1

MicroD

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, S.R.C. 1970, c. C-30.

**A Neural Controller For
Collision-Free Movement of Robot Manipulators**

by

Daryl H. Graf

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

August 15, 1988

© copyright

1988, Daryl H. Graf

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse, ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-46285-X

The undersigned recommend to the faculty of Graduate Studies
and Research acceptance of the thesis
A Neural Controller for The Collision-Free movement of Robot Manipulators
submitted by Daryl H. Graf. B.Sc.
in partial fulfilment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Chairman, School of Computer Science

Carleton University

August 15, 1988

ABSTRACT

In this thesis we present a new approach for the collision-free control of robot manipulators moving in a workspace cluttered with obstacles. An adaptive robot arm controller is described that is based on a layered neural network architecture. Starting with no knowledge of the kinematics of the arm, the controller autonomously learns eye-hand coordination and collision avoidance constraints during an initial unsupervised training phase. After training, once the robot has selected a target point by 'glancing' at it with its sensors, the controller automatically generates a sequence of arm movements that position the end-effector near the intended target without touching obstacles. Collision-free movement is produced regardless of the number or arrangement of obstacles in the workspace. Moreover, no additional learning is required if the obstacle set is changed —

only one training phase is needed. The performance of the adaptive controller is analysed, and simulation results are presented. An in-depth survey of the motivation, terminology, concepts and fundamental algorithms associated with neural network computation is also provided.

For Pauline

ACKNOWLEDGEMENTS

I would like to express my heart-felt thanks to Dr. Wilf Lalonde, my thesis advisor, for his encouragement, enthusiasm, sound technical advice and guidance. Most of all I would like to thank him for sharing with me countless enjoyable hours spent exploring minds, brains and the neural basis of behaviour. His curiosity, imagination, and willingness to explore new ideas remain a constant inspiration to me.

I would also like to thank Dr. John Oommen for pointing out the link between neural networks and non-parametric pattern classifiers, and for many interesting discussions regarding Kohonen's self-organizing maps.

Finally, I would like to thank my wife Pauline. Her encouragement and love made this thesis possible. Newlyweds who have gone through the throws of a Master's Thesis together (one does not do this alone) will appreciate the importance of patience, understanding and good humour. Pauline provided me with all of these things, and more. Her organizational, mathematical and proofreading skills were invaluable to the production of this thesis. Pauline, I love you.

TABLE OF CONTENTS

CHAPTER I: Introduction	1
1.1. Organization of the Thesis	2
1.2. Note to the Reader	3
CHAPTER II: A Neural Network Primer	5
2.1. Neural Networks are Machine Learning Systems	5
2.1.1. Symbolic Learning Systems	6
2.1.2. Neural Learning Systems	7
2.2. A Quick Tour of the Brain	14
2.2.1. The Brain Is Constructed From Information Processing Sheets ..	14
2.2.2. The Cellular Level	24
2.2.3. The Sub-Cellular Level	28
2.3. A Simple Model and Notation	30
2.3.1. The Neural Model	31
2.3.2. The Network Model	34
2.3.3. The Functional Level	39
2.4. A Survey of Algorithms and Techniques	39
2.4.1. Arbitrary Pattern Association Algorithms	40
2.4.2. Auto-Association and Pattern Completion Algorithms	62
2.4.3. Structure Mimicry Algorithms	75
2.4.4. Feature Detection Algorithm	85
CHAPTER III: The Robot Control Problem	100
3.1. An Introduction to Robotics	101
3.2. Robotics Terminology	102
3.3. The Manipulator Control Problem	105
3.3.1. The Traditional Approach to Coordinated Movement Control	106

3.4. Is This Really the Best Approach ?	114
3.4.1. The Real-World is a Hostile Place For Robots	114
3.4.2. Nature Has Already Solved These Problems	115
3.4.3. Adaptive Learning Systems for Motor Control.....	116
3.5. Chapter Conclusions	119
CHAPTER IV: A Neural Arm Controller	120
4.1. Arm Control As Simultaneous Satisfaction of Multiple Constraints.....	121
4.1.1. Target Points Can be Expressed as Eye Configuration Vectors ...	121
4.1.2. Target Points Can Also be Expressed as Arm Configuration Vectors	123
4.1.3. Arm Movements Are Curves Through Arm Configuration Space	124
4.1.4. Arm Movement Planning is Equivalent to Finding a Curve.....	126
4.1.5. Initial and Final Physical Configurations Constrain Arm Control Paths	126
4.1.6. Joint Range Characteristics Constrain Arm Control Paths	127
4.1.7. Repeatable Configuration Restrictions Constrain Arm Control Paths.....	130
4.1.8. Obstacles Constrain Arm Control Paths.....	131
4.1.9. Finding Obstacle Regions.....	132
4.1.10. Further Constraints.....	133
4.2. A Neural Controller for Collision-Free Movement of Robot Manipulators..	133
4.2.1. The Robot Architecture.....	134
4.3. The Adaptive Arm Controller Design	138
4.3.1. Finding Constraint Structures is a Learning Machine Problem....	138
4.3.2. The Adaptive Arm Controller Architecture.....	140

4.4. Operation of the AAC	156
4.4.1. The Training Phase	156
4.4.2. The Arm Movement Phase	165
4.4.3. On-Going Adaptation	168
4.4.4. Extending the AAC to Realistic Robots	170
4.5. A Geometric Analysis of the AAC	172
4.5.1. Voronoi Tessellations	173
4.5.2. Voronoi Tessellations and the AAC	174
4.5.3. Mapping Voronoi Tessellations Into the Workspace	179
4.5.4. A Geometric Interpretation of the Inter-Map Associative Laws ...	184
4.5.5. Collision Uncertainty Constrains Path Finding	196
4.5.6. Controlling System Precision	198
4.6. A Simulation of The Adaptive Arm Controller	207
4.6.1. General Approach	207
4.6.2. The User Interface	209
4.6.3. Testing Adaptation to Arbitrary Robot Shapes	248
4.6.4. Example Voronoi Tessellations	256
4.6.5. Learning and The Relationship Between AAC Size and Positioning Error	256
4.6.6. Problems Encountered	265
4.7. Chapter Conclusions	268
CHAPTER V: Directions for Further Research	271
5.1. Reducing the Number of AAC Neurons	271
5.2. Using Optical Technologies for Motor Control	273
5.3. Reconstructing Images from Touch Information	273
5.4. Reaching for Targets Outside the Workspace	274
5.5. Adaptation to Velocity and Acceleration Kinematics	275
REFERENCES:	276

LIST OF FIGURES

Figure 2.1: A Black-Box View of a Neural Learning System	8
Figure 2.2: An Arbitrary Pattern Associator	9
Figure 2.3: An Auto-Associator	10
Figure 2.4: A Regularity Detector	11
Figure 2.5: A Structure Mimic	11
Figure 2.6: The Cortical Tube	15
Figure 2.7: Cortical Sheets	17
Figure 2.8: Simplified Eye-Head Arrangement	22
Figure 2.9: An N-Dimensional Map Can Fill an N+1-Dimensional Space	23
Figure 2.10: A Cortical Hyper-Column	24
Figure 2.11: A Typical Neuron	25
Figure 2.12: Signal Processing Characteristics of a Neuron	27
Figure 2.13: A Simple Model Neuron	32
Figure 2.14: Typical Output Signal Functions	33
Figure 2.15: An Ambiguous Sentence	35
Figure 2.16: Five Important Neural Network Architectures	37
Figure 2.17: Perceptron Architecture	41
Figure 2.18: Bayes' Rule Finds Decision Regions and Boundaries	44
Figure 2.19: A 2-Dimensional Feature Space	46
Figure 2.20: Straight Lines Can Often Approximate Non-Linear Decision Boundaries	46
Figure 2.21: Perceptron Discriminant Functions are Linear	47
Figure 2.22: Firing Thresholds are Analogous to Weights	47
Figure 2.23: A Geometric Interpretation of the Activation Level of a Perceptron Neuron	48
Figure 2.24: Perceptrons Learn by Descending a Criterion Surface	49
Figure 2.25: A Trick to Simplify Analysis	51
Figure 2.26: Non-Linearly Separable Classes Require Multi-Layered Perceptrons	53

Figure 2.27: Every Neuron Must Be Taught	54
Figure 2.28: Back Propagation Architecture	57
Figure 2.29: Hopfield Architecture	63
Figure 2.30: The Energy Landscape of a Hopfield Network	64
Figure 2.31: Boltzmann Machine Architecture	68
Figure 2.32: Local Vs Global Minima	70
Figure 2.33: The Boltzmann Probability Function	71
Figure 2.34: On-Center Off-Surround Lateral Connections	76
Figure 2.35: Stable Bubbles of Activation	78
Figure 2.36: The Topology Preserving Behaviour of the Network	80
Figure 2.37: Weight Vectors Approximate Training Sample Distributions	81
Figure 2.38: Kohonen Architecture	82
Figure 2.39: Functional Level Cognitron Architecture	87
Figure 2.40: Un-Modifiable Intra-Cognitron Connections	88
Figure 2.41: Un-Modifiable Intra-Excitatory Layer Connections	89
Figure 2.42: Modifiable Inter-Cognitron Layer Connections	89
Figure 2.43: Learning is Uniformly Distributed	93
Figure 2.44: Lateral Competition Distributes Learning	94
Figure 2.45: Information Flow Between Excitatory Cells in Adjoining Cognitron Layers	97
Figure 3.1: A Generic Robot	102
Figure 3.2: Joint Types	104
Figure 3.3: Finding the Kinematics of a Horn Robot	109
Figure 3.4: Tsutsumi Arms	117
Figure 4.1: Human Arms have 6 Degrees-Of-Freedom	123
Figure 4.2: Arm Trajectories are Curves in Arm Configuration Space	125
Figure 4.3: In-Range Constraint Regions	128
Figure 4.4: Unrestricted Joint Spaces	129
Figure 4.5: A Repeatable Fault	131

Figure 4.6: Architecture of the Adaptive Robot	135
Figure 4.7: The Observation Map as A Camera Image	142
Figure 4.8: Arm Configuration Map Receptive Fields	144
Figure 4.9: Network Paths in Arm Configuration Space	145
Figure 4.10: The Three Maps of the AAC	150
Figure 4.11: Inter-Map Associative Connections	150
Figure 4.12: The Permanent Constraint Neuron	151
Figure 4.13: Bus Architecture for M_a and M_e	153
Figure 4.14: M_o to M_a Connection Growth	160
Figure 4.15: Continued M_o to M_a Connection Growth	161
Figure 4.16: M_o to M_a Connections Find Configurations That Intersect Obstacles	162
Figure 4.17: M_e to M_a Connection Growth	163
Figure 4.18: Continued M_e to M_a Connection Growth	164
Figure 4.19: The Target Selection Step	166
Figure 4.20: The Constraint Determination Step	167
Figure 4.21: The Collision-Free Movement Step	168
Figure 4.22: Voronoi Tessellations	174
Figure 4.23: Kohonen Competition Generates Voronoi Tessellations	175
Figure 4.24: Non-Uniform M_a Distribution	176
Figure 4.25: The Foveating Configuration Regions of Eye Configuration Space	177
Figure 4.26: $VT(M_a)$ and $VT(M_e)$ Have Toroidal Tessellations	178
Figure 4.27: The $VT(M_o)$ Tessellation is a Grid	179
Figure 4.28: Aspects of the Kinematics of the Robot	180
Figure 4.29: The $VT(M_a)$ to Workspace Mapping	181
Figure 4.30: Characteristic Shape of Mapped $VT(M_a)$ Regions	182
Figure 4.31: Aspects of the $VT(M_e)$ to Workspace Mapping	183
Figure 4.32: The $VT(M_e)$ to Workspace Mapping	184
Figure 4.33: $VT(M_o)$ Represents Obstacle Locations	185

Figure 4.34: $VT(M_a)$ Represents Configurations that Touch a Point	185
Figure 4.35: $VT(M_e)$ Represents Configurations that Foveate a Point	186
Figure 4.36: Connections Grow When Regions of Mapped $VT(M_a)$ and Mapped $VT(M_e)$ Intersect	187
Figure 4.37: Worst Case Number of Mapped $VT(M_a)$ and Mapped $VT(M_e)$ Region Intersections	188
Figure 4.38: Grids of Equal Size Have More Intersections	189
Figure 4.39: How Obstacles Inhibit Arm Configuration Neurons	190
Figure 4.40: Mo to Ma Connections Capture the Avoidance Set of each Arm Configuration Neuron	191
Figure 4.41: Non-Optimal Learning	193
Figure 4.42: The Density of Training Samples Needed to Learn Obstacle Avoidance Constraints	194
Figure 4.43: Some Network Paths Cause Arm Collisions	195
Figure 4.44: Positioning Error	199
Figure 4.45: Worst Case Positioning Error	198
Figure 4.46: Worst Case Avoidance Error	200
Figure 4.47: Worst Case Avoidance Error	201
Figure 4.48: The Adaptive Arm Controller Simulation Window	210
Figure 4.49: Simulation Window Menus	211
Windows 1 through 35	213-247
Figure 4.50: Weight Distribution	250
Figure 4.51: Mapped M_a Weights	251
Figure 4.52: Mapped M_e Weights	252
Figure 4.53: $VT(M_e)$ and $VT(M_a)$	253
Figure 4.54: $VT(M_a)$ Mapped Into The Workspace	254
Figure 4.55: $VT(M_e)$ Mapped Into The Workspace	255
Figure 4.56: Average Positioning Error Vs Trials	258

Figure 4.57: Average Minimum Positioning Error Vs Trials	259
Figure 4.58: Positioning Error Variance Vs Trials	260
Figure 4.59: Minimum Positioning Error Variance Vs Trials	261
Figure 4.60: Actual / Predicted Positioning Error Ratios	262
Figure 4.61: Average Number of M_e to M_a Connections	263
Figure 4.62: Number of Neurons Versus Positioning Error	264
Figure 4.63: Avoidance Error	266

CHAPTER 1: Introduction

One of the characteristics of all animals — including man — that we most often take for granted is a graceful and seemingly effortless control of movement. Men have long dreamed of building machines capable of the same degree of motor skill. Although the fledgeling field of robotics has taken some important steps in this direction, we still have a long road to travel before our artificial systems attain the same degree of flexibility and precision of movement that natural systems exhibit. This level of sophistication will rely less on new material technologies and mechanical designs than it will on better approaches for the design of the control systems that guide the robot and facilitate its interaction with the environment.

Much research has been devoted to finding robot control techniques that are ever faster, cheaper and more precise. Generally speaking, the systems that have resulted from this research are still limited to specialized application domains, and although they are successful within these narrow confines, they are unable to deal with the complexity and demands of even the simplest real-world environments. Part of the problem is that these systems, designed as conventional computer programs, lack an important characteristic that all living, moving beings share — autonomous adaptive learning. As we shall see, learning is an essential ingredient in any robust, general purpose motor control system. Until we can build a robot that can improve its performance with practice, that can learn to accommodate mechanical and sensory changes without human intervention, and that can do these things quickly, we will never achieve the full potential that robotic systems have to offer in all walks of life.

It is the contention of this thesis that there is much to be gained from a study of the most sophisticated motor control system we know — the human brain. Nature has obviously already found solutions to all of the problems that we are now struggling with. It is our belief that a cross-disciplinary exchange of ideas between the neurosciences, cognitive psychology, computer science and robotic engineering will lead to powerful new techniques and profitable directions for future research. Luckily, there is already a body of work that brings these areas together in a study of the computational properties of brain models. The family of distributed, adaptive systems that have resulted from this work are popularly known as neural networks¹. These systems exhibit a variety of interesting computational properties that are a useful addition to the engineer's toolbox.

In this thesis we shall examine a new approach for the collision-free control of robot manipulators moving in a workspace cluttered by obstacles. This approach is based on a layered,

¹ Neural networks are also commonly referred to as connectionist systems, parallel distributed processing models, neuromorphic systems and neuromimes.

neural network architecture that adapts to the sensori-motor kinematics of any arbitrarily shaped robot manipulator during an initial unsupervised training phase. After training, the robot selects a target by 'glancing' at it with sensors, and the controller then generates a joint-interpolated arm trajectory that moves the robot's end-effector into position. This approach is offered as a starting point for further research into adaptive learning systems for collision-free manipulator control². It is intended to be a small step along the road toward flexible robotic systems that are better adapted to operation in the domains we humans take for granted. It has several advantages over traditional solutions to the arm control problem, and extends previous work on neural manipulator control. Limitations of the system are analysed and simulation results are presented.

1.1. Organization of the Thesis

Chapter 2, "A Neural Network Primer", is designed to introduce the general reader to the area of neural computation. This chapter is an in-depth look at the motivation, terminology, concepts and fundamental algorithms associated with neural network computation. It is one of the only comprehensive introductions to the subject currently available³ and will serve the reader well as a foundation for the material presented in this thesis and for further reading. The chapter consists of two parts. The first is a quick review of what is known about the computational architecture of the brain. Readers unfamiliar with neuroanatomy are encouraged to familiarize themselves with the terminology introduced here. It is used throughout the thesis and is common to much of the neural network literature. The second part is a survey of six representative neural network algorithms. These algorithms are presented in a common notation for the first time. Understanding these algorithms is a key to understanding almost all of the important work that has been done in this area over the last two decades. A number of concepts which are important to the thesis are introduced in the survey. Those readers with a foundation in non-parametric pattern classification techniques may wish to skip the first two algorithms.

Chapter 3, "The Robot Control Problem", introduces the general reader to the area of robotics, with special emphasis on position kinematics for the control of robot manipulators. This chapter serves as a prelude to the description of the adaptive arm controller. It identifies issues that are important to manipulator control, gives the reader a flavour for the traditional approach to robot

² This approach is also notable as a successful application of neural computing techniques to a non-trivial problem domain.

³ A good two volume textbook is available on the subject which goes into detail regarding philosophy and general concepts [Rumelhart, et. al., 1986], and several very short surveys have been published which present algorithms with very little background explanation [e.g. Lippmann, 1987]. Almost nothing is currently available to give the interested reader a comprehensive but reasonably short overview of the subject.

manipulator control, and argues for the role of adaptive learning in robot control systems. Readers familiar with robotics can proceed directly to chapter 4.

Chapter 4, "A Neural Controller for Collision-Free Arm Control", is a detailed description of an adaptive neural learning system for robot manipulator control. The chapter is divided into five parts. In the first part, we show that collision-free arm control can be reduced to the problem of finding a curve in a multidimensional space that satisfies a set of kinematic and sensory constraints. In the second part, we show how to construct a neural system to adaptively discover these constraints through a process of unsupervised learning, and — in parallel — employ them to rapidly find collision-free arm trajectories. In the third part, we use a powerful conceptual tool, the Voronoi tessellation, to analyse the learning and operational behaviours of the controller. Sources of system imprecision are investigated, and further requirements are discovered that guarantee collision-free movement of the arm. We derive equations that predict the number of training samples required, and that show the relationship between the number of neurons used to implement the controller and the resulting accuracy of movement. In the fourth part we present the results of extensive simulation studies that support the analysis, and suggest heuristics to increase the accuracy of the controller. Finally, conclusions are presented and the relative advantages and disadvantages of the system are summarized. Those readers already familiar with robotics and neural systems can begin with the second part of this chapter. Readers wishing a quick overview of the system should read the chapter conclusions first, and then skim the second part of the chapter, finishing with a review of the simulation windows in the fourth part.

Chapter 5, "Directions for Further Research", suggests directions for future research that are natural extensions of this work.

The reader will find that [Graf 1988b] contains the complete source code for the simulator described in chapter 4. Readers wishing to duplicate the or extend the experiments described in this thesis may find this a useful resource.

Finally, an extensive bibliography is provided which contains many useful references for readers interested in further investigating the neural network paradigm, and robotics.

1.2. Note to the Reader

The reader will notice that some words and phrases are printed in bold-face text. These represent terms and concepts of special importance that will appear again in this thesis, and which are common to the neural network and robotics literature. The reader should be sure that he/she understands these items before continuing to the next chapter. Italics are occasionally used to give emphasise to important points.

Footnotes are used throughout the thesis to make asides and provide additional references. Footnote reference marks are re-numbered from one at the beginning of each new chapter.

CHAPTER II: A Neural Network Primer

Although neural network computation has recently begun to attract a greater degree of academic and industrial interest, it remains an esoteric field, confined to a small body of researchers. One reason for this is the cross-disciplinary nature of the subject. Neural network computation spans fields as diverse as artificial intelligence [Anderson, 1983; Cottrell & Small, 1983; Gallant, 1987, 1985; Grossberg, 1980; Kosko, 1987; Minsky, 1986; Palm, 1984; Rumelhart, et. al., 1987, 1986b; Touretsky, 1988, 1987; Treurniet, 1986], neurobiology [Baron, 1987; Harth, et. al., 1987; Lara & Arbib, 1985; Grossberg, 1987a; Marr & Poggio, 1976; Nagano & Miyajima, 1983], the theory of computation [Hecht-Nielsen, 1987; Judd, 1987], cognitive psychology [Grossberg, 1987a, 1987b, 1986; Rumelhart & McClelland, 1987; Rumelhart, et. al., 1986b; Zipser, 1984], pattern recognition [Carpenter & Grossberg, 1987; Kohonen, 1986a; Nilsson, 1965; Rosenblatt, 1957, Rumelhart, et. al., 1986b], electronics [Graf, et. al., 1986; Hubbard, et. al, 1986; Mead, 1987; Palm, 1985] and optics [Abu-Mostafa & Psaltis, 1987a; Anderson, 1986; Gindi, et. al., 1987; Gmitro & Gindi, 1987; Kosko, 1988b; Psaltis, 1987a]. This has made it notoriously difficult to find, and keep abreast of, relevant developments in the literature. For historical reasons, many seminal papers in the field have been tucked away in rather obscure journals, and communication between workers in the area has been limited.

This chapter is intended to introduce the general reader to the subject by (1) describing fundamental concepts and definitions, (2) developing a neural network model common to much of the literature, and (3) surveying a collection of algorithms representative of the field. In order to give the reader a sense of perspective, section 2.1 puts neural network research into the context of machine learning systems. Although it cannot be said that there is a general theory of neural network computation, there is a diverse and growing body of algorithms, techniques and tricks that have their origins in neural models. Section 2.2 is a review of the most relevant biological data that has lead to the development of these models. Section 2.3 presents the model and associated notation which will be used throughout the thesis. Finally, a survey of six important algorithms is presented in section 2.4 to give the reader a deeper appreciation of this unusual type of information processing.

2.1. Neural Networks are Machine Learning Systems

Learning is an important component of systems that exhibit intelligent behaviour [Charniak, 1985]. It is not sufficient that an intelligent system work with a fixed set of facts about its world. Conditions within most environments are constantly changing — as new events occur, old facts become outdated. Any system faced with the prospect of interacting with such an environment in

an intelligent way must be able to update its store of knowledge by observing the environment. Those aspects of the environment that the system observes will be referred to as its **learning domain**.

Machine learning is a branch of artificial intelligence which strives to mechanize the learning process and incorporate it into artificial systems. Most often, these systems are computer programs which execute on conventional digital computers. A significant amount of effort has been devoted to the learning problem since the early 1950's and several important advances have been made [Michalski, et. al., 1986; Winston, 1984]. Neural networks can best be described in the framework of machine learning. Ryszard Michalski, an influential investigator in the area, has identified three major paradigms of machine learning research; (1) **symbolic concept acquisition**, (2) **knowledge-intensive, domain-specific learning**, and (3) **neural modeling** [Michalski, et. al., 1986]. These paradigms differ in several important respects: first, in the way that they represent knowledge about the world (i.e. model the environment), second, by the amount of built-in knowledge they require before learning can begin, and finally, in the way that they modify their knowledge representations during learning.

2.1.1. Symbolic Learning Systems

Learning systems based on the first two paradigms are qualitatively quite similar. Knowledge about a particular domain is represented by symbolic data structures — called the system's **knowledge-base** — such as semantic networks, decision trees, production rules or logic expressions [Charniak, et. al., 1987]. This knowledge most often represents high-level concepts. Information about a particular concept is stored in a few well defined, addressable locations in this knowledge-base. For example, characters in a Shakespearian play could be represented as nodes in a semantic network, and the relationships between them by connecting links [Winston, 1984]. Both paradigms are commonly referred to as **symbolic learning systems** because symbols are used to represent facets of the domain.

The learning process may or may not require a teacher. Some systems learn a concept by observing examples and counter-examples of the concept that are presented by a teacher. We will refer to examples supplied by the teacher as **training samples**. This type of **supervised learning system** employs generalization and induction heuristics to refine a data structure that represents the attributes of the concept and the relationships between it and other concepts already known. For example, the system INDUCE [Michalski & Chilausky, 1980] uses hundreds of examples of soybean disease symptoms to produce generalized disease descriptions that are often better for diagnosis than the human generated variety.

Alternatively, the system may be an **unsupervised learning system** which does not require a teacher. This type of system begins the learning process with a knowledge-base that already contains unproven assertions about the domain. It must employ rules of inductive and deductive inference to prove these assertions and discover new concepts on its own. A classic example of this type of system is the program AM [Davis & Lenat, 1982] which discovers new mathematical concepts by applying transformations to a set of initial concepts and then employs interestingness-heuristics to identify which concepts should be further developed.

In either case, symbolic learning systems start the learning process with a relatively large amount of knowledge. Some of this knowledge consists of procedural rules that operate on the knowledge-base to find useful generalizations and deduce new truths. Other forms of knowledge represent the minimal set of concepts and relationships that new concepts can be built upon. The domain specific nature of this knowledge often makes it impossible to apply the same learning system to different learning domains. This problem, called **brittleness**, is typical of many symbolic learning systems and is an active area of machine learning research. Once learning is complete, the system is left with a knowledge-base that it can manipulate to answer questions about the environment or formulate plans. The symbolic nature of the knowledge-base makes it possible to pick up knowledge about a particular concept and easily incorporate it into similar systems¹.

2.1.2. Neural Learning Systems

The neural modeling paradigm takes its inspiration from the neurosciences. Learning is an important survival characteristic. Consequently, a wide range of organisms have evolved with varying degrees of learning ability. The most sophisticated example of an organic learning system is the human brain. Clearly, the very abilities that we wish to add to our machines have already successfully been dealt with by mother nature! It is for this reason that the neural modeling paradigm strives to understand the design principles employed in the brain in order to borrow techniques for application in artificial systems. As all organic nervous systems are constructed from very large numbers of densely interconnected information processing cells called neurons, we often refer to artificial systems based on these techniques as **neural learning systems**, **neural networks** or **connectionist systems**².

¹ Transporting concept knowledge is not as easy as it might seem at first glance, but the point being made here is that it is at least theoretically possible.

² Strictly speaking, Michalsky also includes genetic algorithms in the neural modelling paradigm. Genetic systems take their inspiration from the evolutionary process. The goal of a genetic algorithm is to evolve (learn) a data structure or expression which is well adapted to solving a particular problem. The component parts of this structure are subject to mutation rules, and the survival of mutant structures is governed by a criterion function which supplies selection pressure. Such systems have demonstrated very good learning performance on non-trivial learning problems [Ackley, 1987].

In the simplest analysis, a neural learning system can be viewed as a black box (Figure 2.1). We may think of the box as a filter that transforms input signals it receives — from sensors that observe the learning domain — into appropriate output signals. The input and output signals are simply numeric patterns that do not have any underlying meaning from the system's point of view. All that is important to a neural system is that regularities that exist in the domain being observed are represented in these patterns [Stone, 1986]. For example, the presence or absence of an observable object might be represented by the two states of a bit in a binary string. There are many — possibly millions — of parameters which specify the exact input to output transformation that takes place. The pattern transformation that the black-box performs is anthropomorphically referred to as its **behaviour**. Unlike symbolic learning systems which learn by building symbolic descriptions of the domain, neural systems learn by adjusting the input to output transformation of the black-box such that the behaviour that results parallels some aspect of the learning domain's behaviour. There are three paradigms of neural learning which result: (1) **associative learning**, (2) **regularity discovery**, and (3) **structure mimicry**.

A Black-Box View of a Neural Learning System

Figure 2.1

In the associative learning paradigm, the neural system learns to associate observed input patterns with output patterns. The first type of associative learning, called **arbitrary pattern association**, is depicted by Figure 2.1a. Here, sensors provide the system with two sets of input channels. These channels represent different, though related, aspects of the learning domain. For example, the first set might represent the visual pattern produced by a flower, while the second set represents its odour. When events take place in the environment which *consistently* cause the patterns to appear together, the system learns to transform the first pattern into the second. In future, when the system detects the first pattern, it will reproduce the second pattern as output. This sort of transformation is often viewed as a kind of 'reminding' or 'predictive' behaviour and can be used as such in an appropriately designed system [Mozier, 1986]. Pattern classification is a

special case of arbitrary pattern association in which the first pattern represents features of an object, and the second pattern represents a descriptor of the object such as its name or ID number³.

Figure 2.2 An Arbitrary Pattern Associator

The issue of consistency of association is important and deserves further elaboration. Real world environments rarely provide perfectly consistent pattern associations. Deviations may occur due to uncertainty, noise, or limitations of the sensory equipment. For example, although patterns A and B are associated 95% of the time, there may be rare occasions when they are not. We do not want these few cases to prevent the system from learning the A to B association. A related problem occurs when the underlying structure of the learning domain allows A to be legally associated with more than one pattern. For example, A might be consistently associated with B 20% of the time, C 70% of the time and D 10% of the time. Some arbitrary pattern associators handle both problems by reproducing the second pattern with the same frequency that it appears in the domain [Hinton & Sejnowski, 1985]. Others ignore the lower frequency associations and simply learn the highest frequency association [Rumelhart, et. al., 1986a]. The details of the underlying learning mechanism determine the behaviour of the system.

The second type of associative learning is called **auto-association** (Figure 2.3). In this arrangement, sensory inputs form a single pool of input channels. During learning, the system is presented with a sequence of patterns that represent observed environmental events. After learning, when an incomplete or corrupted pattern is presented, the system will output the pattern observed during learning that is 'most similar' to the incomplete pattern. The metric used to determine similarity between patterns is a function of the underlying learning mechanism and varies from system to system. However, it often resembles the Hamming distance metric [Lippmann, 1987]. This **pattern completion** behaviour is often employed as a form of associative memory

³ Michalski refers to the neural learning paradigm as 'neural modelling and decision theoretic techniques'. The decision theoretic techniques he refers to are an important aspect of the field of statistical and non-parametric pattern classification. There is a strong theoretical and historical relationship between Bayes decision theory and neural learning systems [Anderson, 1987]. Section 2.4.1.1. illustrates this relationship in terms of a neural network called the perceptron.

[Hopfield, 1982]. Patterns observed by the system consist of the information to be remembered and a unique recall key. Later, corrupted or incomplete versions of the key can be used to recall the rest of the information. For example, an early auto-associative memory developed by Teuvo Kohonen stored several hundred digitized pictures of faces [Kohonen, et. al., 1976]. This memory employed facial features (eyes, mouths, etc) as the recall keys to quickly retrieve the appropriate face.

Figure 2.3 An Auto-Associator

In the regularity detection paradigm, the system learns which features of the learning domain occur most frequently and are therefore the most 'interesting' or useful. The system becomes selectively sensitized to one of these features and will output a signal only when the feature is present in the observable environment. For example, a regularity detector observing a random sequence of printed letters might become sensitized to vertical, horizontal or diagonal line segments of the appropriate length that consistently occur as a component feature of many letters (see Figure 2.4). Hierarchies of such detectors can be arranged so that outputs of one level become the inputs of the next. Systems built along these lines can become sensitized to ever more complex and abstract combinations of underlying domain features [Fukushima, et. al., 1983]. The important feature of regularity detection is the ability to learn what is important about the domain and to ignore irregular — randomly occurring — features. We shall refer to these irregular features as noise. The learning mechanism employed for regularity detection has an associated evaluation function that ultimately determines the precise meaning of 'regularity', and the sensors used to collect data about the environment determine which features are actually considered. Initial conditions, internal to the learning system and usually random in nature, ultimately decide which of the many regular features the system will become sensitive to.

Figure 2.4 A Regularity Detector

The final neural modeling paradigm, **structure mimicry**, arranges the *parameters* that control system behaviour such that the structure of this arrangement mimics the structure of the learning domain as shown in Figure 2.5. This type of system requires that the parameters are all of the same type, and can be represented as vectors within a special **parameter space**. Initially, the parameter vectors are randomized. As the system observes the domain, structure inherent in the training data is used to re-arrange the parameter vectors in parameter space. For example, the local point density of the parameter vector distribution might mimic the distribution of the training patterns in a domain feature space. The resulting parametric structure ties the behaviour of the system to the structure of that part of the domain being mimicked. The precise relationship between parameters and behaviour varies from system to system. As we shall see, structure mimicry can be a powerful learning technique that gives the neural system a low-level — but deep — 'understanding' of the structures that control events within the domain.

Figure 2.5 A Structure Mimic

Typically, the **pattern-level learning** performed by neural systems is much lower-level than the **conceptual-level learning** performed by symbolic systems. Symbolic learning makes effective use of the pre-determined *meaning* of training data, whereas neural learning is primarily concerned with the *structure of correlation* between input and output patterns. Meaning is given to this structure only by the *type of feature* that the sensors respond to, and the *type of action* that occurs in response to output signals. It has been argued however, that meaning is also inherent in the isomorphisms that occur between conceptual level events and events that occur within the neural system black-box during the input to output transformation process [Smolensky, 1986]. In any event, most neural learning system research to date has concentrated on sub-cognitive learning tasks, such as low-level vision, speech analysis, simple sentence structure classification and simple motor movements⁴. On the other hand, this freedom from underlying meaning allows us to design neural learning systems that are far less domain specific and brittle than most symbolic learning systems. As far as the neural system is concerned, the pattern that it observes may equally well represent the temperature of an exothermic reaction or the first bar of a Chopin sonata. Whether what it learns about the patterns is useful or meaningful is determined by how well the system designer makes use of the resulting behaviour of the network. Using the former system to control a chemical plant lends meaning to the system's behaviour whereas using the second system for this purpose is likely less meaningful.

The underlying mechanism that relates system parameters to system behaviour may vary somewhat across implementations. However it is always essentially non-symbolic in nature⁵. By this we mean that there is no single token or parameter that represents a particular pattern or behaviour. Instead, a distributed and active form of representation is used in which every parameter contributes equally to every behaviour. Also, no single parameter is essential to the qualitative behaviour of the system. Removing a random 10% selection of the data from a symbolic learning system knowledgebase will invariably have drastic effects on the performance of the system, whereas removing a similar number of control parameters from a neural learning system usually has little effect on the systems behaviour⁶. Unlike conventional symbolic learning systems

⁴ This is rapidly beginning to change however. A number of new research programs have been started to investigate the use of neural networks in conceptual level learning tasks [e.g. Touretsky, 1988].

⁵ This is not to say that neural mechanisms cannot in turn be represented by a symbolic data structure. Indeed, this is commonly done when writing a digital computer simulation of a neural learning system. However, the encoding of knowledge into such a system is essentially non-symbolic [McClelland, et. al., 1986; Touretsky & Hinton, 1985].

⁶ The detailed behaviour of the system will change. Associations between a few patterns may be lost, sensitization to features may be impaired and the fidelity of structure mimicry may be reduced, but the overall quality of this information will be retained. This cannot be said for symbolic learning systems.

which make use of tokens, schemata, strings and pointers, neural control parameters are always real-valued numbers. A disadvantage of the non-symbolic approach is that the contents of the knowledgebase cannot easily be understood by a casual human observer. Finally, this opacity combined with the highly distributed nature of the knowledgebase makes it very difficult to remove a particular association and transport it to a similar system⁷. However once a neural system has learned, its entire knowledgebase can be duplicated as many times as desired.

Another characteristic of neural learning systems is the small amount of initial knowledge that is required before learning can begin. Normally, the system parameters are randomized before learning. In fact, this is absolutely necessary in some systems to overcome a problem known as **symmetry breaking** which we shall consider shortly. Other systems require that some of the system parameters begin with values in a pre-specified range. Apart from this, nothing further is required⁸.

As we have seen, neural learning systems reduce learning to the problem of finding a set of parameters that cause desired behaviour. Given the large number of control parameters, this may seem like a monumental task. However, neural learning systems take advantage of adaptive, distributed algorithms to simplify the problem. A simple adaptive learning law is distributed over a network of neuron-like devices, typically independent processors. Each processor is associated with several of the system control parameters, and it combines these with locally available information, repeatedly performing the same calculation. After each calculation, the result is used to modify its parameters. The end result is that initially random behaviour converges to desired behaviour as the system observes the environment. Although the local adaptive learning law is very simple, interactions among interconnected processors can lead to highly complex, global behaviour. This complex behaviour is referred to as the **emergent behaviour** of the system in the sense that it is not explicitly programmed into any part of the knowledgebase, but rather, emerges from processor interactions. In the early days of neural learning research, emergent behaviour was not well understood and was regarded with a certain degree of awe and mystery [Rosenblatt, 1961]. Since then, research into distributed algorithms and the mathematics of dynamical systems [Cohen, 1983; Duda & Hart, 1973; Hirsh, 1988, 1987] has dispelled this

⁷ This is not true of some very simple neural learning systems like those described in [Hinton, et. al., 1986] and [Jordan, 1986].

⁸ In a sense, the designer encodes a certain degree of domain knowledge into the system when he selects sensors to observe only those aspects of the environment that are relevant to the learning task. This function is akin to the evolutionary development of appropriate sensory systems in animals. This type of encoding seems slight when compared with the selection of just-so heuristics and initial concepts that often precedes a symbolic learning task.

mystique, leading to a much clearer and more realistic view of the advantages and disadvantages of neural learning systems.

The last characteristic that distinguishes neural learning systems from symbolic learning systems concerns speed of processing. Currently, most neural learning systems are written as high-level language simulations that are executed by a traditional von Neuman computer. When implemented this way neural learning methods tend to be very slow because many control parameters require modification and the learning laws are incremental in nature. However, the distributed nature of neural knowledge bases makes it very easy to implement neural learning systems on massively parallel hardware [Abu-Mostafa & Psaltis, 1987; Graf, et. al., 1986; Mead, 1987]. Implemented in this way neural learning systems hold the promise of being exceptionally fast.

2.2. A Quick Tour of the Brain

We will now introduce the fundamental terminology and concepts associated with neural learning systems by examining their chief source of inspiration — the human brain. This will be an extremely simplified view of brain anatomy, concentrating on those features that are relevant to the algorithms and architectures described in this thesis. A quick tour of the brain will also help put the current collection of neural models into proper perspective relative to this remarkably complex organic computing device.

When one first examines the roughly spherical, cauliflower textured human brain, it is tempting to think of it as an undifferentiated mass of cells which comprises a single organ of cognition. This is far from the truth however, as a closer look reveals that the brain is not one organ, but, in fact, is a highly ordered society of hundreds of specialized, intricately structured and interconnected information processing organs. To make matters even more complicated, we have not one, but rather, two mirror image sets of these organs which are arranged in hemispherical packages on either side of our head. A closer look at the cellular makeup of these organs and their interconnections reveals a whole taxonomy of cell types and functions, most of which are still poorly understood. Let us proceed by identifying the simple underlying architectural and organizational principles upon which the system is built.

2.2.1. The Brain Is Constructed From Information Processing Sheets

At the highest structural level, the brain is essentially a hollow tube filled with information processing organs that have been crumpled into the tiny space afforded by our skulls. This arrangement is schematically illustrated by Figure 2.6. This crumpling gives the brain its distinctive convoluted surface texture. The tube is composed of the brain structures commonly referred to as the cortex and cerebellum though we shall refer to it as the **cortical tube**. It is differentiated into

right and left halves by an imaginary plane which passes through the head from top to bottom and from front to back. Each half of the cortical tube contains myriad brain structures such as the hypothalamic nuclei, the limbic system, the basil ganglia and most of the brainstem. Each of these structures⁹, which we shall call **sub-cortical nuclei**, is precisely duplicated and arranged in mirror image on either side of the cortical tube. If we were to slice the tube into its two symmetrical halves, we would discover that only the first 2 mm layer consists of information processing cells. The rest of the volume of the cortical tube not taken up by sub-cortical nuclei is either empty space or is filled with the biological equivalent of wires¹⁰ that connect cells in the tube to cells in the nuclei and other points in the body.

Figure 2.6 The Cortical-Tube

If we now flatten the two halves of the cortical tube, we will be left with two pancakes of brain tissue each roughly 1000 cm² in area, 18 cm in radius and about 2 mm thick (Figure 2.7a). Let us refer to these as **cortical sheets**. In general, each sub-cortical nucleus also consists of one or more sheets of cells which are either folded over on themselves or are also arranged in a tube. If

⁹ Except the pineal gland [Roberts, 1987].

¹⁰ The wires are single, incredibly long (one meter or more) nerve cells which are wrapped in fatty, neurolemma (myelin sheath) cells for insulation - the mind boggles.

we similarly unfold and flatten these nuclei, we will form a series of satellite pancakes, called **sub-cortical sheets**, that surround the cortical sheet (Figure 2.7b). Very roughly speaking, in humans the cortical sheet is concerned primarily with sensory processing, object recognition, sensori-motor integration, movement planning and abstract thought. From an evolutionary viewpoint, it is a relatively new development in brain architecture that is most developed in man. The sub-cortical sheets are all phylogenetically much older, and are primarily concerned with processing emotion, the basic drives of hunger, thirst and sex, cognitive map making, autonomic reflex, and the control of memory.

Connecting wires are mostly grouped into bundles called **tracts** or **capsules**. Inputs from sensors (e.g. visual, auditory, tactile, chemical and motor-feedback) arrive via a series of preprocessing stations collectively known as the **thalamus**¹¹ where they are separated, enhanced and connected to specific regions within the cortical and sub-cortical sheets as shown in Figure 2.7c. Connections between cortical and sub-cortical sheets on the same side of the brain also mainly pass through the thalamus, although connections between sheets on opposite sides of the brain are mostly made by a massive tract called the **corpus callosum**. Connections between regions of a cortical sheet tend to be direct connections that do not pass through the thalamus (Figure 2.7d). It is thought that the thalamus plays an important role in attention because of its strategic location at the center of information flow [Crick, 1984; Noback & Demarest, 1981; Palm, 1984]. For some reason, almost all thalamic connections involve circular feedback connections. Many theories have been advanced to explain this [Crick & Asanuma, 1986], but there is still little agreement as to the fundamental need for feedback at this architectural level. Outputs from the sheets normally do not pass through the thalamus but directly connect to organs and muscles in the body. At this level of detail, all connections — whether I/O related or between regions on a sheet — are very specific pathways that appear to be coded for genetically. It is only at the cellular level that a high degree of variation exists between individuals. Experience-driven, self-organizing laws seem to play an important part in determining the specific connections that are made after birth [Kohonen, 1984c; von der Malsberg, 1973]. For the sake of completeness, we should also mention the odd fact that processing for the left half of the body and sensory fields is performed by the right set of cortical and sub-cortical sheets and vice versa. It is not understood why this should be so, but it is commonly assumed that it is a currently unimportant genetic holdover from an earlier stage of evolutionary development.

¹¹ The single exception to this rule is the olfactory bulb. Smell is a phylogenetically ancient sensory modality which probably developed prior to the thalamus.

Figure 2.7 Cortical Sheets

The cortical and sub-cortical sheets are not monolithic blocks of cells. Instead, we find several interacting levels of structure.

2.2.1.1. The Cortical Sheet is Divided into Regions of Specialization

Some sheets are actually composed of many distinct regions, each of which specializes in processing a particular type of information. In the section of the cortical sheet known as the neocortex, there are somewhere between 50 and 100 regions called **Broadman areas**. These areas have slightly different cellular architectures, although the overall cell structure is consistent from area to area. It has been suggested that all Broadman areas share a common, underlying information processing mechanism which nature has refined to more efficiently handle each type of specialized computation. Some of these areas, called **primary sensory areas** process specific sensory modalities like vision and hearing. Connections from the thalamus terminate directly on these primary sensory areas. The regions adjacent to or surrounding the primary sensory areas are divided into **secondary and tertiary sensory areas**. Cells from these areas take their inputs from a specific primary sensory area and appear to become sensitive to higher levels of data abstraction. For example, in the primary visual system (Broadman area 17), cells process relatively low-level data concerning illumination, colour and movement fields. Cells in the secondary and tertiary visual areas seem to process information about more abstract shapes such as geometric figures, and commonly experienced objects (hands, cups, symbols, etc). Other Broadman areas, called **motor areas**, are concerned with planning muscle movements. Much less is known about the motor areas than the sensory areas. Finally, the remaining area of the neocortex seems to be divided into **association areas**. Cells within these areas take their inputs from a mixture of other sensory and motor areas and appear to become sensitive to inter-relationships between combinations of sensory and motor experiences. For example, the frontal association areas have been shown by P.E.T. scan studies to contain semantic data structures that link visual recognition cells for written words (secondary visual area), and auditory recognition cells for spoken words (secondary auditory area) to cells in the motor area that encode the sequence of vocal tract movements that allow the listener to speak the word.

2.2.1.2. Broadman Areas are Topographically Ordered

Another common feature of many processing regions is a tendency for the physical arrangement of cells within the region to mimic the topology of the input or output space that they are sensitive to. This preservation of the topology of the space in the network is called **topographical ordering**. For example, cells in the primary auditory area (Broadman area 41) are sensitive to different values of the pitch of sounds. Remarkably enough, the layout of these cells is a perfect representation of pitch space. As we move from one end of the area to the other, cells respond to higher and higher values of pitch in a continuous way. Any cell in the area can be related to its sensitivity by its relative position along the pitch axis. Another example is the primary

somatic — touch — area (Broadman areas 1, 2 and 3) in which the arrangement of cells sensitive to touch at particular points on the body are arranged as a map of the human body, with the head at one end and the feet at the other. Cells of the primary visual area are arranged as a two-dimensional projection of the visual scene onto the cortical sheet. Primary motor area (Broadman area 4) cells which cause muscle movements are arranged according to the relative location within the body of the muscle groups they control. The bat cortex has been shown to contain a two dimensional map of a space representing distance versus cross-sectional area of targets located by sonar. It has also been suggested that human association areas preserve the topology of cognitive spaces, mapping cells to spacial representations of the data structures they encode (trees, linear lists, etc) but no empirical evidence yet exists to support this notion [Kohonen, 1984c].

2.2.1.3. Broadman Areas are Organized into Bands of Specialization

A further level of processing specialization occurs in Broadman areas. When a sensory modality can be divided into a number of related feature spaces, bands of cells¹² develop which are particularly sensitive to one type of feature. For example, signals representing data about a particular point in the visual field are received from the retinas by way of the lateral geniculate nuclei in the thalamus. These signals can be divided into information about the left and right eye views of this point. Furthermore, within these stereoscopic divisions, some signals will encode information about illumination, others about colour, and still others about movement at that point. Cells within the primary visual area become sensitive to select types of signals.

These cells are not randomly intermixed within the Broadman area. Rather, they are arranged in adjacent, or overlapping bands. Adjacent bands seem to encode mutually exclusive or competitive data. For example, bands that encode right eye views and left eye views are adjacent. Overlapping bands, on the other hand, encode data that can be mixed. Bands that encode colour may overlap bands that encode movement. Bands for joint movement, pressure and bending of hairs have been shown to exist in the primary somatic sensory area and bands for right ear-left ear signals exist in the primary auditory area. Other evidence is accumulating for similar bands within the primary motor areas, the sensory parietal areas and several higher visual areas.

2.2.1.4. Information Processing Sheets are Layered

Cortical and sub-cortical sheets are invariably divided into two or more well defined layers of cells. These layers are distinguished by the number and type of cells that they contain. For example, the neocortex may be conveniently divided into four layers; a deep layer (on the inside of the cortical tube), a middle layer, an upper layer, and a superficial layer (on the outside of the

¹² Rather reminiscent of zebra-stripes or some of the patterns that result from reaction diffusion equations.

cortical tube)¹³. The number of layers within the sub-cortical sheets varies somewhat from nuclei to nuclei. For example, the superior colliculus is divided into seven layers while the lateral geniculate nucleus is divided into two functional groupings of three layers [Hubel & Wiesel, 1986; Noback & Demarest, 1981; Sparks & Jay, 1987]. Although the available experimental data is rather sketchy, layering of the sheets seems to occur for several reasons; (1) to physically separate different types of data, (2) to generate levels of data abstraction, and (3) to provide associations between events in different sensory spaces.

2.2.1.5. Layers as Data Separators

The lateral geniculate nucleus (LGN), a preprocessing station for visual data within the thalamus, provides a good example of the use of layering to separate different types of data. Data arrives at the LGN from both retinas, but before it can be relayed to the primary visual area where left and right eye data are segregated into adjacent bands of cells, signals from left and right retinas must be separated. To this end, the LGN projects the left eye data into three layers and the right eye data into three layers. Further separation based on movement, colour and illumination may take place within the three layers [Livingstone, 1988]. Connections then project from each of these layers to the appropriate bands in the primary visual area.

2.2.1.6. Layers as Data Abstractors

An example of successive layers that encode levels of data abstraction occurs in the primary visual area. Cells in the middle layer of this area — called **simple cells** — receive illumination signals directly from cells in the LGN. The LGN cells send a signal whenever a spot of light (or darkness) appears at a particular point within the visual field. Simple cells combine several of these signals to become sensitive to a slightly more abstract pattern: short bands of light or darkness that appear at a specific location within the visual field and are oriented at a specific angle. Outputs from the simple cells in the middle layer project to **complex cells** in the upper layer. These complex cells in turn are sensitive to slightly more abstract shapes; bands of a particular orientation that may appear in a range of locations in the visual field. The next stage of abstraction seems to occur when outputs from complex cells leave the primary visual area via the superficial layer and terminate on cells in the secondary and tertiary areas as discussed in section 2.2.1.1. The cells at the end of a chain of abstraction are often referred to as **grandmother cells**¹⁴. Data abstraction by cellular layers is a commonly used technique in the brain.

¹³ For historical reasons most neurophysiologists refer to six layers of cells, but it appears that more than four layers occur because of differentiation of the basic four.

¹⁴ The idea being that we must have a single cell somewhere that outputs a signal when we see grandma — a very high level of abstraction indeed. There is no direct evidence for this sort of thing, although cells have been found in

2.2.1.7. Layers as Inter-Spatial Associators

The superior colliculus is a small structure that appears to be primarily responsible for translating sensory signals into unconscious muscle movements. When an unexpected movement, touch or sound occurs, the superior colliculus sends signals to your eye and neck muscles to make your eyes focus on the source of the disturbance. This process is called **startle reflex foveation**.

In mammals, the superior colliculus is composed of seven layers. Neuroanatomists have divided these into two functional divisions: the superficial layers and the deep layers. The superficial layers receive visual data from the retina and primary visual area. This visual data is topographically ordered, forming small, 2-dimensional, real-time projections of the visual scene. The deep layers receive inputs from the primary auditory area, the primary somatosensory area and muscle feedback data from the eye and neck muscles. Topographic maps are formed which represent the stereophonic sound field surrounding the head, the skin surface of the body and the current position of the combined eye and neck muscles. This last map encodes what we shall call a **configuration space**. Each point in this space represents a configuration of eyes and head that foveates on a particular point in the real world. To make this important concept perfectly clear, consider the simplified eye-head arrangement in Figure 2.8a. In this simple, two dimensional example, contraction of a single neck muscle determines the orientation of the head, and contraction of a similar muscle determines the orientation of the eye. Figure 2.8b depicts the associated two dimensional configuration space. Two points A and B in this space represent combinations of muscle contractions that foveate the eye on two different points (Figure 2.8c). In the case of systems with more than two muscles, the system is either represented as several two dimensional configuration space maps for muscle pairs, or a single map that somehow represents all points in a higher dimensional space. Figure 2.9 shows how a two dimensional map can be made to fill a higher dimensional space in such a way that cells in the map are sensitive to a representative sample of configurations.

the monkey cortex that fire whenever the outline of a monkey's hand is in the monkey's field of view [Hubel and Wiesel, 1986]. These cells are invariant to rotation, translation and scaling of the hand, and are very specific to hand-like shapes.

Figure 2.8 Simplified Eye-Head Arrangement

It has been shown [Sparks & Jay, 1987] that stimulation of a cell within the ~~deep~~ configuration space map causes muscle contractions that appropriately configure the head and eyes. A mechanism that is not currently understood associates cells in the visual and auditory field maps with cells in the configuration space map that cause foveation at the appropriate point in space. For example, it has been shown that stimulation of visual field and auditory field cells both cause

corresponding excitation of configuration space cells that foveate the eyes on the source of the disturbance. Consequently, when a startling event takes place, the superior colliculus is able to make us look at it in surprise¹⁵. The mechanism that causes associations to form between cells in neighbouring maps is unknown, however it has been suggested that chemical signals cause connection growth between the layers. This mechanism would account for the need to stack the layers near one another. We shall refer to this process as **inter-spacial association**. There is ample evidence that inter-spacial association occurs regularly throughout the brain and appears to be an important computational mechanism. Whenever there is a specialized problem to solve, nature creates a stack of topographically ordered layers that represent spaces important to the solution of the problem.

A layer of neurons can fill a higher dimensional configuration space by 'folding' into it as a space filling surface.

Figure 2.9
An N-Dimensional Map Can Fill an N+1-Dimensional Space

2.2.1.8. Multiple Levels of Structure Result in Hyper-Columns

The collection of cells that lies at the intersection of two or more bands of specialization are called a **hyper-column** (Figure 2.10). The volume of intersection is normally about 1 mm² in cross-sectional area and about 2 mm deep. All of the cells in this multi-layered, roughly cylindrical structure are sensitive to approximately the same specialized set of sensory features. Furthermore, the topographical ordering of the area ensures that all cells in the hypercolumn represent these

¹⁵ This is a simplification as the superior colliculus is only one part of a much more complicated startle-reflex system.

features for the same point in the area's sensory, motor or association space. Neighbouring hyper-columns represent other combinations of features for the same point in space. In this way, every point in the area's space of specialization is represented by hyper-columns that are sensitive to a range of features that might exist at that point. This, combined with the fact that cells within a hyper-column connect to each other in preference to neighbouring hypercolumns, suggests that the hyper-column is the modular building block from which all cortical sheet recognition and cognition structures are built.

Part of the Cortical Sheet

Figure 2.10 A Cortical Hyper-Column

Further levels of organization are currently unknown.

2.2.2. The Cellular Level

Let us now have a closer look at the cellular substrate from which all of the preceding structures are formed. The total number of cells in the cortical and sub-cortical sheets is on the order of several trillion! Most of these cells are not actually involved in information processing, however. Cells which do process information are called **neurons**. The remaining cells, called **glial cells**, provide neurons with a host of support functions — nourishing them, carrying away

their wastes, and insulating them from each other. There may be up to 10 to 50 times as many glial cells as there are neurons. We are only interested in the brain's information processing capability and shall therefore concentrate on the neurons.

There are a total of about 100 billion neurons in the brain, equally divided between the two mirror image sets of cortical and sub-cortical sheets. One cortical sheet is composed of about 20 billion neurons. On average, each Broadman area contains several hundred million cells, each hyper-column contains an average of 100,000 cells and each hyper-column layer contains about 25,000 cells. The remaining sub-cortical sheets share approximately 30 billion cells in amounts ranging from tens of millions to one billion cells.

A Typical Neuron

Figure 2.11 A Typical Neuron

Although there may be several hundred types of neurons — the exact number is unknown — almost all neurons share a number of distinctive features. Figure 2.11a depicts a 'generic' neuron. It is not surprising that many of these features are similar to those found in man-made information processing devices. Input ports called **dendrites** carry information into the neural body, or **soma**, where data is processed. A single output channel called an **axon** extends from the soma for some distance, and then divides into a number of output ports called **axon collaterals** that make contact with the dendrites of other neurons. Some neurons have a slightly different output arrangement in which many short axons extend from the soma instead of a single long axon. The connecting point where an axon collateral meets a dendrite is actually a specialized structure called a

synapse or synaptic bud. Neurons sometimes make synaptic connections with the somas or axons of other cells but the majority of inter-neuron connections occur at dendrites.

The exact arrangement of neurons varies considerably between each type of information processing sheet. There is also a large amount of variation between different layers in the same sheet. For example, in the neocortex, the deep layer contains large-bodied pyramidal neurons whose axons leave the cortical sheet for other areas of the brain, the middle layer is densely packed with stellate neurons whose axons extend into the upper layer, the upper layer contains small-bodied pyramidal neurons which send their axons to other areas of the cortical sheet, and the superficial layer contains mostly axons and dendrites from the lower layers. Architectural differences between Brodman areas are mostly due to variation in the number of cells within each layer, but may also involve the differentiation of one layer into two or more sub-layers containing similar types of neurons.

Most neurons within a sheet connect to their immediate neighbours, forming local circuits. Those that do make long distance connections occasionally traverse the entire diameter of the brain. Local connections, within a layer, appear to be largely random, however, long distance connections are generally not. As described in section 2.2.1.1., regions of specialization connect to a specific set of other regions, and there is some evidence that these connections are ordered in some way. A typical neuron will connect to as many as 10,000 of its neighbours. The total number of synaptic connections in the brain is on the order of 10^{15} ! Although neurons continually grow and absorb synapses, there is little evidence to suggest that significant axonal growth occurs after the first five years of infancy. Indeed, neurons do not regenerate and the total number of neurons gradually declines in the adult human as cells die and are not replaced. It seems that, except for rare instances, we must make use of the neural machinery that we are born with.

The information processing behaviour of a typical neuron appears to be relatively simple. At the moment, the accepted view is that a neuron is not a general purpose computer. Instead, it is like an electronic component that exhibits a mix of analog and digital properties. Communication between neurons is most often in the form of trains of rapid changes in the electrical potential of the cell membrane, called **spikes** or **action potentials** as shown in Figure 2.12a. Early researchers thought that information was encoded by the temporal pattern of spikes, but today this notion is generally regarded as false¹⁶. Spiking appears to be the neural approach to dealing with attenuation of signals over long distances. Indeed, some short distance axons transmit smoothly varying potentials instead of spike trains. The membrane of the neural soma appears to time average the

¹⁶ Although there are still a few researchers who claim to find statistical correlations between temporal spike train patterns and certain cognitive behaviours.

inputs that it receives from its dendrites. The resulting membrane potential is called the **activation level** of the cell. A slowly changing characteristic of the cell, called the **dynamic firing threshold**, is generated within the soma or axon of the cell by a poorly understood process. Once the activation level of the cell crosses this firing threshold, the neuron will output its own signal, in the form of a spike train or graded action potential. For this reason, neurons are often called **thresholding devices**, and they are said to **fire** when they output a signal. The amplitude of the output signal tends to remain constant. The frequency of the signal is related to the difference between the activation level and firing threshold. This relationship is poorly understood, but it appears to take the form of a non-linear **sigmoidal function** as shown in Figure 2.12b. Below the firing threshold, small activation levels do not produce an output signal, thus giving the neuron a certain immunity to noise. Near the firing threshold, the output frequency quickly increases. Activation levels beyond the firing threshold soon reach a limiting output frequency. This limitation, called the **latency period** of the cell, is imposed by the physical chemistry of the neural membrane. The output frequency of most neurons does not exceed 1,000 to 2,000 spikes per second.

(a) Flow of Information in A Neuron

(b) A Sigmoidal Activation Function

Figure 2.12 Signal Processing Characteristics of a Neuron

A surprising characteristic of neurons is their very slow computing speed. A neuron may take anywhere between 1 and 5 milliseconds to respond to an input signal with its own output signal. This is six orders of magnitude slower than the switching rates common to most man-made digital components. And yet, human brains are capable of performing certain types of computational tasks orders of magnitude more quickly than conventional digital computers. We will return to this unusual fact again in section 2.3.2.

2.2.3. The Sub-Cellular Level

Let us now consider the details of the most important neural structure — the synapse. Figure 2.11 depicts a typical synapse associated with a generic neuron. The cell that grows the synapse on its axon is called the **pre-synaptic cell**. The cell that receives the connection at its dendrite is called the **post-synaptic cell**. There is always a small gap of about 20 to 30 nm between the synapse and the post-synaptic cell. This gap is known as the **synaptic junction**. The synapse is filled with sacs of a chemical called a **neurotransmitter**. When an output signal propagating down the length of the pre-synaptic cell's axon reaches the synapse, the change in membrane potential causes sacs of neurotransmitter to migrate toward the synaptic junction and dump their contents into this space. The neurotransmitter flows across the junction in about 1 microsecond, and binds to special receptor molecules on the membrane of the post-synaptic cell. These bindings in turn cause the polarity of the dendrite to change. Most of the neurotransmitter is taken up again by the synapse for reuse.

There are two types of synapses each determined by the type of neurotransmitter they contain. The first type is called an **excitatory synapse** and contains either glutamate or aspartate. These neurotransmitters make it more likely that the polarity change at the post-synaptic dendrite will cause an input signal to begin the long, slow process of propagation to the post-synaptic soma. In other words, output signals sent across an excitatory synapse tend to excite the post-synaptic cell, and increase its activation level. The second type is called an **inhibitory synapse** and contains gamma-amino butyric acid. This neurotransmitter inhibits propagation of input signals in the post-synaptic cell. Therefore, output signals sent across an inhibitory synapse tend to depress the post-synaptic cell, and decrease its activation level. Although we will often speak of **excitatory and inhibitory connections**, we are actually referring to the type of synapse that establishes the connection. A synapse never contains both types of neurotransmitter. Synapses have been shown to contain peptides that mediate the effects of the neurotransmitters. In fact, some glial cells also produce peptides that may modulate information flow. Not much is known about the processing effect of peptides. The balance between the number of inhibitory and excitatory synapses seems to be a very important factor in the brain. About 80% of synapses are excitatory but the effect of the

remaining inhibitory synapses is quite important. Inhibitory synapses seem to be most strongly associated with local circuits and appear to assist in keeping neural outputs from triggering a domino-like chain of signals that spread unchecked through the entire brain. Long distance connections are always excitatory.

A very important observation is that the efficiency with which a synapse transmits a signal varies over time. This efficiency refers to the degree to which the synapse excites or inhibits the post-synaptic cell. A signal transmitted over an excitatory synapse with a high transmission efficiency would greatly increase the post-synaptic cell's activation level, while the same signal sent over an excitatory synapse of low efficiency would hardly increase the activation level at all. Very efficient inhibitory synapses would tend to greatly reduce the post-synaptic cell's activation level, while inefficient inhibitory synapses would have only a small effect. The rate of change of synaptic efficiency seems to be quite slow — on the order of hours or days. It is not clear what causes a change in efficiency [Bear, et. al., 1987] but it is likely that either the number of binding sites on the post-synaptic dendrite change or the amount of synaptic surface in contact with the dendrite changes. There is secondary evidence to support both claims. We will often refer to the efficiency of the synapse as its **connection strength** or **weight**. The brain is a powerful computational device, and clearly it must have both short term and long term memory storage systems.

Where then does memory lie among the myriad neurons that comprise the information processing sheets? Donald O. Hebb, an American psychologist, proposed a theory of memory in 1949 that has been expanded into the currently accepted explanation. Hebb suggested that the slow variability of the 10^{15} synaptic connections would make an ideal long term storage mechanism. Hebb suggested the following biologically plausible mechanism to control the change in connection strength: the strength of a synapse should be increased if the pre-synaptic and post-synaptic cells are firing simultaneously. This mechanism has since come to be known as **Hebb's law**. Hebb felt that this would tend to make it more likely that circuits which fired together in the past are active together in the future, thereby 'remembering' their past relationship. This rule would act as a self-organizing device that would create coalitions of functionally related circuits. Hebb called these circuits **cell assemblies**. As we shall see, almost all neural learning systems employ learning rules and self-organizing laws derived from Hebb's original suggestion.

At this point, a word of caution seems wise. Although entire libraries of data have been compiled by neurophysiologists on the response properties of neurons, almost nothing concrete is known about the level of computation that an individual cell can perform. We may eventually discover that neurons are actually general purpose turing machines that maintain sophisticated data

structures and perform algorithmic computations. There is weak evidence to suggest that patches of dendritic membrane can act as independent signal modifiers, and that sub-cellular mechanisms within the soma of the cell may perform signal modification [Crick & Asanuma, 1986]. However, most of the evidence so far suggests a simpler model of the neuron as an integrator of input signals and a thresholding device. It seems likely that high-level computation is derived from the interactions of networks of interconnected neurons arranged into the more sophisticated architectural structures described above.

2.3. A Simple Model and Notation

Our ultimate goal is to create artificial systems that exhibit general purpose cognitive and learning abilities by employing design principles borrowed from the brain. In order to achieve this goal we must develop simplified models of brain structures, and then explore these models for their computational properties. We have seen that the brain has many levels of structural complexity. From a modelling standpoint, there are four organizational levels to contend with: (1) the **neural level**, which is concerned with the most fundamental computational unit of the brain — the neuron; (2) the **network level**, which deals with the computational properties of networks of interconnected neurons; (3) a **functional level**, which combines networks having specific computational properties into modular devices that operate in specialized problem domains; and (4) a **systems level**, that combines specialized functional components to create general purpose learning systems. To date, almost all work has concentrated on the first two levels. A few attempts have been made to design functional level devices, but more work needs to be done at the network level before significant progress can be made. Needless to say, little thought has been given to the systems level yet.

At this point it is worth digressing to contrast the symbolic and neural learning system approaches from the levels viewpoint. Historically, the neural approach has progressed upward from the lowest organizational level. The working philosophy has been one of understanding high-level behaviour by identifying simple underlying systems. However, issues of knowledge representation have forced a trend toward higher-levels of complexity and abstraction. On the other hand, the symbolic learning field has examined the problem from the other end. The emphasis has been on system design using high-level functional principles without regard for details of implementation. Gradually, the symbolic camp has been forced by performance issues¹⁷ and the need for more powerful primitives to examine lower-levels of organization. In the end, as neural

¹⁷ For example, the problem of making systems that do not slow down as they accumulate more knowledge has led to underlying issues of parallelism.

systems move upward, and symbolic systems descend, the two approaches will probably find a common ground. In the mean time, both schools will benefit from a freer exchange of ideas¹⁸.

Let us now make use of our excursion into neuroanatomy to infer the outlines of a simple two-level model.

2.3.1. The Neural Model

We will take the view that neurons are simple computing devices that repeatedly perform the same computation using local information received from synaptic connections. These low level computing devices are alternately referred to as **neurons**, **cells**, **nodes**, **processors**, and **units**. We will employ the term **neuron** in this thesis¹⁹. Figure 2.13 is a schematic of our simplified neuron. As we will commonly deal with more than one neuron, each cell is assumed to have a unique integer ID. It will occasionally be convenient to refer to different types of neurons which we will call **cell types**. It is assumed that all neurons of a particular cell type have IDs that are consecutively numbered starting from 1. The notational device used to refer to a particular neuron will be a lowercase letter with a subscript. The lowercase letter will denote the cell type of the neuron. However, in keeping with much of the literature, the letter 'u' is reserved to mean a neuron of any type. A numeric subscript will refer to the ID of a specific cell, whereas a lowercase letter subscript will mean that the reference applies to any neuron of the appropriate type. For example, v_{14} refers to the 'v' type neuron with ID 14, and r_j refers to the j^{th} 'r' type neuron, where j could represent any appropriate ID; u_5 refers to all neurons with ID 5; u_k is an example of a general reference to any neuron.

¹⁸ We refer here to the inevitable friction that arises between practitioners of any two schools of thought that seem to be at odds. There have been a number of unfortunate incidents that have interfered with research in the area [Rumelhart and Zipser, 1986c].

¹⁹ A number of researchers prefer the term unit as it does not commit them to the neuron as the fundamental unit of computation in the brain. In their view, a unit may be a sub-cellular structure, or a coalition of cells acting in cooperation — i.e. a hyper-column. Due to lack of evidence for sub-cellular computation, and evidence that suggests hyper-columns are involved in data abstraction, we will call a spade a spade and refer to neurons as neurons. For those readers who feel uncomfortable with this term, in place of *neuron* read *processor*.

Figure 2.13 A Simple Model Neuron

Although inputs do occur at the soma and axon of real neurons, the effects of these connections are not well understood. Thus, for the sake of simplicity, we will assume that all inputs are received at the dendrites. Recall that inputs are modified by the efficiency of their synaptic connections. The connection strength between two neurons will be denoted by a lowercase greek letter followed by brackets that enclose the pre and post-synaptic neuron references. The greek letter will be used to refer to different classes of connections. For example, the α class connection between pre-synaptic neuron u_j and post-synaptic neuron u_i will be denoted to by $\alpha(u_j, u_i)$. Note the ~~order~~ of references inside the brackets. It will also be necessary to refer to the set of connection strengths associated with inputs to a neuron as a vector of real-valued components. When this is our intention we will denote this vector as a bold lowercase greek letter followed by brackets that enclose the post-synaptic neuron reference. For example, suppose that post-synaptic neuron u_m receives β class connections from three neurons u_i , u_j and u_k . The three dimensional vector of input connections for u_m is denoted by

$$\beta(u_m) = \begin{bmatrix} \beta(u_i, u_m) \\ \beta(u_j, u_m) \\ \beta(u_k, u_m) \end{bmatrix}$$

A neuron integrates the signals it receives to produce an activation level. The function $f(I, C)$ that relates the set of input signals I and the set of synaptic connections C to the activation level will depend on the type of neuron being modeled and the details of the model. We will refer to the

activation level of neuron u_i by the shorthand notation $a(u_i)$. Some types of neurons²⁰ do not receive inputs from other neurons. For example, consider the rod and cone cells of the retina. Their activation levels are determined directly by the amount of light striking the soma. It will always be clear from context how the activation level is determined if it is important to the model.

A Linear
Thresholding Function
(a)

A Hard-Limiting
Function
(b)

A Sigmoidal
Function
(c)

Figure 2.14 Typical Output Signal Functions

²⁰ Always sensory neurons. These types of neurons may also receive inputs from other neurons, in which case the activation level of the cell will be the sum of $f(I, C)$ and the function $f'(x)$, where x represents the physical parameter being sensed.

The output signal of a neuron u_i is determined by a function $g(a(u_i), \Theta(u_i))$ that takes as its parameters the activation level of the neuron and its firing threshold, denoted by $\Theta(u_i)$. Many models simplify the relationship between activation level and output signal by assuming a simple, linear threshold function such as Figure 2.14a. Other simple models employ a binary, 'hard-limiting' function like that shown in Figure 2.14b. Still other models assume a continuous 'sigmoidal' function like that shown in Figure 2.14c. We will use the shorthand notation $o(u_i)$ to refer to the output signal of neuron u_i . Since inputs are almost exclusively signals sent by other neurons, we will use output signal notation to refer to input signals as well.

The accepted view is that there is no information content in the patterns of spike trains — only the frequency is important. Therefore we will model signals between neurons as continuous, positive, real-valued analog signals. The range of values will be determined by the output function used. Excitatory synapses will be modeled as positive real-valued numbers, and inhibitory synapses will be modeled as negative real-valued numbers. The more efficient the synapse, the larger the absolute value of the connection strength. The signal received by a neuron is modified by the synaptic efficiency. We will model this modification by taking the product of the output signal received and the connection strength of the synapse. A very efficient excitatory synapse will produce large positive inputs, while a very efficient inhibitory synapse will tend to produce very small negative inputs. This suggests a simple sum of products for the activation level function. This is in fact the most popular model used, although there are exceptions to this rule.

2.3.2. The Network Model

Psychologists have known for some time that humans are able to perform very sophisticated feats of cognition within about 500 milliseconds. In this short span of time we can locate an object in a cluttered scene, analyse the semantic content of a sentence, recall the best action to perform in the given context and perform complicated motor actions. These and other related tasks take the most sophisticated computer programs orders of magnitude longer to perform²¹. This is even more remarkable when one considers that the basic computing component of the brain is six orders of magnitude slower than those employed by digital computers. If we conservatively assume that the average neuron has a response time of 1 millisecond, we see that the human brain must be capable of solving these complicated problems in about 500 computational steps.

We can characterize almost all of the tasks that humans find easy to perform but which are difficult to computerize as tasks requiring **simultaneous satisfaction of multiple**

²¹ Of course, there are a few specialized tasks that computers can perform better than humans — arithmetic computations for example.

constraints [McClelland, et. al., 1986]. By this we mean that many pieces of information are kept in mind at once, and each item is used to simultaneously constrain the set of solutions to the problem. Moreover, every item of information simultaneously influences every other item. For example, in the domain of word recognition, given a fraction of a second, most people will interpret the pattern of line segments in Figure 2.15 as the sentence, "THE CAT.". Notice that the middle symbol of each word is interpreted differently even though it is the same in each word. The 'T' and 'E' letters both contribute to constrain the interpretation of the middle letter as 'H', while the 'C' and 'T' both tend to constrain the interpretation to be 'A'. In addition, correct interpretation of one word tends to constrain interpretation of the other. If the second word is interpreted as a noun it is likely that the first word will be interpreted as a preposition rather than a noun or verb. Similarly, the movements that we make when reaching for a cup are constrained by the initial configuration of our arm, the location of the cup, the size and apparent heaviness of the cup, and its contents. Sentence interpretation involves constraining the interpretation of words both by the syntax of the sentence and the semantics of other words in the sentence. For example, in the sentence, "I like the ____.", the syntax of the sentence, (use of the preposition "the"), tends to constrain us to read any word in the blank as a noun. Thus, syntax influences semantics. On the other hand, given the sentence, "I like ____ eat.", the semantics of the last word, (a verb), make it likely that we will read the word in the blank as 'to' rather than 'the'. Semantics thereby constrains syntax.

Figure 2.15 An Ambiguous Sentence

The speed of neural processing and the importance of simultaneous satisfaction of multiple constraints suggest that computation in the brain is largely parallel in nature, not serial. This notion is consistent with the most striking characteristic of the brain — the unbelievably large number of computing elements available. Therefore, although it is counter to the traditional assumptions of computer science, we will assume that we have an infinite supply of simple neural computing elements at our disposal. The second most striking characteristic of the brain is the high degree of interconnectedness among neurons — there are on the order of 10^{15} synapses in the brain. Therefore, it is not unreasonable to assume that, in the general case, our

networks can be fully interconnected²². A common reaction to these assumptions is that we are somehow cheating by allowing them. Of course, it is thought, the solution to any problem becomes trivial if you have enough processors to throw at it. Nothing could be further from the truth! Those working in the field of distributed computing and parallel algorithm design have long appreciated the difficulty of trying to coordinate and control even small networks of processors — we are further limited by the need to keep our neurons simple. As we point out in section 5.2., our assumption of virtually unlimited neurons and connections may not be unrealistic. Optical technologies are available which could yield millions of neurons, and billions of connections per neurocomputer.

We will use the following notation to refer to networks of neurons. A neural network is a set of neurons denoted by an upper case letter. The upper case letter refers to the type of network being modeled. When there is only one type of neuron in the network, the cell type will be the lowercase equivalent of the network letter unless otherwise stated. For example, the 'K' class network containing five cell type 'k' neurons would be denoted by the set $K = \{k_1, k_2, k_3, k_4, k_5\}$. The function N takes the network set as its parameter and returns its size. For example, $N(K) = 5$.

The pattern of interconnections between neurons can take many forms. Five patterns have been extensively studied. The first is the **unconnected network** shown in Figure 2.16a. In this network, every neuron exists in isolation from the others. Surprisingly complex behaviours can arise from such networks. The second form is the **laterally connected network** shown in figure 2.16b. In this network, Each neuron connects to its immediate neighbours. The extent of its reach is sometimes referred to as its radius of influence. This type of network is very common in biological nervous systems, and can be found in many parts of the brain and sensory system. Its properties have been studied for decades, and its behaviours are relatively well understood. The third form is the **feed-forward network**. In this network, neurons are arranged into a stack of layers as shown in Figure 2.16c. The layers are numbered consecutively from bottom to top. Inputs to the network connect to neurons in the bottom-most layer. Outputs from the network are taken from neurons in the top-most layer. Connections between neurons are directed from lower layer neurons u_i to higher layer neurons u_j , such that $i < j$. The fourth form of network is called the **asymmetric network**. This network is fully connected and connections are assumed to be unidirectional as shown in Figure 2.16d.

²² Every neuron is connected to every other neuron in the network.

An Unconnected Network A Laterally Connected Network A Feed-Forward Network An Asymmetric Network A Symmetric Network

Figure 2.16 Five Important Neural Network Architectures

The fifth type of network is called the **symmetric network**. This is also a fully connected network, however, the connections are assumed to be bi-directional; ie. information — output

signals — flows both ways over the connection. Although there is no evidence for bi-directionality of biological connections, this assumption is sometimes made to simplify mathematical analysis. One can think of a bidirectional network as being equivalent to a symmetrical network in which reciprocal connections between neurons have exactly the same connection strength. Many other connection patterns are also possible. We will denote the connection pattern of a network by a square matrix with rows for every neuron in the network, and columns for every neuron in the network. This is called the **weight matrix** of the network. The components of the weight matrix represent connection strengths between the pre-synaptic row neurons and the post-synaptic column neurons. The weight matrix is denoted by a bold upper case letter (usually **W**). Components of the matrix are denoted by the lower case version of the same letter followed by row and column subscripts (e.g. w_{32} denotes the connection from u_3 to u_2). A connection strength of 0.0 indicates no connection between two neurons.

Another network level issue is synchronous vs. asynchronous updating. By **synchronous updating**, we mean that there is a global clock that triggers all neurons (or a subset of neurons) to simultaneously calculate a new output signal from available inputs. By **asynchronous updating** we mean that every neuron has an associated probability that at a given instant in time it will calculate a new output signal. This is usually simplified, such that, at every instant in time, one neuron is randomly selected²³ from the network and updated. Both forms of updating are attempts to model continuous, time varying neural processing in the brain, and both arise from constraints imposed by simulating neural networks on digital computers. Various authors have pointed to the need for analog hardware in which the passage of time is not simulated but arises naturally [Mead, 1987]. As we shall see, both approaches have been used, although synchronous updating is historically more popular. We will assume synchronous updating throughout subsequent chapters unless otherwise stated.

From the available evidence, we can infer that the only long term change that occurs in the brain involves modification of synaptic connections. This change occurs through the growth of new connections, the subsequent modification of connection efficiencies, and the decay of old connections. The growth and decay of connections is actually a special case of connection strength modification. If a connection strength in the weight matrix changes from 0.0 to a non-zero value, this is equivalent to the growth of a new connection. Similarly, connection decay is equivalent to a non-zero connection strength changing to 0.0. As we shall soon see, the firing threshold of a neuron can also be treated mathematically as the connection strength associated with an imaginary neuron that always outputs the signal 1.0. We will refer to the function h that governs the change

²³ With uniform likelihood across all neurons.

in connection strength as the **weight modification rule**. In section 2.1.2, we described neural learning systems as black-boxes that hide the details of an input to output transformation. This transformation was determined by myriad control parameters. The reader has probably already guessed that the distributed system responsible for this transformation is a network of neurons, which transforms input signals received by sensors into output signals that leave the network. This transformation is the result of the interaction of many neurons computing activation levels and output signals. The control parameters are represented by synaptic connection strengths and firing thresholds that determine the activation level of each processor and control its output signal. We have said that learning is the process of finding the right set of control parameters. Therefore, it should now be clear that the weight modification rule is the essence of learning. Many weight modification rules are possible, but almost all the work to date employs a variation of Hebb's law. Although there is no direct biological evidence to support this law, there is significant secondary evidence. Also, no other theory of memory and learning has stood the test of time as well. The weight modification rule is generally a function $h(\alpha(u_i, u_j), o(u_i), a(u_j), o(u_j), T)$ that takes as its parameters the value of the weight before modification, the output signal sent by the pre-synaptic cell, the activation level and/or the output signal produced by the post synaptic cell, and sometimes a special training signal T that we will discuss in section 2.4.

2.3.3. The Functional Level

At the functional level, we must be able to refer to more than one network. A network of neurons will be denoted by an upper case letter followed by a subscript. The capital letter will refer to the type of network, and the subscript a particular network of this type. Further research will be required at the functional level before we can expect much progress at the systems level.

2.4. A Survey of Algorithms and Techniques

This section is intended to give the reader a better understanding of neural learning systems by presenting and analysing six important algorithms. These algorithms are all described in the common notation introduced in section 2.3., using terms that will be employed later to describe the design of a neural robot arm controller. The algorithms are grouped according to the three major paradigms of neural learning systems described in section 2.1.2. The reader is encouraged to understand the surveyed algorithms as they introduce some very important concepts, techniques and terminology that is best seen in the context of a real neural model. It is one thing to speak in generalities as we have been doing, and quite another to actually try to understand example algorithms. Although this is not a definitive survey — such a survey would fill a text book — these algorithms have been carefully selected to give the reader an understanding of the key research that much of the current literature is built upon and makes reference to. This background

material is a good foundation for further reading and research. Much of the current work is derivative of these six ideas²⁴. It is our fond hope that this section will leave the reader with the essential *flavour* of this kind of computation.

It is also worth noting that these algorithms represent attempts to model the equivalent of a single layer of a single hyper-column in the brain²⁵. The actual simulations that have been carried out typically involve fewer than 10,000 neurons — less than 10% of the neurons in a single hyper-column²⁶. Yet even these primitive networks exhibit surprisingly useful behaviours. We suggest that the reader think of these behaviours as powerful primitives from which 'intelligent' learning systems might be built.

The largest amount of effort has gone into developing neural networks that are able to learn arbitrary associations between patterns. Much of this work has been done under the guise of pattern classification techniques, and adaptive signal processing systems [Duda & Hart — Chapter 5, 1973; Widrow, et. al., 1987; Widrow, 1985].

2.4.1. Arbitrary Pattern Association Algorithms

2.4.1.1. The Perceptron

In 1959 Frank Rosenblatt developed one of the most influential and controversial neural network algorithms — the **perceptron**. Rosenblatt wanted to understand how the human brain learns to classify such a wide range of sensory data — words, images, textures, odours, etc. We begin our lives with no ability to distinguish one object, or sensation, from another, and yet over a few years, if we encounter enough examples of these things, we learn to classify them. Furthermore, we are very good at **generalizing** from examples. For example, once we have encountered a few dozen examples of a bicycle, we can correctly classify bicycles that we have never encountered before. If a novel object is very similar to a class of objects that we already know, we might incorrectly classify it at first. However, if we are corrected by a teacher, and are given the opportunity to observe more examples of the new class of objects, we will eventually learn to distinguish them from others. All forms of sensory stimuli are merely patterns of neural

²⁴ This is not to say that no original work has been done in the last few years, but a large amount of effort has gone into understanding why these algorithms work, how they can be made to work better, and how they can be applied to various domains. After all, most new ideas are built upon old ones.

²⁵ The cognitron is supposed to represent one function of a group of hyper-columns. Still it does not claim to model all the capabilities of an entire hyper-column.

²⁶ The current research bottleneck is speed of simulation. It takes a very, very long time to simulate a fully connected network of 10,000 neurons. However, better hardware is being developed as described in section 5.2., and as it becomes available we will see more sophisticated models appear.

signals by the time they reach the brain. Rosenblatt hypothesized that a common neural mechanism must associate all sensory patterns in a class with a single neuron that represents the class.

The Network Model

Rosenblatt's perceptron network is an unconnected architecture that consists of a single layer of neurons as shown in Figure 2.17. We will refer to the network as the set

$$P = \{p_1, p_2, p_3 \dots p_{nc}\},$$

where $N(P) = n_c$ = the maximum number of pattern classes. Inputs to the perceptron arrive from a set of sensory cells

$$S = \{s_1, s_2, s_3 \dots s_{ns}\},$$

where $N(S) = n_c$ = the maximum number of sensory cells. We shall refer to the strengths of connections between the sensory neurons and perceptron neurons by $\alpha(s_i, p_j)$.

Figure 2.17 Perceptron Architecture

The Neuron Model

Sensory cells produce real-valued outputs. Perceptron neurons sum the products of incoming signals and connection strengths to produce an activation level. The firing threshold is then subtracted from this value. The output signal is a simple, two-state, hard-limiting function that outputs a -1.0 if the activation level is lower than the neuron's firing threshold, or a +1.0 if the neuron fires.

$$a(p_j) = \left(\sum_{i=1}^{N(S)} o(s_i) \cdot \alpha(s_i, p_j) \right) - \Theta(p_j) \quad [2.4.1.1.1]$$

$$o(p_j) = \begin{cases} 1.0 & \text{if } a(p_j) > 0 \\ -1.0 & \text{if } a(p_j) \leq 0 \end{cases} \quad [2.4.1.1.2]$$

The Learning Algorithm

The perceptron learning algorithm is an example of a supervised learning scheme. A teacher is required to correct the network when it outputs incorrect values. The correction is called a teaching signal and is denoted by $T(p_j)$. It is assumed that a teacher already knows how many classes there are. Before learning begins, the teacher assigns a unique class to each neuron in P . The goal of the network is to learn to classify input patterns such that, when a pattern is presented, the neuron associated with the class of the pattern outputs the value +1.0, and all other neurons output the value -1.0. If we consider the network output signals to be an $N(P)$ -dimensional vector $\mathbf{o}$, correctly classified patterns will produce a sequence of linearly separable vectors. This is a form of pattern association where arbitrary input patterns are associated with binary output patterns. The learning algorithm proceeds in 7 steps:

1. $\forall i,j$: the teacher initializes firing threshold $\Theta(p_j)$ and weight $\alpha(s_i, p_j)$ to random, real-valued numbers²⁷,
2. The teacher randomly selects a training sample and presents it to the perceptron, noting the class of the training pattern,
3. $\forall j$: calculate $a(p_j)$ and $o(p_j)$,
4. $\forall j$: the teacher provides neuron p_j with a teaching value $T(p_j)$ that represents the signal that the neuron should have output for the training sample pattern. $T(p_j)$ will equal 1.0 if the training pattern class is the same as the class assigned to neuron p_j . $T(p_j)$ will equal -1.0 otherwise.
5. $\forall i,j$: neuron p_j updates its weights according to the rule

$$\alpha(s_i, p_j) = \alpha(s_i, p_j) + \eta (T(p_j) - o(p_j)) o(s_i), \quad [2.4.1.1.3]$$

where η is a small proportionality constant — typically 0.01.

6. $\forall i,j$: neuron p_j updates its firing threshold according to the rule

$$\alpha(s_i, p_j) = \alpha(s_i, p_j) + \eta (T(p_j) - o(p_j)), \quad [2.4.1.1.4]$$

where η is a small proportionality constant — typically 0.01.

7. goto 2.

²⁷ Typically small numbers [in the range -1.0 to +1.0].

The algorithm repeats until the perceptron correctly classifies all training samples. Although this network model is remarkably simple, it *will* learn to correctly classify patterns under the correct circumstances. As training examples are presented, the network weights converge toward a set of values that cause the correct input pattern to output pattern transformation. Moreover, it tends to correctly classify novel patterns that belong to the classes that it was trained on.

This learning and generalization behaviour generated a great deal of excitement in the computer science community during the early 1960's. Unfortunately, it soon became apparent that the approach was limited to very simple types of problems [Minski & Papert, 1969]. In the general case, the weights do not converge. Several strong claims made by Rosenblatt for the power of perceptron learning were not substantiated, and interest and funding began to decline at the close of the 1960's. However, an intuitive understanding of this algorithm and its weaknesses is an important step toward an understanding of all arbitrary pattern associators.

Learning Behaviour Analysis

To understand how the network learns, we must take a short detour into **Bayesian decision theory**. Suppose that we have a set of classes $C = \{c_1, c_2, c_3 \dots c_n\}$. For example, these classes could represent diseases. Suppose that medical science has statistically determined the probability that a patient has disease c_i . This probability is called the **a priori probability** of the class, $P(c_i)$. If we are asked to classify a disease without being allowed to observe the patient, our best guess would be to choose the disease that has the largest a priori probability. This way, we will minimize our probability of failure. Now, suppose that the next time we are asked to classify a disease we are more reasonably allowed to observe the patient and collect a data value x . This data value can represent any significant feature of the patient; e.g. the patients temperature. How should we use this value to modify our diagnosis ?

Bayes provides us with an answer. Consider variable x to be a continuous random variable whose distribution depends on the state of nature (ie. type of disease). The observable x forms a one-dimensional space of possible values which we will call the **feature space**. Each disease c_i will have an associated density function over feature space, where the probability that x falls in the range $[x_i, x_j]$ when the disease is c_i is given by the area under the density function between values x_i and x_j (Figure 2.18a). The area under the entire curve is always 1.0. We will refer to this curve as the **state-conditional probability density function** $p(x | c_i)$. Figure 2.18b depicts an example state-conditional probability density function for a 2 class problem.

(a) State Conditional Probability Density Function $p(x | c_i)$

(b) Two Class Density Functions

(c) Density Functions Transformed into A Posteriori Probabilities

Figure 2.18 Bayes' Rule Finds Decision Regions and Boundaries

Bayes tells us that to minimize our probability of failure, we should first compute the a posteriori probability of each of the diseases as follows

$$P(c_i | x) = \frac{p(x | c_i) \cdot P(c_i)}{\sum_{j=1}^n p(x | c_j) \cdot P(c_j)}, \quad [2.4.1.1.5]$$

and then select the disease that has the largest resulting a posteriori probability²⁸. The a posteriori function transforms the state conditional probability density functions into new curves over the feature space as shown in Figure 2.18c. Let us refer to these functions as **discriminant functions**, $g(c_1)=P(c_1 | x)$, $g(c_2)=P(c_2 | x)$, etc. Given a point x in feature space, we select the disease with the highest discriminant function value at that point. The point x_d where two discriminant functions have the same value represents a border between regions in which observables diagnose different diseases. This is called a **decision boundary**. Decision boundaries divide the feature space into one — possibly unbounded — **decision region** for each disease. Figure 2.19 shows that these properties extend to higher dimensions in which we have more than one observable. When this is the case, we represent observables as a **feature vector** $x = [x_1, x_2 \dots x_m]$. These vectors define an m -dimensional feature space in which observable items (e.g. diseases) are represented by points. The curves where the discriminant functions intersect project onto feature space as non-linear decision boundaries.

Figure 2.20 illustrates a 2-dimensional feature space and decision boundaries for three classes. These boundaries tend to be non-linear in the general case²⁹, as shown in Figure 2.20a, however, there are many cases where the decision boundaries can be approximated by straight lines³⁰. This is illustrated by Figure 2.20b. The clusters of circles, triangles and squares in feature space represent the vectors associated with a random selection of objects in each class. The mean and standard deviations of these clusters depend on the statistical details of the class.

²⁸ See [Duda & Hart, 1973; Neter, et. al., 1978] for a good introduction to Bayes decision theory. The first reference also proves that this technique minimizes error.

²⁹ The shape of the decision boundaries clearly depends on the shapes of the a posteriori functions for the classes. Points on the decision boundary are points where the Mahalanobis distance to the mean of two or more a posteriori functions are equal. If we assume gaussian distributions, this is given by

$$(x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) = (x - \mu_2)^T \Sigma_2^{-1} (x - \mu_2) = \dots$$

μ_i and Σ_i are the mean and covariance matrix of the distributions.

³⁰ The decision boundaries will be straight lines if the covariance matrices are diagonal (the observable features are statistically independent and each feature has the same variance), or the covariance matrices of the classes are equal.

Figure 2.19 A 2-Dimensional Feature Space

The Non-Linear Decision Boundaries in (a) can be Approximated by Straight-Lines as in (b)

Figure 2.20
Straight Lines Can Often Approximate
Non-Linear Decision Boundaries

Perceptrons do NOT compute the statistics of the training data distributions and apply Bayes' rule to find the non-linear decision boundaries. Instead, perceptrons replace the a posteriori discriminant functions with linear functions. These linear functions describe simple hyper-planes like those shown in Figure 2.21, rather than the uni- or multi-modal surfaces described by the a posteriori discriminant functions shown in Figure 2.19. Training sample data is used to iteratively modify the coefficients of the linear functions in order to position the hyper-planes so that their intersections produce **straight line approximations** to the desired decision

boundaries. These lines of intersection always project onto feature space as straight line decision boundaries which can be used to approximate the true, underlying non-linear decision boundaries for the classes. Once appropriate decision boundaries have been found that correctly classify the training samples, other, novel patterns from the class will tend to cluster in the same region, and be correctly classified. This is how the perceptron generalizes from its learning experiences.

Figure 2.21 Perceptron Discriminant Functions are Linear

Figure 2.22 Firing Thresholds are Analogous to Weights

The linear discriminant functions are actually the activation level functions of the neurons. Recall the activation level function 2.4.1.1. and expand the summation term. This sum of products that approximates the integration function of neurons is a linear function,

$$a(p_j) = \alpha(s_1, p_j) \cdot o(s_1) + \alpha(s_2, p_j) \cdot o(s_2) + \dots + \alpha(s_N(s), p_j) \cdot o(s_N(s)) + \Theta(p_j) \cdot 1.$$

The variables in the function are the input signals received from the sensory cells. The coefficients are the connection strengths.

Notice that the firing threshold is mathematically equivalent to a connection weight. It is as if our linear equation has another term which represents a connection from an imaginary sensory cell that permanently outputs the value 1 (Figure 2.22). This is a trick common to most neural models [Duda & Hart, 1973; Lippmann, 1987; Rumelhart, et. al., 1986b] which greatly simplifies the analysis.

The properties of this linear equation are illustrated in Figure 2.23. We can view the $N(S)$ weights associated with each neuron (plus the firing threshold 'weight') as a **weight vector** in an $(N(S)+1)$ -dimensional **weight space**. The dimensionality of the weight space will be the same as the feature space, because there is exactly one weight per input signal. Imagine plotting the weight vector in feature space. In this case the feature axes are simply the outputs of the sensory cells. The linear discriminant function is equivalent to the inner product of the weight vector and a feature vector

Figure 2.23
A Geometric Interpretation of the
Activation Level of a Perceptron Neuron

The inner product is proportional to the distance³¹ of the feature vector $\mathbf{o}$ to a line normal³² to the weight vector $\alpha(p_j)$. This normal line divides the plane into two half-planes. The inner product is positive for feature vectors that fall into the half-plane containing the weight vector and negative for feature vectors that fall into the other half-plane. The closer the feature vector is to the normal line, the smaller the inner product. Feature vectors that fall on the normal line produce an inner product of 0.

Figure 2.24 Perceptrons Learn by Descending a Criterion Surface

As in all neural learning systems, the problem is reduced to finding the right weight values. How is this accomplished? A powerful approach employed by many neural learning systems is **gradient search**³³. The idea is to define a **criterion function** that relates weight vectors to a measure of how well system behaviour matches optimal behaviour. A function is chosen that outputs zero if all training patterns are correctly associated with the output pattern, and positive values otherwise. A function with a smooth surface and no local minima is preferred. Once an appropriate criterion function has been defined, the gradient is found with respect to the weights. The negative of this gradient is of course the vector that points in the direction of steepest descent

³¹ This distance is actually equal to $\frac{\alpha(p_j)^T \mathbf{o}}{\|\alpha(p_j)\|}$.

³² For two points $\mathbf{o}_1$ and $\mathbf{o}_2$ on this line, the inner product is 0. Therefore, $\alpha(p_j)^T \mathbf{o}_1 = \alpha(p_j)^T \mathbf{o}_2$ and $\alpha(p_j)^T (\mathbf{o}_1 - \mathbf{o}_2) = 0$. From this we see that the weight vector $\alpha(p_j)$ must be normal to this line.

³³ This technique that is actually a common optimization technique which occurs in many contexts — engineering, management science, the humanities, statistics, etc.

[Grossman, 1977]. Figure 2.24 illustrates this criterion function. In order to improve the system's behaviour — ie. minimize the criterion function — we need only change the weight vector in the direction of the negative gradient vector. Assuming we make small enough steps down the criterion surface, we will eventually find an appropriate set of weights as shown in the diagram.

Let us simplify the problem slightly and analyse the behaviour of a single neuron in P. Assume that we have two classes of training samples, A and B. We want our neuron to output a positive value when presented with patterns from class A and a negative value when presented with patterns from class B. As we saw above, the linear discriminant function will do this as long as the line normal to the weight vector separates the two classes (Figure 2.25a). First notice that there are many weight vectors that will produce an appropriate normal line. The shaded region in Figure 2.25b represents the set of weight vectors that produce the desired system behaviour. Secondly, notice that if we multiply all the patterns in class B by -1 (flip them by 180 degrees — Figure 2.25c), the same set of weight vectors will always produce a positive inner product as can be seen in Figure 2.25d. Only incorrectly classified patterns will produce a negative inner product. Therefore, assuming that we have flipped the class B patterns, the sum of all the patterns that are incorrect will be negative until a normal line is found that correctly separates all the training sample patterns. We need only multiply this sum by -1 to find an appropriate criterion function. If the set of incorrectly classified patterns is I, we can express this criterion function as

$$E(\alpha(p_j)) = -1 \cdot \sum_{o \in I} \alpha(p_j) \cdot o = -1 \cdot (\alpha(p_j) \cdot o_1 + \alpha(p_j) \cdot o_2 + \dots). \quad [2.4.1.1.6]$$

Taking the negative of the gradient we have

$$-\nabla E(\alpha(p_j)) = - \frac{\delta E(\alpha(p_j))}{\delta \alpha(p_j)} = o_1 + o_2 + \dots = \sum_{o \in I} o. \quad [2.4.1.1.7]$$

The i^{th} weight component of the gradient is

$$- \frac{\delta E(\alpha(s_i, p_j))}{\delta \alpha(s_i, p_j)} = o(s_i)_1 + o(s_i)_2 + \dots = \sum_{o \in I_i} o(s_i). \quad [2.4.1.1.8]$$

This is the amount by which we wish to change each weight vector in order to improve system behaviour. The neuron does not have available the sum of incorrectly classified signals. Therefore, we will simply replace the sum with a correction term that is zero if the pattern is correctly classified (ie. the output signal of the neuron is equal to the expected output of the neuron), and non-zero if not. If the pattern is a class A pattern, we will want this correction term to be positive. If the pattern is a class B pattern, we must compensate with the trick of multiplying by -1. Therefore we want the correction term to be negative. All three requirements are nicely handled by

simply subtracting the output signal of the neuron from the expected output signal of the neuron — the training value. Finally, we want the change in the weight vector to be small enough that we do not over-shoot the solution region. Therefore we multiply the negative gradient term by a small constant, η . Putting these requirements together, the weight change rule becomes equation 2.4.1.1.3. Since the firing threshold is simply another weight on a connection from an imaginary neuron outputting the value 1, equation 2.4.1.1.4. is a special case of 2.4.1.1.3. Luckily, the perceptron criterion function presents a smooth bowl shaped surface, and gradient descent on this surface is not hampered by local minima.

Figure 2.25 A Trick to Simplify Analysis

Rosenblatt and others [Duda & Hart, 1973] have proven that the perceptron algorithm will converge on a solution in finite time, provided that such a solution can be approximated by straight

line decision boundaries. The weight change rule works for networks of any size³⁴. This technique of gradient search appears many times in the context of neural learning systems.

The Minsky-Papert Analysis

In 1969, Marvin Minsky and Seymour Papert published a thorough analysis of the perceptron [Minsky & Papert, 1969] in which they characterized training sample data that the perceptron is unable to classify correctly. Distributions of training samples that are non-linearly separable³⁵ cannot be classified by linear discriminant functions. A simple example of a two class problem that cannot be separated by a linear perceptron decision boundary is shown in Figure 2.26a³⁶. In this case and others like it, the perceptron learning algorithm oscillates and does not converge. Another algorithm due to Widrow and Hoff [Widrow, 1985] that is derived from a least mean squares criterion function³⁷ will find a least mean square solution. The 'best fit' decision boundary that results minimizes the mean square error between expected output signals and actual output signals. The resulting neural model and learning rule, called the MADALINE network³⁸, is exactly the same as the perceptron except that the output function is linear, and does not produce binary values. The MADALINE algorithm still does not work well for complex distributions. Minsky and Papert cast serious doubt on the ability of future perceptron-like machines to handle real world problems. Research in the area virtually came to a halt for 16 years.

³⁴ With two or more neurons the decision boundaries consist of points where two hyper-planes meet, ie. $a(p_i) = a(p_j)$. Therefore, $\alpha(p_i)T_o = \alpha(p_j)T_o$ and $(\alpha(p_i) - \alpha(p_j))T_o = 0$. From this we see that feature vectors that fall on this decision boundary produce an inner product of zero and therefore the decision boundary is normal to $\alpha(p_i) - \alpha(p_j)$. It is the *difference* between weight vectors that counts in multi-neuron perceptrons.

³⁵ Usually multi-modal.

³⁶ This is often called the XOR problem [Rumelhart, et. al., 1986b].

³⁷ $E(\alpha(p_j)) = \sum_{k=1}^m \|\alpha(p_j)T_o - T_k\|^2$, for k training samples and teaching signals T , and $-\nabla E(\alpha(p_j)) = -2(\alpha(p_j)T_o - T)$, $\therefore$

$\alpha(s_i, p_j) = \alpha(s_i, p_j) + \eta (T(p_j) - \alpha(p_j))\alpha(s_i)$.

³⁸ In pattern recognition circles this neural model has been adopted as the Least Mean Square algorithm or Widrow-Hoff Procedure [Stone, 1986].

These two classes of training sample cannot be separated with a linear decision boundary

(a)

(b)

Decision Boundaries in Input Space

(c)

Final Separating Decision Boundary in Transformed Space

(d)

Figure 2.26
Non-Linearly Separable Classes Require
Multi-Layered Perceptrons

Had Minsky and Papert extended their analysis to multi-layer perceptrons, they would have discovered that the perceptron is actually capable of handling many complex learning problems. Figure 2.26b shows a multi-layer, feedforward perceptron architecture. Although the samples are not separated by the decision regions generated by the neurons in the first layer (Figure 2.26c), the first layer of neurons transforms the training samples into an output space in which they are linearly separable. These transformed points are in turn used as the inputs to the next layer of neurons, and are correctly separated by this layer (Figure 2.26d). The vertically striped region represents class A and the horizontally striped region represents class B. Intuitively, this process is analogous to building decision regions from intersections of the half-planes that result from linear decision boundaries. Given enough neurons and enough layers, weight vectors can be found that generate arbitrarily complex decision boundaries that are very good approximations of the non-linear a posteriori boundaries (see Figure 2.27). Lippmann has identified a hierarchy of decision region types and the perceptron architectures required to approximate them [Lippmann, 1987].

(a)

Multi-Layer Perceptrons can
Generate Complex Decision
Boundaries

(b)

Figure 2.27 Every Neuron Must Be Taught

Unfortunately, even though multi-layer machines can be designed by hand to perform complex classifications, the perceptron learning algorithm is insufficient to find them. Consider for a moment that the teacher must tell every neuron whether it is generating the correct signal — even

neurons in the lower layers as we see in Figure 2.27a (above) there may be many training signals. In order for the teacher to know what signals are needed to produce the correct lower layer transformations, he must know enough about the underlying distributions that he could simply construct the weight vectors by hand, or compute the statistics for a Bayes classifier. In either case, the need for a learning machine disappears.

2.4.1.2. Back Propagation

It was not until 1985 that a learning algorithm for multi-layer neural architectures was found. As is often the case in scientific research, three groups working independently came across similar learning rules at about the same time. D.B. Parker appears to have been the first to publish the algorithm he called Learning-Logic [Parker, 1985]. Y. LeCun also published an algorithm that is roughly similar to Parker's [Le Cun, 1985]. The Parallel Distributed Processing Research Group at the University of California, San Diego, followed a year later with essentially the same algorithm, this time called **Back Propagation**. The PDP algorithm was cast in terms of neural computation and very quickly gained in popularity, until at this time it is the most widely known neural learning algorithm.

Back propagation is essentially an extension of the perceptron and MADALINE algorithms. It generalizes these algorithms in three important ways. First and most importantly, back propagation makes powerful multi-layer networks look like single layer perceptrons. The teacher only provides training signals to neurons in the last (output) layer. The algorithm automatically provides neurons in lower layers with training signals derived from the output layer that can be used to update weight vectors. In a sense, training signals are propagated backwards through the network. Hence the name back propagation. Secondly, output patterns do not have to be binary valued. Any pattern of real valued inputs can be associated with any pattern of real-valued outputs in the range of the output signal function. This means that the network can handle a larger class of associations³⁹. Thirdly, back propagation allows the network designer to employ any differentiable, monotonically increasing function as the neuron's output signal equation. This is important for two reasons. First, arbitrary output signal functions mean that non-linear discriminants may be used to transform inputs and find decision boundaries. This can be important when working with training sample data that has very unusual distributions. Currently there is no formal approach for the selection of output signal functions, but studies have suggested that it is important to find the best function empirically [Baum, 1986]. Secondly, organic neurons appear to use a roughly sigmoidal output

³⁹ Actually back propagation does the same thing as a perceptron — it splits up a feature space into decision regions. However, the continuous output signal functions allow real numbered outputs. If the decision regions are small enough, the perceptron can act as a fairly good arbitrary pattern associator.

signal function. Back propagation has provided a mechanism to make more biologically accurate neural models. The algorithm has been applied to a wide variety of problems; speech compression and recognition [Watrous & Shastri, 1987], computer vision [Yang & Guest, 1987], plant control [Psaltis, et., al., 1987], game playing [Mozer, 1986], robot control [Elsley, 1988], and others. In practice it compares favourably to all other non-parametric pattern recognition techniques and arbitrary pattern associators. The algorithm works particularly well in highly non-gaussian problem domains.

The Network Model

The back propagation network is a feed-forward architecture with two or more layers and any number of processors per layer as shown in Figure 2.28. The network processors are referred to by the set

$$B = \{b_1, b_2, b_3 \dots b_{nc}\}.$$

The processors in the i^{th} layer are denoted by B_i . The number of layers in the network is given by $NL(B)$. Unlike the perceptron, sensory cells are included in the network. Layer B_1 is composed entirely of sensory cells. This layer is called the **input layer**. The output values of these cells are forced to the appropriate training sample pattern. This process of setting the outputs of these cells is called **clamping** the input layer neurons. We shall refer to the strengths of connections between neurons in adjacent layers by $\alpha(b_i, b_j)$, where $b_i \in B_k$ and $b_j \in B_{k+1}$. Outputs are taken from the last layer of the network, $B_{NL(B)}$. This layer is called the **output layer**. All other layers (if any) are called **hidden layers**.

The Neuron Model

Input layer neurons have their output signals clamped to real-valued numbers. All other back propagation neurons sum the products of incoming signals and connection strengths to produce an activation level. The firing threshold is then subtracted from this value. The output signal is an arbitrary, nondecreasing, differentiable function. Such functions are sometimes called **semi-linear**⁴⁰. Usually a biologically plausible sigmoidal function is used [Rumelhart, et. al., 1986b]. The sine function is also popular [Baum, 1986].

⁴⁰ Because part of the function often has linear characteristics such as the ramp in the middle of a sigmoidal function.

Figure 2.28 Back Propagation Architecture

For an arbitrary neuron b_j in layer B_{k-1}

$$a(b_j) = \left(\sum_{b_i \in B_{k-1}} o(b_i) \cdot \alpha(b_i, b_j) \right) - \Theta(b_j) \quad [2.4.1.2.1]$$

$o(b_j)$ = any semi-linear function

for example, the sigmoid $o(b_j) = \frac{1}{1 + e^{-a(b_j)}}$, [2.4.1.2.2]

or the sine $o(b_j) = \sin(a(b_j))$.

The Learning Algorithm

From the teacher's viewpoint, back propagation networks behave the same as perceptrons. A training pattern is input at one end of the network, and outputs pop out the other end. There are two differences however. First, the outputs of layer B_i are the inputs to layer B_{i+1} . So after inputs are clamped at the input layer, signals must be propagated from lower layers up through the

network until the output layer generates output signals. This is a synchronous process as the neurons in each layer fire in unison. The second difference is that after the teacher provides training signals to the output layer neurons exactly as in the perceptron algorithm, these signals are propagated backward to the next lower layer, and so on until the input layer is reached. The training signals are combined at each layer and used to modify connection strengths. The learning algorithm proceeds in 7 steps:

1. $\forall i, j$: the teacher initializes firing threshold $\Theta(b_j)$ and weight $\alpha(b_i, b_j)$ to random, real-valued numbers,
2. The teacher randomly selects a training sample and clamps the input layer neurons, noting the class of the training pattern,
3. PROPAGATE FORWARD

For $i := 2$ To $NL(B)$ Do

$\forall j \in B_i$: calculate $a(b_j)$ and $o(b_j)$;

4. $\forall j \in B_{NL(B)}$: the teacher provides neuron b_j with a teaching value $T(b_j)$ that represents the signal that the neuron should have output for the training sample pattern.
5. $\forall j \in B_{NL(B)}$: output layer neuron b_j updates its weights according to the generalized output layer rule

$$\delta(b_j) = (T(b_j) - o(b_j)) \frac{\delta o(b_j)}{\delta a(b_j)} \quad [2.4.1.2.3]$$

$$\forall i \in B_{(NL(B)-1)}: \alpha(b_i, b_j) = \alpha(b_i, b_j) + \eta o(b_i) \quad [2.4.1.2.4]$$

If we use the sigmoidal output signal function given by equation 2.4.2.3.2 we have

$\forall i \in B_{(NL(B)-1)}$:

$$\alpha(b_i, b_j) = \alpha(b_i, b_j) + \eta (T(b_j) - o(b_j)) o(b_j) (1 - o(b_j)) o(b_i) \quad [2.4.1.2.5]$$

6. PROPAGATE BACKWARD

For $j := (NL(B)-1)$ DownTo 2 Do

hidden layer neuron b_j updates its weights according to the generalized output layer rule

$$\delta(b_j) = \frac{\delta o(b_j)}{\delta a(b_j)} \sum_{k \in B_{(j+1)}} \delta(b_k) \cdot \alpha(b_j, b_k) \quad [2.4.1.2.6]$$

$$\forall i \in B_{(j-1)}: \alpha(b_i, b_j) = \alpha(b_i, b_j) + \eta o(b_i) \quad [2.4.1.2.7]$$

If we use the sigmoidal output signal function given by equation 2.4.2.3.2 we have

$$\delta(b_j) = o(b_j) (1 - o(b_j)) \cdot \sum_{k \in B_{(j+1)}} \delta(b_k) \cdot \alpha(b_j, b_k) \quad [2.4.1.2.8]$$

$$\forall i \in B_{(j-1)}: \alpha(b_i, b_j) = \alpha(b_i, b_j) + \eta o(b_i) \quad [2.4.1.2.9]$$

7. goto 2.

The algorithm repeats until the network correctly classifies all training samples. The range of the outputs are determined by the output signal function.

Some simple empirical studies have been performed to determine the speed of learning in the network. The number of presentations needed to correctly classify the distribution shown in Figure 2.26a with $\eta=0.25$ is given by

$$280 - 33 \log_2 H,$$

where H is the number of hidden neurons in a single layer between the input and output layers. This relation holds for $H \leq 40$. Reducing speed of learning by increasing the number of hidden neurons seems to apply to all learning domains. However, further study is required.

Learning Behaviour Analysis

Back propagation is also a form of gradient search. In this case the criterion function used is the sum of squared differences between expected network outputs and actual network outputs at the output layer

$$E(o(b_j)) = \frac{1}{2} \sum_{j \in B_{NL(B)}} (T(b_j) - o(b_j))^2. \quad [2.4.1.2.10]$$

As required, this function outputs zero if no errors occur, and a positive number otherwise. It is also differentiable. However it will generally have many local minima. If we assume i, j and k are level numbers in ascending order, the derivation of the learning algorithm proceeds as follows:

1. We wish to find the negative of the gradient of the criterion function with respect to the weights. By the chain rule

$$-\nabla E(o(b_j)) = -1 \cdot \frac{\delta E(o(b_j))}{\delta a(b_j)} \cdot \frac{\delta a(b_j)}{\delta \alpha(b_i, b_j)} \quad [2.4.1.2.11]$$

2. The second term above finds the derivative of the activation level function with respect to the weights as follows

$$\begin{aligned} \frac{\delta a(b_j)}{\delta \alpha(b_i, b_j)} &= \frac{\delta}{\delta \alpha(b_i, b_j)} \cdot \sum_{b_m \in B_i} o(b_m) \cdot \alpha(b_m, b_j) \\ &= o(b_i) \end{aligned} \quad [2.4.1.2.12]$$

3. The first term above finds the change in the gradient with respect to the activation level function. Once again by the chain rule we have

$$-1 \cdot \frac{\delta E(o(b_j))}{\delta a(b_j)} = -1 \cdot \frac{\delta E(o(b_j))}{\delta o(b_j)} \cdot \frac{\delta o(b_j)}{\delta a(b_j)} \quad [2.4.1.2.13]$$

4. The second term in 2.4.1.2.12 is dependent on the actual output signal function that is used. The solution to the first term will depend on the layer the neuron belongs to. For example, if the neuron is an output layer cell, the term is just the derivative of the criterion function 2.4.1.3.9 above, with respect to the output signal function term as follows

$$\frac{\delta E(o(b_j))}{\delta o(b_j)} = -(T(b_j) - o(b_j)). \quad [2.4.1.2.14]$$

If the neuron belongs to a hidden layer, this relationship is less obvious. We want to make use of the error terms at the next layer up. Therefore, we are interested in knowing how the error term varies with changes in the output of b_j . The error term of the b_j node will depend on the sum of the error terms of the b_k nodes it is connected to

$$\begin{aligned} \frac{\delta E(o(b_j))}{\delta o(b_j)} &= \sum_{b_m \in B_k} \frac{\delta E(o(b_j))}{\delta o(b_j)} \\ &= \sum_{b_m \in B_k} \frac{\delta E(o(b_j))}{\delta a(b_m)} \cdot \frac{\delta a(b_m)}{\delta o(b_j)} \\ &= \sum_{b_m \in B_i} \frac{\delta E(o(b_j))}{\delta a(b_m)} \cdot \frac{\delta}{\delta o(b_j)} \sum_{b_n \in B_k} \alpha(b_n, b_m) o(b_n) \\ &= \sum_{b_m \in B_k} \frac{\delta E(o(b_j))}{\delta a(b_m)} \cdot \alpha(b_j, b_m). \end{aligned} \quad [2.4.1.2.15]$$

5. Finally, we can substitute the results of steps 2 - 4 back into 2.4.1.2.9 to find the negative gradient. There are two solutions due to step 4.

a) For output layer neurons

$$-\nabla E(o(b_j)) = (T(b_j) - o(b_j)) \cdot \frac{\delta o(b_j)}{\delta a(b_j)},$$

which is the same as learning rule 2.4.1.2.3.

b) For hidden layer neurons

$$-\nabla E(o(b_j)) = \frac{\delta o(b_j)}{\delta a(b_j)} \sum_{k \in B(j+1)} \delta(b_k) \cdot \alpha(b_j, b_k),$$

which is the same as learning rule 2.4.1.2.5.

The Problem Of Local Minima

As we have already stated, the back propagation criterion function does not present a smooth landscape for gradient search. As we move our weight vectors across the criterion function surface, we will encounter local hills and valleys. The negative gradient points in the local direction of steepest descent, so it is quite possible that we will move to the bottom of a local valley rather than a global minimum (zero). If η is very small and we are taking tiny steps, even small local minima will be a problem. On the other hand, if we make η too large, we could over-shoot the solution set in weight space. Local minima always present a problem when gradient search is used. However, a number of **avoidance heuristics** have been developed which greatly improve the chances that a global minimum will be found.

The back propagation algorithm employs a very simple avoidance heuristic called **weight vector momentum**. For this heuristic, we imagine that the learning algorithm is represented by a marble rolling on the criterion function surface. Gravity (the learning rules) makes the marble roll in the direction of steepest descent — the negative gradient direction. If the marble has mass, momentum will make it continue to roll in the same direction. This momentum will tend to make the marble escape shallow depressions in the surface — it will roll up the side of a shallow depression and over the top. Of course if the depression is deep, the marble will not be able to climb all the way out before it begins to roll back again. It will eventually come to rest at the global minimum. As long as the local minima are reasonably shallow, the marble will tend to find the global minimum. The back propagation algorithm makes use of this analogy by introducing a momentum term into the learning rules that tends to make the weight vector continue moving in the same direction. The change in the weight vector at time t denoted by $\Delta\alpha(b_i, b_j)_t$ is just the negative gradient plus fraction ρ of the previous change $\Delta\alpha(b_i, b_j)_{t-1}$

$$\Delta\alpha(b_i, b_j)_t = \eta (-\nabla E(o(b_j))) o(b_i) + \rho \Delta\alpha(b_i, b_j)_{t-1} \quad [2.4.1.2.16]$$

2.4.2. Auto-Association and Pattern Completion Algorithms

In this section we will look at two closely related auto-association networks.

2.4.2.1. Hopfield Networks

At the lowest architectural levels, brain regions do not exhibit layering, banding or other fine structure. Instead, they appear to be dense, randomly interconnected networks of neurons. Within the local neighbourhood of a neuron (a radius of about 5,000 neurons), the network is virtually fully connected. In 1982, John Hopfield published a theoretical study of the behaviour of a simple class of asynchronous symmetric networks with a fully connected architecture. Hopfield was not interested in the learning behaviour of these networks. His interest was in the **dynamical behaviour** of the network given an arbitrary set of weights. Suppose that we have a network with N neurons. We can define an N -dimensional **network state space** with one axis for each neuronal output signal. At time t , the network is represented by a single point in this space, whose coordinates are the set of output signals generated by the network at that instant in time. We will refer to this point as the **instantaneous state of the network**. As neurons compute their activation level and output signal functions, the behaviour of the network is described by the curve that the changing state of the network traces. These curves are what we mean by the dynamical behaviour of the network. Hopfield wanted to know if a network, left to run unattended, would generate curves that oscillate through the same states, or whether it would find a stable state and remain in that state indefinitely. Hopfield was able to characterize the dynamical behaviour of these primitive networks [Hopfield, 1982]. He found that they exhibit a complex and potentially very useful emergent behaviour that we will discuss shortly. This is another example of a very simple organizational law that generates a rich global behaviour in a neural network. Hopfield's initial work has been extended to asymmetric networks as well, with the same qualitative results.

The Network Model

The simplest type of Hopfield network is a fully connected symmetric architecture as shown in Figure 2.29. The network processors are referred to by the set

$$H = \{h_1, h_2, h_3 \dots h_{nc}\}.$$

There are no specialized sensory cells in the network. It is assumed that the user can set the initial activation level values of all neurons. In addition, the user is allowed to set the network weights to any values desired. We are only interested in the dynamical behaviour of the network. Connections occur between all neurons in the network. We shall refer to the strengths of connections between neuron h_i and h_j by $\alpha(h_i, h_j)$. Note however, that Hopfield assumes $\alpha(h_i, h_j) = \alpha(h_j, h_i)$, to simplify the analysis.

Figure 2.29 Hopfield Architecture

Unlike the preceding network models, Hopfield networks are updated asynchronously. At each instant in time, a neuron is selected at random, and its activation level and output signal are calculated. At the next instant in time, another neuron is randomly selected and so on as long as the network runs. Each time a neuron is selected, there is a chance that the network state might change. If the neuron changes state (changes its output signal), the network state changes.

The Neuron Model

Hopfield neurons are simple thresholding neurons like those in the perceptron model. The activation level is the sum of modified input signals less the firing threshold. The output signal is a simple binary thresholding function. If the activation level is positive, the neuron outputs 1.0 otherwise it outputs 0.0.

$$a(h_j) = \left(\sum_{i=1}^{N(H)} o(h_i) \cdot \alpha(h_i, h_j) \right) - \Theta(h_j) \quad [2.4.2.1.1]$$

$$o(h_j) = \begin{cases} 1.0 & \text{if } a(h_j) > 0 \\ 0.0 & \text{if } a(h_j) \leq 0 \end{cases} \quad [2.4.2.1.2]$$

Network Behaviour

Hopfield discovered that, regardless of the weight values, this class of network always changes state in a way that minimizes the equation

$$E(H) = -\frac{1}{2} \sum_{i=1}^{N(H)} \sum_{j \neq i} o(h_i) \cdot o(h_j) \cdot \alpha(h_i, h_j). \quad [2.4.2.1.3]$$

Hopfield likened this minimization behaviour to the entropy behaviour of Ising spin-glass models, and — unashamedly mixing metaphors — called this equation the **energy of the network**. Since the energy depends on the output signals produced by the network neurons, we can add an energy axis to our network state space and imagine an energy surface that varies with the states. Given a random starting position in state space, the network always moves in a direction that minimizes the energy value. This is analogous to a marble rolling downhill on the energy landscape⁴¹ (see Figure 2.30). Once again we have a neural network behaviour that can be explained in terms of gradient search.

Figure 2.30 The Energy Landscape of a Hopfield Network

⁴¹ Strictly speaking, this landscape is a discrete hypercube because of the binary thresholding output signal function. We will represent the energy landscape as a continuous surface because it is easier to picture, and because Hopfield has shown that his results extend to continuous valued output functions.

To see why this is so, consider the change in energy that results from a random neuron, h_i , changing its output signal. The difference in $E(H)$ that occurs if $o(h_i)$ changes from $o(h_i)'$ to $o(h_i)''$ is

$$\Delta E(H) = -\Delta o(h_i) \sum_{j \neq i} o(h_j) \cdot \alpha(h_i, h_j) = - (o(h_i)'' - o(h_i)') \sum_{j \neq i} a(h_j).$$

Given the output signal rule this equation will always be negative, indicating a decrease in $E(H)$ with every state change. If $a(h_i)$ is positive, and we are changing state then $o(h_i)'$ must be 0.0 and $o(h_i)''$ must be 1.0. Substituting, we get a negative change in $\Delta E(H)$. If $a(h_i)$ is negative, and we are changing state then $o(h_i)'$ must be 1.0 and $o(h_i)''$ must be 0.0. Substituting, we get a negative change in $\Delta E(H)$ once again. If the state doesn't change, $\Delta E(H)$ will remain constant.

This is a remarkably useful result. By arranging the energy landscape in the appropriate way, we can control the behaviour of the network. For example, we can 'remember' binary patterns (e.g. digitized pictures, control signals, etc.) by locating energy minima at network states that correspond to the appropriate binary pattern. If the network starts processing with a state similar to a stored state (near a minima), it will fall into the minima, and stabilize on the stored pattern. In this way, corrupted or incomplete versions of the stored patterns can be used to 'recall' the uncorrupted versions. Hopfield and others have employed this behaviour as an associative memory [Hopfield, 1982; Hinton & Sejnowski, 1985]. Of course, any other problem that can be reduced to this type of minimization behaviour can be solved by a Hopfield network (e.g. travelling salesman problems, etc).

A problem can occur if two stored patterns are quite close in state space. As one might expect, their minima will interfere, producing a single large minimum that makes it difficult to distinguish between the two stored patterns. Consequently, the memory works best when patterns are uniformly (or randomly) distributed over state space. Hopfield has shown empirically that on average, $0.15 \cdot N(H)$ random patterns can be stored without significant interference. Many modified versions of Hopfield networks have been proposed that significantly increase storage capacity.

Another problem that we have already seen in the context of the back propagation algorithm is the problem of local minima. Arbitrary energy landscapes are rarely smooth. As the algorithm descends along the landscape, there is a high probability of encountering a local trough and stabilizing there. Neural learning systems that make use of Hopfield networks must, therefore, incorporate an avoidance heuristic.

The precise computational behaviour of a Hopfield network depends on the shape of the energy landscape. This shape is dependent on the set of network connection strengths (ie. the weight term in 2.4.2.1.3). Therefore, we see that the problem of learning useful system behaviours is once

again reduced to finding the right set of weights. Hopfield has proposed several techniques for correctly positioning minima. However, these generally require that all the patterns be available at once, and that the connections be calculated globally. This kind of hand crafted weight matrix is not useful in a distributed general purpose learning system that must make incremental changes from sensory observations. We will turn instead to a neural learning scheme developed for Hopfield networks called the Boltzman machine.

2.4.2.2. The Boltzman Machine

In 1983, G.E. Hinton and T.J. Sejnowski proposed a learning algorithm for Hopfield networks called the **Boltzman machine**. As we shall see in the analysis section, the name is derived from the Boltzman distribution.

Binary patterns taken from the environment are provided as training data. In Hinton and Sejnowski's terminology, each bit in a training pattern represents a **predicate hypothesis** about the environment. For example, in a medical diagnosis domain, the first bit might represent the presence or absence of fever, the second bit might be an indication of vomiting, and so on for other symptoms. Presumably, some of the bits would also represent the presence or absence of a particular disease, given those symptoms. One of these disease bits, pre-determined by the teacher, would be set to the true state for each training sample. A subset of the network neurons provide an interface to the environment. Each training pattern bit is associated with a neuron from this subset. When a training pattern is presented, each neuron's output signal is clamped to the appropriate binary value. As training progresses and weights change, energy minima are generated for those states that capture pattern regularities in the training data.

Later, when some of the interface neurons are clamped by the user, the system completes the rest of the pattern by falling into the energy minima nearest the starting state. For example, if the user clamps a set of symptoms, the system will complete the pattern by filling in a disease bit associated with those symptoms. In terms of the energy landscape, the system has learned that an energy minimum exists at the state associated with the completed symptom-disease pattern. Any initial state with the symptom sub-pattern will tend to be located on or near the slope of the minimum well, and the system will 'fall into' the stable, low energy symptom-disease state. Moreover, because of the stochastic nature of the algorithm, the system will reproduce patterns with the same probability of occurrence as observed in the training data. For example, if symptom sub-pattern X has 80% likelihood of association with disease A, and 20% likelihood of association with disease B, then the system will tend to complete the pattern for disease A 80% of the time and disease B 20% of the time. If, during training, two diseases were just as likely to have the same (or

similar) symptoms, the energy minima will occur at a state in which both disease bits are true. In this case the completed pattern will indicate that both diseases are possible given the symptoms.

It has been shown empirically that the learning ability of the Boltzman machine is improved by increasing the number of non-interface neurons [Hinton & Sejnowski, 1986a]. Adding hidden neurons increases the number of connection weights, and gives the algorithm finer control of the energy landscape shape. As yet, there is no formal theory to help the designer select the minimum number of neurons necessary for a given problem domain.

There are many areas in which the Boltzman machine has potential application. Research has been conducted in fields as diverse as: machine vision [Gmitro & Gundi, 1987], robot manipulator control [Tsutsumi, et. al., 1987], speech recognition [Bridle & Moore, 1984], knowledge representation [Rumelhart, et. al. — Volume 2, 1986b], sentence understanding, expert systems [Treurniet & Croft, 1986], pattern recognition, and biological modeling [Harth, et. al., 1987]. Variations on the Boltzman machine algorithm have been developed which learn faster in some problem domains; e.g. the Cauchy machine.

The Network Model

Boltzman machines are simple Hopfield networks. They have a fully connected, symmetric architecture as shown in Figure 2.29. The network processor set is

$$M = \{m_1, m_2, m_3 \dots m_{nc}\}.$$

This set is usually divided into two subsets. The set $I(M)$ represents the interface neurons for network M , and the set $H(M)$ represents the remaining hidden neurons. It is possible that $H(M) = \emptyset$, if there are no hidden neurons. Connections occur between all neurons in the network. We shall refer to the strengths of connections between neuron m_i and m_j by $\alpha(m_i, m_j)$. These connections are symmetrical, such that $\alpha(m_i, m_j) = \alpha(m_j, m_i)$.

It is sometimes convenient to describe the network in terms of a knowledge representation data structure. From this viewpoint, neurons are predicate (true-false) hypotheses about the learning domain. The binary neural output signals represent the values true and false. Connections between neurons represent constraints between hypotheses. A positive connection strength between hypotheses m_i and m_j means that the two hypotheses support one another. Thus, m_j is true if m_i is true ($m_i \Rightarrow m_j$) and m_j is false if m_i is false ($\neg m_i \Rightarrow \neg m_j$), and vice versa. An m_i negative connection strength between hypotheses m_i and m_j means that the two hypotheses negatively support one another: m_j is true if m_i is false ($\neg m_i \Rightarrow m_j$) and m_j is false when m_i is true ($m_i \Rightarrow \neg m_j$), and visa versa. Hidden neurons represent domain micro-features that the learning system discovers from

regularities inherent in the training samples. These micro-features represent second and third order hypotheses⁴² about the domain, increasing the system's knowledge representation capability.

Boltzman machines are updated asynchronously.

Figure 2.31 Boltzmann Machine Architecture

The Neuron Model

Boltzman machine neurons are quite similar to Hopfield neurons. The activation level is still the sum of modified input signals less the firing threshold. The output signal remains a binary

⁴² That is, hypotheses other than user supplied interface hypothesis. These are predicates that express relationships between the interface hypothesis and other 'hidden' hypothesis. For example, connections between interface hypothesis X, Y, Z, P, Q, R might express the relationships $X \Rightarrow Y$, $R \Rightarrow Z$ and $P \Rightarrow Q$ while hidden neurons and their connections would express more complex relationships like $\neg Q \Rightarrow (R \ \& \ \neg Y)$.

thresholding function. However, it acquires the value 1.0 with probability $p(m_j)$. We will discuss this output function further in the learning behaviour section below

$$a(m_j) = \left(\sum_{i=1}^{N(H)} o(m_i) \cdot \alpha(m_i, m_j) \right) \cdot \Theta(m_j) \quad [2.4.2.2.1]$$

$$p(m_j) = \frac{1}{1 + e^{-a(m_j)/T}} \quad [2.4.3.2.2]$$

where T is the system temperature

$$o(m_j) = \begin{cases} 1.0 & \text{with probability } p(m_j) \\ 0.0 & \text{with probability } 1 - p(m_j) \end{cases} \quad [2.4.2.2.2]$$

The Simulated Annealing Avoidance Heuristic

Hopfield networks suffer from the local minima problem because the energy landscape usually contains many shallow minima not associated with stored information. In 1983, S. Kirkpatrick, then at IBM, published a probabilistic avoidance heuristic for gradient search problems. The basic idea is quite simple — instead of always moving down the energy landscape, occasional uphill steps are allowed which can carry the system out of shallow depressions.

Occasional uphill steps slow the gradient search algorithm because the system tends to wander further before finding a global minimum. Working with C.D. Gelatt and M.P. Vecchi, Kirkpatrick found that the fastest way to find a global minimum was to start the search with a high probability of uphill steps and then gradually lower the probability to 0. He compared this process to **annealing** in which a metal is heated and then gradually cooled according to an **annealing schedule**. This process strengthens the metal by forcing it into a very low energy state. The high temperature state of the metal is analogous to the high probability of uphill steps and the annealing schedule is analogous to the gradual lowering of this probability. These similarities prompted Kirkpatrick and his associates to name their heuristic **simulated annealing** [Kirkpatrick, et. al., 1983].

Figure 2.32 depicts an energy landscape with a shallow local minimum A and a deep global minimum B. The hump between A and B is called an **energy barrier**. Assume that our system has a low, constant probability of making an uphill step. A sequence of uphill steps will allow the system to climb some distance up the energy barrier. A short sequence of uphill steps is more probable than a long sequence, so the further the system has to climb, the less likely it will get there. Notice that from the point of view of A, the distance it has to climb to cross the energy barrier is shorter than the distance B has to climb. Therefore, there is a much greater probability that the system will cross the energy barrier from A to B than from B to A. This is why the system

tends to end up at the global minimum B. This process can take a long time if the probability of uphill steps is small. Annealing attempts to reduce the time needed by making uphill sequences very likely. Unfortunately, this makes it less likely that the system will wind up at B since it will be almost as easy to cross from B to A as from A to B. The annealing schedule allows a good compromise to be found. By lowering the probability slowly we will pass through the point of optimal compromise between speed of uphill sequences and probability of winding up at B. Stopping at a low probability level is also good because a system that has found B will tend to stay there.

Figure 2.32 Local Vs Global Minima

As we have already seen, the direction of movement along the energy landscape is controlled by the output signal thresholding function. In order to make occasional uphill steps we introduce probability $p(m_j)$. This is yet another example of a sigmoidal function. Figure 2.33 illustrates how a 'temperature' parameter τ affects the shape of this function. Continuing with the physical system analogy, at high temperatures the probability that we will make an uphill step (choose output signal 0.0 instead of 1.0, or 1.0 instead of 0.0) remains relatively high for most activation values. At lower temperatures, the function has a sharp probability transition at the firing threshold, and it behaves exactly like the familiar Hopfield output signal function 2.4.2.1.2. The value of τ starts

high and is slowly lowered. Empirical studies have shown that this heuristic works best in learning domains where the energy barriers tend to be very high [Hinton & Sejnowski, 1986a].

(a) System Temperature High

(b) System Temperature Low

Figure 2.33 The Boltzmann Probability Function

The Learning Algorithm

The Boltzman learning algorithm is our first example of an unsupervised learning scheme. A teacher is not required to correct the network when it outputs signals. The algorithm proceeds in 5 steps:

1. $\forall i,j$: the user initializes firing threshold $\Theta(m_j)$ and weight $\alpha(m_i, m_j)$ to random, real-valued numbers,
2. ANNEAL WITHOUT INPUTS

e.g. annealingSchedule := (40, 35, 30, 25, 20, 15, 12, 10);

For τ := each temperature in the annealingSchedule Do

For i := 1 to 2 Do

Randomly select a neuron m_j from M until each neuron has been selected an average of one time. For each selection Do

calculate $a(m_j)$, $p(m_j)$ and $o(m_j)$;

$\tau :=$ the lowest temperature in the annealingSchedule;

For $i := 1$ to 10 Do

Randomly select a neuron m_j from M until each neuron has been selected an average of one time. As neurons are updated, for each pair of neurons m_x and m_y in M , compute the probability $p(m_x, m_y)$ that m_x and m_y both output 1.0 at the same time.

3. ANNEAL WITH INPUTS

the annealingSchedule in step 2 is used as follows:

For each training sample Do

A training sample arrives on the input channel and clamps the $I(M)$ neurons,

For $\tau :=$ each temperature in the annealingSchedule Do

For $i := 1$ to 2 Do

Randomly select a neuron m_j from M until each neuron has been selected an average of one time. For each selection Do

calculate $a(m_j)$, $p(m_j)$ and $o(m_j)$;

$\tau :=$ the lowest temperature in the annealingSchedule;

For $i := 1$ to 10 Do

Randomly select a neuron m_j from M until each neuron has been selected an average of one time. As neurons are updated, for each pair of neurons m_x and m_y in M , compute the probability $p_T(m_x, m_y)$ that m_x and m_y both output 1.0 at the same time.

4. UPDATE WEIGHTS

$\forall m_i \in M, \forall m_j \in M: \alpha(m_i, m_j)$

$$\alpha(m_i, m_j) = \alpha(m_i, m_j) + 5(p_T(m_i, m_j) - p(m_i, m_j)) \quad [2.4.2.2.3]$$

5. goto 2.

The algorithm repeats until the teacher is satisfied that the network can correctly complete all the training sample data. 10,000 repetitions of steps 2 to 4 may be required⁴³. There is currently no formal analysis that relates the learning task to the number of iterations required.

Learning Behaviour Analysis

Like the perceptron and back propagation learning algorithms, the Boltzman machine algorithm is an example of gradient search on a criterion function surface. In order to understand how the algorithm is derived, we must understand the relationship between the probability of network states and the network's connection weights.

The probabilistic nature of output signal function 2.4.2.2.2 causes the network to continue changing states even if no inputs are being supplied by a teacher. It has been shown [Hinton & Sejnowski, 1986a; Ackley, et. al., 1985] that at a given temperature, the relative probabilities $P(M_{s1})$ and $P(M_{s2})$ of two network states M_{s1} and M_{s2} is given by the difference between their system energies

$$\frac{P(M_{s1})}{P(M_{s2})} = e^{-(E(M_{s1}) - E(M_{s2})) / \tau} \quad [2.4.2.2.4]$$

This equation follows the Boltzman distribution — hence the name of the model. From 2.4.2.2.4. we see that the network state is related to the network energy and by 2.4.2.1.3 we know that the network energy is related to the synaptic weights $\alpha(m_i, m_j)$. These equations allow us to derive the way in which the probability of a network state changes as the weights are changed

$$\frac{\delta \ln P(M_{s1})}{\delta \alpha(m_i, m_j)} = \frac{1}{\tau} \left(o(m_i)_{s1} \cdot o(m_j)_{s1} - \frac{\sum_{k=1}^{2^N(M)} P(M)_{sk} \cdot o(m_i)_{sk} \cdot o(m_j)_{sk}}{\sum_{k=1}^{2^N(M)} P(M)_{sk}} \right) \quad [2.4.2.2.5]$$

The approach taken by Sejnowski and Hinton is to find a set of network weights such that, when no inputs are applied to the network, equation 2.4.2.2.2. will continue to generate the same states that the network was forced into by the teacher. Moreover, the network should generate these states with the same probability of occurrence as in the training data. Any knowledge representation system that generates patterns with the same probability as the environment is called a **stochastic generative model** of the environment. As the weights define a space of possible

⁴³ For example, Sejnowski and Hinton taught a 43 neuron Boltzman machine to act as an 8-bit binary shifter in 9,000 repetitions. Training patterns consisted of the unshifted pattern concatenated with the shifted pattern. A maximum accuracy of 86% was obtained. On other problems, an accuracy of 99.9% has been obtained.

stochastic generative models according to 2.4.2.2.5, we need only find a set of weights that gives us a 'good' model⁴⁴.

Once again we can make use of the technique of gradient search. This time the criterion function must be a measure of the 'goodness' of a stochastic generative model. A perfect stochastic generative model is one in which the probability distribution of states automatically generated by the network (no inputs) $P(M)$ is exactly the same as the distribution of states generated when the network is clamped by training sample data $P_T(M)$

$$P(M) = P_T(M).$$

Luckily, there is a measure of the distance between two probability distributions due to S. Kullback [Kullback, 1959]. This measure

$$G = \sum_{k=1}^{2^{N(M)}} P_T(M)_{sk} \cdot \ln \frac{P_T(M)_{sk}}{P(M)_{sk}} \quad [2.4.2.2.6]$$

is called the **information gain** or asymmetric divergence. G is a good candidate for our criterion function because it is zero when the distributions are identical and positive otherwise. Sejnowski and Hinton have shown that all the information needed to perform gradient descent on this function relative to a given weight $\alpha(m_i, m_j)$ is available in the correlated behaviour of the output signals of m_i and m_j . Since each neuron can compute the probability that it is active at the same time as its neighbour, the gradient search algorithm can be implemented in a distributed fashion. In particular Sejnowski and Hinton have shown that the change in the information gain as the network weights are changed is given by

$$\frac{\delta G}{\delta \alpha(m_i, m_j)} = -\frac{1}{\tau} (p_T(m_i, m_j) - p(m_i, m_j)), \quad [2.4.2.2.7]$$

where $p_T(m_i, m_j)$ is the probability that neurons m_i and m_j are both outputting 1.0 while the teacher is clamping the network with training sample patterns, and $p(m_i, m_j)$ is the same probability when the network is running freely without any inputs. The learning algorithm is composed of three stages: (1) collect $p(m_i, m_j)$ statistics at each node while inputs are not being applied, collect $p_T(m_i, m_j)$ statistics while inputs are being applied, and (3) modify the network weights according to 2.4.2.2.7.

⁴⁴ Statisticians call this process maximum likelihood estimation. The Boltzman learning algorithm is similar to the Expectation and Maximization (EM) method by Dempster, Laird and Rubin, 1976, and a speech recognition algorithm by Bahl, Jelinek and Mercer, 1983. These methods estimate missing parameters for probability distributions.

2.4.3. Structure Mimicry Algorithms

In this section we shall consider the most important example of the structure mimicry algorithms due to Kohonen.

2.4.3.1. Kohonen's Topology Preserving Maps

As we saw in sections 2.2.1.2. and 2.2.1.7, cortical and sub-cortical sheets are often organized as topographically ordered maps. The response characteristics of neurons in these regions spatially duplicate the structure of our sensory and motor domains. The remarkable thing is that we simply don't have enough genetic material to encode the precise topographic ordering of tens of billions of neurons and their connections, or the detailed ordering of sensory cell axons that project to the brain. Also, there is strong evidence that changing sensory spaces are accommodated for by neural maps during adulthood⁴⁵. These facts suggest that some type of on-going self-organizing mechanism is at work which organizes cortical maps after birth. Presumably our sensori-motor experiences are the training samples that drive this mechanism.

In 1982 Teuvo Kohonen of the Helsinki University of Technology suggested a simple neural model to explain the topographic self-organization of brain maps. This model has become known as **Kohonen's topology preserving map**, **Kohonen's self-organizing map** or simply **Kohonen's algorithm**⁴⁶. The approach has also been suggested as a model for the formation of the cortical bands described in section 2.2.1.3, and has found real-world application in speech to text systems [Kohonen, et. al., 1984a; Kohonen, 1984b]. A highly accurate pattern classification system has also been derived from the model [Kohonen, 1986a] which comes close to the theoretical Bayes classifier limit. A significant advantage of Kohonen's classifier is that it does not need to store training samples. The Kohonen algorithm is also an important element of the adaptive robot arm controller described in chapter 4.

Lateral Feedback and Cortical Bubbles

As we saw in section 2.2.2., it is not unreasonable to model the local neighbourhood of a neuron as a fully connected network. If we expand our view slightly, we will find that the connections between neurons are actually limited by distance. Most connections made by a given

⁴⁵ For example, the somatosensory cortex contains a map of the touch sensors in a monkey's hand. If the nerve from the monkey's middle finger is severed — ie. the somatosensory input space is changed — the cells associated with the finger will gradually start to respond to sensations from the two adjacent fingers. Thus the cortical sheet adjusts to the new space.

⁴⁶ In fact, Kohonen has developed a number of neural algorithms, including an associative memory [Kohonen, 1984c, 1976] and a device called a novelty detector that produces an indication of those aspects of an input pattern that do not correspond to previously observed patterns [Kohonen, 1984c]. The novelty detector has been used as a medical diagnostic device to automatically highlight tumors in chest X-rays.

cell occur within a radius of 5,000 neurons or so. We will refer to these as the **lateral connections** of the cell⁴⁷. Figure 2.34a depicts a highly schematic representation of a slice through the cortical sheet and the connections that occur between neurons. Notice that there is a high degree of **lateral feedback** throughout the cortical sheet.

(a) Lateral On-Center Off-Surround Connections

⁴⁷ In reality, connections do not just occur laterally (to the side) but actually occur in a three dimensional volume around the cell. We will make the simplifying assumption that the network is a perfectly flat plane for the remainder of this section.

(b) The Mexican-Hat Function Describes Lateral Interaction Among Neurons

Figure 2.34 On-Center Off-Surround Lateral Connections

There is evidence to suggest that synaptic connections made by a neuron to its immediate neighbours — and itself — are excitatory. This **on-center excitatory field** probably extends for a radius of 50 to 100 micrometers. The synaptic connections made to neurons surrounding this field are inhibitory and are called the **off-surround field**. This field extends out around the on-center field in a penumbra 100 to 500 micrometers in radius. Kohonen calls this distribution of excitatory vs inhibitory signals the **Mexican hat function** for reasons that should be made apparent by Figure 2.34b. There is evidence for this sort of connectivity pattern throughout the nervous system⁴⁸. Kohonen and others [Ellias & Grossberg, 1985] have modeled the behaviour of simple networks of perceptron-like neurons connected in this fashion. They have found that by starting with a random distribution of output signals, laterally connected neurons tend to form stable clusters of activity after enough time steps have passed. Figure 2.35 illustrates this process for a simple two-dimensional network. The first frame represents the random distribution of output signals over a square 100 neuron network. Neurons are represented by circles of varying sizes — the larger the output signal of the neuron, the larger its circle. As time passes, clusters of highly

⁴⁸ A weak excitatory ring usually surrounds the off-surround field but is ignored by Kohonen.

excited neurons tend to form around the neurons that had the highest initial output signals. The clusters do not overlap and are distributed fairly evenly over the network. Fatigue of neurons under repeated firing eventually causes the clusters to shrink and reform elsewhere where the local output signal is higher. This on-going process of **lateral on-center-off-surround competition** causes clusters of activity to bubble over the surface of the network. Appropriately enough, Kohonen refers to the clusters as **activity bubbles**. The diameter of an activity bubble is determined by the radius of lateral connectivity and by the ratio of excitatory to inhibitory connections. A large on-center field creates larger bubbles. A large off-surround field creates smaller bubbles. The special case where the bubble consists of a single neuron is called a **winner-take-all network**.

Figure 2.35 Stable Bubbles of Activation

As we shall see, Kohonen employs the idea of activity bubbles that appear and shrink at points of highest neural activity in his self-organizing algorithm. Kohonen's model does not actually perform lateral on-center-off-surround competition however. In order to simplify the model Kohonen simply assumes that a single bubble of activity surrounds the neuron that 'responds the best' to a training sample. The meaning of 'responding best' to an input will be made clear in a moment. The neuron that is the center of the bubble of activity for a given training sample is said to have **won the competition**. It is not important to the model how this bubble is found. The only connections received by Kohonen neurons are input connections from sensory cells. As training progresses, the radius of new bubbles gradually become smaller until only one neuron is included in the bubble.

Topology Preserving Behaviour

Before we examine the model, let's have a qualitative look at what the network does. Neurons in a Kohonen network are very similar to the perceptron neurons described in section 2.4.1.1. As

in the perceptron model, every Kohonen neuron receives the same pattern of input signals. We will think of these patterns as feature vectors as we did for the perceptron. We will also think of the vector of input connection weights as a weight vector in input space. Kohonen neurons output a signal that is proportional to the Euclidean distance between the neuron's weight vector and the feature vector it has received. In a sense, Kohonen neurons act like feature detectors, since they will output the smallest⁴⁹ signals for inputs that represent feature vectors in the neighbourhood of the neuron. If neuron weight vectors are spread around the input space, particular neurons will be sensitive to particular kinds of input patterns.

Figure 2.36a depicts the relationship between a physical network in the real world, and the set of weights vectors that it maps to in input space. Notice that if we draw connecting lines between weight vectors in input space that are neighbours in the physical network, the spacial ordering of the network (ie. the ordered, grid-like, planar topology of the network) is not necessarily preserved in the input space (ie. the weights do not form the same ordered grid in input space). Let's call this mapping from the physical network to the weight vectors in the input space the **weight vector image** of the network. As the Kohonen learning algorithm proceeds two things happen.

First, the weight vectors in input space rearrange so that **the weights of neurons that are neighbours in the physical network become neighbours in the input space** [Kohonen, 1987, 1986a, 1984c]. As Figure 2.36b shows, the grid-like topology of the network gradually appears in the weight vector image. Alternatively, one can perform the reverse mapping and think of the topology of the input space being preserved in the response characteristics of the physical network. It is this **response characteristic ordering** that we observe in the brain. If the physical network is of a lower dimension than the input space, its weight vector image in the input will preserve the topology of the network in the input space, but it will also bend and twist in order to fill up as much of the input space as possible. Figure 2.36c shows the initial weight vector image of a one-dimensional network (a string) of neurons and a typical ordering of the image after learning. This ordering has been compared to Peano curves [Kohonen, 1984c].

⁴⁹ Or largest value if the reciprocal of this distance is used (times a small constant), it does not matter as long as the metric used is consistent.

(a) Mapping the Network into Input Space

(b) Weight Vector Relationships Preserve the Network's Topology in the Input Space (2-D Net, 2-D Space)

(c) The Network and Spatial Dimensions Do Not Have to Match (1-D Net, 2-D Space)

Figure 2.36 The Topology Preserving Behaviour of the Network

The second thing that happens is that the local point density of the weight vectors in input space approximates the distribution of training samples [Kohonen, 1987, 1986a, 1984c]. In other words, if feature vectors are only taken from one part of the input space, the weight vectors will only be distributed in that part of the space. Also, if the density of feature vectors is greater in one part of the space, the density of training vectors will be greater there too. Figure 2.37 shows examples of localized distribution and varying density. This sort of behaviour nicely explains the distortions that occur in cortical topographic maps. For example, the somatotopic regions devoted to the tips of the fingers are much larger than other regions because more sensory data arrives from that region of the input space.

Figure 2.37

Weight Vectors Approximate Training Sample Distributions

The Network Model

As already stated, the Kohonen network is not actually laterally connected. Lateral connections are done away with in the model for mathematical expediency. The Kohonen network has a single layer, unconnected perceptron architecture as shown in Figure 2.38. The network processor set is denoted by

$$K = \{k_1, k_2, k_3, \dots, k_{nc}\}.$$

where $N(K) = n_c$ = the maximum number of neural cells. Inputs to the network arrive from a set of sensory cells

$$S = \{s_1, s_2, s_3 \dots s_{ns}\},$$

where $N(S) = n_c$ = the maximum number of sensory cells. We shall refer to the strengths of connections between the sensory neurons and Kohonen neurons by $\alpha(s_i, k_j)$.

Figure 2.38 Kohonen Architecture

The Neuron Model

Sensory cells produce real-valued outputs. Kohonen neurons compute the real-valued Euclidean distance from their weight vector $\alpha(k_j)$ to the incoming feature vector $o(S) = [s_1, s_2, s_3 \dots s_{ns}]$ as their activation level. Clearly the neuron that is closest to a given feature vector will have the smallest activation level. This is the neuron that wins the competition to be the center of the bubble of activity. The output signal is the same as the activation level.

$$a(k_j) = ||o(S) - \alpha(k_j)|| \quad [2.4.3.1.1]$$

$$o(k_j) = a(k_j) \quad [2.4.3.1.2]$$

The Learning Algorithm

The learning algorithm is another example of unsupervised learning. A teacher is not required to correct the system's output during learning. We will describe the algorithm as if a global mechanism exists to find the bubble of activation. The reader should assume that a distributed mechanism like lateral on-center-off-surround competition is actually responsible. Any distributed mechanism will do. As each training sample pattern is presented to the network, the winning neuron is found and the set A of neurons contained in the bubble of activity are identified. Only

2

ALLEN NEW FINE PRINTS
MicroD
100% COTTON 100% POLYESTER 50% COTTON 50% POLYESTER

those neurons in A apply the learning rule to their weights. This rule moves their weight vectors closer to the training sample vector, thus making those neurons respond better to similar patterns in the future. A function γ determines the amount that the weight vectors are moved. This value is quite large to start with, and then rapidly decreases over time. The learning algorithm proceeds in 8 steps:

1. $\forall i, j$: The user initializes the firing threshold $\Theta(k_j)$ and weight $\alpha(s_i, k_j)$ to random, real-valued numbers;

bubbleRadius := one half the minimum side of the network (in terms of neurons);

tick := 1;

2. A training sample arrives on the input channel,
3. $\forall j$: calculate $a(k_j)$ and $o(k_j)$,
4. Find the winning neuron k_w .
5. $A := \{\text{all neurons within radius bubbleRadius of } k_w\}$;
6. $\forall k_j \in A$: neuron k_j updates its weights according to the rule

$$\alpha(k_j) = \alpha(k_j) + \gamma(\text{tick}) \cdot (o(S) - \alpha(k_j)), \quad [2.4.3.1.3]$$

where $\gamma(\text{tick})$ is a function which decreases exponentially from 0.9 to 0.1 during the first 1,000 steps, and then linearly from 0.1 to 0.0 over the next 100,000 steps.

7. bubbleRadius := maximum((bubbleRadius - 1), 1);

tick := tick + 1;

8. goto 2.

The algorithm repeats until the user is satisfied that the weight distribution is topologically ordered and approximates the training sample distribution. There is no simple formal expression to indicate how many learning trials are required, but a good rule of thumb is that $500 \cdot N(K)$ presentations are required for statistical accuracy. A much smaller, representative set of training samples may be cycled through repeatedly if desired. Usually, good topology ordering is achieved if the bubble radius starts out covering half the entire network and then decreases to one node over the first 1,000 trials. The algorithm is more sensitive to the selection of γ . Best results seem to occur when γ starts out close to 1.0 and then decreases to 0.1 over the first 1,000 trials, and then to 0.0 over the next 100,000 trials or so. The algorithm is sensitive to the γ function though and some fine tuning is required by the user, which depends on the size of network.

Network Analysis

The Kohonen algorithm has proven notoriously difficult to analyse mathematically. The best effort so far is by H. Ritter and K. Schulten [Ritter & Schulten, 1986a] of the University of Munich. The simple proof given by Kohonen [Kohonen, 1982a] has been shown to be incorrect. Despite its intractability, the algorithm works well in the laboratory. We will only give a short intuitive explanation here that is derived from long hours of working with simulations of the algorithm. It is not based on a mathematical analysis however, and should be taken with that in mind.

There are two opposing mechanisms that lead to the complex behaviour of the algorithm. The constraint imposed by the first mechanism is that **winning neurons want to move toward their own region of input space**. When a neuron k_w wins, it does so because it is the closest neuron to a feature vector. The learning rule 2.4.3.1.3. moves it closer to this vector so that it will be even more likely to win the competition for similar vectors in the future. Every other neuron also tries to move to the region of space that it is initially closest to. As long as the distribution of training samples is uniform, the learning rule acts on the weight vectors like a uniform force on a set of particles. The moment that the training distribution becomes non-uniform however, the force exerted on particles by dense regions of the distribution will be higher than that exerted by less-dense regions. This imbalance will tend to pull particles toward the mode of these dense regions, forming clusters. As more weight vectors fall toward the mode, leaving behind empty regions of the input space, nearby weight vectors will drift toward these spaces. If this process continues, a disproportionately large number of weight vectors will be devoted to a small number of regions of space. The function γ helps to reduce this process. During the first few thousand trials, γ is large allowing weight vectors to become distributed over the space. As time passes however, the force exerted on weight vectors rapidly declines and they change position very slowly. If the correct value for γ is found, the weight vectors will not have time to completely collapse into regions of high density. The final distribution of weight vectors will be a good approximation of the training sample distribution. Indeed, the Kohonen algorithm is very sensitive to the initial value and decay rate of γ . Notice that so far we have said nothing about preserving the topology of the network in input space. This first mechanism is only responsible for the distribution of weight vectors.

The second, and opposing mechanism is caused by the activity bubbles. This mechanism imposes the constraint that **neurons want to remain next to their physical network neighbours in input space**. The effect of the activity bubble is such that, when a winning neuron k_w moves toward a feature vector, all of its immediate network neighbours are forced to move toward the same region of input space. Winning neurons drag their neighbours with them

when they move. The amount that a neuron moves toward a feature vector is proportional to its distance to the vector. Therefore, neighbours that are very far away across the input space will take large steps and will very rapidly move toward their winning neighbour. This happens simultaneously (more or less) across the entire network. The resulting 'neighbourhood force' is like the tension on a grid made of elastic bands that has been stretched in random ways. Once you let go, the elastic force of the grid will make it return to its original shape. The end result is that the originally crumpled and stretched weight vector image of the network 'unfolds' in input space. The final orientation of the image in input space (upside down, backwards, sideways, etc.) depends entirely on the initial random deformation caused by the selection of random weight vectors⁵⁰. The whole process of unfolding usually occurs in the first 50 to 500 trials.

Another important effect of the activation bubbles seems to occur after the network image has preserved the network topology by unfolding. At this point, when the first constraint moves a winning neuron toward a weight vector, all the neighbours around it tend to collapse toward the same vector. Imagine a shrinking disk of weight vectors. This process seems to help stop the rapid collapse of weight vectors toward dense regions in the training distribution. Although a weight vector is drawn toward this region, half of its neighbours (on the side toward the region) will be drawn away from the region (toward the winning neuron). This mechanism acts as a repulsive force that helps distribute the neurons in a region of high density more smoothly. Gradually, as the radius of the bubble shrinks, this repulsive force lessens (along with the γ function) until only fine changes occur in the distribution during the last 80% of the training run.

2.4.4. Feature Detection Algorithm

In this section we shall consider one of the first successful functional level feature detection algorithms — the cognitron.

2.4.4.1. The Cognition

One of the first, and most successful, functional level neural systems was developed by Kunihiko Fukushima of the NHK Science and Technical Research Laboratories in Tokyo, Japan. In 1975 Fukushima developed a model of the striate cortex⁵¹ hyper-column that he called a **cognitron**. Digitized visual patterns are presented to the cognitron by a two-dimensional retinal network. The output signal values from the retina represent grey scaled or colour pixels. As training progresses, cognitron cells become specialized feature detectors that are sensitive to sub-

⁵⁰ It is interesting to note that the orientation of the somatosensory homunculus is always the same in humans. This suggests that Kohonen's algorithm is certainly not a perfect explanation of the formation of these maps.

⁵¹ Brodmann area 17 is the part of the cortical sheet devoted to processing visual data.

patterns that appear in the training images. These sub-patterns appear regularly and are therefore useful pattern recognition primitives. Moreover, as the model is arranged in data-abstracting layers, cells within these layers become sensitive to higher and higher level patterns until cells in the last layer respond to individual 'objects' within the training images. This arrangement is very similar to the simple-cell $\Rightarrow$ complex-cell arrangement that occurs in the brain. The outcome is that the cognitron makes a good pattern recognition system that learns to recognize objects from pictures. Since his original work on the cognitron, Fukushima has refined the model into a series of networks — 'neo-cognitrons' — that accommodate position invariance of features, add associative memory capability to the image recognition process and introduce a rudimentary attentional system that can single out and attend to recognizable objects in a cluttered scene. The interesting thing about cognitron networks is that they learn to do all these things for arbitrary visual domains without a teacher's help.

In this section we shall look at Fukushima's basic cognitron model. Fukushima has also developed a neocognitron for real time, position invariant recognition of hand written characters [Fukushima, 1983].

The Network Model

The simple cognitron consists of n_l pairs of interleaved planar networks as shown in Figure 2.39. The first network in a pair is an n_x by n_y , laterally connected matrix of excitatory cells that we will refer to by cell type 'e'. The second network in a pair is an n_x by n_y unconnected matrix of inhibitory cells that we shall refer to by cell type 'i'. We will refer to the entire set of excitatory cells by

$$E = \{e_1, e_2, e_3 \dots e_{n_x \cdot n_y}\},$$

and excitatory cell layer k by E_k . Similarly, we will refer to the entire set of inhibitory cells by

$$I = \{i_1, i_2, i_3 \dots i_{n_x \cdot n_y}\},$$

and inhibitory cell layer k by I_k . The k^{th} pair of excitatory and inhibitory cell layers form a single cognitron layer C_k . It will occasionally be convenient to refer to a neuron by its row and column coordinates within a network in this way — $u_k(x,y)$.

Figure 2.39 Functional Level Cognitron Architecture

The first cognitron layer C_1 has a special sensory role: excitatory layer E_1 acts as an n_x by n_y planar retina in which the output signal of each neuron represents the grey scale or colour value of a pixel in a training image. The user represents a training sample image by clamping the neurons of E_1 . Every other excitatory layer E_k receives its inputs from neurons in the preceding excitatory and inhibitory layers E_{k-1} and I_{k-1} . We will refer to these as **inter-cognitron layer connections**. Connections between excitatory and inhibitory layers within a cognitron layer will be referred to as **intra-cognitron layer connections**. Lateral connections between neurons in an excitatory layer will be referred to as **intra-excitatory layer connections**.

Intra-cognitron layer connections are arranged as shown in Figure 2.40. Each inhibitory cell i_j in layer I_k receives fixed excitatory connections from excitatory cells in layer E_k . The region of gray cells in E_k that connect to i_j is called i_j 's **connectable area**⁵² $ca(i_j)$. This area is a square patch of cells⁵³ centered on excitatory cell e_j (notice this is the cell with the same (x,y) coordinates as i_j — ie. it is directly 'underneath' I_k cell i_j in the E_k network). The widths of all connectable areas in the cognitron are identical and fixed, although they should be one quarter to one half the width of the excitatory network. Actually, the connectable areas of cells just inside the border of I_k will not be square as they must terminate at the edge of the excitatory cell network. Clearly, the

⁵² In biological terms, one can think of this as the receptive field of the cell, ie. the region of E_k that it is sensitive to.

⁵³ Or 'circular' or any other closed geometric shape desired.

connectable areas of adjacent inhibitory cells overlap each other in E_k . Synaptic connections from excitatory cells are fixed and unmodifiable, and referred to as ' α ' type connections. We will denote the fixed connection strength between e_j and i_k by $\alpha(e_j, i_k)$. The weight of each α type synapse of i_k is given by

$$\alpha(e_j, i_k) = \frac{1}{N(ca(i_k))} \quad [2.4.4.1.1]$$

where $N(ca(i_k))$ denotes the number of excitatory cells in the connectable area of i_k .

Figure 2.40 Un-Modifiable Intra-Cognitron Connections

Intra-excitatory layer connections are lateral connections similar to those described in section 2.4.3.1. As shown in Figure 2.41, every excitatory neuron e_j in excitatory cell layer E_k receives inhibitory connections from its closest neighbours in E_k (gray neurons). These neighbours form a square patch centered on e_k called the inhibitory area of e_k , $ia(e_k)$. As with connectable areas, the widths of all inhibitory areas in the cognitron are identical and fixed and should be one quarter to one half the width of the excitatory network. Inhibitory areas overlap each other in E_k . Lateral synaptic connections are unmodifiable, and are referred to as ' β ' type connections. We will denote the fixed lateral connection strength between e_m and e_k by $\beta(e_m, e_k)$. The weight of each β type synapse of e_k is given by

$$\beta(e_m, e_n) = \frac{1}{N(ia(e_k))} \quad [2.4.4.1.2]$$

where $N(ia(e_k))$ denotes the number of excitatory cells in the inhibitory area of e_k .

Figure 2.41 Un-Modifiable Intra-Excitatory Layer Connections

Finally, inter-cognitron layer connections are arranged as shown in Figure 2.42. Connections to excitatory cell e_j (black) in cognitron layer C_k arrive from the excitatory and inhibitory layers in cognitron layer C_{k-1} . An inhibitory connection is made from the inhibitory cell in layer I_{k-1} (bold outline) that has the same (x,y) coordinates as neuron e_j in E_k . We will refer to this cell as e_j 's **inhibitory receptive field** $irf(e_j)$ and connection weights from these cells as ' λ ' type connections $\lambda(i_n, e_j)$. Excitatory connections are made from all the excitatory neurons contained in the connectable area of E_{k-1} (gray). We will refer to this collection of cells as e_j 's **excitatory receptive field** $erf(e_j)$. We will refer to the weights on connections from neurons in the receptive field as ' π ' type connections $\pi(e_m, e_j)$. λ and π connections are modifiable. The cognitron learning rule modifies these connections. In the special case of the first cognitron layer C_1 , excitatory connections are received from sensory cells in the retina.

Figure 2.42 Modifiable Inter-Cognitron Layer Connections

Excitatory neurons in each layer become sensitized to particular features in the learning domain. Inhibitory neurons are provided to help control the learning process. In particular, Fukushima introduced inhibitory cells because about 20% of the neurons in the cortex — nonspiny cells — make only inhibitory connections with other neurons. The other 80% are spiny cells which make both excitatory and inhibitory connections with other neurons. Multiple cell types are not uncommon in functional level models [Graf & Lalonde, 1987; Carpenter & Grossberg, 1987]. Excitatory cells in the lowest layer C1 respond to primitive patterns; e.g. vertical, horizontal or diagonal line segments, curves that appear often, simple repeating collections of dots, etc. Excitatory neurons in higher layers respond to features composed of combinations of lower level neuron features; e.g. geometric shapes built from line segments, combinations of geometric shapes, complex combinations of lines and curves, etc. Excitatory neurons on lower layers correspond to simple cells of section 2.2.1.6. Excitatory cells of higher layers correspond to complex cells and secondary and tertiary area cells. The last layer cells Ultimately, the top layer cells are equivalent to grandmother cells.

The Neuron Model

There are two kinds of cognitron neurons: (1) excitatory cells and (2) inhibitory cells.

Excitatory Cells

The activation level and output signal functions of an excitatory cell are modeled closely on neurobiological data, and are a more faithful rendering of the biological reality than the models we have encountered so far. These equations are given by

$$a(e_j)' = \left(\frac{1 + \sum_{e_k \in \text{erf}(e_j)} o(e_k) \cdot \pi(e_k, e_j)}{1 + o(\text{irf}(e_j)) \cdot \lambda(\text{irf}(e_j), e_j)} \right) - 1, \text{ and} \quad [2.4.4.1.3]$$

$$a(e_j) = \left(\frac{1 + a(e_j)'}{1 + \sum_{e_k \in \text{ia}(e_j)} o(e_k) \cdot \beta(e_k, e_j)} \right) - 1, \quad [2.4.4.1.4]$$

$$o(e_j) = \begin{cases} a(e_j) & \text{if } a(e_j) > 0 \\ 0.0 & \text{if } a(e_j) \leq 0 \end{cases} \quad [2.4.4.1.5]$$

Equation 2.4.4.1.3. represents the activation level of an excitatory cell in cognitron layer C_p just after it has received inputs from cognitron layer C_{p-1} , and before it has received lateral inhibitory connections from the other excitatory cells in its own layer. Equation 2.4.4.1.4. represents the activation level of the same cell an instant later, after it has received lateral inhibitory input from its neighbours in layer E_p .

Unlike perceptron-type neurons that sum the products of input signals and connection weights without regard for the type of synaptic connection, cognitron cells sum excitatory and inhibitory synapses separately. Cognitron synapses cannot change from excitatory connections to inhibitory connections — only their relative magnitudes change. The form of activation function employed here has some interesting properties. To more clearly illustrate these properties, we will simplify equations 2.4.4.1.3. and 2.4.4.1.4 and examine how they behave under different conditions. Let π be the total amount of excitatory input received by an excitatory cell and λ be the total amount of inhibition received by the cell. The activation equations can be simplified to the form

$$\left(\frac{1 + \pi}{1 + \lambda} \right) - 1 = \left(\frac{\pi - \lambda}{1 + \lambda} \right) \quad [2.4.4.1.6]$$

As long as the inhibitory input to the cell is small — $\lambda \ll 1.0$ — 2.4.4.1.6 reduces to

$$\pi - \lambda,$$

which is equivalent to the simple sum of inhibitory and excitatory terms that the previous perceptron-like models perform. We can think of these models as being special cases of the cognitron excitatory cell model in which the amount of inhibition is small. On the other hand, if the excitatory and inhibitory inputs to the cell are both large, but the excitatory and inhibitory connection weights increase during learning at about the same rate — $\pi \gg 1$ and $\lambda \gg 1$ — 2.4.4.1.6 reduces to

$$\frac{\pi}{\lambda} - 1,$$

and the inhibitory term 'shunts' the excitation term, ensuring that the output signal of the neuron remains within a small signal range. This **inhibitory shunting** is a very important effect. As we saw in section 2.2.2., organic neurons cannot output infinitely large signals. There is a relatively small dynamic range that the output signal must remain in. At the low end of this range the neuron damps out to no signal, at the high end the neuron limits out at about 1,000 Hz. If connection strengths increase constantly with learning, some mechanism must keep the output signal of neurons in range or neurons would simply limit out and stop changing state. Fukushima suggests that by carefully balancing the amount of excitatory and inhibitory connection strength, inhibitory shunting is sufficient to do this. It is very interesting to note that as long as inhibitory and excitatory inputs increase in proportion — $\pi = \xi \cdot x$ and $\lambda = \mu \cdot x$ — and the amount of excitation is greater than the amount of inhibition (as it is in the human brain) — $\xi > \mu$ — 2.4.4.1.6 is equivalent to

$$\frac{(\xi - \mu) \cdot x}{1 + \mu \cdot x} = \left(\frac{\xi - \mu}{2\mu} \right) \left(1 + \tanh\left(\frac{1}{2} \log \mu \cdot x\right) \right),$$

which is a sigmoidal-like function similar to that observed in the brain⁵⁴.

Inhibitory Cells

Inhibitory cells employ the more familiar perceptron-like activation function

$$a(i_k) = \sum_{e_j \in ca(i_k)} o(e_j) \cdot \alpha(e_j, i_k). \quad [2.4.4.1.7]$$

$$o(i_k) = a(i_k). \quad [2.4.4.1.8]$$

In fact, because all of the α weights from the connectable area are fixed and sum to 1, inhibitory cells actually compute the mean activation level of the cells in their connectable area

$$a(i_k) = \frac{\sum_{e_j \in ca(i_k)} o(e_j)}{N(ca(i_k))}. \quad [2.4.4.1.9]$$

The Learning Algorithm

Cognitrons are unsupervised learning systems that do not require a teacher to correct neural outputs. The cognitron learning rule is a variation of Hebb's law that operates as follows: Suppose we have a cell u_i that connects to a set of cells $U = \{u_j, u_k, \dots, u_n\}$. When u_i fires, the cell in U that has the largest output signal greater than 0.0 has its synapse from u_i increased thereby making u_i 's effect on its output signal greater. In a sense, the cell that 'responds best' to u_i , is made even more sensitive to u_i . All of the other cells in U lose their opportunity to learn and are not modified. On the other hand, if none of the cells in U responds at all to u_i (all their output signals are 0.0), all the synapses from u_i are increased equally by a small amount. This is done in the hope that eventually one of them will begin to respond to u_i as its other connections are changed. Initially, all of the cognitron's modifiable weights are set to random values. By chance, some cells will respond better to certain input patterns than others. The whole point of the cognitron rule is to strengthen these initial specializations, and refine them so that different cells will respond to different features that occur in the training data. The cognitron learning rule will tend to make the system specialize on common training features because cells are only made to respond better to patterns (features) that are observed during training.

⁵⁴ This equation is related to Wever-Fechner's law, an input to output relation used to approximate the behaviour of many sensory cells.

Only excitatory cells learn. Inhibitory cells are provided to balance excitatory signals and to find the mean level of excitation in a connectable region.

Fukushima has shown that this learning rule tends to reinforce a small, uniformly distributed subset of cells in layer E₂ that are sensitive to a particular feature in the training image. Figure 2.43 shows the **relative activation field** across a slice of layer E₂ for a given training image. This is a continuous representation in the variation in activation level across the network. Imagine that we have an infinite number of neurons. Within each receptive field, the most excited neuron is selected for learning. The bottom trace shows the **relative learning field** that results. This represents the distribution of neurons that win the competition to learn. The position of each such neuron is represented by a spike in a continuous neural field. The square peak represents a set of cells in which there were no winners, and each cell learned a small amount. Notice that synaptic increases are spread over the network roughly evenly. It is quite important that these cells be approximately uniform over an excitatory layer. If they are distributed evenly, they will tend to fall into the receptive fields of a large number of excitatory cells in the next higher cognitron layer. In this way, cells in the next higher layer will have a mix of feature detectors in their excitatory receptive fields. The pattern of output signals from these lower level neurons form a kind of image that the higher level neurons become sensitive to. These higher level feature patterns represent combinations of lower level features. This process continues from layer to layer, creating neurons that are sensitive to ever more abstract features.

Figure 2.43 Learning is Uniformly Distributed

(a) Smooth Activation Fields Produce a Single Winner

(b) Lateral Inhibition Helps Break Up Smooth Activation Fields and Promotes Uniform Learning

Figure 2.44 Lateral Competition Distributes Learning

Fukushima suggests that the process of finding the cell with the largest output signal is handled by glial cells that detect sharp peaks in local regions of the relative activation field. Smooth, graded changes in these regions do not produce a response from the glial cells. The situation in Figure 2.44a in which the outputs of cells are arranged in a single peak with a graded slope, will lead to only one neuron being selected for learning. Fukushima introduces lateral inhibition as a heuristic

to 'break up' smooth output signal distributions like this. As Figure 2.44b illustrates, a single sharp peak of activity will be broken by lateral inhibition into a number of roughly uniformly distributed peaks.

The learning algorithm proceeds in 5 steps:

1. $\forall i, j$: the user initializes modifiable weight vectors $\pi(e_k, e_j)$ and $\lambda(i_m, e_j)$ to random, real-valued numbers,

2. A training sample image is clamped onto the layer E_1 retina,

3. PROPAGATE OUTPUT SIGNALS FORWARD

For $k := 1$ To the number of cognitron layers Do

Begin

If $k \neq 1$ Then

$\forall e_j \in E_k$: calculate $a'(e_j)$, $a(e_j)$ and $o(e_j)$;

$\forall i_j \in I_k$: calculate $a(i_j)$ and $o(i_j)$

End;

4. UPDATE WEIGHTS

For $k := 2$ To the number of cognitron layers Do

Begin

$\forall e_j \in E_k$: If e_j and its neighbours are not outputting a signal Then

Begin

$$\pi(e_m, e_j) = \pi(e_m, e_j) + \eta_1 \cdot \left(\frac{1}{N(ca(i_k))} \right) o(e_m); \quad [2.4.4.1.10]$$

$$\lambda(i_n, e_j) = \lambda(i_n, e_j) + \eta_1 \cdot o(i_n) \quad [2.4.4.1.11]$$

End;

If e_j is outputting the largest signal in its area Then

Begin

$$i_n = \text{irf}(e_j);$$

$$z = \frac{1}{N(ca(i_k))};$$

$$\pi(e_m, e_j) = \pi(e_m, e_j) + \eta_2 \cdot z \cdot o(e_m); \quad [2.4.4.1.12]$$

$$\lambda(i_n, e_j) = \lambda(i_n, e_j) + \frac{1}{2} \eta_2 \frac{\sum z \cdot o(e_p)^2 \cdot e_p \cdot \text{erf}(e_j)}{o(i_n)} \quad [2.4.4.1.13]$$

End;

where $\eta_2 < \eta_1 < 0.0$

5. goto 2.

The algorithm repeats at most several hundred times to sensitize excitatory cells. At the end of this time, the user runs through the training samples again, recording which excitatory cells are responsive to each image. These cells are then labeled accordingly and the system may be used as a pattern recognition device.

Learning Behaviour Analysis

The learning rules are derived with two constraints in mind; (1) the modified Hebbian learning rule described earlier, and (2) the required balance between excitation and inhibition demanded by the shunting inhibition activation function. Figure 2.45 shows the path of information flow from a cell e_p in layer C_{k-1} to an excitatory cell e_q in layer C_k . Excitatory stimulation arrives over the $\pi(e_p, e_q)$ connection, and inhibitory stimulation arrives over the $\alpha(e_p, i_j)$ and $\lambda(i_j, e_q)$ connections via the inhibitory cell $i_j = \text{irf}(e_q)$. Excitation and inhibition compete to determine the activation level of the cell. We can represent the relative balance of excitation to inhibition by

$$r = \frac{\text{excitatory link strength}}{\text{inhibitory link strength}} = \frac{\pi}{\alpha \cdot \lambda} \quad [2.4.4.1.14]$$

or we can measure the net effect of e_p on e_q caused by a weight change

$$r = \frac{\Delta \pi(e_p, e_q)}{\alpha(e_p, i_j) \cdot \Delta \lambda(i_j, e_q)} \quad [2.4.4.1.15]$$

Therefore we know that if

- $r = 1$ then excitation and inhibition are equal and the effect of e_p on e_q is cancelled,
- $r > 1$ then excitation exceeds inhibition and the net effect of e_p on e_q is excitatory,
- $r < 1$ then inhibition exceeds excitation and the net effect of e_p on e_q is inhibitory.

Fukushima has selected equations that carefully balance the total amount of excitatory weight increase with the total amount of inhibitory weight increase. There are two cases to consider; (1) when none of the cells e_q connects to fire (ie. e_q does not fire but it does learn), and (2) when e_q outputs the largest signal of all cells that e_p connects to and is the only winner.

Figure 2.45
Information Flow Between Excitatory Cells in
Adjoining Cognitron Layers

In the first case the total amount of weight increase on cells that provide a net excitatory signal must exactly balance the total amount of weight increase on cells that provide a net inhibitory signal. Therefore we want ($r=1$)

$$\sum_{e_j \in \text{erf}(e_q)} \Delta\pi(e_j, e_q) = \sum_{e_j \in \text{erf}(e_q)} \Delta\alpha(e_j, e_q) \cdot \Delta\lambda(\text{irf}(e_q), e_q) \quad [2.4.4.1.16]$$

Noticing that

$$o(\text{irf}(e_q)) = \sum_{e_j \in \text{ca}(e_q)} \Delta\alpha(e_j, e_q) \cdot \Delta\lambda(\text{irf}(e_q), e_q) \quad [2.4.4.1.17]$$

we can substitute 2.4.4.1.17. for $\Delta\lambda(\text{irf}(e_q), e_q)$ in 2.4.4.1.16 and solve for $\Delta\alpha$ and $\Delta\lambda$ yielding the learning rules 2.4.4.1.10. and 2.4.4.1.11.

In the second case the total amount of weight increase on cells that provide a net excitatory signal should be greater than the inhibitory weight change. This allows the cell to respond to the same pattern with a greater level of activation — ie. the cell becomes tuned to that pattern. Fukushima arbitrarily assumes that net excitement should be twice the net inhibition. Therefore we want ($r>1$)

$$\frac{\sum_{e_j \in \text{erf}(e_q)} o(e_j) \cdot \Delta\pi(e_j, e_q)}{\Delta\lambda(\text{irf}(e_q), e_q) \cdot o(\text{irf}(e_q))} = \frac{\text{total excitement}}{\text{total inhibition}} = 2, \text{ and } \therefore \quad [2.4.4.1.18]$$

$$\frac{\sum_{e_j \in \text{erf}(e_q)} o(e_j) \cdot \Delta\pi(e_j, e_q)}{o(\text{irf}(e_q)) \cdot 2} = \Delta\lambda(\text{irf}(e_q), e_q). \quad [2.4.4.1.19]$$

Now, we already know that the Hebbian rule increases the excitatory links by a larger amount in this case. Therefore learning rule 2.4.4.1.12. is exactly the same as 2.4.4.1.10. except that the proportionality constant η_2 must be larger than η_1 . Substituting 2.4.4.1.12. into 2.4.4.1.19 gives us

$$\begin{aligned}\Delta\lambda(\text{irf}(e_q), e_q) &= \eta_2 \frac{\sum_{e_j \in \text{erf}(e_q)} o(e_j) \frac{1}{N(\text{ca}(\text{irf}(e_q)))} o(e_j)}{o(\text{irf}(e_q)) \cdot 2} \\ &= \eta_2 \frac{\sum_{e_j \in \text{erf}(e_q)} o(e_j) \cdot \Delta\alpha(e_j, e_q) \cdot o(e_j)}{o(\text{irf}(e_q)) \cdot 2}, \quad [2.4.4.1.20]\end{aligned}$$

which when simplified is the same as the learning law 2.4.4.1.13.

Finally then, we see that the cognitron is simply a system that enhances the sensitivity of neurons to a given feature. The relatively complex activation level functions and learning rules result from incorporating shunting inhibition into the system to better model the biological system, and from lateral inhibition which is not absolutely necessary but increases the number of high level abstract features that the system becomes sensitive to.

2.5. Chapter Conclusions

In this chapter we have introduced the reader to the subject of neural networks. We have seen that neural networks are learning machines that are based on models of information processing in brain tissue. We have reviewed the high-level computational architecture of the brain upon which these models are based, introducing important terminology and neurophysiological concepts, and have seen that the brain is a highly structured, computational system composed of hundreds of specialized computing centres, each of which is capable of some degree of autonomous adaptive learning.

We have also analysed six neural network algorithms that are representative of the work that has been done in this area to date. We have seen how very simple local computations can lead to complex and useful global behaviours of a network. We have found that many neural network algorithms implement a form of distributed gradient search. Iterative descent along a criterion function surface is used to implement learning, and descent along an energy surface is used to control network dynamics after learning. The iterative nature of gradient descent leads to algorithms that require many thousands of iterations, consequently, when implemented as a computer program on a von Neuman computer, neural network algorithms are very slow. If implemented by massively parallel hardware however, these algorithms promise to be

exceptionally fast. Like any gradient search technique, neural network algorithms are subject to the effects of local minima. A variety of techniques are used to avoid becoming trapped in local minima during descent.

We believe that neural learning machines are a useful addition to the engineer's toolbox. We shall now show how neural network techniques may be applied to the non-trivial sensori-motor control domain. In particular, we shall develop an adaptive controller for the collision-free control of robot manipulators that employs a functional level neural architecture.

CHAPTER III: The Robot Control Problem

The dream of creating life-like machines has fascinated mankind for centuries. Early expressions of this wish can be found in the legends of many cultures, and in the exquisitely crafted automata constructed at the turn of the eighteenth century. In the last half of the twentieth century we have finally acquired the scientific and engineering skills needed to make some of these dreams come true. The modern version of this ancient dream is manifest by the field of robotics.

Today, we see that there are many practical advantages associated with the development of robots. For example, tasks such as undersea construction, nuclear plant maintenance, and interplanetary exploration are too dangerous or too expensive for human beings to perform. Many of these tasks can be safely and inexpensively performed by machines. Other tasks are simply too boring and repetitive for humans to consistently perform, but can be quickly and reliably handled by robots. Robots are also needed when a task requires a degree of precision greater than that attainable by human beings. Most importantly, robots offer a way to cost effectively keep pace with our culture's increasing demand for higher productivity. Robots now regularly assemble machine parts, weld metal structures, paint automobile bodies and perform simple quality control tasks. The decreasing cost of computing power is gradually making robot technology more attractive to industry.

Despite the technological advances that have been made, and the optimism of researchers and the general public, robots have not had as large an impact on these application areas as expected. The primary reason is that the current generation of robots has not acquired a degree of autonomous adaptability sufficient for operation in real-world environments. So far, most robots have been applied to tasks in environments that are carefully constrained and fully understood. Total knowledge of the robot's own structure and capabilities is also commonly assumed. Unfortunately, most real world environments, including the home and office, cannot easily be controlled, and, as we shall see, the robot's own characteristics often change unpredictably. Existing robot control systems are often incapable of handling small, unpredictable problems that humans find easy to deal with. In order to realize the full potential of robotic systems, we must develop a new generation of machines that autonomously learn to deal with the problems that noisy environments and changing mechanical systems introduce. We must progress beyond current limitations before robots can be applied to all areas of human endeavour.

This chapter introduces the reader to the area of robotics. It is intended as a prelude to chapter 4 in which we describe a new approach for robot manipulator control that is less susceptible to environmental errors and mechanical faults than traditional approaches. Without human intervention, this system learns how to control arbitrarily shaped manipulators and autonomously

adapt to many important real-world faults. Section 3.1. defines the four basic types of robotic systems, and section 3.2. introduces basic robotics terminology that is used throughout the remainder of this thesis. Section 3.3. describes the traditional approach to robot manipulator control, introducing issues and useful concepts. Finally, section 3.4. describes some of the problems that traditional control systems encounter in real-world environments and argues that a new approach based on adaptive learning machine techniques is needed.

3.1. An Introduction to Robotics

Although there is no single accepted definition of the term 'robot', we will define a robot to be any mechanical system that exhibits life-like flexibility of motion, where this motion is determined by an autonomous controller — not directly by a human being. Strictly speaking, this definition precludes some systems that are popularly called robots, such as teleoperation devices which move under the remote control of a human user. There are 4 broad categories of industrial robots [Oommen, 1987, Snyder, 1985]:

1. **Limited Sequence Robots:** this type of robot relies primarily on mechanical mechanisms — cams, gears and pegs — to control its motion. Movements are programmed by adjusting screws and inserting cams to determine the position of mechanical stops and modulate movement.
2. **Playback Robots:** these robots repeat an operation that they have been instructed to perform by a human being. Programming is usually accomplished by physically, or remotely, moving the robot through a fixed set of actions. The robot records these actions and subsequently repeats them under its own control.
3. **Numerically Controlled Robots:** the movements of this type of robot are specified in terms of numerical coordinates, and the velocity and acceleration of links. The robot is typically controlled by a digital computer executing a specially written computer program. Movements are usually described in terms of the start and stop positions of the robot's gripping device. The control program automatically generates a detailed sequence of arm movements that move the gripper from its start position to the stop position. Computer control makes the combination of movements available in such robots significantly easier to specify than in 1 and 2 above. In turn, numerically controlled robots are easier to apply to a greater variety of task domains.
4. **Intelligent Robots:** these robots employ sensory recognition, adaptation and learning to determine the precise sequence of movements appropriate to a given situation. To date, most research in this area has focused on extending the capabilities of numerically controlled robots with artificial intelligence techniques. There are many capabilities at many

levels that are necessary for 'intelligent' robot control. At low levels, a robot should be able to improve its performance with practice, it should be able to accommodate changes in its physical structure (without maintenance of the robot), and it should be able to recognize and classify sensory data. Higher-level functions include the ability to map its environment, reason about movements, recognize and use physical objects, and formulate plans to carry out assigned tasks. Chapter 4 introduces a low-level movement controller that gives a robot the ability to *learn* to move its arm without colliding with obstacles, *learn* eye hand coordination, and *adapt* to changes in its physical structure. This system frees the high-level task planner from the details of arm movement and arm maintenance.

Let us now have a closer look at the architecture of a typical robot, and examine some of the problems and traditional solutions associated with the design of a robot control system.

Figure 3.1 A Generic Robot

3.2. Robotics Terminology

A general purpose robot is composed of the four physical structures illustrated in Figure 3.1; (1) the robot **body**, (2) the **propulsion system**, (3) one or more **manipulators**, and (4) a set of **sensors**. The body of the robot is a cabinet of arbitrary shape which houses its power, drive and control systems. The body is either fastened to the floor, or ceiling — as is the case with most modern industrial robots — or it is supported by a propulsion system that moves the entire robot

through its environment. The two types of propulsion systems that have been studied most extensively are wheeled carts and mechanical legs. Sensors are mounted at various points on the body. Finally, one or more arms, called the robot's manipulators, are also mounted on the body. The terms arm and manipulator are used synonymously throughout the robotics literature. Most industrial robots have a single manipulator which is used to perform pick-and-place operations, or is used to operate a tool.

Robot manipulators consist of a series of rigid **links** which are connected by **joints**. The first joint of the manipulator is connected to the body of the robot, and the last joint is connected to the robot's hand — normally called the **end-effector**. The tip of the end-effector will be referred to as the **point-of-touch** of the manipulator. The chain of connected links and joints are numbered consecutively from one, starting with the joint anchored at the body. From the point of view of an arbitrary joint, the link that is closer in the chain to the body is called an **incoming link**, while the link that is farther from the body is called the **outgoing link**. We refer to the number of joints in a manipulator as its **degrees-of-freedom (DOF)**. Most industrial robots have 6 degrees-of-freedom like the human arm. There are two logical types of joints: (1) **revolute joints**, and (2) **prismatic joints** (see Figure 3.2). Revolute joints produce a turning motion that rotates a link around an **axis of rotation** through the center of the joint. The amount that one link is turned relative to the other is called the **joint angle**. The human elbow joint is an example of a revolute joint with incoming and outgoing links attached perpendicular to the axis of rotation. The twisting motion made by the human forearm when a screw-driver is being turned is reproduced when both links are parallel to the axis of rotation. Attaching one link parallel and one perpendicular to the axis of rotation produces the type of motion produced by windmill blade rotating around its shaft. Prismatic joints produce a sliding motion that translates one link along the length of another. The amount that the link is translated relative to the other is called the **joint displacement**. The direction of movement is called the **axis of translation**, and by definition, one link must always be parallel to this axis. Prismatic joints do not occur in nature. Whenever we refer to a joint, it should be understood that it can be either revolute or prismatic. We will sometimes refer to both joint angles and joint displacements by the common term **joint settings**. The mechanism that actually moves a joint is called an **actuator**. The most commonly used actuators are stepper motors, DC motors, hydraulic pistons and pneumatic pistons.

Each combination of joint settings produces a unique **physical configuration** of the manipulator. The position of the end-effector in space is measured relative to a fixed **reference point**, usually at the wrist or the point-of-touch of the end-effector. Similarly, the orientation of the end-effector is measured relative to a fixed axis that passes through the reference point and another fixed point on the end-effector. If the end-effector is a rigid gripper that does not change

shape, it is often treated as another link on the manipulator. In order to place the end-effector at an arbitrary position and at an arbitrary orientation in three-dimensional space, a manipulator must have 6 degrees-of-freedom — three to determine the (x,y,z) position and three to determine the (roll,pitch,yaw) orientation. The set of points that the robot can reach with the end-effector reference point is called the **workspace** of the robot. In the case of a mobile robot, the workspace can be thought of as a region centered on the first joint, that moves with the robot.

Figure 3.2 Joint Types

There are two types of sensors: (1) **environment sensors** that provide information about the environment — e.g. camera images, tactile feedback, voice recognition, etc. — and (2) **feedback sensors** that tell the robot about its physical state — joint position sensors, strain gauges indicating resistance to movement, etc. Most industrial robots do not have sophisticated environment sensors. Instead, they are programmed to move manipulators to predetermined points in space where objects are carefully pre-positioned. Feedback sensors are used to determine the exact physical configuration of the arm during this process. Ingenious schemes are often necessary to ensure that objects to be grasped by the robot always have the same position and orientation¹. Needless to say, adding additional sensory equipment greatly increases a robot's usefulness and applicability to different task domains. Integrating and using sensory data obtained from several sensory modalities is an active area of research.

Important research has been performed over the last decade into robot propulsion systems, especially mechanical walking devices. Most autonomous mobile robots are used to study guidance

¹ Jigs and bowl feeders are often used to position and align objects on a pallet for pick-and-place tasks.

control problems, and few integrate the control of a manipulator with movement. Mobile industrial robots are typically used for mail delivery and automated warehouse management.

3.3. The Manipulator Control Problem

Robot design can be broadly divided into two problem areas: (1) design of the physical robot, and (2) design of the robot's control system. The first area is concerned with mechanical and electrical engineering details of the robot; e.g. the materials to be used, structural requirements of component parts, types of actuators needed, power requirements of the system, etc. The second problem area addresses itself to the issue of how a robot's parts can be moved in a coordinated and intelligent manner to solve an assigned task. Currently, our ability to design strong, flexible, lightweight, and independently powered robots is quite primitive. Yet, the utility of even a poorly constructed robot is greatly increased by a control system that is sophisticated enough to make best use of the available technology.

The robot control problem can be further divided into specialized control issues — e.g. control of manipulators, control of the propulsion system, integrated movement of multiple robots, etc. Many of these control issues are quite closely related. From the point of view of industry, the most fundamental capability is coordinated control of manipulators. The ability of a robot to pick up and manipulate objects in a human-like fashion is essential to the cost-effective production of many goods. Consequently, this problem has received the greatest amount of attention, and will probably continue to do so until human-like performance is achieved. The manipulator control problem can be partitioned into three levels:

1. **The Servo Level** — at the lowest level, individual joints must be controlled. Servo controllers are feedback control systems that compare the current setting of a joint with a target setting. If the joint is not yet at the target setting the servo adjusts the joint's actuator control signal in such a way that the difference between the current joint setting and the target setting is lessened. This process continues until the joint has reached the target setting.
2. **The Coordinated Movement Level** — this next highest level of control determines the joint settings needed to move the end-effector reference point from its starting position, to a pre-determined resting position called a **target point**. Given the initial and goal positions of the end-effector, the path that it must follow through space to reach the target point is computed, and an ordered series of points from the path, called **via points**, are sampled, starting at the initial position, and continuing to the target point. The system computes the joint settings that are needed to place the end-effector at each via point, and the low-level servos move the joints through these positions. The end result is purposeful movement of

the manipulator. If the robot's workspace is cluttered by many obstacles, the movement needed to reach a target point without bumping into an obstacle can be quite complex.

3. **The Task Level** — at this level, the controller translates the goal of the robot into a series of sub-tasks that involve manipulator movements. For example, a robot given the task of placing a box on top of a table may first have to move several books stacked on top of the box. This problem reduces to a series of start and stop positions of the end-effector that allow the robot to grasp each of the books and box in turn, and move them. The task level controller relies on the coordinated movement level controller to determine the detailed arm movements needed to reach these points. The task level control problem is a favorite topic of artificial intelligence research because of the need to recognize sub-goals and formulate abstract plans.

Although many advances have been made at each level, in general, our degree of understanding decreases as we move from lower to higher levels of control. Servo mechanisms have been well understood for many decades. A common approach to coordinated movement control has evolved over the last ten years or so that is accepted and taught as the basis of most computer controlled industrial robots. However, variations on this traditional approach have been tried with various degrees of success, and research continues. Task selection level control is very much at the edge of both robotics and artificial intelligence research and much work remains to be done. Without higher-level reasoning ability, robots will be confined to simple task domains.

It is our contention, however, that improved task level control is not the only bottleneck to general application of robot technology. We feel that limitations of the traditional approach to coordinated movement control must also be overcome. It is to this end that we now examine the coordinated movement problem in more detail.

3.3.1. The Traditional Approach to Coordinated Movement Control

The designer of a coordinated movement level robot control system usually proceeds in two steps: (1) she develops a mathematical model of the specific physical robot, and (2) writes a computer program — invariably executed by a von Neuman-style digital computer — which uses the mathematical model to plan a sequence of joint settings that correspond to a smooth movement of the end-effector. Let us have a brief look at each of these steps to see what sorts of issues are involved.

3.3.1.1. Modelling the Robot Mathematically

Mathematical modelling of a robot manipulator requires attention to three aspects of the physics of moving bodies: (1) **kinematics**, (2) **statics**, and (3) **dynamics**.

Kinematics deals with the geometry and time-dependent movement of a manipulator without considering forces that operate on its links and joints. In particular, kinematic equations relate the setting, velocity and acceleration of the joints to the position, velocity and acceleration of the end-effector reference point. Calculation of the Cartesian coordinates of the end-effector point-of-touch given a set of joint settings — called **forward kinematics** — is straight-forward. Calculation of the joint settings needed to position the end-effector at a particular target point in Cartesian space — called **inverse kinematics** — is very difficult, and may actually be intractable if the manipulator has not been carefully designed. If a manipulator has just enough degrees-of-freedom to cover all target points in its workspace, there will in general be a finite number of ways that the manipulator can touch a target point with its end-effector. This is because inverse kinematics is equivalent to solving a number of equations in the same number of unknowns. The equations are trigonometric polynomials in the joint settings — not linear equations — therefore there will be more than one, but a finite number of, solution arm configurations for a given target point. For the same reason, if we have too many joints, we will find that there are an infinite number of ways of touching a target point. An infinite number of solutions is referred to as a **degeneracy** of the inverse kinematic equations. If we do not have enough joints, there will be no solution for most target points. There are usually some arm configurations that impact our ability to perform kinematic calculations. In these cases, a **singularity** occurs when a denominator in one of the kinematic equations goes to zero. These configurations usually touch the boundaries of the workspace where some of the links are parallel.

Statics deals with the balance of forces and torques required to stop the manipulator from moving because of loading and gravitational effects. A static analysis is performed for an isolated link, and neighbouring links are considered to be contributing forces and torques applied to the link. By taking all such forces into account, expressions can be found that describe the net force and torque acting on the link. These values are then used to apply the appropriate counterforces at the joint actuators.

Dynamics takes both kinematics and statics into account to describe moving manipulators under load. There are three important effects that must be considered, namely: (1) the sensitivity of a particular joint to torque varies with the configuration of the manipulator, (2) additional forces appear that are functions of the products of the angular velocities, and (3) there is coupling between the motions of the links. Joint torques are related to angular acceleration and velocity product terms are thought of as generalized centrifugal forces. Given the joint settings and their rates of change with time, joint torques can be computed that move the arm along a given trajectory.

In this thesis we shall limit our research to the kinematics of manipulator movement. However, the reader should be aware of the importance of the statics and dynamics of manipulator control.

3.3.1.1.1. Kinematic Equations For the Horn Robot

In order to give the reader some appreciation for the kinematics of robot manipulators, we shall now directly derive the forward and inverse kinematics for a simple 2-degree-of-freedom robot that operates in a 2-dimensional workspace. This manipulator was originally analysed by Berthold K.P. Horn, and is commonly referred to as the **Horn robot**. Figure 3.3a depicts the Horn robot. It consists of two joints J_1 and J_2 connected to two links of length L_1 and L_2 . The joint angles of J_1 and J_2 are given by θ_1 and θ_2 respectively. θ_1 is measured as the angular displacement of link 1 from the horizontal x axis. θ_2 is measured as the angular displacement of link 2 from the line that passes through the length of link 1. We shall refer to the position of the point-of-touch of the robot as $p_1=(x_1,y_1)$, and the position of J_2 as $p_2=(x_2,y_2)$.

Let us first derive the forward kinematics of the Horn robot. Referring to Figure 3.3b, we immediately see that

$$x_2 = L_1 \cdot \cos(\theta_1), \text{ and} \quad [3.3.1.1.1.1]$$

$$y_2 = L_1 \cdot \sin(\theta_1). \quad [3.3.1.1.1.2]$$

The x and y components of the right triangle with hypotenuse L_2 and angle $\theta_1+\theta_2$ is given by

$$x_3 = L_2 \cdot \cos(\theta_1+\theta_2), \text{ and} \quad [3.3.1.1.1.3]$$

$$y_3 = L_2 \cdot \sin(\theta_1+\theta_2). \quad [3.3.1.1.1.4]$$

Now, we know that $x_1=x_2+x_3$, and $y_1=y_2+y_3$, therefore substituting equations 3.3.1.1.1., 3.3.1.1.2, 3.3.1.1.3., and 3.3.1.1.4., we can solve for the position p_1 of the end-effector from the joint angles as follows

$$x_1 = L_1 \cdot \cos(\theta_1) + L_2 \cdot \cos(\theta_1+\theta_2), \text{ and} \quad [3.3.1.1.1.5]$$

$$y_1 = L_1 \cdot \sin(\theta_1) + L_2 \cdot \sin(\theta_1+\theta_2). \quad [3.3.1.1.1.6]$$

Differentiating these equations, we can find the velocity of the end-effector from the angular velocities of the joints

$$x_1' = -L_1 \sin(\theta_1) \theta_1' - L_2 \sin(\theta_1+\theta_2) (\theta_1'+\theta_2'), \text{ and} \quad [3.3.1.1.1.7]$$

$$y_1' = L_1 \cos(\theta_1) \theta_1' + L_2 \cos(\theta_1+\theta_2) (\theta_1'+\theta_2'). \quad [3.3.1.1.1.8]$$

Equations can be conveniently expressed in the matrix notation

$$p_1' = J \cdot \theta', \text{ as}$$

$$\begin{bmatrix} x_1' \\ y_1' \end{bmatrix} = \begin{bmatrix} -L_1 \sin(\theta_1) - L_2 \sin(\theta_1+\theta_2) & -L_2 \sin(\theta_1+\theta_2) \\ L_1 \cos(\theta_1) + L_2 \cos(\theta_1+\theta_2) & L_2 \cos(\theta_1+\theta_2) \end{bmatrix} \begin{bmatrix} \theta_1' \\ \theta_2' \end{bmatrix}, \quad [3.3.1.1.1.9]$$

(a) A 2-DOF Horn Robot

(b)

(c)

(d)

Figure 3.3 Finding the Kinematics of a Horn Robot

where $\mathbf{p}_1'$ represents the velocity of the end-effector and θ' represents the angular velocity of the joints. When the kinematic equations can be expressed in this form, $\mathbf{J}$ is referred to as the **Jacobian matrix**. Differentiating 3.3.1.1.9 and presenting it in matrix form we can find the acceleration $\mathbf{p}_1''$ of the end-effector expressed in terms of the angles, velocities and accelerations of the joints as

$$\begin{bmatrix} \ddot{x}_1 \\ \ddot{y}_1 \end{bmatrix} = \begin{bmatrix} -L_1 \sin(\theta_1) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos(\theta_1) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_1 + \ddot{\theta}_2 \end{bmatrix} - \begin{bmatrix} L_1 \cos(\theta_1) & L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin(\theta_1) & L_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} (\dot{\theta}_1)^2 \\ (\dot{\theta}_1 + \dot{\theta}_2)^2 \end{bmatrix} \quad [3.3.1.1.10]$$

Finally then, equations 3.3.1.1.5, 3.3.1.1.6, 3.3.1.1.9, and 3.3.1.1.10 are the forward kinematic equations of the robot.

Now, let us derive the inverse kinematics of the Horn robot. Referring to Figure 3.3c we note the right triangle defined by p_1 and the origin. given the angle $\sigma + \theta_1$ between the hypotenuse of this triangle and the x-axis, we can find the length of the hypotenuse h as

$$h = \sqrt{x_1^2 + y_1^2}.$$

Also we can find θ_1 in terms of the angle σ and x_1 and y_1 by

$$\tan(\sigma + \theta_1) = \frac{y_1}{x_1}, \text{ and therefore}$$

$$\theta_1 = \tan^{-1}\left(\frac{y_1}{x_1}\right) - \sigma. \quad [3.3.1.1.1.11]$$

Figure 3.3d shows the triangle made by p_1 , p_2 and the origin. Now to find σ we can use the cosine rule

$$C^2 = A^2 + B^2 + 2AB\cos(\sigma)$$

by substituting and rearranging as follows

$$L_2^2 = L_1^2 + (\sqrt{x_1^2 + y_1^2})^2 + 2L_1\sqrt{x_1^2 + y_1^2}\cos(\sigma), \text{ so}$$

$$\cos(\sigma) = \frac{L_1^2 + (\sqrt{x_1^2 + y_1^2})^2 - L_2^2}{2L_1\sqrt{x_1^2 + y_1^2}}, \text{ and therefore}$$

$$\sigma = \cos^{-1}\left[\frac{L_1^2 + (\sqrt{x_1^2 + y_1^2})^2 - L_2^2}{2L_1\sqrt{x_1^2 + y_1^2}}\right]. \quad [3.3.1.1.1.12]$$

Substituting 3.3.1.1.1.12 into 3.3.1.1.1.11 we can solve for θ_1

$$\theta_1 = \tan^{-1}\left(\frac{y_1}{x_1}\right) - \cos^{-1}\left[\frac{L_1^2 + (\sqrt{x_1^2 + y_1^2})^2 - L_2^2}{2L_1\sqrt{x_1^2 + y_1^2}}\right]. \quad [3.3.1.1.1.13]$$

Now, from Figure 3.3d we see that

$$B^2 = A^2 + C^2 + 2AC\cos(2\pi - \theta_2)$$

by substituting and rearranging as above we find that

$$\cos(2\pi - \theta_2) = \frac{L_1^2 - L_2^2 - (\sqrt{x_1^2 + y_1^2})^2}{2L_1L_2}.$$

However, $\cos(2\pi - \theta_2) = -\cos(\theta_2)$, so we have

$$-\cos(\theta_2) = \frac{L_1^2 - L_2^2 - (\sqrt{x_1^2 + y_1^2})^2}{2L_1L_2}, \text{ and therefore}$$

$$\theta_2 = \cos^{-1}\left[\frac{x_1^2 + y_1^2 - L_1^2 - L_2^2}{2L_1L_2}\right]. \quad [3.3.1.1.1.14]$$

We now have the joint angles expressed in terms of the position of the end-effector. Now, solving for the inverse velocity and acceleration equations, we can take advantage of the matrix form of equations 3.3.1.1.1.9 and 3.3.1.1.1.10. To find the angular velocities θ' in terms of the joint angles and end-effector position, note from 3.3.1.1.1.9 that

$$\theta' = J^{-1} \cdot p_1'.$$

Therefore we must find the inverse of the jacobian matrix if one exists. If an inverse does not exist we must use the pseudo-inverse. Similarly, from 3.3.1.1.1.10 we have

$$\begin{bmatrix} x_1'' \\ y_1'' \end{bmatrix} = A \cdot \begin{bmatrix} \theta_1'' \\ \theta_1'' + \theta_2'' \end{bmatrix} - B \cdot \begin{bmatrix} (\theta_1')^2 \\ (\theta_1' + \theta_2')^2 \end{bmatrix}$$

and therefore we need to solve

$$\begin{bmatrix} (\theta_1')^2 \\ (\theta_1' + \theta_2')^2 \end{bmatrix} = A^{-1} \cdot \begin{bmatrix} x_1'' \\ y_1'' \end{bmatrix} + A^{-1}B \cdot \begin{bmatrix} (\theta_1')^2 \\ (\theta_1' + \theta_2')^2 \end{bmatrix}$$

Solving for these equations we have

$$\begin{bmatrix} \theta_1' \\ \theta_2' \end{bmatrix} = \frac{1}{L_1 L_2 \sin(\theta_2)} \cdot \begin{bmatrix} L_2 \cos(\theta_1 + \theta_2) & L_2 \sin(\theta_1 + \theta_2) \\ -L_1 \cos(\theta_1) - L_2 \cos(\theta_1 + \theta_2) & -L_1 \sin(\theta_1) - L_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \cdot \begin{bmatrix} x_1' \\ y_1' \end{bmatrix} \quad [3.3.1.1.1.15]$$

$$\begin{bmatrix} \theta_1'' \\ \theta_1'' + \theta_2'' \end{bmatrix} = \frac{1}{L_1 L_2 \sin(\theta_2)} \begin{bmatrix} L_2 \cos(\theta_1 + \theta_2) & L_2 \sin(\theta_1 + \theta_2) \\ -L_1 \cos(\theta_1) - L_2 \cos(\theta_1 + \theta_2) & -L_1 \sin(\theta_1) - L_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \cdot \begin{bmatrix} x_1'' \\ y_1'' \end{bmatrix} + \frac{1}{L_1 L_2 \sin(\theta_2)} \begin{bmatrix} L_1 L_2 \cos(\theta_2) & L_2^2 \\ -L_1^2 & -L_1 L_2 \cos(\theta_2) \end{bmatrix} \cdot \begin{bmatrix} (\theta_1')^2 \\ (\theta_1' + \theta_2')^2 \end{bmatrix} \quad [3.3.1.1.1.16]$$

Finally then, equations 3.3.1.1.1.13, 3.3.1.1.1.14, 3.3.1.1.1.15, and 3.3.1.1.1.16 are the inverse kinematic equations of the Horn robot. Notice that degenerations do not occur in the inverse kinematic equations, but singularities do. For example, the velocity inverse jacobian will be undefined if θ_2 is at 0 radians or π radians. These conditions correspond to the robot touching the borders of its workspace with its end-effector.

The reader may agree that the calculation of kinematic equations for even the simple Horn robot is fairly involved and tedious. The complexity of the equations increases rapidly with additional degrees-of-freedom. The task might seem intractable for arms of six degrees of freedom. However, techniques employing homogeneous coordinate transforms have been developed to simplify the problem. The resulting equations are very complicated and computationally expensive. The equations for a six-degree-of-freedom manipulator will not be reproduced here because of their length — there are hundreds of trigonometric terms — however, the reader is encouraged to examine them in [Snyder, 1985] to get a feel for the complexity of computation involved.

3.3.1.2. Writing a Control Program

Once the robot has been modeled, the second step of traditional control system design is to make use of these equations in a computer program that plans a sequence of joint settings corresponding to a smooth movement of the end-effector. Three important aspects of this problem are:

1. **Procedural Encoding of Kinematic Equations** — In reality the kinematic modelling problem is even more complicated than presented here. The designer must also take into account the shape and thicknesses of links, and specialized restrictions on joint movement. Also, degeneracies and singularities must be handled by the program. Encoding of kinematic equations is not as straight-forward as one might hope.
2. **Eye-Hand Coordination** - It should be possible for the robot to touch a target point selected by its environmental sensors. This problem is usually handled by converting the target point defined in sensory coordinates to a point in an objective, universal Cartesian coordinate frame. From the Cartesian frame we can then apply inverse kinematic equations to find joint settings that cause the end-effector to touch the point. We refer to this possibly infinite set of joint settings as the target arm configuration set.
3. **Trajectory Planning** - Given the initial configuration of the arm, and a target configuration determined by inverse kinematics, what should the intermediate positions of the arm be? A time sequence of appropriate intermediate arm configurations is called the **trajectory** of the arm. The spacial curve traced by the end-effector as it moves through the robot's workspace is called the **path** of the trajectory.

Trajectory planning is relatively straight-forward as long as there are no obstacles that the arm can bump into while moving through its trajectory. Many approaches exist for finding trajectories in obstacle-free workspaces. We will not attempt to give a full accounting of them here. The interested reader is referred to [Snyder, 1985] where a more complete survey is given. In general,

there are two styles of trajectory planning: (1) **Cartesian planning**, and (2) **joint angle planning**.

In the Cartesian planning approach a set of via points are found along a desired Cartesian workspace path that moves the robot's end-effector to the target point. Fine control of the arm requires many via points. Once the via points have been selected, inverse kinematic equations are used to compute the arm trajectory. This process can be computationally expensive because of the heavy use of the inverse kinematic equations. For this reason, most trajectory planners of this sort attempt to reduce the number of via points needed to produce the desired arm movement. A good example of this type of planner is given by Taylor's bounded deviation method.

The joint angle planning approach also finds via points along a curve, however, the curve resides in an **arm configuration space** rather than in the Cartesian workspace. Each axis of arm configuration space represents the range of movement of one of the joints in the arm, and therefore points in this space represent physical configurations of the arm. A curve in arm configuration space corresponds to a smooth and continuous trajectory of the arm that takes the end-effector from the point of touch associated with the first end point of the curve to the target point associated with the second end point of the curve. The end points are therefore given by the initial configuration of the arm, and a target configuration derived from the inverse kinematic equations. Any curve that connects these end-points describes the desired trajectory. It is a simple matter to sample via points along the length of this curve and output them to the servo level controllers. This approach greatly simplifies the computational requirements of the planner, since inverse kinematic calculations are only required once to produce the target configuration end point of the curve. Problems occur however, because the shape of the curve in arm configuration space is related to the Cartesian path of the end-effector by the highly non-linear inverse kinematic mapping. A simple straight-line arm configuration curve may correspond to a complex and time consuming movement of the arm that causes the end-effector to wander about the workspace before reaching the target point. We prefer direct and purposeful movements of the end-effector. Therefore, trajectory planners of this sort attempt to constrain arm configuration space curves in ways that produce reasonable end-effector paths. As we might expect, this often involves additional inverse kinematic computations, and once again we are faced with a trade-off between precision of movement and computational complexity.

When obstacles are introduced into the workspace, the trajectory planning problem becomes vastly more complicated. In order to prevent the arm from bumping into obstacles in its environment, we must now take into account the volume of the workspace intersected by each link at each instant in time along the trajectory. A trajectory which avoids colliding the arm with obstacles in the workspace is called a **collision-free arm movement**. Current approaches for planning collision-free arm movements are generally ad hoc mixtures of the previous two forms of

motion planning coupled with a data structure that models the relationship between arm configurations and the current position of the obstacles. This data structure is traversed to find possible collision-free movements of the arm. Special assumptions are often made regarding the shape of the obstacles — e.g. convex polyhedra — and the arm links — e.g. simple cylinders or line segments. The time complexities of these algorithms are normally tied to the number of obstacles in the workspace and their shapes as well as the number of degrees-of-freedom of the arm. The final complexity is predominated by the type of data structure used, the speed with which it can be built, and the speed with which it can be traversed. Collision-free trajectory planners are an important area of robotics research.

3.4. Is This Really the Best Approach ?

At this point, we hope that readers previously unacquainted with robotics are beginning to appreciate the complex design issues associated with the coordinated movement control problem. If traditional techniques are used to control general purpose robots, a great deal of expense and energy will be needed to design, implement, test and maintain very complex computer programs. Is this really the best way to proceed? Judging from other industries that make heavy use of complex software control systems², the traditional approach will require a huge investment of manpower and years of development time before general purpose commercial robots will be available. Furthermore, the resulting robots will suffer from the same lack of flexibility and extendibility that all large software systems are subject to.

Perhaps we need to consider design approaches that do not rely as heavily on specialized computer programming. An ideal robot control system would automatically configure itself to a particular robot without the need for *any* computer programming. A human would provide a simple 'seed' controller, and the *experiences* of the robot during an initial training phase would self-organize this device into as complex or simple a controller as needed. It is our belief that such a device is possible, and it is the purpose of this thesis to suggest a starting point for further research toward this goal.

3.4.1. The Real-World is a Hostile Place For Robots

While traditional control programs may work well in the laboratory, they often suffer from serious limitations when applied to realistic environments. This is because the real world is a hostile place for robots. The kinematics of a robot may change with time due to metal fatigue, friction, collisions with obstacles and a host of other physical factors. These changes may be gradual or they may occur suddenly. The characteristics of feedback sensors are also subject to

² E.g. the telephone industry, NASA, military weapons systems, etc.

non-linear changes due to temperature effects, and degradation of electronic components. These changes are very hard to predict and model, particularly when cheap components are used. Kinematic and sensory changes introduce error into the control program's kinematic equations. Traditional control programs require that the robot be maintained or that the program be modified to reflect these on-going changes. Finally, a variety of factors often lead to component failure in real systems. Conventional computers are highly intolerant of component failures — loss of a single component can be disastrous. Control programs are unable to function if even a single critical line of code is corrupted by memory failure or human programming error. Reprogramming costs, down-time and physical maintenance can be prohibitively expensive when a factory contains dozens of robots. There are some specialized domains in which maintenance is actually impossible³. Unfortunately, the domains that would most benefit from robot technology are usually those that present the most adverse physical conditions⁴. Even the home and office can be hazardous to the current generation of robots⁵.

Another problem is that real-world environments are often cluttered with complex obstacle sets that change in unpredictable ways. Control programs must rebuild their obstacle representation data structures when obstacles change position or shape. This presents a significant speed problem in environments that contain large numbers of obstacles that move relative to the robot⁶. Although some speed improvements can be expected from parallel hardware, it is not an easy task to re-design serial control programs to take advantage of parallelism.

3.4.2. Nature Has Already Solved These Problems

When we look at the organic systems that we are trying to mimic, we find that nature has already done an excellent job of designing flexible systems capable of adapting to a wide variety of real-world problems. Many animals, notably humans, start life relatively helpless, unable to control their limbs in a coordinated way, yet quite rapidly, through experience and practice, autonomously learn very fine motor control and eye-hand coordination skills. Animals grow, and the kinematics of their manipulators and eye-hand relationships change. Muscle feedback also tends to change with development in non-linear ways. Yet, these slow changes do not interrupt the

³ Consider a Mars rover that tips over on its arm and bends its first link slightly. Sending a maintenance person is out of the question in this case.

⁴ Especially those that most need robot systems — e.g. underwater construction sites, extra-terrestrial terrains, arctic exploration, deep-earth mines, the battle field, etc.

⁵ They contain stairs, moving objects (pets, humans and other machines), unpredictable terrains (shifting furniture), noisy electrical environments (RF from TVs, fluorescent lights, and appliances), and poor lighting conditions, among other things.

⁶ It may be the robot that is moving and the objects that are stationary.

animal's control over its motor functions, indicating the importance of adapting to changes in the body's response to motor control signals. Biological systems are also notable for their fault tolerant characteristics, adapting to physical injury and the death of cells. Animals inhabit real terrains consisting of highly complex and dynamically changing obstacle sets. Real time response is necessary in such settings to ensure survival of the organism. Finally, and perhaps most significantly, natural control systems do not require programming — they learn the control function through doing⁷. Clearly, there is much we can learn from nature.

The key to these remarkable abilities is a high degree of *autonomous adaptive learning*. This adaptation is driven by simple interactions with the real world. As we have already seen, adaptive learning is a fundamental characteristic of the brain. Therefore, just as we turned to the brain as a model for learning systems in chapter 2, it is natural that we turn to it as a model for robot control systems in the next chapter.

3.4.3. Adaptive Learning Systems for Motor Control

Although adaptive learning systems have been proposed for robot control, most of these systems have only been applied to very simple coordinated movement problems. These systems learn in the sense that they improve a robot's ability to follow a single pre-planned path with its end-effector by repeated practice. For a given practice movement trial, a teacher keeps a history of torque applied by the controller at each time step, and the amount of error between the actual trajectory produced and the desired trajectory. This error history is passed through a linear filter which generates a function that is added onto the previous torque history to generate the next torque history. The interactions that occur between the filter and non-linear equations of motion of the arm eventually yield the torque functions needed to correctly move the arm along the path. These techniques are closely related to conventional learning automata methods. A good survey of such systems is given in [Craig, 1988]. This type of approach is adequate for repeated ritual movements that are supplied by a teacher for a given situation, but it is not flexible enough to deal with generalized, environment driven movements. Nor does it attempt to handle the problem of collision-free movement. The primary weakness of this approach is that the system does not remember learned data structures that can be used to generate *any* arbitrary movement.

Neural architectures offer an alternative approach to the design of robot control systems. Like biological systems, neural architectures support a high degree of adaptability and they are capable of learning sensory-motor constraints from training examples. A few authors have investigated

⁷ Actually there may be some underlying instinctive program that provides a primitive set of tendencies or motor operators. In this sense nature has programmed animals through the evolutionary process. Nevertheless, the 'higher' life-forms learn all but the most basic motor capabilities, and refine the instinctive variety through practice.

neural architectures for manipulator control (Bullock and Grossberg³, Grossberg and Kuperstein⁸, Kuperstein¹², Tsutsumi¹⁸), however, very little work has been done to date. Some of this work has a more philosophical than practical bent and has been directed toward understanding the biological mechanisms that are used by the motor control centres of the brain. So far, two techniques have been suggested for robot manipulator control.

Tsutsumi's Original Arm

A Truss Structure Arm

Figure 3.4 Tsutsumi Arms

The first technique, by [Tsutsumi, et. al., 1988; Tsutsumi & Matsumoto, 1987], employs Hopfield network energy minimization behaviour to move an arm to a target point. It is assumed that each joint in the robot's arm is capable of independently determining its Euclidian distance to the target point in Cartesian space (e.g. each joint may have an associated laser range finder). Tsutsumi then relates the energy of the network to the distance of each joint to the target. The network output is the vector of joint angles that determine the position of the end-effector. As the network moves toward a minima in its energy landscape, the arm 'relaxes' into a configuration that tends to minimize the distance of the end-effector from the target point. Tsutsumi has applied this technique to traditional Horn robots as shown in Figure 3.4a as well as to 'truss-structure' arms which consist of a series of members arranged like a bridge as shown in Figure 3.4b. The Tsutsumi arm does not deal very well with obstacles, however. In order to avoid collisions, two obstacle avoidance sensors are attached to each link. These sensors measure the link's distance to every obstacle in the workspace. The energy function of the network is changed so that energy is minimized by maximizing the distance of each link from all obstacles. There are still cases in which the arm can become trapped by obstacles because the network finds a local minimum. Tsutsumi introduces the following heuristic to attempt to deal with these cases: if the network stops changing its activation state before the end-effector reaches the target, an imaginary, circular obstacle is

introduced into the workspace, centered on the end-effector. This 'virtual' obstacle causes the arm to move away from its incorrect resting position. The hope is that by flailing about the workspace, the arm will eventually find a collision-free trajectory that allows it to touch the target point. Although an interesting approach, this idea produces arm trajectories that are highly indirect, and are not guaranteed collision-free.

A rather different approach has been incorporated into systems developed by [Kuperstein, 1987] and [Elsley, 1988]. This approach is used to learn the inverse kinematics of arm movement, and is not intended to generate detailed arm trajectories. Each target point is represented by a unique pattern — either the (x,y,z) coordinates of the point, or a pattern derived from sensory data. Each arm configuration that touches a target point is also represented by a unique pattern — derived from feedback sensors. A random set of these patterns is presented to an arbitrary pattern associator (see section 2.1.2.). A feed-forward architecture is always used which employs a variation of the perceptron algorithm (e.g. [Elsley, 1988] uses back propagation) to learn target point to arm configuration associations. Once learning is complete, given a target point pattern as input, the network will generate an arm configuration that touches the point. If implemented by parallel hardware, this inverse kinematic transformation can be accomplished in only a few clock ticks. These systems have been shown to generalize well to target points that they have not encountered during learning.

This general approach to inverse kinematics exhibits two important properties:

1. The system is able to learn the kinematics of an arbitrarily shaped arm without the need for human programming.
2. If the kinematics of the robot change, the system can be re-taught inverse kinematics once again, thereby adapting to the changes.

Unfortunately, there are also a number of important limitations associated with the approach:

1. The network can only generate one configuration for a given target. In general, there may be an infinite number of configurations that touch a given target. If an obstacle happens to occupy the same volume of space that the output arm configuration occupies, the system will not be of much use.
2. The approach does not address the problem of arm *movement*. In particular, the approach is not concerned with collision-free movement. A conventional algorithmic solution must be used to generate collision-free arm trajectories. In a hybrid system the robot would still be subject to most of the real-world problems that we have described above. The neural network part of the system would not contribute to collision avoidance, and would be reduced to a fast inverse kinematic calculator.

3. There is no well understood relationship between the accuracy of pattern association systems and the number of neurons that are needed.

In spite of these drawbacks, the fact that the network can adapt to particular robot geometries and unplanned changes is quite encouraging.

3.5. Chapter Conclusions

In this chapter we have introduced the reader to the area of robotics, with special emphasis on position kinematics for the control of robot manipulators. We have identified issues that are important to manipulator control, and we have briefly examined the traditional approach to the design of such control systems.

We have also shown that these traditional control systems are subject to many problems caused by unpredictable real-world conditions. Before we can expect to realize the full promise of robots, we will have to design controllers that are flexible enough to operate in natural domains.

Finally, we have argued that nature has already found solutions to all of these problems. An important aspect of any robot control system must be a degree of autonomous adaptive learning. Neural learning systems are a good candidate for this type of control because of their ability to learn from training data. Some preliminary work has already been done that supports this claim.

In the next chapter we extend this work by developing an architecture that, in addition to learning kinematic constraints for eye-hand coordination, also learns obstacle avoidance constraints that can be used to rapidly find collision-free movements of a robot arm. In addition, it is capable of generating more than one potential target configuration for a given target point. We derive a clear relationship between the number of neurons required and the accuracy of the system.

CHAPTER IV: A Neural Controller for Collision-Free Arm Control

This chapter presents a new approach for the control of robot manipulators moving in a workspace containing a changing set of obstacles. An adaptive robot arm controller is presented which is based on a functional level neural architecture. Starting with no knowledge of the kinematics of the arm, the controller is able to autonomously learn eye-hand coordination by simple unsupervised movements during an initial training phase. The robot 'glances' at a target point with its sensors, and the controller quickly generates a sequence of joint interpolated arm movements that position the end-effector near the intended target.

The system is able to control any type of arm, regardless of link shapes, the type or combination of joints, or restrictions on movement imposed by the body of the robot or mechanics of the arm. With very minor modifications, the system can theoretically be made to control arms of any number of degrees-of-freedom. It is also able to automatically adapt to changes in the kinematics of the eye-arm-body arrangement, and non-linear changes in the characteristics of the feedback sensors. Furthermore, the controller learns collision avoidance constraints that allow it to touch targets without colliding with obstacles. The system is not limited to a single arm configuration per target. Rather, it generates a representative set of final arm configurations and then selects a single configuration that can be reached without collision — all in constant time. This process is not affected by the number of obstacles or their shape. The system then generates a sequence of continuous arm movements that carry the arm from an initial configuration to the selected final configuration. The neural design approach makes implementation of the controller by massively parallel hardware a straightforward matter. Massive parallelism would produce an exceptionally fast arm controller.

In practice, the system is limited to arms of 2 or 3 degrees-of-freedom because the number of neurons required rapidly increases with the number of degrees-of-freedom and precision of movement. We are encouraged however by the work of Yasir Abu-Mostafa and Demetri Psaltis [Abu-Mostafa & Psaltis, 1987] which demonstrates the feasibility of optical systems which support hundreds of thousands of neurons and hundreds of millions of modifiable interconnects. Controllers using small numbers of neurons which are capable of less precision may be accurate enough for many applications. These systems would plan broad movements of the manipulator which would be refined near the target by other simple mechanisms [Andrade-Abalarach, 1988]. Empirical results presented in this chapter show that human-sized robot arms can be accurately controlled by very small networks with the addition of simple heuristics.

Section 4.1. describes the collision-free arm control problem in the context of inter-spatial association and the satisfaction of multiple simultaneous constraints. This discussion introduces the

central ideas employed in this arm controller design. Section 4.2. describes the adaptive robot and its primary control systems. Section 4.3. then focuses on the architecture of the adaptive arm controller. Section 4.4. describes the operation of the controller: how it learns, and how it is used to touch a target. Section 4.5 analyses the learning and operational behaviour of the controller, investigates sources of system imprecision, and explores the relationship between system performance and the number of neurons used. Section 4.6. describes a simulation of the adaptive robot and illustrates control of some typical 2 degree-of-freedom arms, and presents results of an empirical study of the positioning accuracy of the controller for human-sized arms. Finally, section 4.7. concludes by summarizing the relative advantages and disadvantages of this approach.

4.1. Arm Control As Simultaneous Satisfaction of Multiple Constraints

As we saw in section 2.3.2., the brain is particularly good at solving problems that require simultaneous satisfaction of multiple constraints. The coordinated and purposeful movement of our bodies is an example of this class of problem. Let us now examine the problem of planning collision-free arm movements in the context of multiple constraint satisfaction. We will see that important kinematic and collision avoidance constraints can be expressed as inter-spacial associations (see section 2.2.1.7.).

4.1.1. Target Points Can be Expressed as Eye Configuration Vectors

We have a variety of senses that we use to analyse the world around us. Humans rely heavily on the visual sense because of its high degree of resolution, and positioning accuracy¹. When we identify a target point visually, how do we convert sensory data into position information that we can use to manoeuvre our hand into the right spot? Some information is given by the coordinates of the target point in our field of vision, but this information is not enough to tell us how far away the target point is. To detect distance, we rely on two effects that occur when we center the target point in the field of vision of each eye²; (1) the stereoscopic disparity of eye angles, and (2) the relative displacement of objects in the images from each eye³. There is actually enough information contained in eye angles alone to detect distance directly by **range triangulation**. In fact, many robotic systems make use of this principle to determine the position and relative orientation of objects by sampling the distance to a representative set of points on the object's surface [Page, et.

¹ Other senses besides vision may be used, but visual point location is the most accurate. The same argument applies to other sensory modalities so we will henceforth assume that the target is located visually.

² This centering process is called **foveation** because the portion of the retina associated with the center of the field of view is the fovea.

³ Visual data contains other information that can be used to refine distance estimates — perspective, texture gradients, relative movement of objects, etc. However, these are secondary clues. Stereoscopic displacement is more accurate.

al., 1983; Porter and Mundy, 1982]. We will therefore concentrate on range triangulation as a source of positioning information for our adaptive robot arm controller. In the case of a robotic system, this range information can be supplied by various types of sensors. In the following discussion we will use the term 'eyes' to refer to any range finding system. Although we use a human metaphor to illustrate these ideas, they apply equally well to artificial sensors.

Muscles anchored in the eye socket allow each eye to move with three degrees of freedom — roll, pitch and yaw. We can think of each eye as being turned by three revolute joints set at right angles to one another. The **line-of-sight** of each eye is the infinite ray extending through the horizontal axis of the eye from the fovea through the center of the pupil. When we foveate on a point, we align the lines-of-sight of both eyes so that they intersect the target point. Only two degrees of freedom — pitch and yaw — determine the orientation of a line-of-sight, so we see that only two degrees of freedom are required by each eye to find the distance to any target point in the half-space immediately forward of the head. Furthermore, every target point in this half-space — except those that lie along an infinite line passing through the volumetric center of both eyes — is uniquely defined by the four eye angles that foveate the eyes.

Let us now define a four-dimensional space such that each axis represents the range of a pitch or yaw angle of one of the two eyes. Vectors in this space correspond to particular orientations of the lines-of-sight of the eyes. We shall call these vectors **eye configuration vectors** and the corresponding space the **eye configuration space**. It will often be convenient to refer to the physical state of the eyes simply as the physical configuration of the eyes. Some eye configuration vectors cause the two lines-of-sight to intersect at a point. These vectors are the ones that foveate the eyes on target points. There is only one vector per point. Therefore, each target point is uniquely associated with an eye configuration vector. There are many other eye configuration vectors that do not cause the lines-of-sight to intersect. These vectors are therefore not associated with a target point. Although these physical configurations do not normally occur in healthy humans, they are nevertheless described by vectors in eye configuration space. In order to distinguish these two types of physical configurations, we will call those that foveate on target points, **foveating eye configurations** and those that do not, **non-foveating eye configurations**. A target point that is being foveated will sometimes be referred to as a **point-of-gaze**. This will help us to distinguish it from other potential target points that the eyes are not currently looking at.

It should now be clear that it is possible to represent a target point by the eye configuration vector that foveates it.

In humans this situation is complicated by the fact that the eyes are mounted in the head, which is free to turn. The human head can also move with three degrees-of-freedom. We must consider all three degrees of freedom because the roll axis of the head does not pass through a point between the eyes. Therefore, to express the position of a target point relative to the body, we require seven degrees-of-freedom — four for the eyes and three for the head. Although our four-dimensional eye configuration space now becomes a seven-dimensional eye-head configuration space, our ability to refer to target points by configuration space vectors is not affected. In order to simplify the following discussion, we will assume that the head is stationary and the eyes are free to move.

4.1.2. Target Points Can Also be Expressed as Arm Configuration Vectors

In section 4.1.1., we created a space to describe the physical configuration of the eyes. Similarly we can create a space to describe the physical configuration of the arm. Each axis of this space represents the range of arm joint settings. Vectors in this space represent specific combinations of joint angles in the case of revolute joints, and joint offsets in the case of prismatic joints. In either case the ranges are continuous and real-valued. We will refer to these vectors as **arm configuration vectors**, and the associated space as **arm configuration space**. The physical state associated with an arm configuration vector will be referred to as the physical configuration of the arm. Consider the space that describes the human arm. Human arms can move with six degrees of freedom — two degrees of freedom are provided by the shoulder, one by the elbow, and three by the wrist. Three complicated joints determine human arm movements, which we can logically describe as combinations of six simple revolute joints as shown in Figure 4.1. The human arm configuration space is therefore six-dimensional. An additional four degrees of freedom are available in each finger, however we will imagine that the hand is held rigid with the index finger extended.

Figure 4.1 Human Arms have 6 Degrees-Of-Freedom

The physical configuration of the arm determines the position and orientation of the hand in the workspace. We will refer to the position of the hand by a reference point located at its tip called the hand's **point-of-touch**. As we saw in section 3.3.1.1., the forward kinematics of the arm determine how each arm configuration vector is related to its associated point-of-touch. In general, there may be many arm configuration vectors that position the point-of-touch at the same point in the workspace⁴.

If we are given a target point within the workspace of an arm, we can represent this point by one or more arm configuration vectors that position the point-of-touch of the hand at the target point.

4.1.3. Arm Movements Are Curves Through Arm Configuration Space

Consider the simple three degree-of-freedom arm and associated three-dimensional configuration space shown in Figure 4.2a. The X-axis of this space represents the angular displacement of a revolute joint J_1 , the Y-axis represents the Euclidian displacement of a prismatic joint J_2 , and the Z-axis represents the angular displacement of a revolute joint J_3 . The physical configurations of the arm associated with the two arm configuration vectors A and B are shown in Figure 4.2b, and their points-of-touch are labeled A' and B' respectively. Suppose that we have a curve that extends from arm configuration vector A to arm configuration vector B. If we examine the physical arm configurations that occur as we move along this curve, we will find that they correspond to a smooth and continuous movement of the arm, rather like consecutive frames in a motion picture. Figure 4.2c illustrates the physical configurations that are associated with this trajectory. By sampling the curve at smaller and smaller intervals, we can make this apparent motion as smooth as desired. Furthermore, by examining the arm configuration vectors in sequence as they occur along the curve, the motion of the arm is guaranteed to take the point-of-touch from A' to B'. Notice however, that the path traced by the point-of-touch is not the same shape as the arm configuration space curve. For example, a straight line path from A to B would result in a curved path through the workspace.

⁴ There can be an infinite number of arm configurations that touch the same point in the workspace.

(a) A 3-DOF Arm and Arm Configuration Space

(b) Arm Configurations Corresponding to the End-Points of the Arm Configuration Space Curve

(c) The Arm Trajectory Associated with the Arm Configuration Space Curve

Figure 4.2

Arm Trajectories are Curves in Arm Configuration Space

4.1.4. Arm Movement Planning is Equivalent to Finding a Curve

The complicated problem⁶ of planning a series of coordinated arm movements is now reduced to the much simpler problem of finding an appropriate curve through arm configuration space. We will refer to these curves as **arm control paths**, and an ordered sequence of arm configuration vectors on this curve as an **arm control plan**. This sequence of vectors represents a sampling of the physical configurations that occur as the arm moves from its initial configuration to a final configuration that positions the point-of-touch at the target point. There are a number of constraints which simultaneously act on the arm control path selection process. We will devote the remainder of this section to a description of these constraints. We will find that there are two types of constraints, (1) **dynamic** and (2) **permanent**. Dynamic constraints depend on the position of the target point in the workspace of the arm and therefore change each time we plan an arm movement. Permanent constraints depend on the specific structure of the arm, and are constant.

4.1.5. Initial and Final Physical Configurations Constrain Arm Control Paths

The simplest constraints on arm control path selection, are the two end points of the curve. These end points are dynamic constraints.

The arm control path must start at the arm configuration vector associated with the physical configuration of the arm at the time the arm control path is generated⁵. Feedback sensors from the arm joints provide the components of this initial arm configuration vector⁶. This is the first constraint on the arm control plan.

The curve must end at an arm configuration vector corresponding to a physical arm configuration that positions the point-of-touch at the target point. We have already seen that there may be many arm configuration vectors that have this property. How can we hope to find such a vector? Recall that a target point can be described both by a set T of arm configuration vectors that touch it, and by the eye configuration vector f that foveates it. If we associate the eye configuration vector f with each of the arm configuration vectors in T , when the eyes glance at the target point (ie. foveate it), we will be able to associatively find a set of potential end points for the arm control curve — the contents of set T . As long as we are looking at the target point, we can find an appropriate end point. Clearly then, the associations from eye configuration space to arm configuration space constrain the set of possible curves to those that terminate at one of the vectors

⁵ Of course the robot could assume another starting position if it is generating hypothetical movements. However, we will assume for the sake of simplicity that the arm is physically in the first arm configuration before movement begins.

⁶ Humans have muscular feedback sensors called neuromuscular spindles and Golgi tendon organs. These do not directly indicate the angle of the arm but certainly do provide unique signals for each arm configuration.

in T . This is the second simultaneous constraint that needs to be satisfied in order to generate an arm control plan, and foveating the target point is an essential step in the process.

4.1.6. Joint Range Characteristics Constrain Arm Control Paths

So far we have assumed that the eye and arm configuration spaces are infinite and unbounded, however, prismatic joint ranges have definite bounds, and revolute joint ranges can also be restricted to finite angular intervals. Even unrestricted revolute joints can be thought of as repeatedly cycling through the restricted range $[0, 2\pi)$ radians. In order to capture these physical configuration restrictions, it is convenient to imagine two geometric limitations imposed on configuration spaces.

The first limitation captures the idea of a restricted range of joint movement. As Figure 4.3a illustrates, range restrictions on joint J_i can be thought of as hyper-planes perpendicular to the J_i axis that each cut the configuration space into two half spaces. The intersection of these half spaces represents an unbounded region of configuration vectors that all have J_i values within range. We will refer to this region as the **in-range joint region** of J_i . The intersection between all in-range regions represents the set of configuration vectors with all joints properly in their respective ranges of operation. We will call this — possibly unbounded — region the **in-range configuration region**. All other regions will be referred to as **out-of-range configuration regions**. Configuration vectors that reside outside of this region correspond to impossible physical configurations. Figure 4.3b illustrates the in-range configuration region for a two degree-of-freedom arm with two restricted joint ranges.

The second limitation captures the idea of unrestricted revolute joints that cyclically revolve through the same angular range $[0, 2\pi)$. In this case, as the joint crosses either range boundary, it wraps around to the other end of the range. For example as we move from 0 to 2π radians, and finally cross the 2π boundary we are back to 0 radians once again. It is as though the joint axis forms a connected circle, and the configuration space closes upon itself at the boundaries of the axis. To make this idea clearer, Figure 4.4 illustrates two example spaces for two degree-of-freedom arms that contain unrestricted joints. The toroidal space in Figure 4.4a represents an arm with two unrestricted joints. The cylindrical space in Figure 4.4b represents an arm with only one unrestricted joint. Higher dimensional configuration spaces have similar cylindrical and toroidal characteristics which are understandably difficult to illustrate.

(a) J1 Restricted to $[\pi/2, 3\pi/2]$

(b) J1 and J2 Restricted to $[\pi/2, 3\pi/2]$

Figure 4.3 In-Range Constraint Regions

(a) Two Unrestricted Joints Form a Toroidal Arm Configuration Space

(b) One Unrestricted Joint Forms a Cylindrical Arm Configuration Space

Figure 4.4 Unrestricted Joint Spaces

Clearly, arm control paths must not intersect out-of-range configuration regions or the arm will attempt to assume an impossible physical configuration. This then is the third constraint to be satisfied for the collision free path planning problem. Also, we must allow arm control paths to wrap around the space in directions of unrestricted movement. These joint characteristics both result from the physical structure of the arm and therefore represent permanent constraints.

4.1.7. Repeatable Configuration Restrictions Constrain Arm Control Paths

It is possible for arm movements to be restricted by constraints other than those imposed by limited joint ranges. In other words, the in-range configuration region may contain arm configuration vectors that represent impossible physical arm configurations. This situation may result from a number of causes. However, there are two primary ones: (1) mechanical defects in the arm, and (2) the shape of the body the arm is attached to (if any).

Although the movement of a joint may be unrestricted, it sometimes happens that mechanical defects can prevent movement of the joint when the arm is in certain positions. For example, gravity may cause a burr in the link of a mechanical arm to prevent the link from sliding past another link if the arm is in the position shown in Figure 4.5a. This effectively limits the range of movement of the associated revolute joint J_i . If the arm is inverted as in Figure 4.5b however, gravity may free the link so that J_i is free to revolve without interruption from the burr. In the imperfect world in which we live, similar repeatable faults are not uncommon, especially when equipment is subjected to heavy wear and physical abuse. We are not concerned here with intermittent, or unreproducible faults, but rather with those faults that repeatedly occur when the arm is in a certain physical configuration. Repeatable faults may lead to arm configuration vectors that the arm will never physically attain because it is prevented from doing so by fault induced joint range restrictions. This situation produces regions in arm configuration space analogous to the out-of-range regions described in section 4.1.6.

Another source of repeatable restrictions on arm movement is the body that the arm is attached to. In this case, an infinite number of arm configuration vectors correspond to physically impossible configurations that reach inside or through the body. Once again, the effect of the body is to introduce regions of illegal arm configuration vectors in arm configuration space.

We will refer to the regions produced by repeatable faults and the body as **restricted configuration regions**. The fourth constraint on arm control paths is that they must not intersect restricted configuration regions. This constraint ensures that the arm control plan takes repeatable mechanical faults into consideration.

Finally, although the arm may be able to assume all possible physical configurations, defects may restrict the ways that the arm can move to a configuration. For example, a joint at 0 radians

may not be able to move directly to Q_1 radians if a mechanical fault prevents movement in that direction. Instead, it may have to revolve through approximately 2π radians in the opposite direction of revolution to achieve this configuration. We will represent these points of restricted movement as curves in arm configuration space called **movement restriction curves**. The fifth constraint on arm control paths is that they must not cross a movement restriction curve.

Figure 4.5 A Repeatab Fault

Both types of restrictions on the arm control path are related to the structure of the arm, and are therefore permanent constraints.

4.1.8. Obstacles Constrain Arm Control Paths

Collisions correspond to intersections of the arm with an object. In order to prevent collisions, we must not plan arm movements that intersect obstacles in the workspace of the arm. Notice that for a given obstacle o_i , there is a set of arm configuration vectors $C(o_i)$ such that each vector is associated with a physical configuration that intersects o_i . It is not important where the obstacle

intersects the arm, or by what amount — any intersection represents a collision. The vectors in $C(o_i)$ map into arm configuration space as one or more irregularly shaped regions. We will refer to these regions as **obstacle regions**. The sixth and last constraint on arm control path selection is that paths must not intersect obstacle regions. Intersecting paths correspond to trajectories of the arm that collide with obstacles. This is a dynamic constraint because the obstacle regions depend on the exact number, shape and position of obstacles within the workspace. In most environments, obstacles are impermanent features of the workspace.

4.1.9. Finding Obstacle Regions

The regions and curves that represent permanent constraints are permanent features of arm control space. Unless the physical structure of the arm changes, these regions and curves will not change. The regions that represent dynamic constraints, however, must be derived from sensory data each time an arm control path is to be found. In section 4.1.5. we saw that the starting point of the curve is derived from joint feedback sensors. We also saw that inter-spacial associations between vectors in eye control space and vectors in arm control space allow us to find end points of the arm control path from eye angle sensors. We require a similar mechanism to find obstacle regions from sensory data.

Let us define a new space, called **observation space**. This space is an image of the workspace. Each workspace point has a corresponding observation space point. Points outside of the workspace are not represented by observation space. This space is used to find obstacle regions in arm configuration space from sensory data.

We will first associate every point in this space with a set of arm configuration vectors, such that for every arm configuration vector, the volume of the arm includes that point. When an obstacle is introduced into the workspace, we can find the set of observation space points that are included in its volume. The union of the arm configuration vectors associated with these points represents the obstacle region for this obstacle within arm configuration space.

When an obstacle o_i appears in the workspace, there will be a set of observation space points $V(o_i)$ such that each point in $V(o_i)$ corresponds to a point in the workspace intersected by o_i . $V(o_i)$ therefore represents the volume of the obstacle that intersects the workspace, and we will call the set the **obstacle volume set**. Observe that the arm is also a physical object which intersects the workspace. Let us refer to the volume of the arm that currently intersects the workspace by the set of observation space points $V(a_j)$, where a_j is the arm configuration vector that represents the physical configuration of the arm. We will call these **arm volume sets**. The contents of an arm volume set depend only on the physical shape of the arm. Unless the physical structure of the arm changes, these observation space sets will not change.

Let us assume that we know the contents of every arm volume set for the arm we are controlling. Let us further assume that we can determine the contents of each obstacle volume set from visual sensory data. Clearly, if *any* obstacle volume set intersects an arm volume set $V(a_j)$, then the arm configuration vector a_j represents a physical arm configuration that collides with at least one obstacle in the workspace. For each observation space point p_k , we can associate p_k with a set of arm configuration space vectors $W(p_k)$ such that p_k is included in the arm volume sets of these vectors

$$W(p_k) = \{a_j \mid \forall a_j \in \text{arm configuration space}, p_k \in V(a_j)\}.$$

The obstacle region $R(o_i)$ for obstacle o_i is therefore the union of sets $W(p_k)$ for each observation point p_k that is contained in the volume of o_i

$$R(o_i) = \bigcup_{\forall p_k \in V(o_i)} W(p_k).$$

4.1.10. Further Constraints

In addition to the fundamental constraints discussed above, there are weak constraints that may aid in the selection of arm control paths. For example, very long arm control paths often correspond to indirect point-of-touch paths that wander about before terminating at the target point. Shorter arm control paths tend to conform to more direct and purposeful movements of the arm. We could add the constraint that arm control paths must also be the shortest curve that satisfies all other constraints. For the purpose of this chapter, we will not concern ourselves with such weak constraints. It is more important that we understand fundamental constraints which we can later augment to make the final arm movement more efficient. Henceforth we will consider valid any arm control path that satisfies the preceding fundamental constraints.

4.2. A Neural Controller for Collision-Free Movement of Robot Manipulators

So far we have seen that there are six important constraints which must be satisfied when generating an arm control plan. Three of these are permanent constraints determined by the physical structure of the arm, and three are dynamic constraints determined by the location of the target point. Of these dynamic constraints, two are determined from sensor data with the aid of inter-spacial associations. To summarize, the six restrictions are:

1. The control path must begin at the arm configuration vector associated with the arm's initial physical configuration. The components of this vector are derived directly from joint position sensors.
2. The control path must terminate at an arm configuration vector that positions the arm's point-of-touch at the target point in the workspace. A set of such vectors can be

associatively derived from the single eye configuration vector that foveates the target point. In other words, we must look at the target point before we can touch it.

3. The control path must not intersect out-of-range configuration regions.
4. The control path must not intersect restricted configuration regions.
5. The control path must not intersect movement restriction curves. -
6. The control path must not intersect obstacle regions. The arm configuration vectors associated with obstacle regions can be associatively derived from points in the observation space included in the obstacle volume.

We have developed a highly visual, geometric interpretation of constraints as points, curves, and regions in arm configuration space. This geometric outlook is a powerful conceptual tool which we will now use to design a neural arm controller for the collision-free control of robot manipulators. The reader is encouraged to acquaint himself with the terms and visual metaphors that were introduced in section 4.1. before proceeding.

4.2.1. The Robot Architecture

A robot arm controller is only one component of a larger system. Figure 4.6 is a schematic representation of the high-level architecture of the adaptive robot. The system is composed of five components; the physical robot, a high-level planning system which selects target objects, two simple algorithmic controllers that provide reflex actions during training, and the adaptive robot arm controller itself.

So far, we have assumed that the workspace of the arm is three-dimensional. In order to simplify the following presentation, we will limit our discussion to a simple 2-dimensional Horn robot with a 2 degree-of-freedom, revolute-jointed arm. In section 4.4.4 we will describe how the controller can be extended to operate in a three-dimensional space, with a multi-degree-of-freedom robot arm. It should be emphasized that the controller can also be extended to handle arms of any number of degrees-of-freedom. Finally, note that the arm links may be of arbitrary shape and the arm joints may be either revolute or prismatic.

4.2.1.1. The Physical Robot

Figure 4.6a depicts the physical robot being controlled. The robot consists of three parts. The first is a body which may be of any arbitrary shape and as complex as desired. The second part is a J degree-of-freedom manipulator which terminates in some form of grasping device. The third part is a set of sensors C_1 through C_n .

The manipulator and sensors do not need to be mounted on the body as long as the first joint of the arm, and the sensor bases are stationary with respect to the body.

The joints J_1 and J_2 each have joint angle sensors which indicate the amounts, θ_1 and θ_2 , by which the associated links L_1 and L_2 are offset from their origin positions. The vector of joint sensor outputs is an arm configuration vector as described in section 4.1.2. The arm configuration space for the 2 degree-of-freedom robot shown in Figure 4.6a is therefore two-dimensional.

(a) The Physical Robot

(b) The Robot Control Systems

Figure 4.6 Architecture of the Adaptive Robot

The sensors must have a clear line of sight to the end-effector. Any type of sensor may be used as long as they collectively provide signals that map one-to-one or many-to-one with target points in the workspace. The same output signal cannot represent more than one target point. For example, laser range finders provide unique outputs for each point in the workspace. Cartesian coordinate information can be extracted from depth maps provided by stereo cameras. Only two output components are needed to specify the direction and distance of target points in a two-dimensional workspace. For the remainder of this chapter we will assume that the sensors are stereo cameras and that the two output signals represent the angles of the camera motors required to center (foveate) the target point in the camera image. We will refer to these cameras as the eyes of the robot and the motor angles of the cameras as the eye angles Ψ_1 and Ψ_2 . The vector of eye angles is an eye configuration vector as described in section 4.1.1., and the corresponding eye configuration space of the robot is two-dimensional. The vector of output components of any type of sensor would also be referred to as an eye configuration vector. Figure 4.6b depicts the four control systems of the robot.

4.2.1.2. The Task Level Controller (TLC)

The task level controller (TLC), is an intelligent, cognitive-level controller which is responsible for determining the overall goal of the robot, and associated task planning.

The robot arm controller we shall describe is merely a tool used by the task level controller. The TLC selects a target point and orients the eyes toward it. It is by glancing at the point that the TLC directs the arm to touch it with the end-effector. This type of **intentional command** is one of the capabilities required to achieve task-level robot programming. Note that, rather than specifying target points in an absolute universal co-ordinate frame, the system employs body-centered eye configuration space vectors.

This thesis will not concern itself with the TLC.

4.2.1.3. The End Effector Tracker (EET)

The end-effector tracker (EET) is a primitive controller that orients the eyes toward the tip of the end-effector whenever the TLC is not otherwise determining the point-of-gaze. The EET is ignorant of the kinematics of the arm. It merely tracks a moving point in space that is coincident with the point-of-touch. The EET might perform this function by tracking a special light source or pattern on the end-effector. The function of the EET is similar to the human autonomic startle reflex which orients the eyes toward a sudden flash of light or movement in the visual field.

We will discuss the role of the EET further in section 4.3.1.

4.2.1.4. The Random Movement Generator (RMG)

The random movement generator (RMG) generates random movements of the arm. In the simplest case, the RMG is a random number generator which continuously changes each joint angle by a small random amount. We assume that if the RMG brings the arm into contact with an obstacle, force sensors prevent the arm from being damaged until its random movements free it. This is merely a reflex action which does not require knowledge of the arm kinematics. A force sensor monitors the joint motor to detect resistance to movement, and reduces current to the motor until resistance ceases because the RMG has moved the arm in another direction. The EET will track the end-effector as the arm moves it through a representative number of workspace points.

We will discuss the role of the RMG further in section 4.3.1.

4.2.1.5. The Adaptive Arm Controller (AAC)

The focus of this section is the adaptive arm controller (AAC). This controller receives joint angle feedback, arm configuration vectors (from the arm, eye angle feedback), eye configuration vectors (from the eyes), and visual information regarding the positions of objects in the workspace. Eye angle vectors could be input from the TLC, rather than directly from the eye angle sensors. This arrangement would allow the TLC to propose target points that the eyes are not currently viewing. However, we will simplify the discussion by assuming the simpler arrangement shown in Figure 4.6.

After generating an arm control plan, the AAC outputs a sequence of configuration vectors to the arm. Low level joint motor controllers move the arm joints to the positions given by the components of each vector. The end result is that the arm moves through the ordered sequence of positions specified by the AAC. Movement of the arm between output vectors is determined by the following simple joint-interpolation algorithm. Each joint, starting at the position specified by a component of the first vector, moves to the position specified by the second vector. All joints take the same amount of time to complete their movements so that the motion of the arm between each vector is fluid, not jerky. To accomplish this, the velocity with which each joint moves is determined by the maximum velocity of the slowest joint J_s . If the old position of joint J_i is θ_i , and the new position of J_i is θ_i' , then the amount of time taken to perform the movement is determined by the velocity of the slowest joint as follows

$$T_s = \frac{|\theta_s - \theta_s'|}{V_s},$$

and the velocity of each joint is given by

$$V_i = \frac{|\theta_i - \theta_i'|}{T_s}.$$

Let us now consider the design of the AAC.

4.3. The Adaptive Arm Controller Design

We have seen that the collision free arm movement problem can be expressed as the constrained selection of a curve in arm configuration space. Furthermore, we have seen that these constraints can be expressed as (1) permanent regions and curves in arm configuration space that are determined by the physical structure of the arm and body, and (2) inter-spacial associations between points in sensory spaces and vectors in the arm configuration space. The details of these associations are in turn determined by relationships between the physical structure of the arm and the type of sensory equipment. We would like to be able to control robots with arbitrary body shapes, manipulator designs, and sensor types. In addition, as we saw in chapter 3, the real world can change supposedly permanent attributes of the robot — ie. changes in arm kinematics and sensor characteristics. This implies the need for a robot arm controller that can autonomously discover the regions and curves that represent structural constraints, and the inter-spacial associations that represent sensory constraints.

In chapter 2 we saw that the brain exhibits computational properties that are well suited to handling problems of this type, namely: (1) quick simultaneous constraint satisfaction, (2) inter-spacial association, and (3) the ability to self-organize constraint structures that are customized to individual physical peculiarities. It is therefore expedient to turn to computational models based on the architecture of the brain — ie. neural learning systems.

4.3.1. Finding Constraint Structures is a Learning Machine Problem

Let us now describe the process of finding and maintaining these constraint structures as a learning machine problem. What kind of training samples are needed to learn these structures, and how can they be provided by an unsupervised learning system?

The first constraint involves finding the start point of the arm control path. This point is always found from joint position sensor data at the time the arm control plan is made. Therefore, learning is not required to find this constraint.

The second constraint involves finding an arm configuration vector associatively from an eye configuration vector such that the point-of-touch of the arm configuration vector coincides with the point-of-gaze of the eye configuration vector. In other words, our training samples must contain an arm configuration-eye configuration vector pair in which the eyes are foveating the same workspace point that the physical arm touches.

The third constraint involves learning where out-of-bound regions are in the arm configuration space. To do this we need to distinguish between in-range and out-of-range arm configuration

space regions. Our training samples must indicate when arm configuration vectors are located at the boundaries of the out-of-range regions.

The fourth constraint involves learning where restricted configuration regions are in the arm configuration space. As long as our training samples do not include arm configuration vectors that are impossible, the absence of impossible vectors — ie. restricted configuration region vectors — implies the location of these regions.

The fifth constraint involves learning where movement restriction curves are in the arm configuration space. This is equivalent to finding the boundaries of out-of-range regions produced by joint range restrictions. Our training samples must indicate when arm configuration vectors are located on these curves, thereby defining their position and shape.

The sixth and final constraint involves learning collision avoidance constraints. This involves finding associations between points in observation space and sets of arm configuration vectors in which the volume of the physical arm contains those points. The training samples must contain an arm configuration vector and the set of observation space points contained by the volume of the arm. As long as the set of training samples covers all legal arm configuration vectors, we will be able to construct an associative mapping between points in observation space and vectors in arm configuration space.

What sort of learning behaviours are required to supply enough training samples that contain the necessary data? There are three requirements:

1. The arm must move through a representative set of arm configuration vectors. By this we mean that the arm must move through the entire workspace⁷. The order of these training samples is not important. Therefore, random arm movements are sufficient to eventually move the arm through a representative set of configurations. These movements are comparable to the uncoordinated, and largely random movements made by newborn babies during critical periods of development.
2. The eyes must track the point-of-touch as the arm moves, providing eye and arm configurations that can be associated to later find the potential final configurations of the arm.
3. The joint sensors must include strain gauges that indicate resistance to random movement — caused by reaching a joint limit, by hitting the robot's body, or by encountering a repeatable structural fault.

In addition, we want all of these requirements to be met by an unsupervised learning system. This need is satisfied by the EET and RMG controllers, which are simple, autonomous,

⁷ As we will see in sections 4.5.4.3. and 4.5.4.5., it is not necessary to move through every configuration — just a subset.

algorithmic devices that do not require human supervision. The EET and RMG are only needed during an initial training phase while the controller learns constraint structures.

Let us now examine how a neural learning system can be constructed to fulfil these constraint satisfaction and learning requirements.

4.3.2. The Adaptive Arm Controller Architecture

There are three issues to be addressed in the design of the AAC architecture: (1) how to represent the three spaces introduced in section 4.1. — the observation space, the arm configuration space, and the eye configuration space; ie. how to represent constraint structures and inter-spacial associations; (2) how to learn these constraint structures from random movements of the arm; (3) how to use these structures to find a constrained arm control path that can be used to generate an arm trajectory to drive the joint motors. Let us examine each of these issues and propose neural mechanisms that satisfy the system requirements. We will then formally define the arm controller model, and describe the detailed operation of the system.

4.3.2.1. Representing the Three Arm Control Spaces

Each space will be represented by a network of neurons. This approach is supported by evidence that the brain also uses specialized networks of neurons to represent each sensory and motor control space (sections 2.2.1.1. and 2.4.2.). Neurons in these networks will be arranged so that the organizational dimension of the network is the same as the dimension of the space that it represents — we will examine this further in section 4.4.4. The three spaces associated with the simple 2-dimensional, 2-degree-of-freedom Horn robot are each two-dimensional. We will assume for the time being that the AAC is constructed from three, two-dimensional, planar neural networks. We can think of the AAC as a layered, functional level neural model.

It is impossible to directly represent an infinite number of points in a space. Therefore we must quantize each space into a set of volumetric regions, called **quanta**, that completely cover the space. Some quanta may be unbounded regions, others may be bounded on all sides. Every point in the space must reside in one quanta. The simplest representational scheme is to make each neuron in the network sensitive to one of these quanta. In other words, the receptive field of each neuron is a quanta. This situation is very similar to that found in the primary sensory regions within the cortical-sheet. Each neuron in such a sheet is sensitive to a specific region of the sensory space (sections 2.2.1.1. and 2.2.1.2.). We will call a network that represents a space in this way a **map** of the space. The network representations of the three arm control spaces are therefore called the **observation map**, the **arm configuration map** and the **eye configuration map**. Neurons in these maps are called **observation neurons**, **arm configuration neurons** and **eye configuration neurons** respectively.

The finer the quantization of a space — ie. the more neurons there are in a map — the more accurate our approximation of the original space. However, we should keep in mind that our maps are only approximate representations of the original spaces, and we can expect some amount of imprecision, called **quantization error**, to be introduced into the AAC's ability to represent constraint structures. We will examine the effects of quantization error in section 4.5.

There are slightly different representational requirements for each space. We will now identify these requirements and find appropriate neural mechanisms to implement each map.

4.3.2.1.1. The Representational Requirements of the Observation Map

The observation map must represent the volume of the workspace currently occupied by objects. This is accomplished by making observation map neurons sensitive to the presence or absence of an object in their receptive field. If any part of an object — e.g. an obstacle or the arm — intersects a receptive field, the neuron will output a signal. If the receptive field is empty, the neuron will not output a signal.

The simplest way to implement such a map is to suspend a digital camera over the two-dimensional workspace of our robot, so that the image produced by the camera just covers the workspace of the robot. The camera produces a high contrast image of the workspace — possibly with the aid of back lighting — such that objects appear in the image as dark pixels and the empty workspace appears as light pixels (Figure 4.7). Each pixel that sees a part of the workspace represents a quanta of observation space. Each neuron of the observation map receives as input the signal produced by a pixel that 'sees' a quanta of observation space. If the pixel is dark, the neuron outputs a signal, otherwise it does not output a signal.

The set of neurons that output a signal at any given time is a distributed representation of the volume of the workspace occupied by objects. If the observation map contained an infinite number of neurons, this set would be unique for every arrangement of the arm and obstacles in the workspace. Since we have a finite number of neurons, the same set may represent similar arm-obstacle arrangements. It does not matter which neuron in the observation map corresponds to a given pixel because the topography of the observation space is not a constraint on the arm control problem.

The arm configuration map must be capable of representing two things: (1) the points, curves and regions associated with constraints, and (2) arm control paths — curves that avoid intersecting the constraint structures. The arm configuration map is an image of arm configuration space which represents these geometric structures. It is topographically ordered, so that neurons which are neighbours in the physical map have receptive fields that are close neighbours in the arm configuration space.

Points in arm configuration space have associated states — they either belong to one of the constraint structures, or they represent points that could lie on a valid arm control path. Similarly, the neurons in the arm configuration map represent small regions of the space that either contain points that are part of a constraint structure or contain only points that could be included in a valid arm control path. Constraint structures are therefore directly represented by a pattern of neurons in the network, and arm control paths are represented by a 'path' — a chain of neurons — through the network.

The Robot Workspace

Resulting Mo Activation
Pattern Represents the
Occupied Volume of the
Workspace

Figure 4.7 The Observation Map as A Camera Image

4.3.2.1.2. The Representational Requirements of the Arm Configuration Map

Topographical ordering is important because it preserves the important geometrical relationships between constraint structures and arm control paths. If the network is a good approximation of the arm control space, we can find an arm control path by (1) locating two neurons whose receptive fields contain the end points of the curve, and (2) constructing a connected chain of neurons from the start neuron to the end neuron such that the neurons in the chain do not contain constraint structures in their receptive fields. Any curve that is confined to these receptive field regions in arm configuration space is guaranteed not to intersect a constraint structure, and therefore represents a collision free movement. The AAC must find one such curve, so we will select one point from each region — any point will do — and we will generate a curve that passes through these points. Straight-line interpolation between the points is the most

straight-forward approach. If our curve ventured into a receptive field that contained some constraint structure points, it might include such a point and collide with an obstacle or attempt to assume an impossible physical configuration. In section 4.5.5. we will see that there is another limitation on the curve selection process that prevents this from happening.

Figure 4.8 Arm Configuration Map Receptive Fields

So far we have seen that the more neurons we use in a map, the better our approximation of the associated space will be. Permanent constraint regions can be quite large. It would be an inefficient use of neurons to devote a large proportion of our available supply to represent these structures.

Arm control paths will never pass through these regions and the interiors of the regions are 'dead zones' that assist in path selection. A better solution is to devote neurons only to the borders of these regions. This leads to the situation depicted by Figure 4.8 in which the regions 'collapse' to much smaller collections of neurons in the network. Figure 4.8a shows the effect of devoting neurons to a uniform coverage of the arm configuration space. On the left we see the receptive fields of the neurons in the arm configuration space and on the right we see the pattern of neurons in the network that have receptive fields intersecting the constraint region (shaded in gray). In Figure 4.8b we see the same network with receptive fields covering the boundary of the region only. Notice that the number of 'wasted' neurons is very small. Most of the neurons are devoted to covering the rest of configuration space more accurately. The border neurons are necessary to prevent arm control paths (derived from chains of neurons) from crossing the constraint region. As Figure 4.9a shows, without border neurons, a path through the network can correspond to a path that intersects the region. Figure 4.9b shows that the border neurons steer the path away from the constraint region entirely.

Without permanently constrained neurons, paths through the network can correspond to arm control paths that cross regions representing impossible arm configurations.

Network paths that avoid permanently constrained neurons correspond to arm control paths that avoid crossing regions representing impossible arm

Figure 4.9 Network Paths in Arm Configuration Space

4.3.2.1.3. The Representational Requirements of the Eye Configuration Map

The eye configuration map represents the eye configuration space. Each neuron is sensitive to a small region of eye configuration space. If the current eye configuration vector falls in the receptive field of a neuron, the neuron will output a signal. Otherwise the neuron will not output a signal. One and only one neuron will be active at any time.

There are many non-foveating eye configurations that do not correspond to target points in the workspace (section 4.1.1). The arm control problem is not concerned with eye configuration vectors associated with non-foveating eye configurations. It would be wasteful to have neurons that are only sensitive to non-foveating eye configuration vectors. Therefore a requirement of the eye configuration map is to devote all receptive fields of eye configuration map neurons to regions of eye configuration space containing foveating eye configuration vectors.

4.3.2.1.4. Configuration Maps are Kohonen Maps

We have identified three requirements for the arm configuration map: (1) each neuron must have an associated vector within its receptive field that can be used for straight-line interpolation of arm control paths, (2) the map must not devote neurons to the interiors of constraint regions (although it should devote neurons to the boundaries of such regions), and (3) the map must be a topographically ordered representation of the space. The Kohonen topology preserving map model is particularly well suited to handle these three requirements (see section 2.4.3.). Inputs to the arm configuration map (or Kohonen network) are arm configuration vectors taken from the joint position sensors.

Each neuron in the Kohonen model has an associated weight vector with the same dimensionality as the input pattern vectors. Consequently, these weight vectors can be considered to reside in the input space. Furthermore, the position of a neuron's weight vector determines its behaviour — one neuron in the network will win the competition to represent an input vector, and this is always the neuron with the closest weight vector to the input vector. These properties satisfy the first representational requirement of the arm configuration map. The neuron's receptive field will be defined by those inputs that make the neuron win the Kohonen competition. These are the arm configuration vectors that are nearest neighbours to the neuron's weight vector. The arm control path interpolation point for the neuron is given by its weight vector. This type of quantization by neighbourhood relationships is called **vector quantization**.

The arm configuration vectors continuously generated by the joint position sensors on the arm are sampled at regular intervals and presented to the network. During training, as the arm moves randomly through the workspace, these vectors will be distributed throughout the regions of arm configuration space that correspond to possible physical configurations of the arm. As we saw in

section 2.4.3., the structure mimicry property of Kohonen's learning algorithm is that it distributes weight vectors throughout the input space — the arm configuration space in this case — with the same local point density as the training samples. This means that the majority of receptive fields — containing the weight vectors — will be devoted to regions outside of the constraint regions. Those weight vectors near the boundaries of the constraint regions will satisfy the second representational requirement.

The third representational requirement is met by the topology preserving property of Kohonen's algorithm. After learning is complete, the topology of the network is preserved within input space by the neighbourhood relationships between weight vectors. In other words, neurons that are neighbours in the physical network will have receptive fields that are close neighbours in arm configuration space.

The two representational requirements of the eye configuration map are: (1) the neuron that detects the eye configuration vector associated with the current point-of-gaze must output a signal, and (2) no neurons should exist with receptive fields that contain only non-foveating eye configuration vectors. Once again, the properties of Kohonen's learning algorithm make it particularly suitable to implement the eye configuration map. The inputs to this network are eye configuration vectors taken from the camera angle sensors.

The neuron with the weight vector closest to the current input vector — ie. with the receptive field that contains the input vector — wins the competition to represent the input. If we modify the Kohonen output function slightly so that it outputs a signal only if the neuron wins the competition, the first representational requirement is satisfied for the eye configuration map. The second requirement is satisfied by the structure mimicry property of Kohonen networks. During training, as the EET causes the eyes to track the point-of-touch, the training samples received by the network will be distributed to those regions of the eye configuration space that contain foveating eye configurations. Consequently, weight vectors will be distributed to these same regions, and the receptive fields of neurons will therefore contain foveating eye configurations.

4.3.2.1.5. Representing Inter-Spatial Associations

Inter-spatial associations are associations between points in two spaces. We will represent associations by 'growing' a physical connection between neurons whose receptive fields contain the two points. Then by sending an output signal from a neuron u_i along these connections, the signal will activate the neurons that u_i is connected to, thereby 'associatively' relating u_i with this set of neurons. As we saw in section 2.4.1.1. connection growth is equivalent to modifying a connection strength of zero to a non-zero value.

In order to make appropriate connections between maps, we require a Hebbian rule that describes the growth and decay of these physical connections in terms of a connection strength value — ie. the weight of the connection. Intuitively, connections are grown between neurons that are outputting a signal simultaneously. This is consistent with Hebb's law described in section 2.2.3. Connections must also be allowed to slowly decay over time if they are not periodically reinforced by the two neurons outputting simultaneous signals. Connection decay allows associations that are not periodically reinforced to be 'forgotten' — ie. the connection will eventually disappear. This is necessary if the associations between spaces change due to some unforeseen circumstance.

An eye configuration vector that foveates on a target point needs to be associated with all arm configuration vectors that touch the same point. During training, as the EET tracks the point-of-touch, the eye configuration map will receive from the eye motors the eye configuration vector that is to be associated with the arm configuration vector that the arm configuration map receives from the joint position sensors. Both maps are Kohonen networks. As we have seen, one neuron in each map whose receptive field contains the input vector will be activated. The Hebbian rule described above will therefore grow a connection between these neurons. If the arm moves through its entire sequence of physical configurations that touch that target point, then those arm configuration neurons will all be associated with the eye configuration neuron which foveates on that target point. This process will grow appropriate associative connections for all target points.

Similarly, connections are grown between neurons that are simultaneously active in the observation and arm configuration maps. Let us assume that the arm is the only thing in the workspace during training. As we have seen, a set of neurons that represent the region of the workspace intersected by the volume of the arm will be active in the observation map. The one neuron in the arm configuration map whose receptive field contains the arm configuration vector for the physical arm configuration observed will be active at the same time. Therefore, connections will be grown from all these observation neurons to the single arm configuration neuron. If the arm moves through all physical configurations, then every arm configuration neuron will be associated with all the observation neurons that represent its volume. If we look at these connections from the point of view of a single observation neuron, its connections allow it to associatively activate all the arm configuration neurons that best represent the arm configuration vectors whose volumes intersect at least one point in the observation neuron's receptive field.

The connection growth process satisfies the inter-spacial association requirements described in section 4.3.2.1.5.

4.3.2.1.6. Boundary Inhibition

So far, we have not described how we will distinguish neurons with receptive fields containing permanent constraint structures from other neurons. We can use the same inter-spatial associative law to grow inhibitory connections from a special neuron called the **permanent constraint neuron** to neurons whose receptive fields include points in out-of-range regions, restricted configuration regions, and movement restriction curves. This process works as follows: the permanent constraint neuron is sensitive to resistance to movement — it only outputs a signal when the strain gauge in the joints indicate resistance in the training sample data. Arm configuration neurons that are active at the same time as this neuron must reside at the boundary of one of these constraint structures. If the arm is allowed to randomly wander long enough, it will eventually encounter all the limitations on its freedom to move. Therefore we can expect all neurons bordering the constraint regions to eventually output a signal at the same time as the permanent constraint neuron. The associative law will therefore grow connections between these neurons. After training is complete, the permanent constraint neuron is permanently activated, and its associative laws are disabled. The signal that it sends down these connections will inhibit all constraint structure neurons. The path finding process must therefore avoid paths to neurons that are inhibited.

4.3.2.2. The AAC Model

We will now present a more formal description of the AAC architecture — introducing notation, and mathematical descriptions of system behaviour.

4.3.2.2.1. The Functional Model

Figures 4.10 to 4.12 illustrate the functional level architecture of the AAC. As illustrated by Figure 4.10, the AAC consists of three network layers called **maps**. The notational device used to refer to the three network layers will be an upper case 'M' — for map — followed by a lower case subscript which denotes the type of cell in the map. The observation map with neurons of cell type 'o' — for observation — is designated by M_o . The arm configuration map with neurons of cell type 'a' — for arm — is designated by M_a . The eye configuration map with neurons of cell type 'e' — for eye — is designated by M_e .

Figure 4.10 The Three Maps of the AAC

Figure 4.11 Inter-Map Associative Connections

4.3.2.2.2.

Figure 4.12 The Permanent Constraint Neuron

Every neuron in M_0 can connect to every neuron in M_a as shown in Figure 4.11. Similarly, every neuron in M_e can connect to every neuron in M_a . We will refer to these inter-spatial associative connections as **inter-map connections**. Before the training phase begins, all inter-map connection weights are initialized to 0.0, indicating no connection. We will employ a biological metaphor and speak as if connections do not initially exist, but are grown between maps during learning. We refer to the connection between M_0 neuron o_i and M_a neuron a_j by $\omega(o_i, a_j)$ and the connection between M_e neuron e_k and M_a neuron a_j by $\omega(e_k, a_j)$. The permanent constraint neuron is denoted by p_1 , and is shown in Figure 4.12.

The AAC is a synchronous (clocked) model.

The Network Model

M_0 is an unconnected network that consists of a single layer of neurons described in section 4.3.2.1.1. We will refer to the set of neurons in M_0 as

$$O = \{o_1, o_2, o_3 \dots o_{n_0}\},$$

where $N(M_0) = n_0$ = the number of neurons in M_0 . Inputs to M_0 arrive from a set of CCD pixels in a digital camera. We will refer to these pixels as the set of sensory cells

$$S = \{s_1, s_2, s_3 \dots s_{ns}\},$$

where $N(S) = n_s$ = the number of sensory cells. There is a one-to-one mapping between sensory cells and M_o neurons. All connection strengths between cells in S and neurons in O are set to 1.0 and are unchangeable.

M_a is a Kohonen-style network which has a single layer, unconnected perceptron architecture as described in section 4.3.2.1.4. The neuron set is denoted by

$$A = \{a_1, a_2, a_3 \dots a_{na}\}.$$

where $N(M_a) = n_a$ = the number of neurons in M_a . Network inputs arrive from joint position sensors in the arm. The set of joint position signals are designated by the J -dimensional arm configuration vector

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dots \\ \theta_J \end{bmatrix},$$

where θ_i is the signal from the i^{th} joint. In general, the dimensionality of the network is the same as that of the input vectors. As in the Kohonen model, all neurons receive the same inputs. We shall refer to the strengths of connections between the joint position sensors and M_a neurons by $\mu(\theta_i, a_j)$. We will refer to the vector of connection strengths received by neuron a_j as

$$\mu(a_j) = \begin{bmatrix} \mu(\theta_1, a_j) \\ \mu(\theta_2, a_j) \\ \dots \\ \mu(\theta_J, a_j) \end{bmatrix}.$$

Outputs are directly derived from input weight vectors. Only one M_a neuron is allowed to output its weight vector at any given time as an arm trajectory vector θ' . We will discuss how these neurons are selected shortly.

$$\theta' = \begin{bmatrix} \mu(\theta_1, a_j) \\ \mu(\theta_2, a_j) \\ \dots \\ \mu(\theta_J, a_j) \end{bmatrix}.$$

M_e is also a Kohonen-style, single layer, unconnected perceptron architecture. The neuron set is denoted by

$$E = \{e_1, e_2, e_3 \dots e_{ne}\}.$$

where $N(M_e) = n_e$ = the number of neurons in M_e . Inputs to the network arrive from eye motor angle sensors. The set of eye angle signals are designated by the Y -dimensional eye configuration vector

$$\Psi = \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \dots \\ \Psi_Y \end{bmatrix},$$

where Ψ_i is the signal from the i^{th} eye. As in M_a , all neurons in M_e receive the same inputs. We shall refer to the strengths of connections between the eye angle sensors and M_e neurons by $\mu(\Psi_i, e_j)$. We will refer to the vector of connection strengths received by neuron e_j as

$$\mu(e_j) = \begin{bmatrix} \mu(\Psi_1, e_j) \\ \mu(\Psi_2, e_j) \\ \dots \\ \mu(\Psi_Y, e_j) \end{bmatrix}.$$

Figure 4.13 shows a simple network architecture for the Kohonen-style networks M_a and M_e which allows us to carry out the Kohonen competition algorithm in parallel without lateral inhibition. All neurons u_j have unique IDs arranged according to a consistent numbering plan. For example, if the network U is a square 2-dimensional network of $N(U)$ neurons, ID_j is associated with the neuron at

$$\text{row}(ID_j) = \frac{ID_j}{\sqrt{N(U)}}, \text{ and}$$

$$\text{column}(ID_j) = ID_j \bmod \sqrt{N(U)}.$$

Figure 4.13 Bus Architecture for M_a and M_e

Each neuron u_j connects to an I/O bus — the XBUS — which carries the J -dimensional arm configuration vector for M_a or the Y -dimensional eye configuration vector for M_e . Associated with

each vector component C_i and each neuron u_j is the weight $\mu(C_i, u_j)$. Each u_j also connects to a Kohonen competition bus — the CBUS. This wire has the special characteristic that it retains the smallest activation value written to it during a competition. Using this bus we can find an activation bubble for a network of any size in constant time. Each neuron performs the equivalent of the following short algorithm:

1. Time step 1 - all u_j write their real-valued activation level to the CBUS. The smallest activation level remains on the CBUS as value v_1 .
2. Time step 2 - all u_j compare their activation level with v_1 . Those with activation levels greater than v_1 drop out of the competition.
3. Time step 3 - all u_j still in the competition write their IDs to the CBUS. The smallest ID, ID_w , remains on the CBUS as value v_2 .
4. Time step 4 - all u_j read the ID of the winning neuron u_w from the CBUS. Each neuron within the bubble of neighbours surrounding the winner modifies its weights according to the learning law equations shown below in section 4.4.1.1. If the width of a 2-dimensional network is $\sqrt{N(U)}$ and the bubble radius at time t is $R_t < \sqrt{N(U)}$, the set of neurons in the bubble is

$$B_t = \{u_k \mid (\text{row}(ID_w) - R_t) \leq \text{row}(ID_k) \leq (\text{row}(ID_w) + R_t) \text{ AND} \\ (\text{column}(ID_w) - R_t) \leq \text{column}(ID_k) \leq (\text{column}(ID_w) + R_t)\}.$$

4.3.2.2.3. The Neuron Model

As explained in section 4.3.2.1.1., the sensory cells connected to M_0 neurons are naturally implemented as pixels in a camera image. We will assume that the outputs of these pixels are binary valued. As we will see in section 4.4.1. below, it is necessary to distinguish between neurons activated by the robot's arm and neurons activated by other objects. Therefore there are two sets of activation value equations.

DURING THE TRAINING PHASE:

$$a(o_i) = \begin{cases} 1.0 & \text{if arm is in receptive field of pixel} \\ 0.0 & \text{otherwise} \end{cases} \quad [4.3.2.2.3.1]$$

AFTER THE TRAINING PHASE:

$$a(o_i) = \begin{cases} 1.0 & \text{if object other than arm in receptive field of pixel} \\ 0.0 & \text{otherwise} \end{cases} \quad [4.3.2.2.3.2]$$

$$o(o_i) = a(o_i)$$

M_a neurons are based on the Kohonen neuron. Their output signal functions have been augmented to include inter-spatial association connections and to mark neurons as either the initial

configuration neuron, a target configuration neuron, an inhibited colliding configuration neuron, or a collision free configuration neuron. Neurons can be thought of as having an internal state derived from input signals which represents these different control path conditions. This state can be stored as two binary bits or, in the spirit of the neural paradigm, by the level of the output signal the neuron generates. The activation level of the neuron is simply the Euclidean distance of the neuron's weight vector from the current input vector. As we shall see, the output signal of the neuron is used to determine how connections are grown between maps during the initial training phase, therefore for the sake of clarity, we provide separate output signal functions for the training and subsequent operational phases. Close examination of these equations reveals that the training equations are simply a subset of the operational phase equations.

$$a(a_j) = ||\theta - \mu(a_j)|| \quad [4.3.2.2.3.3]$$

DURING THE TRAINING PHASE:

$$o(a_j) = 1.0 \quad \text{if } a(a_j) = \min \forall q ||\theta - \mu(a_q)|| \quad [4.3.2.2.3.4]$$

$$0.0 \quad \text{otherwise (not winner or target)}$$

AFTER THE TRAINING PHASE:

$$o(a_j) = 1.0 \quad \text{if } a(a_j) = \min \forall q ||\theta - \mu(a_q)|| \quad \text{OR} \quad [4.3.2.2.3.5]$$

$$o(e_k) \cdot \omega(e_k, a_j) > 0.0 \quad [4.3.2.2.3.6]$$

$$-1.0 \quad \text{if } o(o_i) \cdot \omega(o_i, a_j) > 0.0 \quad \text{OR}$$

$$o(p_1) \cdot \omega(p_1, a_j) > 0.0 \quad [4.3.2.2.3.7]$$

$$0.0 \quad \text{otherwise (not winner or target)} \quad [4.3.2.2.3.8]$$

During learning, equation 4.3.2.2.3.4. is used. This equation simply causes the neuron to output a signal if it is the current CBUS competition winner. This ensures that during learning, only one arm configuration neuron is active at a time. After learning, equation component 4.3.2.2.3.5. performs the competition activation. A neuron activated this way after learning represents the start point (initial configuration) of an arm control path, and is called the **initial configuration neuron**. Equation component 4.3.2.2.3.6. represents signals received from the eye configuration map. Neurons activated this way represent end points (potential target configurations) of the arm control path and are called **target configuration neurons**. Equation component 4.3.2.2.3.7. represents inhibitory signals received from the observation map and permanent constraint neuron. Neurons inhibited in this way represent dynamic and permanent constraint structure configurations that cannot be included in an arm control path. These are referred to as **inhibited neurons**. Equation component 4.3.2.2.3.8. represents the rest state of the neuron if it is not actively excited

or inhibited by inputs. These neurons represent collision-free arm configurations that can be included in potential arm control paths and are called **control path neurons**.

M_e neurons are also based on Kohonen neurons. Their activation levels represent the Euclidean distance of the neuron's weight vector from the input vector. They output a signal only if they win the CBUS competition. Their behaviour is the same during learning and subsequent operational phases.

$$a(e_k) = ||\theta - \mu(e_k)|| \quad [4.3.2.2.3.9]$$

$$o(e_k) = \begin{cases} 1.0 & \text{if } e_k \text{ winner} \\ 0.0 & \text{if } e_k \text{ not winner} \end{cases} \quad [4.3.2.2.3.10]$$

During learning, the permanent constraint neuron is activated whenever there is resistance to movement in any of the joint motors. This behaviour is simply expressed as

$$a(p_1) = \begin{cases} 1.0 & \text{if resistance to movement detected} \\ 0.0 & \text{otherwise} \end{cases} \quad [4.3.2.2.3.11]$$

$$o(p_1) = a(p_1). \quad [4.3.2.2.3.12]$$

4.4. Operation of the AAC

The AAC operates in three logical phases: (1) an unsupervised **training phase** in which the AAC adapts to the specific sensory and kinematic constraints of the system, (2) an **arm movement phase** in which the TLC employs the AAC to reach for target points, and (3) an **on-going adaptation phase** in which the AAC autonomously adapts to unforeseen changes in the kinematics or sensor characteristics of the robot.

4.4.1. The Training Phase

During the training phase, M_a and M_e networks create topographic mappings of the arm and eye configuration spaces, and inter-map connections grow from M_o to M_a and from M_e to M_a .

Connections from M_o to M_a represent collision-avoidance constraints as described in section 4.1.9. These connections are derived from the volume of the workspace that is occupied by the arm in each training sample. We will simplify the problem of growing connections by assuming that neurons activated by the arm can be distinguished from neurons activated by the body and other obstacles. How can this be accomplished in a real implementation? There are many possibilities. For example, we could paint the arm a different colour from its surroundings, and make the neurons in M_o only sensitive to this colour during training. Or, we could identify the arm by the fact that it moves in the visual field. The human visual system has a special information channel reserved for information about moving objects [Kronauer & Zeevi, 1985; Livingstone, 1988]. Other simple solutions are also possible. We will also require that non-permanent obstacles

be removed from the workspace during the training phase so that the arm can occupy a representative set of workspace volumes.

Before training begins, all M_a and M_e input weight vectors μ are set to random values between 0 and 2π radians⁸. All inter-map connections ω are set to 0. The TLC then activates the RMG and EET which causes the end-effector to randomly wander through a representative set of workspace points, and the eyes to track the point-of-touch. Sensors are sampled at regular intervals⁹ to provide training data for the three maps. This data changes the activation patterns of the three maps by (1) activating M_0 neurons with receptive fields intersected by the arm, (2) activating the M_a neuron that best represents the arm configuration vector θ , and (3) activating the M_e neuron that best represents the eye configuration vector Ψ .

In order to prevent the weight changes that take place in the M_a and M_e networks from interfering with inter-map connection growth, a two step training cycle is used: (1) M_a and M_e are first allowed to distribute their weight vectors across the configuration spaces, and then (2) once M_a and M_e have stabilized connections are grown between maps. The first step is called the **self-organization step**, and the second step is called the **connection growth step**.

4.4.1.1. The Self-Organization Step

Kinematic constraints on arm movement and points-of-gaze are represented by the shapes of the distributions of θ and Ψ vectors respectively. During this step, self-organizing laws distribute M_a weight vectors, $\mu(a_j)$, over the region of the arm configuration space that corresponds to legal movements of the arm and M_e weight vectors, $\mu(e_j)$, are distributed over the region of eye configuration space that corresponds to eye angles that focus the eyes on points of gaze in the workspace of the robot. Furthermore, the M_a and M_e network topologies are preserved by this process as described in section 2.4.3. Equations 4.4.1.1.1 and 4.4.1.1.2 are the weight modification rules used. These equations are expressed in the discrete time format where subscript 't' represents the system time.

$$\mu(a_j)_{t+1} = \mu(a_j)_t + \lambda_t \cdot \beta(a_j)_t \cdot (\theta_t - \mu(a_j)_t) \quad [4.4.1.1.1]$$

$$\mu(e_k)_{t+1} = \mu(e_k)_t + \lambda_t \cdot \beta(e_k)_t \cdot (\Psi_t - \mu(e_k)_t) \quad [4.4.1.1.2]$$

λ_t is a function which exponentially decreases from 0.5 to 0.01 during the first 1000 iterations of the self-organizing step. $\beta(p)_t$ is a function with value 1.0 if neuron p falls within radius r of the

⁸ The AAC is not very sensitive to the initial values of the input vectors and any small random values will do; e.g. values between 0 and 1.

⁹ Random intervals work equally well.

winning (active) neuron and 0.0 if it does not. Radius r decreases from 0.5 times the minimum width of the network to 1 neuron during the first 1000 iterations.

4.4.1.2. The Connection Growth Step

Eye/hand coordination constraints (inverse kinematics) are represented in the training data by temporal associations between θ and Ψ vectors (see sections 4.1.5. and 4.3.1.). Obstacle avoidance constraints are represented by temporal associations between M_0 activation patterns and θ vectors (see sections 4.1.8. and 4.3.1.). Regions of permanent constraint within arm configuration space are represented by temporal associations between resistance to movement and θ vectors. During this step, connections must be grown between neurons that are temporally associated — ie. that are active at the same time in adjacent networks or the permanent constraint neuron. Recall that Hebb's law (section 2.2.3.) requires that connections be grown between cells that are simultaneously active. This is precisely the behaviour we need to create inter-map connections and permanent constraint connections.

The connection growth step can be further refined into an **inter-map association step** and a **permanent constraint determination step**.

4.4.1.2.1. The Inter-Map Association Step

How can these connections be found in a distributed manner? We require a weight modification law that each neuron can implement using only locally available information about the activity of neurons it receives connections from. Furthermore, this law must be hebbian, increasing the weight whenever a neuron receives an input signal over the connection at the same time that it is generating an output signal. As long as the robot's operational behaviour reinforces the association, the connection should continue to exist. However, as we saw in chapter 3, on-going changes caused by real-world damage may make a given associative connection obsolete. Our weight modification law should be capable of resetting the connection to zero — ie. forgetting the association — if it is no longer reinforced by the operational behaviour of the robot. By this, we mean that as the robot performs its tasks, each movement it makes can be employed as a new training sample, generating a temporal association. If the neurons associated by a given connection are never simultaneously active after the initial training phase, some aspect of the robot's geometry must have changed, and the connection should be allowed to decay to zero, effectively forgetting the obsolete association.

Equation 4.4.1.2.1.1 is an example of a weight modification rule with the desired properties. This equation grows connections between each M_0 neuron o_i and all M_a neurons a_j that correspond to arm configurations that intersect the receptive field of o_i . The learning term has two parts — (1) a Hebbian part, $\xi \cdot \alpha(o_i) \cdot \alpha(a_j)$, that causes the connection to rapidly increase if there is an input

signal at the same time that the neuron generates an output signal, and (2) a decay part, $-\chi \cdot \omega(o_i, a_j)$, that very slowly decreases the connection strength if the Hebbian part is zero. Coefficients ξ and χ control the rate of connection growth and decay respectively. As long as $\xi \gg \chi$ the learning term will be positive when both neurons are simultaneously outputting positive signals. The rest of the time the second term will be very small and negative, causing the connection to decay toward zero. The term 'e' is a signal that outputs the value 1.0 if the EET is tracking the end-effector, and 0.0 if it is not. Equation 4.4.1.2.1.2 is similar to 4.4.1.2.1.1 except that it is used to grow connections between M_e neurons e_k and all M_a neurons a_j that correspond to arm configurations that position the end-effector at the points-of-gaze represented by e_k . In this equation, the operator denoted by $[f(x)]^+$ is a thresholding function that outputs the value of $f(x)$ as long as it is positive, otherwise it outputs 0:

$$[f(x)]^+ = \begin{cases} f(x) & \text{if } f(x) > 0.0 \\ 0.0 & \text{otherwise} \end{cases}$$

$$\omega(o_i, a_j)_{t+1} = \omega(o_i, a_j)_t + (\xi \cdot \alpha(o_i)_t \cdot \alpha(a_j)_t - \chi \cdot \omega(o_i, a_j)_t) \cdot e_t \quad [4.4.1.2.1.1]$$

$$\omega(e_k, a_j)_{t+1} = \omega(e_k, a_j)_t + (\xi \cdot [\alpha(e_k)_t \cdot \alpha(a_j)_t]^+ - \chi \cdot \omega(e_k, a_j)_t) \quad [4.4.1.2.1.2]$$

A family of related equations has been extensively studied by [Grossberg, 1987a, 1987b].

These laws record causal associations between network patterns. As Figure 4.14 shows, at any given point in time, a set of connections are grown from M_o to M_a that record the volume of space occupied by a given arm configuration. Figure 4.15 shows the state of the network at some future time when connections have been grown that record the volume of a second arm configuration. Finally, as Figure 4.16 shows, after training is finished, if an obstacle is placed into the workspace, its activation pattern will inhibit neurons in M_a such as the one shaded gray that record arm configurations that intersect it. Other neurons, like the one outlined in bold, are not inhibited.

Similarly, these laws grow connections from M_e to M_a that record causal associations between patterns that represent eye configurations and points-of-touch, as they occur during training (Figures 4.17 and 4.18). After training these connections can be used as a record of the inverse kinematic mapping between points-of-gaze in the workspace and arm configurations that touch them.

Figure 4.14 M_o to M_a Connection Growth

Figure 4.15 Continued M_0 to M_a Connection Growth

Figure 4.16
 M_0 to M_a Connections Find Configurations
That Intersect Obstacles

Figure 4.17 M_e to M_a Connection Growth

Workspace

Ma

After training the final set of connections record the inverse kinematic transformation

Me

Figure 4.18 Continued M_e to M_a Connection Growth

4.4.1.2.2. The Permanent Constraint Determination Step

As described in sections 4.1.6. and 4.1.7., paths through M_a must not cross regions of configuration space corresponding to impossible arm movements. The simple associative mechanism described in section 4.4.1.2.1. above is used to inhibit neurons on the boundary of such regions. Boundary inhibition guarantees that the path finder will generate paths which circumvent these regions, rather than crossing them. During training the permanent constraint neuron is active when an arm joint reaches its limit of movement or a joint motor sensor detects resistance to movement (due to the body of the robot — see equation 4.3.2.2.3.11). Equation 4.4.1.2.2.1 describes inhibitory connection growth from p_1 to all M_a neurons that represent arm configurations bordering these regions. This equation is the same type as equations 4.4.1.2.1.1 and 4.4.1.2.1.2. After training, neuron p_1 is permanently activated to inhibit bordering configurations.

$$\omega(p_1, a_j)_{i+1} = \omega(p_1, a_j)_i + (\xi \cdot [\alpha(p_1)_i \cdot \alpha(a_j)_i])^+ - \chi \cdot \omega(p_1, a_j)_i \quad [4.4.1.2.2.1]$$

4.4.2. The Arm Movement Phase

Once training is complete, and the AAC has adapted itself to a robot's path planning constraints, the TLC may take control of the eyes and select target points that it wishes the arm to touch. Obstacles may enter the workspace now that training is finished. We will assume for the moment that weights are not updated after the training phase.

During the training phase, the M_o activation pattern only represented the workspace volume occupied by the arm. Now that training is complete, we want M_o to represent all objects *except* the arm. Only those M_o neurons activated by potential obstacles must inhibit M_a neurons that represent arm configurations that intersect them. Therefore, once training is complete, the same mechanism that was used to distinguish between the arm and other objects is again used, this time to activate neurons that have non-arm objects in their receptive fields.

The behaviour of the AAC during the arm movement phase proceeds in three steps.

4.4.2.1. The Target Selection Step

During this step the TLC selects a target point by adjusting the eye motor angles to foveate the target point. On command from the TLC the three maps update their activation levels from current sensory data as shown in Figure 4.19. The M_o activation pattern represents the volume of the workspace currently occupied by obstacles, M_a activates a neuron that represents the initial configuration of the arm (ie. the current arm configuration), and M_e activates a neuron that represents the selected point-of-gaze (ie. the current eye configuration). The M_o activation pattern is derived from environmental sensor data — e.g. a high contrast camera image in the case of a

two-dimensional workspace — and the M_a and M_e activation patterns are derived from CBUS competitions in constant time.

Figure 4.19 The Target Selection Step

4.4.2.2. The Constraint Determination Step

During the next time step, shown in Figure 4.20, active M_o neurons send signals along their outgoing connections to M_a . In parallel, these signals inhibit a set of M_a neurons corresponding to arm configurations that intersect obstacles in the workspace. Other neurons that correspond to impossible arm configurations are inhibited by signals from the permanent constraint neuron. These inhibited neurons must not be included in the arm configuration path. During the same time step the winning M_e neuron sends excitatory signals to a set of target configuration neurons in M_a . Some of these neurons, inhibited by M_o , correspond to target configurations that intersect obstacles in the workspace. The remaining uninhibited target configuration neurons represent possible final configurations of the arm which successfully place the end-effector near the desired target point. In this way, the system is able to plan a number of potential goal configurations which correctly position the end-effector in one time step.

4.4.2.3. The Collision-Free Movement Step

We now have all the information we need to select a collision-free arm control path. The initial configuration neuron represents the first end point of the arm control path. One of the uninhibited target configuration neurons is selected at random as the second end point of the arm control path. Inhibited neurons may not be included in the arm control path so the arm control problem is now reduced to finding a chain of neighbouring control path neurons that extends from the initial configuration neuron to the selected target neuron. The weight vectors associated with this chain of neurons (shown in Figure 4.22) is output to the XBUS as the final arm trajectory. Actual movement of the arm is smoothed by joint interpolation as described in section 4.2.1.5.

Figure 4.20 The Constraint Determination Step

Path finding is a well understood computational problem. The AAC can use any standard serial or parallel path finding algorithm to perform this step. We will not elaborate on existing path finding algorithms in this thesis. What about the weaker constraint of path shape mentioned in section 4.1.10? Which of the many possible paths between the initial configuration neuron and the selected target neuron is best path? This question depends on what the user regards as desirable characteristics of an end-effector path. A path that meanders about the workspace is probably less desirable than a direct path. Also, paths which minimize the total amount of mechanical wear on the joints are probably more desirable than other paths. Choosing the shortest arm control path through M_a is sensible approach because shortest length paths tend to correspond to relatively direct end-

effector paths, and they tend to minimize the combined amount of joint displacement. The algorithm for topographically connected systolic arrays described in [Miller & Stout, 1985] is capable of finding the shortest path in $O(\sqrt{N(M_a)})$ steps.

A chain of weight vectors from the M_a network correspond to points along a similar continuous and connected curve in arm control space because the space is represented topographically by M_a . Quantization error and the distortions of arm configuration space that result from the self-organization step require that an additional constraint be applied to the path finding process before M_a paths can be guaranteed collision-free. We will return to the path finding problem again in section 4.5.5. to examine this last constraint.

Figure 4.21 The Collision-Free Movement Step

4.4.3. On-Going Adaptation

Once the training phase has stopped, unplanned changes in the robot's kinematics and sensor calibrations may occur. How can we continue to adapt the system to these changes? We examine two simple strategies in this section.

The simplest approach is to allow poor performance of the AAC to trigger another complete training phase. For example, arm collisions would be detected by the joint motor strain gauges, and environmental sensors could be used to determine if the distance between the target configuration points-of-touch and the target point is greater than allowed. If either of these conditions occurs, it is reasonable to assume that some aspect of the eye-arm-body system must have changed. The robot would re-activate the RMG and EET controllers and reperform the self-

organization and inter-map association steps¹⁰. This approach may not work in some cases. For example, if the workspace contains large obstacles during the new training run that are removed after training is complete. In this case the robot will incorrectly learn that this region of space cannot be intersected by the arm. Of course, this situation would eventually trigger another training run which could fix the problem. This situation will not occur if the workspace obstacles change their position during training. As long as the arm can eventually assume a representative set of configurations, learning will not be affected. Note that in this scenario, the ω weight change laws described by equations 4.4.1.2.1.1, 4.4.1.2.1.2, and 4.4.1.2.2.1 are disabled (ξ and χ are set to 0.0) during the arm movement phase.

What if it is not possible for the robot to pause long enough to perform a complete training phase? What if the amount of time that large mobile obstacles remain motionless in the workspace is longer than the duration of a training run? Another strategy that accommodates these situations is to allow on-going training as the changes occur. Two requirements must be met to perform on-going training:

1. The EET must be allowed to track the end-effector as the robot arm moves through planned trajectories. After the TLC has selected a target point and an arm control path has been found, it must activate the EET and relinquish control of the eyes until the arm has finished its movement. During this time the system switches to training phase mode and the arm is sampled as it completes its trajectory. These movements are used as training samples to simultaneously drive the self-organization and inter-map association laws.
2. When it is not otherwise occupied, the TLC must occasionally 'exercise' the arm by selecting practice target points that are randomly distributed over the entire workspace¹¹. Nothing more than a random number generator is needed to select these values. Movements to random locations give the arm a chance to practice moving through regions of the workspace that it rarely visits. If the task domain is such that target points are naturally distributed over the entire workspace, this process is not needed.

The first requirement is a simple way to spread the training process through time. If no changes occur during the lifetime of the robot, then the distribution of M_a and M_e weights will not change,

¹⁰ It is not necessary to re-randomize the input vector weights, but the inter-map connection weights should be re-set to 0.0. Some informal results derived from the simulations described in section 4.6. suggest that self-organization time is significantly lessened by starting with a distribution that is close to, but not exactly, the final training distribution, and using a bubble size $\beta(u_k)$ of 2 that decreases to 1 over the next 1,000 iterations.

¹¹ This is an extremely simple requirement — random number generator is sufficient to find these locations. Alternatively, the RMG can be switched on between arm movements to let the arm wander about before the TLC selects another target point.

and the existing inter-map connections will continue to be reinforced. If a change does occur however, an altered distribution of training samples will be produced by practice movements and by attempts to touch TLC selected target points. Kinematic-sensory changes will be reflected by changing M_a and M_e weight distributions, and by changes in inter-map connections as some connections decay because they are no longer reinforced. Of course, the performance of the system will be affected by these on-going weight modifications until enough training samples have been generated to allow the AAC to 'catch up' to the kinematic changes in the robot. However, if the changes are small or gradual, the performance of the system will not be significantly impacted — most inter-spacial associations will still be correct. If the changes are sudden and dramatic, there will be a more pronounced degradation in AAC performance until re-adaptation is complete. The following simple strategies help reduce re-adaptation time. Either strategy can be triggered when the system detects a serious error such as a collision or a large distance between the target point and point-of-touch after movement. First, the radius function, $\beta(u_i)$, can be increased to a value of 2 and the λ function can be allowed to decay from 0.5 to 0.01 for 500 iterations of the self-organization algorithm. This promotes faster self-organization. The decay coefficient χ can also be increased to forget incorrect associations faster¹². Secondly, if serious errors persist, the system can simply trigger a complete training phase as described above. This last strategy would be employed only as a last-resort.

The designer may wish to try to balance the distribution of random arm movements produced by the second strategy with movements triggered by the TLC so as to produce a uniform distribution of on-going training samples. However, it may be to his advantage to allow an imbalance to exist. A greater density of weight vectors in regions of the configuration spaces associated with TLC triggered movements will tend to increase the AAC's accuracy in the regions of the workspace it must frequently work in. The result will be an increase in accuracy where it is most needed (for the given task domain) at the expense of less precision in rarely visited regions. We will see exactly why this should be so in section 4.5. The more uniform the distribution of targets in the workspace, the less pronounced this affect will be.

4.4.4. Extending the AAC to Realistic Robots

Practical implementation considerations aside, the AAC is not limited to controlling 2-dimensional, 2 degree-of-freedom; revolute jointed Horn robots. It can be extended in two ways: (1) from 2-deimensional to 3-dimensional robots, and (2) by adding additional degrees-of-freedom to the arm being controlled.

¹² Selecting an appropriate balance between ξ and χ is a subject for further research, however, some of the work by Grossberg [Grossberg, 1987a] provides a good starting point.

A three dimensional robot requires an additional degree-of-freedom to specify points in its workspace. This is easily accomplished by mounting the camera eyes on a pallet that rotates around an axis parallel to the line intersecting the center of both eyes. There are now three degrees-of-freedom to uniquely identify each target point in the three dimensional workspace. A new IO-BUS wire must be provided to supply M_e processors with this new eye angle, and to provide each processor with an additional weight vector component.

M_0 must also be extended to represent cubic volumes of the workspace, rather than simple 2-dimensional areas. The reader may wonder how we can quickly determine the pattern of activation in M_0 . We can no longer simply hang a camera over the workspace because obstacles will hide regions of the workspace from view. A simple approach to the problem is to use the disparity between the two stereo camera images to generate a depth map of what the robot sees. This depth map can be generated in parallel by the simple neural network system proposed by [Marr & Poggio, 1976]. In this system, the activation levels of an array of neurons indicate the depth of objects from the camera. It would be very simple to use this activation level information to trigger the appropriate M_0 level neuron to represent an object at a certain depth in the scene¹³. A variety of neural networks have already been proposed which have properties ideal for implementing M_0 . The reader is referred to [Frisby, 1980; Gmitro & Gindi, 1987]. In this way, in constant time we can fill in a pattern of M_0 activations that corresponds to the surfaces of obstacles that are visible to the robot.

This simple approach is obviously not sufficient to guarantee collision-free movement of the arm. The path planner may generate a trajectory in which the arm attempts to reach behind the visible surface of a large obstacle and encounters its hidden side. If this happens, resistance to movement will be detected. This resistance can be used to trigger the temporary inhibition of the corresponding — ie. current — M_a neuron that represents the arm configuration that collides with the obstacle. Repeated encounters with the obstacle will gradually inhibit the set of M_a neurons that should not be included in arm control paths, and collision-free movement will eventually be possible. Once the obstacles move however, the M_a neurons inhibited by this process will have to be reset, and the process can begin again. This mechanism may be sufficient for many domains in which the obstacle set does not dynamically change. Other, more sophisticated systems that generate a representation of the hidden volume of objects from visual data can be incorporated into the AAC for more sophisticated domains. The subject of building and maintaining mental images of objects

¹³ For example, the M_0 neurons could be made selectively sensitive to the amount of incoming signal from the depth map neurons. If the activation level of the depth map neuron represents an object inside the range of a M_0 neuron it would fire.

from visual data is a complex topic in its own right, and we will not pursue it further here. Suffice it to say that the AAC does not impose limitations on this process, and the success of the AAC is obviously dependent on the ability of the robot to collect sufficient information regarding the positions of obstacles in the workspace, as expected.

In order to add a degree of freedom to the robot arm, we must modify the M_a network. An IO-Bus wire must be added for each new joint, and the M_a CBUS algorithm must be changed so that the bubble of processors that surround the winning neuron is a cube, not a square. In other words, we must imagine that the M_a network has been extended from an array of neurons to a cubic lattice of neurons. The M_a neurons are not actually interconnected, so this extension does not require a physical change in the network. The neighbourhood relationships between neurons are simply a function of the CBUS competition algorithm and ID numbering plan described in section 4.3.2.2.2. By ensuring that the dimensionality of the M_a network matches the dimensionality of arm configuration space, we make it possible to topographically order the network during learning. This is essential to the path finding process. It was not necessary to extend the M_e network in this way because M_e does not have to be topographically ordered — all that matters is the distribution of the weight vectors in eye configuration space. As we saw in section 2.4.3., the Kohonen algorithm will continue to distribute weights correctly even if the dimensionality of the network space is lower than that of the input space.

Note that in order to ensure the same approximate distance between weight vectors in the configuration spaces, we must increase the number of neurons whenever the system is extended. Unfortunately, increasing the degrees-of-freedom of the M_a and M_e networks causes an exponential explosion in the number of neurons to ensure constant coverage of the volume of the space. For example, if the width of the M_a network is N , and the number of degrees-of-freedom of the arm is D , then we would require N^D neurons. This is not a problem for the M_e network, because no more than 3 degrees-of-freedom are ever needed to represent points in a 3-dimensional workspace. However, as we shall see, it places a purely practical limitation on the AAC to the control of 2 and 3 degree-of-freedom arms. We will speak more of this later.

4.5. A Geometric Analysis of the AAC

Now that the architecture and basic operation of the AAC have been outlined, a number of important questions remain. For example, can we predict how many connections will be grown between layers? How can we best visualize the pattern of this connection growth? Can we estimate how long the system will take to learn? How accurate can we expect movement to be? Can we gain some insight into sources of imprecision? Will any path through arm configuration space be collision free? If not, what constraints must we impose on the path finding process? What is the

effect on the system of changing the number of network neurons? How do we know how many neurons we need for a particular application?

We must answer all of these questions — at least qualitatively — in order to judge the effectiveness of the AAC as a robot control system. This section employs a powerful conceptual tool called the Voronoi tessellation to help us understand the behaviour of the AAC in geometric terms. We will use this geometric viewpoint as the theoretical basis upon which answers can be found.

4.5.1. Voronoi Tessellations

Given two points v_i and v_j on the plane, which we shall refer to as **defining points**, the set of points which are closer to v_i than to v_j is the half-plane which contains v_i and is bordered by the perpendicular bisector of the line segment $\overline{v_i v_j}$ (Figure 4.22a). If we refer to this containing half-plane as $H(v_i, v_j)$ then, for an arbitrary set S of N defining points on the plane, the locus of points which are closer to v_i than to any other defining point in S is the intersection of $N-1$ half-planes

$$VR_i = \bigcap_{i \neq j} H(v_i, v_j). \quad [4.5.1.1]$$

The resulting locus is a convex polygonal region which has no more than $N-1$ sides (Figure 4.22b). We shall refer to this polygonal region as the **Voronoi region** VR_i for defining point v_i . Equivalently, each point p_j in VR_i satisfies the property that the Euclidean distance between p_j and v_i is less than or equal to the distance between p_j and all other points $k \neq i$ in S

$$VR_i = \{p_j \mid \forall k \neq i, \|p_j - v_i\| \leq \|p_j - v_k\|\}. \quad [4.5.1.2]$$

The N Voronoi regions associated with the defining points in S partition the plane into a convex network which is referred to as a **Voronoi tessellation**¹⁴ (Figure 4.22c). The vertices of this tessellation are called **Voronoi vertices**, and the line segments connecting these vertices (which comprise the edges of all the polygonal Voronoi regions) are called **Voronoi edges**.

Each point in S is associated with a unique Voronoi region, therefore, if a point $(x, y) \in VR_i$, then v_i is the nearest neighbour of (x, y) . Points of the plane which reside on a Voronoi edge bordering two Voronoi regions VR_i and VR_j are points that are equidistant from the associated defining points v_i and v_j of S . In the case of a 2-dimensional plane, every Voronoi vertex is the common intersection point of exactly three Voronoi edges. Therefore, each Voronoi vertex is equidistant from three vertices of S .

¹⁴ The term tessellation refers to a mosaic covering of a two dimensional surface. Here, the term refers to the network of vertices and connecting lines that outline an imaginary mosaic superimposed on the plane. Each tile in the mosaic is a Voronoi region.

Figure 4.22 Voronoi Tessellations

Voronoi tessellations have proven to be a very useful tool in a number of fields because, in a powerful sense, they contain all of the proximity information defined by the set of points S . In addition to being employed as powerful theoretical tools in computational geometry [Preparata & Shamos, 1985], Voronoi tessellations are also commonly used in disciplines as diverse as archaeology [Hodder-Orton, 1976], ecology [Pielou, 1977] and molecular physics [Preparata and Shamos, 1985]. We shall use the concept of Voronoi tessellation to illuminate the process of inter-map connection growth.

4.5.2. Voronoi Tessellations and the AAC

The three network layers of the AAC each produce Voronoi tessellations of their respective input spaces. These tessellations and the relationships between them are useful metaphors for AAC behaviour during the learning and operational phases. We shall now examine the source of the tessellations and use them to illustrate the poorly defined character of inter-map connection growth.

4.5.2.1. Kohonen Competitions Define M_a and M_e Tessellations

As we have seen in section 2.4.3., the activation level of each neuron u_i in a Kohonen network is proportional to the Euclidean distance between an input signal vector x_j and the vector w_i of connection weights associated with the neuron's input channels. The neurons in a Kohonen network compete to 'represent' the input signal vector x_j . The winner of the competition is the neuron u_c with the smallest activation value a_c after presentation of x_j . This neuron is therefore associated with the weight vector w_c closest to x_j in the input space.

Similarly, it should be clear that u_c will also win the competition to represent *any* input signal vector x_j which is closer to w_c than all other weight vectors w_k , $\forall k \neq c$. We can therefore represent the set of input vectors which will cause u_c to win the competition by

$$VR_c = \{x_j \mid \forall k \neq c, \|x_j - w_c\| \leq \|x_j - w_k\|\}. \quad [4.5.2.1]$$

Comparison of equations 4.5.2.1 and 4.5.1.2 reveals that this set of input vectors is equivalent to the set of points in a Voronoi region. The weight vector w_c is the defining point for this Voronoi region. Each neuron u_i in a Kohonen map then, is associated with a unique Voronoi region VR_i in its input space. These regions contain input vectors which are nearest neighbours to their defining weight vectors.

The complete set of Voronoi regions for a Kohonen network therefore defines a Voronoi tessellation of the network's associated input space (Figure 4.23). The defining points of this tessellation are the network weight vectors. We shall use $VT(M_a)$ to refer to the Voronoi tessellation of arm configuration space by the M_a network. $VR(a_i)$ will refer to the Voronoi region of $VT(M_a)$ associated with M_a neuron u_i . Similarly, $VT(M_e)$ and $VR(e_i)$ will refer to the Voronoi tessellation and Voronoi regions for the eye configuration map M_e .

Figure 4.23 Kohonen Competition Generates Voronoi Tessellations

4.5.2.1.1. Weight Distributions Determine Tessellation Shapes

The precise shape of these tessellations depends entirely on the final distribution of weight vectors in input space. This distribution is in turn dependent on the initial random locations of the weight vectors prior to learning and the detailed presentation order of sample input vectors during training. We cannot, therefore, easily predict the precise details of the final shape. However, knowing something about the expected distribution of weight vectors in the input spaces allows us to characterize the shapes of the tessellations in qualitative terms.

Figure 4.24 Non-Uniform M_a Distribution

4.5.2.1.1.1 The Shape of $VT(M_a)$

As described in Section 4.4.1., training data is sampled as the end-effector moves through a uniform random distribution of points in the Cartesian workspace of the robot. Although this workspace distribution is uniform, the distribution of points in arm configuration space associated with these training samples will not be. To understand why this is so, consider Figure 4.24 which represents the robot workspace (on the left) with a uniform distribution of points touched by the end-effector. The outermost circle represents the path of the end-effector as we hold the value of θ_2 constant at 0° and vary θ_1 from 0 to 2π . The arm will assume these configurations to touch sample points which lie on or near this circle. The inner circle represents the path of the end-effector as we hold the value of θ_2 constant at π and vary θ_1 from 0 to 2π . These will be the arm configurations which touch sample points on or near the small circle. The circumference of the outer circle is greater than the inner circle and more sample points fall on or near it. Therefore, there will be a greater density of training configurations on the line $\theta_2=0$ than on the line $\theta_2=\pi$ in arm

configuration space. The Kohonen algorithm will devote more neuron weights to the regions of higher density as shown to the right.

4.5.2.1.1.2. The Shape of $VT(M_e)$

The distribution of weight vectors in eye configuration space is highly non-uniform. Not all eye configurations cause the lines-of-sight to intersect at a point-of-gaze. Many configurations orient the eyes in widely divergent directions and cannot be used to select a target point. The shaded region in the right half of Figure 4.25 represents the region of eye configuration space corresponding to correct, foveating configurations. A configuration exists for each point on the workspace plane, since each line-of-sight can be aligned with any point by rotating the eye. During training however, the eyes track the end-effector which, by definition, cannot leave the workspace of the robot. Therefore the greatest — in fact, only — density of training samples will occur within a subset of the shaded region. The shape of this region is determined by the shape of the workspace which varies from robot to robot. A typical workspace shape is shown on the left of the Figure. The last observation to be made is that the diagonal line shared by the two shaded triangular regions represents configurations that target a point at infinity (the lines-of-sight are parallel). As we move away from this line within these regions, the target points move closer to the robot. Target points are infinitely compressed as we move toward $\Psi_1 = \Psi_2$, so to speak. We can expect the density of weights to be greater near this line and rapidly lessen as we move away from it.

Figure 4.25
The Foveating Configuration Regions of Eye Configuration Space

4.5.2.1.2. $VT(M_a)$ and $VT(M_e)$ Tessellate Toroids

Finally, we must take into account the toroidal nature of the arm configuration and eye configuration input spaces. Normally, Voronoi tessellations of the plane will include regions which are infinite and unbounded. As the arm and eye configuration spaces are closed, however, all of the Voronoi regions generated by the M_a and M_e networks will be finite and bounded by Voronoi edges. Figure 4.26 illustrates this. The configuration space is shown in Figure 4.26a. Notice that the Voronoi regions wrap around the space and are all closed polygons because the space is toroidal as shown in Figure 4.26b.

Figure 4.26 $VT(M_a)$ and $VT(M_e)$ Have Toroidal Tessellations

4.5.2.2. Observation Space is Also Tessellated

Although the observation map M_o is not a Kohonen network, it is convenient, for uniformity of exposition, to imagine that it defines a Voronoi tessellation of the observation space.

Let us consider the receptive field of each neuron in the observation map M_o to be a Voronoi region $VR(o_i)$. Together, these regions constitute a Voronoi tessellation $VT(M_o)$ of observation space. Strictly speaking, the receptive fields of neurons in M_o may overlap as long as the entire space is covered. Indeed, two or more neurons may share the same receptive field, in effect exactly overlapping the same region of space. We will make the following simplifying assumptions, without loss of generality: (1) the receptive fields cover the workspace region of observation space, (2) they do not overlap, (3) there is one and only one neuron associated with

each receptive field, and (4) each receptive field is associated with a unique neuron. These assumptions are only introduced to simplify the terminology used in the following sections; they allow us to refer to receptive fields of M_0 neurons as Voronoi regions. They are not absolutely necessary for the analysis, however, as each Voronoi region in $VT(M_0)$ will be examined in isolation from the others. Given these simplifying assumptions, it is easiest to think of each $VR(o_i)$ as a square cell within a rectilinear grid overlapping the workspace¹⁵ (Figure 4.27). In this case, the defining points of $VT(M_0)$ are imaginary points at the exact center of each grid cell.

Figure 4.27 The $VT(M_0)$ Tessellation is a Grid

4.5.3. Mapping Voronoi Tessellations Into the Workspace

It is possible to gain a much deeper understanding of the inter-map associative laws described in section 4.4.1.2.1., by relating them to the three Voronoi tessellations produced by the M_0 , M_a and M_e networks.

To this end we will now examine how each of these tessellations can be mapped into the Cartesian workspace of the robot. Recall that each input vector contained in a Voronoi region $VR(u_i)$ will cause its associated neuron u_i to become active. Mapping $VR(u_i)$ into the Cartesian workspace will allow us to relate events that occur in the workspace to the activation of neuron u_i .

¹⁵ As shown in Figures XX and XX, the bordering regions of a Voronoi tessellation extend to infinity. We shall ignore the bordering regions surrounding our grid without loss of generality. Such borders can be added without changing the basic results of this section.

3

Micro-D

4.5.3.1. The $VT(M_0)$ to Workspace Mapping

The observation map M_0 is a quantized representation of the Cartesian workspace of the robot which consists of the area within reach of the robot's end-effector. It is therefore trivial to perform this mapping by superimposing $VT(M_0)$ onto the workspace without distortion, as shown in Figure 4.27.

4.5.3.2. The $VT(M_a)$ to Workspace Mapping

In order to map the arm configuration space tessellation onto the Cartesian workspace, we will transform each point in arm configuration space onto the workspace point touched by the end-effector when in that configuration. This reference point is the point-of-gaze tracked by the visual sensors during the inter-map association learning stage. This mapping is simply the forward kinematics of position for the robot in question. As explained in section 3.3.1.1., this mapping will be many-to-one in the general case. For example, our Horn robot would map to the workspace points (x,y) as follows:

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2), \text{ and} \quad [4.5.3.2.1]$$

$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2). \quad [4.5.3.2.2]$$

Figure 4.28 Aspects of the Kinematics of the Robot

Clearly, if $\theta_2 = 0$ (all configurations in which the arm is fully extended) equations 4.5.3.2.1 and 4.5.3.2.2 will map to unique points at the limit of reach (Figure 4.28a). If $L_1 = L_2$, and $\theta_2 = \pi$, the equations reduce to

$$x = L_1 (\cos\theta_1 - \cos\theta_1) = 0 \quad \text{and} \quad [4.5.3.2.3]$$

$$y = L_1 (\sin\theta_1 - \sin\theta_1) = 0, \quad [4.5.3.2.4]$$

and all configurations map to the origin (Figure 4.28b). Otherwise, pairs of arm configuration points — corresponding to elbow-up and elbow-down configurations — will typically map onto the same workspace point (Figure 4.28c).

This many-to-one mapping tells us that our Voronoi regions will overlap one another once they have been mapped into the Cartesian workspace. In addition, we can expect Voronoi regions which share a Voronoi edge to continue to share the same Voronoi edge once they have been mapped into the workspace, because neighbouring points in the arm configuration space must map to neighbouring points in the workspace.

Figure 4.29 The $VT(M_a)$ to Workspace Mapping.

To aid the intuition, we may think of the tessellation $VT(M_a)$ as a rectangular rubber sheet, and the mapping process as the act of stretching and folding the $VT(M_a)$ sheet to make it fit the contours of the Cartesian workspace. Imagine lifting the doughnut shaped $VT(M_a)$ out of arm configuration space and placing it into the Cartesian workspace such that points in the tessellation which position the end-effector at a point in the workspace are aligned with that workspace point as shown in Figure 4.29. Figure 4.30 shows how the approximately square Voronoi regions in the arm configuration space map to curved parallelograms in the workspace. Regions which have θ_2

components between 0 and π will overlap regions with θ_2 components between π and 2π radians once they have been mapped into W_s .

Figure 4.30 Characteristic Shape of Mapped $VT(M_a)$ Regions

4.5.3.3. The $VT(M_e)$ to W_s Workspace Mapping

A similar approach must be taken to map the eye configuration space tessellation onto the Cartesian workspace. In this case we transform each point in eye configuration space into the point-of-gaze associated with the configuration. For example, the 2-degree-of-freedom eye configurations for our Horn robot would map to the workspace points (x,y) as follows:

$$x = R \cos \Psi_1 - \frac{W}{2}, \quad [4.5.3.3.1]$$

$$y = R \sqrt{\sin \Psi_1}, \quad [4.5.3.3.2]$$

$$R = \frac{W \sin \Psi_2}{\sin(\Psi_2 - \Psi_1)}. \quad [4.5.3.3.3]$$

Because the line-of-sight of each eye can be aligned with any point on the workspace plane, every workspace point can be mapped onto an eye configuration (Figure 4.31a). It is *not* true, however, that every eye configuration point can be mapped onto a workspace point.

There are two types of non-mappable points. First, we note that equation 4.5.3.3.3 is undefined if $\Psi_1 = n 2\pi \Psi_2$, for all integer values of n . The lines-of-sight of these configurations are parallel and do not intersect at a point-of-gaze. All such configurations fall on the bold diagonal line in Figure 4.31b.

Secondly, x and y are undefined if the Euclidean distance R in 4.5.3.3 is negative. This will occur for configurations in which $(0^\circ < \Psi_2 \leq \pi)$ and $(\Psi_1 > \Psi_2)$, or $(\pi < \Psi_2 \leq 2\pi)$ and $(\Psi_1 < \Psi_2)$. The lines-of-sight of these configurations are neither parallel nor do they intersect at a point-of-gaze. Such configurations fall outside of the shaded region in Figure 4.31c.

For completeness, we shall map all non-mappable configurations to a point at infinity in the workspace plane.

Figure 4.31 Aspects of the $VT(M_e)$ to Workspace Mapping

Because the $VT(M_e)$ to workspace mapping is not many-to-one for points of gaze in the reachable workspace of the arm, the tessellation will not fold over on itself; only stretching distortions will occur. Once again let us imagine lifting the tessellation and transferring it to the workspace plane. This time however, we must stretch the bottom triangular region in Figure 4.25 to cover the infinite half plane above the line $y=0$. The top triangular region in Figure 4.25 is stretched to cover the half plane below this line. Figure 4.32 illustrates this process.

Figure 4.32 The VT(M_e) to Workspace Mapping

4.5.4. A Geometric Interpretation of the Inter-Map Associative Laws

Having characterized the Voronoi tessellations produced by the M_o , M_a and M_e networks, and their mappings into the workspace, we can now describe the inter-map associative laws in geometric terms. This will help us visualize the process of inter-map connection growth, provide insight into sources of system imprecision and estimate the number of neurons required to achieve a particular degree of accuracy.

4.5.4.1. Associations Between Workspace Events and Processors Activations

Our mapped Voronoi tessellations are powerful representations of the associations between events that occur at a given point p_i in the workspace and the set of neurons which are activated by these events.

Figure 4.33 $VT(M_o)$ Represents Obstacle Locations

Figure 4.34 $VT(M_a)$ Represents Configurations that Touch a Point

For example, suppose an obstacle appears in the robot's workspace. If a point p_i on the surface of the obstacle is an element of the mapped region $VR(o_j)$, then neuron o_j will be activated in map M_o . Therefore, the mapped $VT(M_o)$ captures information about the set of neurons that are activated by an arbitrary obstacle. Each mapped region of $VT(M_o)$ that intersects the obstacle by at least one point, corresponds to a neuron that will be activated by the obstacle (Figure 4.33). The same point p_i may fall into more than one mapped region of $VT(M_a)$ — due to folding of the tessellation

during mapping¹⁶. In this case, the mapped tessellation captures associations between arm configurations that touch p_i and the set of neurons that are activated by these configurations¹⁷ (Figure 4.34). Similarly, $VT(M_e)$ captures associations between eye configurations that gaze at p_i and the neurons that they activate (Figure 4.35).

Figure 4.35

$VT(M_e)$ Represents Configurations that Foveate a Point

4.5.4.2. M_a to M_e Connection Growth

All the information needed to predict the optimal growth of connections between the arm configuration map M_a and the eye configuration map M_e , is expressed by intersections that occur between regions in the corresponding mapped tessellations. Under ideal learning conditions, neuron a_i will grow a connection to every neuron e_j , such that, $\forall j$ $VR(e_j)$ intersects $VR(a_i)$ by at least one point.

To understand why this is so, recall that during the learning phase, at any moment in time the eyes are tracking a point-of-gaze p_g , and the end-effector is touching a workspace point p_t . Point p_g will be included in a mapped region $VR(e_g)$ thereby activating M_e neuron e_g . Similarly, point p_t will be included in a mapped region $VR(a_t)$ thereby activating M_a neuron a_t . The inter-map association laws will grow a connection between e_g and a_t because both will be active at the same

¹⁶ In the case of our 2-DOF arm, there will be at most two such regions.

¹⁷ The same region $VR(a_i)$ may contain the point p_i twice. This will occur if the region is folded over onto itself during mapping. This just means that two arm configurations that touch the same point are both closer to the weight vector of processor a_i than any other weight vector.

time. Now, recall that during training the EET ensures that the point-of-gaze coincides with the point touched by the end-effector. Therefore, p_g and p_t will actually correspond to the same workspace point p_c , which we shall call the **point of coincidence**. Since regions $VR(e_g)$ and $VR(a_t)$ both include point p_c , we must conclude that $VR(e_g)$ and $VR(a_t)$ intersect at this point. Ideally, during training the robot arm will eventually assume every possible configuration and thereby touch every reachable workspace point; ie. every point of coincidence p_c . Therefore, connections will be grown between *all* neurons with Voronoi regions that intersect at least one point in the reachable workspace of the robot. See Figure 4.36.

During training, a point of coincidence p_c causes the same M_e neuron to simultaneously activate with different M_a neurons — connections are grown between the M_e and M_a neurons. The M_e and M_a neurons all have Voronoi regions which, when mapped into the workspace, include point p_c . It follows that connections are grown between all M_a and M_e neurons that have intersecting Voronoi fields (see text).

Figure 4.36
Connections Grow When Regions of
Mapped VT(M_a) and Mapped VT(M_e) Intersect

4.5.4.3. How Fast Can We Learn Eye-Hand Coordination ?

We now see that it is not necessary to provide a training sample for *every point* in the workspace, rather, it is only necessary to provide one training point for *every region of intersection*.

This observation allows us to relate the speed of learning to the number of neurons in the M_a and M_e networks. It should be clear that the number of training examples required to achieve optimal connection growth must increase linearly with the number of regions of intersection. Because the number of Voronoi regions in a tessellation is determined by the number of neurons in

the associated network, we can relate the number of regions of intersection to the size of the networks.

In the worst-case, the highly pathological tessellations depicted in Figure 4.37 will lead to $O(N(M_a) \cdot N(M_e))$ areas of intersection. Voronoi tessellations that result from such a linear arrangement of defining points are however, an extremely poor approximation of the actual tessellations that would occur for any conceivable workspace. Given a uniform density of training points over the workspace, defining points (weight vectors) mapped from the arm or eye configuration spaces tend to be uniformly distributed, and therefore, elongated Voronoi regions are unlikely to occur. There will certainly be some variation in the amount of elongation of regions, but most will conform to roughly circular convex polygonal shapes. Therefore, a much better approximation of the tessellations is given by a rectilinear grid. We can maximize the number of regions of intersection by making some additional reasonable assumptions:

1. The average area of Voronoi regions in $VT(M_a)$ and $VT(M_e)$ is equal. As Figure 4.38 illustrates, more regions of intersection occur if the grid regions are approximately the same size.
2. The number of neurons in M_a is equal to the number of neurons in M_e . The maximum number of regions of intersection is determined by the size of the smallest network; $\min(N(M_a), N(M_e))$.

Perpendicular Alignment of Mapped M_a and M_e Weight Vectors Yields $O(P(M_a) * P(M_e))$ Regions of Intersection

Figure 4.37
Worst Case Number of
Mapped $VT(M_a)$ and Mapped $VT(M_e)$ Region Intersections

Given these assumptions, the maximum number of regions of intersection RI is given by

$$RI = 2^{D_w} \cdot \min(N(M_a), N(M_e)),$$

[4.5.4.3.1]

where D_w is the number of dimensions of the workspace ($D_w=2$ for our simple Horn robot).

64 Regions of
Intersection

36 Regions of
Intersection

25 Regions of
Intersection

Figure 4.38 Grids of Equal Size Have More Intersections

The probability P_L that learning is optimal — ie., a connection is grown between all neurons with intersecting regions — is equivalent to the probability that a training sample is selected for every region of intersection, such that the sample's point of coincidence p_c falls somewhere within the region of intersection. Given our simplifying assumptions, this problem is equivalent to the combinatoral problem of distributing r identical balls (the training samples) into n distinct cells (the regions of intersection) such that each cell contains at least one ball. There are

$$\binom{n-1+r}{r}$$

[4.5.4.3.1]

ways to distribute r balls into n cells with any number of balls per cell. Next we distribute n balls into the cells in order to ensure that there is at least one ball per cell. As the balls are identical, there is only one way to do this. The number of ways of distributing the remaining $(r-n)$ balls into the n cells is

$$\binom{n-1+(r-n)}{(r-n)}$$

[4.5.4.3.2]

The probability that there is at least one ball per cell is the ratio between the number of combinations with one ball per cell and the total number of combinations with any number of balls per cell. Simplifying 4.5.4.3.2 and dividing by 4.5.4.3.1, we have

$$\frac{\binom{r-1}{r-n}}{\binom{n-1+r}{r}}$$

[4.5.4.3.3]

Therefore, an estimation of the probability of optimal learning is given by

$$P_L \equiv \frac{\binom{TS-1}{TS-RI}}{\binom{RI-1+TS}{TS}}, \quad [4.5.4.2.4]$$

where TS is the number of training samples provided.

Voronoi Region for Ma Neuron a_i

An obstacle that intersects a receptive field inside the receptive field set of a_i ($PRF(a_i)$) will inhibit Ma neuron a_i . Notice that arm configurations in $VR(a_i)$ may not intersect the obstacle. This is the source of avoidance error.

Figure 4.39 How Obstacles Inhibit Arm Configuration Neurons

4.5.4.4. M_a to M_o Connection Growth

Kohonen competition ensures that one and only one neuron will be active in the M_a and M_o networks at each instant in time. This is not true of the M_o network. Workspace obstacles are represented in M_o by patterns of multiple neuron activations. Let us examine these activation patterns in the context of learning collision avoidance constraints.

During training, the arm acts as an obstacle that intersects M_o receptive fields and thereby activates M_o neurons. Simultaneously, an M_a neuron corresponding to the arm configuration is activated and connections are grown between M_o and M_a neurons.

Figure 4.40
 M_o to M_a Connections Capture the Avoidance Set
of each Arm Configuration Neuron

Under ideal learning conditions the arm will assume every possible configuration and touch every possible workspace point. After learning, each neuron a_i will have grown a connection to an associated set $A(a_i)$ of M_o neurons. We will call this set the **avoidance set** of a_i .

To understand how $A(a_i)$ is formed, consider the Voronoi region for a_i , $VR(a_i)$. See Figure 4.39. Each point $p_k \in VR(a_i)$ corresponds to a unique arm configuration that occupies an area of the workspace, $W(p_k)$. The light gray region in Figure 4.39 represents the range of arm configurations associated with this region. Since $VT(M_o)$ covers the entire workspace, $W(p_k)$ must intersect at least one Voronoi region of $VT(M_o)$. Because these regions represent the receptive fields of neurons in M_o , we will refer to their union as the **receptive field set of the configuration point p_k** ,

$$CRF(p_k) = \bigcup_j \{VR(o_j) \mid W(p_k) \cap VR(o_j) \neq \Phi\}, \quad [4.5.4.4.1]$$

where Φ = the empty set. The corresponding receptive field set of the M_a neuron a_i is the union of all the receptive field sets of configurations included in the Voronoi region $VR(a_i)$,

$$PRF(a_i) = \bigcup_{p_k \in VR(a_i)} CRF(p_k). \quad [4.5.4.4.2]$$

These receptive fields are contained within the dark border (and are hatched). Finally, the avoidance set of a_i is the set of M_o neurons associated with the regions contained by the receptive field set of this neuron,

$$A(a_i) = \{o_j \mid VR(o_j) \in PRF(a_i)\}. \quad [4.5.4.4.3]$$

Connections grown between M_a and M_o neurons represent the avoidance set (see Figure 4.40). When obstacles are introduced into the workspace after learning, any obstacle which intersects a receptive field $VR(o_j) \in PRF(a_i)$ will activate M_o neuron o_j and consequently send an inhibitory signal to M_a neuron a_i , thereby excluding the configurations in $VR(a_i)$ from the plan path. For any given obstacle O_b , the set $I(O_b)$ of M_a neurons it will inhibit is given by,

$$I(O_b) = \{a_i \mid O_b \cap VR(o_j) \neq \Phi, VR(o_j) \in PRF(a_i)\}. \quad [4.5.4.4.4]$$

Finally, let us define the **neuron shadow** of M_a neuron a_i to be the union of points which are contained by receptive fields in $PRF(a_i)$,

$$SH(a_i) = \{p_k \mid p_k \in VR(o_j) \wedge VR(o_j) \in PRF(a_i)\}, \quad [4.5.4.4.5]$$

The contents of the avoidance set depend on the precise details of the shape of $VT(M_a)$ and the physical robot arm. Nonetheless, we can use our understanding of this set to derive several properties which are important when learning obstacle constraints.

4.5.4.5. How Fast Can We Learn Obstacle Avoidance Constraints?

The preceding discussion assumed that learning conditions are ideal. There is a danger that the avoidance set will be incomplete under less ideal conditions. Suppose that we have a Voronoi region $VR(a_i)$ as shown in Figure 4.41. The shadow $SH(a_i)$ — shown in black — will occur if training points are sparse and only training points at p_a and p_b fall within $VR(a_i)$. The shadow $SH(a_i)$ that would occur under ideal learning conditions has been super-imposed onto Figure 4.41 in gray. If an obstacle intersects any gray region without intersecting the black region, neuron a_i will remain active even though there is a danger that a point p_c on a path through $VR(a_i)$ corresponds to an arm configuration C that intersects the obstacle.

We now see how optimal learning of collision constraints depends on the density of training points that occur in arm configuration space. The training sample distribution must be dense enough to ensure that the shadow of each neuron is complete. Unfortunately, the relationship

between the density of training points and the shape of $SH(a_i)$ depends on the exact kinematics and physical shape of the arm as well as the shapes of the M_0 receptive fields, and is quite complex. This makes it difficult to predict the minimal density needed for any arbitrary arm. However, we may note the conditions under which optimal collision avoidance learning is preserved.

Sparse training sample sets can lead to non-optimal avoidance sets, and this can lead to collisions with obstacles (see text).

Figure 4.41 Non-Optimal Learning

As long as a sub-region of $VR(a_i)$ corresponds to a set of arm configurations that all generate the same $PRF(a_i)$ only one training sample is needed in the sub-region to learn avoidance constraints for all these configurations. If a training sample does not fall in such a region, obstacle

constraints for these configurations will not be prohibited. Therefore, our density of training samples must be high enough to guarantee that all such regions are covered. Unfortunately, the shapes of these sub-regions depend on the specific shapes of the $VT(M_0)$ regions and the arm links and joints.

Only one training sample is needed in the small sub-region of $VR(a_i)$ above in order to learn obstacle constraints for all configurations in the region. The smallest diameter of the smallest of these regions determines the density of training samples needed to guarantee collision avoidance.

Figure 4.42

The Density of Training Samples Needed to Learn Obstacle Avoidance Constraints

(a) Plan an Ma Path From Neuron S to F

(b) The Direct Route From A to C Crosses the Inhibited Region For D and Therefore Produces an Arm Control Path That is Not Collision-Free

(c) The Path A-B-C Does Not Cross the Inhibited Region For D and is Therefore Collision-Free

Figure 4.43 Some Network Paths Cause Arm Collisions

The shape of a neuron shadow will depend on two things; (1) the shape of the physical arm, and (2) the number of neurons in M_a . The shape of the arm determines the area of the workspace occupied by each arm configuration, an area that will be unique for each type of arm. The number of neurons in M_a determines the average size of each Voronoi region in $VT(M_a)$. Each neuron a_i in an infinitely large M_a network will correspond to exactly one arm configuration, p_k , and $PRF(a_i) = CRF(p_k)$. In this case, $SH(a_i)$ is simply the union of all points which are elements of receptive fields in $CRF(p_k)$. As we decrease the number of neurons, the number of receptive fields which intersect configurations in $VR(a_i)$ may change. If the original $CRF(p_k)$ regions continue to cover all configurations in $VR(a_i)$, $SH(a_i)$ will not change. However, if any of the newly added configurations is not covered by $CRF(p_k)$ or fewer regions are needed to cover all of the new configurations, $SH(a_i)$ will change. The point at which the original $CRF(p_k)$ regions no longer properly cover the set of arm configurations in $VR(a_i)$ will depend on the shape of the arm relative to the shapes of the receptive fields. See Figure 4.42.

4.5.5. Collision Uncertainty Constrains Path Finding

In an ideal world, our representation of arm configuration space would be continuous, and arm movements could be represented by any continuous path through the free regions of this space. Unfortunately, a truly continuous representation by the M_a network would require an infinite number of neurons — one for each arm configuration point. Therefore, we create an approximation of the configuration space by using fewer neurons to represent larger numbers of configuration points. These neurons thereby quantize the space. This discrete quantization imposes constraints on the path planning process. To better understand these constraints and devise an appropriate path planning algorithm, we make use of the Voronoi tessellation $VT(M_a)$.⁶

Let us assume that the starting arm configuration A in Figure 4.43c activates neuron a_1 , and a target configuration C activates neuron a_3 . Figure 4.43a shows a section of the M_a Kohonen network which contains neurons a_1 and a_3 , and neighbouring neurons a_2 and a_4 . Figure 4.43b illustrates a typical set of arm configuration weights associated with these neurons ($\mu(a_1)$ through $\mu(a_4)$) and the Voronoi regions that they define ($VR(a_1)$ through $VR(a_4)$). Our goal is to plan a connected network path from neuron a_1 to neuron a_3 in M_a .

Let us suppose that an obstacle is introduced into the workspace that intersects one of the arm configurations C associated with neuron a_3 (Figure 4.43c). A simple, though naive, solution is to plan a path through M_a which avoids a_4 by directly connecting neighbouring neurons a_1 and a_3 . However, a straight line path from $\mu(a_1)$ to $\mu(a_3)$ crosses the inhibited (shaded) Voronoi region $VR(a_4)$, thereby possibly colliding with the obstacle. It is, of course, possible that $VR(a_4)$ configurations on this path do not collide with the obstacle — only one of the infinite number of

$VR(a_3)$ configuration points need collide with the obstacle to inhibit a_3 . Thus, uncertainty is introduced because the planning path may or may not include an a_4 configuration that collides with the obstacle. One of the effects of quantization then, is to introduce collision uncertainty into paths that cross inhibited Voronoi regions.

How might we avoid introducing collision uncertainty into our movement paths? The answer is to constrain our paths as follows; the path must cross from one un-inhibited Voronoi region to another at a point on a Voronoi edge shared by the two regions. The simplest type of curve that satisfies this constraint consists of line segments that connect defining weight vectors to points which bisect shared Voronoi edges. As described in Section 4.5.1., Voronoi regions are convex polygons. From the definition of a convex set [Preparata and Shamos, 1985], the line segment $\mu_i p_b$ connecting weight vector μ_i to bisecting point p_b is entirely contained in $VR(a_i)$. Therefore, $\mu_i p_b$ will not cross from the interior of $VR(a_i)$ to any other — possibly inhibited — region of $VT(M_a)$, and all configurations on this line segment are guaranteed collision-free. Similarly, the line segment $\mu_j p_b$ which continues the path to region $VR(a_j)$, will also be collision-free. All line segments on the path are collision free, and therefore the entire arm movement path is collision-free. The network path in Figure 4.43a that connects A to B to C allows our robot to avoid colliding with the obstacle that inhibits a_4 because the line segments that connect these weights cross the edges shared by their end-point regions.

How can we ensure that our network paths are only those that cross at a point on an appropriate Voronoi edge? There are two possibilities:

1. The path finder can directly compute the Voronoi diagram from the Kohonen weights using a conventional serial algorithm such as that described in [Preparata & Shamos, 1985]. This will take $O(N(M_a)\log N(M_a))$ time optimally. Then, walking the list of Voronoi edges and pairs of defining points that results, we can access each pair of weights that define the edge in constant time, and create pairs of line segments that connect each weight with a point on the edge (say its mid-point). This operation will take $O(N(M_a))$ time, and once learning is complete a graph of line segments connecting weights and mid-points corresponds — that represents collision-free paths — can be computed in $O(N(M_a)\log N(M_a))$ time. This data structure can then be used by a serial path planner.
2. If the Kohonen competition performed by the M_a and M_e networks is continuous rather than discrete, we can grow connections between neurons that share an edge. As the arm swings through the workspace it will assume configurations that fall exactly on $VT(M_a)$ edges. The CBUS competition can allow all neurons with the same activation to win (ie) leave out step 4). This means that occasionally two neurons will activate simultaneously

because the arm has crossed their shared edge. We can take advantage of this simultaneous activation to grow a connection using our Hebbian connection growth laws. Therefore by allowing intra- M_a connection growth we can grow the collision-free path graph. We will not pursue this idea but it is an interesting topic for further research.

4.5.6. Controlling System Precision

As we have seen, given enough training samples, the AAC will duplicate the distribution of training samples in layers M_a and M_e and will optimally grow connections between the three maps. However, even after optimal learning has taken place, there are two remaining sources of system imprecision. Both forms of error can be attributed to quantization of the three AAC input spaces. In this section we characterize these types of imprecision and show how they are related to the number of neurons in the three AAC networks. To illustrate this relationship, we derive a rough estimate of the number of neurons needed for our simple 2 degree-of-freedom, revolute jointed arm.

4.5.6.1. Positioning Error

The first type of imprecision is the amount of error inherent in the target arm configurations that the AAC derives from $\omega(e_k, a_j)$ connections. Once the TLC has selected a target point to be touched by the arm, connections from M_e activate a set of target M_a neurons which represent potential arm configurations that position the end-effector near the target point. These final configurations of the arm do not necessarily actually touch the target point, however. Rather, the end-effector reference points fall within a certain circle of radius E_p centered on the target point (Figure 4.44). We shall refer to this radius of allowable error as the positioning error of the AAC.

What is the source of this positioning error? The final configuration of the arm is taken from the weight vector associated with a target M_a neuron a_i . We know from Section 4.5.4.2., that this neuron is activated by an M_e neuron e_j because the Voronoi regions $VR(a_i)$ and $VR(e_j)$ overlap when mapped into the workspace; e_j has been activated because the target point-of-gaze p_g is included in the mapped $VR(e_j)$, and the final position of the end-effector will be the workspace point p_f associated with the mapped weight vector μ_i which defines $VR(a_i)$. There is no guarantee that p_g and p_f will be coincident. Therefore the amount of error introduced by quantization of the maps is given by the Euclidean distance between the point-of-gaze and the mapped weight vector p_f in the workspace.

We are now in a position to relate the number of neurons in M_a and M_e to the maximum positioning error E_p . The worst-case error E_p will occur when the mapped regions $VR(a_i)$ and $VR(e_j)$ intersect at a single point such that this point and the endpoints of the maximum diagonals of the regions are all co-linear. In this case, E_p is maximized if a_i and e_j lie at opposite ends of the

line segment produced by the two diagonals (Figure 4.45). In turn, the average length of these diagonals (Voronoi region size) is determined by the number of neurons in the M_a and M_e networks.

Figure 4.44 Positioning Error

Figure 4.45 Worst Case Positioning Error

4.5.6.2. Avoidance Error

The second type of imprecision due to quantization is associated with obstacle avoidance constraints that the AAC derives from $\omega(o_i, a_j)$ connections. As we have seen in Section 4.5.4.4., M_a neuron a_i will be inhibited whenever an obstacle intersects any of the receptive fields of M_e neurons contained in its avoidance set $A(a_i)$. Recall that a_i actually represents a range of arm configurations. Many of these may not collide with the obstacle but are excluded from the planning process anyway in order to guarantee that configurations on a path through active neurons in M_a are collision-free. Because some collision-free arm configurations are inhibited by the AAC, the

path finding process will not always find all potential solution paths. We will refer to this process as **conservative inhibition**.

(a)

(b)

Figure 4.46 Worst Case Positioning Error

The effect of conservative inhibition is to prevent any part of the physical arm from approaching an obstacle closer than a distance E_a . We will refer to E_a as **avoidance error**. Figure 4.46 illustrates the type of situation in which avoidance error can be important. In Figure 4.46a two obstacles form a tunnel through which the arm must reach to touch the target. The receptive fields in $PRF(a_2)$ are shown superimposed onto the workspace. A portion of the associated

$VT(M_a)$ is shown to the right. The bottom region represents the initial configuration of the arm and the top region represents the activated target configuration. The only path to the target is through the middle region. The shaded regions represent regions containing colliding configurations. Since neither of the obstacles intersects any of the receptive fields in $PRF(a_2)$, the middle region remains active and a path can be found as shown. However, if the obstacles are a little closer to the arm and either one intersects any of the receptive fields in $PRF(a_2)$, the middle region neuron will be inhibited and no path will be found through the inhibited region (Figure 4.46b).

The avoidance error E_a is clearly tied to the size of receptive fields in M_0 . Figure 4.47 depicts the worst case conditions that determine the maximum size of E_a . Consider a single receptive field $VR(o_j) \in PRF(a_i)$. We know that $VR(o_j)$ contains at least one point p_r on the robot arm. In order for conservative inhibition to occur, at least one point p_o must also be contained in $VR(o_j)$. The distance between these points is E_a . In order for E_a to be maximized, it must be the maximum diagonal of $VR(o_j)$. The maximum value of E_a for a given AAC will be the largest such diagonal in $VT(M_0)$. If we make the reasonable assumption that all receptive fields are of equal size, the maximum diagonal is determined by the number of neurons in the M_0 network.

Figure 4.47 Worst Case Avoidance Error

4.5.6.3. Estimating the Number of Processors Needed

Let us now apply our knowledge of positioning and avoidance error to the task of estimating the number of neurons that we will require in each layer of the AAC. This estimate will be subject to the precise details of the Voronoi region shapes but can be taken as a rough approximation of the neuron requirements for 2-DOF Horn robots.

In order to proceed, we must make some simplifying assumptions about the shapes of the Voronoi regions in $VT(M_0)$, $VT(M_a)$ and $VT(M_e)$. Fortunately, we can control the shape of $VT(M_0)$ and we have seen qualitative descriptions of $VT(M_a)$ and $VT(M_e)$ in sections 4.5.2.1.1.1. and 4.5.2.1.1.2.

We will assume that $VT(M_0)$ is a square grid with an equal number of cells per side. If the diameter of the workspace is D_w and there are $N(M_0)$ neurons, then each receptive field $VT(M_0)$ is a square with side

$$s_0 = \frac{D_w}{\sqrt{N(M_0)}} \quad [4.5.6.3.1]$$

and diagonal

$$E_a = d_0 = \frac{\sqrt{2} \cdot D_w}{\sqrt{N(M_0)}} \quad [4.5.6.3.2]$$

From section 4.5.6.2. we know that $E_a = d_0$, and therefore by substituting E_a for d_0 and rearranging 4.5.6.3.2, the number of neurons needed in M_0 to ensure an avoidance error E_a is

$$N(M_0) = \frac{2 D_w^2}{E_a^2}, \quad [4.5.6.3.3]$$

In order to size the M_a and M_e networks for a given value of E_p , we need to know the maximum diameter of the characteristic sickle shaped regions in $VT(M_a)$. Let us assume that M_a is a square, 2-dimensional Kohonen network. We further assume that the joints are able to rotate through 2π radians¹⁸. Figure 4.42 qualitatively shows the distribution of weight vectors in $VT(M_a)$. Notice that a uniform random distribution of points of gaze during learning distributes more neurons to the $\theta_2=0$ region of arm configuration space than the $\theta_2=\pi$ region. This distribution compensates for the larger circumference of the circle enscribed by the arm in $\theta_2=0$ configurations. More neurons lead to smaller Voronoi regions which are similar in size to the regions at $\theta_2=\pi$. We can simplify our estimate by assuming that $VT(M_a)$ is a uniform grid covering arm configuration space and calculating the maximum diameter of these regions from the regions at $\theta_2=0$. This is a safe assumption to make since because fewer neurons will be devoted to $\theta_2=0$, yielding larger Voronoi regions than anywhere else and therefore a safe — though very conservative — estimate of their maximum diameter.

Given a square M_a network with $N(M_a)$ neurons, the side of each region at $\theta_2=0$ will be

$$s_a = \frac{2\pi}{\sqrt{N(M_a)}} \text{ radians.} \quad [4.5.6.3.4]$$

The maximum length of this side when mapped to the workspace depends on the maximum reach of the robot (L_1+L_2), and is given by

$$s_a' = 2\pi(L_1+L_2) \cdot \frac{s_a}{2\pi} = s_a (L_1+L_2). \quad [4.5.6.3.5]$$

¹⁸ Normally the body will prevent many of the arm configurations possible. In order to derive an upper-bound estimate we can safely ignore the body. The larger the body, the smaller the area of arm configuration space covered by the M_a network, and the fewer M_a processors needed to achieve the same degree of precision.

Now, the side s_a'' must always be less than or equal to the length s_a' since

$$s_a'' = 2\pi L_1 \cdot \frac{s_a}{2\pi} = s_a L_1 \leq s_a'. \quad [4.5.6.3.6]$$

Therefore, the diagonal — which is the maximum diameter — of each region cannot be greater than twice the length of s_a , and an extremely conservative estimate for the mapped diameter of the region is

$$d_a' = 2s_a'. \quad [4.5.6.3.7]$$

The defining points in $VT(M_a)$ almost always occur at or near the centroid — midpoint of the diagonal — of the Voronoi regions, therefore, a more reasonable, though still conservative, estimate of the diameter is given by

$$d_a' = s_a' = \frac{2\pi(L_1+L_2)}{\sqrt{N(M_a)}} \quad [4.5.6.3.8]$$

Rearranging 4.5.6.3.7 and substituting for s_a' , we can now solve for $N(M_a)$,

$$N(M_a) = \left(\frac{2\pi(L_1+L_2)}{s_a} \right)^2. \quad [4.5.6.3.9]$$

Next, we need to know the diameter of regions in $VT(M_e)$. We shall approximate the roughly diamond shaped regions characteristic of $VT(M_e)$ by squares. In this case we know that all the neurons in M_e are devoted to the region of eye configuration space which maps to the workspace. Therefore, the sum of the area of the mapped $VT(M_e)$ regions should be roughly the area of the workspace itself. The area of the — possibly doughnut shaped — workspace is given by

$$W_a = 4\pi L_1 L_2, \quad [4.5.6.3.10]$$

and we can approximate the area of the $VT(M_e)$ regions by

$$a_e = \frac{W_a}{N(M_e)}. \quad [4.5.6.3.11]$$

The area of a square $VT(M_e)$ can also be related to the length of the side of the region s_e as follows,

$$a_e = s_e^2, \quad [4.5.6.3.12]$$

and rearranging and substituting 4.5.6.3.11 we solve for s_e in terms of the number of M_e neurons as follows

$$s_e = \sqrt{\frac{4\pi(L_1 L_2)}{N(M_e)}}. \quad [4.5.6.3.13]$$

The diagonal of the square region is

$$d_e = \sqrt{\frac{8\pi(L_1 L_2)}{N(M_e)}}. \quad [4.5.6.3.14]$$

Finally then, from section 4.5.6.1. we know that $E_p = d_a' + d_e$ so

$$E_p = \frac{2\pi(L_1+L_2)}{\sqrt{N(M_a)}} + \sqrt{\frac{8\pi(L_1L_2)}{N(M_e)}} \quad [4.5.6.3.15]$$

It is clear from equations 4.5.6.3.2, and 4.5.6.3.14 that the error terms can be made as small as desired at the expense of rapidly increasing network sizes.

4.5.6.4. Minimum Neuron Requirements

Suppose that we wish to use the equations derived in the last section to design an AAC with a given degree of accuracy. To reduce costs we always want to use the minimum number of neurons necessary to achieve this degree of accuracy. Equation 4.5.6.3.3 can be used to find the number of M_0 neurons directly from the desired avoidance error, but positioning error depends on the combined number of neurons in the M_a and M_e networks. Given a value for E_p , how can we find the sizes of the M_a and M_e networks such that the total number of neurons is minimized?

We wish to minimize the function: $f = N(M_a) + N(M_e)$

subject to the constraint: $E_p = \frac{2\pi(L_1+L_2)}{\sqrt{N(M_a)}} + \sqrt{\frac{8\pi(L_1L_2)}{N(M_e)}}$

This is a classic problem of constrained minimization and is easily solved with Lagrange multipliers as follows:

Let $x = N(M_a)$, $y = N(M_e)$, $a = 2\pi(L_1+L_2)$, and $b = \sqrt{8\pi(L_1L_2)}$.

Minimize $f = x + y$

subject to the constraint: $E_p = \frac{a}{\sqrt{x}} + \frac{b}{\sqrt{y}}$

which can be expressed as: $g(x,y) = \frac{a}{\sqrt{x}} + \frac{b}{\sqrt{y}} - E_p = 0. \quad [4.5.6.4.1]$

Taking partial derivatives of these equations, two parallel vectors orthogonal to $f(x,y)$ and $g(x,y)$ are

$$\begin{aligned} \nabla f &= \frac{\delta y}{\delta x} i + \frac{\delta x}{\delta y} j = i + j \\ \nabla g &= \frac{\delta(ax^{-1/2})}{\delta x} i + \frac{\delta(by^{-1/2})}{\delta y} j \\ &= \left(-\frac{1}{2}x^{-3/2}\right)a i + \left(-\frac{1}{2}y^{-3/2}\right)b j \\ &= -\frac{a}{2}x^{-3/2} i - \frac{b}{2}y^{-3/2} j \end{aligned}$$

Applying a Lagrange multiplier, we have

$$\nabla f = \lambda \nabla g \quad \therefore$$

$$i + j = \lambda \left(-\frac{a}{2} x^{-3/2} \right) i + \lambda \left(-\frac{b}{2} y^{-3/2} \right) j$$

Separating the vector components, we have

$$1 = \left(-\frac{a}{2} x^{-3/2} \right) \lambda \quad \text{and} \quad [4.5.6.4.2]$$

$$1 = \left(-\frac{b}{2} y^{-3/2} \right) \lambda. \quad [4.5.6.4.3]$$

Equating 4.5.6.2.15 and 4.5.6.2.16 gives us

$$-\frac{a}{2} x^{-3/2} = -\frac{b}{2} y^{-3/2}. \quad [4.5.6.4.4]$$

Rearranging 4.5.6.2.17

$$y^{-3/2} = \frac{a}{b} x^{-3/2},$$

$$y^{-3} = \left(\frac{a}{b} \right)^2 x^{-3},$$

$$y = \left(\frac{a}{b} \right)^{-2/3} x. \quad [4.5.6.4.5]$$

Rearranging 4.5.6.2.14

$$0 = \frac{a}{\sqrt{x}} + \frac{b}{\sqrt{y}} - E_p,$$

$$\frac{b}{\sqrt{y}} = E_p - \frac{a}{\sqrt{x}},$$

$$\frac{b^2}{\sqrt{y}} = \left(E_p - \frac{a}{\sqrt{x}} \right)^2,$$

$$y = \frac{b^2}{\left(E_p - \frac{a}{\sqrt{x}} \right)^2}. \quad [4.5.6.4.6]$$

Equating 4.5.6.4.5 and 4.5.6.4.6, we can solve for x

$$\left(\frac{a}{b} \right)^{-2/3} x = \frac{b^2}{\left(E_p - \frac{a}{\sqrt{x}} \right)^2},$$

$$\left(\frac{b}{a} \right)^{1/3} \sqrt{x} = \frac{b}{\left(E_p - \frac{a}{\sqrt{x}} \right)},$$

$$\frac{\left(\frac{b}{a}\right)^{1/3} \sqrt{x}}{b} = \frac{1}{\left(E_p - \frac{a}{\sqrt{x}}\right)},$$

$$\frac{b}{\left(\frac{b}{a}\right)^{1/3} \sqrt{x}} = E_p - \frac{a}{\sqrt{x}},$$

$$\frac{b \cdot a^{1/3}}{b^{1/3}} x^{-1/2} = E_p - a x^{-1/2},$$

$$a^{1/3} b^{2/3} x^{-1/2} + a x^{-1/2} = E_p,$$

$$E_p = x^{-1/2} (a + a^{1/3} b^{2/3}),$$

$$x^{-1/2} = \frac{E_p}{(a + a^{1/3} b^{2/3})},$$

$$x = \frac{(a + a^{1/3} b^{2/3})^2}{E_p^2} \quad [4.5.6.4.7]$$

Now substituting 4.5.6.4.7 into 4.5.6.4.6 we can solve for y

$$y = \frac{b^2}{\left(E_p - \frac{a E_p}{(a + a^{1/3} b^{2/3})^{1/3}}\right)^2} \quad [4.5.6.4.8]$$

Substituting the original values of x, y, a, and b into 4.5.6.4.7 and 4.5.6.4.8, we now have equations to compute the three network sizes that minimize the total number of neurons in the AAC for a given degree of accuracy.

$$N(M_0) = \frac{2 D_w^2}{E_a^2}, \quad [4.5.6.4.9]$$

$$N(M_a) = \left(\frac{(2\pi(L_1 + L_2)) + (16\pi(L_1 + L_2)(L_1 L_2))^{1/3}}{E_p} \right)^2. \quad [4.5.6.4.10]$$

$$N(M_e) = \left(\frac{8\pi(L_1 L_2)}{\left(E_p - (2\pi(L_1 + L_2)) \frac{E_p}{(2\pi(L_1 + L_2)) + ((2\pi(L_1 + L_2))(8\pi(L_1 L_2)))^{1/3}}\right)^2} \right)^2. \quad [4.5.6.4.11]$$

These equations give us a theoretical basis for understanding the simplest version of the adaptive arm controller described in this thesis. Network sizes very rapidly increase as positioning error decreases. Although the number of neurons may at first seem prohibitively large, empirical results presented in the next section suggest that simple path planning heuristics can reduce the number of neurons needed by as much as an order of magnitude. Without further refinements of the approach however, technologies other than VLSI will be needed to provide the large quantities of neurons

and inter-map connections needed to control complex arms with a high degree of precision. Nevertheless, this approach does demonstrate the possibility of collision-free control of general robot manipulators by unsupervised adaptive learning machines.

4.6. A Simulation of The Adaptive Arm Controller

In order to better evaluate the ideas outlined in this chapter, we have written a simulation package in ParcPlace™ Smalltalk¹⁹ which runs on Mac II personal computers. We selected Smalltalk as the language of implementation because the object-oriented programming paradigm is particularly well suited to the simulation of interactions among a set of neural networks²⁰. Furthermore, Smalltalk's pre-defined graphics and window control classes are ideal for rapid development of graphical user-interfaces. We will assume that the reader is familiar with the Smalltalk programming environment — those readers who are not are directed to [Goldberg & Robson, 1983] for a description of the language. We will not describe the implementation details of the simulator in this thesis. Those readers interested in duplicating the simulator will find a complete listing of the source code in [Graf 1988b].

In this section we present simulation results for a representative set of Horn robots. Our simulations support the geometric analysis in section 4.5., and suggest directions for further research.

4.6.1. General Approach

The simulator is intended to demonstrate the learning behaviour of the AAC during its initial, two-step training phase, and the capability of a trained controller to touch target points. The user is allowed to specify the eye-arm-body shape of a 2-dimensional, 2 degree-of-freedom, revolute jointed Horn robot, and he can examine the state of the AAC at any time during or after learning. The simulation is limited to simple Horn robots so that the eye and arm configuration spaces can be easily displayed during learning. A 2-dimensional M_a network also makes it possible for the user to select network paths interactively. Note that it will be a relatively simple matter to extend the simulation to more complicated robots for future research at the expense of a more complicated user-interface.

Random arm movement during training is simulated by selecting a random distribution of points within reach of the robot. Two approaches have been used to select random workspace points. In the first, each joint angle is changed by a small random amount and forward kinematic

¹⁹ ParcPlace Smalltalk is a trademark of ParcPlace Systems, Inc.

²⁰ As described in chapter 2, neural networks are naturally viewed as black box 'objects' with simple interface protocols.

equations are used to determine the point-of-touch. The point-of-touch serves as the random workspace point, and the inverse kinematic equations of the eyes (section 4.5.3.3.) are used to find the corresponding foveating eye configuration. This approach leads to a uniform random distribution of points in arm configuration space but a non-uniform distribution in the workspace. In the second approach a uniform random distribution of workspace points is directly generated by a random point generator. Inverse kinematic equations for the arm (section 4.5.3.2.) are used to select an arm configuration that touches the point, and test is performed to determine if the arm intersects the robot's body. If no intersection occurs, the workspace point is accepted and the corresponding eye configuration is found. The distribution of arm configuration vectors in arm configuration space that results from this process is non-uniform. Simulations have been performed using both approaches, and the results compared. We have found that the average ability of the arm to touch selected points is approximately the same after each type of training. The results presented in the following sections assume uniform workspace distributions.

These random arm and eye configurations are used as training samples during a simulation of the self-organization and inter-layer association steps. At each time step of the self-organization simulation, a pair of training configurations is presented to the M_a and M_e layers and one iteration of the Kohonen algorithm is executed for both maps. After a user defined number of iterations, the system switches to a simulation of the inter-map association step. The difference equations described in sections 4.4.1.2.1. and 4.4.1.2.2. are not directly simulated. Instead, connections are recorded for each pair of adjacent map neurons that are active during a time step. This process simulates *idealized* growth of connections. A typical simulation run for an AAC containing 2000 neurons executing 100,000 self-organization time steps and 10,000 inter-map association time steps requires approximately 96 hours of real time to complete. If the system were implemented in fully parallel hardware and each time step took 1 millisecond to complete (a conservative figure), the robot could complete its entire training run in less than five minutes. As we will see, good results can be obtained within 3000 self-organization time steps, reducing this training duration to approximately 10 seconds. It is likely that, with regard to speed of training, the maximum velocity of the arm joints will always be the limiting factor in a fully parallel implementation.

Once the training simulation has finished, the user is free to test the control capability of the AAC by selecting a target point. The simulator displays the resulting target configuration neurons, and the user can select a network arm control path. The resulting trajectory of the robot arm is animated.

Rather than performing a standard training run, the user may opt to generate learning curve data. In this case, the system stops the self-organization step at pre-defined intervals (1, 25, 50, 100, 200, 500, 1000, 2000, 10000, 30000, and 100000 ticks) and grows a new set of inter-map

connections. Each time new connections are grown, the robot automatically attempts to touch a user defined number of uniform random workspace points. The Euclidean distance from each target configuration point-of-touch to the target point is calculated as the positioning error of the robot. Statistics are collected and written to a file for future reference.

4.6.2. The User Interface

The user opens the simulation window by evaluating the following Smalltalk statement

AdaptiveArmEditor openOn: (AdaptiveArm new).

An icon which represents the position of the upper left corner of the window appears, and the user positions it by moving the mouse cursor and clicking the mouse button at the screen position desired. A new icon representing the bottom right corner of the window appears and the user sizes the window by dragging the mouse. A box illustrating the size of the window expands and contracts as the mouse cursor moves. When satisfied with the window size, the user clicks the mouse button and the simulation window is displayed. The window consists of four equally sized sub-views as shown in Figure 4.48. The contents of these views are scaled to fit the selected window size. Note that all windows has been turned on their sides in this thesis to allow them to fit on the page. The top of each window is next to the left margin, and the right of the window is at the top of the page.

The upper left view in Figure 4.48 represents the 2-dimensional area of Cartesian space that contains the robot and its workspace. The view is always centered on the first joint in the arm, and is scaled to include the entire workspace. The robot's body is represented by a simple polygon — initially a triangle. The robot's arm is represented by two dots that represent its joints, and two line segments that represent its links. Initially, the links are both of the same length. The eyes of the robot are represented by X's to either side of the body, and the current lines of sight of the eyes are shown as dotted lines that converge at the point-of-gaze. A menu of actions can be recalled by the user as shown in Figure 4.49. These actions allow the user to interactively draw the shape of the robot's body with the mouse, to specify the length of each arm link, and to specify the position of each eye. In addition, the user may interactively add and remove polygonal obstacles to the robot's workspace. Once the user is satisfied with the robot's shape, he may select menu items to start the self-organization step, the inter-map association step or generate learning curve data. As learning proceeds, dots appear in the workspace which represent the point-of-touch associated with each training sample. The user is able to judge the quality of coverage of the workspace and see the shape of the workspace from the distribution of these points. Menu items also exist to animate the arm and eye trajectories that the user has defined (as described below); save the robot and its environment in a file for future reference, input Voronoi diagram data for the arm and eye

configuration spaces, or map these diagrams into the workspace of the robot. Once training is complete, the user can specify a target point to be touched by the AAC by positioning the mouse cursor at the point and clicking the mouse button.

Figure 4.48 The Adaptive Arm Controller Simulation Window

Figure 4.49 Simulation Window Menus

The top right view in Figure 4.48 represents the activation pattern of M_0 neurons. Active neurons are represented by black rectangles and inactive neurons are represented by white rectangles. To speed the simulation, only the outlines of objects in the workspace are represented by M_0 activations. Interior regions of obstacles cannot be intersected by an arm unless the outline of the obstacle is also intersected. Since any intersection is sufficient to inhibit an M_a neuron, only the outline regions are important to the simulation. In an actual adaptive robot all observation space regions occupied by an obstacle would be activated. The topographical ordering of the activation pattern in M_0 is an artifact of the coding technique used to find the active border neurons and is not a necessary characteristic of the network. The menu for the M_0 view shown in Figure 4.49 allows the user to set the size of this network, and to artificially activate neurons in the map to observe the effect on the M_a layer.

The bottom right view of Figure 4.48 can be used to represent the arm configuration space or the pattern of activations in M_a . When displaying the arm configuration space, the network weight vectors are shown mapped into the space in the manner of Kohonen as shown in the Figure. Weight vectors which are horizontal and vertical grid neighbours in M_a are connected by line segments. This display format makes it easy for the user to estimate how well the network has preserved its topology in the input space. If any line segments cross, the network has not yet finished 'unfolding' (refer to section 2.4.3. for an explanation of the Kohonen algorithm). A large dot represents the position of the last training vector to have been input to the network. When the pattern of M_a activations is being displayed, as we shall soon see, each neuron is represented by a dot. The size of the dot represents the distance of the neuron's weight vector from the last input vector. Large dots mean that the weight vector is close to the input vector, and small dots mean that the weight vector is farther away. This display gives the user a qualitative feel for the Kohonen competition process. The neuron that has won the Kohonen competition is represented by a rectangle. Neurons that are marked as potential target configurations by signals from M_e neurons are represented by inverted triangles. Neurons that are marked as colliding arm configurations by signals from M_0 neurons are represented by shaded rectangles. As shown in Figure 4.49, the menu for the M_a view allows the user to switch between the arm configuration space display and the M_a activation pattern. It also allows the user to change the size of the M_a network, and to mark neurons which are on the arm control path by dragging the mouse cursor over them or clicking on them. The user can also display the Voronoi tessellation of the arm configuration space produced by the final M_a weight vectors.

The bottom left view is identical to the bottom right view except that the data displayed pertains to the eye configuration space and the M_e network.

Window 1

Window 2

Window 3

Window 4

Window 5

Window 6

Window 7

Window 8

Window 9

Window 10

Window 11

Window 12

Window 13

Window 14

Window 15

Window 16

Window 17

Window 18

Window 19

Window 20

Window 21

Window 22

Window 23

Window 24

Window 25

Window 26

Window 27

Window 28 Time = 0

Window 29 Time = 500

Window 30 Time = 1000

Window 31 Time = 100000

Window 32

Window 33

Window 34

Window 35

4.6.3. Testing Adaptation to Arbitrary Robot Shapes

To demonstrate the AAC's ability to adapt to arbitrary robot geometries, we now present a series of simulation windows for an AAC with 1,156 M_o neurons, 400 M_a neurons and 400 M_e neurons. These windows are taken from simulation runs for two different arm-body combinations. Let us examine the windows for the first simulation run in some detail.

The first 4 windows represent snapshots of the self-organization step at 0 time steps, 500 time steps, 1,000 time steps and 100,000 time steps respectively. The bottom left and right views in these windows display the grid of M_e and M_a network weights mapped into the eye and arm configuration spaces respectively. At time step 0, the networks are completely randomized and the AAC has no knowledge of the kinematics of the associated robot. We found that both networks consistently finish unfolding in their respective input spaces within 500 time steps. The local point densities of the network weights have also recognizably begun to approximate the training point distributions by this time. By time step 1,000 the distribution of weight vectors is very close to the final distribution. The next 99,000 time steps are spent making small refinements to the weight vector distribution. The best approximation to the training distribution occurs within the borders of the weight grid. The accuracy of the distribution is poorer along the edge of this grid, but it is still fine enough to allow accurate movement of the arm. As we will see in the next section, the positioning accuracy of the AAC does not improve significantly beyond 3000 iterations of the Kohonen algorithm — 100,000 iterations are clearly not needed.

The next window represents the state of the system after inter-map connections have been 'grown' and before obstacles have been introduced into the workspace. The bottom two views now display the activation patterns for the M_e and M_a networks. Notice that the neurons that win the Kohonen competition to represent the current eye and arm configurations are marked by rectangles. The M_a view also contains neurons inhibited by the permanent constraint neuron discussed in section 4.4.1.2.2. and by the body of the robot. These neurons are shaded gray to indicate that they should not be included in arm control paths. This set of neurons represents the permanent kinematic constraints on arm movement imposed by the body of the robot (see sections 4.1.7.). The receptive fields of these neurons border (or in a few cases are entirely contained by) restricted configuration regions in the arm configuration space. Range restrictions are not imposed on joints in this simulation. Notice that the permanently inhibited neuron set prevents link 1 from revolving through a full 2π radians because the body is in its way. Link 2 is free to rotate through a full 2π radians except when link 1 is at its extreme positions and the body interferes.

The next two windows illustrate that, although in general, straight line arm control paths translate to curved end-effector paths in the workspace, curved arm control paths can be found that represent straight line end-effector-paths.

The next 4 windows represent a movement of the arm in the presence of obstacles. In the first window we see the initial state of the robot after a set of obstacles have been introduced into its workspace. Notice that the set of inhibited M_a neurons is now much larger than the permanently inhibited set in the last window. The newly inhibited neurons have been inhibited by connections from active M_o neurons, and represent arm configurations that intersect at least one obstacle. The current eye and arm configurations are represented by the winning M_e and M_a neurons. The next window illustrates the robot selecting a target point on the far side of an obstacle in its workspace at the next time step (the target was actually selected by the user with the mouse). Notice that the winning M_e neuron has changed to reflect this new eye configuration and that the M_a network has not changed. At the next time step, illustrated by the next window, the active M_e neuron has sent signals to four M_a neurons that correspond to arm configurations that position the tip of the arm near the target point. The physical configurations that are produced by the weight vectors of these target neurons are drawn into the workspace view to give the user some idea of the average accuracy of the controller. Notice that two of the target configurations intersect an obstacle. As desired, the associated target configuration neurons fall into the shaded region of M_a (at the top of the M_a view) and will not be included in arm configuration paths. In the next window, the user has selected an example M_a arm control path that leads from the winning initial configuration neuron to one of the uninhibited target neurons. The associated arm trajectory is shown in the workspace view. Two more example movement sequences are also shown.

A similar sequence of simulation run windows is also shown for another robot geometry. Notice that some arm control paths have been selected which wrap around the toroidal arm configuration space. It is interesting to note that although the simulation does not indicate which neurons share a Voronoi edge, all arm trajectories we have selected have correspond to collision-free movements of the arm. Cases where this is a problem very rarely occur in the Voronoi tessellations that have been produced in our simulations, and are only ever a problem when the arm is very close to an obstacle.

Eight robot geometries have been simulated for this AAC. Five robot geometries have also been simulated for AAC's of varying sizes. All simulations have produced similar collision-free arm movements — although as we shall see below, the degree of positioning error varies as expected with the number of neurons in the AAC.

Figure 4.50 Weight Distribution

Figure 4.51 Mapped M_a Weights

Figure 4.52 Mapped M_e Weights

Figure 4.53 $VT(M_e)$ and $VT(M_a)$

Figure 4.54 VT(M_a) Mapped Into The Workspace

Figure 4.55 $VT(M_e)$ Mapped Into The Workspace

4.6.4. Example Voronoi Tessellations

In order to evaluate the reasonableness of the geometric analysis and minimum neuron estimates derived in section 4.5.6.4., Voronoi tessellations have been generated for a variety of robot geometries. Figure 4.50 shows the distribution of M_e and M_a weights for an example robot²¹. In Figure 4.51, the M_a weights have been mapped into the workspace by the forward kinematic equations of the robot.² The approximately uniform distribution of weights over the robot's workspace after mapping from the decidedly non-uniform distribution in arm configuration space demonstrates that the Kohonen algorithm has done a good job of approximating the training set distribution. Figure 4.52 similarly maps the M_e weights into the workspace, and once again we see that the highly non-uniform distribution in eye configuration space is a good approximation of the underlying training set. As expected, the distribution of weights along the horizontal line passing through both eyes is less accurate because all points-of-gaze along this line are represented by the same four eye configurations. The bottom left and right views in Figure 4.53 illustrate the Voronoi tessellations that are produced by Kohonen competitions in M_e and M_a respectively. Figure 4.54 shows the M_a tessellation after mapping into the workspace. Notice that it has the same flattened toroidal shape that was described in section 4.5.2.1.2. Figure 4.55 shows the M_e tessellation after mapping into the workspace. The simplifying assumption we made in section 4.5.6.3. regarding the shapes of these mapped M_e regions clearly does not hold near the edges of the workspace, however it is still a good average approximation.

4.6.5. Learning and The Relationship Between AAC Size and Positioning Error

In section 4.5.6.4. we derived an estimate for the number of random training samples needed to produce an acceptable level of inter-map connection growth. We were unable to say much about the number of iterations of the Kohonen algorithm needed to approximate the training sample distributions however, because the behaviour of the algorithm is highly complex and has not been formally analysed with enough precision. In this section we present empirical results that illustrate the speed of the self-organization step and help us select the minimum number of Kohonen iterations required for good performance of the system. These results also confirm the exponential relationship between network size and positioning accuracy that we derived.

Learning curves — as described in section 4.6.1. above — were generated to produce a plot of positioning error versus number of self-organization time steps. Average positioning error is

²¹ The Voronoi diagram generator used is limited to a maximum of 200 defining points so an AAC with 1056 M_o neurons, 196 M_a neurons and 196 M_e neurons was used for these tests.

maximized if the Voronoi regions of $VT(M_a)$ are distributed over the entire configuration space because, in this case, the average size of a $VT(M_a)$ regions is larger. Therefore, to obtain worst case results, simulations were performed for a robot with no body. This situation allows both joints to revolve through 2π radians, distributing training samples over the entire arm configuration space. The arm was of roughly human proportions, consisting of two revolute joints attached to links of equal length — 15 inches each. Ten repetitions of the experiment were performed and results averaged for five different sizes of controller. Each AAC tested had equally sized M_e and M_a networks with either 4, 16, 64, 100, 196, or 400 neurons. Since the observation space map does not affect positioning accuracy, each AAC contained a 4 neuron M_b network to reduce storage requirements and simulation time.

Figure 4.56 is a plot of average positioning error versus the number of ticks of the simulation clock. The learning behaviour of the system is consistent for all controllers tested. In each case, average error rapidly decreases during the first 1000 iterations of the Kohonen algorithm, and by 3,000 iterations accuracy levels off. There is some continued improvement in accuracy to 100,000 iterations but the amount of improvement during this period is insignificant.

Figure 4.56 Average Positioning Error Vs Trials

Figure 4.57 is a plot of average positioning error variance versus simulation ticks. We see that the variation in positioning error also decreases in roughly the same manner as the average amount of positioning error. Therefore, not only does the distance between the end-effector and the target point decrease, but the end-effector positions of all target configurations tend to form tighter clusters around the target as time passes. Once again improvement in accuracy virtually ceases beyond 3,000 iterations.

Figure 4.57 Average Minimum Positioning Error Vs Trials

Figures 4.58 and 4.59 represent the average minimum positioning error and the variation in average minimum positioning error. Both plots show characteristic rapid improvement during the first 3,000 iterations and subsequent slow improvement over the next 97,000 iterations. The variation in average minimum positioning error is extremely small once learning is complete. It is also apparent that the average *minimum* positioning error is significantly better than the average positioning error. Why should there be such a large difference? Investigations of mapped Voronoi tessellations and target configuration distributions have revealed that approximately 10% to 20% of target configurations associated with each M_e neuron have a point-of-touch far outside the cluster of remaining target configurations for that neuron. Thus a small percentage of poor arm configurations are responsible for significantly increasing the average positioning error. Excluding these poor target configurations from the learning curve test brings the average positioning error

closer to the average minimum. This suggests a simple heuristic to increase the accuracy of the AAC. As the AAC is used to touch target points, average positioning error statistics can be recorded for each target configuration (this assumes a method for estimating this error visually is available, and that the averages can be stored for each target — possibly as the strength of connection between the M_e and M_a layers). After compilation of these statistics, the path finding process would try to find paths to target configurations in order of average error, starting with the target with lowest error, and moving to the next highest if this one is inhibited (the signal level received by the target from the M_e layer would indicate the average error if it is stored as the connection strength). This simple strategy would reduce the average positioning error while allowing the number of proposed targets to remain the same.

Average Variance vs Number of Trials

Figure 4.58 Positioning Error Variance Vs Trials

Figure 4.59 Minimum Positioning Error Variance Vs Trials

It is interesting to examine the relationship between the predicted positioning error, the average positioning error and the average minimum positioning error. The top curve in Figure 4.60 is a plot of the ratio of actual average error to predicted error versus time. As we see, the actual average errors are consistently smaller than the predicted values, although they are never more than 60% smaller. As the number of neurons increases, the difference between the actual and predicted values decreases. The bottom curve in Figure 4.60 illustrates the rather startling relationship between average minimum positioning error and predicted positioning error. The average minimum is *exactly* one order of magnitude smaller than the predicted value — at least within the range of network sizes tested.

Figure 4.60 Actual / Predicted Positioning Error Ratios

Figure 4.61 shows a plot of the average number of target configurations associated with each eye configuration neuron. Once again, by 3,000 iterations this value has stabilized. About 10 configurations are grown from each M_e neuron to the M_a layer regardless of the number of neurons in the AAC. This consistency is not surprising given that the M_e and M_a network sizes tested were always equal. The *relative* sizes of the $VT(M_e)$ and $VT(M_a)$ regions will always be approximately

equal in this case. We expect the number of target configurations to vary with the difference between M_e and M_a network sizes.

Figure 4.61 Average Number of M_e to M_a Connections

Finally, Figure 4.62 illustrates the relationship between positioning error and the combined size of the M_e and M_a networks. The number of neurons axis is a logarithmic scale. The top scale represents positioning error in terms of percentage of possible error, and the bottom scale represents actual error in inches. The three curves represent the predicted positioning error, average

positioning error, and average minimum positioning error. The predicted values are for the minimum combined network sizes and were calculated using equations 4.5.6.4.10. and 4.5.6.4.11. The average positioning error and average minimum positioning error curves have been extrapolated from the curves in Figure 4.60.

Total Number of Neurons Vs Positioning Error

Figure 4.62 Number of Neurons Versus Positioning Error

The AAC performed better than predicted on average, however, the empirical results do suggest that the network size to error relationship derived in section 4.5.6.4. is a good

approximation. Differences between the analysis and the empirical results occur because of the simplifying assumptions made with regard to the tessellation region shapes. As we have seen, the addition of a few simple heuristics to the path finding process should improve average performance significantly — shifting it closer to the average minimum curve and reducing the number of neurons needed by approximately one order of magnitude.

In order to give the reader a better feel for the implications of these results, let us look at some actual numbers. To control a human sized arm with an average of 1.5% accuracy (1 inch positioning error), the predicted combined M_a - M_e size is 85,000 neurons. Extrapolated empirical results show that approximately 10,000 neurons are actually needed to achieve this degree of accuracy. Finally, we have seen that by making the path planning mechanism smarter we should be able to reduce the number of neurons needed to approximately 8,500. Recall that fully connected prototype optical networks with 10,000 neurons have already been constructed to implement the Hopfield algorithm [Abu-Mostafa & Psaltis, 1987] — more than the number of neurons and connections needed to implement a controller with 1.5% average minimum error. As we have seen, the variation in positioning error will be quite small for a controller of this size. Remember that the AAC will produce collision-free movements of the arm regardless of the amount of positioning error.

For tasks requiring less accuracy, say 8% (5 inch positioning error), the predicted combined M_a - M_e size is only 5,210, the extrapolated empirical results show that only 800 neurons are needed, and about 500 neurons are required for this degree of average minimum positioning error. It may be possible to employ VLSI to produce these small densities of neurons and connections. Other simple mechanisms such as those described in [Andrade-Abalarach, 1988] could be used to manoeuvre the end-effector closer to the target using visually derived information regarding the amount of positioning error.

4.6.6. Problems Encountered

Four problems have been encountered during simulations. The first is a result of avoidance error and has already been mentioned in section 4.5.6.2. As Figure 4.63 shows, it is possible for some target configurations which do not intersect an obstacle, but which nevertheless fall within the receptive field of an M_o neuron, to be inhibited. In this figure six of the eight target configuration neurons have been inhibited by avoidance error and will not be considered for arm control path.

Figure 4.63 . Avoidance Error

The obvious solution is to increase the number of M_0 neurons to reduce the expected avoidance error value. Careful analysis of AAC simulations has revealed that neurons inhibited by avoidance

error are invariably neighbours to at least one uninhibited neuron. This suggests the following last-resort heuristic. If the path planner discovers that all target neurons are inhibited or unreachable, select a target neuron that is bordered by at least one uninhibited neuron. If such a target neuron exists, plan a path to it, and reduce the velocity of the arm joints significantly during the last quarter of the uninhibited neuron trajectory. In this way, the AAC will slow the arm near the obstacle and gently attempt to position the end-effector at the inhibited configuration point-of-touch. If resistance to movement is encountered, the AAC can plan another path from its current position to the next inhibited target. In this way the AAC can easily explore all potential avoidance error configurations in an intelligent fashion at the expense of having to gently 'feel' its way around the target area.

The Kohonen algorithm exerts less force on each weight vector at the edge of the M_e and M_a networks than it does on weight vectors on the interior of the network. These neurons only have neighbours which tug on them from three sides of the network (see the Kohonen algorithm description in section 2.4.3.). Consequently, the edge of the network does not align with the edges of its configuration space. The result is that target configurations never quite reach the edge of the workspace. This is not a significant problem however and several possible solutions exist. The first solution is to increase the number of neurons in the M_a network. The second solution is to modify the Kohonen algorithm to take the imbalance in forces on a neuron into account. In this case the amount of pull exerted on one neuron by another would be weighted by the relative positions of the neurons in the network. Alternatively, the toroidal nature of the input space could be taken into account, and the network could be considered a toroid. In this case the network would have no edges and forces exerted on each neuron are equal in all directions. This second method has recently been implemented and initial indications at the time of this writing are that it is successful in dealing with edge effects. Further experimentation is required however.

The third problem is caused by the poor distribution of mapped eye configuration weights along the line passing through the center of both eyes. Positioning accuracy along this line is poor. This is not a significant problem and there are three solutions. The first solution is to add another degree of freedom to the sensors (ie. implant the eyes in a rotating head). This will allow every point in the workspace to be associated with unique eye configurations. This is an expensive solution however, because it increases the dimensionality of eye configuration space by one. The second solution is to do nothing. Biological organisms, including man, perform most of their tactile and manipulatory work in front of their bodies, not to the side. Simply avoiding awkward targets along this line and selecting appropriate positions for the eyes is not an unreasonable solution. The third solution is to allow the robot to rotate itself so that targets at the side of its head can be positioned in front of it. These targets are very easy to identify from the four eye

configurations that are required to gaze at them. Little extra intelligence would be required to implement this third solution.

The fourth and final problem is dependent on the initial random condition of the network. In approximately 5% of simulations either the M_a or M_e network failed to 'unfold' in its input space. Instead, the network took on a twisted shape. In about a third of these cases the network eventually unfolded itself — although it required about 200,000 iterations to do so. In the remaining cases — terminated after 200,000 iterations — the network remained twisted. After approximately 100 self-organization time steps it is always clearly apparent that the network has become twisted and will remain that way. Twisting never begins after 100 time steps. The simplest solution is to examine the networks after 100 iterations and restart the training process with new random M_e and M_a weights if twisting has begun. A more elegant solution has not yet been found.

4.7. Chapter Conclusions

In this chapter we have presented a new approach for the control of general robot manipulators that possesses eight important characteristics.

1. The AAC is in principle capable of autonomously learning kinematics and workspace constraints for any arbitrary robot anatomy, dispensing with the need for specialized kinematic programming.
2. The AAC learns collision avoidance constraints which greatly simplify the collision-free trajectory planning process for arbitrarily cluttered workspaces.
3. The adaptive nature of the AAC makes it possible to automatically adapt the system to changes in eye-arm-body relationships and feedback sensor calibrations, reducing the need for maintenance.
4. The AAC does not require a system of universal, object based, or robot based coordinate frames. Targets are selected in terms of egocentric sensory coordinates — by simply 'glancing' at them.
5. The AAC can be made sensitive to the domain specific distribution of arm movements. It will automatically increase the accuracy of the arm in those regions of the workspace where precision is most needed. High precision in important regions of the workspace is increased at the expense of lower precision in regions that are rarely used.
6. The AAC is tolerant of random noise in the training data. A small number of incorrect training samples will not qualitatively alter the system's precision immediately after training. Errors that do occur are accommodated for by the process of on-going adaptation described above.

7. If implemented by massively parallel hardware, the AAC is capable of planning *arbitrarily complicated arm movements* very quickly. Inverse kinematics are calculated in constant time, and the speed of parallel path finding algorithms is a function of the number of neurons in the M_a network (e.g. the systolic array algorithm by [Miller & Stout, 1985] can compute minimal paths in $O(\sqrt{N(M_a)})$ time steps). Furthermore, the speed of the AAC is not related to the number or shape of obstacles cluttering the workspace.
8. The AAC is tolerant of component failures. Traditional control programs are highly intolerant of failures. If a single component in a conventional computer fails, or if a single line of code is corrupted, the results are catastrophic. A parallel implementation of the AAC may experience total failure of a random distribution of neurons and inter-map connections and yet continue to operate. Immediately after failure the system will no longer produce collision-free movements in certain regions of the workspace, but it will continue to function as before everywhere else. After automatic re-adaptation, the system will again produce collision-free movements at the expense of less overall accuracy of movement.

This approach improves upon previous neural arm control schemes in six ways.

1. Collision avoidance constraints are learned which allow movement in the presence of obstacles. With an appropriate path planner this movement is *guaranteed* to be collision-free.
2. Only one initial unsupervised training phase is required to learn obstacle constraints for all possible numbers, sizes and configurations of obstacles. This means that if the obstacle set changes in any way, the robot does not have to relearn. The system learns about the robot geometry and its egocentric workspace, not about particular obstacle sets.
3. More than one target arm configuration is proposed for a given target point, increasing the likelihood of finding a successful trajectory in very cluttered workspaces.
4. 'Virtual' obstacles and other heuristics are not needed to avoid collisions.
5. The system handles target points outside of the robot's workspace very well. The arm simply reaches toward these points.
6. There is a well understood relationship between system precision and the number of neurons required in each network. The AAC can be made as precise as desired by increasing the number of network neurons.

The primary disadvantage of the approach is the rapid increase in the number of neurons needed to increase precision and add more degrees-of-freedom to the arm. Some simple strategies have been suggested to decrease the number of neurons required; but further research is needed to

find alternate architectures that require smaller networks. Optical technologies may hold the greatest promise for providing enough neurons and connections to control arms with high degrees of precision. Less accuracy is sufficient for some tasks, in which case it may be possible to implement enough neurons and inter-map connections using conventional VLSI technology.

Simulation results have been presented which demonstrate the performance of the system for arbitrary 2-dimensional robots with 2-degree-of-freedom manipulators.

Finally, we feel that the advantages of the AAC are important enough to warrant further research into collision-free control of general robot manipulators by similar unsupervised adaptive learning machines.

CHAPTER V: Directions for Further Research

Autonomous adaptive learning is a fundamental capability of any flexible, general purpose robot control system. We have described a neural network architecture for the adaptive, collision-free control of robot manipulators in order to demonstrate the utility of such architectures in the sensori-motor control domain. We have shown that this system has a number of advantages over traditional robot arm control techniques, and that it extends previous work on neural arm control systems. We have also shown that, although the system is *theoretically* capable of controlling arbitrarily complicated arms, it is limited by practical considerations to arms of two or three degrees of freedom, with accuracies of approximately 1.5%. Clearly, the system described here is a useful conceptual tool, but further research will be needed to make it a practical approach for industrial robot design.

This short chapter identifies directions for further research. It suggests a number of promising avenues of investigation that may lead to new approaches, improvements in accuracy and new capabilities. The ideas presented here not been formally analysed — they should be taken in the spirit of creative speculation.

5.1. Reducing the Number of AAC Neurons

The primary limitation of our approach is the exponential explosion of neurons needed to increase the control capability of the system. The size of the controller is determined by two factors

1. The number of degrees-of-freedom of the arm. If we wish the width of the network to be 'N' neurons, and we have D degrees-of-freedom, in the worst case we will require a total of N^D neurons.
2. The degree of precision required. The number of neurons required increases exponentially as positioning and avoidance errors are reduced.

The first factor is a potentially serious limitation. 6 degree-of-freedom robots are common in industry because this number of degrees-of-freedom are required to position an end-effector at a point in a 3-dimensional workspace with arbitrary orientation. How might we reduce the number of neurons required to construct a 6 degree-of-freedom arm? One possibility is to reduce the 6-dimensional arm configuration space to two 3 degree-of-freedom configuration spaces. To accomplish this, we divide the 6 degree-of-freedom arm into a 3 degree-of-freedom wrist attached to a 3 degree-of-freedom arm. Each of these 3 degree-of-freedom systems is controlled by a separate adaptive controller. The M_0 networks of these controllers represent the contents of their respective workspaces. The arm controller represents the wrist in M_0 as a solid object with the shape and dimension of the wrist's workspace. The M_0 view of the wrist controller represents

objects relative to the point of attachment with the arm. To touch an object, the TLC selects a target point offset from the object by the radius of the wrist workspace. After the arm moves the wrist into position, a second target point on the object surface visible to the wrist is selected by a wrist mounted sensor, and the wrist controller moves the hand into grasping position. This approach may reduce the possibility of finding a trajectory because the wrist must be modeled as a bulky object at the end of the arm to ensure that it does not collide with objects during movement regardless of its configuration. Other approaches which divide the arm into a series of joint pairs may also be possible. Such approaches would reduce the exponential neuron increase to a linear increase with the number of degrees-of-freedom.

The second factor relates accuracy to number of neurons. The heuristics described in section 4.6. can improve the accuracy of the system without increasing the number of neurons. Further research is required to simulate and refine these mechanisms:

1. Use distributed average positioning error statistics to prioritize path planning to target configurations. This technique reduces the overall average positioning error of the system.
2. Plan paths to configurations inhibited by avoidance error if all target configurations have been inhibited. This mechanism increases the success of the system even if avoidance error is large, relaxing the need for a large M_0 network.
3. Toroidal M_a and M_e network topologies help reduce positioning error variance for configurations near the edges of arm configuration space.
4. Arrange for the robot's tasks to occur primarily in front of its environmental sensors, not off to the side. This reduces the robot's positioning error.

As we have seen, only a few thousand neurons are required to implement an AAC with a positioning error of 5%. The most expedient approach may be to simply allow this much error, and add a simple, visually guided mechanism to move the end-effector the remaining 5% of the distance to target. A simple, cheap learning automata is described in [Andrade-Abalarach, 1988] that is a potential candidate for this mechanism. This system uses a measure of the distance between end-effector and target point to guide the end-effector to a target without performing inverse kinematic computations. This automata is incapable of obstacle avoidance, however, an AAC with 5% positioning error should be accurate enough to position the end-effector near enough to the target obstacle so that no obstacle is between it and the point of gaze. The automata is then free to move the arm the remaining distance without fear of collision. The resulting hybrid controller successfully combines the strengths of both approaches. It will be profitable to explore this and other mechanisms that refine AAC movement near a target.

5.2. Using Optical Technologies for Motor Control

Another way to increase AAC performance is to find an inexpensive technology that is capable of supporting very high densities of neurons and connections.

Current semi-conductor technology permits fabrication of inexpensive VLSI devices that contain millions of simple digital switching components. Research has begun into fabrication techniques that will allow high densities of analog components at the same cost [Graf, et. al., 1986; Hubbard, et. al., 1986; Mead, 1987]. Prototype chips have been built with hundreds of fully inter-connected neurons to perform various learning algorithms. Similar chips can in principle be manufactured to perform the Kohonen algorithm as well. Such chips would be sufficient to implement AAC's with low levels of precision. The primary limitation of existing VLSI techniques is the amount of area needed to lay out connections between neurons. It may be necessary to move to 3-dimensional substrates to implement very large networks.

Much more promising is the new generation of optical computing systems being designed specifically for neuro-computing applications [Anderson, 1986; Gindi, et. al., 1987; Gimitro & Gindi, 1987; Psaltis, et. al, 1987]. There is reason to believe that this technology may yield very high densities of fully connected, cheap, analog devices. For example, Demetri Psaltis and Yasir Abu-Mostafa [Abu-Mostafa & Psaltis, 1987] have developed a working Hopfield-style associative memory that contains 10,000 processors, and 100,000,000 connections. With further development, this approach may be a good candidate to provide the density of processors and connections required to directly implement a highly accurate adaptive arm controller for more than two degrees-of-freedom. Neurons are extremely cheap as they consist of dots of gallium arsenide on a silicon substrate. Light is reflected off the dots. The amplitude of the reflected light represents the activation level of the processor. The reflectance properties of the front surface of the dots changes depending on the total amount of light impinging on the rear surface. The activation level is controlled by passing the reflected beam of light through a transmission hologram that redirects the many reflected beams onto the back of the dots. Weights are represented by the intensities of the beams of light that leave the hologram. A one cm³ piece of photorefractive crystal could provide weights for a trillion connections. Research into optical self-organizing maps is necessary, as well as methods to adaptively alter the weight matrix hologram as learning proceeds.

5.3. Reconstructing Images from Touch Information

Suppose our robot reaches inside a box, or around a partition into a region of its workspace that it cannot directly observe. It is relatively straight-forward to enable the AAC to build up a rough image of the objects that are contained in this area by the sense of touch. The resulting image is the set of workspace volumes that the objects occupy — ie. a pixellated image of the objects.

Two additions to the AAC are needed to implement this ability: (1) connections must be grown between the M_0 and M_e networks, and (2) the M_0 network must be topographically organized in the same manner as the M_a and M_e networks. Hebbian connection growth similar to 4.4.1.2.1.1 can be used. During training, the volume of the workspace occupied by the point-of-touch activates a single M_0 neuron. As the EET tracks the end-effector, connections are grown between the M_e neuron that corresponds to foveating a target point, and the M_0 neuron that corresponds to the workspace volume that contains the target. Once training is complete, the robot is allowed to let its arm wander randomly about the unobservable region of the workspace. As the arm moves, a single M_a neuron will be activated by the CBUS competition. When the end effector encounters an object, touch sensors trigger the following flow of information:

1. The currently active M_a neuron sends an excitatory signal backwards, along the connections from M_a to M_e . This activates one or more M_e neurons that represent eye configurations foveating the current point-of-touch of the arm.
2. These neurons in turn send excitatory signals to neurons in the M_0 network that represent the volume of the workspace occupied by their point-of-gaze — ie. the current point-of-touch of the arm.

In this way, as the end-effector encounters objects, it 'fills-in' the volume of the workspace they occupy. The topographic ordering of M_0 makes it trivial to extract a 3-dimensional image of the objects from M_0 .

5.4. Reaching for Targets Outside the Workspace

When given a target point outside its workspace, the AAC generates a trajectory that causes the arm to reach as far as it can toward the target. This is a sensible — and seemingly intentional — response to an impossible task. It improves upon several other neural arm control schemes which can produce random target configurations under these circumstances [Elseley, 1988; Kuperstein, 1987]. It is desirable to have the AAC recognize when a target is outside its reach. It would also be useful to have the AAC learn to direct the propulsion system of the robot to move toward the target until it is within reach. How might we implement this behaviour?

Let us add another set of neurons that represent the direction of movement of the robot. For example we might have 360 neurons in a new network called M_p (for propulsion map), where each neuron represents a compass direction. If one of these neurons is activated, the robot's propulsion system moves the robot in the appropriate direction (very simple to arrange). We will allow connections to grow between M_e and M_p according to our Hebbian growth rules. Suppose that during training, a teacher causes the EET to track target points outside of the workspace. The teacher also activates the neuron that moves the robot toward the current target. As each such

training sample is generated, connections will grow between the M_e neuron that foveates an unreachable point-of-gaze and the M_p neuron that causes the robot to move toward the point. This process associates a direction of foveation with motor control signals that move the robot in this direction.

We also need an adaptive mechanism that the AAC can use to trigger movement when the target is out of reach. All the information needed to make this decision is available in the geometry of the $VT(M_e)$ tessellation. Notice from the example $VT(M_e)$ tessellation shown in Figure 4.55 that neurons with weight vectors bordering the region of workspace foveating eye configurations, generally have elongated Voronoi regions that extend into the regions of non-workspace targets and non-foveating configurations. These neurons represent configurations that are either at the edge of the workspace, or are completely outside the workspace. If a target outside of the workspace is foveated, it will activate one of these neurons. Suppose that during training, M_p neurons are only activated by the teacher when the target point is out of reach. In this case, M_p neurons will only grow connections to M_e neurons that foveate points outside of or near the edge of the workspace.

After training, when the TLC selects a target outside of — or at the limit of — the AAC's reach, the border neuron that wins the M_e competition will send an excitatory signal to the M_p neuron that, when activated, propels the robot toward the target. The AAC will also reach toward the target as the robot moves. movement will stop when the target is within easy reach and the propulsion system is no longer triggered. It is assumed that the TLC or the EET keeps the target foveated during movement. Although the movement of the arm will be collision-free, this process is too simplistic to produce movements of the robot's body that are also collision free. Further research is warranted. See [Lalonde & Graf 1988] for a discussion of how the techniques described by this thesis can be extended to other domains such as process control. Steering of autonomously piloted vehicles are a natural extension of the work presented here.

5.5. Adaptation to Velocity and Acceleration Kinematics

As we saw in chapter 3, there is more to robot control than position kinematics. The kinematics of velocity, and acceleration as well as statics and dynamics must also be accommodated. Further research is required to explore systems for adaptation to forces. A paper by Ritter and Schulten [Ritter & Schulten, 1986b] describes a neural learning system that employs a Kohonen self-organizing map to learn to apply the forces necessary to balance a broom on end. It may be possible to integrate similar techniques into the AAC architecture. A closer look is recommended.

References:

1. Abu-Mostafa, Y.S. and Psaltis, D., *Optical Neural Computers*. Scientific American, No. 256, Pgs. 88-95, March, 1987.
2. Ackley, D.H., A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers, 1987.
3. Ackley, D.H., Hinton, G.E. and Sejnowski, T.J., *A Learning Algorithm for Boltzmann Machines*. *Cognitive Sciences*, vol. 9, pp. 147-169.
4. Allen, P.K., Robotic Object Recognition Using Vision and Touch. Kluwer Academic Publishers, 1987.
5. Anderson, A., *Learning From a Computer Cat*. *Nature*, vol. 331, February 25, 1988, pp. 657-659.
6. Anderson, C.H., *The Bayes Connection*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 3, pp. 105-112.
7. Anderson, C.W., *Strategy Learning with Multilayer Connectionist Representations*.
8. Anderson, D.A., *Optical Resonators and Neural Networks*. AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986, pp. 12-17.
9. Anderson, J. and Rosenfeld, E., Neurocomputing: Foundations of Research. MIT Press, 1988.
10. Anderson, J.A., *Cognitive and Psychological Computation with Neural Models*. IEEE, Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 5, September/October, 1983, pp. 799-815.
11. Andrade-Abalarach, M.N., *Some Learning Applications for Robot Manipulators*. Carleton University School of Computer Science, Masters Degree Thesis, 1988.
12. Arbib, M.A. and House, D.H., *Depth and Detours: an Essay on Visually Guided Behaviour*. in Vision, Brain and Cooperative Computation, Arbib, A., Hanson, A.R., Eds., MIT Press, 1987, pp. 127-163.
13. Baron, R.J., The Cerebral Computer. Lawrence Erlbaum Associates, Inc., 1987.
14. Barto, A.G., *An Approach to Learning Control Surfaces by Connectionist Systems*. in Vision, Brain and Cooperative Computation, Arbib, A., Hanson, A.R., Eds., MIT Press, 1987, pp. 665-701.
15. Baum, E.B., *Generalizing Back Propagation to Computation*. AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986, pp. 47-53.

4 of /de 4

16. Bear, M.F., Cooper, L.N. and Ebner, F.F., *A physiological Basis for a Theory of Synapse Modification*. Science, vol. 237, 1987, pp. 42-48.
17. Braitenberg, V., Vehicles - Experiments in Synthetic Psychology. The MIT Press, Cambridge, Massachusetts, 1984.
18. Bridle, J.S. and Moore, R.K., *Boltzmann Machines for Speech Pattern Processing*. Proceedings of The Institute of Acoustics, vol. 6, part 4, 1984, pp. 315-322.
19. Brooks, R.A., *Planning Collision-Free Motions for Pick-And-Place Operations*. The International Journal of Robotics Research, vol. 2, no. 4, Winter 1983.
20. Bullock, D., *Neural Networks for Adaptive Robotics*. Notes for a tutorial given at the IEEE First Annual International Conference on Neural Networks, San Diego, California, June 21-24, 1987.
21. Carpenter, G.A. and Grossberg, S., *A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine*, Computer Vision, Graphics, and Image Processing, vol. 37, 1987, pp. 54-115.
22. Charniak, E. and McDermott, D. Introduction to Artificial Intelligence. Addison-Wesley Publishing Company, 1985.
23. Charniak, E., Riesbeck, C.K., McDermott, D. and Meehan, J.R., Artificial Intelligence Programming. Lawrence Erlbaum Associates, Inc., 1987.
24. Cohen, M.A., *Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks*. IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-13, no. 5, September/November, 1983, pp. 815-826.
25. Cottrell, G.W. and Small, D.L., *A Connectionist Scheme for Modelling Word Sense Disambiguation*. TR 122, Computer Science Department, University of Rochester; Cognition and Brain Theory 6, 1, 1983, pp. 88-120.
26. Craig, J.J., Adaptive Control of Mechanical Manipulators. Addison-Wesley Publishing Company, Inc., 1988.
27. Crick, F.H.C. and Asanuma, C., *Certain Aspects of the Anatomy and Physiology of the Cerebral Cortex*. in Parallel Distributed Processing: Volume 2, McClelland, J.L., et. al. Eds., MIT Press, 1986.
28. Crick, F.H.C., *The Function of the Thalamic Reticular Complex: The Search-Light Hypothesis*. Proceedings of the National Academy of Sciences, vol. 81, 1984, pp. 4586-4590.
29. Davis, R. and Lenat, D.B., Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill Advanced Computer Science Series, McGraw-Hill International Book Company, 1982.
30. Duda, R.O. and Hart, P.E., Pattern Classification and Scene Analysis. Wiley-Interscience, John Wiley and Sons, Inc., 1973.

31. Ellias, S.A. and Grossberg, S., *Pattern Formation, Contrast Control, and Oscillations in the Short Term Memory of Shunting On-Center Off-Surround Networks*. Biological Cybernetics, 1985, vol. 52, pp. 141-152.
32. Elsley, R.K., *A Learning Architecture for Control Based on Backpropagation Neural Networks*. Proceedings of the Second IEEE International Conference on Neural Networks, San Diego, California, July 24-27, 1988, vol. 2, pp. 587-594.
33. Elsley, R.K., *A Learning Architecture for Control Based on Backpropagation Neural Networks*. Proceedings of the Second IEEE International Conference on Neural Networks, San Diego, California, July 24-27, 1988, vol. 2, pp. 587-594.
34. Erdi, P. and Barna, Gy., *Self-Organizing Mechanism for the Formation of Ordered Neural Mappings*. Biological Cybernetics, 51, 1984, pp. 93-101.
35. Frisby, J.P., *Seeing: Illusion, Brain and Mind*. Oxford University Press, 1980.
36. Fukushima, K. and Miyake, S., *A Neural Network Model for the Mechanism of Feature-Extraction*. Biological Cybernetics, 50, 1984, pp. 377-384.
37. Fukushima, K., *A Hierarchical Neural Network Model for Associative Memory*. Biological Cybernetics, 50, 1984, pp. 105-113.
38. Fukushima, K., *A Neural Network Model for Selective Attention in Visual Pattern Recognition*. Biological Cybernetics, 55, 1986, pp. 5-15.
39. Fukushima, K., *Cognitron: A Self-Organizing Multilayered Neural Network*. Biological Cybernetics, 20, 1975, pp. 121-136.
40. Fukushima, K., Miyake, S. and Ito, T., *Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition*. IEEE Transactions on Man, Machines and Cybernetics, Vol. SMC-13, No. 5, September/October, 1983, pp. 826-834.
41. Gallant, S.I., *Automatic Generation of Expert Systems from Examples*. Proceedings of the Second Conference on Artificial Intelligence Applications, Miami Beach, Florida, December 11-13, 1985, pp. 313-319.
42. Gallant, S.I., *Connectionist Expert Systems*. To Appear in the Communications of the ACM, November 5, 1987.
43. Geschwind, N., *Specializations of the Human Brain*. Scientific American, No. , Pgs ,
44. Gindi, G., Gmitro, A. and Parthasarathy, K., *Winner-Take-All Networks and Associative Memory: Analysis and Optical Realization*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 3, pp. 607-614.
45. Gindi, G.R., Gmitro, A.F. and Parthasarathy, K., *Winner-Take-All Networks and Associative Memory: Analysis and Optical Realization*. Proceedings of the First IEEE

- International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 3, pp. 607-614.
46. Gmitro, A., Gindi, G., *Optical Neurocomputer for Implementation of the Marr-Poggio Stereo Algorithm*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 3, pp. 599-606.
 47. Gmitro, A.F. and Gindi, G.R., *Optical Neurocomputer for Implementation of the Marr-Poggio Stereo Algorithm*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 3, pp. 599-606.
 48. Goldberg, A. and Robson, D., Smalltalk-80: The Language and Its Implementation. Addison-Wesley, 1983.
 49. Graf, D.H., *NARM: The Design of A Neural Robot Arm Controller*. School of Computer Science Technical Report SCS-TR-133, Carleton University, April, 1988a.
 50. Graf, D.H., *NARM-SIM: A Neural Network Simulator for Adaptive, Collision-Free Robot Manipulator Control*. School of Computer Science Technical Report SCS-TR-141, Carleton University, September, 1988b.
 51. Graf, D.H. and LaLonde, W.R., *A Neural Controller For Collision-Free Movement of General Robot Manipulators*. Proceedings of the Second IEEE International Conference on Neural Networks, San Diego, California, July 24-27, 1988, vol. 1, pp. 77-84.
 52. Graf, H.P., Jackel, L.d., Howard, R.E., Straughn, B., Denker, J., Hubbard, W., Tennant, D., Schwartz, D., *VLSI Implementation of a Neural Network Memory With Several Hundreds of Neurons*. AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986, pp. 182-187.
 53. Grossberg, S., *Adaptive Resonance in Development, Perception and Cognition*. SIAM-AMS Proceedings, vol. 13, 1981, pp. 107-156.
 54. Grossberg, S., *How Does a Brain Build a Cognitive Code?*. Psychological Review, vol. 87, no. 1, January, 1980, pp. 1-51.
 55. Grossberg, S., The Adaptive Brain I - Cognition, Learning, Reinforcement, and Rhythm. North-Holland, Elsevier Science Publishing Company, Inc., 1987.
 56. Grossberg, S., The Adaptive Brain II - Vision, Speech, Language, and Motor Control. North-Holland, Elsevier Science Publishing Company, Inc., 1987.
 57. Grossberg, S., *The Adaptive Self-Organization of Serial Order in Behaviour: Speech, Language, and Motor Control*. in Pattern Recognition by Humans and Machines, Academic Press, Inc., 1986, pp. 187-295.
 58. Grossman, S.I., Calculus. Academic Press, Inc., New York, 1977.

59. Harth, E., Unnikrishnan, K.P. and Pandya, A.S., *The Inversion of Sensory Processing by Feedback Pathways: A Model of Visual Cognitive Functions*. Science, vol. 237, pp. 184-187.
60. Hecht-Nielsen, R., *Kolmogorov's Mapping Neural Network Existence Theorem*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 3, pp. 11-14.
61. Hinton, G.E. and Sejnowski, T.J., *Learning and Relearning in Boltzmann Machines*. in Parallel Distributed Processing: Volume 1, McClelland, J.L., et. al. Eds., MIT Press, 1986a.
62. Hinton, G.E. and Sejnowski, T.J., *Learning in Boltzmann Machines*. Cognitiva 85, Paris, June 4-7, 1985.
63. Hinton, G.E., McClelland, J.L. and Rumelhart, D.E., *Distributed Representations*. in Parallel Distributed Processing: Volume 1, McClelland, J.L., et. al. Eds., MIT Press, 1986b.
64. Hirsch, M., *Convergence in Neural Nets*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987; vol. 2, pp. 115-126.
65. Hirsch, M., *Dynamical Systems Review*. Notes for a tutorial given at the IEEE Second Annual International Conference on Neural Networks, San Diego, California, July 24-27, 1988.
66. Hodder, I. and Orton, C., Spatial Analysis in Archaeology. Cambridge University Press, Cambridge, England, 1976.
67. Hopfield, J.J. and Tank, D.W., *Neural Computation of Decisions in Optimization Problems*. Biological Cybernetics, 1985, vol. 52, pp. 141-152.
68. Hopfield, J.J., *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*. Proceedings of the National Academy of Sciences, USA, Vol. 79, April, 1982, pp. 2554-2558.
69. Hubbard, W., Schwartz, D., Denker, J., Graf, H.P., Howard, R.E., Jackel, L., Straughn, B. and Tennant, D., *Electronic Neural Networks*. AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986, pp. 227-235.
70. Hubel, D.H. and Wiesel, T.N., *Brain Mechanisms of Vision*. Scientific American, No. , Pgs. , 1986.
71. Jourdan, M.I., *An Introduction to Linear Algebra in Parallel Distributed Processing*. in Parallel Distributed Processing: Volume 1, McClelland, J.L., et. al. Eds., MIT Press, 1986.
72. Judd, S., *Learning in Networks is Hard*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 2, pp. 685-692.

73. Kirkpatrick, S., Gelatt, D.D. and Vecchi, M.P., *Optimization by Simulated Annealing*. Science, vol. 220, 1983, pp. 671-680.
74. Kohonen, T., *Analysis of a Simple Self-Organizing Process*. Biological Cybernetics, vol. 44, 1982, pp. 135-140.
75. Kohonen, T., *Clustering, Taxonomy, and Topological Maps of Patterns*. Proceedings of the Sixth International Conference on Pattern Recognition. Munich, Germany, 1982, pp. 114-128.
76. Kohonen, T., *Learning Vector Quantization For Pattern Recognition*. Helsinki University of Technology, Department of Technical Physics, SF-02150 Espoo, Finland, Report TKK-F-A601, November, 1986.
77. Kohonen, T., Makisara, K. and Saramaki, T., *Phonotopic Maps — Insightful Representation of Phonological Features for Speech Recognition*. Proceedings of the Seventh International Conference on Pattern Recognition. Montreal, Canada, 1984, pp. 182-185.
78. Kohonen, T., *Representation of Sensory Information in Self-Organizing Feature Maps* AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986, pp. 271-277.
79. Kohonen, T., Reuhkala, K., Makisara, K. and Vainio, L., *Associative Recall of Images*. Biological Cybernetics, vol. 22, 1976, pp. 159-168.
80. Kohonen, T., Rittinen, H., Reuhkala, E. and Haltsonen, S., *On-Line Recognition of Words From a Spoken Vocabulary*. Information Sciences, vol. 33, 1984, pp. 3-30.
81. Kohonen, T., *Self-Organization and Associative Memory*. Berlin-Heidelberg-New York-Tokyo: Springer-Verlag, 1984.
82. Kohonen, T., *Self-Organizing Maps*. Notes for a tutorial given at the IEEE First Annual International Conference on Neural Networks, San Diego, California, June 21-24, 1987.
83. Kosko, B., *Adaptive Inference in Fuzzy Knowledge Networks*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1988, vol. 2, pp. 261-271.
84. Kosko, B., *Bidirectional Associative Memories*. IEEE, Transactions on Systems, Man, and Cybernetics, vol. 18, no. 1, January/February, 1988, pp. 49-60.
85. Kronauer, R.E. and Zeevi, Y.Y., *Reorganization and Diversification of Signals in Vision*. IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 1, January/February, 1985, pp. 91-101.
86. Kuperstein, M., *Adaptive Visual-Motor Coordination in Multijoint Robots Using Parallel Architecture*. IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March, 1987, pp. 1595-1602.

87. Lalonde, W.R. and Graf, D.H., *Neuroplanners: Mechanisms for Subcognitive Control*. Submitted for publication - Proceedings of the Neuro-Nimes 88 Conference, Nimes, France, November 15, 1988.
88. Lara, R. and Arbib, M.A., *A Model of the Neural Mechanisms Responsible for Pattern Recognition and Stimulus Specific Habituation in Toads*. Biological Cybernetics, vol. 51, 1985, pp. 223-237.
89. Le Cun, Y., Proceedings Cognitiva 85, 1985, pp. 599-604.
90. Lippmann, R.P., *An Introduction to Computing with Neural Nets*. IEEE ASSP Magazine, April, 1987, pp. 4-20.
91. Livingstone, M.S., *Art, Illusion and the Visual System*. Scientific American, No. 258, Pgs. 78-85, January, 1988.
92. Lozano-Perez, T., *A Simple Motion-Planning Algorithm for General Robot Manipulators*. IEEE Journal of Robotics and Automation, Vol. RA-3, No. 3, June 1987, pp. 224-238.
93. Marr, D. and Poggio, T., *Co-operative Computation of Stereo Disparity*. Science, vol. 194, 1976, pp. 283-287.
94. McClelland, J.L., Rumelhart, D.E. and Hinton, G.E., *The Appeal of Parallel Distributed Processing*. in Parallel Distributed Processing: Volume 1, McClelland, J.L., et. al. Eds., MIT Press, 1986.
95. Mead, C., *Silicon Models of Neural Computation*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 1, pp. 91-106.
96. Michalski, R.S. and Chilausky, R.L., *Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis*. International Journal of Policy Analysis and Information Systems, vol. 4, no. 2, 1980.
97. Michalski, R.S., Carbonell, J.G., Mitchell, T.M., Machine Learning (Volumes I and II). Morgan Kaufmann Publishers, Inc., 1986.
98. Miller, R. and Stout, Q.F., *Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 2, March, 1985, pp. 216-228.
99. Minsky, M.L. and Papert, S.A., Perceptrons - An Introduction to Computational Geometry (Expanded Edition). The MIT Press, Cambridge, Massachusetts, 1969.
100. Minsky, M.L., The Society of Mind. Simon and Schuster, New York, 1986.
101. Mozer, M.C., *RAMBOT: A Connectionist Expert System that Learns by Example*. ICS Report 8610, Institute for Cognitive Science, University of California, San Diego, La Jolla, California, August, 1986.

102. Nagano, T. and Miyajima, S., *A Neural Network Model for the Developmental Process of Hypercomplex Cells*. IEEE, Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 5, September/October, 1983, pp. 847-851.
103. Neter, J., Wasserman, W. and Whitmore, G.A., Applied Statistics. Allyn and Bacon, Inc., Boston, Massachusetts, 1978.
104. Nilsson, N.J., Learning Machines - Foundations of Trainable Pattern-Classifying Systems. McGraw-Hill, Inc., 1965.
105. Noback, C.R. and Demarest, R.J., The Human Nervous System: Basic Principles of Neurobiology. McGraw-Hill, Inc., 1981.
106. Oommen, B.J., *Notes for a First Course in Robotics and Computer Vision*. School of Computer Science, Carleton University, Canada, 1986.
107. Overton, K.J., *Robot Tactile Sensing and Schema Control*. in Vision, Brain and Cooperative Computation, Arbib, A., Hanson, A.R., Eds., MIT Press, 1987, pp. 511-524.
108. Page, N., Snyder, W.E. and Rajala, S.A. *Turbine Blade Image Processing System*. First International Conference on Advanced Software in Robotics, Liege, Belgium, May, 1983.
109. Palm, G., Bonhoeffer, T., *Parallel Processing for Associative and Neuronal Networks*. Biological Cybernetics, 51, pp. 201-204.
110. Palm, G., Cell Assemblies: A New Approach to Artificial Intelligence. Springer-Verlag, 1984.
111. Parker, D., *Learning-Logic*, TR-47, Centre for Computational Research in Economics and Management Science, MIT, April, 1985.
112. Petersen, S.E., Fox, P.T., Posner, M.I., Mintun, M. and Raichle, M.E., *Positron Emission Tomographic Studies of the Cortical Anatomy of Single-Word Processing*, Nature, vol. 331, February 18, 1988, pp. 585-589.
113. Pielou, E., Mathematical Ecology. Wiley-Interscience, New York, 1977.
114. Porter, G.B. and Mundy, J.L. *A Noncontact Profile Sensing System for Visual Inspection*. Sixth International Conference on Pattern Recognition, Munich, October, 1982.
115. Preparata, F.P. and Shamos, M.I. Computational Geometry - An Introduction. Springer-Verlag New York ZInc, 1985.
116. Psaltis, D., Sideris, A. and Yamamura, A., *Neural Controllers*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987b, vol. 4, pp. 551-558.
117. Psaltis, D., Wagner, K. and Brady, D., *Learning in Optical Neural Computers*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987a, vol. 3, pp. 549-556.

118. Ritter, H. and Schulten, K., *On the Stationary State of Kohonen's Self-Organizing Sensory Mapping*. Biological Cybernetics, vol. 54, 1986, pp. 99-106.
119. Ritter, H. and Schulten, K., *Topology Conserving Mappings for Learning Motor Tasks*. AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird, Utah, AIP, 1986, pp. 377-381.
120. Roberts, P.A., Neuroanatomy. Springer-Verlag, Inc., 1987.
121. Rosenblatt, F., Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Washington, D.C.: Spartan Books, 1961.
122. Rosenblatt, F., *The Perceptron — a Perceiving and Recognizing Automaton*. Report 85-460-1, Cornell Aeronautical Laboratory, Ithica, New York, January, 1957.
123. Rumelhart, D., *Parallel Distributed Processing*. Notes for a tutorial given at the IEEE Second Annual International Conference on Neural Networks, San Diego, California, July 24-27, 1988.
124. Rumelhart, D.E. and McClelland, J.L., *Learning the Past Tenses of English Verbs: Implicit Rules or Parallel Distributed Processing?* in Mechanisms of Language Aquisition, MacWhinney, B. Ed., Erlbaum, Hillsdale, New Jersey, 1987.
125. Rumelhart, D.E. and Zipser, D., *Feature Discovery by Competitive Learning*. in Parallel Distributed Processing: Volume 1, McClelland, J.L., et. al. Eds., MIT Press, 1986c.
126. Rumelhart, D.E., Hinton, G.E. and Williams, R.J., *Learning Representations by Back Propagation*. Nature, vol. 323, October 9, 1986a, 533-536.
127. Rumelhart, D.E., McClelland, J.L. and the PDP Research Group, Parallel Distributed Processing - Explorations in the Microstructure of Cognition (Volumes I and II). The MIT Press, Cambridge, Massachusetts, 1986b.
128. Sejnowski, T.J. and Hinton, G.E., *Seperating Figure from Ground with a Boltzman Machine*. in Vision. Brain and Cooperative Computation, Arbib, A., Hanson, A.R., Eds., MIT Press, 1987, pp. 703-724.
129. Sejnowski, T.J. and Rosenberg, C.R., *NETalk: A Parallel Network that Learns to Read Aloud*. The Johns Hopkins University Elecrical Engineering and Computer Science Technical Report JHU/EECS-86/01, 1986.
130. Smolensky, P., *Neural and Conceptual Interpretation of PDP Models*. in Parallel Distributed Processing: Volume 2, McClelland, J.L., et. al. Eds., MIT Press, 1986.
131. Snyder, W.E., Industrial Robots: Computer Interfacing and Control. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.
132. Sparks, D.L. and Jay, M., *The Role of the Primate Superior Colliculus in Sensori-Motor Integration*. in Vision. Brain and Cooperative Computation, Arbib, A., Hanson, A.R., Eds., MIT Press, 1987, pp. 109-128.

133. Staugaard, A.C., Jr., Robotics and AI: an Introduction to Applied Machine Intelligence. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.
134. Stone, G.O., *An Analysis of the Delta Rule and the Learning of Statistical Associations*, in Parallel Distributed Processing: Volume 1, McClelland, J.L., et. al. Eds., MIT Press, 1986.
135. Strang, G. Linear Algebra and its Applications. New York: Academic Press, 1976.
136. Touretsky, D.S., *Representing Conceptual Structures in a Neural Network*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 2, pp. 279-285.
137. Touretzky, D.S. and Hinton, G.E., *Symbols Among the Neurons: Details of a Connectionist Inference Architecture*. IJCAI Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, California, August 18-23, 1985, vol. 1, pp. 238-243.
138. Townsend, M., Discrete Mathematics: Applied Combinatorics and Graph Theory. The Benjamin/Cummings Publishing Company, 1987.
139. Treurniet, W.C. and Croft, C.M., *Problem Solving and Concept Learning with a Distributed Memory*. Technical Memorandum TS0063/86, Directorate of Behavioural Research, Department of Communications, Government of Canada, April 11, 1986.
140. Tsutsumi, K. and Matsumoto, H., *Neural Computation and Learning Strategy for Manipulator Position Control*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 4, pp. 525-534.
141. Tsutsumi, K., Katayama, K. and Matsumoto, H., *Neural Computation for Controlling the Configuration of a 2-Dimensional Truss Structure*. Proceedings of the Second IEEE International Conference on Neural Networks, San Diego, California, July 24-27, 1988, vol. 2, pp. 575-586.
142. von der Malsberg, C., *Self-Organization of Orientation Sensitive Cells in the Striate Cortex*. Kybernetik, vol. 14, 1973, pp. 85-100.
143. Watrous, R.L. and Shastri, L., *Learning Phonetic Features using Connectionist Networks: An Experiment in Speech Recognition*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987a, vol. 4, pp. 381-388.
144. Whitney, D.E., *Neuronal Coding and Robotics*. Technical Comments, Science, vol. 237, 1987, 300-301.
145. Widrow, B., Adaptive Signal Processing. Prentice-Hall, 1985.
146. Widrow, B., Winter, R.G. and Baxter, R.A., *Learning Phenomena in Layered Neural Networks*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. 2, pp. 411-430.

147. Winston, P.H., Artificial Intelligence (Second Edition). Addison Wesley Publishing Company, Inc., 1984.
148. Yang, H. and Guest, C., *Performance of Backpropagation for Rotation Invariant Pattern Recognition*. Proceedings of the First IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987a, vol. 4, pp. 365-370.
149. Zipser, D., *Biologically Plausible Models of Place Recognition and Goal Location*. in Parallel Distributed Processing: Volume 2, McClelland, J.L., et. al. Eds., MIT Press, 1986.
150. Zipser, D., *Representing Cognitive Maps in Parallel Networks*. Proceedings of the Cognitive Science Society, Boulder, 1984.
151. Zipser, D. and Anderson, R.A., *A Back-Propagation Programmed Network that Simulates Response Properties of a Subset of Posterior Parietal Neurons*. Nature, vol. 331, February 25, 1988, pp. 679-684.

END

06-07-89

FIN