + 1$. The index inserted in the right child is $|\pi| - i + 2$, and the highest index in the multi-Lie bracket is $|\pi| - i + 1$, and the left child is the same except for a reversal of positions p_i and p_{i-1} because the new index is inserted into the highest index position. This satisfies Assumption 3. As we have already seen, all terms in level i produce children that have the same index in their highest position, except for the one term that satisfies Assumption 1. Its children have different indices in the highest position, satisfying Assumption 2. The induction, therefore, succeeds.

Let us examine the pairs of leaves that share the same parent and have the same index in their highest position. As we saw in the proof of Lemma 4.4, the children of every node are of opposite sign. Since we also have shown that they are otherwise equal if they have the same index in their highest position, they must cancel each other in a summation of the leaf terms. Therefore, only the one pair of leaves satisfying Assumption 2 remain. The fact that Assumption 3 is satisfied means that the insertions into the right-most children always produce descending order, and its left sibling will also share this order except that the last insertion inverts the highest two index positions. As siblings, the two terms will be of opposite sign, but we may arbitrarily choose the sign order since they differ by -1 , and spanning vector fields of opposite sign span the same distribution. The two remaining terms therefore have the desired form, and the proof is complete. ■

The first shunting inhibition network that we consider is the simplest — a network of neurons that all make the same valence of synaptic connections.

Theorem 4.5

An open monosynaptic shunting inhibition network, of n neurons with m inputs, has d -dimensional reachability distribution

$$\Delta_* = \text{span} \{ \mathbf{D}(\mathbf{y}) \mathbf{W}_i^s : i \in [0, n + m] \}.$$

where

$$\mathbf{D}(\mathbf{y}) = \text{diag}[\gamma^s].$$

All other definitions are as in Theorem 4.1.

Proof: We employ the machinery introduced in the above lemmas to prove that multi-Lie brackets of length ≥ 2 are zero. In the following we let $s \in \{0, 1\}$ represent the valence of the network, and $1 - s$ the opposite valence. The selection of values for B_i and E_i ensure that the faux neuron has weight vector $\mathbf{W}_0^0 = 0$, and therefore, can be treated as a homosynaptic neuron with the network valence.

Observe that, for monosynaptic nets, given a multi-Lie bracket of length d , we have

$$\sum_{i_1=0}^1 \cdots \sum_{i_{d-1}=0}^1 \bigodot_{j=1}^{d-1} \mathbf{W}_j^{i_j} = \bigodot_{j=1}^{d-1} \mathbf{W}_j^s. \quad (4.4.12)$$

Assume that this is not the case. Then there is a term in the summation such that, for some $1 \leq k \leq d - 1$, variable $i_k = 1 - s$, and

$$\left(\bigodot_{j=1}^{k-1} \mathbf{W}_j^{i_j} \right) \odot \mathbf{W}_k^{1-s} \odot \left(\bigodot_{j=k+1}^{d-1} \mathbf{W}_j^{i_j} \right) \neq 0.$$

However, the definition of monosynaptivity tells us that $\mathbf{W}_k^{1-s} = 0$, and therefore

$$\left(\bigodot_{j=1}^{k-1} \mathbf{W}_j^{i_j} \right) \odot 0 \odot \left(\bigodot_{j=k+1}^{d-1} \mathbf{W}_j^{i_j} \right) = 0,$$

which contradicts the assumption.

Now, substituting equation (4.4.12) into Lemma 4.10.

$$\begin{aligned}
 \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|\pi|-1}}^{(1)} \odot \mathbf{g}_{p_{|\pi|}}^0 &= -\gamma^s \odot \mathbf{W}_{|\pi|}^s \odot \bigodot_{j=1}^{|\pi|-1} \mathbf{W}_j^{t_j} \\
 &= -\gamma^s \odot \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{t_j}. \tag{4.4.13}
 \end{aligned}$$

Let us consider the case when $|\pi| \geq 2$. Lemma 4.10 permits us to identify (4.4.13) with the repeated Lie derivative $\mathcal{L}_{\mathbf{g}_{p_{|\pi|}}} \mathcal{L}_{\mathbf{g}_{p_{|\pi|-1}}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1}$. Since equation (4.4.13) is not dependent on the ordering of the indices in multi-index p , we may conclude that

$$\begin{aligned}
 \mathcal{L}_{\mathbf{g}_{p_1}^1} \mathcal{L}_{\mathbf{g}_{p_{|\pi|-1}}^1} \cdots \mathcal{L}_{\mathbf{g}_{p_2}^1} \mathbf{g}_{p_1}^1 &= \mathcal{L}_{\mathbf{g}_{p_{|\pi|}}^2} \mathcal{L}_{\mathbf{g}_{p_{|\pi|-1}}^2} \cdots \mathcal{L}_{\mathbf{g}_{p_2}^2} \mathbf{g}_{p_1}^2. \\
 \forall p^1, p^2 &\in \Phi(|\pi|).
 \end{aligned}$$

Since Lemma 4.12 tells us that a multi-Lie bracket is the difference between two such equal Lie derivatives, we may conclude that, for any $\pi \in \Omega_d^{(0,n+m)}$, $d \geq 2$,

$$\left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdots \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] = \mathbf{0}.$$

The recursion (4.2.4) tells us that

$$\begin{aligned}
 \Delta_* &= \Delta_0 \\
 &= \text{span} \{ \mathbf{g}_0 \cdots \mathbf{g}_{n+m} \} \\
 &= \text{span} \{ \gamma^s \mathbf{W}_0^s \cdots \gamma^s \mathbf{W}_{n+m}^s \}.
 \end{aligned}$$

This completes the proof. ■

So we see that the spanning set of vector fields for the reachability distribution Δ_* has a simple form in which the network weight vectors are transformed by a characteristic diagonal matrix. This result is particularly appealing since it allows us to define a nonlinear coordinate transformation that *locally* flattens the reachability manifold of a monosynaptic shunting network around state space points that satisfy

a simple regularity condition. This, in turn, tells us that the computational power of monosynaptic shunting inhibition networks is equivalent to linear control systems on open regions of the state space. Furthermore, the nature of the coordinate change is directly related to the nonadaptive terms of the neuron equations. The following definition and corollary to Theorem 4.5 express this result.

Definition 4.13

Given a monosynaptic shunting inhibition network of n neurons with valence s ($s = 0$ if excitatory, $s = 1$ if inhibitory), and state space $\mathbb{R}^n$, a regular point is a member of the following set of states

$$\left\{ \mathbf{y} : \mathbf{y} \in \mathbb{R}^n, \quad \text{and} \quad \prod_{i=1}^n \gamma_i^s(\mathbf{y}) \neq 0 \right\}.$$

Corollary 4.2

Around each regular point, $\mathbf{y}_0$, of an open monosynaptic shunting inhibition network of n neurons with m inputs, there exists a nonlinear change of coordinates $\mathbf{z} = \Phi(\mathbf{y})$, and real valued matrices $\mathbf{A}_{\mathbf{y}_0}$ and $\mathbf{B}_{\mathbf{y}_0}$ such that the network's reachability distribution is equivalent to that of the linear control system

$$\dot{\mathbf{z}} = \mathbf{A}_{\mathbf{y}_0} \mathbf{z} + \mathbf{B}_{\mathbf{y}_0} \mathbf{u}.$$

Proof: If $\prod_{i=1}^n \gamma_i^s(\mathbf{y}_0) \neq 0$ at regular point $\mathbf{y}_0$, then $\frac{1}{\gamma_i^s(\mathbf{y}_0)}$ is defined for all $1 \leq i \leq n$, and the diagonal matrix $\text{diag}[\gamma_i^s(\mathbf{y}_0)]$ is nonsingular with inverse

$$\mathbf{D}^{-1}(\mathbf{y}_0) = \begin{bmatrix} \frac{1}{\gamma_1^s(\mathbf{y}_0)} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\gamma_n^s(\mathbf{y}_0)} \end{bmatrix}.$$

Letting

$$\mathbf{z} = \Phi(\mathbf{y}) = \mathbf{D}^{-1}(\mathbf{y}) \mathbf{y}, \tag{4.4.14}$$

we see from Theorem 4.5 that, in z coordinates, at $\mathbf{z}_0 = \Phi(\mathbf{y}_0)$ an open monosynaptic shunting inhibition network has reachability distribution

$$\begin{aligned}\Delta_*(\mathbf{z}_0) &= \text{span} \{ \mathbf{D}^{-1}(\mathbf{y}_0) \mathbf{D}(y) \mathbf{W}_i^s : i \in [0, n+m] \} \\ &= \text{span} \{ \mathbf{W}_0^s, \mathbf{W}_1^s, \dots, \mathbf{W}_{n+m}^s : . \} \end{aligned} \quad (4.4.15)$$

Since the weight vectors $\mathbf{W}_i^s$ in (4.4.15) are constant, Δ_* is a linear d -dimensional subspace. Let the vectors $\mathbf{b}_1, \dots, \mathbf{b}_d$ be a basis for Δ_* at $\mathbf{z}_0$.

Now, we may construct a linear system that has reachability properties locally equivalent to an open monosynaptic shunting inhibition network under coordinate change (4.4.14). It is well known from linear control theory [Won79] that the a linear control system of the form

$$\dot{\mathbf{w}} = \mathbf{A}\mathbf{w} + \mathbf{B}\mathbf{u} \quad (4.4.16)$$

is able to reach points on the linear subspace

$$\Delta_* = \text{Im} \left(\begin{bmatrix} \mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{A}^2\mathbf{B} \dots \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \right).$$

Choose any linear system with an arbitrary d -dimensional reachability subspace spanned by the basis $r_1, \dots, r_d$. Then there exists a linear change of coordinates $\mathbf{z} = \mathbf{T}\mathbf{w}$ such that

$$\mathbf{T} \begin{bmatrix} r_1 \dots r_d \end{bmatrix} = \begin{bmatrix} b_1 \dots b_d \end{bmatrix}.$$

In these coordinates the transformed system (4.4.16)

$$\begin{aligned}\dot{\mathbf{z}} &= \mathbf{T}\dot{\mathbf{z}} \\ &= \mathbf{TAT}^{-1}\mathbf{z} + \mathbf{TBu} \end{aligned} \quad (4.4.17)$$

has reachability subspace

$$\Delta_* = \text{Im} \left(\mathbf{T} \begin{bmatrix} \mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{A}^2\mathbf{B} \dots \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \right) = \text{span} \{ b_1, \dots, b_d \} = \Delta_*.$$

Setting $\mathbf{A}_{y_0} = \mathbf{TAT}^{-1}$ and $\mathbf{B}_{y_0} = \mathbf{TB}$, we see that system (4.4.17) is locally equivalent to the open monosynaptic shunting inhibition network with Δ_* at $\mathbf{y}_0$. ■

In the next theorem, we see that if we allow a mix of homosynaptic neurons, then Δ_* is the sum of the monosynaptic reachability distribution and the diagonal transformation $\mathbf{D}(y)$ that is applied to all spanning vectors in the monosynaptic case, becomes a function of the pattern of valences associated with each spanning field.

Definition 4.14

Two multi-indices π_1 and π_2 which contain indices of homosynaptic neurons, will be said to be valence equivalent ($\pi_1 \equiv \pi_2$), if and only if their corresponding homosynaptic valence patterns are equal (i.e. $q_j^1 = q_j^2$ for $1 \leq j \leq |\pi|$). This relation partitions any set of multi-indices, $\mathcal{S}$, into equivalence classes, $\mathcal{S}_k$, $1 \leq k \leq v$, each of which contains all $\pi \in \mathcal{S}$ that have the same homosynaptic valence pattern.

Definition 4.15

A valence reduction of multi-index set $\mathcal{S}$, is a largest subset of $\mathcal{S}$ such that its members all have unique homosynaptic valence patterns. More formally, the relation $\equiv$ of Definition 4.14 partitions $\mathcal{S}$ into v equivalence classes, $\mathcal{S}_k$, $1 \leq k \leq v$, each of which contains all of the permutations of $\pi \in \mathcal{S}$ that have the same homosynaptic valence pattern. The valence reduction of $\mathcal{S}$ is a set formed by selecting one representative multi-index at random from each equivalence class. It is clear that, in general, the valence reduction of $\mathcal{S}$ is not unique¹⁰.

Theorem 4.6

An open homosynaptic shunting inhibition network, of n neurons with m inputs, has a d -dimensional reachability distribution

$$\Delta_* = \text{span} \{ \mathbf{D}(\mathbf{y}) \mathbf{W}_i^s : i \in [0, n+m] \} + \text{span} \left\{ \mathbf{D}_\pi(\mathbf{y}) \mathbf{v}_\pi : \pi \in \mathcal{V} \left(\Omega_d^{(0, n+m)} \right), |\pi| \geq 2 \text{ and } q_1 \neq q_2 \right\}.$$

where

$$1. \mathcal{V} \left(\Omega_d^{(0, n+m)} \right) \in \Omega_d^{(1, n+m)} \text{ is any valence reduction of } \Omega_d^{(0, n+m)}.$$

¹⁰In fact, there will be $\prod_{j=1}^v |\mathcal{S}_k|$ valence reductions with v equivalence classes.

2. $q = q_1 / \cdots / q_{d-1}$ is a homosynaptic valence pattern for π .
3. $\mathbf{D}(\mathbf{y}) = \text{diag}[\gamma^s]$.
4. $\mathbf{D}_\pi(\mathbf{y}) = \text{diag}[\gamma^0 - \gamma^1]$.
5. $\mathbf{v}_\pi = \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j}$.

All other definitions are as given in Lemma 4.4, Lemma 4.7 and Theorem 4.2.

Proof: First, we consider the case where $|\pi| \geq 2$. Let q be the homosynaptic valence pattern for the network. Following the same line of reasoning as in the monosynaptic case, we see that for a multi-Lie bracket of length d ,

$$\sum_{i_1=0}^1 \cdots \sum_{i_{d-1}=0}^1 \bigodot_{j=1}^{d-1} \mathbf{W}_j^{i_j} = \bigodot_{j=1}^{d-1} \mathbf{W}_j^{q_j}. \quad (4.4.18)$$

Now, substituting equation (4.4.18) into Lemma 4.10,

$$\begin{aligned} \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|\pi|-1}}^{(1)} \odot \mathbf{g}_{p_{|\pi|}}^0 &= -\gamma^{q_{|\pi|}} \odot \mathbf{W}_{|\pi|}^{q_{|\pi|}} \odot \bigodot_{j=1}^{|\pi|-1} \mathbf{W}_j^{q_j} \\ &= -\gamma^{q_{|\pi|}} \odot \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j}. \end{aligned} \quad (4.4.19)$$

which, after substitution into Lemma 4.10 and Lemma 4.12, yields

$$\begin{aligned} \left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdots \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] &= -\gamma^{q_{\pi_2}} \odot \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j} + \gamma^{q_{\pi_1}} \odot \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j} \\ &= (\gamma^{q_{\pi_1}} - \gamma^{q_{\pi_2}}) \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j}. \end{aligned} \quad (4.4.20)$$

If we consider the field $\gamma^{q_{\pi_1}} - \gamma^{q_{\pi_2}}$ for all possible valence assignments q_{π_1} and q_{π_2} , we see that it is zero if the valences are equal and $\pm(\gamma^0 - \gamma^1)$ otherwise. Substituting into equation (4.4.20), we see that the only case that we must consider in the spanning

set of Δ_* is

$$\left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdots \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] = (\gamma^0 - \gamma^1) \bigodot_{j=1}^{|\pi|} \mathbf{W}_j^{q_j}.$$

as claimed in the theorem.

Finally, we consider the case where $|\pi| = 1$. In this case, $\mathbf{g}_i = \gamma^{q_i} \mathbf{W}_i^{q_i}$. Lemma 4.1 tells us that Δ_* is the span of all multi-Lie brackets with length ≥ 1 , and so the GNF network vector fields also span Δ_* . ■

Now, we may generalize this result somewhat by allowing a mix of heterosynaptic neurons in a shunting inhibition network. The following theorem tells us that the form of the multi-Lie bracket becomes a sum of transformed vectors, for these networks.

Theorem 4.7

An open shunting inhibition network with heterosynaptic neurons has a d -dimensional reachability distribution Δ_ spanned by Lie multibrackets of the form*

$$[\tau_d \cdot [\tau_{d-1} \cdots [\tau_2 \cdot \tau_1]]] = \sum_{q_1=0}^1 \cdots \sum_{q_{d-1}=0}^1 \mathbf{D}_q \mathbf{v}_q \quad (4.4.21)$$

where $q = [q_1 \dots q_{d-1}]$, as determined by the above summation, and all other definitions are as given in Theorem 4.5.

Proof: This proof follows exactly the same argument as that for Theorem 4.6 (the details of which are omitted to avoid repetition), except that the valencies of the fields in the repeated Lie derivative terms vary from term to term. Consequently, we must keep track of these valencies with the summation of Lemma 4.10. This satisfies the claim. ■

4.4.2 Bidirectional Associative Memories, Hopfield Networks, BSB Networks, and the McCulloch-Pitts Model

As we saw in Sections 2.2.3.1 through 2.2.3.6, a number of important network models share effectively the same simple, fully interconnected run-time architec-

ture. These include Bidirectional Associative Memories (BAMs), Hopfield networks, BSB networks, the McCulloch-Pitts model and myriad variations on these networks that have appeared in the literature. In this section we derive a spanning set for the reachability distribution of these networks. We use the definitions of Section 4.3.1 throughout this section.

Recall that all of the above models can be put into GNF form with the substitutions

$$a_i(y_i) = 1, \quad b_i(y_i) = -y_i. \quad (4.4.22)$$

$$s_{j,i}^-(y_i) = 1, \quad s_{j,i}^+(y_i) = 1. \quad (4.4.23)$$

In order to put the synaptic weights into the correct notation, we let

$$\begin{aligned} \dot{\mathbf{M}} &= \begin{bmatrix} \mathbf{0} & \mathbf{M} \\ \mathbf{M}^T & \mathbf{0} \end{bmatrix} \quad \text{for the BAM.} \\ &= \mathbf{M} \quad \text{for the Hopfield net. and} \\ &= \mathbf{B} \quad \text{for the BSB and McCulloch-Pitts model.} \end{aligned}$$

$\dot{\mathbf{M}}$ can be divided into excitatory and inhibitory matrices, as follows

$$H_{j,i}^+ = \dot{M}_{j,i}, \quad \text{iff } \dot{M}_{j,i} \geq 0. \quad (4.4.24)$$

$$H_{j,i}^- = \dot{M}_{j,i}, \quad \text{iff } \dot{M}_{j,i} \leq 0. \quad (4.4.25)$$

In these networks, a pool of $m \leq n$ neurons may be individually clamped by 1 positive input each. Consequently, we have¹¹ $\mathbf{J}^- = \mathbf{0}$ and $\mathbf{J}^+ = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_m]$.

The following lemma is analogous to Lemma 4.9 for shunting inhibition networks.

Lemma 4.13

For a BAM, Hopfield network, BSB network, and McCulloch-Pitts model with n

¹¹The canonical orthonormal set of basis vectors are denoted $\mathbf{e}_1, \dots, \mathbf{e}_n$.

neurons. and one unique input per neuron.

$$\begin{aligned}\gamma_0^{0(0)}(\mathbf{y}) &= -\mathbf{y}, & \gamma_0^{1(1)}(\mathbf{y}) &= \mathbf{0}, \\ \gamma_0^{0(1)}(\mathbf{y}) &= -\mathbf{1}, & \gamma_0^{1(1)}(\mathbf{y}) &= \mathbf{0}, \\ \gamma_0^{0(r)}(\mathbf{y}) &= \mathbf{0}, & \gamma_0^{1(r)}(\mathbf{y}) &= \mathbf{0}, \quad r \geq 2.\end{aligned}$$

and, for, $n+1 \leq j \leq n+m$,

$$\begin{aligned}\gamma_j^{0(0)}(\mathbf{y}) &= \mathbf{1}, & \gamma_j^{1(1)}(\mathbf{y}) &= \mathbf{1}, \\ \gamma_j^{0(r)}(\mathbf{y}) &= \mathbf{0}, & \gamma_j^{1(r)}(\mathbf{y}) &= \mathbf{0}, \quad r \geq 1.\end{aligned}$$

Proof: The result is immediate, and is obtained by substituting Definitions (4.4.22) through (4.4.22) into Definition 4.9 and taking derivatives. ■

As in the shunting inhibition case (see Lemma 4.10), the form of a repeated Lie derivative of network vector fields reduces to a single Hadamard product of associated derivative fields. However, Lemma 4.13 allows us to simplify the resulting vector field further.

Lemma 4.14

Given a BAM, Hopfield network, BSB network, or McCulloch-Pitts model with n neurons and one unique input per neuron, a repeating Lie derivative of network vector fields $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n+m}$, with multi-index p , may be expressed as the single Hadamard product

$$\begin{aligned}\mathcal{L}_{\mathbf{g}_{p_{|p|}}} \mathcal{L}_{\mathbf{g}_{p_{|p|-1}}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} \\ = \begin{cases} \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|p|-1}}^{(1)} \odot \mathbf{g}_{p_{|p|}}^0 & \text{if } p_j = 0, 1 \leq j \leq |p| - 1 \\ \mathbf{0} & \text{if } p_j \neq 0 \text{ for any } 1 \leq j \leq |p| - 1. \end{cases}\end{aligned}$$

Proof: The proof proceeds exactly as in Lemma 4.10, allowing us to conclude that

$$\mathcal{L}_{\mathbf{g}_{p_{|p|}}} \mathcal{L}_{\mathbf{g}_{p_{|p|-1}}} \cdots \mathcal{L}_{\mathbf{g}_{p_2}} \mathbf{g}_{p_1} = \mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|p|-1}}^{(1)} \odot \mathbf{g}_{p_{|p|}}^0.$$

However, we know from Lemma 4.13 that $\gamma_j^{0(r)}(\mathbf{y}) = \mathbf{0}$, any Hadamard product that includes this a derivative field reduces to zero, and therefore the claimed result follows. ■

Lemma 4.15

Given a BAM, Hopfield network, BSB network, or McCulloch-Pitts model with n neurons and one unique input per neuron, the Hadamard product of derivative fields $\mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|p|-1}}^{(1)} \odot \mathbf{g}_{p_{|p|}}^0$, where $p_j = 0, 1 \leq j \leq |p| - 1$, can be expressed as

$$\begin{cases} 1 & \text{if } p_{|p|} = 0 \\ \mathbf{H}_j^1 - \mathbf{H}_j^0 & \text{if } p_j \neq 0 \text{ and } 1 \leq j \leq n \\ -\mathbf{e}_j & \text{if } p_j \neq 0 \text{ and } n+1 \leq j \leq n+m. \end{cases}$$

Proof: The proof proceeds exactly as in Lemma 4.11, allowing us to conclude that

$$\mathbf{g}_{p_1}^{(1)} \odot \cdots \odot \mathbf{g}_{p_{|p|-1}}^{(1)} \odot \mathbf{g}_{p_{|p|}}^0 = \left(\gamma_{|p|}^1 \odot \mathbf{W}_{|p|}^1 - \gamma_{|p|}^0 \odot \mathbf{W}_{|p|}^0 \right) \odot \left(\sum_{t_1=0}^1 \sum_{t_2=0}^1 \cdots \sum_{t_{|p|-1}=0}^1 \bigodot_{j=1}^{|p|-1} \mathbf{W}_j^{t_j} \right).$$

We can simplify this further for the three cases in the lemma:

1. If $p_{|p|} = 0$, then from the definition of $\gamma_{|p|}^0$ and $\gamma_{|p|}^1$ in Lemma 4.13,

$$\begin{aligned} & \left(\gamma_{|p|}^1 \odot \mathbf{W}_{|p|}^1 - \gamma_{|p|}^0 \odot \mathbf{W}_{|p|}^0 \right) \odot \left(\sum_{t_1=0}^1 \sum_{t_2=0}^1 \cdots \sum_{t_{|p|-1}=0}^1 \bigodot_{j=1}^{|p|-1} \mathbf{W}_j^{t_j} \right) \\ &= (\mathbf{0} \odot \mathbf{1} + \mathbf{y} \odot \mathbf{1}) \bigodot_{j=1}^{|p|-1} \mathbf{1} \\ &= \mathbf{1}. \end{aligned}$$

2. If $p_k \neq 0, 1 \leq k \leq n$, then,

$$\begin{aligned} & \left(\gamma_k^1 \odot \mathbf{W}_k^1 - \gamma_k^0 \odot \mathbf{W}_k^0 \right) \odot \left(\sum_{t_1=0}^1 \sum_{t_2=0}^1 \cdots \sum_{t_{|p|-1}=0}^1 \bigodot_{j=1}^{|p|-1} \mathbf{W}_j^{t_j} \right) \\ &= (\mathbf{1} \odot \mathbf{H}_k^1 + \mathbf{1} \odot \mathbf{H}_k^0) \bigodot_{j=1}^{|p|-1} \mathbf{1} \\ &= \mathbf{H}_k^1 - \mathbf{H}_k^0. \end{aligned}$$

3. If $p_k \neq 0$, $n + 1 \leq k \leq n + m$, then.

$$\begin{aligned}
 & (\gamma_k^1 \odot \mathbf{W}_k^1 - \gamma_k^0 \odot \mathbf{W}_k^0) \odot \left(\sum_{t_1=0}^1 \sum_{t_2=0}^1 \cdots \sum_{t_{|p|-1}=0}^1 \bigodot_{j=1}^{|p|-1} \mathbf{W}_j^{t_j} \right) \\
 &= (\mathbf{1} \odot \mathbf{J}_k^1 - \mathbf{1} \odot \mathbf{J}_k^0) \bigodot_{j=1}^{|p|-1} \mathbf{1} \\
 &= -\mathbf{e}_j.
 \end{aligned}$$

This completes the proof. ■

Lemma 4.16

Given a heterosynaptic shunting inhibition network of n neurons with m inputs, all spanning multi-Lie brackets with length ≥ 2 are zero.

Proof: The proof proceeds exactly as in Lemma 4.12, yeilding

$$\begin{aligned}
 & \left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdots \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \mathbf{g}_{\pi_{|\pi|}} \right] \right] \right] = \\
 & \mathcal{L}_{\mathbf{g}_{\pi_2}} \mathcal{L}_{\mathbf{g}_{\pi_1}} \mathcal{L}_{\mathbf{g}_{\pi_3}} \cdots \mathcal{L}_{\mathbf{g}_{\pi_{|\pi|-1}}} \mathbf{g}_{\pi_{|\pi|}} - \mathcal{L}_{\mathbf{g}_{\pi_1}} \mathcal{L}_{\mathbf{g}_{\pi_2}} \mathcal{L}_{\mathbf{g}_{\pi_3}} \cdots \mathcal{L}_{\mathbf{g}_{\pi_{|\pi|-1}}} \mathbf{g}_{\pi_{|\pi|}}.
 \end{aligned} \tag{4.4.26}$$

Combining Lemmas 4.14 and 4.15, we know that the two repeated Lie derivatives must be equal because they are both functions of $\pi_{|\pi|}$. Consequently the difference between these terms is zero, and the lemma is proven. ■

We are now ready to state the main result of this section which is applicable to the set of neural networks considered, namely BAMs, Hopfield networks, BSB networks, and McCulloch-Pitts model. We find that the reachability distribution is linear, and is spanned by constant vector fields.

Theorem 4.8

For a BAM, Hopfield network, BSB network, or McCulloch-Pitts model with n neurons and one unique input per neuron, the d -dimensional reachability distribution is as follows:

$$\Delta_* = \text{Im}(\dot{\mathbf{M}}) + \text{span} \{ \mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{n+m} \}.$$

Proof: From Lemma 4.16 we know that multi-Lie brackets of length greater than one do not contribute to the spanning set. Therefore, recursion (4.2.4) must terminate at step 0, and we need only consider the set spanned by the GNF network vector fields $\mathbf{g}_j$ for $1 \leq j \leq n + m$. The fields $1 \leq j \leq n$ contribute the constant vectors

$$\mathbf{g}_j = \mathbf{a} \odot \mathbf{s} \odot \mathbf{H}_j^+ - \mathbf{a} \mathbf{H}_j^- = \mathbf{H}_j^+ - \mathbf{H}_j^- = \hat{\mathbf{M}}_j,$$

and the fields $1 \leq j \leq m$ contribute the vectors

$$\mathbf{g}_j = \mathbf{a} \odot \mathbf{s} \odot \mathbf{J}_j^+ - \mathbf{a} \mathbf{J}_j^- = \mathbf{J}_j^+ = \mathbf{e}_j,$$

which completes the proof. ■

4.5 Reachability Results for Closed GNF Networks

In Section 4.3 we considered the case where the network neurons are not interconnected. Thus, instead of receiving output signals from all other neurons in the network, a given neuron was assumed to be receiving *only* external input signals. Although results for such open networks provide bounds on the network reachability distribution that are sufficient for many applications, we are also interested in the more realistic situation in which the network neurons are interconnected. Therefore, in this section we consider what happens to the reachability distribution of a GNF network when connections are established between neurons — i.e. the intra-network connections are ‘closed’, as in equation (4.2.1). Indeed, one may think of a closed network as a special case of an open network in which the first n inputs functions are equated with the outputs of the n network neurons, i.e. $d_j(x_j)$, for $1 \leq j \leq n$. It seems intuitively obvious that reducing the set of allowable input functions will restrict the reachability set of a closed network to a subset of its equivalent open network reachability set. Indeed, we will show that the closed reachability sets are submanifolds of the open ones, and we will characterize the spanning sets of the closed reachability distribution Δ_* .

4.5.1 Supporting Definitions and Lemmas

In a closed GNF network, the drift dynamics are governed by the interaction between the interconnected neurons. We must capture this fact by defining the closed GNF network vector fields in terms of the open ones as follows:

Definition 4.16

We define the closed GNF network vector fields, $\hat{\mathbf{g}}_0(x), \hat{\mathbf{g}}_1(x), \dots, \hat{\mathbf{g}}_m(x)$, to be

$$\begin{aligned} \hat{g}_{0,i} &= \hat{f}_i(\mathbf{x}) \\ &= g_{0,i} + \sum_{k=1}^n d_k(x_k) g_{k,i}(x_i), \\ &\text{for } 1 \leq i \leq n. \end{aligned} \tag{4.5.1}$$

$$\begin{aligned} \hat{g}_{j,i} &= g_{n+j,i}(x_i), \\ &\text{for } 1 \leq i \leq n, \text{ and } 1 \leq j \leq m. \end{aligned} \tag{4.5.2}$$

This definition allows us to transform the closed network equation (4.2.1) into the form of an Isidori-style control system

$$\dot{\mathbf{x}} = \hat{\mathbf{f}}(\mathbf{x}) + \sum_{j=1}^m \hat{\mathbf{g}}_j(x) u_j.$$

With these definitions in mind, we may begin the derivation of the spanning set for closed GNF network reachability distributions. We begin with a corollary to Lemma 4.1, in which we see that a closed GNF network reachability distribution is spanned by multi-Lie brackets of the same general form as an open GNF network.

Corollary 4.3

The d -dimensional reachability distribution Δ_* of a closed GNF network with n neurons and m inputs is of the form

$$\begin{aligned} \Delta_* &= \text{span} \left\{ \mu_\pi : \pi \in \hat{\Pi}_d^{(0,m)} \right\}, \quad \text{where} \\ \mu_\pi &= \begin{cases} \hat{\mathbf{g}}_{\pi_1} & \text{if } |\pi| = 1 \\ \left[\hat{\mathbf{g}}_{\pi_1}, \left[\hat{\mathbf{g}}_{\pi_2}, \dots, \left[\hat{\mathbf{g}}_{\pi_{|\pi|-1}}, \hat{\mathbf{g}}_{\pi_{|\pi|}} \right] \right] \right] & \text{if } |\pi| > 1. \end{cases} \end{aligned}$$

Proof: The proof is immediate by replacing the open GNF network fields in Lemma 4.1 by the closed GNF network fields of Definition 4.16. ■

Although the closed network spanning set has the same form as the open network spanning set, the expansion of the multi-Lie brackets is quite different.

The difference is due to the fact that the open network field $\mathbf{g}_0$ is not of the same form as the closed network field $\hat{\mathbf{g}}_0$. $\hat{\mathbf{g}}_0$ is a sum of open network vector fields, all but one of which are weighted by neuron output functions. Furthermore, the elementary functions $\hat{g}_{0,i}$ are no longer functions of the associated coordinate x_i , but rather the entire coordinate $\mathbf{x}$. Lemma 4.5 and Lemma 4.6, therefore, no longer apply. In fact, as we shall see, the closed network multi-Lie brackets are *linear combinations* of a subset of related open network multi-Lie brackets at each point x .

The following lemma re-states an identity that allows us to reduce a single Lie bracket involving sums of fields into a sum of brackets of single fields, and two sums of weighted vector fields.

Lemma 4.17 ([Isi89], Section 1.3)

$$\left[\sum_{k_1=1}^{n_1} c_{k_1} \mathbf{f}_{k_1}, \sum_{k_2=1}^{n_2} c_{k_2} \mathbf{f}_{k_2} \right] = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} c_{k_2} [\mathbf{f}_{k_1}, \mathbf{f}_{k_2}] - \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_2} \left(\mathcal{L}_{\mathbf{f}_{k_2}} c_{k_1} \right) \mathbf{f}_{k_1} + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} \left(\mathcal{L}_{\mathbf{f}_{k_1}} c_{k_2} \right) \mathbf{f}_{k_2}.$$

Proof: The proof proceeds by straightforward algebra. since.

$$\begin{aligned}
& \left[\sum_{k_1=1}^{n_1} c_{k_1} \mathbf{f}_{k_1} \cdot \sum_{k_2=1}^{n_2} c_{k_2} \mathbf{f}_{k_2} \right] \\
&= \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \left(\frac{\partial c_{k_2}}{\partial \mathbf{x}} \mathbf{f}_{k_2} c_{k_1} \mathbf{f}_{k_1} + c_{k_1} c_{k_2} \frac{\partial \mathbf{f}_{k_2}}{\partial \mathbf{x}} \mathbf{f}_{k_1} \right) - \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \left(\frac{\partial c_{k_1}}{\partial \mathbf{x}} \mathbf{f}_{k_1} c_{k_2} \mathbf{f}_{k_2} + c_{k_2} c_{k_1} \frac{\partial \mathbf{f}_{k_1}}{\partial \mathbf{x}} \mathbf{f}_{k_2} \right) \\
&= \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} (\partial c_{k_2} \mathbf{f}_{k_2}) \mathbf{f}_{k_1} - \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_2} (\partial c_{k_1} \mathbf{f}_{k_1}) \mathbf{f}_{k_2} + c_{k_1} c_{k_2} \left(\frac{\partial \mathbf{f}_{k_2}}{\partial \mathbf{x}} \mathbf{f}_{k_1} - \frac{\partial \mathbf{f}_{k_1}}{\partial \mathbf{x}} \mathbf{f}_{k_2} \right) \\
&= \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} c_{k_2} [\mathbf{f}_{k_1}, \mathbf{f}_{k_2}] - \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_2} (\mathcal{L}_{\mathbf{f}_{k_2}} c_{k_1}) c_{k_1} + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} (\mathcal{L}_{\mathbf{f}_{k_1}} c_{k_2}) c_{k_2}.
\end{aligned}$$

This completes the proof. ■

4.5.2 Results for Closed GNF Networks

We are now ready to state our main results for this section — that a closed network multi-Lie bracket may be described as a weighted sum of open network multi-Lie brackets. Substituting the expressions for the open network fields given in Theorem's 4.1 to 4.8 into the multi-Lie bracket expression given in the following theorem yields the reachability distribution for closed GNF networks.

Theorem 4.9

The multi-Lie brackets that span the reachability distribution $\hat{\Delta}_$ of a closed GNF network. Each multi-Lie bracket spanning a closed GNF network Δ_* with multi-index $\pi \in \hat{\Pi}_d^{(0,m)}$, is a weighted sum of multi-Lie brackets spanning the Δ_* of the same GNF network with open connections, i.e.*

$$\begin{aligned}
& \left[\hat{\mathbf{g}}_{\pi_1 \pi_1}, \left[\hat{\mathbf{g}}_{\pi_1 \pi_1 - 1}, \dots, [\hat{\mathbf{g}}_{\pi_2}, \hat{\mathbf{g}}_{\pi_1}] \right] \right] (\mathbf{x}) = \\
& \sum_{i_1=2}^2 \sum_{i_2=0}^2 \dots \sum_{i_1 \pi_1=0}^2 \sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \dots \sum_{j_1 \pi_1=p_{\pi_1}}^{q_{\pi_1 \pi_1}} c^{i,j}(\mathbf{x}) \beta^{i,j}(\mathbf{x}),
\end{aligned}$$

where

1. $i = i_1/i_2/\cdots/i_{|\pi|}$ is a multi-index taking on the $3^{|\pi|-1}$ combinations of indices $i_1 = 2$ and $i_j \in \{0, 1, 2\}$, $2 \leq j \leq |\pi|$, and $j = j_1/j_2/\cdots/j_{|\pi|}$ is a multi-index with indices in the integer range $[0, n + m]$.
2. the $c^{i,j} \in C^\infty$ are scalar weighting functions.
3. $\beta^{i,j}$ is the open GNF network multi-Lie bracket $\left[\mathbf{g}_{p_{|\pi|}} \cdot \left[\mathbf{g}_{p_{|\pi|-1}} \cdots \left[\mathbf{g}_{p_2} \cdot \mathbf{g}_{p_1} \right] \right] \right]$, with multi-index $p = \varphi(1, j_1, |\pi|)$ defined recursively as¹²

$$\varphi(r, k, l) = \begin{cases} k & \text{if } r \geq l. \\ \varphi(r+1, j_r/k, l) & \text{if } i_r = 0. \\ \varphi(r+1, k, l) & \text{if } i_r = 1. \\ \varphi(r+1, j_r, l) & \text{if } i_r = 2. \end{cases}$$

Proof: It will be convenient in the following derivation to give all of the closed network vector fields the same syntactic form.

$$\dot{\mathbf{g}}^{\pi_k} = \sum_{j_k = p^{\pi_k}}^{q^{\pi_k}} d_{j_k}^{\pi_k} \mathbf{g}_{j_k}.$$

by letting

$$p_0 = 1, \quad q_0 = n$$

$$p_k = n + k, \quad q_k = n + k \quad 1 \leq k \leq m.$$

$$d_j^0(x) = d_j(x_j), \quad p_0 \leq j \leq q_0, \quad \text{and.}$$

$$d_j^k(x) = 1, \quad p_k \leq j \leq q_k.$$

We may now construct a trinary evaluation tree for an arbitrary closed network multi-Lie bracket by repeated application of Lemma 4.17. The evaluation tree is constructed by expanding Lie brackets, starting with the inner-most bracket

¹² The notation j_r/k signifies the creation of a multi-index of length $l+1$ by prepending an index onto the first position of length l multi-index k .

first, working outward. The nodes of the tree represent algebraic terms. The root node represents the innermost, length 1 bracket $\dot{\mathbf{g}}_{\pi_1}$. The children of the k^{th} level $r - 1$ node $\eta_{(r-1),k}$ represent the three terms that result from applying expansion of Lemma 4.17 to $[\dot{\mathbf{g}}_{\pi_r}, \eta_{(r-1),k}]$. The sum of nodes $\eta_{r,1}, \dots, \eta_{r,3^{r-1}}$ at level $r > 1$ equals the Lie bracket formed from the closed network field $\dot{\mathbf{g}}_{\pi_r}$ and the sum of the nodes at the previous level. Level r therefore represents the multi-Lie bracket

$$\begin{aligned} \sum_{k=1}^{3^{r-1}} \eta_{r,k} &= \left[\dot{\mathbf{g}}_{\pi_r} \cdot \sum_{k=1}^{3^{r-2}} \eta_{(r-1),k} \right] \\ &= \sum_{k=1}^{3^{r-2}} [\dot{\mathbf{g}}_{\pi_r} \cdot [\dot{\mathbf{g}}_{\pi_{r-1}} \cdots [\dot{\mathbf{g}}_{\pi_2} \cdot \dot{\mathbf{g}}_{\pi_1}]]] . \end{aligned}$$

For $1 \leq k \leq 3^{r-1}$, the sum of the $3^{|\pi|-1}$ leaf nodes in level $|\pi|$ is equal to the length $|\pi|$ closed network multi-Lie bracket.

To uniquely identify the nodes of the evaluation tree at level $r > 1$, we introduce the multi-index i of three state variables $i_k \in \{0, 1, 2\}$, $2 \leq k \leq r$. The three child branches below each parent node correspond to the three terms of the expansion in Lemma 4.17 which are labeled 0, 1 and 2 respectively. We interpret an assignment to variable i_r as a choice of branch from a node at level $r - 1$ to a node at level k . Each assignment to the multi-index i identifies a level r node by the unique sequence of $r - 1$ branchings that lead to it from the root node. For completeness we assign the state variable $i_1 = 2$ to be the label of the root node. Since the tree has $|\pi|$ levels, we will need $|\pi|$ state variables to label the leaf nodes.

The proof of the closed network multi-Lie bracket expansion continues by induction on the levels of the evaluation tree. We make the inductive assumption that the nodes labeled by $i = i_1/i_2/\dots/i_r$ at level r are of the form

$$\sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_r=p_{\pi_r}}^{q_{\pi_r}} c^{i,j,r}(\mathbf{x}) \beta^{i,j,r}(\mathbf{x}) , \quad (4.5.3)$$

where $c^{i,j,r}$ is a scalar weighting function, and $\beta^{i,j,r}$ is a multi-Lie bracket of open GNF network fields with multi-index given by $\varphi(1, j_1, r)$ as defined above.

The base case is proved by investigating the root level with $r = 1$. Since the root node represents the summation

$$\hat{\mathbf{g}}_{\pi_1} \sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} d_{j_1}^{\pi_1} \mathbf{g}_{j_1}.$$

the inductive assumption is satisfied by letting $c^{t,j,1} = d_{j_1}^{\pi_1}$, and observing that:

$$\begin{aligned} \mathbf{g}_{j_1} &= \mathbf{g}_{\mathcal{A}(1,j_1,1)} \\ &= \beta^{t,j,1}. \end{aligned}$$

We must consider the three level r children that an arbitrary level $r - 1$ node contributes to level r by application of Lemma 4.17 to

$$\left[\sum_{j_r=p_{\pi_r}}^{q_{\pi_r}} d_{j_r}^{\pi_r} \mathbf{g}_{j_r} \cdot \sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_{r-1}=p_{\pi_{r-1}}}^{q_{\pi_{r-1}}} c^{t,j,r-1}(\mathbf{x}) \beta^{t,j,r-1}(\mathbf{x}) \right].$$

There are three cases to consider that represent the three children in level r :

Branch 0 ($i_r = 0$): Lemma 4.17 tells us that this child node will be of the form

$$\sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_r=p_{\pi_r}}^{q_{\pi_r}} d_{j_r}^{\pi_r} c^{t,j,r-1} [\mathbf{g}_{j_r} \cdot \beta^{t,j,r-1}].$$

The inductive assumption is satisfied for this case, because the multi-Lie bracket has multi-index $j_r/\varphi(1, j_1, r-1)$, and we may assign $c^{t,j,r} = d_{j_r}^{\pi_r} c^{t,j,r-1}$.

Branch 1 ($i_r = 1$): In this case the child node is of the form

$$- \sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_r=p_{\pi_r}}^{q_{\pi_r}} c^{t,j,r-1} \left(\mathcal{L}_{j_1,j,r-1} d_{j_r}^{\pi_r} \right) \mathbf{g}_{j_r},$$

and we may assign $c^{t,j,r} = d_{j_r}^{\pi_r} c^{t,j,r-1}$ (absorbing the change in sign into the weighting function). The inductive assumption is again satisfied, because the length 1 multi-Lie bracket $\mathbf{g}_{j_r}$ has index j_r as expected.

Branch 2 ($i_r = 2$): In this final case the child node is

$$\sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_r=p_{\pi_r}}^{q_{\pi_r}} d_{j_r}^{\pi_r} (\mathcal{L}_{\mathbf{g}_{j_r}} c^{i \cdot j \cdot r-1}) \beta^{i \cdot j \cdot r-1},$$

and we may assign $c^{i \cdot j \cdot r} = -d_{j_r}^{\pi_r} \mathcal{L}_{\mathbf{g}_{j_r}} c^{i \cdot j \cdot r-1}$. The multi-Lie bracket $\beta^{i \cdot j \cdot r-1}$ has the same index as in the previous level satisfying the inductive assumption.

Clearly, if $r \geq |\pi|$ we are at a leaf node and the induction terminates. Since each branch leading to a node in level r satisfies the assumption, we may conclude that the induction succeeds, and the assumption holds, at all levels of the tree.

Consider the leaf nodes of level $r = |\pi|$, and recall that the original closed network multi-Lie bracket is equal to the sum of the corresponding leaves. Since each leaf node has form (4.5.3), if we assign

$$c^{i \cdot j} = c^{i \cdot j \cdot |\pi|}, \beta^{i \cdot j} = \beta^{i \cdot j \cdot |\pi|},$$

and taking the sum of the values of the leaves over all the possible state variables i_k , $1 \leq k \leq |\pi|$, we get

$$\sum_{i_1=2}^2 \sum_{i_2=0}^2 \cdots \sum_{i_{|\pi|=0}}^2 \sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_{|\pi|=p_{\pi_{|\pi|}}}}^{q_{\pi_{|\pi|}}} c^{i \cdot j} \beta^{i \cdot j},$$

which completes the proof. ■

Now that we know that the closed multibrackets are weighted contributions of the open multibrackets, we are naturally interested in knowing how the weights vary with x . The following lemma provides the closed form of the weighting functions in the previous lemma.

Theorem 4.10

The scalar functions that weight the sum of open network multi-Lie brackets in a

closed network multi-Lie bracket are of the form

$$c^{i,j} = \sum_{k_1=1}^n \cdots \sum_{k_{|\pi|=1}}^n \sum_{l_1=1}^1 \sum_{l_2=1}^2 \cdots \sum_{l_{|\pi|=1}}^{|\pi|} \prod_{z=1}^{|\pi|-1} \eta_i D_{|\pi|}^2 D_{|\pi|-1}^2 \cdots D_{z+1}^2 \left([\delta_{i_z}(1) (\beta^{i,j_z,z-1} - 1) + 1] [\delta_{i_z}(2) (\mathbf{g}_{\pi_{j_z}, k_z} - 1) + 1] \hat{D}_z^1 d_{j_z}^{\pi_{j_z}} \right).$$

where

1. *the sign of the expansion is given by*

$$\begin{aligned} \eta_i &= (-1)^{|\pi| - \sum_{i=1}^{|\pi|} (1 - i_i)^2} \\ &= \begin{cases} +1 & \text{if even number of 1's in } i. \\ -1 & \text{otherwise.} \end{cases} \end{aligned}$$

2. *the conditional differential operators are defined as*

$$\begin{aligned} \hat{D}_v^u &= \begin{cases} \partial x_{j_v} & \text{if } i_u = u. \\ \mathcal{I} & \text{otherwise.} \end{cases} \\ D_v^u &= \begin{cases} \frac{\partial}{\partial x_{j_v}} & \text{if } i_v = u \text{ and } z = l_v. \\ \mathcal{I} & \text{otherwise.} \end{cases} \end{aligned}$$

All other definitions are as in Theorem 4.9.

Proof:

First, we will prove that the weighting functions $c^{i,j,r}$, for all nodes labeled by multi-index i in levels $1 \leq r \leq |\pi|$, have a particular recursive definition. Then we show that the closed form of this recursion yields the claimed result when summed over all the leaf nodes in the tree.

The proof initially proceeds along the same lines as that of Theorem 4.9 — that is to say, by induction on the levels of the closed network multi-Lie bracket tree. We will make use of definitions made in that proof.

The inductive assumption shall be that all nodes at each level r of the tree have weighting functions with recursive definition:

$$c^{i,j,r} = \vartheta(1, 1, r), \quad \text{where}$$

$$\vartheta(r, \xi, l) = \begin{cases} \xi & \text{if } z \geq l. \\ \vartheta(z + 1, d_{j_z}^{\pi_{j_z}}(x_{j_z}) \xi, l) & \text{if } i_z = 0. \\ \vartheta(z + 1, \left(\beta_{j_z}^{i,j,z-1}(\mathbf{x}_{j_z}) \frac{\partial}{\partial x_{j_z}} d_{j_z}^{\pi_{j_z}}(x_{j_z}) \right) \xi, l) & \text{if } i_z = 1. \\ -\vartheta(z + 1, \sum_{k_z=1}^n d_{j_z}^{\pi_{j_z}}(x_{j_z}) \mathbf{g}_{\pi_{j_z}, k_z}(\mathbf{x}_{k_z}) \frac{\partial}{\partial x_{k_z}} \xi(\mathbf{x}), l) & \text{if } i_z = 2. \end{cases} \quad (4.5.4)$$

We take the base case to be level 1. As shown in the proof of Theorem 4.9, the root node has weighting functions $d_{j_1}^{\pi_{j_1}}$ which trivially satisfy the assumption. At an arbitrary level $r > 1$, all nodes are produced by one of three branches below a node at level $r - 1$. To prove the assumption, we must consider these three cases:

Branch 0 ($i_r = 0$): In this case, by assumption, and by the first term in Lemma 4.17, the j^{th} weighting function of the i^{th} level r node is

$$c^{i,j,r} = d_{j_r}^{\pi_{j_r}} \vartheta(1, 1, r - 1).$$

This is exactly of the form given in the recurrence for branch 0, and the assumption succeeds.

Branch 1 ($i_r = 1$): We have, by Lemma 4.17 and Theorem 4.9, the following weight-

ing function:

$$\begin{aligned}
c^{i,j,r} &= -\vartheta(1, 1, r-1) \left(\mathcal{L}_{\beta^{i,j,r-1}} d_{j_r}^{\pi_{j_r}} \right) \\
&= -\left\langle \partial d_{j_r}^{\pi_{j_r}}, \beta^{i,j,r-1} \right\rangle \vartheta(1, 1, r-1) \\
&= -\left(\sum_{k_r=1}^n \beta_{k_r}^{i,j,r-1} \frac{\partial}{\partial x_{k_r}} d_{j_r}^{\pi_{j_r}} \right) \vartheta(1, 1, r-1) \\
&= -\left(\beta_{j_r}^{i,j,r-1} \frac{\partial}{\partial x_{j_r}} d_{j_r}^{\pi_{j_r}} \right) \vartheta(1, 1, r-1).
\end{aligned}$$

since $\frac{\partial}{\partial x_{k_r}} d_{j_r}^{\pi_{j_r}}(\mathbf{x}) = 0$ if $k_r \neq j_r$. This proves the second case of the recursion.

Branch 2 ($i_r = 2$): Finally, by the last term of Lemma 4.17 the branch 2 weighting function is

$$\begin{aligned}
c^{i,j,r} &= d_{j_r}^{\pi_{j_r}} \left(\mathcal{L}_{\mathbf{g}_{j_r}} \vartheta(1, 1, r-1) \right) \\
&= d_{j_r}^{\pi_{j_r}} \langle \partial \vartheta(1, 1, r-1), \mathbf{g}_{j_r} \rangle \\
&= d_{j_r}^{\pi_{j_r}} \sum_{k_r=1}^n \mathbf{g}_{\pi_{j_r}, k_r} \frac{\partial}{\partial x_{k_r}} \vartheta(1, 1, r-1).
\end{aligned}$$

After rearranging terms, the final equality matches the recurrence for branch 2, satisfying the inductive assumption.

Finally, if $r \geq |\pi|$ we are at a leaf node and the induction terminates successfully for all levels of the evaluation tree.

Having characterized the recursive form of the weighting functions, we now derive a closed form for the recursion $\vartheta(1, 1, |\pi|)$. It is apparent from equation (4.5.4) that the recursion proceeds through the branchings from the root node to the leaf node i , at each step r multiplying by the previous result, and possibly applying a differential operator to it. This is essentially an iterative process. At each step of the recursion, $i_r \in \{0, 2\}$ contributes a function $d_{j_r}^{\pi_{j_r}}$, and $i_r = 1$ contributes $\frac{\partial}{\partial x_{k_r}} d_{j_r}^{\pi_{j_r}}$. We can summarize this contribution by applying the conditional differential operation

$\hat{D}_r^r d_{j_r}^{\pi_{j_r}}$, where

$$\hat{D}_v^u = \begin{cases} \frac{\partial}{\partial x_{j_v}} & \text{if } i_v = u. \\ \mathcal{I} & \text{otherwise.} \end{cases}$$

applying the derivative only if $i_r = 1$. The resulting term is then multiplied by 1 if the branch is 0, by $\mathbf{g}_{\pi_{j_r}, k_r}$ if the branch is 1, and by $\beta_{j_r}^{i_r, j_r, r-1}$ if the branch is 2. Observing that

$$\delta_{i_j}(u)(v-1)+1 = \begin{cases} v & \text{if } i_j = u. \\ 1 & \text{otherwise.} \end{cases}$$

we may represent this conditional multiplication as

$$\left[\delta_{i_j}(2) \left(\beta_{j_r}^{i_r, j_r, \pi-1} - 1 \right) + 1 \right] \left[\delta_{i_j}(1) \left(\mathbf{g}_{\pi_{j_r}, \pi-1} - 1 \right) + 1 \right] \hat{D}_r^r d_{j_r}^{\pi_{j_r}}. \quad (4.5.5)$$

When $i_j = 2$ on step j , we must take the derivative of the product of the preceeding $j-1$ terms. As we saw in the proof of Lemma 4.5, we may express this derivation as the summation

$$\sum_{l_1=1}^1 \sum_{l_2=1}^2 \cdots \sum_{l_{|\pi|}=1}^{|\pi|-1} \prod_{z=1}^{|\pi|-1} D_{|\pi|}^2 D_{|\pi|-1}^2 \cdots D_{z+1}^2 T_{i,j},$$

where the conditional operator D_v^u is defined as

$$D_v^u = \begin{cases} \frac{\partial}{\partial x_{j_v}} & \text{if } i_v = u \text{ and } z = t_v. \\ \mathcal{I} & \text{otherwise.} \end{cases}$$

and $T_{i,j}$ represents the term in equation (4.5.5). Finally, since each branch $i_r = 2$ introduces a summation over variable k_r , we may take advantage of the linearity of the differential operators to move equation (4.5.5) inside the summations. This results in

$$\sum_{k_1=1}^n \cdots \sum_{k_{|\pi|}=1}^n \sum_{l_1=1}^1 \sum_{l_2=1}^2 \cdots \sum_{l_{|\pi|}=1}^{|\pi|-1} \prod_{z=1}^{|\pi|-1} D_{|\pi|}^2 D_{|\pi|-1}^2 \cdots D_{z+1}^2 T_{i,j}. \quad (4.5.6)$$

Now, consider the leaf nodes of level $|\pi|$. The sign of each leaf node is determined by the change in sign that occurs as we recurse down the path from the root to the leaf. It is clear from recursion (4.5.4) that the sign flips each time $i_r = 1$. Therefore, the sign of the leaf node is positive if the number of ones in $i_1/\dots/i_{|\pi|}$ is even, and negative otherwise. Explicitly,

$$\eta_i = -1^{(|\pi| - \sum s=1^{|\pi|} (1-i_s))^2}.$$

Multiplying equation (4.5.6) by this sign function, and substituting equation (4.5.5) for $T_{i,j}$, produces the claimed weighting function expansion, and completes the proof. ■

4.6 Chapter Conclusions

In this chapter we have performed a reachability analysis of seven families of GNF networks. The primary motivation for our study has been the desire to adaptively learn and enforce general nonlinear manifold constraints in these networks. We have discovered that most families are non-holonomic, and that only the trivial manifold constraint is robustly learnable for these classes of network. On the other hand, the few families that are holonomic do have manifold constraints that can be robustly learned. However, these constraints are severely limited and unlikely to be of use in most application domains. We conclude that we must make modifications to the network model in order to be able to employ the manifold constraint paradigm.

Quite apart from our interest in the manifold constraint paradigm, our results provide a better understanding of the mechanisms that shape the dynamics of a broad class of continuous-time connectionist networks. In particular we have clearly exposed the different contributions that are made by the adaptive and non-adaptive terms of the neuron equations to the controllability properties of the networks. We conclude that the non-adaptive terms have a significant role to play in shaping controllability, and that the network designer can make use of our results to make a

sensible choice of these terms.

As mentioned earlier, the results presented in this chapter are currently being compiled into two papers [Gra98g, Gra98b] which summarize the results for open and closed GNF networks, and summarize the implications of these results for the manifold constraint paradigm and the selection of the non-adaptive terms. In [Gra98g] we apply our results for closed networks to give the explicit closed form expressions for general and classical GNF networks.

Chapter 5

The Observability Properties of GNF Networks

Any large computational system is invariably divided into interconnected computational modules. Biological nervous systems are no exception¹. Any study of the computational characteristics of such a system must take into account the degree to which the computational state of one module can influence the state of another. In the case where (at least some of) the modules are continuous time GNF networks, we need to understand the role that the network output function plays in the transmission and reconstruction of the activation state. The question of how much knowledge one network can obtain about the activation state of another through observation of its output state is the subject of this chapter.

In the sequel we will assume that a computational module consists of a GNF network which has been described in Chapter 2. In particular, the reader should observe that the details of their output fields are found in Section 2.2.1.

As a natural sequel to Chapter 4, in this chapter we begin a study of the

¹The divisions occur at many levels. For example, the CNS is divided into multiple computational organs (e.g. cortex, limbic structures, etc.), and these structures are divided into specialized computational networks (e.g. Brodmann areas in cortex). Further tessellation into a mosaic of columnar units is common, and even these are divided into layers of specialized networks.

observability properties of GNF networks. Specifically, we characterize a partition of a network's activation space into sets of states that produce identical output dynamics, and are therefore indistinguishable to an observer. In other words, we divide the activation space of the network into equivalence classes, where two states are equivalent ($\mathbf{x}_1 \sim \mathbf{x}_2$) if and only if a network produces exactly the same output space dynamics when initialized at these points, and given the same input signals. These equivalence classes, then, describe internal representations that are computationally equivalent to outside observers.

As we shall discover, these equivalence classes are constant dimensional nonlinear differential manifolds that locally foliate the neighbourhood of any given activation state. We refer to them as *observability manifolds*, and it is their tangent spaces, or *observability distributions*, that we will characterize in this chapter.

5.1 Contributions of the Chapter

Our main contribution is to establish a duality between the reachability distribution and observability distributions of open GNF networks. This is quite typical in control theoretic domains [Won79, Isi89, NvdS90], but we shall show that apart from such a 'mathematical' duality the two concepts are also 'syntactically' dual. In other words, the observability distribution characterizes a partition of a constraint manifold into equivalence classes. Each class is a submanifold of states that produce identical network computations from the perspective of an external observer.

Apart from our intrinsic interest in this duality for its own sake, the main appeal of our result is that it provides us with a tool that permits us to state observability results for general open heterosynaptic, homosynaptic and monosynaptic GNF networks. In addition, we are able to show that the observability distribution for closed versions of these networks is derived from linear combinations of co-vector fields that span the co-distribution annihilating the open network observability dis-

tributions. We characterize the function coefficients for these linear combinations.

The results of this chapter are presently being compiled in the paper [Gra98i].

5.2 GNF Observability is Dual to GNF Reachability

It is known [Won79, Isi89, NvdS90] that, at a conceptual level, the task of finding observability manifolds is mathematically dual to that of finding reachability manifolds. As we saw in Chapter 3, given an activation state $\mathbf{x}_0$, the set of equivalent states is a maximal integral submanifold with its tangent space equal to the largest distribution that is contained in the kernel of the output function (i.e. $\text{span}\{\partial h_1, \dots, \partial h_p\}^\perp$) and invariant to the GNF network fields $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$. We may state the dual problem as one of finding the smallest nonsingular involutive co-distribution containing $\text{span}\{\partial h_1, \dots, \partial h_p\}$ and co-invariant under the vector fields $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$, i.e. $\Omega_\bullet = \langle \mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m | \text{span}\{\partial h_1, \dots, \partial h_p\} \rangle$. The dual $\Omega_\bullet^\perp$ is a completely integrable distribution that foliates the state space into equivalence classes as desired.

The statement of the dual problem is analogous to the reachability problem described in Section 3.3, however, the definition of co-invariance is subtly different than that for invariance as introduced in Appendix B. The following definitions are derived from the discussion in [Isi89]:

Definition 5.1

The derivative of a co-vector field ω along a vector field τ is an operator that produces another co-vector field as follows

$$\mathcal{L}_\tau \omega = \tau^T \left[\frac{\partial \omega^T}{\partial \mathbf{x}} \right]^T + \omega \frac{\partial \tau}{\partial \mathbf{x}}.$$

Definition 5.2

A co-distribution Ω is co-invariant under vector field τ if it is closed under the derivative along τ , i.e.

$$\mathcal{L}_\tau \omega \in \Omega. \quad \text{for all } \omega \in \Omega.$$

The above definition of co-invariance is chosen so that, when a co-distribution Ω is co-invariant under τ , its dual $\Delta = \Omega^\perp$ is invariant under τ and visa-versa (see [Isi89] for a proof). Consequently, we are assured that $\Omega_\star^\perp$ is involutive, as required.

To clarify the relevance of $\Omega_\star^\perp$ to the problem of characterizing a network's observability manifolds, consider the coordinate change $\mathbf{z} = \Phi(x)$ that flattens this completely integrable distribution. The invariance of $\Omega_\star^\perp$ under the fields $\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m$, combined with the fact that

$$\partial h_i(\mathbf{z}) = \left[0, \dots, 0, \frac{\partial h_i(\mathbf{z})}{\partial z_{d+1}}, \dots, \frac{\partial h_i(\mathbf{z})}{\partial z_n} \right] \quad \text{for } 1 \leq i \leq p.$$

allows us to represent the system as follows

$$\begin{aligned} \dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{z}_2) + \sum_{j=1}^m \mathbf{g}_{1,j}(\mathbf{z}_1, \mathbf{z}_2) u_j \\ \dot{\mathbf{z}}_2 &= \mathbf{f}_2(\mathbf{z}_2) + \sum_{j=1}^m \mathbf{g}_{2,j}(\mathbf{z}_2) u_j \\ \mathbf{y} &= \mathbf{h}(\mathbf{z}_2). \end{aligned} \tag{5.2.1}$$

The network dynamics are partitioned into subdynamics on two canonical subspaces, and the output hides the dynamics of the first subspace. Therefore, two initial values with the same $\mathbf{z}_1$ coordinates will produce identical output dynamics. The linear subspace consisting of all such points is therefore an observability manifold, and its image in $\mathbf{x}$ coordinates is the maximal integral submanifold with tangent space equal to $\Omega_\star^\perp$.

5.3 Observability of Open GNF Networks with Output Summation Fields

Since we are interested in deriving observability results for GNF systems, the class of output fields that we choose to study is the axial summation fields defined in Section 2.2.1 of Chapter 2.

In the sequel our strategy will be to employ the recursion

$$\begin{aligned}\Omega_0 &= \text{span} \{ \partial h_1, \dots, \partial h_p \} \\ \Omega_k &= \Omega_{k-1} + \sum_{i=0}^{n+m} \text{span} \{ \mathcal{L}_{\mathbf{g}_i} \omega : \omega \in \Omega_{k-1} \}.\end{aligned}\quad (5.3.1)$$

introduced in Section 3.4 to compute $\Omega_{\star}^{\perp}$ on the local neighbourhood of any state $\mathbf{x}_0 \in \mathbb{R}^n$. We will also adopt the approach taken in Chapter 4 and analyze the observability of open and closed GNF networks separately.

Our goal in this section is to characterize the observability manifolds for a GNF network with output summation fields by describing the co-vector fields that span the co-distribution $\Omega_{\star}$. The dual distribution $\Omega_{\star}^{\perp}$ is the tangent space of the observability manifolds. We will make use of the definitions and notation for open GNF networks found in Section 4.2.4 without further comment.

Our first result establishes a structural duality between the spanning sets of $\Omega_{\star}$ and $\Delta_{\star}$ for open GNF networks. We begin by introducing a notational device that will help illuminate this connection.

Definition 5.3

The co-bracket is an operator which produces a vector field from a vector field τ and a co-vector field ω . It is similar in structure to the familiar Lie bracket, and it is defined as follows:

$$[\tau(\mathbf{x}) \cdot \omega(\mathbf{x})] = \frac{\partial \omega(\mathbf{x})^T}{\partial \mathbf{x}} \tau(\mathbf{x}) + \frac{\partial \tau(\mathbf{x})}{\partial \mathbf{x}} \omega(\mathbf{x})^T.$$

The following definition and technical lemma lead up to the main duality theorems.

Definition 5.4

A diagonal vector field τ or diagonal co-vector field ω is one with diagonal Jacobian.

i.e.

$$\frac{\partial \tau(\mathbf{x})}{\partial \mathbf{x}} = \text{diag} \begin{bmatrix} \frac{\partial \tau_1(x_1)}{\partial x_1} \\ \vdots \\ \frac{\partial \tau_n(x_n)}{\partial x_n} \end{bmatrix}, \quad \text{and} \quad \frac{\partial \omega(\mathbf{x})^T}{\partial \mathbf{x}} = \text{diag} \begin{bmatrix} \frac{\partial \omega_1(x_1)}{\partial x_1} \\ \vdots \\ \frac{\partial \omega_n(x_n)}{\partial x_n} \end{bmatrix}.$$

Lemma 5.1

Given an open GNF network with n neurons, the derivative of a co-vector field ω along any network field $\tau \in \{\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m\}$ is a transposed co-bracket, i.e.

$$\mathcal{L}_\tau \omega = [\tau, \omega]^T.$$

Proof: Recall the definition of the open GNF network fields (see Definition 4.1). We know that τ is a diagonal vector field because the i^{th} elementary function of any open network field depends only on the i^{th} coordinate. This symmetry implies that

$$\left[\frac{\partial \tau}{\partial \mathbf{x}} \right]^T = \frac{\partial \tau}{\partial \mathbf{x}}.$$

and, therefore, we have

$$\begin{aligned} \mathcal{L}_\tau \omega &= \left[\frac{\partial \omega^T}{\partial \mathbf{x}} \tau + \omega \left[\frac{\partial \tau}{\partial \mathbf{x}} \right]^T \right]^T \\ &= \left[\frac{\partial \omega^T}{\partial \mathbf{x}} \tau + \omega \frac{\partial \tau}{\partial \mathbf{x}} \right]^T \\ &= [\tau, \omega]^T \end{aligned}$$

as claimed. ■

Theorem 5.1 (Structural Duality Theorem)

Given a p -dimensional output field $\mathbf{h}(\mathbf{x}) \in \mathcal{F}_{\text{sum}}$, and an open GNF network with n neurons driven by m inputs, a repeated derivative of the gradient of the i^{th} output channel along a sequence of network fields given by the length d multi-index $\pi \in \Omega_d^{(0, n+m)}$ may be written as the transpose of a multi-co-bracket, i.e.

$$\mathcal{L}_{\mathbf{g}_{\pi_1}} \mathcal{L}_{\mathbf{g}_{\pi_2}} \cdots \mathcal{L}_{\mathbf{g}_{\pi_d}} \partial h_i = [\mathbf{g}_{\pi_1}, [\mathbf{g}_{\pi_2}, \cdots [\mathbf{g}_{\pi_d}, \partial h_i^T]]]^T. \quad (5.3.2)$$

Proof: We begin by proving that the transposed co-bracket of a diagonal vector field τ and a diagonal co-vector field ω is also a diagonal co-vector field. To see that this is true, observe that under these assumptions

$$\begin{aligned} [\tau, \omega]^T &= \left[\text{diag} \begin{bmatrix} \frac{\partial \omega_1(x_1)}{\partial x_1} \\ \vdots \\ \frac{\partial \omega_n(x_n)}{\partial x_n} \end{bmatrix} \tau + \text{diag} \begin{bmatrix} \frac{\partial \tau_1(x_1)}{\partial x_1} \\ \vdots \\ \frac{\partial \tau_n(x_n)}{\partial x_n} \end{bmatrix} \omega \right]^T \\ &= \begin{bmatrix} \frac{\partial \omega_1(x_1)}{\partial x_1} \tau_1(x_1) + \frac{\partial \tau_1(x_1)}{\partial x_1} \omega_1(x_1) \\ \vdots \\ \frac{\partial \omega_n(x_n)}{\partial x_n} \tau_n(x_n) + \frac{\partial \tau_n(x_n)}{\partial x_n} \omega_n(x_n) \end{bmatrix}. \end{aligned}$$

which is again a diagonal co-vector field.

The remainder of the proof proceeds by induction on the length d of π . We require that

1. equation (5.3.2) is true, and
2. that $\mathcal{L}_{\mathbf{g}_{\pi_2}} \cdots \mathcal{L}_{\mathbf{g}_{\pi_d}} \partial h_i^T$ is a diagonal co-vector field.

We take $d = 1$ as our base case. Lemma 5.1 applies in this case, so the first condition holds. As for the second condition, observe that the i^{th} output channel gradient is a diagonal co-vector field because

$$\begin{aligned} \partial h_i(\mathbf{x})^T &= \sum_{j=1}^n h_{j,i}(x_j) \\ &= \sum_{j=1}^n \left[\frac{\partial h_{j,i}(x_j)}{\partial x_1} \cdots \frac{\partial h_{j,i}(x_j)}{\partial x_n} \right]^T \\ &= \left[\frac{\partial h_{1,i}(x_1)}{\partial x_1} \cdots \frac{\partial h_{n,i}(x_n)}{\partial x_n} \right]^T. \end{aligned}$$

$[\mathbf{g}_{\pi_1}, \partial h_i]^T$ is therefore diagonal and the second assumption also holds.

For arbitrary $d > 2$, we have

$$\mathcal{L}_{\mathbf{g}_{\pi_1}} \Psi, \quad \text{where } \Psi = \left[\mathbf{g}_{\pi_2}, \cdots \left[\mathbf{g}_{\pi_d}, [\partial h_i]^T \right] \right].$$

By the inductive hypothesis Ψ is a diagonal co-vector field, and therefore, Lemma 5.1 tells us that condition 1 is true. Furthermore, equation (5.3) satisfies condition 2 because, $\mathbf{g}_{\pi_1}$ is a diagonal vector by Definition 2.3 and Ψ is a diagonal co-vector field. The induction succeeds and the original claim is proven. ■

Now that we have shown a duality between repeated derivatives and multi-co-brackets, we may use it to prove the following theorem which demonstrates that the co-vector fields that span Ω_* can be described as transposed multi-co-brackets. This is a form which is structurally dual to that of the vector fields that span Δ_* .

Theorem 5.2 (Observability/Reachability Theorem)

The d -dimensional observability distribution $\Omega_^\perp$ of an open GNF network with n neurons and m inputs and observed through a p channel output field $\mathbf{h} \in \mathcal{F}_{sum}$ is dual at each state $\mathbf{x}_0 \in \mathbb{R}^n$ to a co-distribution of the form*

$$\begin{aligned} \Omega_* &= \text{span} \left\{ \omega_\pi : \pi_{|\pi|} \in \{1, 2, \dots, p\} \text{ and } \pi_1 / \dots / \pi_{|\pi|-1} \in \hat{\Pi}_{d-1}^{(0, n-m)} \right\}, \quad \text{where} \\ \omega_\pi &= \begin{cases} \partial h_{\pi_1} & \text{if } |\pi| = 1 \\ \left[\mathbf{g}_{\pi_1} \cdot \left[\mathbf{g}_{\pi_2} \cdot \dots \cdot \left[\mathbf{g}_{\pi_{|\pi|-1}} \cdot \partial h_{\pi_{|\pi|}} \right] \right] \right] & \text{if } |\pi| > 1. \end{cases} \end{aligned}$$

Proof: We begin by proving that

$$\omega_\pi = \begin{cases} \partial h_{\pi_1} & \text{if } |\pi| = 1 \\ \mathcal{L}_{\mathbf{g}_{\pi_1}} \mathcal{L}_{\mathbf{g}_{\pi_2}} \dots \mathcal{L}_{\mathbf{g}_{\pi_{|\pi|-1}}} \mathbf{g}_{\partial h_{\pi_{|\pi|}}} & \text{if } |\pi| > 1. \end{cases} \quad (5.3.3)$$

The proof of the theorem can be tedious if performed from first principles. However, if we observe that there is an exact correspondence between the recursive algorithm (4.2.4) for computation of Δ_* and algorithm (5.3.1) for computation of Ω_* , the proof follows exactly as in the case of Lemma 4.1. Indeed, we need only substitute Ω_k for Δ_k and replace the Lie bracket with the derivative of a co-vector field along a vector field in the proof of Lemma 4.1 to obtain equation (5.3.3).

Our last step is to use Theorem 5.1 to express the repeated derivatives in equation (5.3.3) as the transpose of a multi-co-bracket. This satisfies the claim. ■

Remark 5.1 *The importance of Theorem 5.2 lies in the similarity between the multi-co-brackets spanning Ω_* and the multi-Lie brackets spanning Δ_* . The form of the co-bracket*

$$[\tau(\mathbf{x}) \cdot \omega(\mathbf{x})] = \frac{\partial \omega(\mathbf{x})^T}{\partial \mathbf{x}} \tau(\mathbf{x}) + \frac{\partial \tau(\mathbf{x})}{\partial \mathbf{x}} \omega(\mathbf{x})^T.$$

is identical to that of the Lie bracket

$$[\tau(\mathbf{x}) \cdot \omega(\mathbf{x})^T] = \frac{\partial \omega(\mathbf{x})^T}{\partial \mathbf{x}} \tau(\mathbf{x}) - \frac{\partial \tau(\mathbf{x})}{\partial \mathbf{x}} \omega(\mathbf{x})^T.$$

except for the sign of the second term. Of course, the expansion of the multi-co-brackets will be correspondingly similar to that of the multi-Lie brackets, taking into account this sign change. Therefore, all of the lemmas of Section 4.3.1 that derive a closed form description of the multi-Lie brackets spanning Δ_ in terms of the network weights have dual versions for Ω_* . This fortunate situation is, to some extent, unexpected as the definition of co-invariance does not lead to such a close correspondence in the general case. It is the symmetry of the GNF network field Jacobians and the Jacobians of the axial summation output gradients which allows this.*

For the reader's convenience we now state the dual observability lemmas. By virtue of Remark 5.1, the proofs of these lemmas are very similar to their duals in Section 4.3.1. We therefore limit our proofs to sketches that highlight the differences due to the above sign change.

Lemma 5.2

A multi-co-bracket may be written as the sum of Lie derivatives

$$[\tau_d \cdot [\tau_{d-1} \dots [\tau_2 \cdot \omega_1]]] = \mathcal{L}_{\tau_d} [\tau_{d-1} \dots [\tau_2 \cdot \omega_1]] + \mathcal{L}_{[\tau_{d-1} \dots [\tau_2 \cdot \omega_1]]} \tau_d$$

Proof: The proof of the lemma is identical to that of Lemma 4.2, accounting for the difference between the signs of terms in the co-bracket and the Lie bracket. ■

Lemma 5.3

A Lie bracket $\mathcal{L}_{\gamma}\tau_d$, with respect to a multi-co-bracket $\gamma = [\tau_{d-1} \cdots [\tau_2, \omega_1]]$, may be written as a difference of Lie derivatives

$$\mathcal{L}_{[\tau_{d-1}, \dots, [\tau_2, \omega_1]]}\tau_d = \mathcal{L}_{\tau_{d-1}}\mathcal{L}_{[\tau_{d-2}, \dots, [\tau_2, \omega_1]]}\tau_d + \mathcal{L}_{[\tau_{d-2}, \dots, [\tau_2, \omega_1]]}\mathcal{L}_{\tau_{d-1}}\tau_d.$$

Proof: The proof is again identical to that of Lemma 4.3, accounting for the sign difference between the co-bracket and Lie bracket. ■

Lemma 5.4

Each multi-co-bracket $\left[\tau_{\pi_1}, \left[\tau_{\pi_2}, \dots, \left[\tau_{\pi_{|\pi|-1}}, \tau_{\pi_{|\pi|}}\right]\right]\right]$, is the sum of repeating Lie derivatives of the form

$$\mu_{\pi} = \sum_{p \in \Phi(d)} \mathcal{L}_{\tau_{p_1}} \mathcal{L}_{\tau_{p_2}} \cdots \mathcal{L}_{\tau_{p_{|\pi|-1}}} \tau_{p_{|\pi|}}. \quad (5.3.4)$$

where $\tau_{\pi_i} \in \{\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{n-m}\}$ for $1 \leq i \leq |\pi| - 1$, $\tau_{\pi_{|\pi|}}$ is a co-vector field and $p = p_1/p_2/\cdots/p_{|\pi|}$ is a member of the set $\Phi(|\pi|)$ defined in Lemma 4.4.

Proof: The proof is the same as that for Lemma 4.4. The only difference is that the signs of the nodes in the evaluation tree (see Figure 4.1) are all positive rather than alternating because all of the terms in Lemmas 5.2 and 5.3 are positive. Consequently, the function $\sigma(p)$ of Lemma 4.4 that captures the alternating sign has constant value +1 and is absorbed into the sign of the repeated Lie brackets in summation (5.3.4). ■

In the theorems that follow from the above lemmas it is convenient to work with the gradients of the output channels $\partial h_i(\mathbf{x})$, $1 \leq i \leq p$, and the open GNF network fields in a uniform manner. In particular, we wish to rewrite the transposed output channel gradients in the same syntactic form as the open GNF network fields. To this end, we introduce the following notation that is in the same spirit as the ‘faux neuron’ introduced in Definition 4.2.

Definition 5.5

We consider the transposed gradients of the output channels $\partial h_i(\mathbf{x})^T$, $1 \leq i \leq p$, to be synaptic gains on inputs from additional faux neurons. We treat these virtual

neurons in exactly the same manner as the faux neuron introduced in Section 4.3. They are indexed in the range $n + m + 1$ to $n + m + p$ to distinguish them from the open GNF network fields.

We refer to the following associated vector fields as output channel fields. The following notation will be used in the remainder of this section unless otherwise stated. For $1 \leq i \leq p$ and $n + m + 1 \leq j \leq n + m + p$ we have

$$\begin{aligned} s_{j,i}^+(x_i) &= \frac{h_{j,i}(x_i)}{a_i(x_i)}, \quad s_{j,i}^-(x_i) = 0, \\ H_{i,j}^+ &= 1, \quad H_{i,j}^- = 0, \\ g_{j,i} &= a_i(x_i) \left(s_{j,i}^-(x_i) H_{i,j}^+ - s_{j,i}^+(x_i) H_{i,j}^- \right). \end{aligned} \quad (5.3.5)$$

Definition 4.11. Lemmas 4.5, 4.6 and 4.7, and Corollary 4.1 may now be applied to the observability analysis simply by increasing the range of open GNF network fields to include the output channel fields with indices $n + m + 1$ to $n + m + p$.

We now present three theorems that characterize the observability distribution for open GNF nets with output summation fields.

Theorem 5.3

An open monosynaptic GNF network, of n neurons with m inputs and valency $s \in \{0, 1\}$, has a d -dimensional reachability distribution

$$\Omega_{\pi}^+ = \text{span} \left\{ \mathbf{D}_{\pi}(\mathbf{y}) \mathbf{v}_{\pi} : \pi_{|\pi|} \in \{1, \dots, p\} \text{ and } \pi_1 / \dots / \pi_{|\pi|-1} \in \Omega_d^{(0, n+m)} \right\}^T.$$

where

$$1. \mathbf{D}_{\pi}(\mathbf{y}) =$$

$$\sum_{p \in \Phi(|\pi|)} \sum_{i_1=1}^1 \sum_{i_2=1}^2 \dots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \text{diag} \left[\left(\bigodot_{j \in \pi^0} \gamma_0^0(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \odot \left(\bigodot_{j \in \pi^0} \gamma_j^s(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \right]$$

$$2. \mathbf{v}_\pi = \bigodot_{j \in \overline{\pi^0}} \mathbf{W}_j^s T.$$

Proof: The proof proceeds exactly as in Theorem 4.1. except that the function $\sigma(p)$ has constant value +1 as in Lemma 5.4. ■

Theorem 5.4

An open homosynaptic GNF network of n neurons with m inputs and p channel output summation field has a d -dimensional reachability distribution

$$\Omega_\pi^+ = \text{span} \left\{ \mathbf{D}_\pi(\mathbf{y}) \mathbf{v}_\pi : \pi_{|\pi|} \in \{1, \dots, p\} \text{ and } \pi_1 / \dots / \pi_{|\pi|-1} \in \Omega_d^{(0, n+m)} \right\}^T.$$

where

1. $q = q_1 / \dots / q_{|\pi|}$ is the homosynaptic valence pattern for multi-index $p \in \Phi(|\pi|)$.

2. $\mathbf{D}_{\pi, q}(\mathbf{y}) =$

$$\sum_{p \in \Phi(|\pi|)} \sum_{i_1=1}^1 \sum_{i_2=1}^2 \dots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \text{diag} \left[\left(\bigodot_{j \in \overline{\pi^0}} \gamma_0^0(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \odot \left(\bigodot_{j \in \overline{\pi^0}} \gamma_j^{q_j}(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \right]. \quad \text{and}$$

$$3. \mathbf{v}_\pi = \bigodot_{j \in \overline{\pi^0}} \mathbf{W}_j^{q_j} T.$$

Proof: The proof is identical to that of Theorem 4.2. ■

Theorem 5.5

An open heterosynaptic GNF network, of n neurons with m inputs, has a d -dimensional reachability distribution

$$\Delta_\pi = \text{span} \left\{ \sum_{q_1=0}^1 \dots \sum_{q_{|\pi|-1}=0}^1 \sum_{q_{|\pi|}=0}^1 \mathbf{D}_{\pi, q}(\mathbf{y}) \mathbf{v}_{\pi, q} : \pi_{|\pi|} \in \{1, \dots, p\} \text{ and } \pi_1 / \dots / \pi_{|\pi|-1} \in \Omega_d^{(0, n+m)} \right\}^T.$$

where

1. q is the multi-index containing the summation variables $q_1/\cdots/q_{|\pi|}$.

2. $\mathbf{D}_\pi(\mathbf{y}) =$

$$\sum_{p \in \Phi(|\pi|)} \sum_{i_1=1}^1 \sum_{i_2=1}^2 \cdots \sum_{i_{|\pi|-1}=1}^{|\pi|-1} \text{diag} \left[\eta(q) \left(\bigodot_{j \in \pi^0} \gamma_0^0(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \odot \left(\bigodot_{j \in \pi^0} \gamma_j^{q_j}(\mathbf{y})^{(\sum_{k=j}^{|\pi|-1} \delta_j(i_k))} \right) \right].$$

where

$$\eta(q) = \prod_{j=1}^{|\pi|} (1 - 2q_j).$$

3. $\mathbf{v}_\pi = \bigodot_{j \in \pi^0} \mathbf{W}_j^s T$.

All other definitions are as in Theorem 4.2.

Proof: The proof is identical to that of Theorem 4.3. ■

5.4 Observability of Closed GNF Networks with Output Summation Fields

The characterization of $\Omega_\star^+$ for closed networks also benefits from the Structural Duality Theorem 5.1. We will freely use the definitions of sections 4.5 and 5.3 without further comment.

We begin by establishing the equivalent of Lemma 4.17, in which we show that a co-bracket of a weighted combination of diagonal vector fields and a weighted combination of diagonal co-vector fields may be written as a weighted sum of co-brackets and vector fields. This result will allow us to express the closed network spanning set of $\Omega_\star$ as a dual of $\Delta_\star$, just as we did for open GNF networks.

Lemma 5.5

$$\left[\sum_{k_1=1}^{n_1} c_{k_1} \mathbf{f}_{k_1} \cdot \sum_{k_2=1}^{n_2} c_{k_2} \omega_{k_2} \right] = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} c_{k_2} [\mathbf{f}_{k_1} \cdot \omega_{k_2}^T] + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_2} \left(\mathcal{L}_{\mathbf{f}_{k_1}^T} c_{k_1} \right) \omega_{k_2}^T + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} \left(\mathcal{L}_{\omega_{k_2}^T} c_{k_2} \right) \mathbf{f}_{k_1}.$$

Proof: The proof proceeds by straightforward algebra as in Lemma 5.5.

$$\begin{aligned} & \left[\sum_{k_1=1}^{n_1} c_{k_1} \mathbf{f}_{k_1} \cdot \sum_{k_2=1}^{n_2} c_{k_2} \omega_{k_2} \right] \\ &= \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \left(\frac{\partial c_{k_2}}{\partial \mathbf{x}} \omega_{k_2}^T c_{k_1} \mathbf{f}_{k_1} + c_{k_1} c_{k_2} \frac{\partial \omega_{k_2}^T}{\partial \mathbf{x}} \mathbf{f}_{k_1} \right) \\ & \quad + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \left(\left[\frac{\partial c_{k_1}}{\partial \mathbf{x}} \mathbf{f}_{k_1} \right]^T c_{k_2} \omega_{k_2}^T + c_{k_2} c_{k_1} \frac{\partial \mathbf{f}_{k_1}}{\partial \mathbf{x}} \omega_{k_2}^T \right) \end{aligned}$$

and the symmetry of the diagonal Jacobians permits us to write

$$\begin{aligned} &= \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} (\partial c_{k_2} \omega_{k_2}^T) \mathbf{f}_{k_1} + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_2} (\partial c_{k_1} \mathbf{f}_{k_1}) \omega_{k_2}^T + c_{k_1} c_{k_2} \left(\frac{\partial c_{k_2}}{\partial \mathbf{x}} \omega_{k_2}^T + \frac{\partial \omega_{k_2}^T}{\partial \mathbf{x}} \mathbf{f}_{k_1} \right) \\ &= \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} c_{k_2} [\mathbf{f}_{k_1} \cdot \omega_{k_2}^T] + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_2} \left(\mathcal{L}_{\mathbf{f}_{k_1}^T} c_{k_1} \right) \omega_{k_2}^T + \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} c_{k_1} \left(\mathcal{L}_{\omega_{k_2}^T} c_{k_2} \right) \mathbf{f}_{k_1}. \end{aligned}$$

This completes the proof. ■

Notice that the only difference between the sum in Lemma 4.17 and that in Lemma 5.5 is the change in sign of the second term. Therefore, we can use Lemma 5.5 to easily establish theorems for closed GNF networks that are dual to those in section Section 4.5. In each case the proof proceeds exactly as it did for the reachability version of the theorem, except for the change in sign in Lemma 5.5.

Theorem 5.6

Each multi-co-bracket spanning a closed GNF network Ω_ with multi-index π s.t. $\pi_1 \in \{1, \dots, p\}$, and $\pi_{|\pi|}/\dots/\pi_2 \in \hat{\Pi}_d^{(0,m)}$ and output field $\mathbf{h}(\mathbf{x}) \in \mathcal{F}_{\text{sum}}$ is a weighted*

sum of multi-Lie brackets spanning the Ω_* of the same GNF network with open connections, i.e.

$$\begin{aligned} & \left[\dot{\mathbf{g}}_{\pi_{|\pi|}} \cdot \left[\dot{\mathbf{g}}_{\pi_{|\pi|}-1} \cdots \left[\dot{\mathbf{g}}_{\pi_2} \cdot \dot{\mathbf{g}}_{\pi_1} \right] \right] \right] (\mathbf{x}) = \\ & \sum_{i_1=2}^2 \sum_{i_2=0}^2 \cdots \sum_{i_1 \pi_1=0}^2 \sum_{j_1=p_{\pi_1}}^{q_{\pi_1}} \cdots \sum_{j_1 \pi_1=p_1 \pi_1}^{q_1 \pi_1} c^{i,J}(\mathbf{x}) \beta^{i,J}(\mathbf{x}). \end{aligned}$$

All functions definitions are as in Theorem 4.9.

Proof: The proof is identical to that of Theorem 4.9. ■

And finally, the description of the weighting functions are defined analogously to those in Theorem 5.7.

Theorem 5.7

The scalar functions that weight the sum of open network multi-co-brackets in a closed network multi-co-bracket are of the form

$$\begin{aligned} c^{i,J} = & \sum_{k_1=1}^n \cdots \sum_{k_1 \pi_1=1}^n \sum_{l_1=1}^1 \sum_{l_2=1}^2 \cdots \sum_{l_{|\pi|}=1}^{|\pi|} \\ & \prod_{z=1}^{|\pi|-1} \eta_l D_{\pi_z}^2 D_{\pi_{|\pi|-1}}^2 \cdots D_{z+1}^2 \\ & \left(\left[\delta_{i_j}(2) \left(\beta^{i,J,|\pi|-1} - 1 \right) + 1 \right] \left[\delta_{i_j}(1) \left(\mathbf{g}_{\pi_{j_1 \pi_1-1}} - 1 \right) + 1 \right] \hat{D}_z^1 d_z^{\pi_z} \right). \end{aligned}$$

All definitions are as in Theorem 4.10 and Theorem 4.9.

Proof: The proof is identical to that of Theorem 4.10. ■

Applying the descriptions of the open GNF network spanning vectors obtained in the last section to Theorems 5.6 and 5.7 yields the spanning set of $\Omega_*^\perp$ for closed GNF networks as desired.

5.5 Chapter Conclusions

In this chapter we have performed an observability analysis of general GNF networks. We have succeeded in characterizing the co-distribution dual to the observability distribution for these networks. This is the usual method of describing the observability manifolds of nonlinear systems. These results have been expressed for general heterosynaptic, homosynaptic and monosynaptic networks, and therefore also apply to the classical continuous-time shunting inhibition, BAM, Hopfield/Tank, BSB and McCulloch-Pitts networks. The observability distribution induces a partition of the reachability manifolds into sets of activation states that are computationally equivalent in the sense that, starting from any of these states, the network will produce exactly the same outputs for a given input signal.

In deriving these results, we have introduced an operator, the co-bracket, that establishes a duality between the equations that characterize the observability co-distribution and the reachability distribution for general GNF networks. This is a syntactic duality (as opposed to a mathematical duality) that permits us to directly apply the results of the last chapter to the observability analysis.

We have discovered that the co-vector fields that span the observability co-distribution for closed GNF networks are linear combinations of the co-vector fields that span the observability distribution for open GNF networks. Using these we have provided closed form expressions for the coefficient functions on these linear combinations.

The results of this chapter are presently being compiled in the paper [Gra98i].

Chapter 6

Enforcing Manifold Constraints by Additive Compensation

The implications of Chapters 4 and 5 for the manifold constraint paradigm are essentially negative in character. We have discovered that a very broad class of continuous-time connectionist models (the GNF networks) cannot robustly constrain their dynamics to a differentiable sub-manifold of activation space if the network is driven by time-varying inputs. However, we have argued in Chapter 1 that the ability to adaptively learn such constraints would be a useful tool for the network designer. We are naturally led to ask, “Are there modifications that can be made to the GNF network model that will permit robust learning and enforcement of manifold constraints?” In this chapter we begin a study that answers this question in the affirmative. In particular, we introduce two holonomic hybrid connectionist/control systems and apply them to the task of learning and enforcing *linear* constraint manifolds. In subsequent chapters we will extend these results to include very general classes of nonlinear manifold constraints.

There are at least two ways, other than direct modification of the network weights¹, that the dynamic behaviour of a GNF network can be modified. The first is

¹We will defer the design of a layered, recurrent and holonomic network that supports unsuper-

to augment the neuron equations with a term, $\mathbf{v}(\mathbf{x})$, that compensates for dynamical components that direct the network's state space trajectory away from the desired constraint manifold. The resulting hybrid compensator/network is of the form

$$\begin{aligned} \dot{\mathbf{x}} = & \mathbf{a}(\mathbf{x}) \left[\mathbf{b}(\mathbf{x}) + \sum_{j=1}^n \left(\mathbf{s}_j^+(\mathbf{x}) \mathbf{H}_j^+ - \mathbf{s}_j^-(\mathbf{x}) \mathbf{H}_j^- \right) d_j(x_j) \right. \\ & \left. + \sum_{k=1}^n \left(\mathbf{s}_{n+k}^+(\mathbf{x}) \mathbf{J}_k^+ - \mathbf{s}_{n+k}^-(\mathbf{x}) \mathbf{J}_k^- \right) u_k \right] \\ & + \mathbf{v}(\mathbf{x}). \end{aligned}$$

In this chapter we provide a closed form for $\mathbf{v}(\mathbf{x})$ that confines the dynamics of the compensated system to any desired nonlinear constraint manifold. In doing so, however, we must relax the usual connectionist assumption that the system dynamics are due to strictly local computations at each neuron. We illustrate our results by compensating a class of GNF networks that performs continuous-time back propagation.

A second approach is to design a feedback controller that applies input signals to steer the network into the desired region of activation space. These inputs compensate for unwanted dynamical components in much the same way that the compensating terms do, however, the controller is less invasive since it does not require modifications to the network equations. On the other hand, the degree of control that can be exerted is limited by the terms that connect the inputs to the network dynamics. We discuss this possibility as a fruitful area for future research in Chapter 9.

We are also concerned with the issue of selectively applying manifold constraints to the dynamics associated with the activations of a subset of neurons.

vised learning of manifold constraints by direct modification of the network weights until Chapter 8.

6.1 Contributions of the Chapter

In Sections 6.4 and 6.4.1 we introduce two manifold constraints that permit us to place constraints on the transient dynamics of a GNF network with respect to submanifolds of the activation space. We also introduce definitions that permit us to apply multiple constraints to the sub-dynamics associated with non-overlapping subsets of neurons that we refer to as cell assemblies in Section 6.2.

In Section 6.4 we provide results for the first of these constraints — *modular invariance* of a distribution under a GNF network vector field. We provide existence proofs for additive compensators that enforce modular invariance for any choice of completely integrable distribution and GNF network. Since our proofs are constructive, we also show how such a compensator can, in principle, be found for any set of smooth GNF neuron equations.

In Section 6.4.2 we provide results for the existence of compensators that enforce *modular reachability* constraints. Once again, our proof is constructive and we can, in principle, find the compensator equation for any smooth GNF network fields.

We illustrate our results in Section 6.5.2, by showing how the equations of a continuous time back propagation network may be augmented to enforce linear reachability constraints on the run-time dynamics of modular assemblies. Furthermore, we show how these reachability constraints can be learned by incorporating a version of Kohonen’s novelty filter into the neuron equations. We also show that the learning of associative memories (fixed points) can be performed simultaneously with the acquisition of reachability constraints.

Finally, we conclude with a discussion of the results and chapter conclusions in Section 6.6.

6.2 Modularity of Manifold Constraints

The neurons in a GNF network can be partitioned into subsets in a variety of ways. One partition that has had a significant role to play in the neural network literature is that between visible and hidden neurons, as described in the following definition:

Definition 6.1

The visible neuron set, $\mathcal{N}_v$, contains the indices of those neurons that are driven by at least one external input, i.e.

$$\forall i \in \mathcal{N}_v \exists j \text{ such that } 1 \leq j \leq n \text{ and } J_{i,j} \neq 0.$$

The hidden neuron set, $\mathcal{N}_h$, consists of the remaining neuron indices, such that

$$\forall i \in \mathcal{N}_h J_{i,j} = 0 \text{ for } 1 \leq j \leq n.$$

Consequently, $\mathcal{N}_h \cup \mathcal{N}_v = \mathcal{N}$.

Another type of partition that has had an important role to play in connectionism is that of the cellular assembly. From the early days of Hebb's work [Heb49] to the more modern studies of mixtures of experts [JJNH91], the idea of decomposing a large network into smaller computational modules has proven useful. Such divisions allow us to compartmentalize computational responsibilities in a network — e.g. it may be that only a subset of neurons in a large network are concerned with the representation of an item, or provide output signals that control an external plant — make use of divide-and-conquer strategies, and bring multiple specialized agents to bear on a given problem.

In keeping with these ideas, we want to partition a network into cellular assemblies, and then apply manifold constraints to the sub-dynamics of the neurons in each assembly in a noninteracting manner, and without affecting the overall dynamics of the network. The following definitions provide terms and notation for the application of manifold constraints to individual assemblies.

Definition 6.2

Assembly k is a subset of r neurons with index set $\mathcal{N}_k$, where

$$\mathcal{N}_k = \text{span} \{i_{k,1}, i_{k,2}, \dots, i_{k,r}\} \subseteq \text{span} \{1, 2, \dots, n\}, \quad \text{and } i_{k,1} \leq i_{k,2} \leq \dots \leq i_{k,r}.$$

In the interest of simplicity we will identify the assembly with its index set and speak of ‘assembly $\mathcal{N}_k$ ’.

We will assume that, while all assemblies in a GNF network are non-overlapping, not all of the neurons in a network need belong to an assembly. Therefore, given q assemblies, we define the unconstrained set as

$$\mathcal{N}_u = \mathcal{N} - \bigcup_{j=1}^q \mathcal{N}_j.$$

For our purposes, an assembly is formed in order to apply a manifold constraint to its dynamics as reflected in the following definitions.

Definition 6.3

The n neurons of a network provide a natural canonical basis on $\mathbb{R}^n$, where the j^{th} neuron is associated with orthonormal basis vector $\mathbf{e}_j$. The dynamics of assembly $\mathcal{N}_k$ occur on subspace

$$\mathbf{S}_k = \text{span} \{ \mathbf{e}_j : j \in \mathcal{N}_k \}.$$

A constraint on the dynamics of assembly $\mathcal{N}_k$ represents a constraint on the overall network dynamics in this subspace. We will therefore refer to $\mathbf{S}_k$ as the k^{th} constraint space. The orthogonal space

$$\mathbf{S}_u = \left(\sum_{j=1}^q \mathbf{S}_j \right)^{\perp}$$

is the unconstrained space.

Definition 6.4

In order to distinguish the coordinates of the k^{th} constraint space from the rest, we partition the state space coordinates into complementary sub-coordinates. For

example, given q assemblies we have

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{c_1} \\ \vdots \\ \mathbf{x}_{c_q} \\ \mathbf{x}_u \end{bmatrix}, \quad \text{where} \quad \mathbf{x}_{c_k} = \begin{bmatrix} x_{i_k,1} \\ \vdots \\ x_{i_k,r} \end{bmatrix}.$$

In the event that there is only one assembly under consideration, we will simplify the notation slightly, referring to the assembly as $\mathcal{N}_c$, and its constraint space as S_c , with coordinates $\mathbf{x}_c$.

6.3 Modular Linearizing Transforms

It greatly simplifies the analysis of modular manifold constraints if we are able to apply a change of coordinates that ‘linearizes’ the chosen manifold in the designated sub-space while leaving the complementary space unmodified. In the following technical lemma we see that, if the manifold is of constant dimension, we can always find a nonlinear change of coordinates that achieves this modular linearization. Specifically, we show that there exists a change of coordinates smoothly deforming a constraint subspace $S_k = \text{span}\{\mathbf{e}_{i_{k,1}}, \dots, \mathbf{e}_{i_{k,r}}\}$ such that a constraint manifold $\mathcal{M}$ is flattened and aligned with a canonical d -dimensional subspace, while the unconstrained subspace S_u is left unmodified.

Lemma 6.1

For any choice of r -dimensional subspace

$$S_c = \text{span}\{\mathbf{e}_i : i \in \mathcal{N}_c\} \subseteq \mathbb{R}^n,$$

d -dimensional differentiable manifold $\mathcal{M} \subseteq S_c$, and $\mathbf{x}_0 \in \mathcal{M}$, there exists a local

nonlinear change of coordinates Φ defined on a neighbourhood $\mathcal{U}$ of $\mathbf{x}_0$ such that

$$\mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} z_1 \\ \vdots \\ z_d \\ 0 \end{bmatrix}, \quad \forall \mathbf{x} \in \mathcal{M}.$$

Proof: Choose any distribution, Δ , that is a completion of the distribution $T_{\mathbf{x}}\mathcal{M}|_{\mathbf{x} \in \mathcal{U}}$, and that is locally completely integrable on some neighbourhood $\mathcal{U}$ of $\mathbf{x}_0$. That such a distribution exists is a consequence of the differentiability of $\mathcal{M}$: differentiability of a manifold implies the existence of a chart $\{\mathcal{U}, \mathbf{z} = \varphi(\mathbf{p})\}$, such that $\mathcal{M} = \varphi^{-1}(\text{span}\{\mathbf{e}_1, \dots, \mathbf{e}_d\})$. If we choose

$$\Delta(\mathbf{p}) = \underbrace{\frac{\partial \varphi^{-1}}{\partial \mathbf{z}}}_{\mathbf{z} = \varphi(\mathbf{p})} \text{span}\{\mathbf{e}_1, \dots, \mathbf{e}_d\},$$

then $\mathcal{U}$ is foliated by constant dimensional manifolds $\mathcal{M}_i$ with $T\mathcal{M}_i \subseteq \Delta$, and $T\mathcal{M} \subseteq \Delta$. In general, other distributions might also exist with the desired properties over other, ‘larger’, neighbourhoods. We also define $\Delta^\perp(\mathbf{x})$, the distribution that is the orthogonal complement of $\Delta(\mathbf{x})$ at every $\mathbf{x} \in \mathbb{R}^n$. These complementary distributions can be used to decompose any vector field τ into the sum of two complementary fields, i.e. $\tau(\mathbf{x}) = \tau_\Delta(\mathbf{x}) + \tau_{\Delta^\perp}(\mathbf{x})$, where τ_Δ is the projection of τ onto Δ along $\Delta^\perp$, and $\tau_{\Delta^\perp}$ is the projection of τ onto $\Delta^\perp$ along Δ .

As Δ is completely integrable, Frobenius’ theorem (see Section B.11) tells us that there exists a diffeomorphism, $\mathbf{z}_c = \Phi_c(\mathbf{x}_c)$, on S_c such that $\Phi(\mathbf{x}_0) = \mathbf{0}$ and $\Phi(\mathcal{M}) = \text{span}\{\mathbf{e}_1, \dots, \mathbf{e}_d\}$. Using this diffeomorphism we may construct the nonlinear coordinate transformation with the desired properties as follows:

$$\mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} \mathbf{z}_c \\ \mathbf{z}_u \end{bmatrix} = \begin{bmatrix} \Phi_c(\mathbf{x}_c) \\ \mathbf{x}_u \end{bmatrix}.$$

and its inverse

$$\mathbf{x} = \Phi^{-1}(\mathbf{z}) = \begin{bmatrix} \mathbf{x}_c \\ \mathbf{x}_u \end{bmatrix} = \begin{bmatrix} \Phi_c^{-1}(\mathbf{z}_c) \\ \mathbf{z}_u \end{bmatrix}.$$

Note that, the constructive nature of Frobenius' theorem allows us to, in principle, construct Φ . ■

This linearization of the manifold finds application in the following theorems, where we choose the constraint space associated with a cell assembly as the subspace in Lemma 6.1.

6.4 Modular Invariance

The first type of manifold constraint that we introduce is modular invariance of a distribution under a GNF network vector field. This form of constraint is interesting in its own right, and it is also of central importance when enforcing modular reachability constraints (introduced in the next section).

A distribution Δ is said to be *invariant under vector field $\mathbf{v}$* if it is closed under the Lie bracket product with $\mathbf{v}$, i.e. ,

$$[\mathbf{v}, \tau] \in \Delta, \quad \forall \tau \in \Delta.$$

We extend this concept for application to cell assembly dynamics as follows:

Definition 6.5

A *completely integrable distribution Δ* is modularly invariant under an n neuron GNF network field τ for cell assembly $\mathcal{N}_k$ if

1. $\Delta(\mathbf{x}_{c_k})$ is defined on constraint space S_k , and
2. Δ is locally invariant on S_k , i.e.

$$\left[\tau(\mathbf{x}), \begin{bmatrix} \theta(\mathbf{x}_c) \\ \mathbf{0} \end{bmatrix} \right] \subseteq \Delta(\mathbf{x}_c) + (\mathbb{R}^n - S_k) \quad , \quad \forall \theta \in \Delta.$$

where

$$\tau \in \text{span}\{\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m\}, \quad \text{and} \quad \tau_{c_k}(\mathbf{x}) = \begin{bmatrix} \tau_{i_{k,1}}(x_{i_{k,1}}) \\ \vdots \\ \tau_{i_{k,r}}(x_{i_{k,r}}) \end{bmatrix}.$$

This modularity of invariance is particularly important when the network is to be partitioned into cellular assemblies, and we wish to influence the sub-dynamics of each assembly independently.

To better understand the intuitive meaning of modular invariance under a GNF network, consider the case where the assembly consists of the entire network, i.e. $\mathcal{N}_c = \mathcal{N}$. As we see in the following theorem of Isidori (which we reproduce here without proof for the reader's convenience) invariance allows us to put a vector field into a simplified form.

Theorem 6.1 (Isidori, Lemma 6.3)

Given a d -dimensional completely integrable distribution Δ (defined on $\mathbb{R}^n$) which is invariant under vector field $\mathbf{v}$, then at each point $\mathbf{x}_0$ there exists a neighbourhood $\mathcal{U}$, and a coordinate transformation $\mathbf{z} = \Phi(\mathbf{x})$ on which $\mathbf{v}$ has the form

$$\mathbf{v}(\Phi(\mathbf{x})) = \begin{bmatrix} \hat{v}_1(z_1, \dots, z_d, z_{d+1}, \dots, z_n) \\ \vdots \\ \hat{v}_d(z_1, \dots, z_d, z_{d+1}, \dots, z_n) \\ \hat{v}_{d+1}(z_{d+1}, \dots, z_n) \\ \vdots \\ \hat{v}_n(z_{d+1}, \dots, z_n) \end{bmatrix}$$

Proof: See [Isi89], Section 1.6. ■

Therefore, if a distribution is invariant to *all* of a GNF network's fields, we

can find a change of coordinates such that the network has the form

$$\begin{aligned}\dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{z}_2) + \sum_{j=1}^m \mathbf{g}_{1,j}(\mathbf{z}_1, \mathbf{z}_2) u_j \\ \dot{\mathbf{z}}_2 &= \mathbf{f}_2(\mathbf{z}_2) + \sum_{j=1}^m \mathbf{g}_{2,j}(\mathbf{z}_2) u_j.\end{aligned}$$

In $\mathbf{z}$ coordinates the network is seen to have two dynamical components. In the $\mathbf{z}_1$ component is dependent on the value of both $\mathbf{z}_1$ and $\mathbf{z}_2$, but the $\mathbf{z}_2$ component is not dependent on $\mathbf{z}_1$. Since this is the case, any two initial $\mathbf{z}$ coordinates with the same $\mathbf{z}_2$ value will produce exactly the same $\mathbf{z}_2$ sub-dynamics (although their $\mathbf{z}_1$ sub-dynamics will differ). Within the neighbourhood $\mathcal{V} = \Phi(\mathcal{U})$ on which Φ is defined, the local dynamics of the network transport d -dimensional hyperplanes along an orthogonal $n - d$ -dimensional trajectory. These hyperplanes are of course the original leaves of the foliation induced by Δ , so in $\mathbf{x}$ coordinates the network dynamics transport nonlinear d -dimensional differentiable manifolds through state space. See Figure 6.1 for an illustration of the network dynamics.

In the event that the controlled assembly is a proper subset of the network neurons, then only the assembly sub-dynamics are constrained to behave this way.

6.4.1 Modular Reachability

Control of the transient run-time dynamics of a GNF network can often be viewed in terms of constraining the reachability of the network to a particular manifold of state space. For example, if the outputs of a network are used to control a chemical process, there may exist a manifold of ‘safe’ states that represent an acceptable range of control signals. In this case we want to constrain the evolution of the network dynamics to lie in this safety zone. Alternatively, the network states may encode symbolic data structures [Smo91] which are related by continuous transformations. Such a set forms a manifold in the network’s state space. Forcing the network to evolve on this manifold is equivalent to a guarantee that it will produce

Figure 6.1: Dynamics of Modular Invariance

legal sequences of transformations leading from the currently represented structure to an associatively recalled structure. Many other interpretations of the network state space, in fields as diverse as speech synthesis and robotics, also lead to manifolds that are important trajectory constraints. We shall refer to a constraint manifold (on the constraint space associated with assembly $\mathcal{N}_k$) as a *modular reachability constraint*.

One may think of reachability constraints as computational invariants that guarantee the correctness of a cell assembly's computational trajectories even under the influence of external inputs and inputs from other assemblies. Alternatively, we may consider reachability constraints as nonlinear 'traps' that capture the dynamics of a network, and allow a designer to 'sculpt' the dynamics of a GNF network to fit particular interpretations of the state space.

6.4.2 Enforcing Modular Invariance with Additive Compensation

We now present a lemma that tells us that the neuron equations of a GNF network can be modified to make any well-behaved distribution modularly invariant under the network vector fields. Specifically, we derive the form of a compensating term that can be added to the network evolution equations to induce modular invariance within a cellular assembly without affecting the dynamical properties of the rest of the network.

Our first lemma is a constructive existence proof for the compensating term that makes a distribution modularly invariant to one of the GNF network fields.

$$\begin{aligned}\mathbf{f} &= \mathbf{a}\mathbf{b} + \sum_{j=1}^n \mathbf{a} \left(\mathbf{s}_j^+ \mathbf{H}_j^+ - \mathbf{s}_j^- \mathbf{H}_j^- \right), \\ \mathbf{g}_k &= \mathbf{a} \left(\mathbf{s}_{n-k}^+ \mathbf{J}_k^+ - \mathbf{s}_{n-k}^- \mathbf{J}_k^- \right).\end{aligned}$$

To simplify the notation in this section and the next, we will assume that

$$\mathcal{N}_c = \{1, 2, \dots, r\}. \quad (6.4.1)$$

This assumption can be made without loss of generality, because the neuron indices can always be permuted to ensure that it is true.

Theorem 6.2

Given a d -dimensional completely integrable distribution $\hat{\Delta}$ defined on $\mathbb{R}^r$, a heterogeneous GNF network of n neurons with m inputs, and any choice of r neurons with constraint set $\mathcal{N}_c = \text{span}\{i_1, \dots, i_r\}$, there exists a compensating term $\mathbf{v}_0(\mathbf{x})$ that makes Δ modularly invariant under network vector field $\tau \in \{\mathbf{f}, \mathbf{g}_1, \dots, \mathbf{g}_m\}$.

Proof: Let $\Delta(\mathbf{x}) = \hat{\Delta}(\mathbf{x}_c) + S_u$ be the $d + (n - r)$ -dimensional extension of d -dimensional distribution $\hat{\Delta}(\mathbf{x}_c)$ to $\mathbb{R}^n$. Recall from Definition 6.5 that modular invariance of $\hat{\Delta}$ under τ for $\mathcal{N}_c$ requires that

1. $\hat{\Delta}$ be defined on S_c , and

$$2. \left[\tau, \begin{bmatrix} \theta \\ \mathbf{0} \end{bmatrix} \right] \in \Delta, \text{ for all } \theta \in \hat{\Delta}.$$

By definition, the first condition is true. In order to further simplify the remainder of the proof, we now show that the second condition is also true if and only if

$$[\tau, \mathbf{b}_j] \in \Delta, \quad 1 \leq j \leq d + (n - r), \quad (6.4.2)$$

where the

$$\mathbf{b}_j = \begin{cases} \begin{bmatrix} \beta_j \\ \mathbf{0} \end{bmatrix} & \text{if } 1 \leq j \leq d. \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_{j-d} \end{bmatrix} & \text{if } d+1 \leq j \leq d + (n - r). \end{cases}$$

are a basis for Δ , and the β_j , $1 \leq j \leq d$, and $\mathbf{e}_j$, $1 \leq j \leq (n - r)$ are bases for $\hat{\Delta}$ and S_u respectively². The proof of condition (6.4.2) will permit us to work with a finite basis, greatly simplifying the remainder of the proof of Theorem 6.2.

Clearly, condition (6.4.2) is necessary because the basis fields $\mathbf{b}_j$ are, by definition, all elements of Δ . As for sufficiency, assume that condition (6.4.2) is true. Then, given any vector field $\theta \in \Delta$ where $\theta(\mathbf{x}_c) = \sum_{j=1}^{d+(n-r)} c_j(\mathbf{x}_c) \mathbf{b}_j(\mathbf{x}_c)$ for some smooth scalar function coefficients c_j , we have (see Lemma 4.17)

$$[\tau_c, \theta] = \sum_{j=1}^d c_j [\tau_c, \mathbf{b}_j] + \sum_{j=1}^d (\mathcal{L}_{\tau_c} c_j) \mathbf{b}_j.$$

The terms in the first sum are elements of Δ by assumption, and those in the second sum are elements of Δ by definition. Therefore sufficiency is proven, and we

²Since $S_u = \text{span}\{\mathbf{e}_r, \mathbf{e}_{r+1}, \dots, \mathbf{e}_n\}$ by Definition 6.3 and simplifying assumption (6.4.1).

may concentrate on providing a compensating term that satisfies the basis condition (6.4.2).

Assume for the moment that we can find a function matrix $\mathbf{P}_\Delta(\mathbf{x})$ that acts as a projection³ onto the distribution $\Delta(\mathbf{x})$ at every $\mathbf{x}$. Then $\mathbf{P}_\Lambda(\mathbf{x}) = \mathbf{I} - \mathbf{P}_\Delta(\mathbf{x})$ is the projection onto a distribution Λ such that subspaces $\Lambda(\mathbf{z})$ and $\Delta(\mathbf{z})$ are complementary⁴ at every $\mathbf{x}$. To prove this, we must show that $\text{Im}(\mathbf{P}_\Delta(\mathbf{x})) \cap \text{Im}(\mathbf{P}_\Lambda(\mathbf{x})) = \text{span}\{\mathbf{0}\}$, and $\text{Im}(\mathbf{P}_\Delta(\mathbf{x})) + \text{Im}(\mathbf{P}_\Lambda(\mathbf{x})) = \mathbb{R}^n$. Assume that $\text{Im}(\mathbf{P}_\Delta(\mathbf{x})) \cap \text{Im}(\mathbf{P}_\Lambda(\mathbf{x})) = S \neq \text{span}\{\mathbf{0}\}$. Then for every vector $\mathbf{v} \in S$ we have

$$\mathbf{P}_\Delta(\mathbf{x}) \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{v} = \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{P}_\Delta(\mathbf{x}) \mathbf{v} = \mathbf{v}. \quad (6.4.3)$$

However,

$$\begin{aligned} \mathbf{P}_\Delta(\mathbf{x}) \mathbf{P}_\Lambda(\mathbf{x}) &= \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{P}_\Delta(\mathbf{x}) \\ &= \mathbf{P}_\Delta(\mathbf{x}) - (\mathbf{P}_\Delta(\mathbf{x}))^2 \\ &= \mathbf{P}_\Delta(\mathbf{x}) - \mathbf{P}_\Delta(\mathbf{x}) = \mathbf{0}. \end{aligned}$$

and, so, we see that the only vector that satisfies condition (6.4.3) is zero. Therefore, $S = \text{span}\{\mathbf{0}\}$, which contradicts the assumption. Continuing with the proof of the second condition, for any $\mathbf{x} \in \mathbb{R}^n$, we have

$$\begin{aligned} \mathbf{x} &= \mathbf{x} + \mathbf{P}_\Delta(\mathbf{x}) \mathbf{x} - \mathbf{P}_\Delta(\mathbf{x}) \mathbf{x} \\ &= \mathbf{P}_\Delta(\mathbf{x}) \mathbf{x} + (\mathbf{I} - \mathbf{P}_\Delta(\mathbf{x})) \mathbf{x} \\ &= \mathbf{P}_\Delta(\mathbf{x}) \mathbf{x} + \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{x}. \end{aligned}$$

Therefore, complementarity of Δ and Λ is proven, and we have seen that every vector field can be decomposed into its projection onto Δ and Λ .

Our next task is to isolate the component of the bracket that does *not* lie in Δ . This is the component that we wish to cancel with a compensating term.

³A projection onto a subspace S is a matrix $\mathbf{P}$ such that $\text{Im}(\mathbf{P}) = S$ and $(\mathbf{P})^2 = \mathbf{P}$.

⁴Two subspaces $S_1, S_2 \subseteq \mathbb{R}^n$ are *complementary* if $S_1 + S_2 = \text{span}\{\mathbf{0}\}$ and $S_1 \cap S_2 = \mathbb{R}^n$. They are *orthogonal complements* if, in addition to being complementary, for each $\mathbf{x}_1 \in S_1$ and $\mathbf{x}_2 \in S_2$ we have $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle = 0$.

We know that a system augmented by compensator $\mathbf{v}$ will have Lie brackets for $1 \leq l \leq d + (n - r)$ that can be decomposed as follows:

$$\begin{aligned} [\tau + \mathbf{v}, \mathbf{b}_l] &= [\tau, \mathbf{b}_l] + [\mathbf{v}, \mathbf{b}_l] \\ &= \mathbf{P}_\Delta(\mathbf{x}) [\tau, \mathbf{b}_l] + \mathbf{P}_\Lambda(\mathbf{x}) [\tau, \mathbf{b}_l] + [\mathbf{v}, \mathbf{b}_l]. \end{aligned}$$

Therefore, in order to cancel the unwanted dynamical component that is *not* an element of Δ , we must choose a compensator $\mathbf{v}$ such that, for all $1 \leq l \leq d + (n - r)$,

$$[\mathbf{v}, \mathbf{b}_l] = -\mathbf{P}_\Lambda(\mathbf{x}) [\tau, \mathbf{b}_l]. \quad (6.4.4)$$

But can we characterize a family of compensators that satisfy condition (6.4.4)? Recall from the theorem of Frobenius that, because Δ is controlled invariant, a diffeomorphism

$$\mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} \Phi_c(\mathbf{x}_c) \\ \mathbf{x}_u \end{bmatrix}$$

exists such that

$$\frac{\partial \Phi_c(\mathbf{x}_c)}{\partial \mathbf{x}_c} \Delta(\mathbf{x}_c) = \text{span} \{ \mathbf{e}_1, \dots, \mathbf{e}_d \}.$$

We claim that a completely integrable Δ is modularly invariant under vector field τ if and only if the network equations are augmented with additive compensator

$$\mathbf{v}(\mathbf{x}) = \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{p}(\mathbf{x}) \dots$$

where

$$\begin{aligned} \mathbf{P}_\Lambda(\mathbf{x}) &= \mathbf{M}^{-1}(\mathbf{x}) \dot{\mathbf{P}} \mathbf{M}(\mathbf{x}) . \\ \mathbf{M}(\mathbf{x}) &= \frac{\partial \Phi(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{M}^{-1}(\mathbf{x}) = \underbrace{\frac{\partial \Phi^{-1}(\mathbf{z})}{\partial \mathbf{z}}}_{\mathbf{z}=\Phi(\mathbf{x})} . \quad \text{and} \\ \mathbf{p}(\mathbf{x}) &= \tau(\mathbf{x}) + \mathbf{q}(\mathbf{x}) , \quad \text{such that} \quad \mathbf{q}(\mathbf{x}) \in \Delta(\mathbf{x}) . \end{aligned}$$

To prove sufficiency, we move to $\mathbf{z}$ coordinates. Converting (6.4.2) to $\mathbf{z}$ coordinates we have

$$\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{v}(\Phi^{-1}(\mathbf{z})) = \dot{\mathbf{P}}\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z})).$$

Since Φ ‘flattens’ Δ we are free to choose basis fields $\beta_l = \mathbf{M}^{-1}(\mathbf{x}) \mathbf{e}_l$, $1 \leq l \leq d$, and therefore the left hand side of condition (6.4.4) can be expressed in $\mathbf{z}$ coordinates as⁵

$$\begin{aligned} & [\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{v}(\Phi^{-1}(\mathbf{z})) \cdot \mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{b}_l(\Phi^{-1}(\mathbf{z}))] \\ &= \left[\dot{\mathbf{P}}\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z})) \cdot \begin{bmatrix} \delta_l(1, d) \mathbf{e}_l \\ \delta_l(d+1, d+(n-r)) \mathbf{e}_l \end{bmatrix} \right] \\ &= [0] \dot{\mathbf{P}}\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z})) + \frac{\partial \dot{\mathbf{P}}\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z}))}{\partial \mathbf{z}} \\ &= \dot{\mathbf{P}} \frac{\partial \mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z}))}{\partial \mathbf{z}} \begin{bmatrix} \delta_l(1, d) \mathbf{e}_l \\ \delta_l(d+1, d+(n-r)) \mathbf{e}_l \end{bmatrix}. \end{aligned} \quad (6.4.5)$$

Similarly, converting the right side of condition (6.4.4) to $\mathbf{z}$ coordinates yields

$$\begin{aligned} & -\dot{\mathbf{P}} [\mathbf{M}(\Phi^{-1}(\mathbf{z})) \boldsymbol{\tau}(\Phi^{-1}(\mathbf{z})) \cdot \mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{b}_l(\Phi^{-1}(\mathbf{z}))] \\ &= \dot{\mathbf{P}} \left[\dot{\mathbf{P}}\mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z})) \cdot \begin{bmatrix} \delta_l(1, d) \mathbf{e}_l \\ \delta_l(d+1, d+(n-r)) \mathbf{e}_l \end{bmatrix} \right] \\ &= \dot{\mathbf{P}} [0] \mathbf{M}(\Phi^{-1}(\mathbf{z})) \mathbf{p}(\Phi^{-1}(\mathbf{z})) + \frac{\partial \dot{\mathbf{P}}\mathbf{M}(\Phi^{-1}(\mathbf{z})) \boldsymbol{\tau}(\Phi^{-1}(\mathbf{z}))}{\partial \mathbf{z}} \\ &= \dot{\mathbf{P}} \frac{\partial \mathbf{M}(\Phi^{-1}(\mathbf{z})) \boldsymbol{\tau}(\Phi^{-1}(\mathbf{z}))}{\partial \mathbf{z}} \begin{bmatrix} \delta_l(1, d) \mathbf{e}_l \\ \delta_l(d+1, d+(n-r)) \mathbf{e}_l \end{bmatrix}. \end{aligned} \quad (6.4.6)$$

Now, since

$$\mathbf{P}_\Lambda(\mathbf{x}) \mathbf{p}(\mathbf{x}) = \mathbf{P}_\Lambda(\mathbf{x}) \boldsymbol{\tau}(\mathbf{x}) + \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{q}(\mathbf{x}) = \mathbf{P}_\Lambda(\mathbf{x}) \boldsymbol{\tau}(\mathbf{x}).$$

⁵The two parameter Dirac delta function is defined as $\delta_b(a, c) = \begin{cases} 1 & \text{if } a \leq b \leq c, \\ 0 & \text{otherwise.} \end{cases}$

we see that equation (6.4.6) is identical to (6.4.5), if we select $\mathbf{v}(\mathbf{x}) = \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{p}(\mathbf{x})$. Thus, sufficiency is proven.

To prove necessity, we assume that

$$[\tau + \mathbf{v}, \mathbf{b}_l] = [\mathbf{v}, \mathbf{b}_l] + [\tau, \mathbf{b}_l] \in \Delta, \quad (6.4.7)$$

and show that $\mathbf{v}(\mathbf{x}) = \mathbf{P}_\Lambda(\mathbf{x}) \mathbf{p}(\mathbf{x})$ follows. Observe that

$$[\mathbf{v}, \mathbf{b}_l] = -\mathbf{P}_\Lambda(\mathbf{x}) [\mathbf{v}, \mathbf{b}_l], \quad (6.4.8)$$

because, projecting both sides of equation (6.4.4) onto Λ we have

$$\mathbf{P}_\Lambda(\mathbf{x}) [\mathbf{v}, \mathbf{b}_l] = -(\mathbf{P}_\Lambda(\mathbf{x}))^2 [\tau, \mathbf{b}_l] = -\mathbf{P}_\Lambda(\mathbf{x}) [\tau, \mathbf{b}_l] = [\mathbf{v}, \mathbf{b}_l].$$

Combining conditions (6.4.4) and (6.4.8) we see that

$$[\mathbf{P}_\Lambda(\mathbf{x}) \mathbf{v}, \mathbf{b}_l] = [\mathbf{P}_\Lambda(\mathbf{x}) \tau, \mathbf{b}_l],$$

which implies that $\mathbf{P}_\Lambda(\mathbf{x}) \mathbf{v} = \mathbf{P}_\Lambda(\mathbf{x}) \tau$. Therefore, we can select any $\mathbf{v}$ that projects to the same element of Λ that τ projects to, i.e.

$$\mathbf{v} = \tau + \mathbf{q}, \quad \text{where } \mathbf{q} \in \Delta,$$

as claimed. This proves necessity.

We have proven the necessity and sufficiency of a particular form of $\mathbf{v}$ that assumes the existence of a projection onto completely integrable distribution Δ (or equivalently onto its complement Λ). Therefore, to prove existence, we need only show that such a projection exists. This is immediate, because Frobenius' theorem guarantees the existence of the coordinate transformation Φ under which Δ is a constant linear subspace (therefore having at least one complementary subspace), and a projection always exists for every pair of complementary linear subspaces [Goh86]. ■

6.4.3 Enforcing Manifold Constraints by Additive Compensation

Our main result for this section is a constructive proof that the evolution equations of a heterogenous GNF network can always be augmented to constrain the network dynamics to a constant dimensional differentiable manifold $\mathcal{M}$. Furthermore, the sub-dynamics associated with any *subset* of network neurons can be constrained independently of the dynamics of the rest of the neurons. This is an important result because it means that it is possible to ‘sculpt’ the dynamics of the network to fit application specific interpretations of the state space.

Once again, without loss of generality, we will make assumption (6.4.1) to simplify the exposition.

Theorem 6.3

Given a heterogenous GNF network with system of neuron equations

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{j=1}^m \mathbf{g}_j(\mathbf{x}) u_j = \mathbf{q}(\mathbf{x}, \mathbf{u}) .$$

then for any choice of r neurons with index set $\mathcal{N}_c$, nonlinear d -dimensional differentiable manifold $\mathcal{M} \subseteq S_c$, and initial state $\mathbf{x}_0 \in \mathcal{M}$, there exists a matrix function $\tilde{\mathbf{M}}(\mathbf{x})$ and neighbourhood $\mathcal{U}$ of $\mathbf{x}_0$ such that the augmented system of neuron equations

$$\dot{\mathbf{x}} = \tilde{\mathbf{M}}(\mathbf{x}) \mathbf{q}(\mathbf{x}, \mathbf{u}) .$$

has sub-dynamics locally constrained to lie on $\mathcal{M} \cap \mathcal{U}$.

Proof: For the first part of the proof, recall that Lemma 6.1 specifies the existence of a family of completely integrable distributions $\mathcal{F}_M$ that contain d -dimensional manifold $\mathcal{M}$ as a leaf. Pick any $\hat{\Delta} \in \mathcal{F}_M$, with basis $\beta_1(\mathbf{x}) \dots \beta_d(\mathbf{x})$, and let $\Delta = \hat{\Delta} + S_c$ be

its extension to $\mathbb{R}^n$. Then Δ has basis

$$\mathbf{b}_j = \begin{cases} \begin{bmatrix} \beta_j \\ \mathbf{0} \end{bmatrix} & \text{if } 1 \leq j \leq d. \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_{j-d} \end{bmatrix} & \text{if } d+1 \leq j \leq d+(n-r). \end{cases}$$

Theorem 6.2 tells us that, in order for Δ to be modularly invariant under each and every network vector field, for $1 \leq l \leq d+(n-r)$ we must have

$$\begin{aligned} [\mathbf{f}, \mathbf{b}_l] &\in \Delta, \quad \text{and} \\ [\mathbf{g}_j, \mathbf{b}_l] &\in \Delta, \quad 1 \leq j \leq m. \end{aligned} \tag{6.4.9}$$

From Theorem 6.2 we know that there exists a nonsingular function matrix $\mathbf{M}(\mathbf{x})$ such that the additive compensators

$$\begin{aligned} \mathbf{v}_0(\mathbf{x}) &= \mathbf{M}^{-1}(\mathbf{x}) \dot{\mathbf{P}}\mathbf{M}(\mathbf{x}) \mathbf{f}(\mathbf{x}), \quad \text{and} \\ \mathbf{v}_j(\mathbf{x}) &= \mathbf{M}^{-1}(\mathbf{x}) \dot{\mathbf{P}}\mathbf{M}(\mathbf{x}) \mathbf{g}_j(\mathbf{x}) u_j, \quad 1 \leq j \leq m. \end{aligned} \tag{6.4.10}$$

satisfy conditions (6.4.9). We claim that Δ is modularly invariant under $\mathbf{q}(\mathbf{x}, \mathbf{u})$ if and only if we use compensator

$$\begin{aligned} \mathbf{v}(\mathbf{x}) &= \mathbf{v}_0(\mathbf{x}) + \sum_{j=1}^m \mathbf{v}_j(\mathbf{x}) u_j, \quad \text{i.e.} \\ [\mathbf{q} + \mathbf{v}, \mathbf{b}_l] &\in \Delta. \end{aligned} \tag{6.4.11}$$

To prove necessity, observe that

$$\mathbf{q} + \mathbf{v} = \hat{\mathbf{f}} + \sum_{j=1}^m \hat{\mathbf{g}}_j u_j$$

where

$$\begin{aligned} \hat{\mathbf{f}} &= \mathbf{f} + \mathbf{v}_0, \quad \text{and} \\ \hat{\mathbf{g}}_j &= \mathbf{g}_j + \mathbf{v}_j, \quad \text{for } 1 \leq j \leq m. \end{aligned} \tag{6.4.12}$$

Theorem 6.2 tells us that, for $1 \leq l \leq d + (n - r)$,

$$\begin{aligned} [\hat{\mathbf{f}}, \mathbf{b}_l] &= [\mathbf{f} + \mathbf{v}_0, \mathbf{b}_l] \in \Delta, \quad \text{and} \\ [\hat{\mathbf{g}}_j, \mathbf{b}_l] &= [\mathbf{g}_j + \mathbf{v}_j, \mathbf{b}_l] \in \Delta, \quad 1 \leq j \leq m. \end{aligned}$$

and (6.4.9) is satisfied, completing the proof of necessity. As for sufficiency, assume that equation (6.4.11) is true, then applying Lemma 6.1 we have that

$$\begin{aligned} \mathbf{v}(\mathbf{x}) &= \mathbf{M}^{-1}(\mathbf{x}) \hat{\mathbf{P}}\mathbf{M}(\mathbf{x}) \mathbf{q}(\mathbf{x}, \mathbf{u}) \\ &= \mathbf{M}^{-1}(\mathbf{x}) \hat{\mathbf{P}}\mathbf{M}(\mathbf{x}) \mathbf{f}(\mathbf{x}) + \sum_{j=1}^m \mathbf{M}^{-1}(\mathbf{x}) \hat{\mathbf{P}}\mathbf{M}(\mathbf{x}) \mathbf{g}_j(\mathbf{x}) u_j \\ &= \mathbf{v}_0(\mathbf{x}) + \sum_{j=1}^m \mathbf{v}_j(\mathbf{x}) u_j. \end{aligned}$$

For the second part of the proof, we show that the matrix function

$$\hat{\mathbf{M}}(\mathbf{x}) = \mathbf{I} + \mathbf{M}^{-1}(\mathbf{x}) \hat{\mathbf{P}}\mathbf{M}(\mathbf{x})$$

satisfies the claim of Theorem 6.3. Clearly,

$$\begin{aligned} \hat{\mathbf{M}}(\mathbf{x}) \mathbf{q}(\mathbf{x}, \mathbf{u}) &= \mathbf{q}(\mathbf{x}, \mathbf{u}) + \mathbf{M}^{-1}(\mathbf{x}) \hat{\mathbf{P}}\mathbf{M}(\mathbf{x}) \mathbf{q}(\mathbf{x}, \mathbf{u}) \\ &= \mathbf{q}(\mathbf{x}, \mathbf{u}) + \mathbf{v}(\mathbf{x}), \end{aligned}$$

which, as we have seen in the first part of the proof, makes Δ modularly invariant under all of the vector fields of the network. We claim that the vector fields (6.4.12) of the compensated network are also elements of Δ . In $\mathbf{z}$ coordinates the compensated system can be expressed as

$$\begin{aligned} \dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{z}_2) + \sum_{j=1}^m \mathbf{g}_{1,j}(\mathbf{z}_1, \mathbf{z}_2) u_j \\ \dot{\mathbf{z}}_2 &= \mathbf{f}_2(\mathbf{z}_2) + \sum_{j=1}^m \mathbf{g}_{2,j}(\mathbf{z}_2) u_j, \end{aligned}$$

because of modular invariance. Now, recall that we chose Φ such that

$$\frac{\partial \Phi}{\partial \mathbf{x}} \Delta = \Phi(\mathcal{M}) = \text{span}\{\mathbf{e}_1, \dots, \mathbf{e}_d\}.$$

Therefore, if we select $\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{0} \end{bmatrix} \in \frac{\partial \Phi}{\partial \mathbf{x}} \Delta$, then the system dynamics become

$$\begin{aligned} \dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{0}) + \sum_{j=1}^m \mathbf{g}_{1,j}(\mathbf{z}_1, \mathbf{0}) u_j \\ \dot{\mathbf{z}}_2 &= \mathbf{0}. \end{aligned}$$

and the network activation state cannot leave the manifold. This completes the proof of the original theorem. ■

The construction of the compensator in the previous theorem also tells us what will happen to the dynamics of the network if it does not initially start on the manifold $\mathcal{M}$. If the initial state of the network is noisy, the cell assembly will be constrained to evolve on the unique nonlinear leaf of $\hat{\Delta}$ that contains $\mathbf{x}_0$, rather than the intended manifold $\mathcal{M}$. We capture this in the following corollary to Theorem 6.3.

Corollary 6.1

Given a heterogenous GNF network, any choice of r neurons $\mathcal{N}_c$ and a completely integrable distribution $\hat{\Delta}$ that locally foliates a neighbourhood $\mathcal{U}$ of the initial network state $\mathbf{x}_0$, the sub-dynamics of assembly $\mathcal{N}_c$ are locally constrained to lie on the unique leaf $\mathcal{M} \cap \mathcal{U}$ of the foliation that contains $\mathbf{x}_0$.

Proof: Given the coordinate transformation $\mathbf{z} = \Phi(\mathbf{x})$ described in the proof of Theorem 6.3, and an arbitrary $\mathbf{x}_0 \in \mathcal{U}$, the transformation $\mathbf{z} = \Phi(\mathbf{x}) - \Phi(\mathbf{x}_0)$ shifts the manifolds in $\mathbf{z}$ coordinates such that the augmented network can be put into the form

$$\begin{aligned} \dot{\mathbf{z}}_1 &= \mathbf{f}_1(\mathbf{z}_1, \mathbf{0}) + \sum_{j=1}^m \mathbf{g}_{1,j}(\mathbf{z}_1, \mathbf{0}) u_j \\ \dot{\mathbf{z}}_2 &= \mathbf{0}. \end{aligned}$$

Therefore, we may conclude that the network dynamics are confined to the maximal integral submanifold of Δ that contains the initial state. ■

In general, the compensator violates the usual assumption of local computation since the i^{th} neuron equation now depends on the activation values (time derivatives) of all of the other neurons. However, the designer of a hybrid system based only in part on neural network techniques, may be less interested in fidelity with biology than in the dynamical behaviour of the system being investigated. Furthermore, as we shall see in the next section, there are special cases where the compensator can be given a ‘neural’ realization.

6.5 Adaptive Learning of Linear Reachability Constraints

Consider the modular computational system discussed in Section 2.2.1 and illustrated in Figure 2.1. In this section we are interested in the case where a GNF network acts as a connectionist module that, given a command signal $\mathbf{s}_i$ as input, controls the states of r modules by providing r reference signals $\mathbf{x}_{c_k}$, $1 \leq k \leq r$ that guide the states of the dependent modules to an associatively recalled ‘memory’ state $\mathbf{p}_{c_k}$, $1 \leq k \leq r$. In addition, the r reference signals must belong to a (possibly highly nonlinear) connected region of each subspace S_k , in order to perform the current computation. This scenario occurs in control applications, where an input function must not deviate from a connected set of ‘safe’ values determined by the physical operating limitations on a controlled output device, or a family of computational data structures that are encoded into the state vectors (e.g. see Chapter 8 on connectionist type constraints for tensor product encoded data structures).

In the remainder of this section we will illustrate the use of Theorem 6.3 (and implicitly Theorem 6.2) by designing an adaptive connectionist module that learns to satisfy the above requirements. We will restrict our construction to the special case where the constraint regions are linear subspaces, unknown to the connectionist module prior to the adaptation process.

6.5.1 A Compensated Continuous-Time Feed Forward Network

In order to satisfy the design constraints of the previous section.

1. we will employ a recurrent back propagation network to perform the desired pattern association of pattern pairs

$$s_i \quad \text{and} \quad \mathbf{p}_i = \begin{bmatrix} \mathbf{p}_{i,1} \\ \vdots \\ \mathbf{p}_{i,r} \end{bmatrix}, \quad \text{for } 1 \leq i \leq q. \quad \text{and}$$

2. we will incorporate into this network an adaptive realization of the compensator in Theorem 6.3 to enforce the desired modular reachability constraints on output assemblies $\mathcal{N}_k$, $1 \leq k \leq r$.

We assume that either the controlling network's activation state is available, or the network is completely observable and its state can be reconstructed by the controlled modules.

The special form of *continuous time back propagation network* that we shall consider has neuron dynamics of the form

$$\begin{aligned} \dot{x}_i(t) = & -\alpha x_i(t) + \beta \sigma \left(\sum_{j=1}^n w_{ij} x_j(t) \right) \\ & + \sum_{k=1}^m g_{ik} u_k(t). \end{aligned}$$

where α and β are constants and $\sigma(\cdot)$ is the familiar 'sigmoidal' output function. Expressed in vector field notation the system becomes

$$\dot{x} = f(x) + [g_1 \dots g_m] u. \quad (6.5.1)$$

To simplify the presentation without loss of generality, we will consider only a single controlled module which receives input signals from assembly $\mathcal{N}_c$. In order to find a particular compensator for this network we must select a matrix function

$\mathbf{P}_\Delta(\mathbf{x})$ that behaves as a projection onto $\Delta(\mathbf{x})$ at every $\mathbf{x}$. In $\mathbf{z}$ coordinates we can easily find a canonical projection in constraint space S_c such that $\Lambda = \Delta^\perp$, i.e.

$$\begin{aligned}\dot{\mathbf{P}}_\Delta(\mathbf{z}) &= \begin{bmatrix} \mathbf{I}^{d \times d} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \\ \dot{\mathbf{P}}_{\Delta^\perp}(\mathbf{z}) &= \begin{bmatrix} \mathbf{I}^{r \times r} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} - \dot{\mathbf{P}}_\Delta(\mathbf{z}) = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}^{(r-d) \times (r-d)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{P}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (6.5.2)\end{aligned}$$

Therefore, $\mathbf{P}_{\Delta^\perp}(\mathbf{x}) = \mathbf{M}^{-1}(\mathbf{x}) \dot{\mathbf{P}}_{\Delta^\perp}(\mathbf{z}) \mathbf{M}(\mathbf{x})$.

Our task now is to show how learning of *linear* manifold constraints can be incorporated into a learning law for the fixed point problem⁶. We extend the dynamic back propagation algorithm [Pin87] for systems of the form (6.5.1). Our network will learn to associate input patterns with desired network equilibria, subject to the constraint that the network dynamics must evolve along a subspace that is spanned by these observed equilibria. Without a manifold constraint, fixed points not encountered during training will cause trajectories to exit this linear manifold. The set $\mathcal{N}_c$ of constrained neurons coincides with the set of output neurons.

We will consider the case where the manifolds of interest are linear subspaces of $\mathbb{R}^n$. In this event, matrices M and M^{-1} are any constant linear coordinate transformations such that $\Delta = M^{-1}IM$, i.e. $\mathbf{P}_{\Delta^\perp}(\mathbf{x}) = \mathbf{M}^{-1} \dot{\mathbf{P}}_{\Delta^\perp}(\mathbf{z}) \mathbf{M}$. We will assume that training vectors are presented which are elements of the legal constraint space Δ . Given prior knowledge of Δ we could use the singular value decomposition to compute an orthogonal projection $\hat{\mathbf{P}}_{\Delta^\perp}$ onto $\Delta^\perp$ for this training set, and then hardwire the compensator $\mathbf{v}(\mathbf{x})$ into the network equations. However, we are interested in *adaptively* finding the augmenting term $\mathbf{v}(\mathbf{x})$. To this end, it is sufficient to use any technique that converges to a projection matrix on $\Delta^\perp$ given training data consisting of vectors that span the desired subspace Δ .

⁶Also called the associative or error-correcting memory problem [Kos88, Hop84].

With the addition of compensating field $\mathbf{v}(\mathbf{x})$, the network equations can be put into the form

$$\dot{x} = -\alpha x + \beta(\mathbf{I} - \mathbf{P}_{\Delta^\perp})\sigma(Wx) + G(x)u \quad (6.5.3)$$

Theorem 6.3 clearly applies because, in z coordinates, the network Lie brackets take the form

$$\begin{aligned} & \left[-\alpha M(z)\Phi^{-1}(z) + \beta M(z)\sigma(W\Phi^{-1}(z)) \cdot \begin{bmatrix} e_l \\ \lambda_{zl} \end{bmatrix} \right] \\ &= \begin{bmatrix} 0 \\ \frac{\partial \lambda_{zl}}{\partial z} \end{bmatrix} (-\alpha M(z)\Phi^{-1}(z) + \beta M(z)\sigma(W\Phi^{-1}(z))) - \alpha I \begin{bmatrix} e_l \\ \lambda_{zl} \end{bmatrix} \\ & - \left(\begin{bmatrix} \gamma_1 \cdots \gamma_n \end{bmatrix} + \beta M(z)D(\Phi^{-1}(z))WM^{-1}(z) \right) \begin{bmatrix} e_l \\ \lambda_{zl} \end{bmatrix} \end{aligned} \quad (6.5.4)$$

$$\begin{bmatrix} g_k \cdot \begin{bmatrix} e_l \\ \lambda_{zl} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\partial \lambda_{zl}}{\partial z} \end{bmatrix} M(z)g_k - \frac{\partial M(z)g_k}{\partial z} \begin{bmatrix} e_l \\ \lambda_{zl} \end{bmatrix} \quad (6.5.5)$$

for $1 \leq k \leq m$ and $1 \leq l \leq d$, where

$$\gamma_i = \frac{\partial M(z)}{\partial z_i} [\beta \sigma(W\Phi^{-1}(z)) - \alpha \Phi^{-1}(z)] .$$

e_l is the l^{th} d -dimensional canonical basis vector of $\Delta(\Phi^{-1}(z))$.

$$\begin{aligned} M(z) &= \begin{bmatrix} \frac{\partial \Phi_z}{\partial z} & 0^{r \times (n-r)} \\ 0^{(n-r) \times r} & I^{(n-r) \times (n-r)} \end{bmatrix} , \\ M^{-1}(z) &= \begin{bmatrix} \frac{\partial \Phi_z^{-1}}{\partial z} & 0^{r \times (n-r)} \\ 0^{(n-r) \times r} & I^{(n-r) \times (n-r)} \end{bmatrix} . \end{aligned}$$

and $D(y)$ is a diagonal matrix with on-diagonal elements $D_{ii}(y) = \frac{\partial \sigma(y)}{\partial y_i}$.

6.5.2 A Law for Simultaneous Learning of Fixed Points and Reachability Constraints

Consider the following theorem of Kohonen, which we reproduce here without proof (see [Koh89]).

Theorem 6.4

Given a subspace $\Delta = \text{span}\{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_q\}$, where the $\hat{\mathbf{x}}_i$ are not necessarily linearly independent, and the following dynamics

$$\begin{aligned} \mathbf{P}_0 &= \mathbf{I}, \\ \frac{\partial \mathbf{P}_t}{\partial t} &= \mathbf{P}_t \mathbf{x} \mathbf{x}^T \mathbf{P}_t, \end{aligned} \tag{6.5.6}$$

where $\mathbf{x} = \mathbf{x}_i$ if $\lfloor t \bmod T \rfloor = 0$ for $T > 0$, then

1. $\lim_{t \rightarrow \infty} \text{Im}(\mathbf{P}_t) = \Delta^\perp$ asymptotically, and
2. $\mathbf{P}_{Tk} = \mathbf{P}_{T(k-1)} - \frac{\mathbf{P}_{T(k-1)} \mathbf{x}_k \mathbf{x}_k^T \mathbf{P}_{T(k-1)}}{\mathbf{x}_k^T \mathbf{P}_{T(k-1)} \mathbf{x}_k} - \epsilon(t) \mathbf{I}$
for $t \geq 0$, $k \geq 0$, $0 \leq \epsilon(t) \leq 1$, and $\lim_{t \rightarrow \infty} \epsilon(t) = 0$.

Proof: See [Koh89] for a proof. ■

We may conclude from Theorem 6.4 that the matrix $\mathbf{P}_t$ asymptotically converges to the projector onto the orthogonal complement of Δ if any spanning set is repeatedly ‘presented’ to the evolution equations. Furthermore, by making the presentation times longer we can ensure that $\mathbf{P}_t$ is arbitrarily close to being idempotent for all time, i.e.

$$\mathbf{P}_{Tk}^2 \approx \mathbf{P}_{Tk}.$$

This is precisely the behaviour that we require to adaptively find $\mathbf{P}_{\Delta^\perp}(\mathbf{x})$ from example vectors contained in the linear subspace Δ . Therefore, we incorporate evolution equation (6.5.6) into the continuous back propagation learning law in order to realize an adaptive version of our additive reachability constraint compensator.

Following the development of Pineda [Pin87], it is essential that the network dynamics are stable during weight adaptation. However, as we have seen, novelty filter adaptation preserves the symmetry and idempotence of a matrix, so that, if $P^\perp$ is initialized to I , it will remain a projection matrix during adaptation. Since the effect of this matrix is to remove the offending dynamical component orthogonal to Δ , we conclude that the resulting dynamics are the same as the dynamics on the factor space $\mathbb{R}^n / \Delta^\perp$. Therefore, the manifold constraint will preserve system stability as required for the dynamic back propagation algorithm described in [Pin87].

Weight updates that associate an input pattern with a network equilibria, x^0 , follow the negative of the gradient of the standard back propagation algorithm, yielding

$$\frac{dw_{jk}}{dt} = -\rho \sum_{p \in \mathcal{N}_c} \tau_p \frac{\partial x^0}{\partial w_{jk}} \quad (6.5.7)$$

where τ_p is the training error between the desired p^{th} fixed point value and the actual p^{th} neuron output. Decay rate ρ is selected so that weight dynamics are slow enough to allow the run time dynamics to remain close to an equilibrium state. Since the fixed points of the augmented system are given by the equilibrium constraint

$$\alpha x^0 = \beta \mathbf{P}_\Delta \sigma(Wx^0) + Gu \quad (6.5.8)$$

we can take the derivative of both sides to obtain

$$\frac{\partial x^0}{\partial w_{jk}} = \beta \mathbf{P}_\Delta D(x^0) W \frac{\partial x^0}{\partial w_{jk}} + D(x^0) \begin{bmatrix} 0 \\ \vdots \\ \delta_{ij} x_k^0 \\ \vdots \\ 0 \end{bmatrix} \quad (6.5.9)$$

with diagonal matrix $D(x)_{ii} = \frac{\partial \sigma_i(W_i x)}{\partial x_i}$. Solving for $\frac{\partial x^0}{\partial w_{jk}}$ and substituting into (6.5.7) we obtain the weight update equation (where $\hat{\mathbf{P}}$ is from equation (6.5.2))

$$\begin{aligned} \frac{dw_{jk}}{dt} &= \rho \beta \frac{\partial \sigma_j(W_j x)}{\partial x_k} x_k^0 \\ &\quad \sum_{p \in \mathcal{N}_c} \tau_p \left[\alpha I - \beta \begin{bmatrix} \hat{\mathbf{P}}^{r \times r} & 0 \\ 0 & 0 \end{bmatrix} \right]_{ij}^{-1} \\ &= \rho \beta \frac{\partial \sigma_j(W_j x)}{\partial x_k} x_k^0 \sum_{p \in \mathcal{N}_c} \tau_p L_{ij}^{-1}. \end{aligned} \quad (6.5.10)$$

Pineda avoids the inverse matrix L^{-1} in equation (6.5.10) by introducing the relation $z = L^{-1}\Gamma$, where Γ is the vector of teaching signals τ_p (zero for non-output nodes). This is interpreted as the equilibrium condition for the dynamical system

$$\dot{z}^0 = -Lz^0 + \Gamma. \quad (6.5.11)$$

Equations (6.5.6), (6.5.10) and (6.5.11) correspond to an adaptive learning law that simultaneously learns fixed point constraints and a compensator for linear reachability constraints.

6.6 Chapter Conclusions

In this chapter we have taken the first step toward the design of hybrid connectionist/control systems that are able to learn and enforce manifold constraints on the system dynamics. These systems still retain most of the desirable characteristics of continuous-time connectionist networks. For example, they continue to support soft computations which require approximation of nonlinear mappings, associative recall of memories, simultaneous satisfaction of multiple constraints and graceful degradation of performance. What has been lost is the strictly distributed nature of the equations of evolution.

The main contribution in this chapter has been to show that compensators and feedback controllers exist which confine the dynamics of a GNF network to a nonlinear constraint manifold in activation space. We have also made an initial foray into the task of applying these existence results by deriving a weight update rule that learns to confine the transient dynamics of a continuous-time back propagation network to linear constraint manifolds. The problem of learning a general class of nonlinear manifold constraints is the subject of the next two chapters.

We have also considered the issue of selectively constraining the subdynamics of a group of neurons that represent a cell-assembly. Our results permit more than one constraint manifold to be applied simultaneously to different non-overlapping assemblies.

The results of this chapter appear in [Gra98d, Gra98a, Gra97] which describe the use of compensators to adaptively enforce manifold constraints, and the modified back propagation network of Section 6.5.1.

Chapter 7

Principal Manifold Approximation

In previous chapters we have discovered that GNF networks are, in general, nonholonomic. We were able to conclude, therefore, that algorithms capable of learning and enforcing *general* nonlinear manifold constraints in the presence of noise do not exist. We did find, however, that GNF network architectures can be augmented, in at least two ways, to permit learning and enforcement of *linear* manifold constraints. We are naturally motivated to ask whether these techniques can be generalized to robustly learn and enforce *nonlinear* manifold constraints. In this chapter and the next we answer this question in the affirmative.

Specifically, we examine the problem of approximating a foliation (i.e. partition) of the state space into nonlinear manifolds by randomly sampling the dynamics of a model system that evolves on the leaves of this foliation. In Section 7.2 we describe the problem in an abstract setting and identify a number of assumptions that must be made in order to reliably find a solution. The particular solution method that we develop — Principal Manifold Approximation (PMA) — is outlined in Section 7.3. We present this method without reference to the architectural de-

tails of the learning system so that the reader may acquire intuition without the distraction of issues peculiar to a connectionist realization. Section 7.4 wraps up with general discussion and conclusions.

In Chapter 8 we introduce a batch learning algorithm for PMA that has a connectionist interpretation. This algorithm is then applied to the design of an adaptive compensator and feedback controller that learn and enforce manifold constraints in general nonholonomic GNF networks. The resulting *holonomic* systems are connectionist in spirit, but are better described as hybrid connectionist/control systems. We will also look at the dual problem of applying nonlinear manifold constraints to determine the observability characteristics of a control system.

7.1 Contributions of the Chapter

To our knowledge, Principal Manifold Approximation is the first method that has been proposed to approximate a state space foliation from sparse, noisy velocity estimates. A literature exists for the approximation of individual nonlinear manifolds from a noisy sampling of states, but none of these techniques deals with the issue of estimating an entire partition of the space. The method that we present can be implemented in many ways. We provide a simple serial algorithm that uses conventional numerical techniques to illustrate one such symbolic implementation, and give the reader a feel for the inherent polynomial-time complexity of the problem.

Our general method of modeling locally completely integrable distributions as partial derivatives of a parameterized coordinate transformation is novel. The particular family of regularized RBF mappings that we introduce ultimately relies on an observation that has also been made by Mustafa-Ivaldi [MI92]. However, our goals and application develop along very different lines than those of Mussa-Ivaldi.

Our use of Principal Components Analysis to model tangent spaces of a nonlinear foliation is a novel application of this method. Our characterization of the smoothness of neighbouring PCA estimates and bounds on these estimates in terms

of the gap metric is also new.

Our method is particularly well suited for implementation in a connectionist setting because the approximated foliation can be represented by a vector of weights or activation values. Manifold constraints can therefore be stored as long-term control information, or can be loaded into short-term memory by associative recall. Therefore, we may also regard PMA as offering an entirely novel form of run-time connectionist control knowledge.

7.2 The Manifold Approximation Problem

The problem that we are concerned with in this chapter will be called the *manifold approximation problem*. Consider the learning system illustrated in Figure 7.1. The system observes the behaviour of a dynamic environment through a noisy sensory apparatus. The environment is modeled as a generator of continuous *behavioural events*. It is the task of the learning system to construct a representation of the constraints that shape these behaviours. The sensory apparatus sees the environment through n sensors that output a vector of real-valued measurements. Each behavioural event therefore begins at an initial measured state, and evolves for a finite time along a connected path through $\mathbb{R}^n$. However, only intermittent samples of the state and velocity of this trajectory are output to the learning system, because of limitations imposed by the internal dynamics of the sampling process and other factors. The learning system is able to hold a finite number of state/velocity pairs, and the entire contents of this buffer are available to the learning algorithm.

The fundamental modeling assumption that we will make is that each behavioural event is constrained to lie on the leaf of a nonlinear foliation of $\mathbb{R}^n$, as shown in Figure 7.2. It is the goal of the learning system to discover a representation of this foliation from the noisy, incomplete observations provided by the sensory apparatus. We are not interested in modeling any single event trajectory, but rather the manifold constraints that represent the deeper structure of the environment. Recall

Figure 7.1: Learning System for Manifold Approximation Problem

from Appendix B that a foliation represents a partition of the environmental state space into an uncountable number of manifold constraints. We emphasize that the learning system is required to find an approximation of the *entire partition* rather than a single manifold.

7.2.1 Manifold Approximation is Underconstrained

The problem we have described is very demanding of the learning system because it is highly underconstrained. If we hope to find a meaningful solution, we must further constrain the problem by making some reasonable *a priori* assumptions¹ about the nature of the environmental events and the foliation that shapes them.

7.2.1.1 Uniform Sampling Assumptions

Since the foliation is unbounded, but the observation buffer is finite, it is unreasonable to expect the learned approximation to have a high degree of fidelity over the entire state space. However, in any practical application, the environment will have a limited dynamic range and the observable states will be confined to a

¹These assumptions are a type of *inductive bias* [Sha94]. They reduce the solution space that we must search to foliations that are compatible with our knowledge of the problem domain.

Figure 7.2: The Environmental Foliation - Three leaves (manifolds) from a 2-dimensional foliation of $\mathbb{R}^3$ and six behavioural events. Notice that the events are constrained to evolve on only one manifold - they cannot "hop" from one to another. They may intersect themselves and each other.

bounded region of state space. Therefore, we ask only that the approximation be accurate in this region.

Figure 7.3: The Dynamic Region $\mathcal{U}$ - $\mathcal{U}$ is the region in which the events occur. The foliation should be accurate here. $\mathcal{U}$ is uniformly sampled.

Definition 7.1

The dynamic region of the environment, $\mathcal{U} \subset \mathbb{R}^n$, is a compact bounded (possibly disconnected) subset of $\mathbb{R}^n$. It is contained in the closed n -dimensional hypercube defined by the minimum and maximum measurements that each sensor can make given the dynamic range of the environment. See Figure 7.3. The exact shape of the

dynamic region will depend on the internal details of the environment, and is not important in the sequel.

It is reasonable to expect that, if we are willing to wait long enough, we will eventually make a rich enough set of observations to delineate the shape of the foliation on the dynamic region, $\mathcal{U}$. In particular, we will assume that the environment produces events of sufficient variety that the states in $\mathcal{U}$ are sampled with uniform probability. The case where the sampling density varies over $\mathcal{U}$ is considered further in Chapter 8. In addition, we assume that, in the limit, the event velocities that are observed at a state span the tangent space of the foliation leaf at each state. Clearly, if this were not true, we would not have enough information to estimate the dimensionality and orientation of the tangent spaces.

7.2.1.2 Stationary Foliation Assumption

Another assumption that we will make is that, although the environmental events are dynamic, the deeper structure shaping these events (i.e. the foliation) is stationary. We regard the foliation as a deeper, and presumably longer-term, structure than the behavioural events themselves. The foliation may actually change slowly as long as the speed of the learning algorithm is fast relative to these changes. In this case, the algorithm can be executed repeatedly to ‘track’ this gradual drift. Assumptions of stationarity are typical of many parameter estimation schemes [Joh88].

7.2.1.3 Sparse Sampling Assumptions

Although we have assumed that, in the limit, behavioural events produce velocity vectors that span the tangent space at any given state, we still have insufficient information to reconstruct the tangent spaces from a finite set of observations. To see this, consider that the probability of sampling the same point twice is zero, given a continuous sampling distribution. Therefore, we expect to have only one velocity

estimate at any given state, i.e. the velocity estimates are ‘sparse’. Consider three nearby samples such as those shown in Figure 7.4(a). It is unclear whether these should be modeled with a two dimensional distribution as shown in Figure 7.4(b), or a one dimensional distribution which diverges in this region of space as shown in Figure 7.4(c). In fact, *any* finite set of samples at unique states can be modeled with a smooth one dimensional foliation, regardless of the dimensionality of the true foliation. This is undesirable, however, as it overfits the data, incorrectly constraining event trajectories to lie on one dimensional manifolds. A higher dimensional manifold, as in Figure 7.4(b), has the advantage of economy of description (more states are included in a single constraint) and richness of expression (a greater variety of trajectories can be generated that agree with the constraint). Without a linearly independent basis for the tangent space we cannot know the true tangent space or its dimensionality. In order to alleviate this sparse sampling ambiguity, we will make the following two assumptions.

7.2.1.3.1 Complete Integrability Assumption The first sparse sampling assumption is that the tangent spaces of the foliation are the values of a *completely integrable* distribution on domain $\mathcal{U}$. Such a distribution is involutive and nonsingular (see Appendix B). Involutivity assures us that the distribution induces a partition of $\mathbb{R}^n$ into maximal integral submanifolds, and nonsingularity assures us that the tangent spaces at every point are all of the same dimension. This is a reasonable assumption in practice, since many dynamical systems of interest exhibit at least local complete integrability over large regions of state space [Isi89]. As we shall see in Section 7.3.1, complete integrability has additional representational advantages as well.

7.2.1.3.2 Stiffness Assumption Our second sparse sampling assumption is that the distribution obeys a *stiffness constraint* on $\mathcal{U}$. By this we mean that all of the tangent spaces in any radius r subset of $\mathcal{U}$ are ‘sufficiently similar’ that we would

Figure 7.4: Sparse Sampling Ambiguity - (a) Do the sparse samples in region R tell us the foliation is... (b) 1-dimensional or (c) 2-dimensional?

accept any one of them as an estimate of any of the others in that neighbourhood. A manifold with tangent spaces that show a high degree of similarity over a given region cannot have as great a degree of curvature on this region as one with a greater degree of variation on the same region. Hence, we say that the first manifold is ‘stiffer’ than the second. It is reasonable to impose *some* degree of stiffness on the learning problem, because every differentiable manifold embedded in $\mathbb{R}^n$ obeys such a constraint on a bounded region $\mathcal{U}$ for a small enough r [Boo86].

The intent of the stiffness assumption is to insure that there is sufficient regularity among the tangent spaces of $\mathcal{U}$ that a random sampling of velocity vectors in each subset of a regular covering of $\mathcal{U}$ (see Figure 7.5) can be combined to reconstruct good local approximations of the tangent spaces throughout $\mathcal{U}$. We consider these issues further in Sections 7.3.2.1 and 7.3.2.2.

A convenient measure of the similarity of two tangent spaces S_1 and S_2 is given by the *gap metric* on the space of subspaces² $\mathcal{S}$

$$\Gamma(S_1, S_2) = \|\mathbf{P}_{S_1} - \mathbf{P}_{S_2}\|.$$

where $\mathbf{P}_{S_1}$ and $\mathbf{P}_{S_2}$ are the unique orthogonal projection matrices onto the respective tangent spaces. This distance measure has the following properties ([Goh86]):

1. $0 \leq \Gamma(S_1, S_2) \leq 1$.
2. $\Gamma(S_1, S_2) = 0$ if and only if $S_1 = S_2$.
3. $\Gamma(S_1, S_2) = 1$ if and only if $S_1 = S_2^\perp$ or S_1 and S_2 have different dimensions.

Intuitively, the ‘angle’ between two tangent spaces is very small if the gap is small. The gap is maximized when the spaces are maximally different³, i.e. orthogonal.

The stiffness assumption consists of a gap value γ , which represents the greatest deviation permitted between the tangent space of the foliation and its

²The metric space of subspaces is described in [Goh86].

³The gap is also maximal when the spaces are of different dimension, however, because of the assumption of complete integrability, we will always be comparing tangent spaces of the same dimension.

Figure 7.5: Approximating The Tangent Spaces on a Covering of $\mathcal{U}$

approximation, and a positive radius r , which determines the minimum distance over which tangent spaces at two points may vary by γ . Setting the stiffness to 0 for any radius $r > 0$ permits only linear foliations of the space, because only a linear foliation has tangent spaces that are equal at every point.

7.2.1.4 Gaussian Sampling Noise Assumption

Another source of ambiguity is the noise introduced during sampling of the event trajectory velocities. Noise in the sampling apparatus can introduce a nonzero velocity component from the orthogonal complement of the true tangent space. See Figure 7.6. Consequently, even if we observe a set of velocity vectors spanning the tangent space at a state, estimation noise can obscure the true tangent space and even alter its apparent dimensionality.

In Section 7.3.2 we will employ a statistical estimation technique to recover the tangent space at a sample point in the presence of noise. Therefore, we must make an assumption about the underlying probability distribution, namely, that the noise is additive and Gaussian⁴ with zero mean at every state in $\mathcal{U}$. The estimated velocity $\hat{\mathbf{v}}$ is derived from the true velocity $\mathbf{v}$ by

$$\hat{\mathbf{v}} = \mathbf{v} + \mathbf{u},$$

where $\mathbf{u}$ is a random variable with distribution $N(0, \sigma^2)$.

It is only reasonable to expect the sampling apparatus to perform measurements with an accuracy that is independent of the state being measured — at least over the dynamic range of the environmental events being observed. A Gaussian noise distribution is a standard assumption made by many successful parameter estimation techniques [Joh88].

⁴What is most important is additivity and zero mean. Normality is not strictly necessary. However, it is usually assumed in the extensive literature on connectionist principal components analysis, and so we make the same assumption here.

Figure 7.6: Gaussian Noise in the Velocity Estimates

7.3 Principal Manifold Approximation

We are now ready to introduce the Principal Manifold Approximation (PMA) method⁵ — our strategy for solving the manifold approximation problem under the assumptions of Section 7.2.1. Our intent in the present section is to give an abstract description of PMA. We wish to highlight the intuition behind the method without worrying about the details of a particular connectionist realization. We discuss these implementation issues in Chapter 8.

The first issue we consider is one of representation. How should the learning system represent the environmental foliation? We then turn to the problem of estimating the local tangent spaces at each of the sampled states. Finally, we examine how best to select a representation that is consistent with the local tangent space estimates and the assumptions of Section 7.2.1.

7.3.1 Vector Representations for PMA

There are many ways in which the learning system might represent the foliation that underlies the environment's behaviours. In this section we see that foliations can be represented by the elements of a vector space. We will elaborate on this scheme further in Section 7.3.3. As always, when choosing a representational scheme, we are concerned with issues of *completeness* and *consistency*. By completeness we mean that the scheme is able to represent every foliation that satisfies the assumptions of Section 7.2.1, and consistency implies that only those foliations can be represented.

As we saw in Chapter 3, a nonlinear foliation can be described⁶ by a differential geometric distribution, $\Delta(\mathbf{x})$, that maps each state $\mathbf{x} \in \mathcal{R}^n$ to the tangent space of the foliation leaf containing $\mathbf{x}$. Such distributions figure prominently in the theory of nonlinear control. Since we are ultimately interested in applying the results

⁵The name derives from our use of principal components in Section 7.3.2 to represent the tangent spaces of the manifolds in the environmental foliation.

⁶Alternatively, we can view the leaves of the foliation as maximal integrable submanifolds of $\Delta(\mathbf{x})$.

of learning to the control of connectionist networks. the learning system will express its estimate of the foliation in terms of a geometric distribution.

Perhaps the most obvious way to describe a smooth, nonsingular distribution is to specify a set of smooth vector fields that span it at each state in $\mathcal{U}$. i.e.

$$\Delta(\mathbf{x}) = \text{span} \{ \mathbf{f}_1(\mathbf{x}) \dots \mathbf{f}_d(\mathbf{x}) \}, \quad \forall \mathbf{x} \in \mathcal{U}.$$

However, we cannot guarantee that this type of description is consistent with the assumption of complete integrability without introducing additional constraints on the vector fields. We must ensure that the dimensionality of the distribution remains constant on $\mathcal{U}$ — therefore, linear independence of the fields must be assured for every $\mathbf{x} \in \mathcal{U}$. Equally important, and more difficult to satisfy, is the condition of mutual involutivity of the vector fields (see Appendix B). As for completeness, the distribution must agree with the tangent spaces at the sampled states and exhibit some degree of smoothness at other points. The following definitions introduce notions of completeness and consistency that are suited to these requirements.

Definition 7.2

A family of representations is PMA consistent if and only if every element will reconstruct a d -dimensional completely integrable distribution (of dimension $1 \leq d \leq n$) which is locally completely integrable on dynamic region $\mathcal{U}$.

Definition 7.3

A family of representations is PMA complete up to order q if and only if, for any collection of vectors $\mathbf{x}_i \in \mathbb{R}^n$ and d -dimensional subspaces $S_i \in \mathcal{S}$, $1 \leq i \leq q$, there is a representation that can be used to reconstruct a distribution $\hat{\Delta}$, such that

$$\hat{\Delta}(\mathbf{x}_i) = S_i, \quad 1 \leq i \leq q.$$

We must find a class of representations that are both PMA complete⁷ and

⁷We will abuse our terminology slightly by referring to PMA completeness without specifying the order. We will do this when the order is unimportant to the discussion or when it is obvious from context.

PMA consistent. The following theorems point the way.

Theorem 7.1

Let F_D be a family of smooth mappings from $\mathbb{R}^n$ to $\mathbb{R}^n$, parameterized by vectors from the q -dimensional vector space V . If, for any collection of vectors $\mathbf{x}_1, \dots, \mathbf{x}_q \in \mathcal{U}$ and d -dimensional subspaces $S_1, \dots, S_q \in \mathcal{S}$, $1 \leq d \leq n$, there is a mapping $\Phi \in F_D$ with Jacobian that acts as a surjective linear transformation

$$\left. \frac{\partial \Phi_r}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_i} (\mathbf{x}_i) : \mathbb{R}^d \oplus 0^{n-d} \mapsto S_i, \quad 1 \leq i \leq q.$$

then V is PMA complete up to order q .

Proof: This contradicts the assumption, and the theorem is proven. ■

Theorem 7.2

Let F_D be a family of smooth mappings from $\mathbb{R}^n$ to $\mathbb{R}^n$ parameterized by vectors from the q -dimensional vector space V . Then, if the members of F_D are locally diffeomorphic on dynamic region $\mathcal{U}$, then V is a PMA consistent family of representations.

Proof: The proof proceeds by contradiction. V is not PMA consistent iff it is not locally completely integrable on $\mathcal{U}$. Assume that this is the case. Let $\theta \in V$ and $\mathbf{h}(\mathbf{x}, \theta)$ be a member of F_D . By definition, $\mathbf{h}^{-1}(\mathbf{x}, \theta)$ is defined on the region

$$\mathcal{Z} = \mathbf{h}(\mathcal{U}, \theta),$$

and the respective Jacobians are written

$$\begin{aligned} \frac{\partial \mathbf{h}(\mathbf{x}, \theta)}{\partial \mathbf{x}} &= \begin{bmatrix} \mathbf{b}_1(\mathbf{x}, \theta) & \dots & \mathbf{b}_n(\mathbf{x}, \theta) \end{bmatrix}, \quad \text{and} \\ \frac{\partial \mathbf{h}^{-1}(\mathbf{x}, \theta)}{\partial \mathbf{x}} &= \begin{bmatrix} \omega_1(\mathbf{x}, \theta) \\ \vdots \\ \omega_n(\mathbf{x}, \theta) \end{bmatrix}. \end{aligned}$$

These Jacobians are both nonsingular, on $\mathcal{U}$ and $\mathcal{Z}$ respectively, by the Inverse Function Theorem. Therefore, the vector fields $\mathbf{b}_1(\mathbf{x}, \theta), \dots, \mathbf{b}_n(\mathbf{x}, \theta)$ are linearly independent at every $\mathbf{x} \in \mathbb{R}^n$, as are the co-vector fields $\omega_1(\mathbf{x}, \theta), \dots, \omega_n(\mathbf{x}, \theta)$. Any d columns

of $\frac{\partial \mathbf{h}(\mathbf{x}, \theta)}{\partial \mathbf{x}}$ must therefore span a locally finitely generated distribution Δ . We may consider only the first d columns, without loss of generality, since any other set can be so ordered by renumbering the axes. Linear independence of its basis ensures that Δ is nonsingular.

It must be the case that the gradients $\omega_{d+1}(\mathbf{x}, \theta) \dots \omega_n(\mathbf{x}, \theta)$ of functions $h_{d+1}^{-1}(x, \theta) \dots h_n^{-1}(x, \theta)$ span the co-distribution that annihilates Δ , since

$$\frac{\partial \mathbf{h}(\mathbf{x}, \theta)}{\partial \mathbf{x}} \mathbf{h}^{-1}(\mathbf{x}, \theta) = \mathbf{I}.$$

Frobenius' theorem (see Appendix B) tells us that a nonsingular distribution, which has an annihilator spanned by gradients, is also involutive and, therefore, completely integrable. Consequently, for any $1 \leq d \leq n$, any choice of parameter θ and $\mathbf{x} \in \mathcal{U}$ the distribution $\left[\Delta = \text{span} \left\{ \frac{\partial \mathbf{h}(\mathbf{x}, \theta)}{\partial x_1}, \dots, \frac{\partial \mathbf{h}(\mathbf{x}, \theta)}{\partial x_d} \right\} \right]$ is completely integrable. This contradicts the assumption that Δ is not locally integrable on $\mathcal{U}$, thereby proving the theorem. ■

So we see that the existence of a family of parameterized mappings that satisfy the theorems is sufficient to induce a set of representations that is both PMA consistent and PMA complete. Furthermore, this set has the structure of a vector space. This latter property is particularly desirable in connectionist implementations of PMA, where the representations need to be stored in long-term memory as synaptic weight vectors, or in short term memory as activation state vectors.

The following terminology will be useful in the sequel.

Definition 7.4

A family of PMA representations, V , is a set of representations that is both PMA consistent and PMA complete up to order q .

Definition 7.5

A family of parameterized mappings is said to induce a set of PMA representations V if and only if it satisfies Theorems 7.1 and 7.2.

Definition 7.6

We will say that a parameterized mapping $\Phi(\mathbf{x}, \theta)$ interprets a PMA representation θ if it can be used to uniquely define a completely integrable distribution from θ . Strictly speaking it is the algorithm that uses $\Phi(\mathbf{x}, \theta)$ to construct θ that performs the interpretation, but it will be convenient to speak of the mapping as the interpreter.

In Section 7.3.3.1.1 we will introduce a family of regularized radial basis mappings that induces a set of vector PMA representations. As we shall see, searching this set for a representation that is a good approximation to the observed velocity estimates reduces to the satisfaction of a system of linear constraints on the parameter vectors.

7.3.2 Local Approximation by Principal Components Analysis

It is our desire that the learning system find an approximation to the true distribution Δ which ‘locally’ agrees with the observations at each sampled state, $\mathbf{x}_i$. Therefore, we must first consider how to approximate these local tangent spaces. We are faced with two problems when attempting to find a local approximation from the velocity estimates: (i) sample sparseness, and (ii) noise in the estimates. As we shall see, both problems can be dealt with using the assumptions of Section 7.2.1.

7.3.2.1 Tangent Space Estimation for Non-Sparse Observations

Let us first consider approximating the tangent space at a single observation state, $\mathbf{x}_i$. Suppose for a moment, contrary to our assumption of sparse sampling, that we have a set of noisy velocity estimates that span the tangent space $\Delta(\mathbf{x}_i)$ of the foliation leaf at $\mathbf{x}_i$. The noise will introduce velocity components in all directions, thereby obscuring the true tangent space. However, if the signal to noise ratio is high enough, the variance in the velocity estimates will be greater in the ‘plane’ of the tangent space. We will use a statistical technique called Principal Components Analysis (PCA) to reconstruct an estimate of $\Delta(\mathbf{x}_i)$ from this difference in the variance

of the observed velocities.

Given a d -dimensional leaf at $\Delta(\mathbf{x}_i)$, the aim of PCA is to find a set of d orthonormal vectors in $\mathbb{R}^n$, called *principal components*, that account for as much of the variance in the velocity estimates at $\mathbf{x}_i$ as possible. The principal components are ordered such that the first is oriented in the direction of maximum variance, the second points in the direction of maximum variance within the subspace orthogonal to the first component, and so on. The i^{th} principal component is orthogonal to the previous $i - 1$, and represents the direction of maximal variance in the subspace orthogonal to that spanned by the previous components. We refer to the subspace spanned by the first i components as the i^{th} principal subspace.

More formally, using our assumption of additive zero-mean Gaussian estimation noise, we define $\mathbf{v}$ to be an n -dimensional random velocity vector. The covariance matrix of the velocity estimates is given by

$$\Sigma = E[\mathbf{v}\mathbf{v}^T].$$

where $E[\mathbf{z}] = \int p(\mathbf{z})\mathbf{z}d\mathbf{z}$ is the usual expectation operator. The following variation of the PCA theorem (see, for example, [Dec96]) characterizes the d principal components in terms of Σ :

Theorem 7.3

Let the velocity estimate covariance matrix Σ have eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ and respective orthonormal eigenvectors $\eta_1, \dots, \eta_n$. Then the j^{th} principal component $\mathbf{w}_j$ of velocity estimate vector $\mathbf{v}$ is the j^{th} eigenvector η_j .

Proof: The proof proceeds by induction on the principal component index j . As the base case, $j = 1$, consider the first principal component $\mathbf{w}_1$. This unit vector is, by definition, oriented in the direction of maximal variance of the velocity estimates at state $\mathbf{x}_i$. Since the covariance matrix is symmetric, it can be written in terms of its eigenvalues and eigenvectors as follows

$$\Sigma = \sum_{i=1}^n \lambda_i \eta_i \eta_i^T. \quad (7.3.1)$$

The variance of $\mathbf{v}$, measured in the direction of the first principal component, is therefore given by

$$\begin{aligned}
 \sigma_1^2 &= E \left[(\mathbf{v}_1^T \mathbf{w})^2 \right] \\
 &= E \left[\mathbf{v}_1^T \mathbf{w} \mathbf{w}^T \mathbf{v}_1 \right] \\
 &= \mathbf{w}_1^T \Sigma \mathbf{w}_1 \\
 &= \sum_{i=1}^n \lambda_i (\eta_i^T \mathbf{w}_1)^2.
 \end{aligned} \tag{7.3.2}$$

By inspection of (7.3.2), the variance along the first principal component is maximized when the principal component is coincident with eigenvector η_1 or its negation.

We will now proceed with the induction. So far, we have seen that Theorem 7.3 is true for the base case. For arbitrary principal component index j , we assume that it is also true for index $j - 1$. Since, by definition, the j^{th} principal component is orthogonal to the previous $j - 1$ components, we have

$$\eta_i^T \mathbf{w}_j = 0 \quad 1 \leq i \leq (j - 1).$$

Therefore, for the j^{th} unit magnitude principal component to maximize the variance in the subspace orthogonal to the $(j - 1)^{th}$ principal eigenspace, i.e.

$$\sigma_j^2 = \sum_{i=1}^n \lambda_i (\eta_i^T \mathbf{w}_j)^2,$$

we must have

$$\eta_i^T \mathbf{w}_j = \begin{cases} \pm 1 & \text{if } i \neq j, \\ 0 & \text{if } i > j. \end{cases}$$

This implies that the j^{th} principal component is the j^{th} unit eigenvector orthogonal to the previous $j - 1$ principal components, i.e.

$$\mathbf{w}_j = \eta_j.$$

The induction therefore succeeds and the proof is complete. ■

In other words, if it were possible to repeatedly sample the velocity at the same state $\mathbf{x}_i$, a good estimate of the tangent space $\Delta(\mathbf{x}_i)$ is given in the limit by the d^{th} eigenspace $\text{span}\{\eta_1, \dots, \eta_d\}$ of the covariance matrix Σ .

In what sense is this estimate ‘good’? It can be shown [Dec96] that the d^{th} principal subspace S , estimated from the velocity vectors, is the subspace that minimizes the Least Mean Squared Error (LSE)

$$E[|\mathbf{v} - \text{Proj}_S(\mathbf{v})|].$$

where $\text{Proj}_S(\mathbf{v})$ is the orthogonal projection onto S . In other words, the expected difference between the noisy velocity estimates and the nearest elements of S is minimized. In the limit, this estimate is equal to the true tangent space.

Although we will employ a connectionist technique to perform PCA in Chapter 8, many other well established algebraic methods can also be used to efficiently find the principal components from Σ (see, for example, [Dec96], [GVL96] or [Pre92]).

7.3.2.2 Tangent Space Estimation for Sparse Observations

We have seen that PCA is an effective method for dealing with noisy velocity estimates at a single point. However, PCA cannot be directly applied to the manifold approximation problem because velocity estimates are too sparsely sampled. Indeed, as we saw in Section 7.2.1.3, we cannot expect to have more than one velocity estimate at any given state. Consequently, the covariance matrix computed at that state will be singular, with one nonzero eigenvalue. The only principal subspace with nonzero variance will, therefore, be 1-dimensional regardless of the dimensionality of the tangent space — another example of the ambiguity of dimension due to sparse sampling.

In order to obtain enough information to reconstruct the tangent space at a given state, we will enrich the data set at the state by including weighted velocity

estimates from neighbouring states.

The relative magnitudes of the vectors in the PCA data set determine how much they contribute to the solution. This issue is known in PCA circles as the *scaling problem* [Dec96]. Vectors with small magnitude make a smaller contribution to the variation in the data set. Consequently, they have a smaller role to play in determining the orientation of the principal components. When the relative importance of the data items is unknown, the usual approach in the PCA literature is to give all vectors equal weight by normalizing their magnitudes. However, our confidence in the applicability of a velocity vector to the tangent space estimate at $\mathbf{x}$ diminishes with its distance from $\mathbf{x}$, and therefore the weights will also diminish in magnitude with distance.

This heuristic is justified by two of the assumptions in Section 7.2.1.3. The assumption of complete integrability tells us that neighbouring tangent spaces will have the same dimension and, therefore, can be meaningfully compared in the gap metric. The stiffness assumption tells us that the orientations of these subspaces in $\mathbb{R}^n$ will be similar, i.e. states within a Euclidean distance of r will differ by at most gap distance γ . As we see from the following theorem, we can expect PCA to yield a tangent space that is a good approximation of tangent spaces in this region.

Remark 7.1 *Let $\mathcal{B}(\Gamma) \subset S$ be a neighbourhood of gap radius γ centered at d -dimensional subspace $S \in S$. Let Π be a set of normalized vectors drawn with uniform random probability from the subspaces in $\mathcal{B}(\Gamma)$ and coloured with zero-mean Gaussian noise $N(0, \sigma^2)$. Let Σ_Π be the associated covariance matrix. Then the d^{th} principal subspace of Σ_Π is an element of $\mathcal{B}(\Gamma)$.*

Justification of the remark: [Kat82] tells us that for real symmetric matrices, such as a covariance matrix, the eigenspace of the matrix varies continuously with the entries of the matrix. The d^{th} principal subspace associated with samples drawn from two subspaces S_1 and S_2 must minimize the reconstruction error, and therefore, in the limit, the distance between vectors on S_1 and S_2 and their projection on the

principal subspace cannot exceed the gap distance between S_1 and S_2 . Similarly, to minimize the reconstruction error on samples drawn from subspaces in $\mathcal{B}(\Gamma)$ the eigenspace must also be an element of $\mathcal{B}(\Gamma)$. A rigorous proof would formalize these concepts, but in the interest of time we have omitted the details.

We introduce the following matrix valued function to compute the covariance matrix at a sample state $\mathbf{x}$ from the tuples, $(\mathbf{v}_i, \mathbf{x}_i)$, $1 \leq i \leq q$, in the learning system buffer:

$$\Sigma(\mathbf{x}_j) = \sum_{i=1}^q \rho(|\mathbf{x}_i - \mathbf{x}_j|) \mathbf{v}_i \rho(|\mathbf{x}_i - \mathbf{x}_j|) \mathbf{v}_i^T. \quad (7.3.3)$$

The radial basis function $\rho(\mathbf{v}_i, |\mathbf{x}_i - \mathbf{x}_j|)$ weights the velocity estimate $\mathbf{v}_i$ by an amount that depends (approximately) inversely on the distance between its sample state on the distance between its sample state $\mathbf{x}_i$ and $\mathbf{x}_j$. The choice of function ρ may vary from application to application. The essential characteristic of its profile is that it is radially symmetrical with maximum weight at $\mathbf{x}_j$ and negligible weight beyond radius of confidence r . Within this radius we must choose the scaling factor so that the amplitude of the sampling noise does not obscure the contribution of vectors from similar tangent spaces.

7.3.2.3 Determination of Dimensionality

Since PCA will find n orthonormal principal components for any diagonalizable covariance matrix, some ambiguity still remains regarding the dimensionality of the tangent space. We can either make an *a priori* assumption about the dimension, or use the variance along each principal component⁸ to resolve this ambiguity in a heuristic fashion.

We will use a simple heuristic based on the fact that normalization of the velocity vectors by weighting function ρ in equation (7.3.3) will make the variance of

⁸The variance along the normalized j^{th} principal component direction is given by $[\Sigma \mathbf{w}_j = \sigma_j^2 = \sum_{i=1}^n \lambda_i (\eta_i^T \eta_j) = \lambda_j]$. Therefore, the variance is equal to the eigenvalue associated with the principal component eigenvector.

the principal components roughly equal in directions that are almost co-planar with the true tangent space, but relatively small in orthogonal directions. Therefore, in the limit, it is reasonable to expect the variances to drop significantly at principle component index $d + 1$ if the true tangent space is d -dimensional.

Using the assumption of complete integrability, and therefore the consistent dimension of Δ on $\mathcal{U}$, we can strengthen our confidence in the estimated dimensionality by taking the median of dimension estimates over all of the estimated tangent spaces.

In the event that the estimated dimension shows significant variation across the sampled states, then either the distribution is not globally completely integrable and it violates assumption 7.2.1.3.1, or the confidence radius r is too large for the allowable error γ and the stiffness assumption is incorrect.

7.3.3 Global Approximation by Regularized Functions

Once we have estimated the tangent space $\Delta(\mathbf{x}_i)$ at each sample point $\mathbf{x}_i$, for $1 \leq i \leq q$, we must find a representation of Δ that has a ‘good fit’ with these estimates. This is not a well-posed problem, because there are an uncountable number of completely integrable distributions to choose from. We shall approach the problem from the perspective of multivariable function approximation ([PG96], [Pow87]). We begin by introducing the fundamental ideas behind approximation of scalar functions by regularization techniques. Then we apply these ideas to define a family of PMA representations and a method for selecting a particular representation that ‘best’ fits the tangent space estimates.

7.3.3.1 Surface Reconstruction from Sparse Data Points

Let us digress for a moment to consider the problem of reconstructing a scalar function φ with domain $\mathbb{R}^n$, from a sparse and finite sampling of function values at sample vectors $\mathbf{x}_1, \dots, \mathbf{x}_q$. This problem is underconstrained since there

is not enough information in the sample values to uniquely reconstruct the entire n -dimensional function surface. We need to make an *a priori* assumption that the mapping $\varphi : \mathbb{R}^n \mapsto \mathbb{R}$ is not entirely random. Such an assumption gives us enough knowledge to suggest a reasonable approximation to the mapping at vectors other than the sampled set. Without such structural knowledge the reconstruction cannot take place.

One of the weakest assumptions we can make is that the function is smooth. By this we mean that the value at one point depends on other values nearby. In fact, we may interpret the stiffness assumption of Section 7.2.1.3.2 as a smoothness constraint. Many techniques, under the heading *standard regularization*, have been developed to take advantage of smoothness constraints for function approximation problems. A variational approach is generally taken: a cost functional, which incorporates a smoothness constraint, is defined over a space of approximating functions, and then a search is made for the approximation that minimizes the functional ([PG96]). The goodness of fit of approximating function $\hat{\varphi}$ to the sample data is high when the cost is low, and vice versa. The functional, which has two terms, is defined as follows:

$$C(\hat{\varphi}) = \sum_{i=1}^q (\hat{\varphi}(\mathbf{x}_i) - \varphi(\mathbf{x}_i))^2 + \lambda \|D\hat{\varphi}\|^2. \quad (7.3.4)$$

The first term measures how well the approximating function $\hat{\varphi}$ agrees with the true function at each of the sample points. It is not enough to have a good match at these points, however. We must also take into account how well the approximation agrees with the smoothness constraint. This is the role of the second term. The measure of smoothness is given as a differential operator norm⁹. If the approximation has a large degree of variation from point to point, i.e. is not smooth, then the norm will be large. This value depends on the particular operator D .

Note that the functional incorporates a trade off between the smoothness and accuracy of the approximation. An approximation that does not exactly agree

⁹We will assume that the L^2 norm is used.

with the true function values may be ‘better’ than one that does if it is in better agreement with the smoothness assumption. Non-negative scalar λ controls the relative importance of smoothness versus accuracy. If the smoothness assumption does not agree with the true function, but the weight λ given to the assumption is large, then the approximation that minimizes the functional might not match the true values of the function.

In [PG96] Poggio and Girosi have shown¹⁰ that the solution to (7.3.4) lies in a q -dimensional subspace of the space of smooth functions, provided that the function $\varphi(\mathbf{x})$ does not lie in the null space of D . They show that a basis for the solution space is given by the q symmetric Green’s functions, $G(\mathbf{x}, \mathbf{x}_i)$, $1 \leq i \leq q$, of the self adjoint differential operator $\hat{D}D$ ¹¹. The i^{th} Green’s function is symmetric about the sample vector $\mathbf{x}_i$, so we say that the function is ‘centered’ at this point. A solution to the approximation problem can be expressed as a linear combination of these functions

$$\hat{\varphi}(\mathbf{x}) = \frac{1}{\lambda} \sum_{i=1}^q c_i G(\mathbf{x}, \mathbf{x}_i). \quad (7.3.5)$$

where the coefficients c_i , $1 \leq i \leq q$, are constant scalars. The coefficients of (7.3.5) are found by solving the following linear system of equations for the constant vector $\mathbf{c}$:

$$\mathbf{G}\mathbf{c} = \mathbf{y} \quad (7.3.6)$$

$$\Leftrightarrow \begin{bmatrix} G(\mathbf{x}_1, \mathbf{y}_1) & & G(\mathbf{x}_1, \mathbf{y}_q) \\ & \ddots & \\ G(\mathbf{x}_q, \mathbf{y}_1) & & G(\mathbf{x}_q, \mathbf{y}_q) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_q \end{bmatrix} = \begin{bmatrix} \varphi(\mathbf{x}_1) \\ \vdots \\ \varphi(\mathbf{x}_q) \end{bmatrix}. \quad (7.3.7)$$

The symmetry of the Green’s functions around their centers stems from the fact that $\hat{D}D$ is self-adjoint. If D is also translationally and rotationally invariant, then the

¹⁰A similar and more rigorous derivation than Poggio and Girosi’s is also given in [Wab90].

¹¹ $\hat{D}$ is the adjoint of D .

Green's functions will be Radial Basis Functions (RBFs) of the form

$$G(\mathbf{x}, \mathbf{y}_i) = G(|\mathbf{x} - \mathbf{x}_i|).$$

Micchelli [PG96] has shown that, in the case where the Green's functions are RBFs, the square matrix $\mathbf{G}$ will be nonsingular when the Green's functions are centered at the sample vectors, i.e. $\mathbf{y}_i = \mathbf{x}_i$. In this case (7.3.7) has a unique solution.

In the event that $\varphi(\mathbf{x})$ lies in the null space of D , the solution will exist outside of the subspace spanned by the Green's functions. In this case the above solution must be augmented by a term $\epsilon(x)$ which is an element of the null space of D [MI92]. Poggio and Girosi [PG96] have studied the case where the centers of the Green's functions are placed at points other than the sample vectors. They show that, given enough Green's functions distributed uniformly in the region containing the sample vectors, an arbitrarily accurate approximation of any continuous function can always be found. In other words, as the number of centers $q \rightarrow \infty$, $\epsilon(x) \rightarrow 0$ uniformly. Note that, in the event that the number of Green's functions does not match the number of sample points, matrix $\mathbf{G}$ will not be square. In this event, the Moore-Penrose pseudo-inverse [Str80], $\mathbf{G}^+$, can be used instead of $\mathbf{G}$. The resulting solution minimizes the functional's squared error term

$$\sum_{i=1}^q (\hat{\varphi}(\mathbf{y}_i) - \varphi(\mathbf{y}_i))^2. \quad (7.3.8)$$

7.3.3.1.1 Regularized PMA Representations We now use the method of standard regularization to define a family of representations that are both PMA consistent and PMA complete. As our approach is based on Theorems 7.1 and 7.2 of Section 7.3.1, we will first define a family of locally diffeomorphic parameterized mappings and then show that they induce the desired representation.

The following definitions and theorem characterize a set of vectors that can be used to represent environmental foliations.

Definition 7.7

The set of regularized PMA representations is defined as

$$V = \left\{ \mathbf{v} : \mathbf{v} = \text{Vec}(\theta^T) \text{ where } \theta \in \mathbb{R}^{n \times \hat{q}} \text{ such that } \theta \text{ is maximum rank} \right\}.$$

It is known [Boo86] that the set of maximum rank matrices associated with the elements of V forms a differentiable manifold in $\mathbb{R}^{n \times \hat{q}}$. Since V can be identified with this manifold, we will speak of V having this manifold structure.

Definition 7.8

The family of order- $\hat{q}$ regularized RBF mappings, F_{RM} contains mappings which satisfy the following conditions:

1. The elementary functions of $\Phi(\mathbf{x}, \theta) \in F_{RM}$ are standard regularization functions

$$\Phi_i(\mathbf{x}, \theta) = \sum_{j=1}^{\hat{q}} \theta_{i,j} G(\mathbf{x}, \mathbf{y}_j).$$

as discussed in Section 7.3.3.

2. Parameter matrix θ is maximum rank.
3. D is any translationally and rotationally invariant second order linear differential operator that is the same for all members of F_{RM} .
4. $G(\mathbf{x}, \mathbf{y})$ is the Green's function for operator D centered at $\mathbf{y}$ as described in Section 7.3.3.
5. $\hat{q} \geq n$ and the centers $\mathbf{y}_1, \dots, \mathbf{y}_{\hat{q}}$ are unique.
6. $\Phi(\mathbf{x}, \theta)$ is invertible on $\mathcal{U}$.

Theorem 7.4

The order- $\hat{q}$ family F_{RM} induces the set of regularized PMA representations V .

Proof: For any $\mathbf{v}$, let $\theta \in \mathbb{R}^n$ such that $\theta_i = [\mathbf{v}_{in+1} \cdots \mathbf{v}_{in+n}]$. We will refer to θ as the parameter matrix associated with $\mathbf{v}$. $\Phi(\mathbf{x}, \theta) \in FRM$ interprets θ as follows. Let the Jacobian of the mapping be

$$\frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{f}_1(\mathbf{x}) & \cdots & \mathbf{f}_n(\mathbf{x}) \end{bmatrix}.$$

The columns are of the form

$$\mathbf{f}_j(\mathbf{x}) = \begin{bmatrix} \sum_{i=1}^{\hat{q}} \theta_{1,i} \frac{\partial G(\mathbf{x}, \mathbf{y}_{1,i})}{\partial \mathbf{x}} \\ \vdots \\ \sum_{i=1}^{\hat{q}} \theta_{n,i} \frac{\partial G(\mathbf{x}, \mathbf{y}_{n,i})}{\partial \mathbf{x}} \end{bmatrix}, \quad 1 \leq i \leq \hat{q}, 1 \leq j \leq n \quad (7.3.9)$$

and are smooth vector fields on $\mathbb{R}^n$, because $G(\mathbf{x}, \mathbf{y}_{1,i}) \in C^\infty$. Therefore, the span of the first d of these vector fields defines a smooth distribution on $\mathbb{R}^n$

$$\hat{\Delta}(\mathbf{x}) = \text{span} \{ \mathbf{f}_1(\mathbf{x}) \dots \mathbf{f}_d(\mathbf{x}) \}. \quad (7.3.10)$$

Since this procedure works for any element of V , V satisfies the definition of the family of PMA representations. ■

So, the proof of Theorem 7.4 tells us how to interpret the representations in V . Next, we need to show that the regularized PMA representations are both PMA consistent and PMA complete. That this is true is the consequence of the next theorem, but first we must introduce two lemmas which provide conditions on the centers of the Green's functions that are important to the theorem.

Definition 7.9

A set P of $\hat{q}$ n – dimensional vectors is in regular position iff

1. $\hat{q} > n$
2. $\exists \mathbf{Q} = \{x_1, \dots, x_{n+1}\} \leq \mathbf{P}$ such that

$$(x_i - x_1) \notin \text{span}\{(x_2 - x_1), \dots, (x_{i-1} - x_1)\}, 2 \leq i \leq n+1$$

We shall refer to Q as the regular kernel of P . The following lemma gives the result that the regular position property is generic, i.e. any random choice of $\hat{q}$ vectors, P , is in regular position with respect to U .

Lemma 7.1

The regular position property is generic.

Proof: If we choose a set P of any $\hat{q} > n$ random points from $\mathbb{R}^n$, it is well known that they will be generically distinct and independent¹². Therefore, there will be a subset $Q = \{x_2, \dots, x_{n+1}\} \subseteq P$ of n points that form a basis for $\mathbb{R}^n$. If we translate these points by $-x_1$ such that $x_1 \in \theta$ but $x_1 \notin Q$ they will continue to be a basis for $\mathbb{R}^n$. Linear independence implies that each $(x_i - x_1) \in \text{span}\{(x_2 - x_1), \dots, (x_{i-1} - x_1), (x_{i+1} - x_1), \dots, (x_{n+1} - x_1)\}$ which in turn implies condition 2 of Definition 7.9. Since $\hat{q} > n$ and condition 1 of Definition 7.9 is also satisfied, we may conclude that a random set of vectors is in regular position with probability 1. ■

The importance of this definition is that it introduces a condition that is sufficient to ensure the generic PMA consistency and completeness of any set of regularized PMA representations for any given dynamic region U .

As we shall see, this depends on the fact that given any point $y \in U$ and the set of $\hat{q}$ vectors in regular position, $Y = \{(x_1 - y), \dots, (x_{\hat{q}} - y)\} \text{ span } \mathbb{R}^n$. That this is true is demonstrated by the next lemma.

Lemma 7.2

If P is in regular position, then $\forall y \in U \text{ span}\{(x_1 - y), \dots, (x_{\hat{q}} - y)\} = \mathbb{R}^n$.

Proof: Without loss of generality assume that the regular kernel of P is $Q = \{x_1, \dots, x_{n+1}\}$. The proof proceeds by induction on $x_i \in Q$ first. The inductive hypothesis shall be that $Y_i = \text{span}\{(x_1 - y), \dots, (x_i - y)\}$ has dimension greater than or equal to $(i - 1)$.

¹²Of course, we don't mean statistical independence. The question of linear independence is easily seen to be related to the 'next' random point falling exactly on the thin manifold containing the previous random points — an event with probability zero [Pap91].

Let $S_j = \text{span}\{(\mathbf{x}_2 - \mathbf{x}_1) \dots (\mathbf{x}_j - \mathbf{x}_1)\}$. Note that if $(\mathbf{y} - \mathbf{x}_1) \in Y_j$, then $S_j = Y_j$.

For the base case $i = 1$, this is trivially true since if $\mathbf{y} = \mathbf{x}_1$, then Y_i is 0-dimensional, otherwise it is 1-dimensional.

Assuming the hypothesis for case $i - 1$, we know that if $(\mathbf{y} - \mathbf{x}_1) \in S_{i-1}$ then $Y_{i-1} + (\mathbf{x}_i - \mathbf{y})$ is $(i - 1)$ -dimensional (i.e. the dimension of Y_{i-1} plus 1), because $\mathbf{x}_i \in S_{i-1}$ and $S_{i-1} = Y_{i-1}$.

Therefore, the induction succeeds and for $i = (n + 1)$, the size of the kernel Q , Y_{n+1} is n -dimensional and $Y_{\hat{q}} = \mathbb{R}^n$. ■

Now we need to show the connection between Lemma 7.2 and PMA consistency and completeness. The connection is based on the next lemmas which show that if the centers of $\hat{q}$ Greens' functions are in general position, then matrices important to consistency and completeness are guaranteed to be of maximum rank.

Lemma 7.3

The $\hat{q} \times n$ function matrix $\mathbf{G}(\mathbf{z}) = \left[\frac{\partial G(\mathbf{x}, \mathbf{y}_i)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{z}} \right]_i$ is generically maximum column rank for any $\mathbf{z} \in \mathbf{U} \subseteq \mathbb{R}^n$ and any random sampling of vectors $\mathbf{P} = \{\mathbf{y}_1 \dots \mathbf{y}_{\hat{q}}\}$, $\hat{q} > n$.

Proof: Since $\mathbf{P}$ is generically in regular position by Lemma 7.1, there exists a regular kernel of $\mathbf{P}$, $\mathbf{Q} = \{\bar{\mathbf{y}}_1 \dots \bar{\mathbf{y}}_{n+1}\}$, such that $\text{span}\{(\bar{\mathbf{y}}_1 - \mathbf{z}), \dots (\bar{\mathbf{y}}_{n+1} - \mathbf{z})\} = \mathbb{R}^n$ and therefore contains a basis for $\mathbb{R}^n$. Observe that each covector $\frac{\partial G(\mathbf{x}, \mathbf{y}_i)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{z}} = c_i(\bar{\mathbf{y}}_i - \mathbf{z})$, where $c_i \in \mathbb{R}$ since the gradient of an RBF is symmetrically radial. Consequently, the $\hat{q}$ rows of $\mathbf{G}(\mathbf{z})$ span an n -dimensional rowspace and $\mathbf{G}(\mathbf{z})$ is therefore maximum column rank. ■

Lemma 7.4

The $\hat{q} \times n$ function matrix $\mathbf{G}(\mathbf{z})$ in Lemma 7.3 is generically maximum row rank for any random collection of vectors $\mathbf{P} = \{\mathbf{y}_1 \dots \mathbf{y}_{\hat{q}}\}$, $\hat{q} > n$, on all $\mathbf{z}$ not on a thin $\hat{q} - 1$ -dimensional embedded linear submanifold of $\mathbb{R}^n$.

Proof: From Definition 7.9, we know that $\mathbf{S} = \{(\mathbf{y}_2 - \mathbf{y}_1), \dots (\mathbf{y}_{\hat{q}} - \mathbf{y}_1)\}$ is a $\hat{q} - 1$ -

$\hat{q}$ -dimensional subspace of $\mathbb{R}^n$. Clearly, if $z \in \mathbf{S}$, then $\text{span}\{(\bar{y}_1 - z), \dots, (\bar{y}_{\hat{q}} - z)\} = \mathbf{S}$ and it is $\hat{q} - 1$ -dimensional; otherwise it is $\hat{q}$ -dimensional. Since $\left. \frac{\partial G(\mathbf{x}, \mathbf{y}_i)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{z}} = c_i(\bar{y}_i - \mathbf{z})$, as in Lemma 7.3, the $\hat{q}$ rows of $\mathbf{G}(\mathbf{z})$ span a $\hat{q}$ -dimensional rowspace, i.e. full row rank, unless $\mathbf{z} \in \mathbf{S}$ which is a thin $\hat{q} - 1$ -dimensional submanifold of $\mathbb{R}^n$ as claimed. ■

Lemma 7.5

The $\hat{q} \times nq$ matrix $\hat{\mathbf{G}} = [\mathbf{G}(\mathbf{z}_1) \dots \mathbf{G}(\mathbf{z}_q)]$ is generically maximum row rank for any random sampling of vectors $\mathbf{P} = \{y_1, \dots, y_q\}$ and $\mathbf{Q} = \{z_1, \dots, z_q\}$, where $\hat{q} \leq nq$.

Proof: From Lemma 7.4 we have that each submatrix $\mathbf{G}(\mathbf{z}_j)$ is maximum row rank for any $z_j \in \mathbb{R}^n$ except those on a thin $\hat{q} - 1$ -dimensional linear subspace $\mathbf{S} \subset \mathbb{R}^n$. Observe that the property of $\mathbf{Q} \cap \mathbf{S} = \emptyset$ is generic since the probability of sampling a point z from a lower dimensional submanifold of $\mathbb{R}^n$ is 0. Therefore, each submatrix is generically maximum row rank for random sampling $\mathbf{Q}$. Consequently, $\hat{\mathbf{G}}$ is also maximum row rank. If this were not the case, then $\exists$ a row vector $\hat{\mathbf{G}}_j = \sum_{k \neq j} c_k \hat{\mathbf{G}}_k$ for some coefficients $c_k \in \mathbb{R}^n$. However, $\mathbf{G}(\mathbf{z}_i)_j = \sum_{k \neq j} c_k \mathbf{G}(\mathbf{z}_i)_k, 1 \leq i \leq q$ which contradicts the maximum row rank of $\mathbf{G}(\mathbf{z}_i), 1 \leq i \leq q$. ■

Lemma 7.6

The property of mapping a vector $\mathbf{V} \in \mathbb{R}^s$ to a vector $\mathbf{w} \in \mathbb{R}^r$ is not generic for transformations $\mathbf{T} \in \mathbb{R}^{r \times s}$.

Proof: Consider the element

$$w_i = \langle T_i, \mathbf{V} \rangle, \quad 1 \leq i \leq r \quad (7.3.11)$$

where the quantity $\langle \cdot, \cdot \rangle$ is an inner product of v with row T_i of matrix T . Therefore, it is necessary and sufficient that the covectors that satisfy (7.3.11) (i.e. as row i of T), all belong to an $s - 1$ -dimensional hyperplane (and *not* to an open region) of $\mathbb{R}^s$. Consequently, the probability of selecting a vector from this thin manifold is 0 for $1 \leq i \leq r$ when selecting a random $T \in \mathbb{R}^{r \times s}$. We may conclude that the

mapping property of the lemma is not a generic property for elements of $\mathbb{R}^{r \times s}$, as claimed. ■

Lemma 7.7

The $\hat{q}$ by nq matrix $\hat{\mathbf{G}}$ in Lemma 7.5 is generically maximum column rank for the same $\mathbf{P}$ and $\mathbf{Q}$ if $\hat{q} > nq$.

Proof: Lemma 7.3 tells us that all submatrices $\mathbf{G}(\mathbf{z}_k)$, $1 \leq k \leq q$ are generically maximum column rank. Therefore $\text{rank}(\hat{\mathbf{G}}) \geq n$. If this were not the case, then Lemma 7.3 would be contradicted for every submatrix. It may be the case, however, that the rank is as small as n . We prove that this is generically not the case by induction on the row index j for $\hat{\mathbf{G}}$. Observe, without loss of generality, that the first $j = \text{rank}(\hat{\mathbf{G}})$ rows of $\hat{\mathbf{G}}$ are linearly independent. We will assume the hypothesis for arbitrary row index $j < \hat{q}$. For $j + 1$, we must show that the hypothesis that the first $j + 1$ rows are linearly independent is true. Consider the submatrix $\mathbf{M}$ of $\hat{\mathbf{G}}$ containing the first j rows of $\hat{\mathbf{G}}$, i.e.

$$\mathbf{M} = [\mathbf{M}_1 \dots \mathbf{M}_q],$$

where row k of $\mathbf{M}$ is

$$\hat{\mathbf{G}}_k = [\mathbf{G}_k(\mathbf{z}_1) \dots \mathbf{G}_k(\mathbf{z}_q)].$$

By assumption $\mathbf{M}$ is maximum row rank j . If $\hat{\mathbf{G}}$ is rank j , then the $j + 1^{\text{th}}$ row $g = \hat{\mathbf{G}}_{j+1}$ must be an element of the rowspace of $\mathbf{M}$, i.e.

$$g = [g_1 \dots g_q] = c^T \mathbf{M} \quad \text{for some } c \in \mathbb{R}^j,$$

which implies that $g_i = c^T \mathbf{M}_i$. But this will not be the case generically because by Lemma 7.6, the randomly sampled centers $y_1 \dots y_{\hat{q}}$ and vectors $\mathbf{z}_1 \dots \mathbf{z}_q$ must reside on (a subset of) the thin linear submanifolds of $\mathbb{R}^n$ in order for the matrices $\mathbf{M}_i$, $1 \leq i \leq q$, to all map the coefficient vector c to the vectors $\mathbf{g}_i$, $1 \leq i \leq q$, respectively. We may therefore conclude that the first j rows of $\hat{\mathbf{G}}$ are generically

linearly independent. The induction carries through to $j = \hat{q}$. Since $\hat{q} > nq$, $\hat{\mathbf{G}}$ is generically maximum column row rank and the Lemma is proved. ■

We may now combine the results of Lemmas 7.5 and 7.7 to claim that $\hat{\mathbf{G}}$ is generically maximum rank.

Lemma 7.8

The $\hat{q} \times nq$ matrix $\hat{\mathbf{G}} = [\mathbf{G}(\mathbf{z}_1) \dots \mathbf{G}(\mathbf{z}_q)]$ is generically maximum rank for any random collection of centers $\mathbf{P}$ and sample points $\mathbf{Q}$.

Proof: Immediate by combining Lemmas 7.5 and 7.7. ■

Theorem 7.5

The set V of regularized PMA representations is PMA consistent and PMA complete up to order- $\hat{q}$ for $\hat{q} \geq nq$.

Proof: The proof proceeds in two parts. In the first we prove completeness, and in the second we prove consistency.

PMA completeness up to order- $\hat{q}$ follows from Theorem 7.1, which requires the Jacobians of the mappings in F_{RM} to be varied enough that a distribution exists which matches the tangent space estimates at each sampled state. Consider an element $\Phi(\mathbf{x}, \theta)$ of F_{RM} . Each elementary function $\Phi_i(\mathbf{x}, \theta)$ is a solution of the surface reconstruction problem of Section 7.3.3. It is an element of the space spanned by the $\hat{q}$ Green's functions that minimizes the functional (7.3.4) for the set of sample values. The Jacobian of $\Phi(\mathbf{x}, \theta)$ is given by

$$\frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{g}_1(\mathbf{x}) \\ \vdots \\ \mathbf{g}_{\hat{q}}(\mathbf{x}) \end{bmatrix}, \quad \text{where} \quad (7.3.12)$$

$$\mathbf{g}_i(\mathbf{x}) = \sum_{j=1}^{\hat{q}} \theta_{i,j} \frac{\partial G(\mathbf{x}, \mathbf{y}_{i,j})}{\partial \mathbf{x}}. \quad (7.3.13)$$

Notice that the i^{th} covector $\mathbf{g}_i(\mathbf{x})$ is the gradient of the function $\Phi_i(\mathbf{x}, \theta)$. Equation (7.3.13) therefore permits us to exactly approximate any covector field that is the

gradient of a potential function in the solution space spanned by the Green's functions¹³. Given covectors $\mathbf{v}_i$, which represent samples of the gradient of a function $\Phi_i(\mathbf{x}, \theta)$ sampled at the states $\mathbf{x}_i$, $1 \leq i \leq q$, we can construct an approximation to $\Phi_i(\mathbf{x}, \theta)$ and its associated gradient by solving the following system of linear equations for the $\hat{q}$ – dimensional covector θ_i ,

$$\theta_i \mathbf{G} = \mathbf{v}_i, \quad (7.3.14)$$

where

$$\mathbf{G} = \begin{bmatrix} \left. \frac{\partial G(\mathbf{x}, \mathbf{y}_1)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_1} & \dots & \left. \frac{\partial G(\mathbf{x}, \mathbf{y}_1)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_q} \\ \vdots & \ddots & \vdots \\ \left. \frac{\partial G(\mathbf{x}, \mathbf{y}_q)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_1} & \dots & \left. \frac{\partial G(\mathbf{x}, \mathbf{y}_q)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_q} \end{bmatrix} \quad (7.3.15)$$

$$\mathbf{v} = [\mathbf{v}_{i,1} \dots \mathbf{v}_{i,q}] \quad (7.3.16)$$

This system has an exact solution whenever $\mathbf{G}$ has maximum rank and $\hat{q} \geq nq$. In this case, the Moore-Penrose pseudo-inverse [Str80] of $\mathbf{G}$ is given by

$$\mathbf{G}^+ = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \quad (7.3.17)$$

and the minimum norm solution is

$$\theta_i = \mathbf{v}_i \mathbf{G}^+. \quad (7.3.18)$$

Now, suppose that we have used PCA to estimate the tangent spaces at the sampled states. Let the tangent space estimate at state $\mathbf{x}_i$ be given by the span of the first d orthogonal principal components

$$\mathbf{T}_i = \text{span} \{ \mathbf{w}'_1, \dots, \mathbf{w}'_d \} \quad (7.3.19)$$

¹³This observation has also been made by Mussa-Ivaldi in [MI92]. Our approach differs from his, however, in that we are concerned with the approximation of locally completely integrable distributions, while he restricts his attention to the approximation of individual vector fields with a solenoidal (non-gradient) component.

and let its orthogonal complement be given by the remaining components

$$T_i^\perp = \text{span} \{ \mathbf{w}'_{d+1} \cdots \mathbf{w}'_n \}. \quad (7.3.20)$$

Then we may express the problem of finding a $\Phi(\mathbf{x}, \theta) \in F_{RM}$ with Jacobian spanning these tangent spaces as n covector approximation problems of the form (7.3.18), where

$$\mathbf{v}_{i,j} = [\mathbf{w}'_{1,j} \cdots \mathbf{w}'_{n,j}].$$

Therefore, as long as $\mathbf{G}$ has maximum rank, we can always find an element of F_{RM} that matches any tangent space estimates, and the definition of PMA completeness up to order- q is satisfied.

When can we expect $\mathbf{G}$ to have maximum rank? Lemma 7.7 tells us that this will be true with certainty for any random selection of centers, and is guaranteed to be true on $\mathcal{U}$ if the centers are in general form. Therefore, under these conditions on F_{RM} , equation (7.3.18) tells us that we can always find a $\Phi(\mathbf{x}, \theta)$ that satisfies the requirements of Theorem 7.1. Hence, the vector representation V induced by F_{RM} is PMA complete.

The second part of the proof concerns consistency. Recall that Theorem 7.2 tells us that, in order to show PMA consistency, it is sufficient that we show that every element in F_{RM} is locally diffeomorphic on dynamic region $\mathcal{U}$. Lemma 7.3 also tells us that the Jacobian of an element of F_{RM} is nonsingular on $\mathcal{U}$. Therefore, by the inverse function theorem, we know that elements of F_{RM} satisfy Theorem 7.2, and V must be PMA consistent.

Since V has been shown to be both PMA complete and PMA consistent under the conditions given in Theorem 7.5, the proof is complete. ■

As a consequence of the previous proof we see that with certainty there is a unique solution to each row-wise problem expressed by equation (7.3.18) if $\hat{q} = nq$. Otherwise, we can use the Moore-Penrose pseudo-inverse of the matrix $\mathbf{G}$ to find a

solution with minimum Euclidean norm $|\theta_i^T|$. In the event that $\hat{q} < nq$, the Moore-Penrose pseudo-inverse can also be used to find a least squares approximation that minimizes the squared error term (7.3.8) (i.e. the first term of functional (7.3.4)). Theorem 7.5 also ensures that the mapping $\Phi(\mathbf{x}, \theta)$ is a diffeomorphism in the regions around the sampled states, i.e. the dual co-distribution spanned by the row-wise approximated co-vector fields is linearly independent. Consequently, complete integrability is guaranteed on $\mathcal{U}$. Also, notice that, because $d < n$ for any nontrivial foliation¹⁴, all θ that agree with *any* d -dimensional bases that span the d^{th} principal subspaces are equivalent.

Our final task in this section is to provide a method for finding a representation from V that is a ‘good’ approximation to the true distribution Δ on $\mathcal{U}$. Before we do so, we should clarify what we consider to be a good approximation.

Definition 7.10

Let the PCA tangent estimates at each sample point be given by

$$T_i \text{span} \{ \mathbf{b}_1^i, \dots, \mathbf{b}_n^i \} \quad \text{at state } \mathbf{x}_i, \quad 1 \leq i \leq q.$$

An element of the set of PMA representations $\mathbf{v}$ is a good PMA approximation of the true distribution Δ if and only if it minimizes the squared error norm between the spanning vectors of $\Delta(\mathbf{x}_i)$, the estimated basis for T_i (subject to the assumptions of Section 7.2.1).

The following theorem tells us that, under mild conditions on the centers of F_{RM} , a representation which is a good PMA approximation can be found by solving the system of linear equations that we encountered in the proof of Theorem 7.5.

Theorem 7.6

Given $\hat{q} \geq nq$ Green’s function centers in regular position, a regularized PMA representation that is a good PMA approximation for the observed velocity estimates is

¹⁴When $d = n$ there is only one trivial leaf in the partition, namely, $\mathbb{R}^n$.

given by solving the following n systems of linear equations:

$$\theta_i = \mathbf{v}_i \mathbf{G}^+. \quad (7.3.21)$$

$$(7.3.22)$$

$$\mathbf{v}_i = [\mathbf{w}_{1,1}^i \cdots \mathbf{w}_{n,1}^i \cdots \mathbf{w}_{1,q}^i \cdots \mathbf{w}_{n,q}^i]. \quad (7.3.23)$$

and $\mathbf{G}^+$ is given by equation (7.3.17) of Theorem 7.5.

Proof: The proof is immediate from the proof of Theorem 7.5 in which it was shown that equations 7.3.22 and 7.3.23 minimize the square error norm in Definition 7.10. under the conditions on the theorem. ■

7.3.4 Summary of Principal Manifold Approximation

The reader may find the following algorithmic description of the PMA method helpful. It should be understood, however, that many realizations of the method are possible. This description is primarily useful as a summary of the major subproblems that any realization of PMA must address. How this is done, and in what order, is up to the particular implementation. Indeed, in the next chapter we describe a connectionist implementation that performs all of these steps in parallel.

- Input: (i) Description of $\mathcal{U}$. (ii) q observed sample tuples received from sampling apparatus.
 - Output Parameter vector θ .
1. Initialization: choose qn centers for Green' functions by uniform distribution on $\mathcal{U}$.
 2. For each tuple $(\mathbf{x}_i, \mathbf{v}_i)$ in the learning system buffer:
 - (a) Compute the local weighted covariance matrix $\Sigma(\mathbf{x}_i)$.

- (b) Compute the n principal components $\rho_1, \dots, \rho_n$.
- (c) Compute the local estimate of dimension, d_i .
- 3. Compute the global estimate of dimension as $d = \text{Median}(d_1, \dots, d_q)$.
- 4. Solve the linear equations (7.3.18) of Theorem 7.5 to find θ .

PMA can be efficiently computed (i.e. polynomial time) by a non-connectionist implementation of the above algorithm¹⁵. For example, we have the following complexity estimates for the PMA subtasks:

- 1. q flops: Sample collection.
- 2. $\frac{4}{3}n^3q$ flops: Computation of the PCA tangent space estimates by Golub's Eigenspace Algorithm [GVL89].
- 3. $4n\hat{q}^2 + 8\hat{q}^3$ flops: Computation of $\mathbf{G}^+$ by Golub-Reinsch SVD (see Section C.3.3.6 of Appendix C and [GVL89]).
- 4. $n\hat{q}$ flops: Computation of θ by matrix multiplications.
- 5. nq flops: Computation of the median distribution dimension.

This yields a total complexity of the order $O(4n\hat{q}^2 + 8\hat{q}^3 + \frac{4}{3}n^3q)$.

7.4 Chapter Conclusions

We have introduced a novel method for the approximation of nonlinear foliations from sparse velocity estimates in this chapter. The method is called Principal Manifold Approximation (PMA), and it is suitable for implementation in either a connectionist or a symbolic setting. PMA will form the basis for a connectionist

¹⁵Note that our purpose here is only to show the existence of a polynomial time solution. The algorithm is illustrative only, and we do not claim that it is optimal.

algorithm that learns nonlinear manifold constraints. This connectionist implementation is presented in the next chapter. The material in this chapter is presented without reference to connectionist concerns so that the reader can more easily develop intuition for the underlying ideas. The work on PMA presented in this chapter is captured in papers [Gra98k, Gra98j].

Chapter 8

Connectionist Principal Manifold Approximation

In the previous chapter we introduced the Principal Manifold Approximation (PMA) method as a general technique for solving the manifold approximation problem. Now that we have given an abstract description of PMA, our goals are two-fold: first, to describe a particular implementation of PMA which has a connectionist interpretation; and second, to show how these results can be applied to learn and enforce nonlinear manifold constraints in general GNF networks. The resulting hybrid connectionist/control systems are a class of holonomic networks.

The weight update method that we introduce is an unsupervised batch learning algorithm. As such, it assumes that all velocity estimates are simultaneously available for training. We begin by showing in Section 8.2 that the regularized RBF mappings can be realized as connectionist networks that interpret synaptic weight vectors as regularized PMA representations. Then we derive a weight update rule in Section 8.2.3 that has weight space equilibria at exactly those weight vectors that encode good PMA approximations. This rule is presented as a continuous-time ODE which implements most of the PMA algorithm steps of Section 7.3 in parallel. We

discuss extensions to the method in Chapter 9.

In the second part of the chapter we show how the connectionist PMA learning method can be applied to the problem of actually enforcing the learned nonlinear manifold constraints in the run-time dynamics of continuous-time networks. In Section 8.3 we describe a novel family of recurrent networks, called Holonomic Networks, that are able to learn and apply manifold constraints to their output-driven dynamics. Then, in Section 8.5, we show how GNF network neurons can be combined with the PMA method to restrict the dynamics of the network to the manifold constraints. Our approach is to generalize the control strategy discussed in Chapter 7. Finally, we close with general discussion and conclusions in Section 8.6. We will consider possible extensions to the work in this chapter in Chapter 9.

We describe a novel class of Holonomic networks with zero drift dynamics that can be steered to any point on a learned constraint manifold by appropriate input signals.

We also show how to enforce a manifold constraint on a nonholonomic GNF network. Our approach is to generalize the control strategy discussed in Chapter 6. In Section 8.5 we show how GNF network neurons can be combined with the PMA method to restrict the dynamics of the network to the manifold constraints. Finally, we close with general discussion and conclusions in Section 8.6.

8.1 Contributions of the Chapter

This chapter represents the culmination of our search for a connectionist system that is compatible with the manifold constraint paradigm. The first major contribution presented here is an unsupervised, batch weight-update algorithm, called BR-PMA, capable of learning a partition of activation space into general nonlinear manifold constraints, from intermittent noisy observations of domain behaviours. Consequently, this algorithm can also be applied to the simpler problem of learning any finite number of non-overlapping nonlinear constraint manifolds. The tangent

bundle that is learned is a novel run-time control structure that is a continuous-time dynamical analog to stored program control. Our approach is to use local tangent space dynamics to drive the global weight space dynamics. This technique is of interest in its own right, since it permits us to generalize theoretical results for the local dynamics to the global setting. We are able to offer a proof that the only stable fixed points of the BR-PMA algorithm are correct learning solutions, and therefore, our algorithm does not suffer from the problem of local minima. Furthermore, it is guaranteed to produce a good estimate of the foliation in a local neighbourhood of each observed domain state. We are also able to show that convergence will occur from any initial value with probability 1. Finally, the BR-PMA algorithm has the desirable property that any degree of accuracy can be achieved if enough hidden neurons are added to the network and we are willing to observe the environment long enough.

In addition to the learning algorithm, we describe a modified feed-forward network architecture, called a PMA network, that supports BR-PMA learning. This is a shared weight architecture¹ that is a modified form of RBF network. We view PMA nets as a connectionist module that may be incorporated into fully connectionist or hybrid connectionist/control systems to support manifold constraint learning.

Our second major contribution is to describe a new class of holonomic connectionist architectures that support both manifold constraint learning and a form of classical fixed-point learning. The recurrent output layer consists of continuous-time neurons with decay rates determined by linear combinations of the inputs and the outputs of a PMA network. We refer to these as holonomic nets. We also describe a holonomic hybrid connectionist/control system that employs PMA networks to extend the results of Chapter 6 to include nonlinear manifolds. The evolution equations of these systems are non-local, however, their close relationship with GNF networks permits them to retain many of the desirable properties of connectionist systems.

¹Other shared weight architectures have been studied by A. Waibel [Ben96] and Y. LeCun [Lec90].

We also describe extensions to the BR-PMA algorithm. These include a competitive version of PMA that learns to represent than one state space foliation, and a discrete-time online version of the BRMPA algorithm.

8.2 Connectionist Architectures for PMA

The class F_{RM} of regularized RBF mappings (introduced in Chapter 7) can be expressed as the family of networks illustrated in Figure 8.2(a). The network consists of an input layer, which presents input vector $\mathbf{x}$, a hidden layer of $\hat{q}$ neurons with RBF receptive fields centered at the n -dimensional state vectors $\mathbf{y}_1, \dots, \mathbf{y}_{\hat{q}}$, and an output layer of n linear summation neurons that output the values of the elementary functions $\Phi_i(\mathbf{x}, \theta)$, $1 \leq i \leq n$. The representation θ parameterizing $\Phi(\mathbf{x}, \theta)$ is implemented as a matrix of weights on connections between the hidden and output layers. For simplicity, the connections from the input layer are assumed to have unit weights. An extensive literature exists in which variations on this type of feed-forward network have been studied (see, for example, [PG96]). For reasons that shall become apparent in Section 8.5.1, we shall refer to these networks as *constraint linearizing networks*.

Figure 8.2(b) depicts a network that can be thought of as an interpreter for a regularized PMA representation θ . The network implements the columns of the Jacobian

$$\frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{f}_1(\mathbf{x}) & \cdots & \mathbf{f}_n(\mathbf{x}) \end{bmatrix},$$

where

$$\mathbf{f}_j(\mathbf{x}) = \begin{bmatrix} \sum_{i=1}^{\hat{q}} \theta_{1,i} \frac{\partial G(\mathbf{x}, \mathbf{y}_{1,i})}{\partial \mathbf{x}} \\ \vdots \\ \sum_{i=1}^{\hat{q}} \theta_{n,i} \frac{\partial G(\mathbf{x}, \mathbf{y}_{n,i})}{\partial \mathbf{x}} \end{bmatrix}, \quad 1 \leq j \leq n.$$

We will refer to this type of network as a *regularized PMA net*. An input layer presents the environmental state $\mathbf{x} \in \mathbb{R}^n$ to n populations of hidden neurons. Each

Figure 8.1: Constraint Linearizing Network

Figure 8.2: Regularized PMA Network

population consists of $\hat{q}$ neurons with a particular type of receptive field. Neurons in the i^{th} pool have the receptive field

$$\frac{\partial G(\mathbf{x}, \mathbf{y}_{n,j})}{\partial x_i}$$

centered at the states $\mathbf{y}_j$, $1 \leq j \leq \hat{q}$. The output layer is also divided into n populations — in this case, of linear summation neurons — that are fully connected with a corresponding population in the hidden layer. The representation is again loaded into the network as long-term synaptic weights between the hidden and output layers of each population. The reader is encouraged to verify that the i^{th} output population implements the i^{th} vector field $\mathbf{f}_i(\mathbf{x})$ of Jacobian (8.2).

The learning algorithm that we develop in the following sections is a batch method² that computes the values of representation θ from a ‘buffer’ of velocity estimates.

8.2.1 Connectionist PMA Learning

There has been considerable interest in developing connectionist implementations of the PCA algorithm over the last 15 years. The method that we develop in this chapter can be thought of as extending work in this area to the general nonlinear setting. Several discrete-time connectionist PCA algorithms have been proposed and analysed which learn principal components, (or in some cases, an arbitrary basis), for the principal subspace. Broadly speaking, research in this area has followed two paths. The first line of investigation is best represented by the work of Oja [Oja82, Oja85, Oja92], Williams [Wil85], Sanger [San89], Foldiak [Fol89], Rubner and Tavan [RT89], Hornik and Kuan [HK92] and Plumbley [Plu95]. Most of the algorithms investigated are based on Oja’s seminal paper [Oja82] and make use of the theory of stochastic approximation [KC78] for their stability analyses. We shall refer

²The use of batch learning methods is not unusual for connectionist systems. For example, the perceptron [MP69, Ros58], LMS [WH60], and back-propagation [RHW86] algorithms, to name a few, were originally conceived and analysed as batch algorithms.

to these as Oja-style algorithms. The second approach is motivated by information theoretic concerns, and stems from the important work of Plumbley [Plu91, Plu93]. Extensive contributions have also been made by Deco and Obradovic [Dec96].

Our approach, as presented here, is a logical continuation of the first line of investigation. It is, however, also compatible with an information theoretic point of view, although we do not consider this topic any further here. Before proceeding with the development of a connectionist PMA algorithm, we need to understand the basic ideas underlying the Oja-style algorithms. This is the subject of the next section.

8.2.2 Oja-Style PCA Algorithms

PCA is profitably viewed as a type of unsupervised feature extraction technique. Considered from this viewpoint, the goal is to find a transformation that compresses an n -dimensional input vector $\mathbf{x}$ into a d -dimensional output vector $\mathbf{y}$ (where $d < n$) such that as much information as possible is preserved about the original inputs. Typically, we assume that $\mathbf{x}$ is a zero mean stochastic vector. In the event that the transformation is linear we have

$$\mathbf{y} = \mathbf{W}\mathbf{x}, \quad (8.2.1)$$

where $\mathbf{W}$ is a $d \times n$ matrix. As we have already seen in Section 7.3.2.1 of Chapter 7, it can be shown that the linear transformation that minimizes the least mean squared reconstruction error

$$E [||\mathbf{W}\mathbf{x} - \mathbf{y}||],$$

is the matrix $\mathbf{W}$ with rows that span the same subspace of $\mathbb{R}^n$ as the eigenvectors associated with the d largest eigenvectors of the covariance matrix

$$\Sigma = E [\mathbf{x}\mathbf{x}^T].$$

These rows are, of course, the first d principal components.

Figure 8.3: Regularized PMA Network

The PCA network of Figure 8.3 is a direct implementation of the linear compression transformation (8.2.1). The input vector is presented at the first layer, and the output layer neurons compute the (unthresholded) sum of the input signals modified by the synaptic weights. The weights on inputs to the i^{th} output neuron correspond to the i^{th} row of $\mathbf{W}$. Learning algorithms for this network must ensure that the row-wise weight vectors either converge to the principal components or to eigenvectors that span the d^{th} principal subspace.

An early discrete-time weight update algorithm of Oja, called the constrained Hebbian learning rule [Oja82], introduced a method for finding the first principal component with a one neuron PCA network. This work was quickly generalized into a variety of algorithms for finding all n principal components³. The rule for updating the weight matrix at the t^{th} time step has the same characteristic form for all of these algorithms, namely

$$\Delta \mathbf{W}_t = \alpha_t (\mathbf{y}_t \mathbf{x}_t^T - \mathbf{K}_t \mathbf{W}_t). \quad (8.2.2)$$

The first term, which is a type of Hebbian learning common to all of the algorithms we consider, performs gradient ascent on an energy function that maximizes the sum of output variances. In other words, this term pushes the rows of $\mathbf{W}$ toward the principal components. The main difference between the various Oja-style algorithms occurs in the second term — matrix $\mathbf{K}_t$. This matrix is responsible for maintaining the linear independence of (and otherwise conditioning) the evolving rows of $\mathbf{W}$. It is usually a function of $\mathbf{y}_t$ and $\mathbf{W}_t$. For example, in the Symmetric Error Correction

³Variations on this basic approach have also been proposed to give the network a greater degree of biological plausibility (for example, [Fol89] and [RT89]). However, since these algorithms are consistent with our basic approach, and we are less concerned with biological plausibility in this thesis, we will not consider them further here.

algorithm of Williams [Wil85], it is given by

$$\mathbf{K}_t = \mathbf{y}_t \mathbf{y}_t^\top.$$

and Sanger's [San89] Generalized Hebbian Algorithm uses⁴

$$\mathbf{K}_t = \text{Lower}(\mathbf{y}_t \mathbf{y}_t^\top).$$

Usually, Kushner and Clark's theory of stochastic approximation [KC78] is used to analyse the stability characteristics of these algorithms. The theory seeks to define a continuous-time ODE that represents the expected dynamics of the discrete-time weight matrix. Kushner and Clark give conditions that are easily met for which the trajectories of the discrete-time system will converge with probability 1 to the equilibria of the continuous-time ODE. It is usually much easier to analyze the behaviour of this ODE than to try to directly analyze the stochastic weight update rule. In particular, Hornik and Kuan [HK92] have found that the ODE associated with (8.2.2) is given by

$$\frac{d}{dt} \mathbf{W} = \mathbf{W} \Sigma - \mathbf{K} \mathbf{W}, \quad (8.2.3)$$

where $\mathbf{K} = E[\mathbf{K}_t]$. We are particularly interested in this analytical approach because it provides us with a useful connection between the discrete-time world of connectionist PCA algorithms and the continuous-time world of GNF networks. We will further develop this connection in the proofs of our main results presented in the next section.

8.2.3 Derivation of Batch PMA Learning Rule

Recall from Chapter 7 that the class $\mathcal{V}$ of regularized PMA representations is on an open manifold of the space $\mathbb{R}^{n \times q}$. We will now define an ODE that evolves on this manifold from an initial representation to a stable representation θ that is a good PMA approximation in the sense of Section 7.3. We will see that basins of

⁴Lower($\mathbf{M}$) sets the entries of $\mathbf{M}$ above the diagonal to 0.

attraction of the good PMA approximations cover all but a ‘thin’ submanifold of V and, therefore, any initial representation will evolve to a good approximation with certainty.

Before we present our main theorems, we wish to describe the intuition behind the learning rule. As we follow a trajectory θ_t through V , there is a corresponding continuous-time trajectory that evolves through the space of regularized RBF mappings F_{RM} ⁵. For any function $\Phi(\mathbf{x}, \theta_t)$ on this trajectory, we consider the vector fields that form the columns of the Jacobian

$$\frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{f}_1(\mathbf{x}) & \cdots & \mathbf{f}_n(\mathbf{x}) \end{bmatrix}$$

to be a nonlinear generalization of the concept of principal components. Rather than specifying a single set of principal components in $\mathbb{R}^n$, these *principal fields* specify principal components in a linear n -dimensional tangent space of $\mathbb{R}^n$ anchored *at every state in $\mathbb{R}^n$* . From the standpoint of the PMA algorithm in Chapter 7, it is the principal components specified by the field at each sample state, i.e.

$$\mathbf{f}_1(\mathbf{x}_i) \dots \mathbf{f}_n(\mathbf{x}_i), 1 \leq i \leq q, \quad (8.2.4)$$

that are important. The ODE trajectory in V should be such that these values of the principal fields converge to the local principal components specified by the local, weighted velocity estimates. Recall that the covariance matrix for these weighted estimates is given by equation (7.3.3).

The reader is asked to picture q sets of d vectors — representing the values of the fields (8.2.4) at the q sampled states — independently converging to the local principal components. We have, in effect, q continuous-time connectionist PCA algorithms executing in parallel. Locally, *any* continuous-time dynamics with stable equilibria corresponding only to valid solutions to the local PCA problems will

⁵This is the q -dimensional subspace of Hilbert space that is spanned by the basis of q Green’s functions, as described in Section 7.3.3.

suffice⁶. So, the local dynamics are, in a sense, used to drive the global dynamics. The dynamics at every other state are guaranteed to be ‘sensible’ because the representation dynamics are confined to the PMA consistent representations in V . The local dynamics can be derived from the classic connectionist PCA dynamics by the method of stochastic approximation. Indeed, we will employ equation (8.2.3) as the equations of motion at each sampled state. However, any other equations which have the same equilibria could just as easily have been used.

Our task now is to formalize this intuitive picture. We begin by defining the connectionist batch PMA method as follows:

Definition 8.1

The batch regularized PMA (BR-PMA) learning algorithm involves the following steps:

1. Initialization: Pick an initial $\theta \in V$ at random (e.g. $\theta = [\mathbf{I}^{n \times n} \mathbf{0}^{n \times (q-n)}]$).
2. Weight Update: Update θ continuously according to the ODE

$$\begin{aligned} \frac{d}{dt}\theta &= \begin{bmatrix} \frac{d}{dt}\theta_1 \\ \vdots \\ \frac{d}{dt}\theta_n \end{bmatrix} \quad \text{where} \\ \frac{d}{dt}\theta_i &= [\gamma_i(\mathbf{x}_1) \cdots \gamma_i(\mathbf{x}_q)] \mathbf{G}^+, \\ \gamma_i(\mathbf{z}) &= \sum_{r=1}^n \sum_{s=1}^q \left[\Sigma_{i,r}(\mathbf{z}) \theta_{r,s} \frac{\partial G(\mathbf{x}, \mathbf{y}_s)}{\partial \mathbf{x}_b} \Big|_{\mathbf{x}=\mathbf{z}} - \mathbf{K}_{b,r}(\mathbf{z}) \theta_{i,s} \frac{\partial G(\mathbf{x}, \mathbf{y}_s)}{\partial \mathbf{x}_r} \Big|_{\mathbf{x}=\mathbf{z}} \right]_{b=1 \dots n}. \end{aligned} \quad (8.2.5)$$

and $\mathbf{K}(\mathbf{z})$ is any $d \times d$ matrix function such that

$$\mathbf{K}(\mathbf{z}) + \mathbf{K}(\mathbf{z})^T = 2\mathbf{W}(\mathbf{z}, \theta) \Sigma(\mathbf{z}) \mathbf{W}(\mathbf{z}, \theta)^T. \quad \text{and} \quad (8.2.7)$$

$$\mathbf{W}(\mathbf{z}, \theta) = \left(\frac{\partial \Phi(\mathbf{x}\theta)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{z}} \begin{bmatrix} \mathbf{I}^{d \times d} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \right)^T. \quad (8.2.8)$$

⁶As we shall see in Theorem 8.1, dynamics which ensure that the local vectors remain linearly independent while evolving to the PCA solution are preferred, because they ensure that θ remains in the set V and is therefore PMA consistent and PMA complete for all $t > 0$.

$\Sigma(\mathbf{x})$ and $\mathbf{G}^-$ are given by equations (7.3.3) and (7.3.17) respectively.

We begin by characterizing the dynamics of the approximating distribution $\hat{\Delta}$. We find that the local dynamics of the principal fields that span $\hat{\Delta}$ (i.e. the dynamics of the principal field vectors evaluated at the sampled states) are governed by an ODE that is related to discrete-time principal components analysis algorithms of the form (8.2.2). The terms introduced in Definition 8.1 will be used without comment in the sequel.

Theorem 8.1

The local dynamics of the principal field matrix (8.2.8) is given at each sampled state $\mathbf{x}_1, \dots, \mathbf{x}_q$ by the equations of motion

$$\frac{d}{dt} \mathbf{W}(\mathbf{x}_t, \theta) = \mathbf{W}(\mathbf{x}_t, \theta) \Sigma(\mathbf{x}_t) - \mathbf{K}(\mathbf{x}_t) \mathbf{W}(\mathbf{x}_t, \theta) \quad (8.2.9)$$

for all $t \geq 0$.

Proof: Observe that the BR-PMA weight update law given in equation (8.2.5) can be written as follows

$$\frac{d}{dt} \theta_t [\mathbf{G}(\mathbf{x}_1) \cdots \mathbf{G}(\mathbf{x}_q)] = [\gamma_t(\mathbf{x}_1) \cdots \gamma_t(\mathbf{x}_q)].$$

where

$$\mathbf{G}(\mathbf{z}) = \begin{bmatrix} \left. \frac{\partial G(\mathbf{x}, \mathbf{y}_1)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{z}} \\ \vdots \\ \left. \frac{\partial G(\mathbf{x}, \mathbf{y}_q)}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{z}} \end{bmatrix}.$$

If we expand equation (8.2.6) for the l^{th} sample point, we have

$$\begin{aligned}
 \frac{d}{dt} \theta_i \mathbf{G}(\mathbf{x}_l) &= \left[\sum_{r=1}^n \sum_{s=1}^q \Sigma_{i,r}(\mathbf{x}_l) \theta_{r,s} \frac{\partial G(\mathbf{x}, y_s)}{\partial x_b} \Big|_{\mathbf{x}=\mathbf{x}_l} \right]_{b=1 \dots n} \\
 &\quad - \left[\sum_{r=1}^n \sum_{s=1}^q \mathbf{K}_{b,r}(\mathbf{x}_l) \theta_{i,s} \frac{\partial G(\mathbf{x}, y_s)}{\partial x_r} \Big|_{\mathbf{x}=\mathbf{x}_l} \right]_{b=1 \dots n} \\
 &= [\Sigma_{i,1}(\mathbf{x}_l) \cdots \Sigma_{i,n}(\mathbf{x}_l)] \left[\theta_{a,s} \frac{\partial G(\mathbf{x}, y_s)}{\partial x_b} \Big|_{\mathbf{x}=\mathbf{x}_l} \right]_{a=1 \dots n, b=1 \dots n} \\
 &\quad - \left[\theta_{a,s} \frac{\partial G(\mathbf{x}, y_s)}{\partial x_b} \Big|_{\mathbf{x}=\mathbf{x}_l} \right]_{a=1 \dots n, b=1 \dots n} [\mathbf{K}_{b,a}(\mathbf{x}_l)]_{a=1 \dots n, b=1 \dots n} \\
 &= \Sigma_i(\mathbf{x}_l) [\theta_a \mathbf{G}(\mathbf{x}_l)]_{a=1 \dots n} - \theta_i \mathbf{G}(\mathbf{x}_l) \mathbf{K}(\mathbf{x}_l)^T
 \end{aligned} \tag{8.2.10}$$

Expressing this result in matrix form, we obtain

$$\left[\frac{d}{dt} \theta_i \mathbf{G}(\mathbf{x}_l) \right]_{i=1 \dots n} = \Sigma(\mathbf{x}_l) [\theta_i \mathbf{G}(\mathbf{x}_l)]_{i=1 \dots n} - [\theta_i \mathbf{G}(\mathbf{x}_l)]_{i=1 \dots n} \mathbf{K}(\mathbf{x}_l)^T$$

Observe that we may write

$$\begin{aligned}
 \frac{d}{dt} \frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{x}_l} &= \sum_{k=1}^q \left[\frac{\partial G(\mathbf{x}, y_k)}{\partial x_j} \Big|_{\mathbf{x}=\mathbf{x}_l} \frac{d}{dt} \theta_{i,k} \right]_{i=1 \dots q, j=1 \dots n} \\
 &= \sum_{k=1}^q \begin{bmatrix} \frac{d}{dt} \theta_{1,k} & & 0 \\ & \ddots & \\ 0 & & \frac{d}{dt} \theta_{n,k} \end{bmatrix} \mathbf{G}(\mathbf{x}_l) \\
 &= [\theta_j \mathbf{G}(\mathbf{x}_l)]_{j=1 \dots n}
 \end{aligned} \tag{8.2.11}$$

Therefore, substituting equation (8.2.11) into the time derivative of equation (8.2.10) we have the following expression for the local dynamics of the Jacobian of $\Phi(\mathbf{x}, \theta)$

$$\frac{d}{dt} \frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{x}_l} = \Sigma(\mathbf{x}_l) \frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{x}_l} - \frac{\partial \Phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{x}_l} \mathbf{K}(\mathbf{x}_l)^T. \tag{8.2.12}$$

Substituting (8.2.12) into equation (8.2.8) we obtain equation (8.2.9). Since this holds true for any $1 \leq d \leq n$, the proof is complete. ■

The proof of Theorem 8.1 also demonstrates that, at each $t \geq 0$, the BR-PMA dynamics implicitly solves the standard regularization equation (7.3.5), minimizing the functional (7.3.4) for the vectors evolving according to the local dynamics equations (8.2.2).

Theorem 8.1 is central to the analysis of the convergence behaviour of BR-PMA, because it permits us to make use of many of the stability results derived for connectionist PCA algorithms. Our first application will be in the next lemma, where we find that, for any set of observations, the principal fields are always linearly independent while the weight vector θ converges. This result will be important in proving stability of the BR-PMA algorithm.

Theorem 8.2

As the BR-PMA algorithm converges, the principal vector fields,

$$\frac{\partial \Phi(\mathbf{x}, \theta)}{\partial x_i}, \quad 1 \leq i \leq (n-1),$$

are always linearly independent at each sampled state $\mathbf{x}_1, \dots, \mathbf{x}_q$.

Proof: Since the initial θ_0 is an element of V , it is PMA consistent. This implies that the Jacobian of $\Phi(\mathbf{x}, \theta) \in F_{RM}$ is nonsingular. For any $1 \leq d \leq (n-1)$, consider the matrix (8.2.8). The rows of this matrix, evaluated at each sampled state, represent the evolving local principal components at each time t .

The proof proceeds by contradiction. Assume that the rows of $\mathbf{W}(\mathbf{x}, \theta)$ are linearly dependent at some $t_1 > 0$. Since the $(n-1)$ -dimensional image of $\mathbf{W}(\mathbf{x}, \theta)^T$ is also the $(n-1)$ -dimensional row space of $\mathbf{W}(\mathbf{x}, \theta)$, observe that the $(n-1) \times (n-1)$ matrix

$$\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^T \tag{8.2.13}$$

is nonsingular if and only if the rows of $\mathbf{W}(\mathbf{x}, \theta)$ are linearly independent. Necessity follows from the fact that the rank of a matrix product cannot be larger than the smallest rank of the two matrices. Sufficiency is due to the fact that the image of

$\mathbf{W}(\mathbf{x}, \theta)^\top$ is equal to the row space of $\mathbf{W}(\mathbf{x}, \theta)$. Taking the approach of Plumbley [Plu95][Theorem 2.1] we can express the change in norm of $\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top$ and its inverse with time as

$$\begin{aligned} \frac{d}{dt} \left\| \mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top - \mathbf{I}^{d \times d} \right\| \\ = -4 \text{Tr} \left[\left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right) \mathbf{W}(\mathbf{x}, \theta) \Sigma(\mathbf{x}) \mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right) \right] \\ \leq 0. \end{aligned}$$

and

$$\begin{aligned} \frac{d}{dt} \left\| \left[\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right]^{-1} - \mathbf{I}^{d \times d} \right\| \\ = t - 4 \text{Tr} \left[\mathbf{W}(\mathbf{x}, \theta) \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \right. \\ \left. \left[\left[\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right]^{-1} - \mathbf{I}^{d \times d} \right] \right. \\ \left. = \mathbf{W}(\mathbf{x}, \theta) \Sigma(\mathbf{x}) \mathbf{W}(\mathbf{x}, \theta)^\top \right. \\ \left. \left[\left[\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right]^{-1} - \mathbf{I}^{d \times d} \right] \right. \\ \left. \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta) \right] \\ \leq 0 \end{aligned}$$

Clearly, if $\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top$ is initially bounded and nonsingular then it remains so for all time. But, we know from Definition 8.1 that $\mathbf{W}(\mathbf{x}, \theta)$ is maximum rank at $t = 0$ and so is matrix $\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top$, contradicting the assumption of linear dependence at some time $t_1 > 0$. This completes the proof. ■

These norms also tell us that the matrix $\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top$ converges to the identity matrix $\mathbf{I}^{(n-1) \times (n-1)}$ if θ converges to a stable representation. This fact will find further use below.

In the next theorem we show that, if the BR-PMA algorithm converges it will always converge to a good PMA approximation. In practice, this means that all

of the stable equilibria of the algorithm correspond to weight vectors that represent good approximations to the environmental foliation.

Theorem 8.3

The stable equilibria of the BR-PMA algorithm are all good PMA approximations.

Proof: Theorem 8.2 tells us that the local dynamics of the principal fields are given by the system of equations (8.2.9). An alternative approach for expressing local convergence is to characterize the convergence of the subspace that the principal fields span at each sampled state (i.e. the image of $\mathbf{W}(\mathbf{x}, \theta)$). We may do so by defining an ODE for the evolution of the unique orthogonal projection onto this subspace at each $t \geq 0$. We can form this projection matrix at each t as follows

$$\mathbf{P}(\mathbf{x}, \theta) = \mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta). \quad (8.2.14)$$

because the proof of Theorem 8.2 assures us that inverse matrix $\left[\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right]^{-1}$ exists at every t . Suppose that (8.2.14) is not the orthogonal projection onto the image of $\mathbf{W}(\mathbf{x}, \theta)$, then one of the following statements must be false

$$\mathbf{P}(\mathbf{x}, \theta)^2 = \mathbf{P}(\mathbf{x}, \theta). \quad (8.2.15)$$

$$\mathbf{P}(\mathbf{x}, \theta)^\top = \mathbf{P}(\mathbf{x}, \theta). \quad (8.2.16)$$

$$\mathbf{P}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top = \mathbf{W}(\mathbf{x}, \theta). \quad (8.2.17)$$

However, (8.2.15) is true because

$$\begin{aligned} & \left(\mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta) \right)^2 \\ &= \mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \mathbf{W}(\mathbf{x}, \theta) \\ &= \mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta). \end{aligned}$$

Equation (8.2.16) follows from

$$\begin{aligned}
 & \left(\mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta) \right)^\top \\
 &= \mathbf{W}(\mathbf{x}, \theta)^\top \left[\left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \right]^\top \mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta) \\
 &= \mathbf{W}(\mathbf{x}, \theta)^\top \left[\left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^\top \right]^{-1} \mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta) \\
 &= \mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \mathbf{W}(\mathbf{x}, \theta).
 \end{aligned}$$

Finally, equation (8.2.17) is true since

$$\begin{aligned}
 & \mathbf{P}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \\
 &= \mathbf{W}(\mathbf{x}, \theta)^\top \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right)^{-1} \left(\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^\top \right) \\
 &= \mathbf{W}(\mathbf{x}, \theta)^\top.
 \end{aligned}$$

We must now derive the local dynamics of matrix $\mathbf{P}(\mathbf{x})$. Taking the derivative of equation (8.2.15) gives

$$\frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) = \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta)^2 = \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) + \mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta). \quad (8.2.18)$$

From equation (8.2.18) we have

$$\mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) = \mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta)^2 + \quad (8.2.19)$$

$$\mathbf{P}(\mathbf{x}, \theta)^2 \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) = \quad (8.2.20)$$

$$2\mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta).$$

which implies that

$$\mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt} \mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) = \mathbf{0}. \quad (8.2.21)$$

Therefore, equation (8.2.21) permits us to re-write equation (8.2.18) as

$$\begin{aligned}
 \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) &= \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) - \mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) \\
 &\quad + \mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) - \mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) \\
 &= (\mathbf{I} - \mathbf{P}(\mathbf{x}, \theta)) \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) \mathbf{P}(\mathbf{x}, \theta) + \mathbf{P}(\mathbf{x}, \theta) \frac{d}{dt}\mathbf{P}(\mathbf{x}, \theta) (\mathbf{I} - \mathbf{P}(\mathbf{x}, \theta)).
 \end{aligned} \tag{8.2.22}$$

If desired, the right hand side of (8.2.22) can then be written in terms of $\mathbf{W}(\mathbf{x}, \theta)$.

Williams [Wil85], Oja [OK85, Oja92] and Plumbley [Plu95] have all studied the equilibria of ODE (8.2.22) in the context of connectionist PCA algorithms. In particular, Oja proves that the stable equilibria of (8.2.22) correspond exactly to projections for the stable values of $\mathbf{W}(\mathbf{x}, \theta)$. We may, therefore, conclude that the local dynamics of equation (8.2.22) will all converge to local tangent space estimates that are equivalent to the PCA solution for the corresponding local value of the covariance matrix $\Sigma(\mathbf{x})$. The corresponding equilibria value for θ clearly satisfies the conditions for a good PMA approximation since at equilibrium, for $1 \leq i \leq q$ we have

$$\dot{\Delta}(\mathbf{x}_i) = \text{span} \left\{ \frac{\partial \Phi(\mathbf{x}, \theta)}{\partial x_1}, \dots, \frac{\partial \Phi(\mathbf{x}, \theta)}{\partial x_d} \right\} = T(\mathbf{x}_i), \text{ and}$$

because, by Theorem 7.2, θ is a member of V , a set of PMA representations at every $t \geq 0$, and finally, as we have seen in the proof of Theorem 8.2, at each $t > 0$ θ satisfies equation (??), which minimizes functional (7.3.4). Since the condition for a good PMA approximation is fulfilled by the equilibrium representation θ , the proof is complete. ■

The previous theorem demonstrated that the output of the BR-PMA algorithm is guaranteed to be a good solution to the manifold approximation problem, *when the algorithm converges*. However, we do not have any guarantees that the algorithm will actually converge. In the next theorem we see that our concerns are unfounded in practice because convergence is a generic property of the algorithm.

Theorem 8.4

The BR-PMA algorithm will almost surely converge to a good PMA approximation.

Proof: Plumbley has shown (see [Plu95][Theorem 5.2]) that the function

$$L(W) = \frac{1}{2} [\log \det(\mathbf{W}\mathbf{P}^*\mathbf{W}) - \log \det(\mathbf{W}\mathbf{W})] \quad (8.2.23)$$

is a Lyapunov function which guarantees that the equilibrium $\mathbf{P}^*$ is a stable equilibria of the projection matrix dynamics (equation (8.2.22)) provided that $\mathbf{W}\mathbf{W}^T$ is bounded and nonsingular for all $t \geq 0$. Furthermore, he has used function (8.2.23) to characterize the basin of attraction for $\mathbf{P}^*$ as the set containing all $\mathbf{W}$ such that $\mathbf{W}\mathbf{P}^*\mathbf{W}$ is nonsingular.

Once again, we will use Theorem 8.1 to extend these results to the PMA setting. We know from the proof of Theorem 8.2 that the value of matrix $\mathbf{W}(\mathbf{x}, \theta) \mathbf{W}(\mathbf{x}, \theta)^T$ at each stable point is indeed finite and nonsingular for any stable BR-PMA trajectory. Therefore, the local dynamics of Theorem 8.1 will converge to the local PCA estimates of the foliation tangent spaces for any θ that ensures that

$$\mathbf{W}(\mathbf{x}, \theta) \mathbf{P}^*(\mathbf{x}) \mathbf{W}(\mathbf{x}, \theta)^T \quad (8.2.24)$$

is nonsingular.

When is matrix (8.2.24) nonsingular? Whenever

$$\text{Im}(\mathbf{W}^T(\mathbf{x}, \theta_0)) \cap S^\perp = 0, \quad (8.2.25)$$

where S is the d -dimensional image of $\mathbf{P}^*(\mathbf{x})$. Since $\mathbf{P}^*(\mathbf{x})$ is a rank d , $n \times n$ projection matrix, it can be decomposed into the product

$$\mathbf{P}^*(\mathbf{x}) = \mathbf{Q}\mathbf{Q}^T$$

where $\mathbf{Q}$ is a rank d , $n \times d$ matrix with column space equal to S . We have

$$\mathbf{W}^T(\mathbf{x}, \theta_0) \mathbf{P}^*(\mathbf{x}) \mathbf{W}^T(\mathbf{x}, \theta_0)^T = \mathbf{W}^T(\mathbf{x}, \theta_0) \mathbf{Q} (\mathbf{W}^T(\mathbf{x}, \theta_0))^T. \quad (8.2.26)$$

the product of two $d \times d$ matrices. In order for matrix (8.2.26) to be nonsingular $\mathbf{W}^\top(\mathbf{x}, \theta_0) \mathbf{Q}$ must be nonsingular. This can only happen if the orthogonal projection of $S = \text{Im}(\mathbf{Q})$ onto the row space of $\mathbf{W}^\top(\mathbf{x}, \theta_0)$ is a d -dimensional subspace, i.e. condition (8.2.25) is true. Since the dimensionality of the two subspaces in the condition are $d < n$, any random choice of entries for $\mathbf{W}^\top(\mathbf{x}, \theta_0)$ will almost certainly satisfy the condition for any choice of S . Consequently, any random choice of θ will almost certainly satisfy the condition as well at every local sampled state. This proves the theorem as desired. ■

So, with Theorem 8.4, we see that the BR-PMA algorithm is a reliable connectionist implementation of PMA. If we let $\bar{\Delta}(\mathbf{x})$ be the output of the symbolic PMA algorithm described in Section 7.3, and $\hat{\Delta}(t, \mathbf{x})$ be the representation produced by BR-PMA at each time $t \geq 0$, then, starting from almost any $\theta \in V$, BR-PMA will ensure that

$$\lim_{t \rightarrow \infty} \hat{\Delta}(t, \mathbf{x}) = \bar{\Delta}(\mathbf{x}).$$

8.3 Holonomic Networks

In this section, we show how to construct a continuous-time network that exhibits holonomic dynamics from a PMA network with BR-PMA as the weight update algorithm.

This network is capable of both learning a manifold constraint, and applying it to its own network dynamics. After learning a nonlinear network foliation, the network dynamics will be confined to a leaf of the foliation regardless of the input signals applied. The nonlinear manifold (leaf) will be the one that includes the initial activation value.

Figure 8.4 illustrates the network architecture.

The PMA network (enclosed in the dashed box) produces the d pools of n output neurons, each of which represents a basis vector for $\Delta(\mathbf{x}) = \text{span} \left\{ \frac{\partial \mathcal{A}(\mathbf{x})}{\partial \mathbf{x}_1}, \dots, \frac{\partial \mathcal{A}(\mathbf{x})}{\partial \mathbf{x}_d} \right\}$.

Figure 8.4: Holonomic Network Architecture

Each of these output neurons is now replaced with a Sigma Pi unit which multiplies the input value by a nonspecific gain shared by the whole population. The gain for pool j is input signal u_j to the Holonomic network. In this way, the input vector u acts as the coefficients.

The outputs of the holonomic network are a bank of n continuous-time neurons that take a set of decay rates y_i (where $1 \leq i \leq d$) as their inputs and have the equation of

$$\dot{x}_i = \sum y_i \quad \text{for } 1 \leq i \leq n.$$

In order to complete the recurrent network, the integrated output $x(t)$ is recurrently connected to the PMA network inputs.

The resulting Holonomic Network implements the Isidori system

$$\dot{x} = \sum_{i=1}^d \frac{\partial \phi(\omega)}{\partial \omega_i} \bigg|_{\omega=x} u_i$$

Holonomic networks do not have transient dynamics. This is equivalent to saying that every point of state space is a stable point for a holonomic network. As such, holonomic networks are a useful component of a larger network structure or hybrid system.

As an example of how a holonomic network might be used in such a system, we refer the reader to Figure 3.5.

The holonomic network acts as an output stage that answers the output dynamics are confined to the consistent manifold $\mathcal{M}$.

Since the holonomic network does not introduce any transient dynamics of its own, a second “steering” controller can provide steering signals that give whatever dynamics are required.

The steering controller can be implemented as a second network that learns to map points $\mathbf{x} \in \mathcal{M}$ on the manifold to steering signals that produce the desired linear combination of the local basis at $\mathbf{x}$. This basis is simply the neuron outputs in the Holonomic network that correspond to the vector fields

$$\left. \frac{\partial \phi(\mathbf{y})}{\partial \mathbf{y}_i} \right|_{\mathbf{y}=\mathbf{x}}$$

To train such a network, we can use the following method. We will assume the Steering Control network is a supervised feedforward network that can learn a mapping from pairs (s_i, x_i) . The goal of the Steering Control network is to map each point x in the output space $\mathbb{R}^n$ to an element of $T_{\mathbf{x}}\mathcal{M}$ that satisfies some goal relative to the foliation. The goal can therefore be expressed as a bijective mapping from output space to $T_{\mathbf{x}}\mathcal{M}$, $\gamma : \mathbb{R}^n \mapsto \mathbb{R}^d$. For example, if our goal is to have the system dynamics stable at points on each learned manifold that are locally closest to some point $\mathbf{y} \in \mathbb{R}^n$, then we would design γ as $\gamma(\mathbf{x}) = \mathbf{P}_{\Delta}(\mathbf{x})(\mathbf{y} - \mathbf{x})$ where

$$\mathbf{P}_{\Delta}(\mathbf{x}) = \Delta (\Delta^T \Delta)^{-1} \Delta^T \quad \text{and} \quad \Delta(\mathbf{x}) = \left[\left. \frac{\partial \phi(\mathbf{w})}{\partial \mathbf{w}_1} \right|_{\mathbf{w}=\mathbf{x}} \cdots \left. \frac{\partial \phi(\mathbf{w})}{\partial \mathbf{w}_d} \right|_{\mathbf{w}=\mathbf{x}} \right].$$

Thus $\gamma(\mathbf{x})$ is the orthogonal projection of the directional vector (pointing toward the stable point $\mathbf{y}$) onto $T_{\mathbf{x}}\mathcal{M}$. $\gamma(\mathbf{x})$ is therefore a measure of the local “energy” of the manifold relative to $\mathbf{y}$. If the manifold requires the dynamics at x^0 to move away from $\mathbf{y}$, $|\gamma(\mathbf{x})|$ will be 0. $\gamma(\mathbf{x})$ is a vector that points in the direction of local x^0 otherwise. If $\gamma(\mathbf{x})$ is explicitly known (e.g. by design) then it can be encapsulated as automatic and we may consider the learning algorithm to be unsupervised. If samples of γx are

provided as training data. then the algorithm is supervised. Given the mapping γx we can compute the desired output of the Steering Controller network. This will be any input signals that linearly combine the $\frac{\partial d(\mathbf{x})}{\partial \mathbf{x}_i}$ to produce $\gamma(x)$. Since by Lemma 7.7 we know that

$$\left[\frac{\partial \phi(\mathbf{w})}{\partial \mathbf{w}_1} \bigg|_{\mathbf{w}=\mathbf{x}} \cdots \frac{\partial \phi(\mathbf{w})}{\partial \mathbf{w}_d} \bigg|_{\mathbf{w}=\mathbf{x}} \right]$$

is generically maximum column rank. the desired signal $s = \Delta^{\tau}(\mathbf{x})\gamma(\mathbf{x})$. To train the Steering Controller network we need to only generate a representative set of training pairs (s_i, x_i) either by randomly sampling the value of $\gamma(\mathbf{x})$ on $\mathcal{U}$. or using the values of $\gamma(\mathbf{x})$ given to us as training data. It is well known [LN96] that the desired mapping can be learned to arbitrary accuracy by a Steering Controller network with sufficient weights and training samples.

The semantics of the steering goal are unimportant to the Holonomic network.

Alternatively. we can employ a nonlinear feedback controller as described in Chapter 3 to implement the steering controller.

8.4 A Potential Application of Holonomic Networks: Connectionist Types

In this section we present a method to represent and control operations on symbolic data structures at run-time in connectionist systems. The kinds of structures we have in mind are of the classic AI variety. such as frames. S-expressions. stacks. and semantic nets. We show how holonomic networks can be used to enforce connectionist types on variable bindings in these structures.

8.4.1 Representing Symbolic Structures

Various proposals have been made for representing structured data in connectionist networks. We will employ the encoding method described by Smolensky

[Smo91]. This approach is general enough to represent any symbolic structure given a large enough network, and it can represent useful classes of structures when reasonable resource limitations are imposed. Further investigation may suggest alternative encodings that are easier to control, but Smolensky's approach makes a good departure point.

8.4.1.1 Representation by Role Decomposition

We will view different classes of structures as collections of variables called roles. A subset of the possible roles is associated with each class as its legal role set. A particular instance of a structure may bind a subset of its legal roles to values called fillers. The binding between the role r and the filler f is given by f/r . This approach allows us to reduce the problem of representing arbitrarily complex structures to the simpler problem of representing conjunctions of role/filler bindings.

As examples, consider the representations of strings and S-expressions. To represent the class of strings of length n , we would employ the position roles $R = \{a, b, \dots, z\}$, and the fillers $F = \{a, b, \dots, z\}$. The length 3 string instance "abc" is represented by the bindings $\{a/r_1, b/r_2, c/r_3\}$. To represent the class of S-expressions we let the set of roles R contain all roles of the form r_b , where b is any binary string. A "0" in this string represents the LISP car operation, and a "1" represents the cdr operation. The set F of allowed fillers consists of all legal atoms (but not S-expressions). So, for example, the binding a/r_{01011} stands for "the cadaddr is the atom a", and the S-expression instance (a (b c)) is represented by the bindings $a/r_0, b/r_{01}, c/r_{11}$. We see from this example that the set of roles may be arbitrarily large. Furthermore, a structure may contain an infinite number of bindings, as in the case of an S-expression with circular references.

In [Smo91], it is shown that the traditional logic-based approach to representing structures as sets of elements engaged in mutual relations (e.g. representing "abc" as `left_of(a,b)` and `left_of(b,c)`) is a special case of the role/filler approach. So

we do not lose any generality by employing this method of representation.

Before we continue, we need a formal definition of these ideas. First we need a way to express the idea of legal bindings. A role decomposition F/R for the set of symbolic structures consists of a pair of sets (F, R) and a mapping

$$\mu_{F/R} : F \times R \rightarrow \text{Pred}(S), \quad \text{where} \quad (f, r) \rightarrow f/r.$$

For any pair $f \in F$ and $r \in R$, the predicate $\mu_{F/R}(f, r) = f/r$ is true when f fills the role r . The role decomposition is *recursive* iff $F = S$. We say that it has *single-valued roles* iff for any pair (s, r) where $s \in S$ and $r \in R$, there is at most one $f \in F$ such that $f/r(s)$ is true. So all roles in all structures are bound to a single filler. In other words, a structure with single valued roles only binds one value to each variable. There are no restrictions on the cardinalities of F and R .

Now that we have defined the legal bindings, we need to model the idea that a particular structure is composed of a set of such bindings. To this end we see that a role decomposition of S induces a mapping β called the *role/filler representation* of S , which takes each structure s to its set of role/filler bindings. β is defined as

$$\beta : S \rightarrow 2^{F \times R}, \quad \text{where} \quad s \rightarrow \{(f, r) | s \in S, \text{ and } f/r(s)\}$$

β induces a partial order on the structures in S via the set inclusion relation $\subseteq$, where $s_1 \leq s_2$ iff $\beta(s_1) \subseteq \beta(s_2)$.

We say that a role decomposition is *faithful* iff β is one-to-one. We are interested in faithful role decompositions because they allow structures to be associated with a predicate that has the simple conjunctive form alluded to above. This predicate is defined for each $s \in S$ as

$$\pi_{s_0}(s) = \bigwedge_{(f, r) \in \beta(s_0)} f/r(s).$$

$\pi_{s_0}(s)$ is true when the bindings in s_0 are a subset of those in s . Clearly, if the role decomposition is faithful then the partial order induced by β has a unique least element for which π_{s_0} holds. If this were not true, then another structure would have

the same bindings as s_0 , which contradicts the assumption of faithfulness. Therefore, each structure s is uniquely associated with (recovered from) its predicate π_s , and we have justification for representing the structure as the conjunction of role bindings in π_s .

8.4.1.2 A Tensor Product Realization of Role Decomposition

Our next task is to encode conjunctions of bindings in a form that can be realized in a connectionist setting. As we saw in Chapter 2, connectionist representations are patterns encoded in the activation state of an interconnected network of neurons. Therefore, the space of representational states for a connectionist system can be identified with the dynamical activation state space of the system. This gives it the structure of an $n - dimensional$ vector space V with natural basis $\{e_i\}$, $1 \leq i \leq n$, where each e_i corresponds to a network with all the neurons inactive except the i^{th} neuron which has an activation value of 1.

A *connectionist representation* is a mapping ψ from a set of symbolic structures S to the representational space V . We will describe ψ with respect to the distinguished basis $\{e_i\}$ unless otherwise indicated.

Our first task is to represent role/filler bindings in V . We define R and F to be the p -dimensional vector space V_R and the q -dimensional vector space V_F respectively. A binding between $r \in V_R$ and $f \in V_F$ is represented by the so-called *tensor product* $f/r = f \otimes r \in V = R \otimes V_F$. The components of the resulting pq -dimensional vector f/r are products of the components of f and r , so we have $(f/r)_{ij} = f_i r_j$. Now we can relate these tensor product bindings to the role decomposition F/R of S by defining the connectionist representations ψ_R , ψ_F , and ψ_B of the roles, fillers, and role/filler bindings respectively to be

$$\psi_R : R \rightarrow V_R.$$

$$\psi_F : F \rightarrow V_F.$$

$$\psi_B : \{f/r | f \in F, r \in R\} \rightarrow V_R \otimes V_F, \text{ where } f/r \rightarrow \psi_R(r) \otimes \psi_F(f).$$

Next, we must combine the bindings for a given structure to produce a vector representation $\psi(s) \in V$. This is where we make use of the conjunctive form of structures that have a faithful role decomposition, as described in Section 8.4.1.1. We employ vector addition to represent conjunction⁷ and assert that a connectionist representation ψ employs a *superpositional representation of conjunction* iff

$$\psi(\wedge_i p_i) = \sum_i \psi(p_i) \quad (8.4.1)$$

Note that in (8.4.1) we use a superpositional representation of conjunction so that the representation of the conjunction of bindings is simply the sum of the representations of the bindings. Therefore the tensor product representation of S is given by

$$\psi : S \rightarrow V_R \otimes V_F, \text{ where } \sum_{(f,r) \in \mathcal{J}(s)} \psi_R(r) \otimes \psi_F(f)$$

Letting $r = \psi_R(r)$ and $f = \psi_F(f)$ we see that the structure as conjunction of bindings is represented as the sum of tensor products

$$\psi(\wedge_i f_i/r_i) = \sum_i r_i \otimes f_i.$$

Finally, we can consider how the filler value of a role can be recovered from the structure representation $\psi(s)$. If the vectors that represent the roles are linearly independent it is possible to exactly recover the filler f_i bound to any role r_i . The roles $\{r_i\}$ form a basis in V_R that induces a dual basis $\{\omega_i\}$ in the dual space $V_\star^R$. We need only apply a partial inner product of the dual vector associated with a role to the structure in order to extract the filler, since

$$s \bullet \omega_i = \left(\sum_j f_j \otimes r_j \right) \bullet \omega_i = \sum_j f_j \langle r_j \cdot \omega_i \rangle = \sum_j f_j \delta_{ij} = f_i.$$

⁷There is a long tradition of representing conjunction as addition in connectionist models [Rum86b].

The role vector r_i can be used in place of its dual ω_i if the role vectors are orthogonal.

8.4.2 Connectionist Types

In Smolensky's theory, the role decomposition and role/filler representation mappings $\mu_{F/R}(f, r)$, and β , embody the concept of legal bindings for a structure class S . However, there is no indication how these type invariants can be maintained automatically by a network that realizes structures from this class. In this section we will consider how to represent a type constraint on the set of fillers that can be bound to a role. This is analogous to the idea of enforcing type compatibility in a conventional computer program. In the present case, our intent is to constrain the network so that it is incapable of allowing an incorrect filler assignment, despite being subject to the influence of inputs from other networks in the system.

A variable in a symbolic data structure is usually tagged with a defined type that indicates the format and range of values that it may be bound to. In a symbolic AI system type rules can be expressed as part of the domain theory, or they can be implicitly encoded in the system compiler and/or run-time engine. Type knowledge is typically used to constrain the assignment of values, and thereby partially determine the legal operations that can be applied to a structure. A connectionist system should be able to represent constraints that are analogous to types.

8.4.2.1 Format Tags

In a connectionist system, the "format" of a filler f may be represented by a tag that is encoded in the filler itself. For example, we may recursively interpret the filler f as a structure with role decomposition $\mu_{T/D}$ and tensor product realization

$$f = r_T \otimes t + r_D \otimes d.$$

Role r_T is interpreted as "filler f 's format tag", and role r_D is "the data value of filler f ". The tags t are elements from the type space V_T and the data values

d are from the underlying data space V_D . Clearly, the same technique can be used to encode the format tag into the role r . In this case r is interpreted as a tensor product structure that contains two sub-roles: r_T , described above, and r_I , which represents “the identifier for role r ”.

Although both approaches have their advantages, we will find it useful to encode the format tag in the filler when defining a special class of connectionist types. Furthermore, this approach enables the network that “reads” the filler to base its method of interpretation on a classification of the format tag. This capability is desirable if the filler is able to propagate to a distant network which must perform a calculation based on the format.

8.4.2.2 Formatted Filler Types

Representing the range of data associated with a connectionist type is an interesting problem because of the nonlinearity of the ranges that occur in application areas traditionally associated with connectionist computation. Connectionist systems are particularly well-suited to problems that require the application of “low-level” knowledge, such as speech recognition and motor control. In these applications, the network learns to perform important mappings between nonlinear manifolds of the input and output spaces. For example, a manifold corresponding to workspace locations for a robot arm might map to the manifold of joint configurations that intersect those locations. Alternatively, a manifold of format descriptors might map to classification values that indicate the sex of the speaker. In these sorts of applications the values of interest are confined to a general differential manifold rather than a simple real-valued range. Ideally, the symbolic structures that we represent in a connectionist system should integrate naturally with these types of ranges.

We will say that a connectionist representation $\psi(s)$ has the filler manifold property for role r_i and filler set F iff the role decomposition is faithful

$$\psi_{si}(f_i) = \psi(\wedge_{j=1}^k f_j / r_j),$$

for all r_j , $1 \leq j \leq k$, and f_j , $j \neq i$, held constant and $f_i \in \mathbf{V}_F$.

In other words, the representation of structure $s \in S$ is confined to a differentiable manifold as the value bound to the i^{th} role is varied.

For any role/filler binding representation ψ_B that has the filler manifold property, we may define the family $\mathfrak{F}(\psi_B: T, D)$ of formatted filler types for the format tag set T and the data value set D . This family consists of the sets τ_t such that

$$\tau_t = \{f | \forall d \in D. f = \psi_{fT}(t)\}.$$

and τ_t can be given the structure of a differentiable manifold in $\mathbf{V}_F$. Each such manifold represents the set of filler values that can be associated with a particular format.

8.4.2.3 Realizing Type Constraints in Connectionist Structures

We must now consider how a network that realizes a particular class of structure S might enforce type constraints, thereby ensuring that only valid assignments are made to the structure under influence of inputs from other parts of the system. For the purposes of illustration, we will consider the case of formatted filler types. We will also assume that the role decomposition of S is faithful, and that we employ a superpositional representation of conjunction.

The problem of enforcing type constraints in a connectionist system that realizes this kind of structure representation, is equivalent to the problem of forcing the dynamics of a control system to evolve on a characteristic submanifold of its state space. To see this equivalence, we define the *legal structure set* of the representation ψ as

$$\mathcal{L}_\psi = \{s | s \sum_{i=1}^k f_i \otimes \tau_{t_i}, f_i \in \tau_{t_i}, 1 \leq i \leq k, \tau_{t_i} \in \mathfrak{F}(\psi_V: T, D)\}.$$

where $t_i \in \mathbf{V}_T$ is the format tag of the type associated with role τ_{t_i} . This set consists of all representations of achievable structures that have legal bindings. If $\mathcal{L}_\psi$ can be

given the structure of a differentiable manifold. then it is clear that enforcing type assignment is a matter of restricting the controllable dynamics of the connectionist system to this manifold.

We shall now show that $\mathcal{L}_v$ is a differentiable manifold. Our starting point is to show that the tensor product representation has the filler manifold property with respect to each role in the structure. This allows us to consider a continuous change in a single role/filler binding as evolution of the state of the connectionist system along a trajectory that is confined to a differentiable manifold. The filler manifold property is a consequence of the form of the mapping

$$\psi_{s_i} = f_i \otimes r_i + \sum_{j \neq i} f_j \otimes r_j = \begin{pmatrix} f_{i1}r_i \\ f_{i2}r_i \\ \dots \\ f_{iq}r_i \end{pmatrix} + \theta.$$

where $r \in \mathbf{V}_R$ and $f \in \mathbf{V}_F$ are p- and q-dimensional respectively. and $\theta \in V$.

If we view this representation as a mapping of the q-dimensional manifold $\mathbf{V}_F$ into the pq-dimensional manifold V. we can examine the Jacobian to determine the properties of the image. For all roles of interest⁸, the $pq \times q$ Jacobian matrix

$$\partial\psi_{s_i} = \frac{\partial f_i \otimes r_i + \theta}{\partial f_i} = \begin{pmatrix} r_i & 0 & 0 \\ 0 & r_i & 0 \\ \dots & \dots & \dots \\ 0 & 0 & r_i \end{pmatrix}.$$

has a q-dimensional column space that is spanned by the q linearly independent column vectors. This distribution is spanned by constant vector fields. so the Rank theorem tells us that the associated manifold must be a q-dimensional hyperplane of constant dimension. Therefore, the representation is an immersion of the q-dimensional manifold $\mathbf{V}_F$ into a q-dimensional linear manifold (a q-dimensional subspace) of V.

⁸That is, all roles but the *nil* role $r=0$. This role is not of interest because its tensor product with any filter is always 0.

This manifold will be offset from the origin by the vector θ . The same line of reasoning applied to the tensor product representation of role/filler bindings (i.e. ‘structures’ consisting of one binding) allows us to conclude that the family of formatted filler types also consists of linear manifolds of dimension $\leq q$. The immersion will map these to similarly dimensioned manifolds of V . Finally, since i is arbitrary, we may conclude that ψ has the filler manifold property with respect to each role in the structure as originally claimed.

In light of the filler manifold property, it is easy to see that $\mathcal{L}_v$ is a differentiable manifold. If this were not the case, there would be a legal structure that could not be reached by continuous changes in the role bindings. As we have seen, the linear manifolds that represent the restriction of the immersion $\partial\psi_{s_i}$ span the manifold of legal structures in V .

Given that the set of structures that satisfy a type invariant form a manifold constraint, a holonomic network can be used to learn and enforce the invariant from observations of continuous changes that take the structure from legal binding to legal binding.

8.5 Application of PMA to Nonholonomic Networks

In this section we show that PMA can be applied to the problem of enforcing manifold constraints on the dynamics of nonholonomic networks. That is, for GNF networks that are generically fully-controllable, we show how the distribution learned by a PMA network using the BR-PMA learning algorithm, can be applied to construct the GNF network dynamics to the learned manifold. We show that our results hold locally around each sample point, and we give a further characterization of the extent of these regions in $\mathcal{R}^n$.

In the following Lemmas, we characterize the regions of the GNF network state space on which a learned manifold constraint can be enforced. What is essential here is the invertibility of the regularized RBF mappings. It is important to note

that these regions are determined solely by the characteristics of the RBF mappings and not by the GNF network of its dynamics.

As we saw in Chapter 4 on GNF controllability, GNF networks are almost certainly nonholonomic. Therefore in the remainder of this section we will make the reasonable assumption that the network being controlled is fully controllable. Furthermore, as we have seen in Lemma 7.3, $G(x)$ is generically maximum column rank. We will assume this to be true in the sequel.

Theorem 8.5

The order- $\hat{q}$ regularized RBF mapping $\phi(\mathbf{x}, \theta) \in \mathbb{F}_{\text{RLI}}$ is generically invertible in an open region around each sample point.

Proof: Theorem 8.2 tell us that $\frac{\partial \phi(\mathbf{x}, \theta)}{\partial \mathbf{x}}$ is nonsingular at every sample point. Since the functions $\phi(\mathbf{x}, \theta)$ are C^∞ , the inverse function theorem tells us that there exists an open neighbourhood U_z of each point $z \in \mathbb{R}^n$ on which $\phi(\mathbf{x}, \theta)$ is a diffeomorphism, and therefore invertible. Note that this inverse $\phi^{-1}(\mathbf{x}, \theta)$ is also differentiable and that $\frac{\partial \phi^{-1}(\mathbf{x}, \theta)}{\partial \mathbf{x}}$ also exists on the appropriate region. ■

As a consequence of the above, we can impose a manifold constraint onto a nonholonomic GNF network in the neighbourhood of each sample point. As the density of the points increases, we are guaranteed of a greater degree of control. Furthermore, as the following theorem demonstrates, there is reason to expect that a large region of the space will be controllable.

Theorem 8.6

The order- $\hat{q}$ regularized RBF mapping $\phi(\mathbf{x}, \theta) \in \mathbb{F}_{\text{RLI}}$ of the form

$$\phi(\mathbf{x}, \theta) = \theta \mathbf{g}(\mathbf{x}), \quad \text{where} \quad \mathbf{g}(\mathbf{x}) = \begin{bmatrix} G(\mathbf{x}, \mathbf{y}_1) \\ \dots \\ G(\mathbf{x}, \mathbf{y}_{\hat{q}}) \end{bmatrix}.$$

is invertible in an open neighbourhood around every $\mathbf{x} \in \mathbb{R}^n$ such that

$$\text{kernel}(\theta) \cap \text{span} \left\{ \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}_1} |_{\mathbf{x} = \omega}, \dots, \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}_n} |_{\mathbf{x} = \omega} \right\} = \mathbf{0}.$$

Proof: We first intend to prove that the $n \times \hat{q}$ matrix θ is maximum row rank by Theorem 8.5. We shall show this by contradiction. Assume to start that θ does not have this property. Since $\phi(\mathbf{x}, \theta) = \theta \mathbf{G}(\mathbf{x})$ and $\mathbf{G}(\mathbf{x})$ is maximum column rank, $\phi(\mathbf{x}, \theta)$ must be singular everywhere. This contradicts theorem 8.5, so θ is maximum row rank.

In order for $\phi(\mathbf{x}, \theta)$ to be invertible at a point $\omega \in \mathcal{R}^n$, the inverse function then tells us that it is sufficient that $\frac{\partial \phi(\mathbf{x}, \theta)}{\partial \mathbf{x}}$ be nonsingular at ω . It is well known that the product $\theta \mathbf{G}(\mathbf{x})$ is nonsingular when θ and $\mathbf{G}(\mathbf{x})$ are rank n , and the projection of the $n - \text{dimensional}$ column space of $\mathbf{G}(\mathbf{x})$ onto the $n - \text{dimensional}$ row space of θ is also $n - \text{dimensional}$. For this to occur, it is necessary and sufficient that the orthogonal complement of the row space of θ (i.e. $\text{kernel} \theta$) does not intersect the column space of $\mathbf{G}(\mathbf{x})$ except at $\mathbf{0}$ so that no nonzero column vector of $\mathbf{G}(\mathbf{x})$ projects to $\mathbf{0}$. This proves the theorem. ■

When $\hat{q} \gg n$, which is typical when the number of sample points is large, it is reasonable to expect the likelihood of an intersection to be small.

In order to appreciate our intuitive argument, consider that another way of expressing the condition of Theorem 8.6 is to say that $\phi(\mathbf{x}_0, \theta)$ is invertible on a neighbourhood of x_0 if

$$\text{kernel} \begin{bmatrix} \theta \\ \mathbf{G}^\perp(\mathbf{x}) \end{bmatrix} = \mathbf{0}. \quad (8.5.1)$$

where $\mathbf{G}^\perp(\mathbf{x})$ is the matrix with rows that span the kernel of $\mathbf{G}(\mathbf{x})^T$. If the kernel of θ and column space of $\mathbf{G}(\mathbf{x})$ intersect then there exists a vector $\mathbf{v}$ in the intersection that is orthogonal to the orthogonal complements of both subspaces, i.e.

$$\theta \mathbf{v} = \mathbf{G}^\perp(\mathbf{x}) \mathbf{v} = \mathbf{0}.$$

The set of all such vectors are clearly in the kernel of the matrix in condition (8.5.1).

We are interested in determining whether property (8.5.1) is generic in $\mathcal{R}^n$. If it is, then $\phi(\mathbf{x}, \theta)$ will be non-invertible except at points that occur with probability

0. However, the special nature of the row vectors of $\mathbf{G}(\mathbf{x})$ makes non-invertibility seem very ‘unlikely’ in the following sense.

Assume, for the moment, that $\mathbf{G}(\mathbf{x}_0)$ has a column space such that property (8.5.1) does not hold at $\mathbf{x}_0$. The vectors that form the column space of $\mathbf{G}(\mathbf{x})$ vary continuously with $\mathbf{x}$. Therefore, by choosing an open neighbourhood of $\mathbf{x}_0$ that is sufficiently small, we can reasonably assume that the column vectors of $\mathbf{G}(\mathbf{x}_0)$ vary continuously by an arbitrarily small amount, as will the column space of $\mathbf{G}(\mathbf{x}_0)$ vary by a small amount⁹ in $\mathbb{R}^q$ with respect to certain higher-dimensional subspaces that it is nearly orthogonal to. Since the $(q - n) - \text{dimensional}$ subspace is a ‘thin’ subspace of $\mathbb{R}^q$, we expect that a nearly orthogonal n -dimensional column space of $\mathbf{G}(\mathbf{x})$ can vary on an open region of $\mathbb{R}^n$ without intersecting this higher-dimensional subspace. In this case $\phi(\mathbf{x}, \theta)$ will be invertible on an open region of $\mathbb{R}^n$. Formally proving and characterizing the extent of these open regions remains an open problem.

Now that we have seen the conditions under which ϕ is locally invertible, our next task is to describe how a learned manifold constraint may be imposed on this local region of a nonholonomic network’s state space.

Although we have presented a solution based on BR-PMA here, any other learning algorithm that finds a Principal Manifold representation for a given foliation can be used just as well. What is important is that we have shown in the next section that we are guaranteed to be able to locally control a nonholonomic network on a region where the mapping $\phi(\mathbf{x}, \theta)$ is invertible.

8.5.1 Adaptive Compensation for Nonlinear Constraints

In Chapter 6 we saw that an additive compensator could be used to enforce linear manifolds as constraints on the dynamics of a GNF network. The essential characteristic of linear manifolds that made this possible was the existence of a coordinate transformation that enabled the projection of the dynamic trajectory tangent

⁹As measured, for example, by the gap metric of Chapter 7.

onto the distribution $\Delta^\perp$ orthogonal to the manifold tangent space. This projection represents the unwanted dynamical component which could be subtracted from the uncontrolled dynamics. The remaining velocity vector is confined to the manifold tangent space and therefore the trajectory cannot escape the manifold regardless of the driving inputs.

The reason we did not apply this technique to enforce *nonlinear* manifolds in Chapter 6 was that we lacked an appropriate coordinate transformation for the nonlinear case. Rather than a simple constant linear transformation that applies globally to the unchanging tangent spaces associated with a linear manifold, we require a transformation that changes with each point $\mathbf{x}$. This transformation must align the local d – *dimensional* space tangent to the manifold containing $\mathbf{x}_0$ such that it is spanned by the first d canonical arcs of $\mathbb{R}^n$. Other than this, exactly the same compensation technique applies when the manifold is nonlinear. Finding such a mapping through ad hoc means seems a daunting task. As the following theorem shows, however, the mapping $\phi(\mathbf{x}, \theta)$ parameterized by the foliation represented by $\theta \in V$ is *exactly* the coordinate transformation that is needed whenever its inverse exists.

Theorem 8.7

The order- $\hat{q}$ regularized RBF mapping $\phi(\mathbf{x}, \theta)$ locally transforms the leaves of the foliation represented by θ to flat linear manifolds spanned by the first d coordinate axes on any open region where the inverse $\phi^{-1}(\mathbf{x}, \theta)$ exists.

Proof: It is clear that the Jacobians of the two coordinate transformations must be related as follows

$$\left. \frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \right|_{\mathbf{z}=\phi^{-1}(\mathbf{x})} \frac{\partial \phi^{-1}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{I}. \quad (8.5.2)$$

But we know from Theorem 8.2 that the first d columns of $\left. \frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \right|_{\mathbf{z}=\phi^{-1}(\mathbf{x})}$ span $\hat{\Delta}$ at every $\mathbf{x}$. Therefore, $\frac{\partial \phi^{-1}(\mathbf{x})}{\partial \mathbf{x}}$ takes $\hat{\Delta}$ to the canonical d -dimensional

subspace $\text{span}\{\varepsilon_1, \dots, \varepsilon_d\}$ and Theorem 8.7 is true as claimed¹⁰. The Jacobian $\left. \frac{\partial \alpha(\mathbf{z})}{\partial \mathbf{z}} \right|_{\mathbf{z}=\phi^{-1}(x)}$ similarly takes the canonical subspace (in $\mathbf{z}$ coordinates) to $\hat{\Delta}(x)$. ■

The use of the term constraint linearizing network should now be clear for the networks described in Section 8.2. The effect of the network is to implement a mapping which linearizes a nonlinear constraint manifold. Consequently, we have the following result that states that manifold constraints can be applied to nonholonomic GNF networks on open regions of state space.

Theorem 8.8

An adaptive compensator exists which can enforce a learned manifold on an open neighbourhood of each sample point.

Proof: From Theorem 8.5 we know that the regularized RBF mapping $\phi(\mathbf{x}, \theta)$ is diffeomorphic on an open region around each sample point. Combining Theorem 8.7 with Theorem 6.3 in Chapter 6 we can immediately infer the form of the compensator.

Theorem 6.2 in Chapter 6 tells us that an additive compensator of the form

$$v(x) = \mathbf{P}_{\Delta}(\mathbf{x}) \mathbf{p}(\mathbf{x}) .$$

where

$$\mathbf{P}_{\Delta}(\mathbf{x}) = \mathbf{M}^{-1}(\mathbf{x}) \dot{\mathbf{P}} \mathbf{M}(\mathbf{x}) .$$

$$\mathbf{M}(\mathbf{x}) = \frac{\partial \phi(\mathbf{x}, \theta)}{\partial \mathbf{x}} . \mathbf{M}^{-1}(\mathbf{x}) = \underbrace{\frac{\partial \phi^{-1}(\mathbf{z}, \theta)}{\partial \mathbf{z}}}_{\mathbf{z}=\phi(\mathbf{x}, \theta)} . \text{ and}$$

$$\mathbf{p}(\mathbf{x}) = \tau(x) + \mathbf{q}(\mathbf{x}) . \quad \text{such that} \quad \mathbf{q}(\mathbf{x}) \in \Delta(\mathbf{x}) .$$

will enforce a general nonlinear manifold constraint given by distribution $\hat{\Delta}$, provided that a coordinate transformation $\phi(\mathbf{x})$ can be found that ‘flattens’ $\hat{\Delta}$ at every state in $\mathbf{U}$. ■

With Theorem 8.8 we now have a compensator which can be applied to a GNF network to modify its dynamics.

¹⁰These are direct consequences of (8.5.2) and Frobenius’ Theorem.

8.6 Chapter Conclusions

In this chapter we have described a connectionist implementation of the PMA algorithm. We have shown that with a random sampling of radial basis receptive fields, it is possible to approximate a complete partition of a state space into nonlinear manifolds, from noisy, random observations of dynamics on leaves of this partition. To our knowledge this is the first connectionist algorithm which achieves this.

This implementation paints a new class of Holonomic networks that directly learn an entire partition of the state space into nonlinear manifolds. Furthermore, these networks have recurrent dynamics that are restricted to any desired leaf of this foliation. In addition, these networks do not have intrinsic drift dynamics and so are ideal as components of a larger hybrid system. We stress that the network learns a complete partition of the space into manifolds rather than just a single manifold. It is therefore applicable to situations where multiple non-overlapping constraint manifolds apply. Further, the manifold that is learned is transparently represented in the network unlike most other types of connectionist knowledge of structures which are not easily interpreted by a human.

To our knowledge, this is the first connectionist system which explicitly represents the structures that shape the dynamics of the network. This means that the distribution can be directly read and manipulated, which is a significant advantage in the myriad domains where nonlinear control is used and such distributions are a key aspect of the underlying theory. We have also shown that this network is analogous to more traditional recurrent nets that have intrinsic drift dynamics, in the sense that a holonomic network can be combined with a second connectionist network to learn desired drift dynamics on each manifold in a foliation.

We have also described how to construct a hybrid connectionist control system, derived from a GNF network, that is able to enforce a manifold constraint on any region of the network state space that satisfies a given property. We have

argued that an open region of the state space will exist that satisfies this property.

The results described above are due to the novel BR-PMA algorithm that implements the PMA technique described in Chapter 7. We have shown that the weight dynamics of BR-PMA have a number of desirable characteristics. First, the stable equilibria of the BR-PMA algorithm are all proven to be good approximations to the training data. Therefore, the algorithm does not suffer from the problem of local minima. Second, it can be shown that the representation of the foliation at every sample point is a linearly independent basis that monotonically improves over time. This can be important if learning time is limited or intermittent, and the learning machine must make use of an approximation that is the 'best approximation available so far'.

Finally, BR-PMA is insensitive to initial conditions, and will probably converge to a correct solution from almost any initial weight value. In addition to the Holonomic Network described above, we have also introduced a PMA network that captures the foliation approximately and a constraint linearizing network that learns a coordinate transform that 'flattens' a learned nonlinear partition of the state space.

The results of this chapter are presently being compiled as five papers. In the first [Gra98j], regularized PMA representation and PMA networks are described. The second, [Gra98k], outlines the BR-PMA learning method. In the third [Gra98e], Holonomic networks are described in conjunction with PMA networks. The fourth paper [Gra98f] describes the control of GNF networks through additive compensation, and discusses the possibility of the non-obtrusive enforcement of constraints in GNF networks by state feedback control. Constraint linearizing networks are also discussed in this paper. Finally, [Gra98c] describes the application of Holonomic nets to the enforcement of connectionist types.

Chapter 9

Conclusions and Directions for Future Research

In this thesis we have made several novel contributions to the study of connectivist computation. Our study has been cross-disciplinary, bridging the theory of nonlinear differential geometric state space control and that of computation in a very general family of continuous-time recurrent neural networks.

Specifically, we have developed techniques that permit continuous-time recurrent networks to learn a partition of the network activation space into embedded submanifolds on which the network dynamics must evolve when driven by arbitrary, piecewise continuous input signals. These surfaces, which determine the controllability and observability characteristics of the network, will be called manifold constraints.

We have presented manifold constraints and their associated machinery, as a novel tool in the network designer's kit for determining and reasoning about what a network learns and how this knowledge shapes its run-time behaviour. Our focus has been on the link between learning and the mathematical structures that determine the effect that input signals have on run-time dynamics - a subject that has received

little attention in previous studies. The application of manifold constraints as a computational abstraction is particularly well suited to problem domains in which distinguished manifolds play an important role - domains as diverse as robotics, flight control, and communications. Like the energy landscape paradigm of connectionist computation, manifold constraints encourage the designer to visualize what the network is learning and how these structures are used at run-time. Yet these structures have a rigorous mathematical realization as a locally completely integrable differential geometric distribution that can be lifted directly from the network weights and employed in the construction of other devices. In this sense they are significantly more transparent to human understanding than many connectionist learning techniques.

The most tangible novel contributions made by this thesis are described in Chapters 7 and 8. First we have developed a general method for learning complete, nonlinear foliations (partitions) of a neural state space from noisy observations of the environment dynamics. Second, we have described a connectionist realization of this method as an unsupervised, batch weight-update algorithm. Third, we have shown that this learning method has a number of important characteristics:

1. It produces locally accurate approximations to the foliation that globally improve with the number of neurons and number of training samples.
2. It almost certainly converges to a good approximation independent of the initial weight vectors.
3. It is not trapped by local minima.

Fourth, we have developed a neural network called a PMA network, which learns to directly represent the locally completely integrable distribution associated with the approximated foliation: a mathematical structure that has great utility in the study and control of nonlinear dynamics. Fifth, we have developed a constraint linearizing network that learns a nonlinear change of coordinates that locally

linearizes the manifolds in the foliation. These linearized coordinates not only significantly simplify application-oriented mathematical operations on the foliations, but also simplify the thought process. The sixth contribution is a novel class of continuous-time recurrent neural network, based on PMA networks, that we call Holonomic networks. As the name implies, the run-time dynamics of these networks are confined to the nonlinear leaves of a foliation that is learned by direct modification of the weights, through BR-PMA, as opposed to more complicated control machinery such as additive compensation or CID feedback control. These Holonomic networks have no intrinsic drift dynamics and are therefore ideal components in larger hybrid AI systems, although the addition of a recurrent layer can add desired drift dynamics that are consistent with the learned manifold constraints. Seventh, we have suggested a way to augment a GNF network in order to impose learned manifold constraints on its dynamics. In addition, we have illustrated how manifold constraint learning can be used in conjunction with fixed-point learning for the case of linear foliations. These results can be locally extended to the nonlinear case by replacing novelty filter learning with the BR-PMA algorithm.

In addition to the manifold constraint abstraction and its attendant machinery, we have also performed a novel study of the controllability and observability characteristics of a very general family of continuous-time recurrent neural networks - the Grossberg Normal Form (GNF) networks - from the viewpoint of differential geometric state space control.

In Chapter 2 we introduce the reader to the main conceptual and mathematical tools that form the background of our study. This includes a description of the GNF networks, a family that we have selected to motivate our work. Chapter 3 gives a survey of the relevant control theoretic ideas and tools that we use throughout the subsequent chapters, and a description of where our work fits into the literature on connectionism and control.

In Chapters 4 and 5, we ask two questions that are central to an understanding of the dynamics of any nonlinear, input-driven system. The first, called the

controllability problem, asks: “Which states can the system attain for a general class of input signals?” The second, the dual problem of observability, asks: “How much of the internal system dynamics is visible to an external observer, and how much is hidden?” In Chapter 4, we explore the problem of adaptively learning controllability manifolds for GNF networks. We discover that, in general, learning algorithms do NOT exist for robust learning of controllability manifolds. We also find that several important special case networks do permit robust learning algorithms; however, the class of learnable manifolds is very limited in each of these cases. In Chapter 5 we show that these controllability results have a dual counterpart for the problem of learning observability manifolds.

Our main results are being compiled into the papers [Gra97, Gra98g, Gra98b, Gra98i, Gra98d, Gra98a, Gra98j, Gra98k, Gra98e, Gra98c, Gra98f].

9.1 Directions For Future Work

The work presented in this thesis should be viewed as providing a framework for further exploration of the use of manifold constraints in connectionist computation. There is much room for improvement of the techniques used here, and many extensions have suggested themselves during the course of this study.

9.1.1 Empirical Studies

First and foremost, of course, this work needs to be supported by extensive simulation studies. The C++ framework, VISICON, described in Appendix C and [Gra98n], provides many of the software tools that will be needed to begin this empirical studies.

Of immediate interest is the degree to which ill-conditioned matrices will be a problem. Such matrices will arise if the evolving principal component fields of the BR-PMA algorithm are nearly linearly independent locally at each sample point. A number of methods have been proposed in the connectionist PCA literature for

improving the independence of the principal vectors and these methods should carry over directly to the BR-PMA setting.

9.1.2 Reducing the Number of Neurons

In our work we have assumed the presence of an unlimited number of neurons - a number larger than the number of training samples. We would like to study the effect of reducing the number of neurons on the BR-PMA algorithm stability.

In the event that $\hat{q} < np$ (see Chapter 8), the local ODE trajectory (at each instant of time) minimizes the squared error norm between the PCA trajectory tangent and its projection on the space spanned by the RBF gradient vectors. The conditions on the RBF centers that guarantee stability and the distance of the stable representations and good PMA representations remain open problems.

9.1.3 Other Families of Mappings

We chose to study the family of regularized RBF mappings in detail. However, in the process we encountered and initiated studies on other promising classes of mappings as well.

For example, we recommend studying the class of triangularized mappings of the form

$$\phi(\mathbf{x}, \theta) = \begin{bmatrix} \phi_1(\mathbf{x}_1, \theta) \\ \phi_2(\mathbf{x}_1, \mathbf{x}_2, \theta) \\ \vdots \\ \phi_n(\mathbf{x}_1, \dots, \mathbf{x}_n, \theta) \end{bmatrix} \text{ where } \frac{\partial \phi_i}{\partial \mathbf{x}_i} \neq 0 \quad \forall \mathbf{x} \in \mathbb{R}^n.$$

This class of mappings has the virtue of being *globally* invertible. The derivation of the ODE $\hat{\theta}$ is complicated by the triangular form of the Jacobian of $\phi(\mathbf{x}, \theta)$; however, these issues do not appear to be insurmountable. Global invertibility would make the application of PMA networks for global control of plants

tractable. Some initial results on this are currently available, but have not been included here as they are still only early sketches, and are not fully worked out.

9.1.4 Complete Characterization of Regularized RBF Inverse

We have given a partial characterization of the regions of invertibility of regularized RBF mappings. A more thorough study is called for. This would be of use when determining which areas of a GNF network's state space are to be constrained by a manifold.

9.1.5 Discrete-Time BR-PMA

Although we have not followed up on discrete-time BR-PMA, it is clear that the analysis methods of Chapter 8 can be applied to a discrete-time version of the original scheme. In this case the weight update scheme would be a discrete time algorithm, while the run-time architecture of the PMA network and Holonomic network would remain continuous-time.

9.1.6 Transporting Manifold Constraints Between Networks

The completely integrable distribution that is learned and interpreted by a PMA network is, in a very real sense, a data structure that can be transported from one network and installed into another. In the event that the target network has a higher-dimensional activation space, a diffeomorphic manifold mapping $\psi(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m, m \geq n$ induces the differential map $\left. \frac{\partial \psi(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\psi^{-1}(\mathbf{y})}$ which can be used to directly convert the weight matrix in the source network into a weight matrix in the destination network with no loss of information.

Conditions under which the foliation can be mapped into a smaller activation space while retaining the complete integrability of the source distribution is an open question worthy of study.

Figure 9.1: Competitive PMA Network

9.1.7 Learning Multiple Environmental Foliations by Competitive PMA

The learning model that we describe in Chapter 7 assumes that a single underlying foliation shapes the environment dynamics. It is interesting to consider how we might extend PMA to model the case where each environmental event (i.e. trajectories) may be shaped by a different choice of one of many different foliations.

Consider the competitive PMA described in Figure 9.1. In this system, a collection of PMA networks individually track an observed environmental trajectory

$\mathbf{x}(t)$, and compete to learn from the observed signals. We will assume that the BR-PMA algorithm is employed by each PMA network. For the purposes of discussion, we will also assume that a sample-and-hold mechanism is available to periodically capture an estimate of $\mathbf{x}(t)$ and $\dot{\mathbf{x}}(t)$, and that this is presented to each PMA network.

At arbitrary time t_0 we will assume that the weight vectors θ_i , $1 \leq i \leq m$, are randomized, and therefore the foliations represented by each network have a “random” distribution $\Delta_i(\mathbf{x})$ that respects the constraints of Chapters 7 and 8. We will also assume that the measure $e(t)$ of deviation of the last T trajectory tangent estimates for the current foliation is available from each network at the beginning of an environmental event. $e(t)$ is a measure of how well the current foliation estimate of each network matches the initial segment of the environmental dynamics. These measures are available to a gating controller which, in its simplest form, outputs a 1 for the network with maximum measure - the winner - and 0's for the others - the losers - making a random choice in the event of a tie.

The BR-PMA algorithm in the winning network is enabled by the gating signal, and the algorithm is executed for the last wT samples that the network has previously “won”. In this way, the PMA nets compete to learn, and the foliation that is closest to the current trajectory will be modified to better represent the trajectory. We conjecture that over time, through competition, the foliations that are learned by each PMA will become more specialized at representing trajectories produced by a particular underlying foliation.

We consider these issues further in the paper [Gra98h].

9.2 Manifold Constraints for Continuous-time Pattern Recognition

Although we have concentrated on the problem of generating input driven trajectories that satisfy a manifold constraint, we have also developed most of the machinery required to *recognize* trajectories that obey the constraint. This permits

us to think of manifolds as novel data structures that support the recognition of continuous-time patterns.

We may ask ourselves: ‘Why would we be interested in this capability?’ The domain of continuous-time speech recognition offers a good example. After appropriate preprocessing, the speech signal can be reduced to a continuously varying trajectory $x(t)$ of a measurement vector through a spectral space. Repeated utterances of the same speech unit have a strong tendency to produce similarly shaped trajectories. Various authors (e.g. Kohonen¹) have observed that repeatable variations in the speech production process tend to distribute trajectories over a lower dimensional submanifold of speech space. Different utterances follow differently shaped manifolds, the dimensionality of which may also vary. The ability to learn these manifold shapes gives us a novel data structure which can be used as a prototypical speech template. Existing template methods such as Nearest Neighbour, Dynamic Time Warping, HMMs, and Time-delay Neural Networks attempt to build a discretized model of these manifolds. The BR-PMA method provides us with a method to directly model the underlying continuous speech structures by observing the speech train. Furthermore, since the method actually learns *an entire foliation* of manifolds that represent the variation in observed trajectories, it generalizes from observations in a principled way. Also, at least in principle, there is no ‘analysis window’ during which recognition must be performed. Rather, the match may be performed on trajectories of arbitrary length. However, we need a way to measure how well a speech trajectory matches a learned manifold.

Recall that a PMA network learns the basis of the tangent space $T_{\mathbf{x}}\mathcal{M}$ of a learned manifold. Each basis vector is given by $\frac{\partial \alpha(\mathbf{x}, \theta)}{\partial \mathbf{x}_j}$, represented by one of the d output pools of the network. This basis set permits us to compute the projection of the normalized speech signal velocity $\mathbf{v}(t) = \frac{\dot{\mathbf{x}}(t)}{\|\dot{\mathbf{x}}(t)\|}$ onto the space locally orthogonal

¹In an unpublished work at the Computing Research Laboratories of Nortel, the author also observed this effect.

to the manifold, i.e.

$$e(t) = 1 - \frac{1}{T} \int_{t-T}^t \|\mathbf{P}_{\Delta^\perp}(\mathbf{x}(s)) \mathbf{v}(s)\| ds$$

or

$$e(t) = 1 - \frac{1}{k-j} \sum_{i=j}^k \|\mathbf{P}_{\Delta^\perp}(\mathbf{x}(\mathbf{t}_i)) \mathbf{v}(\mathbf{t}_i)\| \quad \text{for discrete time } t_j \text{ to } t_k$$

where

$$\mathbf{P}_{\Delta^\perp}(\mathbf{x}) = \mathbf{I} - w(w^T w)^{-1} w^T \quad \text{and} \quad w = \left[\frac{\partial \phi(\mathbf{x}, \theta)}{\partial \mathbf{x}_1} \dots \frac{\partial \phi(\mathbf{x}, \theta)}{\partial \mathbf{x}_d} \right]. \quad (9.2.1)$$

This error provides a measure of how well the speech train conforms to the manifold over a period of T time units. As long as it is small, the speech signal conforms to the manifold. If it becomes large, it is an indicator of the fact that the signal has deviated from the original manifold. Although the estimate of $\dot{\mathbf{x}}$ is subject to noise, the averaging effect of the integral in (9.2.1) will tend to dampen this. This approach has the benefit of being invariant to the velocity of the speech train in measurement space. Furthermore, the signal is small as long as the speech train follows any leaf of the foliation, permitting some degree of robustness to the absolute position of the speech vector in measurement space as long as the general ‘shape’ of the foliation is followed. We conjecture that these qualities might be useful to promote generalization and robust recognition.

Another approach which gives a measure of absolute distance to a template manifold might be to generate a ‘shadow’ trajectory on the learned manifold which acts as a prediction of the speech signal if the true trajectory does not deviate significantly from its initial position on the manifold. To generate the shadow $s(t)$, we would employ the local projection of the true velocity $v(t)$ onto the tangent space at the shadow $s(t)$ to obtain the next shadow tangent. The absolute Euclidean distance $|v(t) - s(t)|$ would give a measure of how far the true signal has deviated

from the ‘prediction’. We refer the reader to Section C.3.4.3.2 in Appendix C for a discussion of shadow trajectories.

An apparent limitation of the application of PMA to speech recognition is that these manifold templates do not incorporate information regarding the expected direction of *flow* on the manifold. Therefore, we may wish to employ a second network as in Figure 3.5 in Section 3.7.3 to learn this information and thus resort to Mussa-Ivaldi’s vector field approximation technique. Experimentation is needed to determine the effectiveness of the recognition technique with and without this information.

Another limitation is that the foliation that is learned cannot be self-intersecting. One solution to this might be to extend the measurement space to include measurement time as an $(n + 1)^{th}$ axis. In such a case, self-intersection would be avoided by ensuring that a point x in $\mathbb{R}^n$ that occurs at different times t_1 and t_2 actually represent separate points (x, t_1) and $(x, t_2) \in \mathbb{R}^{n+1}$ respectively. Alternatively, separate manifolds might be learned that represent non-intersecting patches of speech space. A speech signal could then be significantly compressed into a sequence of symbols that represent the patches that it occupies.

It is important to realize that this use of manifolds for pattern recognition purposes is equally applicable anywhere conformance to a single manifold or foliation is important. For example, this approach might be used to recognize the performance of ritual gestures in a robotic system or the compliance of an aircraft control signal to manifolds that represent desired or poor strategies in a flight training exercise.

Certainly, this approach to recognition deserves a closer look. We describe these ideas further in the paper [Gra98m].

Appendix A

A Philosophy and History of Connectionism

This thesis is obviously not for the novice connectionist, however, we do not want such a reader to be completely hindered from following the contents of the thesis. The present brief primer in the field is by no means complete, but it offers the novice an introductory overview and pointers into the literature.

A.1 The Origins of Connectionism

Thousands of years of myth and philosophical thought attest to man's desire to understand the mechanisms of thought. One of the modern expressions of this goal is the study of machine intelligence. To understand the history of connectionism we must consider the on-going dialectic between two rather different approaches to this problem that had arisen by the 1950's.

The cybernetic school of thought had its roots in the control theory community, where the analysis and design of continuous time systems predominated. Cybernetics took its inspiration from biological adaptability, and was primarily concerned with pattern recognition and machine learning. On the other hand, the school

of artificial intelligence was motivated by cognitive psychology. Its focus was on developing computer programs capable of problem solving (e.g. task planning, game playing). The conceptual tool underlying these programs was usually some type of formal logic.

Both schools naturally turned to the human brain for insight. The cyberneticists found the low-level anatomical structures of the brain suggestive of feedback control systems. Consequently, they took a bottom-up approach to modeling intelligence. The AI community, with its interest in the psychology of high level reasoning, employed symbolic logic to develop programmatic models of intelligence in a top-down fashion.

Connectionism has its roots in the cybernetic school of thought, and shares its preoccupation with neurophysiology and low-level cognitive processes. The term *connectionism* refers to the anatomical fact that biological nervous systems are composed of massively interconnected networks of information processing cells called *neurons*. Connectionist systems model these neural networks with varying degrees of biological fidelity. In Section A.3 we examine some features typically incorporated into these models. The computation performed by the network is controlled by parameters that weight signals sent from one neuron to another at a connection point.

During the late 1950's, and early 1960's the cybernetic movement gradually fell out of favor as the AI community became more interested in the cyberneticist's traditional domain of pattern classification. The last vestiges of cybernetics disappeared in the 1970's as research into knowledge base acquisition encompassed the study of machine learning. Today, connectionism is considered a specialization of the AI sub-discipline, *machine learning*. It is often grouped together with the study of cellular automata, genetic algorithms, and classifier systems under the heading of *sub-symbolic AI*.

A.2 A Brief History of Connectionist Models

The earliest work in connectionist systems dates from the 1940's. At this time McCulloch and Pitts [MP43] attracted attention to neural modeling by demonstrating that networks of simple neuron-like threshold elements were computationally equivalent to the propositional calculus. Hebb [Heb49] also suggested an intuitively appealing principle of self-organization for networks of neurons that prompted further work into methods of adaptation and learning. Progress was slow until Rosenblatt [Ros58] published the influential perceptron model in the late 1950's.

Rosenblatt demonstrated how a single layer of simple neuron-like thresholding units could learn to classify binary patterns from training data. Furthermore, he provided a formal proof of convergence for the learning algorithm. This was a startling result for its time, and led to a burst of activity and optimism in the field. To many it seemed that an important key to intelligence had been discovered. Widrow [WH60], Block [Blo62], and many others developed and analyzed variations on the perceptron, and there were many attempts to exploit the adaptive pattern classification capabilities of the model in practical applications. Although the perceptron, undoubtedly, represents a breakthrough in our understanding of how to organize complex coalitions of neurons in order to produce useful global behaviour, by the late 1960's the severe limitations of the approach were becoming apparent. In 1969, Minsky and Papert [MP69] published an insightful and rigorous analysis of the model, stressing the limitations of its classification capabilities. The major weakness was its inability to form anything other than linear discriminant boundaries. Although it was recognized that a multi-layered perceptron would be able to overcome this limitation, a formal strategy for training such a network was unknown. Academic interest in neural networks of all types quickly dissipated, even as economic support for connectionist research became scarce.

During this period when research on the perceptron was promising, a few researchers such as Grossberg [Gro67, Gro68] studied alternative continuous time

recurrent neural network models, where the network is viewed as a nonlinear dynamical system. Although this approach is more faithful to the biology of neural networks, it drew very little attention. Dynamical networks were considerably more difficult to analyze and simulate than the perceptron model, and simultaneously had less intuitive appeal. Furthermore, they did not appear to offer better classification capabilities than the perceptron at a time when the emphasis within the community was primarily on pattern recognition.

Alongside the work on connectionist systems done during the 1960's, methods of problem solving based on deductive inference developed at a steady pace. With the 'collapse' of the perceptron, symbolic logic¹ became a mainstay for knowledge representation and reasoning. This was a time of intense interest in expert systems, and the first steps toward logic-based learning algorithms were being taken. Despite this emphasis on high-level symbolic knowledge, some researchers continued to develop connectionist systems. Communication among these remaining connectionists was limited, and their published work was scattered among many obscure journals.

During this decade there were a few developments that, in hindsight, were clearly important. Werbos [Wer74] developed a supervised parameter estimation method capable of training a multi-layer perceptron. His networks were able to adaptively learn nonlinear discriminants, thereby overcoming the major weakness of the perceptron algorithm. Unfortunately, the significance of Werbos' method was not realized until much later. It sank into obscurity only to be eventually rediscovered independently several times in the years to follow. Grossberg [Gro76] continued with his studies of continuous time dynamical networks, intent on explaining various perceptual phenomena from the cognitive science literature. [ASRJ77] analyzed a type of dynamical pattern classification network called the brain-state-in-a-box, or BSB model. Kohonen [Koh77] developed several networks including associative memories for robust, context sensitive pattern recall, and the self-organizing map

¹When we use the term 'symbolic logic' we include systems that employ various techniques, such as "frames" and "procedural attachments", to increase the efficiency of inference.

for dimensionality reduction. Although all of these models would eventually become influential, they were largely ignored until the early 1980's.

By this time, the limitations of the logic-based approach to AI had become apparent. These symbolic systems are extremely sensitive to noise. They do not generalize knowledge well outside of a very limited domain. They do not scale well as rules are added to the knowledge base, and they perform poorly on problems requiring simultaneous satisfaction of multiple constraints. Alternative methods of knowledge representation and reasoning began to look attractive again. Interest in early connectionist models reawakened, as did interest in evolutionary computing models. A research program was launched at Brown University into localist neural networks for knowledge representation. A group of researchers at the University of California's San Diego campus formed the Parallel Distributed Processing (PDP) group, and began a concerted effort to surpass the limitations of the old perceptron models. New developments began to appear in rapid succession.

In 1984, Hopfield [Hop84] published a paper in the Proceedings of the National Academy of Sciences in which he analyzed a dynamical neural network for associative recall of patterns using a corrupted key. The high visibility of the paper, and its clear and simple proof of convergence, touched off a resurgence of interest in connectionist systems. By this time, Rumelhart and others [RHW86] at the PDP research group had rediscovered Werbos' algorithm for training multi-layer perceptrons, calling it back-propagation, and demonstrated its powerful classification capabilities through empirical studies. Cohen and Grossberg [CG83] had derived sufficient conditions for the stable run time behaviour of a wide class of continuous time and discrete time networks. Ackley, Hinton and Sejnowski [AHS85] developed a statistical network model called the Boltzmann machine describing the run time convergence of Hopfield's model as an annealing metal. The PDP group also published a two volume work [RM86a] building a case for the novel knowledge representation properties of connectionist systems. These books were responsible for increasing the general awareness of connectionism in the AI community, and many independent

research groups began to appear. Around the same time, Carpenter and Grossberg [CG86] published a connectionist theory of unsupervised pattern clustering called adaptive resonance theory. Jordan and Elman [?] proposed recurrent versions of the multilayer perceptron for the classification of time series. In 1986, the first conference on neural networks attracted over 2,500 attendees. Over the next few years, literally hundreds of variations on these models were proposed and studied. However, most of this work was of an empirical nature with little or no theoretical basis. By 1989, this surge of activity had ebbed somewhat as new limitations were encountered and it became clear that a more formal theory was needed.

Although many of the new models were based on techniques from optimization theory or statistics, the nonlinear nature of the networks made a detailed analysis of their performance difficult. Very little was known about how well they would perform on particular problems, or even if they would converge to a solution at all. Empirical studies did verify that connectionist systems had a number of desirable properties that were complementary to those of symbolic AI systems. Classification and function approximation capabilities of neural networks were competitive with, or better than, classical techniques when applied to difficult problems. Connectionist networks performed very well on problems involving the simultaneous satisfaction of multiple constraints. Run time and training performance degraded gracefully in the presence of noise and hardware failures. The parallel nature of these networks allowed very rapid run time computation, even when the number of neurons was large.

On the other hand, these same studies also demonstrated serious limitations, and underscored the lack of theoretical tools to tackle them. For example, convergence rates during learning were very slow, making it difficult to scale networks up from toy problems to industrial-sized problems. Detailed predictions about the generalization capabilities, training set sizes, and effect of failures on performance would not be possible without better theory. Tuning of the various learning parameters and selection of appropriate connection patterns was almost a 'magical' art.

Inference in neural networks involved parameter adjustments in high dimensional spaces, and so explanations of how or what a network learned were not available in human readable form. Furthermore, it was not at all clear how neural networks might be applied to traditional AI problems involving structured, abstract knowledge and deductive inference. Indeed, many of these problems are still active areas of research today.

A few notable theoretical results emerged in the late 1980's. In 1987 Judd [Jud90] completed an investigation into the complexity of loading feed-forward neural networks². He clarified how loading complexity is a function of the degree of interconnectivity in the network, rather than the type of neuron equations. In 1989 Baum, and Haussler [BH89] used Valiant's theory of the learnable to give bounds on the number of training examples sufficient to guarantee valid generalization over novel inputs in feed-forward networks. Abu-Mustafa [?], Kosko [?], and others provided upper bounds on the number of patterns that could be successfully stored and retrieved in an associative memory without cross-talk. Barto [BA85] applied the theory of learning automata to the problem of credit assignment in reinforcement networks. Most results have been framed for popular discrete time, feed forward models rather than the continuous time, fully connected variety.

The connectionist approach has not been without its modern critics, some of whom have a very high stature in the AI community. Fodor and Pylyshyn [FP88] have championed the symbolic approach to AI, and have sparked an ongoing debate about the relative merits of connectionist models and logic based systems. This has been a healthy discussion, which has helped clarify the limitations of current models and focus subsequent research.

The 1990's have seen steady progress in both theory and the development of new models. The new modeling trends have been inspired by cross fertilization from

²The loading problem involves training a network to correctly map a set of inputs to associated output patterns. Loading is a special type of learning in which no generalization to novel inputs is required.

various branches of mathematics, as well as new findings from the neurosciences. Poggio [PG96] has put feed forward network learning on more solid theoretical foundations by relating it to the theory of multivariate function approximation using radial basis functions. Physiologically, it appears that there is some (weak) evidence that the temporal cortex may represent objects in terms of interpolated basis functions. Since training algorithms do not scale well, efforts have been made to break large networks into collections of smaller 'expert' modules. These can be trained faster, and their outputs can be recombined using a gating network at run time. Jacob and Jordan's [JJNH91, JJ94] mixture of experts is a statistical model that requires hand partitioning of the network, but automatically takes advantage of the natural modularity of the problem domain to train each module as a specialist. The discovery by Gray and Singer, that cells in different regions of the cortex fire in phase when processing related information, has led to work on networks of coupled oscillators. Some of these make use of frequency or phase correlation to do variable binding and pattern segmentation. Freeman's discovery [Fre94] of chaotic attractors in olfactory cortex dynamics has inspired further study of coupled chaotic dynamical systems as pattern classifiers. Using chaotic itinerancy as a method of generating sequences of states has now become a common theme in these studies [Kan90, Tsu94].

The control systems community has recently become quite interested in connectionist networks as nonlinear parameter identification schemes for adaptive controllers [PL95, LN96]. Most of this work simply uses existing network models with little modification as components of larger systems. Variations on the back propagation approach to weight adaptation continue to be developed. Cascade correlation and back propagation through time are popular examples. Constructive training algorithms have been developed which augment the network architecture by adding or removing neurons and synapses. A related approach is to use genetic algorithms to explore the space of networks through simulated evolutionary selection.

Theoretical developments in weight adaptation have mostly been driven by Valiant's learning theory. Characterizing the Vapnik-Chervonenkis dimension for

various connectionist systems remains an active area of study [BH89, KM94, Maa95]. Bounds on the complexity of learning FSM's have been published for recurrent feed forward networks. PAC training algorithms now exist for several of the classic connectionist models. Amari [Ama95] has employed differential topology to describe the set of statistical feed forward networks as a differential manifold in the space of exponential distributions. His approach yields powerful geometric insight into statistical training algorithms and has led to the *em* algorithm, which is a weight update procedure. The relationship between this algorithm and more traditional statistical methods has been explored. Various studies have found the underlying mathematical relationships between connectionist models and classical pattern classification techniques.

The control theorists Levin and Narendra [LN96] have given theoretical justification that discrete time nonlinear dynamical systems can be modeled as functions of a sequence of input-output pairs, and that feedback controller equations exist for such models. This is important justification for the use of recurrent feed forward networks as nonlinear identification schemes. Theoretical analyses of the computational power of connectionist systems have also been encouraging. Seigleman and Sontag [SS91, SS92] have shown that recurrent feed-forward networks can simulate universal Turing machines, and Maas [Maa94] has shown similar equivalences for networks of neurons with spiking output signals. Recent work, coming out of the hybrid control systems community [Bra95], has also shown similar simulation capability in a wide class of continuous time dynamical systems. These studies tell us that connectionist systems are, *in theory*, capable of powerful general purpose computation. However, how to achieve this power while remaining consistent with biological constraints remains an important open question. The current generation of connectionist systems are highly specialized in terms of the computations they can perform.

Finally, our history ends with a problem that has a prominent position along the path to understanding human reasoning from the connectionist vantage point. This is the problem of representing and computing with high-level structured knowl-

edge in a connectionist setting. The importance of the problem is underscored by its relevance to the development of more powerful and general purpose computation in connectionist computers. In this thesis we propose that control theory can offer useful insights into these issues. In the following section we discuss the relevance of control theory to enabling general purpose computation.

A.3 A Generic Connectionist Architecture

Connectionist research takes much of its inspiration and terminology from the neurosciences. The motivating idea is that we should base the architecture and organizational principles of intelligent learning machines on those found in the human brain. All connectionist systems attempt to do this to one degree or another. However, most model only the most basic features of real biological nervous systems. Details are ignored, partly in the interests of analytical tractability of the system, and partly because of our limited understanding of the role that they actually play in computation.

A.3.1 Neuroscience Basics

At a low level, the human brain is a network of about 10 billion information processing cells, called *neurons*. On the scale of a few cubic millimeters these cells are massively interconnected. On a larger scale, characteristic connection patterns occur between groups of functionally related neurons. Feedback between cell groups is a striking and ubiquitous feature that occurs at every organizational level. The detailed architecture and functional significance of this complex neural network is only now being teased apart by neuroscientists.

Individual neurons have hundreds of thousands of input channels, called *dendrites*, that carry input signals, or sensory data, from other neurons to the body of the cell. These signals are combined nonlinearly to produce a membrane potential called the *activation level* of the neuron. If this value exceeds a threshold level of

excitement—the *firing threshold*—the neuron outputs its own continuously varying signal. This signal may take the form of a train of spikes, or it may be smoothly varying. The relationship between the activation level of the neuron and its output values is highly nonlinear.

The *output signal* of a neuron reaches neighboring neurons via a large number of output channels called *axon collaterals*. The axon collateral forms a structure called a *synapse* at the point where it connects to another cell. Broadly speaking, there are two types of synapses: *excitatory synapses*, which tend to make the post-synaptic neuron more likely to output a signal, and *inhibitory synapses*, that make the post-synaptic neuron less likely to output a signal. Neuroscientists currently maintain that any given cell forms exclusively excitatory or exclusively inhibitory connections with other cells. A cell may receive a mix of excitatory and inhibitory connections however. The efficiency of signal transmission across a synapse—the *weight* of the synapse—can vary from synapse to synapse. Therefore, the degree of influence that one neuron has over another can be controlled by adjusting its synaptic weights.

Like neurons, synaptic weights also have dynamical behaviour. Although neuroscientists are beginning to understand the detailed chemical mechanisms that enable variation in synaptic weights, very little is known about the actual principles that determine how weight changes are made to accomplish specific computations. Even though many theories have been proposed, no coherent and compelling picture has emerged. A common thread that winds its way through these theories is the idea of a Hebbian learning law.

In the 1940's, the psychologist Hebb [Heb49] proposed a correlative weight update law that increases the magnitude of an excitatory synapse between two cells that are active simultaneously, and diminishes it if the cell activities are out of phase. Intuitively, this law will increase the likelihood that the pattern of firing will occur again for a given set of external inputs, and in this way the neural network can learn to associate input patterns with output patterns. Many variations on this theme

have been proposed in order to learn desirable network behaviours. In fact, there is neurophysiological evidence that Hebbian mechanisms do account for at least one type of synaptic update.

A.3.2 A Simple Mathematical Model

A.3.2.1 The Connection Pattern

In keeping with this simplistic view of neural information processing, connectionist systems consist of a network of n simple processors, called neurons, that are massively interconnected. A network will, in general, have n^2 synaptic connections between neurons, and another nm synapses that connect m external inputs to the n neurons. The inter-neuron and input synapses are represented by real valued matrices of weights, $\mathbf{H} \in \mathbb{R}^{n \times n}$ and $\mathbf{J} \in \mathbb{R}^{n \times m}$, respectively. The synaptic connections terminating on the i^{th} neuron are represented by the i^{th} rows of these matrices. Excitatory synapses are represented by strictly positive weights, and inhibitory synapses by strictly negative weights. We will refer to neurons that connect with other neurons via both excitatory and inhibitory weights as *heterogeneous neurons*; neurons that connect via only one type of weight will be called *homogenous neurons*. A connectionist network that contains only one kind of synapse will be called a *homogenous network*.

Not all connectionist models are fully interconnected. Various interconnection patterns have been studied, although the reasoning behind these schemes is not always driven by theoretical considerations. A very popular approach [Ama95, PG96, RHW86] is the feed-forward architecture shown in Figure A.1(a). There is theoretical support [Cyb89, Fun89] that a two layer feed forward network can represent any desired nonlinear function to any desired degree of accuracy, given sufficient numbers of neurons. Another popular network is a recurrent feed-forward network (see Figure A.1(b)). Recent theoretical results [SS91, SS92] show that such networks are able to encode universal Turing machines, and are therefore computationally very

Figure A.1: Typical Connectionist Network Architectures

powerful. Various attempts have been made to divide networks into modules which can be trained as specialists, and whose outputs can be combined by other networks to produce a desired output [JJNH91, JJ94]. These generally follow the feed-forward style of network. The fully interconnected network in Figure A.1(c), has also often been employed [AHS85, ASRJ77, CG83, Gro67, Hop84, MP91], and it generalizes all other architectures. In this thesis we will consider variations in connection patterns to be fully interconnected networks in which a subset of the synaptic weights are permanently set to zero. All other weights are either positive excitatory connections, or negative inhibitory connections.

A.3.2.2 Neuron Dynamics

Neurons in a network have dynamical behaviour. In some models, such as the feed forward back propagation networks and basis function networks, these dynamics are not emphasized because the network performs a one-step mapping between an input pattern and an output pattern. Even so, in these cases we may consider the network to have a trivial dynamics defined over a single time step. The passage of time can be continuous or discrete depending on whether ordinary differential equations or difference equations are used to describe the dynamics. Both

approaches are used in the literature.

Although both approaches have been explored, discrete time models are more popular because they are easier to encode and simulate by digital computers. What then is the motivation to work with continuous time models? First, neurobiologically faithful simulations of nervous tissue are usually expressed in terms of empirically derived continuous time equations, and often incorporate physical laws that are expressed as differential equations. Second, the preferred method of analysis can dictate the use of continuous time equations. When useful theorems are expressed in terms of continuous time systems, it is convenient to express the network equations in these terms too. Another reason is the generality of approach offered by continuous time dynamical systems. Branicky [Bra95] has recently shown that any difference equation satisfying very mild regularity conditions can be simulated by a continuous time system. The reverse is not true, however, and so by working in the continuous time domain we do not forfeit any generality.

A hallmark of connectionist systems is that the dynamics of neural processors are described by only a few simple functions. Almost all models studied to date are composed of neurons with the same type of functions, even though it is clear that many different types of neurons with different response characteristics are present in vertebrate brains.

The activation value $x_i(\mathbf{d}, \mathbf{u}, \mathbf{H}_i, \mathbf{J}_i)$ of the i^{th} neuron is a function of the n -dimensional vector $\mathbf{d}$ of output signals from neurons in the network, and the m -dimensional vector $\mathbf{u}$ of external inputs. It is further parameterized by the rows of synaptic weights $\mathbf{H}_i$ and $\mathbf{J}_i$ that terminate on the neuron. A typical discrete time activation function is given by the inner product³

$$x_i(t+1) = \langle \mathbf{H}_i, \mathbf{d} \rangle + \langle \mathbf{J}_i, \mathbf{u} \rangle = \sum_{j=1}^n \mathbf{H}_{ij} d_j(t) + \sum_{k=1}^m \mathbf{J}_{ik} u_k(t). \quad (\text{A.3.1})$$

³In the sequel the inner product between $\mathbf{x}$ and $\mathbf{y}$ will be denoted $\langle \mathbf{x}, \mathbf{y} \rangle$.

Figure A.2: Sigmoidal Output Function

and its continuous time equivalent is given by

$$\dot{x}_i(t) = f_i(x_i(t)) + \sum_{j=1}^n \mathbf{H}_{ij} d_j(t) + \sum_{k=1}^m \mathbf{J}_{ik} u_k(t). \quad (\text{A.3.2})$$

where function f_i determines the zero input dynamical behaviour of the neuron, which effectively dictates the natural rate of decay or excitation.

The output value of a neuron is a function of its activation value. The type of function used is usually motivated by the nonlinear relationship between membrane activation and firing threshold in biological neurons. In these cases the output function $d_i(x_i, \theta_i)$ is parameterized by an adjustable firing threshold θ_i , and the function shape is that of a sigmoid. Other models are motivated by mathematical theory, such as the radial basis function networks, and in these cases the output functions are determined by the approximating characteristics required by the theory. A typical sigmoidal output function of the form

$$d_i(x_i) = \frac{1}{1 + e^{-\theta_i x_i}} \quad (\text{A.3.3})$$

is shown in Figure A.2.

A.3.2.3 Network Dynamics

The state of a neural network is described by its n -dimensional vector of activation values, $\mathbf{x}$. Given an initial state vector $\mathbf{x}_0$, the state will evolve from this point along a trajectory through n -dimensional space according to the system of neuron evolution equations. Each neuron corresponds to an independent basis vector, and these collectively form a distinguished basis for the state space. The external inputs presented to the system will modify the trajectory that the network follows.

A number of popular models such as the BAM [Kos88], Hopfield network [Hop84], and BSB [ASRJ77] exhibit deterministic trajectories that evolve toward fixed points in state space. $\mathbf{x}_0$ represents a pattern retrieval key, and the system trajectory evolves within a basin of attraction to the nearest fixed point that represents the recalled pattern. Grossberg [Gro87a, Gro87b], Mikhailov [MP91] and others have investigated the use of limit cycles to represent recalled time series. Networks of phase or frequency correlated oscillators have been studied as mechanisms to perform variable binding and pattern segmentation. Recently, attention has focused on trajectories that are governed by chaotic attractor dynamics [Fre94, Kan90, Tsu94]. These models are inspired by the discovery of chaotic dynamics in olfactory cortex, and the existence of apparently chaotic attractors in EEG data. Chaotic attractors play the same role as the fixed points in more traditional models. However it is the spacial distribution of attractors among modular networks that corresponds to the retrieved memory. Other models, such as the Boltzmann machine [AHS85], are of a statistical nature and can be described as stochastic dynamical systems. These models are often assumed to have no external inputs or only constant external inputs, and very little attention is paid to the role that external data can play in determining their run time computation.

In all of these cases, the run time computation performed by a connectionist model is an interpretation of the network dynamics. We can view a connectionist system as an analog computer with a distributed central processor (the neurons), weights that act as long term memory and stored programs, and an activation state that is the short term working memory of the machine. Very little research effort has been devoted to producing versatile, structured run time dynamics in connectionist systems. Only one or two simple types of dynamics are used today. Most research effort to date has gone into weight adaptation algorithms instead. Consequently, although theory tells us that connectionist computers are in principle computationally equivalent to universal Turing machines, in practice, they are only capable of a few characteristic types of computations. They are not yet flexible enough to manipulate high-level, abstract knowledge, and therefore are significantly weakened as models of human reasoning, and as practical tools for AI.

A.3.3 Learning

Any computation that the network performs is determined by the dynamical trajectory that the network takes through the underlying state space. Since the dynamical equations of the neurons are parameterized by the weights (and thresholds), the dynamics of the network, and therefore its computational behaviour, are controlled by these values. In order to change the computation performed by the network, we must modify the network weights in such a way that the trajectories correspond to the desired computation. Connectionist systems serve the dual purpose of being both the performance engine and the learning algorithm, but the emphasis is usually placed on learning. In this thesis we will place a greater emphasis on control of run time dynamics, but, as we shall see, this will have repercussions for training algorithms.

Connectionism is considered a branch of machine learning. To illustrate its relationship with other machine learning techniques, we will divide the machine

learning literature into three areas: *symbolic empirical learning*, *subsymbolic empirical learning*, and *constructive induction*.

Symbolic empirical learning algorithms analyze examples in order to create and modify symbolic descriptions whose structure is known *a priori*. The motivation for these algorithms stems from a wish to understand how this can be done when very little prior knowledge is available to assist in the analysis. The problem is typically treated as an abstract computational task and few attempts are made to relate it directly to human performance. The knowledge that is created by symbolic empirical learning algorithms must be in a form that is easy for humans to comprehend, so the knowledge representation language is typically restricted to primitive symbolic structures and operators. The inference methods used by these algorithms are falsity preserving, and therefore require that the hypotheses generated be verified by experiment.

Subsymbolic empirical learning algorithms differ from their symbolic counterparts in that a greater emphasis is placed on modeling natural cognitive mechanisms. This is done at the expense of the readability of the acquired knowledge. Connectionist systems, which fit into this category, learn by modifying the coefficients on a parameterized knowledge structure—a simple model network of neurons. The primary motivation of this research was originally to acquire a better understanding of learning mechanisms employed by the brain. Heterogeneous systems employ a genetic algorithm to generate new structures as well as modify associated structure parameters. Such systems have been designed to account for important constraints on human learning identified by cognitive scientists.

Unlike empirical learners, symbolic constructive induction algorithms are knowledge intensive. The motivation for work in this area stems from the fact that humans are able to employ their considerable prior knowledge to create plausible hypotheses from only a few input examples. Whereas empirical learning employs very little prior knowledge, and typically requires a large number of examples to acquire useful hypotheses, constructive induction employs a large amount of prior knowledge

to create plausible hypotheses from very little input information. In addition, empirical learners cannot discover complex relationships and causal dependencies that require high level concepts not found in the inputs, because the hypotheses generated are expressed only in terms of the attributes of the input data or simple transformations on them. Constructive induction systems, on the other hand, can change the representation of the problem or invent new attributes or concepts by employing background knowledge not found in the input examples. These systems typically rely heavily on deductive inference procedures and analysis of the resulting inference chains to guide learning.

In order to perform connectionist constructive induction, we require a better understanding of how high-level symbolic knowledge can be represented and operated on in that setting. Also, more versatile control over the run-time dynamics of the network are needed to allow active knowledge to control the weight update process. The directions which we pursue are related to these two issues and may be suggestive of further work along this line.

For a comprehensive discussion of connectionist learning algorithms see the classic books of Rumelhart, et. al. [RM86a], [RM86b]. For a more current and mathematical treatment see the book of Golden [Gol96].

Appendix B

Differential Geometry Review

Linear algebra is the language of linear control theory. It describes the evolution of systems on the flat manifold $\mathbb{R}^n$. The essential difference between nonlinear control theory and the linear kind is that nonlinear control systems have dynamics that evolve over general differentiable manifolds which are non-Euclidean spaces. This forces us to abandon linear algebra in favour of a more general theory that is able to describe the evolution of systems on manifolds that are only locally¹ diffeomorphic to $\mathbb{R}^n$. This theory is indeed the theory of differential geometry. Needless to say, all the results from linear control theory carry over as special cases (of flat manifold dynamics) in the language of differential geometry.

In this section we introduce the reader to a short list of definitions and concepts that are essential to the sequel. A good overview of differential geometry can be found in the book by Boothby [Boo86], and short summaries relevant to control can be found in the books of Isidori [Isi89, Appendix A] and Nijmeijer and van der Schaft [NvdS90, Chapter 2].

¹A *local* property is one that holds in an open neighborhood of a point. Generally this neighborhood does not extend to the whole manifold that the point is an element of.

B.1 Functions

B.1.1 Smooth Functions

A *smooth* function is one that has partial derivatives of any order. The space of smooth functions is called C^∞ . Most of the results of interest in this thesis hold for smooth functions.

B.1.2 Analytic Functions

Analytic functions are a subset of the smooth functions for which a Taylor expansion exists at every point. The space of analytic functions is called C^ω . An example of a smooth function that is not analytic is

$$f(x) = \begin{cases} 0 & x \leq 0 \\ \frac{-1}{e^x} & x > 0 \end{cases}.$$

Since analytic functions are better behaved than general smooth functions, there are a greater number of control theoretic results that hold for them. We will indicate when our results only hold for analytic functions.

B.2 The Jacobian

B.2.1 Inverse Function Theorem

A diffeomorphism, $f(\mathbf{x})$, is a mapping that is a smooth, invertible mapping, i.e. f and f^{-1} are smooth, and bijective. As an aid to the intuition, think of a diffeomorphic mapping of a space onto itself as a smooth stretching of clay (the space) that does not tear the clay, or introduce nondifferentiable ‘kinks’ into the clay.

We now consider the question of when a mapping is a smooth reversible diffeomorphism. The *inverse function theorem* tells us that a mapping $f(\mathbf{x})$ is diffeomorphic when the Jacobian matrix, ∂f , exists and is nonsingular. Although this

property may not be generally true of the Jacobian at all points. it is the case that the mapping is *locally* diffeomorphic around a point $\mathbf{x}$ if the Jacobian $\partial \mathbf{f}(\mathbf{x})$ exists and is nonsingular at that point.

B.2.2 Rank Theorem

The *rank theorem* tells us that the nonlinear equivalent to the rank of a linear mapping $\mathbf{f}(\mathbf{x}) = \mathbf{M}\mathbf{x}$, is the rank of the Jacobian of the nonlinear function $\mathbf{f}(\mathbf{x})$.

In the linear case this Jacobian is just

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A}.$$

and so the rank is the rank of the matrix $\mathbf{A}$, as expected. It is a standard result of linear algebra that the rank is the dimension of the image space of $\mathbf{f}(\mathbf{x})$, a linear subspace of $\mathbb{R}^n$. For nonlinear $\mathbf{f}(\mathbf{x})$ we have the analogous situation: the rank of the Jacobian is the dimensionality of the general nonlinear manifold that is the image of $\mathbf{f}(\mathbf{x})$.

Since manifolds are locally equivalent to $\mathbb{R}^n$, we can find diffeomorphisms $\mathbf{h}$, $\mathbf{g}$, such that $\mathbf{h} \circ \mathbf{f} \circ \mathbf{g}^{-1}(x_1, \dots, x_N) = (x_1, \dots, x_K, 0, \dots, 0)$. The rank theorem therefore tells us that any mapping $\mathbf{f}(\mathbf{x})$ from an n -dimensional space to an m -dimensional space of rank k behaves locally like a projection of $\mathbb{R}^n$ to $\mathbb{R}^m$.

B.3 What is a Manifold?

A *manifold* is any set of mathematical objects that can be given a topology with a countable basis and the Hausdorff (T2) separability property, such that each object has a neighborhood homeomorphic to an open set of $\mathbb{R}^n$ is a manifold. The dimension of the manifold is locally the dimension n of $\mathbb{R}^n$.

The locally linear structure of the manifold allows us to find a covering set of *charts*, e.g. $\{\mathcal{U}, \mathbf{x} = \mathbf{f}(\mathbf{p})\}$, that locally map open regions of the manifold to $\mathbb{R}^n$.

The mapped values $\mathbf{x} = \mathbf{f}(\mathbf{p})$ are said to be in *local coordinates*. A covering set of charts is called an *atlas*.

A manifold is said to be *differentiable* if an atlas exists in which the change of coordinates between any two overlapping charts is always given by a differentiable function. i.e. $\mathbf{y} = \mathbf{f}_1 \circ \mathbf{f}_2^{-1}(\mathbf{x})$ is a diffeomorphism.

Functions can be defined on a manifold. They map the points of the manifold to real or vector values. Note that we cannot do traditional calculus on such functions in general, since the manifold is not necessarily a vector space. Manifold functions are smooth (analytic) if their image under any chart is smooth (analytic).

B.4 Tangent Spaces

B.4.1 The Problem With Calculus on General Manifolds

Since the dynamics of nonlinear plants evolve on general manifolds we need to define differentiation on manifolds in order to describe the evolution in terms of familiar ODEs. Traditional differentiation and integration cannot be applied to functions defined over manifolds because a canonical basis can rarely be defined that gives the manifold a vector space structure. For example, we can find a basis for $\mathbb{R}^n$ but not for a sphere immersed in $\mathbb{R}^n$.

B.4.2 The Solution to the Problem

We attach an independent vector space, called a *tangent space*, to each and every point of the manifold. All these tangent spaces have canonical bases that are independent of the atlas. They are constructed so that an infinitesimal movement, from point $\mathbf{x}_1$ to $\mathbf{x}_2$, along a curve on the manifold produces a corresponding infinitesimal change in a vector represented within the tangent space for $\mathbf{x}_1$ and the tangent space for $\mathbf{x}_2$. This *coordinate free* connection between movement in the manifold and changes in the respective tangent spaces allows us to do differentiation on

manifolds. A little extra structure lets us do integration as well.

B.5 Tangent Vectors

Each tangent vector, $\psi_{\mathbf{v}}^{\mathbf{p}}$, defined at a point $\mathbf{p} \in \mathcal{M}$, is a special kind of functional - a directional derivative along a vector direction $\mathbf{v}$ at the point $\mathbf{p}$, i.e. a mapping from the space of smooth functions to the reals $\psi_{\mathbf{v}}^{\mathbf{p}} : C^{\infty} \mapsto \mathbb{R}^1$. These functionals are linear, because, given functions f_1 and f_2 :

$$\psi_{\mathbf{v}}^{\mathbf{p}}(af_1) + \psi_{\mathbf{v}}^{\mathbf{p}}(bf_2) = \psi_{\mathbf{v}}^{\mathbf{p}}(af_1 + bf_2)$$

and they obey the Leibnitz property

$$\psi_{\mathbf{v}}^{\mathbf{p}}(f_1 f_2) = f_1 \psi_{\mathbf{v}}^{\mathbf{p}}(f_2) + f_2 \psi_{\mathbf{v}}^{\mathbf{p}}(f_1).$$

Consequently, the tangent vectors are derivations.

B.6 Tangent Space

The *tangent space* $T_{\mathbf{p}}\mathcal{M}$ is the space of tangent vectors (functional derivations) attached to manifold $\mathcal{M}$ at point $\mathbf{p}$. This space is given a linear vector space structure by

$$a\psi_{\mathbf{v}_1}^{\mathbf{p}}(f) + b\psi_{\mathbf{v}_2}^{\mathbf{p}}(f) = (a\psi_{\mathbf{v}_1}^{\mathbf{p}} + b\psi_{\mathbf{v}_2}^{\mathbf{p}})(f).$$

If $\mathcal{M}$ is a smooth m -dimensional hypersurface immersed in $\mathbb{R}^n$, then the tangent space can be identified with the tangent hyperplane at $\mathbf{p}$. The dimension of the tangent space at $\mathbf{p}$ is equal to the dimension of the manifold at $\mathbf{p}$.

B.6.1 The Basis Vector Fields of a Tangent Space

Since any tangent vector is a directional derivative along a unique vector direction $\mathbf{v}$, we can see that

$$\psi_{\mathbf{v}}(f) = \sum_{i=1}^n v_i \frac{\partial}{\partial x_i}(f) = \left\langle \frac{\partial}{\partial \mathbf{x}}, \mathbf{v} \right\rangle (f)$$

for all smooth f defined over the manifold $\mathcal{M}$. We identify the unique vector $\mathbf{v}$ with $\psi_{\mathbf{v}}$, and therefore the basis of the tangent space must be the functionals $\frac{\partial}{\partial x_i}$.

B.7 Numeric Calculations

Notice that the tangent space, tangent vector, and tangent space basis are all abstract, coordinate free ideas. They are useful for talking about the abstract structure of dynamical systems on a manifold without worrying about the intricate details associated with any particular atlas. However, to do any real numeric computations on manifolds we must give concrete (numeric) meaning to the basis vectors of the tangent space. The hurdle we must overcome is one of knowing how to take the partial derivative of the general manifold function f that defines the basis vectors.

To do this we will pick an arbitrary chart $\{\mathcal{U}, \mathbf{x} = \varphi(\mathbf{p})\}$ that includes $\mathbf{p}$, and use the map $f(\varphi^{-1}(\mathbf{x}))$ to locally represent f in $\mathbb{R}^n$. We can therefore take the partial derivative of this local representation of the function in $\mathbb{R}^n$ with respect to the $\mathbf{x}$ coordinates. This means that the coordinate free basis vector $\frac{\partial}{\partial x_i}$ actually stands for $\frac{\partial f(\varphi^{-1}(\mathbf{x}))}{\partial x_i}$, where x_i is in the local $\mathbb{R}^n$ coordinate system induced by the chart.

The numeric representation of a tangent vector $\psi_{\mathbf{v}}^{\mathbf{p}}$ is a real valued vector $\hat{\mathbf{v}}$. The elements of $\hat{\mathbf{v}}$ can be computed for a particular chart (that contains $\mathbf{p}$) by applying the associated tangent vector $\psi_{\mathbf{v}}^{\mathbf{p}}$ to the function elements of the chart map $\varphi(\mathbf{p}) = [x_1(\mathbf{p}) \cdots x_m(\mathbf{p})]^T$. Recall that $\psi_{\mathbf{v}}^{\mathbf{p}}$ is the linear combination of basis tangent

vectors. We have then that the j^{th} element of the tangent vector is

$$\psi_{\mathbf{v}}^{\mathbf{p}}(x_j) = \quad (\text{B.7.1})$$

$$= \sum_{i=1}^n \hat{v}_i \frac{\partial}{\partial x_i}(x_j) \quad (\text{B.7.2})$$

$$= \sum_{i=1}^n \hat{v}_i \frac{\partial x_j(\varphi^{-1}(\mathbf{x}))}{\partial x_i} \quad (\text{B.7.3})$$

$$= \sum_{i=1}^n \hat{v}_i \frac{\partial x_j}{\partial x_i} \quad (\text{B.7.4})$$

$$= \hat{v}_j \quad (\text{B.7.5})$$

because $x_j = x_j(\varphi^{-1}(\mathbf{x}))$. and

$$\frac{\partial x_j}{\partial x_i} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

Notice that these numeric values for $\psi_{\mathbf{v}}^{\mathbf{p}}$ are coordinate chart dependent, and so each chart defines its own numeric basis or coordinate system.

As we perform calculations for systems evolving on a manifold, the system trajectory will move across chart boundaries. We must recompute the numeric representation $\hat{\mathbf{v}}$ as it moves from chart to chart. Given chart 1 $\{\mathcal{U}, \mathbf{x} = \varphi(\mathbf{p})\}$ and chart 2 $\{\mathcal{V}, \mathbf{y} = \xi(\mathbf{p})\}$, then $\mathbf{x}$ coordinates in chart 1 can be converted to $\mathbf{y}$ coordinates in chart 2 with the map $\mathbf{y} = \xi(\varphi^{-1}(\mathbf{x}))$. If we substitute this into the above equations and follow through the straightforward manipulations, we discover that the Jacobian matrix of the local coordinate transformation evaluated at point $\mathbf{p}$, $\frac{\partial x_j(\varphi^{-1}(\mathbf{x}))}{\partial \mathbf{x}}$, forms a matrix that linearly transforms the numeric representation of a tangent vector in chart 1 coordinates into the equivalent representation in chart 2 coordinates, i.e.

$$\hat{\mathbf{v}}_{\mathbf{y}} = \frac{\partial x_j(\varphi^{-1}(\mathbf{x}))}{\partial \mathbf{x}} \hat{\mathbf{v}}_{\mathbf{x}}.$$

B.8 Manifold Mappings and Tangent Space Mappings

Given a diffeomorphic mapping $F : \mathcal{M} \mapsto \mathcal{N}$ of manifold $\mathcal{M}$ into manifold $\mathcal{N}$, the mapping of the tangent space of $\mathcal{M}$ at $\mathbf{p}$ into the tangent space of $\mathcal{N}$ at $\mathbf{F}(\mathbf{p})$ is given by the locally linear matrix mapping $T_{\mathbf{F}(\mathbf{p})}\mathcal{N} = [\partial F(\mathbf{p})] T_{\mathbf{p}}\mathcal{M}$. This is just a generalization of the change of basis transformation described above. The Jacobian matrix $\mathbf{F}_*(\mathbf{p}) = \partial F(\mathbf{p})$ is called the *differential* of the manifold mapping $\mathbf{F}$.

$\mathbf{F}$ is numerically expressed as a mapping between local coordinates, $\mathbf{y} = \mathbf{F}(\varphi^{-1}(\mathbf{x}))$, so that the Jacobian of this local version of $\mathbf{F}$ can be taken with respect to the local Cartesian coordinate systems. Thus,

$$\mathbf{F}_* = \frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \frac{\partial \mathbf{F}(\varphi^{-1}(\mathbf{x}))}{\partial \mathbf{x}}$$

Manifold mappings can be composed to map through more than one manifold, e.g. $\mathbf{H} = \mathbf{G} \circ \mathbf{F}$. The composition of the corresponding tangent space differentials is just the product of the individual differential Jacobian matrices, e.g. $\mathbf{H}_* = \mathbf{G}_* \mathbf{F}_*$.

B.8.1 Manifold Embeddings

When manifold $\mathcal{M}$ is mapped by a diffeomorphism $\mathbf{F}$ into a submanifold $\mathcal{S}$ of a manifold $\mathcal{N}$ of dimension greater than or equal to $\mathcal{M}$, we call this an *embedding*. In this case the rank of the matrix $\partial \mathbf{F}$ is equal to the dimension of $\mathcal{S}$ at all $\mathbf{p} \in \mathcal{M}$. $\mathbf{F}$ is injective, and the topology induced on $\mathcal{S} = \mathbf{F}(\mathcal{M})$ by the topology of $\mathcal{M}$ coincides with the topology of $\mathcal{S}$ as a subset of $\mathcal{N}$. The tangent space of $\mathcal{S}$ will be a linear subspace of the tangent space of $\mathcal{N}$. The Jacobian $\mathbf{F}_*$ will be nonsingular because $\mathbf{F}$ is a diffeomorphism, so the rank of $\mathbf{F}_*$ will be the rank of $\mathcal{M}$. As we vary $\mathbf{F}(\mathbf{p})$, the subspace $\mathbf{F}_* T_{\mathbf{p}}\mathcal{M}$ will swing around in the tangent space of $\mathcal{N}$.

B.9 Vector Fields

A *vector field* is a mapping from points of $\mathcal{M}$ to a single tangent vector of $T_{\mathbf{p}}\mathcal{M}$. A vector field $\mathbf{f}(\mathbf{p})$ is smooth (analytic) if for any chart the changing tangent vector coordinates $c_i(\mathbf{p})$ in $\mathbf{f}(\mathbf{p}) = \sum_{i=1}^n c_i(\mathbf{p}) \frac{\partial}{\partial x_i}$ are smooth (analytic) functions. The space of all smooth vector fields on $\mathcal{M}$ is $V(\mathcal{M})$.

Note that because there is always a smooth change of basis between charts, the $c_i(\mathbf{p})$ will always be smooth/analytic in any basis, and vector fields are coordinate free constructions that can be manipulated at the level of theory without regard to numeric values. A simulation of a control system will, however, require describing the vector fields with respect to a particular atlas.

B.9.1 Ordinary Differential Equations on Nonlinear Manifolds

Vector fields enable the definition of differential equations on a general manifold. An *integral curve* for a given vector field $\mathbf{f}(\mathbf{p})$ is a curve $c : (t_1, t_2) \mapsto \mathcal{M}$ such that the tangent space mapping between manifold (t_1, t_2) and $\mathcal{M}$ is the same as $\mathbf{f}$ at each t , i.e. $\mathbf{c}_* \left(\frac{d}{dt} \right) = \mathbf{f}(\mathbf{c}(t))$. The usual properties of integral curves from the theory of ODEs are preserved in the general manifold setting.

B.9.2 Lie Derivatives

The directional derivative of a manifold function $h(\mathbf{p})$ relative to the tangent vector given by the vector field $\mathbf{f}(\mathbf{p})$ is the *Lie derivative* $\mathcal{L}_{\mathbf{f}}h(p) = \mathbf{f}(\mathbf{p})h = \langle \partial h, \mathbf{f} \rangle$. If $\mathbf{f}$ and h are smooth then $\mathcal{L}_{\mathbf{f}}h$ is also a smooth function. These derivatives appear in many contexts in nonlinear control theory.

B.9.3 Lie Brackets

The *Lie bracket*

$$[\mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x})] = \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) - \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{g}(\mathbf{x})$$

is a type of vector field product that produces another vector field. This product obeys the Lie properties of

1. skew commutivity: $[\mathbf{f}, \mathbf{g}] = -[\mathbf{g}, \mathbf{f}]$.
2. bilinearity: $[a\mathbf{f} + b\mathbf{g}, \mathbf{h}] = a[\mathbf{f}, \mathbf{h}] + b[\mathbf{g}, \mathbf{h}]$, and
3. the Jacobi identity: $[\mathbf{f}, [\mathbf{g}, \mathbf{h}]] + [\mathbf{g}, [\mathbf{h}, \mathbf{f}]] + [\mathbf{h}, [\mathbf{f}, \mathbf{g}]] = 0$.

If two vector fields $\mathbf{f}$ and $\mathbf{g}$ are elements of the tangent space of an embedded submanifold, then the bracket $[\mathbf{f}, \mathbf{g}]$ is an element of the submanifold's tangent space too.

The bracket is 0 only when the integral curves of $\mathbf{f}$ and $\mathbf{g}$ form the diffeomorphic image of a Cartesian coordinate system. Intuitively, a non-zero bracket is a vector field which is proportional to the derivative of the vector difference between the point $p = F \text{vect } f, Fg(e, p)$ and $p_2 = F$ at each point $\mathbf{x}$.

The space of all smooth vector fields $V(\mathcal{M})$ can be given a vector space structure since $a\mathbf{f}(\mathbf{x}) + b\mathbf{g}(\mathbf{x}) = (a\mathbf{f} + b\mathbf{g})(\mathbf{x})$. It can also be given the structure of a Lie algebra by adding the Lie bracket. This extra algebraic structure is useful because, as we shall see below, important control system distributions are derived from certain Lie algebras.

Lie brackets are of importance in control theory mainly because the directions in which we can steer the system state away from the drift dynamics include the closure of all the Lie brackets of the input vector fields $\mathbf{g}_i(\mathbf{x})$.

B.10 Distributions

A *distribution* is a generalization of a vector field². It maps points $\mathbf{p}$ of a manifold $\mathcal{M}$ to *subspaces* of the tangent space $T_{\mathbf{p}}\mathcal{M}$ at each $\mathbf{p}$. Notions of smooth/analytic distributions are defined as for vector fields. Like vector fields,

²This is not to be confused with the notion of a probabilistic or statistical distribution.

distributions are also coordinate free constructions. A *nonsingular distribution* is one for which the subspaces at all $\mathbf{p}$ are of the same dimension d . In practice, we often define a distribution Δ to be the span of a set of vector fields, i.e. $\Delta(\mathbf{p}) = \text{span}\{\mathbf{v}_1(\mathbf{p}) \dots \mathbf{v}_d(\mathbf{p})\}$.

The restriction of a distribution $\Delta(\mathbf{p})$ to the submanifold $\mathcal{S}$ is denoted $\Delta(\mathbf{p})|_{\mathcal{S}}$, and is defined for all $\mathbf{p} \in \mathcal{S}$. There are many different distributions that have restrictions equal to the tangent space of a given embedded submanifold $\mathcal{S}$.

A *smooth distribution* is spanned by smooth vector fields. Even though the sum of smooth distributions is smooth, the intersection of smooth distributions might not be smooth.

Distributions are especially useful for describing the dynamics of nonlinear control systems. As discussed in Chapters 4 and 5, special distributions exist which place bounds on the degree to which a system can be controlled. Much of control theory is involved with finding or constructing various distributions that describe the tangent spaces of state space submanifolds important to a given application. In Chapter 6, we consider the problem of learning distributions that represent computational constraints on the run-time behaviour of continuous time neural networks.

B.10.1 Involutive Distributions

A distribution is *involutive* if it is closed under the lie bracket, i.e. $\forall \mathbf{v}_1, \mathbf{v}_2 \in \Delta$, $[\mathbf{v}_1, \mathbf{v}_2] \in \Delta$. A nonsingular distribution $\Delta = \text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_d\}$ is involutive iff $[\mathbf{v}_i, \mathbf{v}_j] \in \Delta$, $1 \leq i, j \leq d$. This can greatly simplify computations for nonsingular distributions. The *involutive closure* $\text{inv}(\Delta)$ of a distribution Δ is the smallest involutive distribution containing Δ . Although the intersection of involutive distributions is again involutive, the sum of involutive distributions might not be involutive.

We shall see below that the condition of involutivity is required for guaranteeing that a nonsingular distribution has the desirable characteristic of complete integrability.

B.11 Completely Integrable Distributions and Frobenius' Theorem

A special class of nonsingular distributions, called the *completely integrable* distributions, turns out to be particularly well behaved, and therefore extremely important to nonlinear control. A d -dimensional, completely integrable distribution Δ is one whose annihilator $\Delta^\perp$ is spanned by the gradients of $n - d$ real valued smooth functions, i.e. $\Delta^\perp(\mathbf{p}) = \text{span} \{d\lambda_1(\mathbf{p}) \dots d\lambda_{n-d}(\mathbf{p})\}$ for scalar functions $\lambda_k(\mathbf{p})$, $1 \leq k \leq (n - d)$. The reason that this odd sounding condition is so important is that it guarantees that Δ locally partitions the statespace into a foliation with d -dimensional submanifolds as leaves. These submanifolds all have local tangent spaces equal to $\Delta(\mathbf{p})$. Furthermore, we can find a nonlinear change of coordinates $\mathbf{z} = \Phi(\mathbf{p})$ in which these submanifolds are locally 'flattened', and a differential mapping $\partial\Phi$ in which $\hat{\Delta} = \partial\Phi\Delta$ is spanned by d of the vectors $\left\{ \frac{\partial}{\partial p_1}, \dots, \frac{\partial}{\partial p_n} \right\}$. As we shall see, $\hat{\Delta}$ has a greatly simplified form in these coordinates.

Considering the importance of completely integrable distributions to the analysis of nonlinear control systems, it would be nice if there were a relatively simple, necessary and sufficient condition that would guarantee complete integrability for a nonsingular distribution. In fact there is. A theorem by Frobenius tells us that a nonsingular distribution is completely integrable iff it is involutive. For a proof see Chapter 1 of [Isi89].

B.11.1 Invariant Distributions

A distribution Δ is said to be *invariant under vector field* $\mathbf{f}$ if it is closed under the Lie bracket product with $\mathbf{f}$, i.e. ,

$$[\mathbf{f}, \mathbf{v}] \in \Delta, \quad \forall \mathbf{v} \in \Delta.$$

The intuitive meaning of invariance is that, if the d -dimensional distribution Δ is completely integrable, then the vector field $\mathbf{f}(\mathbf{p})$ will be tangent to one of

the manifolds (leaves) $\mathcal{M}$ of the constant dimensional foliation induced by Δ (see Section B.11). Another way of saying this is that at every point in this manifold the vector field is an element of Δ , i.e.

$$\forall \mathbf{p} \in \mathcal{M}. \quad \mathbf{f}(\mathbf{p}) \in \Delta(\mathbf{p}).$$

Yet another way to view this result is to say that the Section B.11 coordinate transformation $\mathbf{z} = \Phi(p)$ that locally flattens the leaf $\mathcal{M}$ allows us to express the vector field $\mathbf{f}$ in the simplified form

$$\mathbf{f}(\mathbf{z}) = \mathbf{f}(\Phi^{-1}(\mathbf{z})) = \begin{bmatrix} f_1(z_1, \dots, z_d, z_{d+1}, \dots, z_n) \\ \vdots \\ f_d(z_1, \dots, z_d, z_{d+1}, \dots, z_n) \\ f_{d+1}(z_{d+1}, \dots, z_n) \\ \vdots \\ f_n(z_{d+1}, \dots, z_n) \end{bmatrix}.$$

From a practical standpoint, invariance means that the flow $\Phi_t^{\mathbf{f}} \mathbf{x}_0(t_0)$ of vector field $\mathbf{f}$ is confined to the leaf $\mathcal{M}$ for all times $t > t_0$ if it begins on the leaf (i.e. $\mathbf{x}_0 \in \mathcal{M}$). This property of ‘confinement’ to a manifold will be of importance in Chapters 6. and 6.

B.12 Covector Fields and Codistributions

The idea of the *dual* of a vector space is often very useful in real analysis, and it also turns out to be useful when working in the differential geometric setting, mainly to make calculations easier. The dual space V^* of an n -dimensional vector space V is the set of all linear real valued functions defined on domain V . The linearity of these functions makes it easy to give V^* the structure of an n -dimensional vector space. Elements of V^* are called *covectors*. Since each covector $\omega^* \in V^*$ is a linear mapping from an n -dimensional column vector $\mathbf{v} \in V$ to a real number,

it is appealing to represent it as a matrix with one row, i.e. $\omega^* = [\omega_1 \ \cdots \ \omega_n]$. The value of the covector ω^* at an element of V is found by performing the matrix multiplication. However, since ω^* is a matrix with only a single row, it has become the convention to represent this transformation as the inner product $\langle \omega^*, \mathbf{v} \rangle$.

We can now define the dual of a vector field and a distribution. A *covector field* is a smooth mapping that assigns to each point $\mathbf{p}$ an element of the dual of the tangent space $T_{\mathbf{p}}\mathcal{M}^*$. We can think of a covector field as a row vector of smooth real valued functions, $\omega^*(\mathbf{p}) = [\omega_1(\mathbf{p}) \ \cdots \ \omega_n(\mathbf{p})]$. Notice that some covector fields (but not all) correspond to the gradient of a function $\lambda(p)$, that is

$$\omega^*(\mathbf{p}) = \partial\lambda(\mathbf{x}) = \frac{\partial\lambda}{\partial\mathbf{x}} = \left[d\frac{\partial\lambda}{\partial p_1}(\mathbf{p}) \ \cdots \ d\frac{\partial\lambda}{\partial p_n}(\mathbf{p}) \right]$$

A *codistribution* Ω is the span of a set of covectors. That is, $\Omega(\mathbf{p})$ assigns the subspace of $T_{\mathbf{p}}\mathcal{M}^*$ that is spanned by the set of covectors $\{\omega^*_1(\mathbf{p}) \dots \omega^*_d(\mathbf{p})\}$ to each point $\mathbf{p}$ of the state space. A particularly useful codistribution is the *annihilator* $\Delta^\perp$ of a distribution Δ , which is the set of covectors which annihilates all the vectors in Δ .

$$\Delta^\perp(\mathbf{p}) = \{\omega^*(\mathbf{p}) \in T_{\mathbf{p}}\mathcal{M}^* : \langle \omega^*(\mathbf{p}), \mathbf{v}(\mathbf{p}) \rangle = 0, \forall \mathbf{v} \in \Delta\}.$$

Appendix C

VISICON—A Simulation Library for the Visualization of Controlled Neurocomputers

Although most of the results described in this thesis are theoretical in nature, we believe that there is also much to be gained from a realization of the systems that we have studied—and the theoretical results that we have obtained—in the form of computer simulations. In addition to providing a way of empirically demonstrating theoretical results, the very act of designing and implementing such a system forces one to re-think the underlying technical issues and proofs from a pragmatic viewpoint. Invariably this complements one's understanding of the issues at stake, and sometimes contributes valuable insights. Furthermore, the task of implementing the underlying mathematical constructs highlights those aspects of the theory that, while sound on paper, are problematic in a practical sense. A concrete implementation can also provide insight into the real meaning of the abstract objects that we work with. Strong 'physical' intuition for these objects can be invaluable to the theoretician when attempting to visualize the effect of feedback on a network in

computational terms. Last, but not least, empirical confirmation of the theory gives one confidence in the results.

In this chapter we describe VISICON (VIsual SIMulator for COntrolled Neurocomputers), a research tool that facilitates the rapid design and implementation of simulated continuous time controlled neurocomputers. VISICON provides a reusable object-oriented class library that models the important mathematical objects that appear in the theory (including those introduced in this thesis).

Although we have not had time to fully pursue empirical simulations of the theoretical results in this thesis, VISICON represents a significant step in this direction. It is a simulation framework which provides many of the tools that are needed to conduct an empirical line of research.

The emphasis in VISICON is on providing the domain expert with a powerful programmatic toolset rather than on providing a user interface that hides the details of the toolset from a naive user.

We begin with a description of the design philosophy employed during the creation of VISICON. We follow this with a description of the main objects modeled by the class abstraction, discussing their most important behaviours, key representational structures and important numerical issues. We conclude with an example that illustrates the use of the VISICON system to create and execute a experimental simulation.

C.1 Motivation

C.2 Design Philosophy

C.2.1 VISICON is Object Oriented

Flexibility is the most important attribute of any simulation tool that is to be used in a research setting. The system must support rapid prototyping so that alternative ideas may be quickly simulated and compared. Changes in viewpoint

are common during research, and can easily impact the design of the objects that must be simulated. Therefore, it is important that modifications be accommodated with minimal impact on existing code. Furthermore, introducing new types of simulated objects, and extending existing ones in an upwards compatible way, should be rendered relatively straight forward.

It is for these reasons that VISICON has been modeled and implemented in an *object-oriented* style. Object oriented design methodologies directly model concepts that domain specialists use when describing events of interest in the domain. Each concept is represented by a type, or *class*, of object. Particular objects that instantiate a class, called *instances*, are created in a running program to simulate corresponding domain objects. Just as the domain objects have behaviours of interest to the researcher, even so do the instances of the corresponding classes possess associated computational behaviours called *methods*. A particular instance method is a set of expressions that are evaluated by sending a *message* to the instance. An instance that receives a message is referred to as the *receiver*. An extended computation is performed by constructing a community of instances that communicate with one another by exchanging messages, thereby triggering an appropriate sequence of computations. This community is a *direct* representation of the research domain.

This is a considerable departure from the traditional *procedural* approach to programming, where the emphasis is on finding a *functional* decomposition of the domain through stepwise refinement, and on representing the domain with abstract *data structures* operated on by a library of functions. We have adopted the object-oriented outlook because of the high degree of flexibility that it offers over the traditional approach - an important feature in a research setting, as we have already seen.

C.2.1.1 Flexibility

There are several object-oriented mechanisms that promote flexibility. First, an object-oriented model preserves domain concepts in the running program. This type of conceptual modularity makes it much easier to identify parts of the system that need to be changed when the researcher changes his conceptual viewpoint. Removing and adding concepts is, to a large extent, like pulling out or plugging in hardware modules. This is in contrast to procedural programs in which domain concepts are translated into abstract data types. Much of the immediacy of the domain is lost in the translation, forcing the researcher to repeatedly shift between the description of the domain and its abstract representation in the machine.

From this perspective, the VISICON library of classes directly reflects the objects that appear in the theorems of this thesis. Submanifolds, distributions, vector fields, and various kinds of neural networks are available. In addition, familiar mathematical objects such as vectors and matrices have been provided, along with a variety of ‘support’ objects.

C.2.1.2 Message Passing and Encapsulation

Another benefit of the object-oriented approach arises from the idea of computation by message passing. The internal representation of an object is ‘hidden’ from its fellows and can only be affected through a message send. Accessing the internals of another object directly is strongly discouraged. Instead, objects *encapsulate* their representations. Encapsulation reduces the degree of interdependence of one object on the internal details of another, thereby facilitating significant changes to the way that we think about and encode an object. This gives the researcher considerable freedom to explore alternative viewpoints.

VISICON classes respect principles of encapsulation, with the exception of a few very special violations that are permitted to enhance the efficiency of the code.

C.2.1.3 Inheritance

Class inheritance is another object-oriented mechanism that allows new classes to be defined as *extensions* of existing ones. In VISICON this produces a tree of classes—called the *inheritance tree*, or *class abstraction*—related by the ‘IS-A’ relationship¹. This tree echoes the conceptual taxonomy that is found in most domains, including that of neural computation and control theory. A new class refines (i.e. specializes) its parent class². It *inherits* the behaviours (and internal representation) of its parent, so its instances are able to respond to the same set of messages as those of the parent class without additional coding on the user’s part. Only *new* behaviours introduced by the new subclass need to be coded. This type of programming by extension allows the user to prototype new concepts rapidly, thereby encouraging a ‘generate and test’ approach to development that is more in keeping with a research setting than more traditional procedural language development.

C.2.2 VISICON Strives for Numerical Accuracy

Although the systems that we study in this thesis are inherently *continuous time* dynamical systems, described by systems of ODEs, on a digital computer we must numerically simulate these continuous time behaviours with *discrete time approximations*. This introduces the issue of maintaining the numerical accuracy and stability of VISICON simulation behaviours.

In VISICON, our goal has been to strive for a high degree of stability and accuracy in numeric computations. For example, as we see in Section C.3.1, VISICON will, by default, represent elements from scalar fields with as much accuracy as the underlying language allows, thereby minimizing round-off errors. Also, as detailed in Sections C.3.3.6 and C.3.5.1, VISICON employs numerical analysis techniques, such

¹Some object-oriented languages also support **multiple** inheritance, in which the classes are arranged in a directed network rather than a tree. However, this type of class abstraction increases the difficulty of maintenance, and so the VISICON library strictly employs a *single* inheritance tree.

²The parent class is called the *superclass*, and classes below the superclass are its *subclasses*.

as the filtering of singular values, to improve the accuracy of calculations involving ill-conditioned matrices. Both of these methods require a greater expenditure of memory and time.

This trade off between speed and space for greater accuracy is part of the VISICON philosophy. However, the user has been given control over this trade off, because, as we shall see below, it is possible that this philosophy leads to simulations that are too slow and memory intensive. VISICON is written in a language and style that promotes speed wherever this does not directly conflict with the goals of accuracy and stability. In addition, encapsulation is occasionally sacrificed to improve performance. This is done, however, in only a few strategically selected cases, and always in a controlled and well documented manner.

In the end, the greatest determinant of accuracy is the choice of multidimensional numerical integration routine that approximates the continuous time dynamics of the system under study. Unfortunately, the history of numerical analysis makes it clear that there is no such thing as a 'most accurate' numerical integration algorithm [Pre92]. Different problems require different integration techniques, and it is impossible to predict which one will be best for a given job. Therefore, VISICON models different types of integrators as objects, and allows the user to easily substitute instances of these different classes in order to experimentally find the one that is best for a given simulation. Currently, VISICON supports non-adaptive Runge-Kutta, adaptive 4th order Runge-Kutta and it is well positioned to also support adaptive Runge-Kutta and Richardson extrapolation. Extension of the VISICON integrator classes to other techniques is greatly simplified through inheritance-based programming. We remind the reader that a higher-order integration technique does not necessarily mean higher accuracy.

C.2.3 VISICON is Portable

Most of the VISICON class abstraction, including the numerical integration package, is written in C++ for speed and platform independence. C++ is currently the most widely used of the available, compiled object-oriented languages. All of the VISICON class library is written in C++.

In the sequel we will provide examples of C++ code that illustrate the use of VISICON classes. In these examples, sending a message called `mess`, with argument `arg`, to an object in variable `obj1`, is expressed as follows:

```
obj1.mess(arg);
```

The reader familiar with Pascal, C or C++ will have no trouble reading these examples. Readers with no experience in programming are referred to the excellent introductory C++ texts [Poh93, Str91].

An important role of VISICON is that of *visualizing* the differential geometric objects underlying computation in neural networks. The plotting tools that are used to render these objects are determined by those available on the host machine. However, to achieve portability, VISICON provides its own plotting classes that encapsulate the differences in these tools, and that support a common plotting protocol. Instances of these classes may be substituted for one another, and, therefore, VISICON is independent of the particular rendering tool used. Section C.3.9 describes the UNIX rendering utilities currently supported by VISICON. To support additional types of renderers, one need only subclass off of the classes described there.

C.2.4 Error Handling in VISICON

VISICON can be compiled with, or without, extensive run-time error checking. These tests are performed by various VISICON objects to make sure that values are in expected ranges, results of calculations are sensible, and that integration is stable. Simulations should be performed with these tests turned on. They should

only be turned off if the need for additional speed out-weighs the need for confidence in the results. The user can turn error checking off by setting the named VISICON constant `VISICON_DEBUG_ON` to false in file `visitype.h`.

Error messages are of the form

```
*** VISICON error 12::  
    Class Vector<T>: index 5 is outside legal bounds [0, 3]
```

and are written to the standard error log on the host machine (usually the user's terminal).

C.3 The VISICON Class Abstraction

In the remainder of this chapter, we shall describe most of the important VISICON classes. For each class, we shall sketch the internal instance representation and describe the most important public behaviours. Source code examples are provided to enhance the description when necessary.

As we have already seen, VISICON classes are written in C++. The overall organization of the C++ source files is as follows:

1. File `visitype.h` contains named constants, data-type definitions and error handling utilities.
2. File `visiutil.h` contains class and template definitions for several of the utility classes described in Section C.3.10.
3. File `visialg.h` contains template definitions for the vector and matrix classes described in Sections C.3.2 and C.3.3, and the basic vector field classes covered in Section C.3.4, excluding the encapsulating fields.
4. File `visidiff.h` contains class definitions for the differential geometric distribution and manifold classes described in Sections C.3.5 and C.3.6. It also

contains class definitions for the distribution encapsulating vector fields described in Section C.3.4.3.

5. File `visinet.h` contains class definitions for all of the neural network classes described in Section C.3.8. and the network encapsulating vector fields described in Section C.3.4.4.
6. File `visiint.h` contains class definitions for the numerical integration objects described below.
7. File `visiplot.h` contains class definitions for the UNIX plotting utility classes referred to in Section C.3.9.

In addition, each of the above classes has a companion class, `*.cc`, containing method source code.

C.3.1 Scalar Data Types

C++ is usually classified as a *hybrid* programming language because it employs both traditional data types and objects to represent data. For the purposes of the VISICON system, scalars are represented by C++ data types, and all other important entities are represented as objects.

C++ offers several datatypes for the representation of scalars. Types **Byte**, **Int**, **Unsigned_Int**, **Long** and **Unsigned_Long** are used to represent elements of the field $\mathcal{Z}$, and types **Float** and **Double** are used to represent elements of the field $\mathcal{R}$. Types that represent elements of the same field employ different numbers of bits to represent the element, and, because more bits permit a greater accuracy of representation, we are presented with a trade off between storage efficiency and numerical accuracy. An ideal choice of a scalar type for VISICON would minimize the number of bits such that the minimal necessary numerical accuracy is maintained. In practice, however, the optimal choice of scalar type varies with the computational

task, particularly when performing numerical integration. Therefore, it is important to give the VISICON user control over the scalar types that are used.

In order to provide the user with flexibility in his choice of scalars, VISICON permits its main library classes to be defined over any of the scalar types. This choice is accomplished through the use of C++ class templates. A class template defines a family of classes that is parameterized by one or more data types used in the definition of the class. Several VISICON classes described below are parameterized by a scalar type name so that, when an instance of the class is created, the user is able to specify the type of scalars to be stored by that particular instantiation of the class.

In addition to class templates, two default types are defined and used throughout the code for the predefined VISICON simulation behaviours. These types, **VISICON_Integer** and **VISICON_Real**, are defined as synonyms for the scalar types **Long** and **Double** respectively. In other words, by default the simulation behaviours use the most accurate representations for scalars. These types can be changed by the user if desired, in file **visitype.h**, and the VISICON system can be recompiled to use them.

The user may also define new types of scalars as objects rather than C++ data types. For example, the user might wish to define the class **Complex** to represent complex numbers. Scalar classes must support all C++ arithmetic operators in order to be compatible with the VISICON class library. In order to simplify the task of defining new scalar classes, the user may refine the VISICON class **Scalar**. The advantage of refining this class is that the user need only add code for the four arithmetic operators **+**, **-**, ***** and **/**. All other arithmetic operations are automatically inherited from **Scalar**.

C.3.2 Class `Vector<Scalar_Type>`

The neural networks that are simulated by VISICON are defined over multidimensional input, output and state spaces. Consequently, we require a class for representing elements from linear vector spaces defined over various scalar fields. The template `Vector` provides just such a family of classes parameterized by the type `Scalar_Type`. For example, a vector $v \in \mathcal{R}^n$ is represented by an instance of the class `Vector(VISICON_Real)`. The value of n is determined when the vector is instantiated.

C.3.2.1 Instance Representation

Vector instances store an `Integer` in private variable `dimension` that represents the number of elements in the vector. The message `dimension()` returns this value. The actual elements in an instance of `Vector<Scalar_Type>` are stored as a contiguous C++ array of `Scalar_Types` addressed by private pointer `elems`. Section C.3.2.3 describes how these elements may be accessed.

C.3.2.2 Instance Creation

`Vector` instances can be created in one of two ways. Typically, a vector is created by specifying its integer dimensionality. In the interests of speed efficiency, such a vector initially contains uninitialized elements. It can be set to the zero vector with the `zero()` message. Alternatively, an instance can be created by *copying* another instance. The following example illustrates the creation of a double precision zero vector from $\mathcal{R}^{100}$, and the creation of a copy of this vector:

```
% double precision zero vector
Vector<Double> v1(100);
v1.zero();

% single precision copy of v1
Vector<Float> v2(v1);
```


C.3.2.3 Element Access and Assignment

Vector elements are accessed and assigned using the familiar square bracket notation for array types in most programming languages. Assignment to an array element makes use of the standard C++ assignment operator. For example, we can assign the fourth element of `v1` to the third element of `v2` as follows:

```
v2[3] = v1[4];
```

In the event that the elements of the two vectors involved in an assignment are of different scalar types, standard C++ coercion rules apply.

C.3.2.4 Vector Equality

There are two views that one may take regarding the equality of vectors. The naive view is to demand that two vectors have identical elements in order to be considered equal. Indeed there are situations in which this test for *exact equality* is appropriate, and, therefore, the standard C++ operators for equality and inequality (`==`, and `!=` respectively) have been overloaded to perform this type of comparison between instances of the class **Vector**.

There is another, perhaps more realistic, view of equality between vectors. As we have already seen, behaviours which involve numerical computations inevitably introduce a degree of round off error into the elements of their vector results. The expected tolerance is represented by the named VISICON constant `VISICON_ROUNDOFF_ERROR`. Taking this expected error into consideration, we may define a test for the *numerical equality* of two vector results, in which they are considered equal if the absolute values of their element-wise differences are all less than or equal to the error tolerance `VISICON_ROUNDOFF_ERROR`. In keeping with this observation, a receiving vector and argument vector `v`, can be compared with the messages `numerically_equal(v)`, and `not_numerically_equal(v)`.

The result of these operators and messages is of type **Boolean** defined in

file `visitype.h`. Note that both types of equality are defined only between vectors of the same dimension. A runtime error will occur otherwise.

C.3.2.5 Arithmetic Operations

The VISICON vector classes have been designed specifically for use by the VISICON simulation utilities so that their computational efficiency is maximized. Like the other algebraic classes described below, class **Vector** overloads many of the standard C++ arithmetic operators. Consequently, the user may compose arithmetic expressions involving vectors and other algebraic classes and scalar data types in a natural syntactic style. For example, the expression

```
v3 = v1 + v2 * c;
```

creates a **Vector** instance **v3** that is the element-wise sum of the **Vector** instance **v1**, and the product of scalar **c** and **Vector** instance **v2**. Notice that the standard C++ rules of algebraic operator precedence apply, as expected, in these types of expressions.

While this style of expression has the advantage of ease of composition and readability, it suffers from an unfortunate speed inefficiency. In general, overloaded C++ operators have the undesirable characteristic that their computed result must be *copied* into an object constructed by the compiler³. When an expression involves vectors of high dimensionality, the time spent copying elements can easily dominate the computation. Consequently, for each overloaded arithmetic operator, VISICON provides the user with a companion behaviour that permits the user to supply, by reference, an object that the intermediate result is to be stored in. In this way the user may manage the holders of intermediate results himself, thereby avoiding unnecessary copy operations.

Arithmetic operations between scalars and vectors include multiplication and division. Operator arguments may be applied in any order. These operators

³This gives the compiler control over garbage collection of these intermediate results.

yield vectors of the same dimension as the argument vector.

Unary operations on vectors include normalizing the receiving vector, and computation of the magnitude of the vector. The normalized vector has elements of the same type as the receiving vector. The magnitude is returned as scalar type **VISICON_Real**.

Binary operations between vectors include addition, subtraction, entrywise Hadamard product, entrywise Hadamard quotient, dot product and, in the case of 3-dimensional vectors only, the cross product. Each of these operations is defined only on vectors of the same dimensionality. In the case of the dot product the scalar result is of the same type genericity as the vector elements. The cross product produces a three dimensional unit vector normal to the arguments, with elements of the same type as the receiving vector. All other operations produce vectors of the same size and element type as the receiving vector.

For the convenience of the user the typical C++ operations of +=, -=, etc. are also supported.

The following example source code demonstrates the use of the arithmetic vector operations:

```
% define four vectors and two scalars.
Vector<VISICON_Real> v1(100), v2(100), v3(100), v4(100);
VISICON_Real c1 = 5.6, c2 = -7.8;

. . . vector elements are initialized . . .

% overloaded binary operations
v1 = c2 * v2;           % scalar product
v2 = v2 / c1;           % scalar quotient
v4 = v1 + v2 - v3;      % vector sum and difference
v1 = v4 * v2 / v1;      % entrywise Hadamard product
                        % and quotient

% functionally equivalent code using companion messages
% intermediate results stored in vector v5
Vector<VISICON_Real> v5(100);
```

```

v2.prod(c2, v1);           % scalar product
v2.quo(c1, v2);           % scalar quotient
v1.sum(v2, v5);           % vector sum . . .
v5.diff(v3, v1);          % . . . and difference
v4.hadamard_prod(v2, v5);  % entrywise Hadamard product . . .
v5.hadamard_quo(v1, v1);   % . . . and quotient

% additional operations
c1 = v1.dot(v2);           % dot product of v1 and v2
v1.cross(v2, v3);          % cross product of v1 and v2 => v3
c2 = v1.magnitude();       % magnitude of v1
v1.normalize();            % unit normalization of v1

```

C.3.2.6 The Undefined Vector

As we shall see below, VISICON models various objects that perform mappings between subsets of linear spaces and sets of vector instances. These objects map regions outside of the domain of definition to a special ‘undefined’ vector which is the only instance of the class **Undefined_Vector**. The undefined vector responds to the message `nil_vector()` by returning the value `true`. All other VISICON vectors respond to `nil_vector()` by returning the value `false`. These behaviours enable objects to test whether they are operating within the domain of a mapping.

C.3.3 Class Matrix<Scalar_Type>

The next algebraic object we shall discuss is the matrix. Many control theoretic objects are ultimately defined in terms of *local* linear transformations that can be represented as a matrix. The synaptic weights that parameterize neural networks can also be usefully represented by a matrix of real valued scalars. It is not surprising then that matrices find utility in VISICON primarily as support for the hidden algebraic representation of other VISICON objects. They are not limited to this role however, and should be considered a generally reusable class of object.

The **Matrix** template provides a facility for representing matrices of any

size over scalar fields represented by a type **Scalar_Type**. For example, instances of the class **Matrix<VISICON_Real>** are elements of $\mathcal{R}^{m \times n}$, where m and n are determined by the user.

C.3.3.1 Instance Representation

Matrix instances store the number of rows and columns in private variables `num_rows` and `num_cols` respectively. These values may be obtained with the messages `row_size()` and `column_size()`. The elements of an instance of **Matrix<Scalar_Type>** are stored as a private array of pointers to **Scalar_Type** called `elems`. Each address in this array points to a row of matrix elements represented by a contiguous C++ array of **Scalar_Types**. Section C.3.3.3 describes how individual elements may be accessed.

C.3.3.2 Instance Creation

Creation of **Matrix** instances is analogous to the creation of vectors. The most common way to construct a matrix is to specify the number of rows and columns. The resulting matrix instance is uninitialized. Alternatively, a copy may be made of an existing matrix.

All of the elements of a matrix may be initialized to 0 with the message `zero()`, to an arbitrary scalar c with the message `fill(c)`, or to random integer or real values in the range $[lb, ub)$ with the messages `fill_random_integer(lb, ub)` or `fill_random_real(lb, ub)` respectively⁴. The diagonal elements of a matrix may be initialized to a scalar value c with the message `fill_diag(c)`, or to the elements of a vector $\mathbf{v}$ with `fill_diag(v)`.

The following example illustrates the two methods of matrix creation:

```
% create a 5 row by 5 column integer matrix with elements
% initialized to integers between -10 and +10
```

⁴The message `fill_random_real` is not defined for matrices with element types representing the field ($\mathbb{Z}$).

```
Matrix<VISICON_Integer> m1(5, 5);
m1.fill_random_integer(-10, 10);

% create a real valued copy of the matrix m1 with diagonal elements
% set to 0.0
Matrix<VISICON_Real> m2(m1);           % create real valued copy
m2.fill_diag(0.0);                   % zero the diagonal
```

C.3.3.3 Element Access and Assignment

Elements of a matrix may be accessed by indexing the matrix with a pair of square brackets. The first pair specifies the row address, and the second pair specifies the column address of the desired element. An equivalent parenthetic syntax is also provided in which the indices are parenthesized and separated by a comma. Assignments are performed using the standard C++ assignment operator.

For example, the element at row 3, column 2 of matrix `m2` can be assigned to row 1, column 4 of matrix `m1` using either of the following expressions:

```
% square bracket syntax
m1[1][4] = m2[3][2];

% equivalent parenthetic syntax
m1(1, 4) = m2(3, 2);
```

The two forms of access syntax may be mixed in a single expression, but this is discouraged to promote readability of VISICON code.

As is the case for vectors, assignment to matrix elements will cause coercion of dissimilar element types according to the standard rules of C++. C++ casts may be performed to force an assignment of incompatible types, however this is discouraged in VISICON to promote maintainability of the code.

C.3.3.4 Matrix Equality

In keeping with the discussion of vector equality in Section C.3.2.4, we support two versions of equality for matrices. The overloaded C++ operators `==`,

and `!=` provide exact equality tests, and the messages `numerically_equal(m)`, and `not_numerically_equal(m)` allow tests for numerical equality between a receiving matrix and the argument matrix `m`. Return values are of VISICON type **Boolean**.

C.3.3.5 Arithmetic and Algebraic Operations

The **Matrix** class supports a complete set of binary arithmetic operations which operate on a receiving and argument matrix. Element-wise addition and subtraction, and element-wise Hadamard multiplication and division messages are supported between matrices of the same size, yielding another matrix of the same size and element type as the receiver. The more typical form of noncommutative matrix multiplication (i.e. composition of linear transformations) is defined between a receiving matrix of size $r \times s$ and an argument matrix of size $s \times t$, yielding a resultant matrix of size $r \times t$ with elements of the same type as the receiver.

Matrices of size $m \times n$ may be right multiplied by an argument vector of dimension n to produce a transformed vector instance of dimension m . The element type of this vector is the same as the receiving matrix. Messages are also provided to take the sum, difference, dot product and cross product of a matrix row or a matrix column with an appropriately sized vector. The vector results are of the same dimension as the argument vector and the same element type as the receiving matrix. The result of the dot product is a scalar of type **VISICON_Real**.

As is the case for vector arithmetic, the arithmetic sum, difference and noncommutative product are represented both by overloaded C++ operators as well as companion messages that allow the user to avoid unnecessary copy operations (as described in Section C.3.2.5).

It is often necessary to transpose a matrix, swapping the elements at row m , column n with the corresponding elements at row n , column m . A behaviour is provided to transpose the elements in a matrix instance destructively, as well as to create a transposed copy without modifying the instance. The destructive version is

only defined for square matrices.

The following source code illustrates the arithmetic and algebraic matrix behaviours:

```
% define seven matrices, three vectors, and a scalar
Matrix<VISICON_Real> m1(3, 25), m2(3, 25), m3(3, 25);
Matrix<VISICON_Real> m4(25, 15), m5(3, 15), m6(25, 3), m7(50, 50);
Vector<VISICON_Real> v1(25), v2(15), v3(3);
VISICON_Real c1;

. . . vector and matrix elements are initialized . . .

% overloaded binary operations and functional transpose
m1 = m1 + m2 - m3;          % matrix sum and difference
m5 = m1 * m4;               % noncommutative matrix multiplication
v1 = m1.transposed() * v3;  % multiplication of transpose by vector

% functionally equivalent code using companion messages
% intermediate results stored in
m1.sum(m2, m1);              % matrix sum . . .
m1.diff(m3, m1);             % . . . and difference
m1.prod(m2, m5);             % noncommutative matrix multiplication
m1.transposed(m6);           % creation of transpose . . .
m6.vector_prod(v3, v1);      % . . . and multiplication by a vector

% some additional operations
m1.hadamard_prod(m2, m3);    % entrywise product
m2.hadamard_quo(m3, m1);     % entrywise quotient m1(i,j) = m2(i,j) / m3(i,j)
c1 = m1.row_dot(3, v1);      % dot product of row 3 and vector v1
m1.column_cross(2, v3, v3);  % cross product of column 2 and vector v3 => v3
m1.row_sum(1, v1, v1);       % sum of row 1 and vector v1 => v1
m1.column_diff(4, v3, v3);   % row 4 minus vector v3 => v3
v3.column_diff(5, m1, v3);   % vector v3 minus row 5 => v3
m7.transpose();              % destructive transpose of square matrix
```


C.3.3.6 The Singular Value Decomposition

As we shall see in the sequel, several important VISICON objects require the representation of a subspace spanned by an arbitrary set of vectors. The set of spanning vectors is naturally represented in VISICON as the column vectors of a matrix. The subspace is then identified with the column space of the matrix. In practice, however, computations that involve the subspace require an orthonormal basis rather than a set of spanning vectors. We therefore require a behaviour that will find an orthonormal basis for the column space of a matrix.

Working with a set of spanning vectors can introduce a related numerical problem. It is well known [GVL89, Pre92] that operations on matrices that have nearly orthogonal column vectors are very stable. If, however, a column vector is independent of, but almost coplanar with, the other columns, calculations with the matrix can become numerically unstable. It is invariably better to discard troublesome, ‘almost linearly dependent’ column vectors from these matrices, replacing them with a zero vector, and thereby reducing the rank of the matrix. Remarkably, calculations with such a ‘numerically reduced rank’ matrix yield computational results that are significantly more reliable than those produced by the original matrix. Therefore, in order to perform reliable calculations, we need to be able to determine the *numerical* as well as regular ranks of a matrix, and remove the offending column vectors that contribute to numeric instability.

Another requirement found in VISICON classes is the ability to find a least squares (i.e. orthogonal) projection of a vector onto a subspace spanned by a set of vectors. In other words, if we represent the spanning vectors as the columns of a matrix, we must be able to find a least square projection of a vector onto the column space of the matrix. This type of projection is intimately related to the problem of finding a least squares solution to an over or under-determined system of linear equations. It is well known [Str80] that both problems require the creation of a Moore-Penrose pseudo-inverse for the matrix in question.

The three problems described above have a common solution in a remarkable matrix factorization called the *singular value decomposition* (SVD). Because of the importance of the above problems, instances of the class **Matrix** are able to decompose themselves into three SVD factors. We briefly review the important facts about the SVD here. The interested reader can find a detailed description of the theory behind the SVD in [GVL89, Str80].

Any $m \times n$ matrix **A** whose number of rows m is greater than or equal to its number of columns n , can be written as the product of an $m \times n$ column-orthogonal matrix **U**, an $n \times n$ orthogonal matrix **W** with positive or zero elements, and the transpose of an $n \times n$ orthogonal matrix **V**, as below:

$$\mathbf{A} = \mathbf{U} \operatorname{diag} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \mathbf{V}^T \quad (\text{C.3.1})$$

The matrices **U** and **V** are each orthogonal in the sense that their columns are orthogonal. Since **V** is square, it is also row-orthonormal. The decomposition (C.3.1) can always be performed.

The SVD exhibits a number of remarkable properties. First, the SVD explicitly constructs orthonormal bases for the null space and column space of **A**. Specifically, the columns of **U** whose same-numbered elements w_j are *nonzero* are an orthonormal set of basis vectors that span the range; the columns of **V** whose same-numbered elements w_j are *zero* are an orthonormal basis for the null space.

Another useful characteristic of the SVD is that it is useful in solving sets of simultaneous linear equations of the form

$$\mathbf{Ax} = \mathbf{b} \quad (\text{C.3.2})$$

where **x** and **b** are vectors. In this case we are interested in finding a vector **x** that satisfies (C.3.2) for a given **b**. It is quite clear from (C.3.1) that if **A** is square, say

$n \times n$, and the w_j are nonzero, then we can solve for $\mathbf{x}$ with the equation

$$\mathbf{x} = \mathbf{V} \left[\text{diag} \left(\frac{1}{w_j} \right) \right] \mathbf{U}^T \mathbf{b}. \quad (\text{C.3.3})$$

What is less clear, is that we can also use equation (C.3.3) to solve for $\mathbf{x}$ when $\mathbf{A}$ is *singular*. This is because the matrix

$$\mathbf{A}^+ = \mathbf{V} \left[\text{diag} \left(\frac{1}{w_j} \right) \right] \mathbf{U}^T \quad (\text{C.3.4})$$

is actually the *Moore-Penrose pseudoinverse*. This well known inverse (see [Str80] for example) selects a solution that is, in the least squares sense, the best one available for the given system (C.3.2). In the event that $\mathbf{b}$ lies in the column space of $\mathbf{A}$ (i.e. it is unique or over determined), then equation (C.3.3) will select the solution $\mathbf{x}$ with smallest length $|\mathbf{x}|^2$. If the system is underdetermined, and $\mathbf{b}$ is *not* an element of the column space of $\mathbf{A}$, then (C.3.3) finds an $\mathbf{x}$ for the *orthogonal projection* of $\mathbf{b}$ onto the column space of $\mathbf{A}$. So, the SVD provides us with a way to do orthogonal projections, as well as solve systems of simultaneous equations. In the event that some of the w_j are zero, all of the above holds true if we simply replace the values $\frac{1}{w_j}$ by zero if $w_j = 0$!

The last characteristic of the SVD that we shall discuss here is of great importance to our goal of providing stable numeric computations in VISICON. The *condition number* of a matrix is defined as the ratio of the largest of the w_j 's to the smallest of the w_j 's. We refer to the w_j 's as the *singular values* of the matrix. A matrix is singular if its condition number is infinite, and it is *ill-conditioned* if its condition number is too large, that is, if its reciprocal approaches the machine's floating point precision (for example, less than 10^{-6} for single precision **Float** scalars, or 10^{-12} for **Double** scalars).

Analytically, a matrix is either singular or it isn't. Numerically, however, the far more common situation is that some of the w_j 's are very small but nonzero, so that the matrix is ill-conditioned. In these cases, some of the column vectors in the matrix are 'almost' linearly dependent with other columns. In that case,

numerical computations involving the matrix may yield very inaccurate results. For example, LU decomposition or Gaussian elimination may produce a solution $\mathbf{x}$ to the system (C.3.2) that has wildly large components whose algebraic cancellation, when multiplying by the matrix $\mathbf{A}$, may give a very poor approximation to the right-hand vector $\mathbf{b}$. This case commonly arises in VISICON when performing orthogonal projections of a vector $\mathbf{b}$ onto the column space of a matrix, as in the discussion of the pseudoinverse above. Fortunately, in such cases, the solution vector $\mathbf{x}$ obtained by *zeroing* the small w_j 's and then using equation (C.3.3) is very often better than *both* the conventional solution *and* the SVD solution where the small w_j 's are left nonzero.

This seemingly paradoxical result works because, by zeroing very small w_j 's, we are throwing away a combination of equations that is so corrupted by round-off error as to be at best useless; usually it is worse than useless since it 'pulls' the solution vector way off towards infinity along some direction that is almost a null space vector.

VISICON classes make use of this last property of the SVD by 'filtering out' the singular values that are near zero. We use the threshold suggested in [Pre92, Chapter 2]. This threshold, represented by the named constant `VISICON_SVD_THRESHOLD` defined in file `visitype.h`, can be changed by the user if so desired. It can also be set on a matrix-by-matrix basis by sending the message `svd.threshold(value)` to a matrix before decomposing it.

VISICON provides the ability to perform the SVD as a matrix behaviour. A matrix in VISICON is factored by the message `svd(w, v)`. This is a destructive factorization where the receiving matrix elements are replaced with those for the factor $\mathbf{U}$. The argument $\mathbf{v}$ represents a pointer to the matrix $\mathbf{V}^5$, and $\mathbf{w}$ represents a pointer to a vector of singular values $\mathbf{w}$. Note, that the SVD assumes that the number of rows in the receiving matrix is greater than or equal to the number of columns, so an arbitrary matrix $\mathbf{A}$ might have to be padded with zero rows before

⁵The storage for $\mathbf{u}$ and $\mathbf{w}$ is allocated by `svd(w, v)`.

being decomposed. The VISICON objects that make use of the SVD handle these details without the user's intervention.

The message `pseudoinverse()` will nondestructively return the Moore-Penrose pseudoinverse of a receiving matrix. This message is provided for completeness, but is not used by existing VISICON classes. Instead, they provide behaviours to perform pseudoinversion directly from the three SVD factors described in the previous paragraph. This strategy saves memory. See, for example, Section C.3.5.3 for a description of the message `project(b)`.

C.3.4 Class `Vector_Field`

As we have seen in Chapters 2 and 3, the types of systems for which the theory of nonlinear control is best developed—in particular for the GNF networks that we study in this thesis—have evolution equations that are characterized as linear combinations of nonlinear vector fields. Furthermore, the static and dynamic controllers that we discuss in Chapter 3 are also expressed in terms of vector fields. Clearly then, vector fields are objects of central importance in the VISICON class abstraction. We will examine the class `Vector_Field`, and several of its derived classes, in this section.

The main purpose of the VISICON vector field hierarchy is to model vector fields that are important in the control of GNF networks, such as those that are involved in determining the reachability of open GNF systems, as described in Chapter 4. However, we have found that it is also often important to be able to construct 'classic' nonlinear and linear systems from a pallet of vector fields with well understood (and easily analyzed) properties. These example systems are useful for integration and regression test purposes because their behaviours are well known. In addition, comparing the dynamical behaviours of these analytically tractable systems to those of GNF systems can shed light on the qualitative dynamics of GNF systems. The VISICON vector field hierarchy therefore also contains a number of

classic and general purpose vector fields.

C.3.4.1 Common Vector_Field Behaviours and Representation

Conceptually, one may think of an n -dimensional vector field as a time sensitive mapping between an m -dimensional manifold⁶ $\mathcal{M}$ and $\mathcal{R}^n$.

$$t \times \mathcal{R}^m \mapsto \mathcal{R}^n$$

All vector field instances allow a sender to map a vector instance $u \in \mathcal{R}^m$ to a vector instance $v \in \mathcal{R}^n$ with the message `vector(u, v)`. In the event that $u \notin \mathcal{M}$, this message will return the undefined vector. The sender is expected to provide appropriately sized vector instances for u and v . All vector classes in the VISICON hierarchy are currently autonomous, so the time variable is ignored in the remainder of this section.

All vector fields must have an associated dimension. Consequently, instances of **Vector_Field** all have a protected **Unsigned_Integer** variable called `num_dim` that may be obtained with the message `dimension()`.

Finally, as we shall see in Section C.3.6, we can create an object that represents the integral curve of a vector field, passing through a point **p**, by sending the vector field the message `integral_curve(p)`.

Class **Vector_Field** is abstract, and so creation of concrete derived vector fields, and their specialized representation and behaviours will be covered in the remainder of this section. The notation introduced in this section will be used below.

C.3.4.2 The ‘Classic’ Vector Fields

A small but strategic set of derived vector field classes have been provided to allow the user to quickly construct classic, and easily analyzed nonlinear control systems. Such example systems can be found in abundance in the classic texts on

⁶In this thesis $\mathcal{R}^m$.

linear and nonlinear control [NvdS90, Isi89], and they can all be constructed from the following specialized subclasses of **Vector_Field**.

The simplest such vector field is the **Constant_Field**. A constant field always returns the same vector $\mathbf{v}$ independent of the input vector $\mathbf{u}$. A constant field c can be constructed by specifying the vector $\mathbf{v}$ as in `Constant_Field c(v)`. Each **Constant_Field** instance stores a private *copy* of $\mathbf{v}$. Alternatively, an uninitialized $\mathbf{v}$ can be constructed by specifying the scalar dimension `Constant_Field c(dimension)`. The contents of $\mathbf{v}$ can be accessed and assigned to using vector style square brackets.

Another particularly simple vector field is the **Identity_Field**. Given a vector $\mathbf{u}$, an identity field always returns $\mathbf{u}$. This field is useful when used in conjunction with the combined fields described below. An n -dimensional **Identity_Field** instance i is created with the expression `Identity_Field i(n)`.

A more general form of the **Identity_Field** is the **Mapping_Field**⁷. Conceptually, this field provides an arbitrary mapping of elements from $\mathbf{u}$ to elements of $\mathbf{v}$. An n -dimensional instance is created with the expression `Mapping_Field m(n)`. The message `map_elements(i, j)` maps the element $\mathbf{u}[i]$ to the element $\mathbf{v}[j]$. Elements of $\mathbf{v}$ that are not so mapped are set to 0.

A common building block of linear and nonlinear systems alike is the **Linear_Field**. Instances of this class apply a linear transformation $\mathcal{T}$ to the argument vector $\mathbf{u}$. Instance `lf`, which applies the linear transformation represented by the $n \times m$ column **Matrix** instance `mat`, is created with the expression `Linear_Field lf(mat)`. Note that a linear field created this way stores a private *copy* of the matrix `mat`. In the event that many instances need to perform exactly the same transformation, the user can elect to save storage by having each such field store a reference to the *identical* matrix object. If the user elects to do this, he is then responsible for deallocation

⁷These classes are both provided to allow a more space and time efficient implementation of the **Identity_Field**. The classes are not related by inheritance because their representations are unrelated

of the matrix. The following code illustrates the creation of two fields `lf1`. and `lf2` that share the same matrix `mat`:

```

Linear_Field lf1, lf2;           % create linear fields
lf1.transformation(mat);        % store matrix in first field
lf2.transformation(mat);        % store same matrix in second field

. . . further processing . . .

delete &mat;                    % user must deallocate matrix

```

Instances of the class **Oscillating_Field** allow the user to create fields having elements that oscillate with respect to any desired canonical axis in $\mathcal{R}^m$. Once an instance is created, the user must indicate the following information for each of the elements $1 \leq i \leq n$:

1. the index position $j \in [1, m]$ of the input vector element $u[j]$ that drives the oscillation of the output vector element $v[i]$.
2. the frequency of the oscillation in cycles per 2π radians.
3. the phase offset of the oscillation from 0 radians, and
4. the gain of the oscillation (1 means no gain).

This information is supplied in the message `oscillating_element(i, j, frequency, phase, gain)`. Alternatively, we may indicate that $v[i]$ does not oscillate and is zero with the message `zero_element(index)`⁸. For example, we may construct the oscillating vector field ($\mathcal{R}^4 \mapsto \mathcal{R}^3$)

$$\begin{pmatrix} \frac{\sin 2\pi u_3 + \frac{\pi}{2}}{4} \\ 10 \sin u_1 \\ 0 \end{pmatrix}$$

with the source code

⁸This could be accomplished by setting the amplitude to 0, however, the message `zero_element` is considered clearer.


```

Oscillating_Field o(3);
o.oscillating_element(1, 3, 2*PI, PI/2, 0.25);
o.oscillating_element(2, 1, 1, 0, 10);
o.zero_element(3);

```

Each **Oscillating_Field** instance stores three private vectors containing the frequency, phase and gain values respectively. The index mapping is represented by a private instance of **Mapping_Field**.

The next vector field class is the **Exponential_Field**. An instance, x , of this class maps the elements of any arbitrary instance of a **Vector_Field** subclass, f , to their exponentials (i.e. $e_i(u) \equiv e^{f_i(u)}$). For example, the vector field

$$\begin{pmatrix} e^{\frac{\sin 2\pi u_3 + \frac{\pi}{4}}{4}} \\ e^{10 \sin u_1} \\ 1 \end{pmatrix}$$

can be constructed from the oscillating field instance o described in the previous paragraph with the expression **Exponential_Field** $x(o)$. Each **Exponential_Field** instance stores a *copy* of the field that it operates on.

The next three field types are derived from the abstract **Combined_Field** class. These combine any number of arbitrary fields, including other combined fields, to produce a new field. For example, an instance of **Summation_Field** represents a linear combination

$$\sum_{i=1}^p c_i(u) f_i(u)$$

of fields f_i , where the c_i are scalar functions. An n -dimensional instance capable of summing up to p fields is created with the expression **Summation_Field** $s(n, p)$. Component fields are added to the instance, one at a time, with the message **add_field**(i, f), where i specifies the index, $1 \leq i \leq p$, of the field in the summation. The p coefficient functions are added as the elements of an arbitrary p -dimensional vector field, c with the message **coefficients**(c). The element index of a coefficient corresponds to the index position of the associated field in the

`add_field` message. Index positions that have not been associated with a field are assumed to be the constant 0 field, and therefore have no effect on the combined field. If a coefficient field has not been specified, then it is assumed that the coefficients are all 1.

Another type of combined field is the **Hadamard_Product_Field**. This field is exactly the same as the **Summation_Field** except that it represents the repeated product

$$\bigodot_{i=1}^p c_i(u) f_i(u),$$

where $\bigodot$ represents the entrywise Hadamard product. Index positions that have not been set with the `add_field` message refer to the constant field $(1 \cdots 1)^T$, and can therefore be ignored. Otherwise, exactly the same behaviours apply.

The last type of combined field, modeled by the **Composed_Field** class, also employs the same representation and message conventions as the **Summation_Field**. Instances differ in two ways. First, the component fields are composed in the order $f_p \circ f_{p-1} \circ \cdots \circ f_1$, and secondly, index positions that have not been associated with a field are assumed to be associated with an instance of **Identity_Field**, and can therefore be ignored.

It should be apparent to the reader that the combined fields together with the other ‘classic’ fields described in this section, allow the user to create a very rich family of nonlinear vector fields.

C.3.4.3 Vector Fields That Encapsulate Distributions

Perhaps more central to thesis are a number of vector fields that are used to encapsulate⁹ other types of VISICON objects that can, in the right context, be profitably thought of as defining a vector field-like mapping. In particular, we shall speak next of vector fields that encapsulate differential geometric distributions, and

⁹By *encapsulating* an object, we mean that these vector fields store the object (privately) as part of their representation so that they may obtain information from the object as needed.

those that encapsulate neural networks and other controllable systems.

Part of the motivation for providing these classes is that, within VISICON, vector fields provide a conceptual lingua franca for modelling the abstract mathematical operations that occur in control theory, and for communicating with VISICON simulation utilities. Encapsulating vector fields hide the differences between a large number of classes that are irrelevant to important simulation behaviours, and provide a simple and robust behavioural interface.

We will begin with the class **Distribution_Field**. As we will see in Section C.3.5, instances of the class **Distribution** represent instantiations of the differential geometric concept of a distribution. In a sense, distributions are a generalization of vector fields, as we saw in Chapter 3. However, rather than mapping vectors to vectors, they map vectors to linear subspaces spanned by a *set* of vectors. A vector field f is said to be an element of a distribution Δ if it is always locally contained in the distribution, i.e. $f(u) \in \Delta(u), \forall u \in \mathcal{M}$. A distribution thereby defines a *family* of vector fields, $\mathcal{F}$.

Instances of class **Distribution_Field**, and its subclasses, are used to represent a particular member of $\mathcal{F}$ that satisfies a property of interest. Currently, these classes are used to compute graphical objects for the visualization of distributions, as will be explained in detail in Sections C.3.5 and C.3.9.

C.3.4.3.1 Axial_Slice_Field Distributions that are well behaved—that satisfy regularity conditions such as complete integrability—induce a local foliation of their domain $\mathcal{M}$ into a set of unique maximal integral submanifolds. The dimension of the submanifolds may vary. As we have seen, such submanifolds are of central importance in this thesis. Therefore, given a point $u \in \mathcal{M}$, we would like to plot a rendering of the submanifold that passes through u . A class that is particularly important when rendering maximal integral submanifolds of a distribution is the **Axial_Slice_Field**.

We will only consider the case of a 2-dimensional submanifold here. The complete rendering algorithm is described in Section C.3.5. The basic idea is to

render a set of curves that represent the intersection of the submanifold passing through u , with parallel planes that ‘slice’ the submanifold at regular intervals. These planes are spanned by a pair of canonical axes—hence the name ‘axial slice’. The reader who imagines cutting a curling orange peel with parallel chops of a butcher’s hatchet will have a reasonable intuition for the kinds of intersection curves that we might expect to obtain when rendering a 2-dimensional submanifold. This process can be repeated with a set of planes in an orthogonal direction to the first to produce a ‘grid’ of *iso-curves* that conform to the surface of the submanifold and clearly define it for visual inspection.

In order to plot these iso-curves for a given distribution, we must find a vector field in $\mathcal{F}$ with *integral curves* that correspond to the desired iso-curves. This is what an instance of **Axial_Slice_Field** represents. The slice field is obtained either by sending a message `axial_slice_field()` to the **Distribution** instance, `dist`, that represents the distribution Δ of interest, or with the expression `Axial_Slice_Field f(dist)`. The field will be used to numerically integrate an integral curve $c(t)$ that has a derivative $\frac{dc(t)}{dt}$ s.t.

1. $\frac{dc(t)}{dt} \equiv f(c(t)), \forall t$, and
2. $f(c(t)) \in \mathcal{P}, \forall t$, where $\mathcal{P}$ is the 2-dimensional slicing plane, normal to the vector n .

In other words, given a point u , and a normal vector n , the slice field f must compute the subspace intersection $\mathcal{S} \equiv \Delta(u) \cap \mathcal{P}$, and pick from $\mathcal{S}$ an appropriate derivative $f(c(t))$.

An **Axial_Slice_Field** instance must work in conjunction with a numerical integration routine to select an appropriate $f(c(t))$ for every t of interest. We may, therefore, assume that there is a *current direction* of travel that is determined by the integration routine. This simply represents either an initial basis vector in $\mathcal{P}$ when we begin the integration, or it is the last derivative to have been computed along the integration path, at the previous time step. The private current direction

vector $\mathbf{v}$ can be initially set with the message `current_direction(v)`. Otherwise, the current direction is automatically updated by message `vector` to store the previous integration derivative. Message `cutting_plane(n)` can be sent to store the plane if intersection $\mathcal{P}$, given its normal vector $\mathbf{n}$.

The $f(c(t))$ selection algorithm operates as follows:

- If the local dimension of $\Delta(u)$ is 0, then we return the 0 vector. The integration routine will treat this as a stable fixed point and render a 0-dimensional submanifold as desired.
- If the local dimension is 1, we test for $\Delta(u) \subset \mathcal{P}$ by computing the dot product between a basis for $\Delta(u)$ and $\mathbf{n}$. If the dot product is close enough to 0¹⁰, then the distribution non-trivially intersects the plane, and we return the unit basis of $\Delta(u)$ that is closest to the current direction of travel (i.e. has positive inner product with it). If the intersection is trivial, we do the same so that the integration algorithm will ‘track’ the submanifold and render it correctly as a continuous integral curve.
- If the local dimension is 2, we compute the intersection between the planes, $\mathcal{P}$ and $\Delta(u)$. This is easily accomplished using simple algebra¹¹. A unit vector defined by the resulting equation of a line is computed, and is returned if it has positive inner product with the current direction of travel, otherwise its negative is returned.
- If the local dimension is ≥ 3 , we are free to return the current direction again. The idea here is that we can only render at most a 3-dimensional submanifold so $\mathcal{P} \subset \Delta(u)$ for all cutting planes $\mathcal{P}$, and *any* straight line is an iso-curve of

¹⁰Note that, due to numerical inaccuracy in these computations, we must allow a very small tolerance on the dot product test for intersection.

¹¹We must test for the pathological case where the two planes are parallel. This is accomplished by taking the dot product between the normal vector $\mathbf{n}$ and a basis of $\Delta(u)$. In the event that they are parallel we cannot slice the submanifold, and so we return the undefined vector to the integration routine to signal the ambiguity.

the submanifold at u . By repeatedly returning the basis of $\mathcal{P}$ that is initially selected as the current direction by the integration routine, we will integrate a straight line path. In this way the rendering routine can build up a set of grids that fill the 3-dimensional rendering space.

As we shall see in Section C.3.5, instances of the **Distribution** classes provide the necessary behaviours for determining its local numerical rank of a distribution, an orthonormal basis for the numerically stabilized distribution, and behaviours for performing numerical integration over submanifolds.

C.3.4.3.2 Shadow_Distribution We are concerned in this thesis with controlling the dynamical behaviour of systems by application of feedback that constrains their evolution to submanifolds of interest. These are actually maximum integral submanifolds determined by an appropriate distribution. In order to test the success of our control, we need a way to measure how well the trajectory that a system follows, as it evolves through time, conforms to the intended submanifold constraint. In other words, we would like to determine if the trajectory is *contained* in the intended submanifold, and if not, we would like a measure of how much it deviates from the submanifold. This measure is particularly relevant in a simulation environment where numerical inaccuracies inevitably creep into the integration of the simulation trajectory. The question of trajectory containment is one of degree in this case rather than a simple binary decision.

The **Shadow_Field** subclasses offer a useful way to track the evolution of a system, starting at initial point p , with respect to the unique maximal integral submanifold $\mathcal{S}$ of distribution Δ , that contains p . The general idea is as follows. If the vector field, f , that represents the evolution dynamics of the controlled system is an element of Δ , then the system trajectory will be contained in $\mathcal{S}$. However, if the trajectory deviates from $\mathcal{S}$, then there will be derivatives along the trajectory that are *not* contained in the underlying distribution (i.e. $\exists p' \in \mathcal{M}$ s.t. $f(p') \notin \Delta(p')$). We define the deviation of the trajectory from $\mathcal{S}$ with respect to an integral trajectory

that is completely contained in $\mathcal{S}$, and which tracks the system trajectory with a ‘minimum’ amount of error. Instances of the **Shadow_Field** subclasses represent vector fields that determine these minimum error *shadows* of the true trajectory on $\mathcal{S}$. The advantage of modeling the shadows as integral curves of vector fields is that they may be operated on, and numerically integrated, just like any other field. There are a number of error measures that we might define, each of which will yield a different shadow. Next, we describe two measures that are modeled by the classes **Derivative_Shadow_Field** and class **Integral_Shadow_Field**. Both measures are based on a type of least squares minimization of error.

The first type of shadow trajectory, represented by an instance of **Derivative_Shadow_Field**, minimizes the derivative error between the true trajectory, $c_f(t)$, and integral curves c_S contained in $\mathcal{S}$ i.e.

$$[c_S \subset \mathcal{S}] \left\{ \int \left| f(c_f) - \frac{dc_S}{dt} \right| dt \right\}.$$

We compute the error measure and minimum error shadow field as follows. The **Derivative_Shadow_Field** instance privately stores a *reference field*, r , that represents the evolution dynamics that produce the true system trajectory. We will see in Section C.3.4.4 several encapsulating vector field classes that can be used to construct such fields. In addition, the instance privately stores the distribution Δ . The orthogonal projection $\varphi_{\Delta(u)}(w)$ of a vector w onto the subspace $\Delta(u)$ finds, in the least squares sense, the derivative that minimizes $w' \in \Delta(u) \{ |w - w'| \}$. Therefore, the instance responds to the message **vector(u)** by computing and returning $\varphi_{\Delta(u)}(r(u))$. The trajectories obtained by numerically integrating the system dynamics r , and shadow dynamics, $\varphi_{\Delta(u)}(r(u))$, are minimally different in terms of their derivatives.

A derivative shadow field can be obtained for a system dynamics field r , and a distribution dist , with the expression **Derivative_Shadow_Field f(r, dist)**. The orthogonal projection is a behaviour of the distribution object, and we shall see below that it is a numerically stable behaviour. Each instance also maintains

two private variables that store the maximum and total derivative error from a sequence of `vector` messages. These variables can be reset to zero, before trajectory integration, with the message `reset_error`, and they can be obtained with the messages `max_error`, and `total_error`, respectively.

A more intuitively, and visually, appealing shadow trajectory is one that minimizes the $\mathcal{L}_2$ norm¹². In other words, the shadow trajectory ‘tracks’ the true trajectory as closely as possible, averaged over its entire length. More than one integral curve in $\mathcal{S}$ may satisfy this constraint. Finding such a trajectory would be computationally expensive however, so we employ a heuristic that ‘locally’ minimizes the metric. By this, we mean that at each time step, t' , of a numerical integration algorithm, we choose the shadow trajectory derivative that minimizes the Euclidean distance between the t'^{th} points on the true system trajectory c_f , and the shadow trajectory c_s (i.e. $\|c_f(t') - c_s(t')\|$). This shadow does not necessarily minimize the $\mathcal{L}_2$ metric because, like any local hill climbing heuristic, it might follow a local topological feature of the submanifold $\mathcal{S}$ that deviates from the globally minimum error path.

This kind of shadow field is represented by an instance of the **IntegralShadowField** class. The internal representation of this type of instance is the same as that for **DerivativeShadowField** instances¹³, and it is constructed the same way. Integral shadow fields also need to know the location of the true trajectory, so they actually compute the trajectory value $c_r(t')$ of the reference field r at each time t' , and store it in a private variable `current_reference_point`. The goal of minimizing the local distance, while constraining the shadow trajectory to lie in the submanifold $\mathcal{S}$, is achieved by selecting the derivative at u according to the following procedure:

$$f(t', u) = \begin{cases} \varphi_{\Delta(u)}(r(u)) & \text{if } u \equiv c_r(t') \\ \varphi_{\Delta(u)}(c_r(t') - u) & \text{otherwise} \end{cases}$$

¹²We are interested in $\mathcal{C}^\infty$ trajectories in this thesis.

¹³The common representation is inherited from class **ShadowField**.

In this way the shadow field exactly tracks the true trajectory whenever possible, and it attempts to ‘catch up’ to the true trajectory as quickly as possible, given the subspace of derivatives locally available in the distribution, after it is forced to deviate from the true trajectory. The result is an integral curve that tracks the true trajectory with *locally* least squared error. The maximum and total Euclidean error for a given integration is stored by each integral shadow field, and can be reset and obtained using the messages described above for derivative shadow fields.

C.3.4.4 Vector Fields That Encapsulate Dynamical Systems

We shall now briefly discuss another family of encapsulating vector fields modeled by the **Dynamical_System_Field** subclasses. These fields allow one to represent the behaviour of a nonlinear dynamical system as a vector field, hiding the peculiarities of the particular type of system, and allowing these systems to be operated on by various simulation utilities that numerically integrate vector field trajectories. For example, we might use a dynamical system encapsulator as a reference field to compute the shadow trajectory for a neural network, as described in Section C.3.4.3.2.

Indeed, the first type of dynamical system that we can encapsulate is a continuous time neural network, described in detail in Section `se:sw-sw-net`. As we saw in Chapter 2, the dynamics of a network are determined by neural evolution equations that are, in turn, parameterized by adaptive synaptic weights. The **CT_Net_Field** subclasses implement these neural evolution equations for various types of network architectures—in particular, for each of the GNF networks described in detail in Chapter 2. The encapsulating vector field stores a private reference to an instance of one of the **CT_Net** subclasses that represents the physical network architecture, and, as we shall see in Section `se:sw-sw-net`, the network also privately stores the vector field in order to represent its activation function. This two way parts relationship gives us the flexibility to view the network as a physical network of neurons, or

as a vector field for abstract control-related computations, and numerical simulation of network dynamics.

Instances of **CT_Net_Field** subclasses are actually created by a neural network object when it initializes itself. Notice that the mapping between **CT_Net** classes and **CT_Net_Field** classes is not one-to-one. This is because several of the neural networks have related evolution functions. Given a network, one can obtain its encapsulating field with the message `activation_field()`, and the encapsulating field will return the network when it receives the message `network()`. The user should use the `activation_field()` message to obtain a network encapsulating field, and should not construct the field directly. Encapsulating fields can be used anywhere a vector field object is required.

A network encapsulating field represents, η , the field that determines the derivative of the *activation space dynamics* of the network, as opposed to the *output dynamics*. The output function of a network, σ , can be obtained directly from a network in vector field form with the message `output_field()`. Alternatively, the output dynamics derivative, v , can be directly obtained from the encapsulating field for time t , and state vector u , with the message `output_vector(t, u, v)`.

The lack of automatic garbage collection in C++ introduces a minor memory allocation issue at this point. The two-way parts relationship between a network and its associated encapsulating field requires proper order of deallocation. Therefore, when either object is deallocated, its companion is also automatically deallocated. From the user's viewpoint, the simplest approach is to rely on the network to automatically deallocate its own encapsulating field.

Since neural networks are intelligent enough to obtain their own encapsulating fields, and the behaviours of all the subclasses are the same, we will not consider the individual subclasses further.

Finally, we briefly mention here that abstract feedback control systems can be constructed from pairs of plant and controller fields with the class **Control_System** and its subclasses. In particular, we shall explore systems in which the plant and con-

troller dynamics are determined by GNF networks. We provide vector fields that can encapsulate entire control systems with **Control_System_Field**. We will employ the control system model in a subsequent chapter where we will present a complete description.

C.3.5 Class Distribution

One of the most important control theoretic objects modeled by the VISICON library is the **Distribution**. As we have seen in Chapters 3 and 4, differential geometric distributions allow us to describe dynamics on nonlinear manifolds by mapping points on the manifold to their associated tangent spaces. When defined over the entire state space, completely integrable distributions induce a local partitioning of the space into submanifolds. A particularly important application of these distributions occurs in the control of the reachability manifold of neural networks. In this section we consider the class **Distribution** and its subclasses.

C.3.5.1 Instance Representation

Conceptually, a differential geometric distribution is a mapping

$$\Delta : \mathcal{M} \mapsto \text{span} \{f_1, \dots, f_d\}.$$

Consequently, an instance of class **Distribution** keeps a private array of **Vector_Field** pointers, called **fields**. C++ permits base class pointers (i.e. pointers to a superclass) to refer to subclass instances, therefore, a distribution may be spanned by any arbitrary set of VISICON vector fields. The size of this set is stored in the private **Unsigned_Int** variable **num_fields**, and may be obtained with the message **num_spanning_fields()**. The dimension of $\mathcal{M}$ is stored in the private **Unsigned_Int** variable **num_dim**, and is retrieved with the message **dimension()**.

Computations involving distributions are typically performed *locally*, that is, with respect to one state space location, $u \in \mathcal{M}$, at a time. The message

`position(u)` allows the user to specify the local context vector u , and `position()` allows the current position to be retrieved. The subspace spanned by the vector fields $\{f_1(u), \dots, f_d(u)\}$ must be represented in a numerically stable manner. This is conveniently accomplished with a **Matrix** instance

$$\begin{bmatrix} f_1(u) & \dots & f_d(u) \end{bmatrix}.$$

pointed to by the private variable `s`. Notice that this matrix is singular in general. The subspace $\Delta(u)$ is represented by column space of this matrix.

Three complications present themselves at this point. First, computations that involve the subspace $\Delta(u)$ are most easily carried out on an *orthogonal* basis of the space, rather than on a set of spanning vectors, such as those in matrix `s`. Secondly, `s` is in general singular—a situation that demands the use of the Moore-Penrose pseudo-inverse if orthogonal projections onto $\Delta(u)$ are required. Finally, it is well known that numerical computations involving matrices can yield meaningless results if the numerical rank of the matrix differs from the true rank. Fortunately, as we have already seen in Section C.3.3.6, the singular value decomposition offers a means for dealing with all three of these problems. Consequently, **Distribution** instances privately store the SVD factors for the matrix `s`. Variable `ub` points to a matrix whose columns are an orthonormal basis for $\Delta(u)$. Variable `vb` points to a matrix with orthogonal columns that are a basis for the null space of `s`. Finally, variable `w` points to a vector that contains the singular values of matrix `s`.

When the message `position(u)` is received by a distribution, it computes matrix `s`, and its SVD factors. The message `unfactored_position(u)` computes `s` without its SVD factors. When the SVD factors are computed, the singular values are automatically filtered by sending message `SDV_filter()` to vector `w`, as described in Section C.3.3.6. The true and numeric ranks are obtained from the filtered and unfiltered versions of `w`, and are stored in private **Unsigned_Int** variables `true_rank`, and `rank` respectively. A numerically stable equivalent of the spanning vector matrix `s` can be obtained with the message `spanning_set()`, which simply multiplies the

filtered SVD factors

$$u \text{diag}(w) v^T.$$

The numerical accuracy of the distribution behaviours described in the sequel are rendered as stable as possible since they used the filtered SVD factors.

C.3.5.2 Instance Creation

Any distribution that is spanned by vector fields that can be represented by instances of the VISICON **Vector_Field** classes can be represented by a VISICON **Distribution** instance. The instance is constructed by specifying the dimension of the distribution, and the maximum number of spanning fields it will store. Spanning vector fields are added, one at a time, with the message `add_field(i, field)`, where `i` is an index that allows the field to be retrieved with the message `field(i)`. For example, the following code creates a singular, smooth distribution, `dist`, spanned by 2 vector fields, that foliates $\mathcal{R}^3$ into concentric cylindrical manifolds with long axis along the z -axis, and a straight line submanifold coincident with the z -axis:

```
% create a constant vector field that points along the z-axis
Constant_Field z_axis(3);
z_axis.zero(); z_axis[3] = 1;

% create a vector field that is tangent
% to a circle in x/y plane, centered at the origin
Mapping_Field circ_tan(3);
circ_tan.map_elements(1, 2, 1);
circ_tan.map_elements(1, 2, -1);

% create the cylindrical foliation distribution
Distribution dist(3, 2);
dist.add_field(1, z_axis);
dist.add_field(2, circ_tan);
```

An important set of control theoretic distributions described in this thesis can be constructed by sending a message to an appropriate VISICON object. For

example, the open and closed GNF reachability distributions described in Chapter 4 are obtained by sending the messages `open_reachability_distribution()` or `closed_reachability_distribution()` to an instance of `GNF_Net` or one of its subclasses. We will introduce additional messages of this sort in relevant chapters of this thesis.

C.3.5.3 Subspace and Submanifold Behaviours

Locally, we can think of a distribution as a linear subspace. Consequently, VISICON distributions provide behaviours associated with local subspaces. The first such message, `basis()`, returns a matrix whose columns are an orthonormal basis for $\Delta(u)$ at the current position u . The message `orthogonal_complement()` returns a matrix whose columns are an orthonormal basis for the local orthogonal complement $\Delta(u)^\perp$. The message `project(b)` returns the projection of a vector b onto the local subspace $\Delta(u)$. This projection is accomplished by using the private SVD factors described in Section C.3.5.1 to perform a Moore-Penrose pseudoinversion. Given b , the vector x that is transformed by the local spanning matrix in variable s to the orthogonal projection onto the column space of s , is computed from the SVD components by the code (see Section C.3.3.6, equation (C.3.3))

```
Vector<VISICON_Real> temp;
u.prod(b, temp);
w.hadamard_prod(temp, x);
v.prod(x, temp);
```

and the projection vector is found by

```
s.prod(temp, x);
```

Several behaviours require a global view of the receiving distribution. For example, message `union(d)` returns a new **Distribution** instance that is the union of the receiving distribution and the distribution d . Another such behaviour, `maximum_integral_submanifold(p)`, returns an instance of the class **Manifold** (see

Section C.3.6) that represents the maximal integral submanifold passing through the point $\mathbf{p}$, and is consistent with the receiving distribution (i.e. the receiver is the tangent space for this submanifold, globally). Such manifolds are of importance when visualizing computational constraints on controlled neurocomputers.

Finally, the messages `distribution_shadow(f, u, c)` and `integral_shadow(f, u, c)` return **Manifold** instances that represent integral curve shadows c of the vector field $\mathbf{v}$, starting at vector $\mathbf{u}$, on the maximal integral submanifold of the receiver that passes through $\mathbf{u}$. See Section C.3.4.3.2 for a detailed description of vector field shadows.

C.3.6 Class Manifold

While distributions are the workhorse concept in control theory, they are not in themselves the real objects of interest. Instead, they are useful mathematical devices for describing the real objects of study—nonlinear *differentiable* manifolds. VISICON recognizes this fact by introducing the **Manifold** class. These objects can be used to retain state space submanifolds, vector field integral curves, leaves of manifold foliations, and maximum integral submanifolds of distributions. Furthermore, unlike distributions, which are usually too abstract to be pictured, differentiable manifolds can be rendered for purposes of visualization when they are of dimension three or less.

C.3.6.1 Instance Representation

Instances of the class **Manifold** represent differentiable manifolds whose tangent spaces are defined by an instance of the **Distribution** class. Each instance therefore stores a private pointer to the distribution, called `dist`. In order to be uniquely specify the submanifold within this distribution, any point contained within the submanifold must be specified. This point is stored in the private vector variable `position`. This point is typically the initial value of a numerical integration of

interest, or it represents a state value that has a computational interpretation of interest.

In VISICON, manifolds exist primarily to be drawn for visualization purposes. Therefore, each instance keeps a pointer to a **Linked_List<Linked_List>** instance called **visual_skeleton**. The visual skeleton of a submanifold consists of a list of iso-curves that define the surface of the submanifold. Each curve is actually a linked list of vectors. A private **Boolean** variable **visual_skeleton_computed** indicates whether the visual skeleton has been computed yet. The visual skeleton can be obtained with the message **visual_skeleton()**. An empty instance of **Linked_List<Vector<VISICON_Real> >** is returned if the visual skeleton has not been computed yet. The utility class **Manifold_Iterator** has been provided to allow the user to iterate over the iso-curves, and the vectors they contain, in a manifold's visual skeleton.

C.3.6.2 Instance Creation

Manifolds are created from various VISICON objects that uniquely describe the tangent space of the manifold. For example, the unique, maximal integral submanifold associated with a distribution, that passes through a vector **u**, can be obtained by sending the **Distribution** instance the message **maximal_integral_submanifold(u)**. A special case of this situation occurs when we wish to represent the integral curve of a vector field **f** (equivalent to a 1-dimensional distribution), that passes through the point **u**. In this case, we send the receiving vector field the message **integral_curve(u)**. The resulting **Manifold** instance stores a 1-dimensional **Distribution** instance spanned by the receiving vector field. Since neural networks and other dynamical systems can be represented as an encapsulating vector field, as we saw in Section C.3.4, instances of **Dynamical_System** also understand the message **integral_curve(u)**. The resulting manifold stores a distribution spanned by the encapsulating field for the dynamical object.

C.3.6.3 Rendering Behaviours

When a plotting utility is told to render a manifold, it sends the manifold the message `draw(plotter, integrator)`. The argument `plotter` is an instance of one of the **Plotter** subclasses that knows how to render on the host machine (see Section C.3.9). The argument `integrator` is an instance of one of the **Integrator** subclasses that has been selected by the sender to compute an iso-curve skeleton for the manifold (see Section C.3.7). The submanifold tests to see whether its `visual_skeleton` variable points to a linked list of curves yet. If it does, the existing set of curves is then given to the plotter for rendering. Otherwise, the integrator is used by the message `compute_visual_skeleton(integrator, t_first, t_last, num_steps)` to compute the skeleton. The integrator to use, the time interval of integration, and the number of integration steps to compute (i.e. resolution of integration) are given by the arguments. A recomputation of the skeleton can be forced at any time by sending this message.

The algorithm for computing the visual skeleton of a manifold works as follows. The basic idea is to produce a set of iso-curves emanating from the `position` vector that criss-cross the surface of the manifold in a distorted grid. These curves comprise the visual skeleton of the manifold. Rendering packages can then plot the surface that these lines define in various ways.

If the manifold is 1-dimensional, there is only one iso-curve that corresponds to the integral curve of the vector field defined by unit vectors in the local 1-dimensional subspaces. This curve can be found by numerically integrating this field. If the manifold is 2-dimensional, the iso-curves are found, conceptually, by intersecting this surface with regularly spaced planes that are parallel to the canonical axes. The difficulty associated with finding such iso-curves is that the manifold does not exist *per se*, only the defining distribution is available. If the manifold is of dimension three or higher, it takes up the entire rendering space, and so its iso-curves can be represented with a three dimensional grid that runs parallel to the canonical

axes.

Clearly, the only numerically interesting case is for the 2-dimensional manifold. This case is handled as follows. Given an intersecting plane, the problem is to select from the distribution Δ a vector field $f \in \Delta$ such that the integral curves of this field run parallel to the plane of intersection. VISICON models just such a vector field as an instance of the encapsulating vector field class **Axial_Slice_Field**. The details of how this vector field object works, and its associated protocol, are given in Section C.3.4.3.

Encapsulating vector fields greatly simplify the determination of the desired iso-curves—all that is required is to integrate along the integral curves of the field. The skeleton algorithm first tests the local numerical rank of the distribution at the **position** vector. This is the same as the dimension of the manifold. If the manifold is of dimension three or higher, a set of iso-curves with regularly spaced points is computed, in the form of a 3-dimensional grid centered at **position**, without resorting to numerical integration. Otherwise, an instance of **Axial_Slice_Field** is created to encapsulate Δ .

If the manifold is 1-dimensional, integral curves running forward and backward in time from vector **position** are computed for this field, using **position** as the initial value. If, on the other hand, the submanifold is 2-dimensional, then a ‘first backbone’ iso-curve is computed by numerically integrating along a pre-determined canonical axis in the positive and negative time directions. A ‘second backbone’ curve is also produced in exactly the same way by integrating along an axis orthogonal to the first. The remaining iso-curves are found by interpolating along these backbone curves. At regular intervals points are selected as initial values, and new curves are integrated ‘perpendicular’ to the backbone. The result is a set of curves that form a distorted grid which adheres to the surface of the manifold.

Points belonging to a curve are stored in integration order as a linked list. The complete set of curves is then stored in another linked list as the **visual_skeleton**. The length of a curve is determined by two factors. First, at each integration point

along a curve, the local numerical rank of the distribution is computed. If the rank changes, then the manifold is only locally defined, and the curve terminates. Failing this, the curve will be integrated for $2 * \text{num_steps}$ points over the time interval bounded by $\pm t_last$.

C.3.6.4 Shadowing Behaviours

We have already described the concept of shadowing in detail in Section C.3.4.3.2. We saw that a vector field may cast different types of ‘shadows’ onto the submanifold of a distribution. As we saw in Section C.3.5.3, it is possible to compute a manifold that represents a shadow by sending messages directly to the distribution. It is also possible to compute the shadow of a vector field onto a given manifold by sending one of the messages `distribution.shadow(f)` or `integral.shadow(f)` to the manifold object. A 1-dimensional shadow manifold is returned which is spanned by an appropriate encapsulating vector field as described in Section C.3.4.3.2.

C.3.7 Class Integrator

One of VISICON’s primary roles is to simulate the dynamical behaviour of neurocontrol systems in order to evaluate the results of computations that they perform, and to visualize the computational trajectories that they produce. The evolution equations of interest in this thesis are sets of first order, continuous time, ordinary differential equations of the form

$$\frac{dx_i(t)}{dt} = f_i(t, y_1, \dots, y_n), i = 1, \dots, n. \quad (\text{C.3.5})$$

We need to integrate these ODEs to produce the trajectory that such a system will follow from a given starting point in state space, for a given interval of time. This is known as the **initial value problem**.

Unfortunately, although the system dynamics are governed by *continuous time* evolution equations, on a digital computer we must content ourselves with

finding a *discrete time approximation* to the true trajectory. The idea underlying any algorithm for solving the initial value problem is always this: Rewrite the dx 's and dt 's in equation (C.3.5) as finite steps Δx and Δt , and multiply the equations by Δt . This gives *discrete time* algebraic formulas for the change in the functions when the independent variable t is 'stepped' by one 'step size' Δt . In the limit of making the step size very small, a good approximation to the underlying continuous time ODE is achieved.

A number of practical numerical methods based on the above idea have been developed. It is important to realize that there is no 'best' method. Each method has its own advantages and disadvantages—see [Pre92] for a good discussion of the trade-offs—and the user must *try* each method in turn, for a given problem, to see which one results in the greatest degree of accuracy in the least amount of time. It is fair to say that decreasing the step size does tend, in the limit, to produce a more faithful rendering of the continuous time dynamics. However, having said this, we note that very small step sizes can require an unacceptable amount of time to produce a solution.

VISICON provides the user with several integration methods, and its class library is easily extended to support additional ones. In VISICON, integration methods are modeled as *objects*, called 'integrators', rather than mathematical *behaviours*. All integrators are instances of the **Integrator** subclasses. This is appropriate because integrators have parts and a common behavioural protocol of their own, and they can be substituted for one another when performing a simulation. To try the same simulation behaviour with different integration techniques, the user repeatedly executes the same simulation message, passing in instances of different **Integrator** subclasses as an argument to the message.

The integration techniques currently available in VISICON are the 4th-order Runge-Kutta method, the *adaptive* 4th-order Runge-Kutta method and the Bulirsch-Stoer method. We briefly described each technique and its associated class in Section C.3.7.2. In the next section we look at the representation and behaviours common

to all of these classes.

C.3.7.1 Common Representation and Integration Behaviours

Class **Integrator** defines the common representation and behaviours inherited by all integrators. All integration techniques require access to the system of ODEs that they are to integrate. Since we represent ODE systems as vector fields in VISICON, every integrator contains a private pointer field that points to an instance of one of the **Vector_Field** subclasses. Any type of vector field may be integrated as long as it represents an integrable system over the range in question. If it does not, an arithmetic error will occur at run time, or a meaningless trajectory will be produced. The vector field can be set and accessed with the messages `field(field)` and `field()` respectively.

An integrator also needs to know the initial state vector to begin integrating from. Therefore, all integrators store an instance of **Vector<VISICON_Real>** in private variable `initial_value`. The initial value can be set and accessed with the messages `initial_value(vector)` and `initial_value()` respectively.

Finally, all integration algorithms need to know the interval of time over which the integration is to be performed. In the case of nonautonomous systems, this time interval determines the details of the time sensitive vector field being integrated. In the case of autonomous systems it is still important as an indicator of how much of the trajectory to integrate. The bounds of the closed interval are stored in the private variables `start_time` and `stop_time`.

The message that causes an integrator to actually perform an integration over the parts described above is, not surprisingly, called `integrate()`. This message constructs an instance of class **Manifold** which contains a fully populated visual skeleton. This skeleton is a linked list that stores the points computed during the integration calculation. See Section C.3.6 for a description of the manifold class and its visual skeleton part.

C.3.7.2 Specialized Types of Integrators

In this section we briefly describe the integration technique currently supported by VISICON, and 2 additional techniques that are planned as extensions to VISICON. To summarize

1. The Runge-Kutta method propagates a solution over an interval by combining the information from several Δt steps (each requiring an evaluation of the vector field $\mathbf{f}$ in equation (C.3.5)), and then using the information obtained to match a Taylor series expansion up to the k^{th} order. We provide the 4th-order Runge-Kutta method.
2. The Bulirsch-Stoer method (a special form of the family of Richardson extrapolation methods) extrapolates a computed result to the value that *would* have been obtained if the step size had been very much smaller than it actually was. In particular, extrapolation to zero step size is the desired goal. The modified midpoint method is used to take each individual step, and rational function extrapolation is used to perform the extrapolation step.

We propose that VISICON be extended to include *adaptive* forms of both of these methods in which the integration algorithm controls the allowable degree of numerical error by automatically adjusting the step size. The nonadaptive version of Runge-Kutta is a faster but less accurate version of the adaptive method. The user may decide to implement higher-order versions of the Runge-Kutta algorithm, however it should be remembered that higher-order does not necessarily translate into higher accuracy. The 4th-order version provided by VISICON is most often used in practice, and represents a mathematically motivated trade-off between order and algorithmic complexity (see [Pre92, Chapter 15]).

Although it is impossible to predict the performance of an integration technique on a particular problem, there are a few practical guidelines that can be followed. Adaptive algorithms are always preferred over the nonadaptive variety, unless

speed is important and only qualitative results are needed. Adaptive Runge-Kutta is used when one does not know any better, or when the Bulirsch-Stoer method fails to work, or when computational efficiency is not an issue. Runge-Kutta almost always succeeds in practice, but it is not usually the fastest technique. Bulirsch-Stoer methods are computationally more efficient on problems where very high accuracy is required, however, they are finicky and require more fine tuning than Runge-Kutta, and they do not behave well on nonsmooth vector fields, or when a singular point occurs inside the interval of integration. It has been said [Pre92] that Bulirsch-Stoer is a high-strung racehorse while Runge-Kutta is for ploughing the fields.

C.3.7.2.1 Class Fourth_Order_Runge_Kutta The most naive way to numerically integrate from an initial point $\mathbf{x}_i$ on the integral curve Φ_f^t (of vector field f) to the next point $\mathbf{x}_{i+1}$, is to simply move a small distance in the direction tangent to the curve, i.e. in the direction $f(t^i, \mathbf{x}_i)$. For example, the *Euler method* moves a distance determined by the step size h as follows:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + hf(t^i, \mathbf{x}_i).$$

However, if we take the Taylor expansion of the integral curve at the new point

$$\Phi_f^{t_i}(\mathbf{x}_{i+1}) = \mathbf{x}_i + \mathbf{f}'(\mathbf{x}_i + hf) + \frac{\mathbf{f}''}{2!}(\mathbf{x}_i + hf)^2 + \dots$$

we see that the error term is $O(h^2)$ —only one power of h smaller than the correction.

The error term is large because we are only making use of the derivative at the start of the integration step. We can do better than this by taking several Euler steps at strategic points, adding them together so that some of the low order error terms in the resulting sum of Taylor expansions cancel out. These sums are all called Runge-Kutta formulae.

Of course, many such schemes that combine different numbers of Euler points are possible. In general, we call a formula that cancels out the first $q - 1$ error terms of the Taylor expansion, a q^{th} -order Runge-Kutta method. It turns out

that for orders q greater than four, more than q evaluations of the vector field are required. So, in this sense, the 4th-order Runge-Kutta method is a natural breakpoint. It is certainly the most often used Runge-Kutta formula. It is evaluated as follows:

$$\begin{aligned}
 k_1 &= hf(t_i, \mathbf{x}_i) \\
 k_2 &= hf(t_i + \frac{h}{2}, \mathbf{x}_i + \frac{k_1}{2}) \\
 k_3 &= hf(t_i + \frac{h}{2}, \mathbf{x}_i + \frac{k_2}{2}) \\
 k_4 &= hf(t_i + h, \mathbf{x}_i + k_3) \\
 \mathbf{x}_{i+1} &= \mathbf{x}_i + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)
 \end{aligned}$$

Instances of the class **Fourth_Order_Runge_Kutta** perform the above computation when integrating. In addition to the time interval inherited from class **Integrator**, these instances also store a step size in private variable **step_size**. The step size can be set directly with the message **step_size(size)**, or it can be set indirectly by specifying how many points are to be evaluated over the time interval. In this case the message is **number_of_steps(number)**. The step size can be retrieved with the message **step_size()**.

C.3.7.2.2 Class Adaptive_Fourth_Order_Runge_Kutta Adaptive integrators change the step size automatically in order to stay below a predetermined threshold of error. Small steps are used when the vector field is particularly wild, and large steps are used when the vector field is smooth and well behaved. The integration algorithm requires information about its performance in the form of an estimate of truncation error.

Instances of the **Adaptive_Fourth_Order_Runge_Kutta** class, which will specialize the class **Fourth_Order_Runge_Kutta**, perform a type of adaptation called *step doubling*. In this scheme each Euler step is taken twice. Once as a full sized step, then again as two half steps. The difference between these two destinations θ is a useful indicator of truncation error along each canonical direc-

tion. The adaptation method adjusts h to keep the elements of θ below a vector of thresholds θ_0 . This is done by finding the element of θ that exceeds its threshold by the greatest amount, and then this amount is used to compute a scaling factor that reduces the step size h . Then the algorithm backs up and tries the integration step at the smaller step size.

The resulting adaptive integration algorithm requires more evaluations than the nonadaptive version, however, because the routine can take larger steps over smooth terrain, it can sometimes be *more* efficient than the nonadaptive method and yet retain higher accuracy in rough terrain.

Instances of this class store an error tolerance vector **tolerances** of the same dimension as the vector field to be integrated. The user can set this vector with the message **tolerances(vector)**. How should these values be selected? If the dependent variables of the vector field vary enormously in magnitude then the error should be set to small values such as $\epsilon = 10^{-6}$. If the variables oscillate, and are bounded by maximum values, then the thresholds should be set to ϵ times these maxima. The thresholds will default to the ϵ value given here.

C.3.8 Class **GNF_Net**

In this thesis we study the application of control theoretic notions to computation in neural networks. In Chapter 2, we described an important family of neural networks—the GNF systems—and we examined several special cases of interest. These networks are modeled in VISICON as instances of the class **GNF_Net** and its subclasses. **GNF_Net** inherits from a more general class, **CT_Net**, that models all continuous time networks, and this class inherits from the most general class **Dynamical_System**. We will examine the neural network classes in this section, and introduce other types of dynamical systems in the chapters that follow.

C.3.8.1 Instance Representation

All **CT_Net** instances represent networks of neurons connected by adjustable synaptic connections. The number of neurons is stored in the variable **num_neurons** and can be accessed with the message **size()**. The number of input channels that can be connected to neurons, and the number of output channels that combine the activation levels of the neurons, are stored in variables **num_inputs** and **num_outputs** and accessed with messages **input_size()** and **output_size()**, respectively.

The adjustable weights are stored, in general, in four matrices—two for the excitatory and inhibitory synaptic weights that connect one input to another, and two for the excitatory and inhibitory synaptic weights that connect the inputs to the individual neurons. Positive and negative synaptic weights are separated to facilitate the separation of synaptic valence into different populations of cells, as is the case in biologically realistic models. Valences can be mixed in a single matrix in subclasses of **CT_Net** if desired. The 2×2 array **weights** contains pointers to instances of **Matrix<VISICON_Real>**. Weights are accessed as described below.

Input functions, output functions, and neural activation functions are all represented by vector fields. The input and output fields may be arbitrary, but the activation dynamics, which are parameterized by the adjustable weights, must be represented by an encapsulating vector field of class **CT_Net_Field** (see Section C.3.4.4). This field stores the network instance so that it has access to the weights, and other important information, stored in the network. The input and output fields are set and retrieved by the messages **input_field(field)**, **input_field()**, **output_field(field)**, and **output_field()**. The activation function is automatically set when a network instance is initially constructed, as described in the next section. The encapsulating activation field can be obtained with the message **activation_field()**.

GNF_Networks store instances of the **Equation** class that represent the

functions a , b , and s in the general GNF evolution equation

$$\dot{x}_i = a_i(x) \left[b_i(x) + \sum_{j=1} n \left(s_i(x) \mathbf{H}_j^+ - \mathbf{H}_j^- \right) + \sum_{k=1} m \left(s_i(x) \mathbf{J}_k^+ - \mathbf{J}_k^- \right) \right].$$

These function objects may be set and retrieved with the messages `a(equation)`, `a()`, `b(equation)`, `b()`, `s(equation)`, and `s()`. When evaluated, they take two arguments: a neuron index, and the current activation state.

C.3.8.2 Weight Access and Assignment

Weight matrices are accessed using a parenthetic syntax in which the first argument specifies the type of connection, and the second indicates the valence of the weights. Within a matrix the rows represent post synaptic neurons that receive a dendritic connection. In the interconnection matrices the columns represent the presynaptic neurons that supply an axonal connection. In the input matrices, the columns represent input channels.

Named constants have been defined to simplify access of the four weight matrices. The first index in parentheses is either `C_WTS` to select the interconnection weights, or `I_WTS` to select the input weights. The second index is either `P_WTS` to select the positive (excitatory) weights, or `N_WTS` to select the negative (inhibitory) weights.

An assignment to a retrieved weight matrix element is made in the usual way.

For example, we add an inhibitory weight, that connects input channel 6 to neuron 10, to the excitatory weight that connects neuron 3 to itself, with the following expression

```
net(C_WTS, P_WTS)[3][3] += net(I_WTS, N_WTS)[10][6];
```

C.3.8.3 Instance Creation

When a network is initially created, one must specify the number of neurons that it contains. It will select default input, output, and activation fields if none are supplied by the programmer. For example, if no input or output fields are supplied, then the constant zero field is used for input (i.e. no input). The identity field of size `num_neurons` is selected as the default output field, producing outputs that equal the activation state of the network. A network always selects its own activation function based on the type of class. Each class of network defines or inherits the private message `set_default_activation_field()`, which is responsible for automatically setting up this field.

Weights must be initialized once a network is created. They may be accessed individually, or one of the following messages can be used to establish a canonical pattern of connections. All of the weights can be set to the same scalar value (typically zero) with the message `set_all_weights(scalar)`, or the weights of connection type `c` and valance `v` can be set with `set_weights(c, v, scalar)`. Alternatively, weight matrices may be fully or symmetrically randomized between the bounds `lower` and `upper` with the messages `randomize_all_weights()`, `randomize_weights(c, v, lower, upper)`, `symmetrize_all_weights()`, and `symmetrize_weights(c, v, lower, upper)`.

In the case of the GNF networks, one may specify the non-adaptive activation function terms *a*, *b*, and *c*. These functions default as follows:

$$\begin{aligned} a_i(x) &= x_i, \\ b_i(x) &= 1, \\ s_i(x) &= a_i(x)b_i(x). \end{aligned}$$

Subclasses of `GNF_Net` represent the familiar continuous time networks described in Chapter 2. They are all ultimately defined in terms of the functions *a*, *b*, and *s*, and the synaptic weights $\mathbf{H}^+$, $\mathbf{J}^+$, $\mathbf{H}^-$, and $\mathbf{J}^-$ (these are the matri-

ces `net(C_WTS, P_WTS).net(C_WTS, N_WTS).net(I_WTS, P_WTS).` and `net(I_WTS, N_WTS)` respectively). The type of network determines the form of these functions, the output function to be used, and the pattern of weight valences. The user need only supply appropriate parameters for these defaults when creating an instance. For example, as we saw in Chapter 2, instances of class **Shunting_Inhibition_Net** are defined by the functions

$$\begin{aligned} a_i(x) &= x_i, \\ b_i(x) &= \frac{AE}{x_i} - A, \\ s_i(x) &= \frac{1}{x_i} (B - x_i + E). \end{aligned}$$

Therefore, when an instance is created, only the three scalars A , B , and E need be specified along with the number of neurons. We will illustrate creation of the various specialized **GNF_Net** subclasses in later chapters.

The following source code illustrates the creation of a GNF network `net` with 10 neurons, symmetrically randomized weights between -10 and +10, self excitation connections set to zero, a classed sigmoidal output function, and two sinusoidal input signals at opposite phase. The nonadaptive activation terms are represented by the **Equation** instances `a`, `b`, and `s`.

```
% create the network with 10 neurons and 2 inputs
GNF_Net net(10, 2);

% initialize the weights
net.symmetrize_all_weights(-10, +10);
net(C_WTS, P_WTS).set_diag(0);
net(C_WTS, N_WTS).set_diag(0);

% set the nonadaptive terms
net.a(a); net.b(b); net.s(s);

% set the input function
net.input_field(new Oscillating_Field(2));
```

```

net.input_field().oscillating_element(1, 1, 1, 0, 1); % 2nd arg = time
net.input_field().oscillating_element(2, 1, 1, PI, 1); % 2nd arg = time

% set the output function
net.output_field(new Sinusoidal_Field(10, net.activation_field()));

```

C.3.8.4 Simulation of Network Dynamics

Numerical simulation of the *activation* dynamics of any neural network is accomplished by computing the integral curve of the network's encapsulating activation field. This field is obtained from the network with the `activation_field()` message, and it can be given to an instance of **Plotter** (Section C.3.9.1), or to an instance of an **Integrator** subclass, for numerical integration of the curve. Refer to Section C.3.4.4 for further information regarding the design of the encapsulating activation field.

The *output* dynamics of a network are found by applying the output field of the network, obtained with the `output_field()` message, to points on the activation space trajectory.

The following code draws the trajectory that passes through the origin, for the network defined in Section C.3.8.3. An adaptive 4th order Runge-Kutta integrator is used.

```

% create the integral curve passing through the origin
Vector<VISICON_Real> origin(10); origin.zero();
Manifold *curve = &net.activation_field().integral_curve(origin);

% create an integrator to compute the points on the curve
Adaptive_Fourth_Order_Runge_Kutta rk4(4, 0, 10, 100);

% plot the curve
GNU_Plot plot;
plot.draw_manifold(*curve, rk4);

```

C.3.9 Class **Plotter**

As described in Section C.2.3. VISICON achieves portability of its rendering utilities by hiding the host specific rendering package behind an instance of one of the **Plotter** classes. The technical details of how the plotter object communicate with their utilities is not of interest here.

One shareware utility is currently supported for the UNIX environment: GNUplot. This utility is also available in other environments. The plotter classes can easily be extended from the current set to accommodate other utilities such as POVRAY.

C.3.9.1 Instance Creation and Rendering Behaviours

Instances are created with an expression such as `GNU_plot p`. Representation is highly variable between classes.

Plotters will render a single submanifold when sent the message `draw_manifold(manifold, integrator)`, or a set of manifolds stored in a linked list when sent the message `draw_manifolds(list, integrator)`. An integrator object must be sent to numerically integrate iso-curves on the manifold. A submanifold of a given distribution that contains a point p can be rendered with the message `draw_submanifold(distribution, p, integrator)`. A derivative shadow and integral shadow, described in detail in Section C.3.4.3.2. can be rendered by sending the messages `draw_derivative_shadow(distribution, field, p)` or `draw_integral_shadow(distribution, field, p)`, respectively. Alternatively, one can first generate a shadow manifold, as described in Sections C.3.5.3 and C.3.6.4. and then have the plotter draw this manifold.

As described in Section C.3.6.3. manifolds are only rendered in the local neighborhood of their `position` vector in which they are of constant dimension.

C.3.10 The Utility Classes

VISICON supplies a number of miscellaneous classes that provide useful utility functions. These include a **Linked_List<Element_Class>** template for defining linked lists of **Element_Class** instances, an **Equation** subtree for representing differentiable equations of interest, and various C++ iterator classes, such as **Manifold_Iterator** and **Linked_List_Iterator**, for iterating through collections which are parts of VISICON objects.

Bibliography

- [AHS85] David H. Ackley, Geoffrey E. Hinton, and Terrance J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*. 9:147–169, 1985.
- [Ama95] Shun-Ichi Amari. Information geometry of the em and em algorithms for neural networks. *Neural Networks*, 8(9):1379–1408, 1995.
- [AS92] Francesca Albertini and Eduardo D. Sontag. For neural networks, function determines form. Report SYCON-92-03, Rutgers Center for Systems and Control, May 1992.
- [ASRJ77] James A. Anderson, Jack W. Silverstein, Stephen A. Ritz, and Randall S. Jones. Distinctive features, categorical perception, and probability learning: some applications of a neural model. *Psychological Review*, 84:413–451, 1977.
- [BA85] A.G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 15:360–375, 1985.
- [Bat94] Stefano Battilotti, editor. *Noninteracting control with stability for nonlinear systems*. Springer-Verlag, London, 1994.

- [Ben96] Yoshua Bengio. *Neural networks for speech and sequence recognition*. International Thomson Computer Press. London, England, 1996.
- [BH89] Eric B. Baum and David Haussler. What size net gives valid generalization ? *Neural Computation*, 1:151–160, 1989.
- [Blo62] H.D. Block. The perceptron: a model for brain functioning. i. *Reviews of Modern Physics*, 34:123–135, 1962.
- [Boo86] William M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press, 2 edition, 1986.
- [BOU94] A. BOUANANI. Study on holonomic and nonholonomic links in a human walking mechanism. *ARCHIVES INTERNATIONALES DE PHYSIOLOGIE DE BIOCHIMIE ET DE BIOPHYSIQUE*, 102(5):C 42-C 42, September 1994.
- [Bra95] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138:67–100, 1995.
- [BT95] J. R. Busemeyer and J. T. Townsend. *Dynamic representation of decision making*. In Port and van Gelder [PvG95], 1995.
- [CG83] Michael A. Cohen and Stephen Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:815–826, 1983.
- [CG86] Gail Carpenter and Stephen Grossberg. Adaptive resonance theory: Stable self-organization of neural recognition codes in response to arbitrary lists of input patterns. *Proceedings of the Cognitive Science Society*, 1986.

- [Cog85] Cognitive Science Society. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*. Hillsdale, N.J. 1985.
- [Cog87] Cognitive Science Society. *Proceedings of the 9th Annual Conference of the Cognitive Science Society*. Hillsdale, N.J. 1987.
- [Coo94] P. A. Cook, editor. *Nonlinear dynamical systems*. Prentice Hall, New York. 2 edition. 1994.
- [CS83] G. W. Cottrell and S. L. Small. A connectionist scheme for modelling word-sense disambiguation. *Cognition and Brain Theory*, 6:89–120, 1983.
- [CS87] E. Charniak and E. Santos. A connectionist context-free parser which is not context-free but then is not really connectionist either. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society* [Cog87], pages 587–612.
- [CT95] Gail Carpenter and Ah-Hwee Tan. Rule extraction: From neural architecture to symbolic representation. *Connection Science*, 7(1):3–28, 1995.
- [Cyb89] G. Cybenko. Approximation by superposition of a sigmoidal function. *Math. Control Systems Signals*, 1989.
- [Dec96] Gustavo Deco. *An Information-theoretic approach to neural computing*. Springer, New York, NY, 1996.
- [Der91] M. Derthick. *Mundane reasoning by settling on a plausible model*, pages 107–157. Volume 1 of Hinton [Hin91], 1991.
- [Doo90] B. F. Doolin. *An Introduction to Differential Geometry for Engineers*. Dekker, New York, NY, 1990.
- [DS96] Ajit T. Dingankar and Irwin W. Sandberg. Tensor product neural networks and approximation of dynamical systems. pages 1–4. Atlanta, Georgia, May 1996. IEEE ISCAS. Invited paper.

- [eZMP96] Hayes M.J.D. et Zsombor-Murray P.J. *Kinematic mapping of 3-legged planar platform with holonomic higher pair*. pages 421–430. In Parenti-Castelli [PC96]. 1996.
- [Fol89] P. Foldiak. Adaptive network for optimal linear feature extraction. *International Joint Conference on Neural Networks-89*. 1:401–405. 1989.
- [FP88] J.A. Fodor and Z.W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*. 28:3–71. 1988.
- [Fre94] Walter J. Freeman. Neural mechanism underlying destabilization of cortex by sensory input. *Physica D*. 75:151–164. 1994.
- [Fro90] Gilbert F. Froment. *Chemical reactor analysis and design*. Wiley. 2 edition. 1990.
- [Fun89] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*. 2:183–192. 1989.
- [Gal93] Stephen Gallant. *Neural Network Learning and Expert Systems*. MIT Press. Cambridge, MA. 1993.
- [GO93] C. Lee Giles and Christian W. Omlin. Extraction, insertion and refinement of symbolic rules in dynamically driven recurrent neural networks. *Connection Science*. 5(3, 4):307–337. 1993.
- [Goh86] Israel Gohberg. *Invariant subspaces of matrices with applications*. Canadian Mathematical Society series of monographs and advanced texts. Wiley. New York, NY. 1986.
- [Gol96] Richard M. Golden. *Mathematical Methods for Neural Network Analysis and Design*. MIT Press. Cambridge, MA. 1996.

- [Gra97] Daryl H. Graf. Incorporating state space constraints into a neural network. In *Proceedings of the International Conference on Neural Networks*. Houston, TX. June 1997. ICNN/INNS.
- [Gra98a] Daryl H. Graf. Combining computational abstractions in connectionist networks: augmenting energy landscapes with manifold constraints. 1998. In preparation.
- [Gra98b] Daryl H. Graf. Connectionist controllability: understanding the contributions of adaptive and non-adaptive terms. 1998. In preparation.
- [Gra98c] Daryl H. Graf. Connectionist types: enforcing type constraints on connectionist variable bindings. 1998. In preparation.
- [Gra98d] Daryl H. Graf. Gnf network dynamics can be compensated to enforce nonlinear manifold constraints. 1998. In preparation.
- [Gra98e] Daryl H. Graf. Holonomic networks: confining network dynamics to sub-manifolds of activation space. 1998. In preparation.
- [Gra98f] Daryl H. Graf. Hybrid connectionist control: using adaptive feedback to enforce manifold constraints. 1998. In preparation.
- [Gra98g] Daryl H. Graf. Manifold constraints: A new paradigm for connectionist computation. 1998. In preparation.
- [Gra98h] Daryl H. Graf. Modular systems of experts that compete to learn reachability constraints. 1998. In preparation.
- [Gra98i] Daryl H. Graf. New observability results for continuous time connectionist networks. 1998. In preparation.
- [Gra98j] Daryl H. Graf. Pma nets: representing nonlinear foliations of space in adaptive connectionist systems. 1998. In preparation.

- [Gra98k] Daryl H. Graf. Principal manifold approximation: a method for learning manifold constraints. 1998. In preparation.
- [Gra98l] Daryl H. Graf. Systems that represent and learn constraints for cognitive dynamics. 1998. In preparation.
- [Gra98m] Daryl H. Graf. Using nonlinear manifolds as a novel class of recognition template. 1998. In preparation.
- [Gra98n] Daryl H. Graf. Visicon: an object-oriented framework for simulating hybrid connectionist control systems. 1998. In preparation.
- [Gro67] Stephen Grossberg. Nonlinear difference-differential equations in prediction and learning theory. *Proceedings of the National Academy of Sciences*. 58:1329-1334. 1967.
- [Gro68] Stephen Grossberg. Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity. *Proceedings of the National Academy of Sciences*. 59:368-372. 1968.
- [Gro73] Stephen Grossberg. Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*. 52:217-257. 1973.
- [Gro76] Stephen Grossberg. Adaptive pattern classification and universal recording: I. parallel development and coding of neural feature detectors. *Biological Cybernetics*. 23:121-134. 1976.
- [Gro87a] Stephen Grossberg. *The Adaptive Brain*, volume 1. Elsevier Science Publishing Co., New York, NY. 1987.
- [Gro87b] Stephen Grossberg. *The Adaptive Brain*, volume 2. Elsevier Science Publishing Co., New York, NY. 1987.

- [GVL89] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Number 3 in Johns Hopkins series in the mathematical sciences. Johns Hopkins U. P., Baltimore, MD. 2 edition, 1989.
- [GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Number 3 in Johns Hopkins series in the mathematical sciences. Johns Hopkins U. P., Baltimore, MD. 3 edition, 1996.
- [Heb49] Donald. O. Hebb. *The Organization of Behaviour, a Neuropsychological Theory*. John Wiley and Sons, New York, NY, 1949.
- [Hin91] Geoffrey Hinton, editor. *Connectionist Symbol Processing*. MIT Press, Cambridge, MA, 1991.
- [HK92] K. Hornik and C. M. Kuan. Convergence analysis of local feature extraction algorithms. *Neural Networks*, 5:229–240, 1992.
- [HN90] Robert Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, Reading, Mass., 1990.
- [Hop84] John Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science*, 81:3088–3092, 1984.
- [HT89] K. J. Holyoak and P. Thagard. Analogical mapping by constraint satisfaction. *Cognitive Science*, 13:295–355, 1989.
- [ICN97] ICNN/INNS. *Proceedings of the International Conference on Neural Networks*. Houston, TX, June 1997.
- [IEE88] IEEE. *Proceedings of the IEEE Intl. Conference on Neural Networks*. San Diego, CA, 1988.
- [IEE91] IEEE. *Proceedings of the I.E.E.E. Conference on Robotics and Automation*. Sacramento, CA, April 1991.

- [IEE92] IEEE. *Proceedings of the IEEE Conference on Robotics and Automation*. Nice, France, May 1992.
- [Isi89] Alberto Isidori. *Nonlinear Control Systems*. Springer-Verlag, Berlin, 2 edition, 1989.
- [Jae97] Herbert Jaeger. From continuous dynamics to symbols. In *Dynamics, Synergetics, Autonomous Agents* [Jcc97]. To appear.
- [Jag94] Arun Jagota. *Representing discrete structures in a Hopfield-style network*, pages 123–142. In Levine and Aparicio [LA94], 1994.
- [Jcc97] *First Joint Conference on Complex Systems in Psychology*. Gstaad, Switzerland, March 1997.
- [JJ94] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the em-algorithm. *Neural Computation*, 6:181–214, 1994.
- [JJNH91] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
- [Joh88] C. Richard Johnson. *Lectures on Adaptive Parameter Estimation*. Prentice-Hall, Englewood Cliffs, N.J, 1988.
- [Jud90] J. Stephen Judd. *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, MA, 1990.
- [Kan90] Kunihiro Kaneko. Clustering, coding, switching, hierarchical ordering, and control in a network of chaotic elements. *Physica D*, 41:137–172, 1990.
- [Kat82] Tosio Kato, editor. *A Short introduction to perturbation theory for linear operators*. Springer-Verlag, New York, 1982.

- [KC78] H. J. Kushner and D. S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, New York, NY, 1978.
- [Kel95] J. A. S. Kelso, editor. *Dynamic Patterns: The Self-Organization of Brain and Behaviour*. MIT Press, Cambridge, MA, 1995.
- [KM94] M Karpinski and A. Macintyre. Quadratic bounds for vc-dimension of sigmoidal neural nets. Connectionists Archive, 1994.
- [Knu66] Donald E. Knudson. *1966 World Gnus Conference*. Genetic algorithms. Dept. of Comp. Sci., MIT, Novosibirsk, February 1966. some note.
- [Koh77] Teuvo Kohonen. *Associative Memory: A System-Theoretical Approach*. Springer-Verlag, Berlin, Heidelberg, 1977.
- [Koh89] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, Heidelberg, 3 edition, 1989.
- [Kos88] Bart Kosko. Bidirectional associative memories. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):49–60, 1988.
- [KP92] S. M. Killough and F. G. Pin. Design of an omnidirectional holonomic wheeled platform prototype. In *Proceedings of the IEEE Conference on Robotics and Automation* [IEE92], pages 84–90.
- [LA94] Daniel S. Levine and Manuel Aparicio, editors. *Neural Networks for Knowledge Representation and Inference*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1994.
- [LC93] Zexiang Li and J. F. Canny. *Nonholonomic Motion Planning*. Kluwer Academic Publishers, Norwell, MA, 1993.

- [LN93] Asriel U. Levin and Kumpati S. Narendra. Control of nonlinear dynamical systems using neural networks - controllability and stabilization. *IEEE Transactions on Neural Networks*, 4, March 1993.
- [LN96] Asriel U. Levin and Kumpati S. Narendra. Control of nonlinear dynamical systems using neural networks - part ii: Observability, identification, and control. *IEEE Transactions on Neural Networks*, 7(1):30-42, 1996.
- [LS91] Gerardo Lafferriere and Hector J. Sussmann. Motion planning for controllable systems without drift. In *Proceedings of the I.E.E.E. Conference on Robotics and Automation* [IEE91], pages 1148-1153.
- [LS97] Gerardo Lafferriere and Hector J. Sussmann. A differential geometric approach to motion planning. Preprint, Dept. of Mathematics, Portland State University, 1997. WWW URL <http://www.mth.pdx.edu/Papers/>.
- [Maa94] Wolfgang Maass. On the computational complexity of networks of spiking neurons. *Advances of Neural Information Processing Systems (NIPS '94)*, 7, 1994.
- [Maa95] Wolfgang Maass. Vapnik-chervonenkis dimension of neural nets. *The Handbook of Brain Theory and Neural Networks*, pages 47-34, 1995.
- [Mac94] Bruce MacLennan. *Continuous symbol systems: The logic of connectionism*, pages 83-122. In Levine and Aparicio [LA94], 1994.
- [Med94] Larry Medsker. *Hybrid Neural Networks and Expert Systems*. Kluwer Academic, Boston, 1994.
- [MI92] Ferdinando A. Mussa-Ivaldi. From basis functions to basis fields: vector field approximation from sparse data. *Biological Cybernetics*, 67:479-489, 1992.

- [Mic86] C.A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*. 2:11–22. 1986.
- [MIG92] Ferdinando A. Mussa-Ivaldi and Simon F. Giszter. Vector field approximation: a computational paradigm for motor control and learning. *Biological Cybernetics*. 67:491–500. 1992.
- [Mii93] Risto Miikkulainen. *Subsymbolic natural language processing: an integrated model of scripts, lexicon, and memory*. MIT Press, Cambridge, Mass., 1993.
- [MKK97] K. Warwick M. Karny and V. Kurkov, editors. *Dealing with Complexity: a Neural Network Approach*. Springer-Verlag, London. 1997.
- [ML94] Richard M. Murray and S. Shankar Li. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, Florida. 1994.
- [Mni96] F. Mnif. An adaptive sliding-mode approach for force/motion control of manipulators under holonomic constraints. *CANADIAN JOURNAL OF ELECTRICAL AND COMPUTER ENGINEERING*. 21(2):73–80. 1996.
- [MP43] W.S. McCulloch and W.H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. 5:115–133. 1943.
- [MP69] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge, MA. 1969.
- [MP91] A.S. Mikhailov and I.Yu. Poteryaiko. Attractive seperatrix loops in connectionist models. *Physica D*. 53:13–24. 1991.
- [MS97] Wolfgang Maass and Eduardo D. Sontag. Analog neural nets with gaussian or other common noise distributions cannot recognize arbitrary regular

- languages. Preprint. <http://www.math.rutgers.edu/~sontag/papers.html>. 1997.
- [MV97] Yves Moreau and Joos Vandewalle. When do dynamical neural networks with and without hidden layer have identical behaviour? Research Report ESAT-SISTA/TR 1997-51. Departement Elektrotechniek. Katholieke Universiteit Leuven. Belgium. June 1997. Submitted to Neurocomputing.
- [NvdS90] Henk Nijmeijer and Arjan van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag. New York. 1990.
- [Oja82] Erkki Oja. A simplified neuron model as a principal components analyzer. *Journal of Mathematical Biology*, 15:267–273. 1982.
- [Oja92] Erkki Oja. Principal components, minor components and linear neural networks. *Neural Networks*, 5:927–935. 1992.
- [OK85] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications*, 106:69–84. 1985.
- [OTG97] Christian W. Omlin, Karvel K. Thornber, and C. Lee Giles. Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks. *IEEE Transactions on Fuzzy Systems*. 1997. Accepted for publication.
- [Pap91] Athanasios Papoulis. *Probability, random variables, and stochastic processes*. McGraw-Hill. New York. 3 edition. 1991.
- [PC96] J. Lenarcic V. Parenti-Castelli, editor. *Recent Advances in Robot Kinematics*. Kluwer. 1996.
- [PG96] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497. 1996.

- [Pin87] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*. 59(19):2229–2232. 1987.
- [Pin94] Gadi Pinkas. *Propositional logic, nonmonotonic reasoning and symmetric networks - on bridging the gap between symbolic and connectionist knowledge representation*. pages 175–204. In Levine and Aparicio [LA94], 1994.
- [PK94] F. G. Pin and M. Killough. A new family of omnidirectional and holonomic wheeled platforms for mobile robots. *IEEE Transactions on Robotics and Automation*. 10(4). August 1994.
- [PL95] D.T. Pham and Xing Liu. *Neural Networks for Identification, Prediction and Control*. Springer-Verlag, New York. 1995.
- [Pla94] Tony A. Plate. *Distributed Representations and Nested Compositional Structure*. Knowledge representation. Graduate Dept. of Comp. Sci., University of Toronto, Toronto. 1994.
- [Plu91] Mark D. Plumbley. On information theory and unsupervised neural networks. Report CUED/F-INFENG/TR 78. Cambridge University Engineering Department. 1991.
- [Plu93] Mark D. Plumbley. Efficient information transfer and anti-hebbian neural networks. *Neural Networks*. 6(2):823–833. 1993.
- [Plu95] Mark D. Plumbley. Lyapunov functions for convergence of principal component algorithms. *Neural Networks*. 8(1):11–23. 1995.
- [Poh93] Ira Pohl. *Object-Oriented Programming Using C++*. Benjamin/Cummings Publishing Co., Redwood City, CA. 2 edition. 1993.
- [Pow87] M. Powell. Radial basis functions for multivariable interpolation. in *Algorithms for Approximation*. Mason and Cox (eds). 78(9):137–167. 1987.

- [Pre92] William H. Press. *Numerical Recipes in C: The Art of Scientific Computing*. U. P., Cambridge, England. 2 edition. 1992.
- [PvG95] R. Port and T. van Gelder, editors. *Mind as Motion*. MIT Press, Cambridge, MA. 1995.
- [RHW86] David.E. Rummelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*. 323:533-536. 1986.
- [RM86a] D.E. Rummelhart and J.L. McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, volume 1. MIT Press, Cambridge, MA. 1986.
- [RM86b] D.E. Rummelhart and J.L. McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, volume 2. MIT Press, Cambridge, MA. 1986.
- [Ros58] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*. 65(6):386-408. 1958.
- [RT89] J. Rubner and P. Tavan. A self-organizing network for principal components analysis. *Europhysics Letters*. 10:693-698. 1989.
- [San89] Terence D. Sanger. Optimal unsupervised learning in a single layer linear feedforward neural network. *Neural Networks*. 2(6):459-474. 1989.
- [SB95] Ron Sun and Lawrence Bookman. *Computational architectures integrating neural and symbolic processes: a perspective on the state of the art*, volume SECS 292 of *Kluwer Intl. Series in Engineering and Computer Science*. Kluwer, Novosibirsk. 1995.

- [SH85] B. Selman and G. Hirst. A rule-based connectionist parsing system. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* [Cog85], pages 212-221.
- [Sha88] L. Shastri. *Semantic networks: An evidential formalization and its connectionist realization*. Pitman, London, 1988.
- [Sha94] Jude Shavlik. Combining symbolic and neural learning. *Machine Learning*, 14(3):321-332, 1994.
- [SL88] H.M. Stoll and L.S. Lee. Continuous time optical neural network. In *Proceedings of the IEEE Intl. Conference on Neural Networks* [IEE88].
- [Smo91] Paul Smolensky. *Tensor product variable binding and the representation of symbolic structures in connectionist systems*, pages 159-216. Volume 1 of Hinton [Hin91], 1991.
- [Son97] Eduardo Sontag. *Recurrent neural networks: Some systems-theoretic aspects*, pages 1-12. In M. Karny and Kurkov [MKK97], 1997.
- [SS91] H.T. Siegelmann and E.D. Sontag. Turing computation with neural nets. *Applied Mathematics Letters*, 4(6):77-80, 1991.
- [SS92] H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. *ACM Workshop on Computational Learning Theory (COLT '92)*, pages 440-449, 1992.
- [Str80] Gilbert Strang. *Linear algebra and its applications*. Academic Press, New York, NY, 2 edition, 1980.
- [Str91] Bjarne Stroustrup. *The C++ programming language*. Addison-Wesley, Reading, MA, 2 edition, 1991.
- [Sun94] Ron Sun. *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. Wiley, New York, 1994.

- [Tou91] D.S. Touretzki. *Dynamic symbol structures in a connectionist network*. pages 5–46. Volume 1 of Hinton [Hin91]. 1991.
- [Tsu94] Ichiro Tsuda. Can stochastic renewal of maps be a model for cerebral cortex? *Physica D*. 75:165–178. 1994.
- [Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*. 27(11):1134–1142. 1984.
- [Val92] Natana Valiveti. Parallel computational geometry on analog hopfield networks. Computational geometry, neural networks. Dept. of Comp. Sci., Carleton University, Ottawa, Canada. 1992.
- [vG97] Tim van Gelder. The dynamical hypothesis in cognitive science. *British Brain Science*. 1997. Accepted for publication.
- [Wah90] G. Wahba. *Splines Models for Observational Data*, volume 59 of *Series in Applied Mathematics*. SIAM, Philadelphia. 1990.
- [Wer74] Paul J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioural sciences. Master's thesis. Harvard University. 1974.
- [WH60] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. IRE WESCON Convention Record, New York. 1960. IRE: 96-104.
- [Wil85] R. Williams. Feature discovery through error-correcting learning. Report 8501. University of California, Institute of Cognitive Science. 1985.
- [Won79] Murray W. Wonham. *Linear multivariable control: a geometric approach*, volume 10 of *Applications of mathematics*. Springer-Verlag, New York. 1979.
- [WP85] D. L. Waltz. . and J. B. Pollack. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*. 9:51–74. 1985.

- [You93] L. S. You. Tracking control designs for both holonomic and nonholonomic constrained mechanical systems - a unified viewpoint. *INTERNATIONAL JOURNAL OF CONTROL*. 58(3):587–612. 1993.

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved