Self Organizing Maps Constrained by Data Structures

By

César A. Astudillo

A thesis proposal submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

April 2011

Your file Votre référence
ISBN: 978-0-494-81544-1
Our file Notre référence
ISBN: 978-0-494-81544-1

The undersigned hereby recommend to
the Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis,

Self Organizing Maps Constrained by Data Structures

submitted by

César A. Astudillo

Dr. Douglas Howe
(Director, School of Computer Science)

Dr. B. John Oommen
(Thesis Supervisor)

Dr. Mohamad Musavi
(External Examiner)

Carleton University
April 2011

Abstract

Within the field of Pattern Recognition (PR) and Machine Intelligence (MI), when one requires useful information from a set of stimuli, the task usually demands the deduction of its structure and stochastic distribution. This endeavor becomes especially challenging when the stimuli belongs to a higher dimensional domain, and its cardinality is large. Through the last few decades, researchers have tried to solve this problem and have faced numerous difficulties, particularly when the learning process is performed without the intervention of a human being. The state-of-the-art records remarkable efforts in the field of Artificial Neural Networks (ANNs) that follow the latter paradigm. Among the set of ANNs, the Self-Organizing Map (SOM), pioneered by Kohonen, is unique due to its interesting theoretical capabilities – which have profound practical significance. However, it is known that under various circumstances, the SOM fails to represent the data accurately.

This thesis presents new families of self-organizing ANNs. They have been designed with the goal of overcoming some of the reported handicaps of the SOM. First of all, the thesis contains a complete survey of the field that pertains to SOMs, which is a contribution to the community in its own right. We then propose a method by which a user-defined tree automatically adapts so as to absorb the essential properties of the stimuli, while it, simultaneously, preserves the original properties of the feature space. The resultant tree reveals multi-resolution capabilities, which are helpful for representing the original data set with different numbers of points. These desirable properties are advantageously utilized to derive classifiers that are capable of learning from labeled and unlabeled samples simultaneously, and the PR implications are demonstrated by a rigorous set of experiments. The thesis thereafter contains a pioneering attempt to merge the areas of the ANNs with the theory of Adaptive Data Structures (ADSs). This is accomplished by considering how the underlying tree *itself* can be rendered dynamic and adaptively transformed. Again, the PR implications of this are also explored. Finally, the thesis also incorporates a hyperplane-based partitioning scheme to accelerate the time required to identify the winner neuron, which is a process that is central to *any* SOM-based strategy.

Acknowledgements

The present Thesis corresponds to the final milestone of a long journey towards obtaining my doctoral degree in Computer Science It is fair to state that this achievement could not be possible without the support of many people Inasmuch as we are human beings more than students or researchers, this document will not be complete without a token of gratitude

First of all, I would like to express my profound respect for my supervisor, Professor John Oommen, who has worked shoulder to shoulder with me in the preparation of this manuscript Frankly speaking, the outstanding intellectual capabilities of John are only comparable to the quality of his spirit He has not only been a guide in the development of the Thesis, but also an advisor in the multi-faceted aspects of life He taught me how to write academic papers, and was able to draw out my research strengths He was also very demanding when it concerned preparing the corresponding papers and chapters His enthusiasm and discipline where key for achieving good quality research results from the original ideas that we had developed

Secondly, but not less importantly, is the fundamental role played by Andrea, my beloved wife I feel very lucky to be with such a strong woman She certainly gave me strength, and helped me to find solutions when the situations were adverse Both our children, Gabriel and Rafael, were also her gift to me Without any doubt, her complementary skills made us an outstanding team!

The financial support for my doctoral studies was partially provided by the Universidad de Talca, Chile This institution is also the university where I am currently starting my career as a professor At this university, there are many who have generously helped me I particularly want to record my thanks to Alfredo Candia, who was pivotal to my decision of staying in the academic world I want to also thank Federico Meza, Edgardo Padilla, Alvaro Rojas, Ruth Garrido, Marcela Pacheco, Matt Bardeen, Rodolfo Allendes, Rodrigo Paredes and Pablo Rojas, among many others, who have helped me individually and together in different issues that arose during my doctoral studies

I am grateful to some of my lab mates, with whom I had spent innumerable hours, during the day and often during night, discussing some of the details of our respective research topics Those conversations served to improve our thoughts about our own work, and also helped to better understand some of the theoretical concepts reported in the literature In this regard, I specially thank Amir-Ali Salehi, Colin Bellinger and Ebaa Fayyoumi They are also my friends – a spontaneous friendship that resulted from the conversations

There are many other persons who, in my opinion, have contributed to this endeavor To mention a few, I would like to thank the inner circles of my family, and in particular, my father Jose "Pepe" Astudillo, my mother Ana Hernández, my brothers Claudio and Alonso, and also Rolando Montoya and Magleny Macias

These acknowledgements will not be complete if I did not include my gratitude to Leopoldo "Leo" Bertossi, Eduardo "Lalo" Troncoso, Fatima "Fafy" da Silva, Pablo Garrido, Joyce Portilla, Daryl Foster, Marta Espinoza, Hector Vera, Gloria Romero, Joe Skuce, Carolina Bórquez, Alan Davoust, Sylviana "Sylvi" Geoffray, Yannick Marcérou, Anna Bogic, Eugenio Ortega, Mauricio Vines, Gerardo Reynaga, Claire Ryan, Edina Sandler, Sandy Herbert, Anna Riethman, Sharmila Namasivayampillai, Dragos Calitoiu, Natalia Villanueva, and many others, including the members of my final Thesis committee

Finally, I am also grateful to the *Comisión Nacional de Investigación Científica y Tecnológica (CONICYT)* and to *NSERC* for partially financing the research aspects of this Thesis

Contents

List of Figures

Acronyms

ABSTSOM Adaptive BSTSOM

ADS Adaptive Data Structure

AI Artificial Intelligence

ART Adaptive Resonance Theory

ANN Artificial Neural Network

AUC Area Under the ROC Curve

BMU Best Matching Unit

BN Bayesian Network

BoA Bubble of Activity

BST Binary Search Tree

BSTSOM Binary Search Tree SOM

CL Competitive Learning

CM Confusion Matrix

CONROT-BST Conditional Rotations for a BST

CONROT Conditional Rotations

CPT Conditional Probabilities Table

CS Computer Science

DAG Directed Acyclic Graph

DS Data Structure

DT D-Tree

ET Evolving Tree

GCS Growing Cell Structures

GG Growing Grid

GHSOM Growing Hierarchical SOM

GNG Growing Neural Gas

GT Growth Threshold

GSOM Growing SOM

HFM Hierarchical Feature Map

HST Hyperplane Search Tree

IB Instance-Based

IID Independently and Identically Distributed

IGG Incremental Grid Growing

IR Information Retrieval

JPD Joint Probability Distribution

k-**NN** k-Nearest Neighbor

k-**CONROT** CONROT for k-ary trees

LVQ Learning Vector Quantization

MAP *Maximum A Posteriori*

ML Machine Learning

MQE Mean Quantization Error

MST Minimum Spanning Tree

MT Monotonic Tree

NB Naive Bayes

NG Neural Gas

NN Neural Network

PCA Principal Component Analysis

PM Pattern Matching

PR Pattern Recognition

QE Quantization Error

RHST Random Hyperplane Search Tree

ROC Receiver Operating Characteristics

SOM Self-Organizing Map

SOTA Self-Organizing Tree Algorithm

SOTM Self-Organizing Tree Map

SVM Support Vector Machine

TOD Threshold Order-Dependent

TSVQ Tree-Structured VQ

TSSOM Tree-Structured SOM

TTOCONROT TTOSOM with Conditional Rotations

TTOSOM Tree-based Topology Oriented SOM

URL Uniform Resource Locator

VQ Vector Quantization

WDBC Wisconsin Diagnostic Breast Cancer

WPL Weighted Path Length

Notation

Variables, Symbols, and Operations

$a \leftarrow a + 1$	The term $a + 1$ is assigned to the variable a
$\arg\min_x f(x)$	The value of x leading to the minimum value of $f(x)$

Mathematical Operations

$\bar{x}$	Mean or average value of x
$\sum_{k=1}^{n} a_k$	The sum of the values $a_1, a_2, \quad , a_n$

Vectors and Matrices

$\mathbf{x}$	A column vector utilizes boldface
$\mathbf{A}$	Matrices utilizes boldface
$\mathbb{R}^d$	d-dimensional euclidean space
$\mathbf{x}^t$	Transpose of the vector $\mathbf{x}$
$\|\mathbf{x}\|$	Euclidean norm of the vector $\mathbf{x}$

Sets

$\mathbb{R}$	The set of real numbers
$\mathbb{N}$	The set of natural numbers
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D},$	A "calligraphic" font is utilized for sets or lists
$\mathbf{x} \in \mathcal{X}$	$\mathbf{x}$ is in the set $\mathcal{X}$
$\mathbf{x} \notin \mathcal{X}$	$\mathbf{x}$ is not an element of set $\mathcal{X}$
$\|\mathcal{X}\|$	The number of elements contained in the set $\mathcal{X}$

Probability, Distributions and Complexity

ω	State of nature
$p(\ )$	Probability density function
$\mathbf{w}$	Weight vector
$O(h(x))$	Big oh order of $h(x)$

Chapter 1

Introduction

1.1 Introduction

Recognizing regularity in patterns is a fundamental problem, of great importance to humans. Our early ancestors had to differentiate between fruits that were ready for harvest from those that were still immature. Of course, the mature fruits tasted better, and were also probably better for health. The difficulty probably lay in recognizing the difference between the fruits by merely *looking* at them.

Admittedly, human beings are very good at recognizing and classifying objects, as in the above example. Researchers have been fascinated by this and have tried to imitate these amazing capabilities of the brain. However, to quote Socrates "As for me, all I know is that I know nothing", as we learn more about *how* to recognize patterns, we also discover more problems associated with the recognition process itself.

Research in Computer Science (CS) has progressed to such an extent that computers today can recognize fingerprints, speech, speakers and faces. In spite of this, we constantly encounter the envelope of our knowledge as we investigate deeper intricacies of the field.

Coincidentally, it has been almost exactly a century, since the horizon of neuroscience was expanded, when Ramón y Cajal [155] explained, for the first time, the complexities of the brain which worked through the interconnection of much-simpler

processing units that he named *neurons* Transferring those initial ideas into the field of CS has led to the development of computational models that invoke the so-called concept of connectionism A particular architecture within this family is the one that leads to the framework of Artificial Neural Networks (ANNs)

From a different point of view, an essential part of almost any algorithm is to choose or design efficient ways to store and organize the data Most of the time, these so-called Data Structures (DSs) are defined in conjunction with a set of procedures that facilitate their usage, while the study of the corresponding efficiencies is frequently linked to those operations Our interest is to focus on a particular subbranch of DSs, namely those which are adaptive Adaptive Data Structures (ADSs) attempt to automatically adapt the configuration of the DS so as to lead to an optimal configuration

The overall goal of this thesis is to merge the the fields of ANNs and ADSs We are aware of no related work that accomplishes this

Further, in our investigation, we also intend to take a closer inspection on various hierarchical sub-divisions of the space in which the characterizing features of the patterns are located

The thesis will attempt to merge these fields so as to emphasize the functional relationship between these different branches of CS from a holistic perspective Our hope is that by venturing on this path, we can glean improved solutions to problems involving the Pattern Recognition (PR) of objects Obviously, as these problems are usually dependent on numerous factors, the processes that we study and the solutions that we propose will vary depending on a variety of criteria Our investigation will also involve the latter criteria

Acquiring information about a set of stimuli in spaces with high dimensionality, usually demands the deduction of its structure While studying these task, we have encountered numerous difficulties, especially in the case when the learning process is performed without the intervention of human beings The state-of-the-art records remarkable efforts in the field of ANNs that follow the latter paradigm, including methods that allow data clustering and visualization in high-dimensional spaces Among

the subsets of theses schemes, the Self-Organizing Map (SOM), pioneered by Kohonen, has attained to a special place "on the podium" due to its amazing theoretical and practical properties The handicap, however, is that it is known that under various circumstances, the SOM fails to represent the data accurately This being the case, our research has motivated us to consider strategies that can overcome these handicaps

In general, the *topology* employed by any ANN that possesses the above-mentioned abilities, has an important impact on the manner by which it will "absorb" and display the properties of the input set Consider for example, the following A user may want to devise an algorithm that is capable of learning a triangle-shaped distribution as the one depicted in Figure 1 1 The SOM tries to achieve this by defining an underlying grid-based topology and to fit the grid within the overall shape as shown in Figure 1 1a (duplicated from [103]) However, from our perspective, a grid-like topology does not naturally fit a triangular-shaped distribution, and thus, one experiences a deformation of the original lattice during the modeling phase As opposed to this, Figure 1 1b, shows the result of applying one of the techniques developed by us, namely the TTOSOM, designed and explained in detail in Chapter 3 As the reader can observe from Figure 1 1b, a 3-ary tree seems to be a far more superior choice for representing the particular shape in question Our hope is that we can "feed" this to the TTOSOM, which will, in turn, use it to represent the distribution, from that perspective, in an enhanced manner

On closer inspection, Figure 1 1b depicts how the complete tree fills in the triangle formed by the set of stimuli, and further, seems to do it *uniformly* The final position of the nodes of the tree suggests that the underlying structure of the data distribution corresponds to the triangle Additionally, the root of the tree is placed roughly in the center of mass of the triangle It is also interesting to note that each of the three main branches of the tree, cover the areas directed towards a vertex of the triangle respectively, and their sub-branches fill in the surrounding space around them in a recursive manner, which we identify as being a holograph-like behavior

Of course, the triangle of Figure 1 1b serves only as a very simple *prima facie* example to demonstrate to the reader, in an informal manner, how both techniques

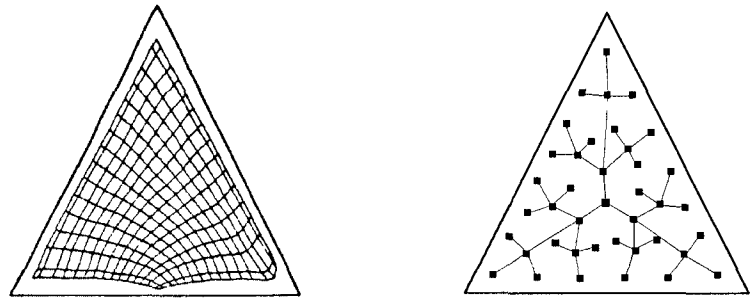


Figure 1.1: How a triangle-shaped distribution is learned through unsupervised learning. (a) The grid learned by the SOM. (b) The tree learned by the TTOSOM.

will try to learn the set of stimuli. It is important to remark that in real-world problems, these techniques can be employed to extract the properties of high-dimensional samples.

One can argue that imposing an initial topological configuration is not in accordance with the founding principles of unsupervised learning, the phenomenon that is supposed to occur without "supervision" within the human brain. As an initial response we argue that this "supervision" is required to enhance the training phase, while the information we provide relates to the *initialization* phase. Indeed, this is in line with the well-accepted principle [57], that very little can be automatically learned about a data distribution if no assumptions are made!

As the next step of our research endeavor, we choose to venture into a world where the neural topology *and* structure are themselves learned during the training process. This is achieved by a scheme called TTOCONROT, in Chapter 5, which, in essence, dynamically extends the properties of the above-mentioned TTOSOM. Again, to accomplish this we need key concepts that are completely new to the field of SOMs, namely those related to tree-based Adaptive Data Structure (ADS). Indeed, as demonstrated by our experiments, the results that we have already obtained have been applauded by the research community[1], and that to the best of our knowledge,

[1]A paper which reported the preliminary results of this study, won the *Best Paper Award* in a well-known international AI conference [12]

have remained unreported in the literature

To explain the work that has been accomplished and the research proposed, we present the motivations and the objectives of the thesis in the following subsection

1.2 Motivations and Objectives

As we have already pointed out, this thesis proposes to investigate the SOM, which is family of ANNs that are capable of learning in an unsupervised manner The main focus of the present thesis is on a particular sub-family of Neural Networks (NNs) possessing properties of the original SOM philosophy, where a tree dictates the neural topology

1.2.1 Survey of the Field

There are numerous variants of the SOM In general, it is possible to differentiate and classify them according to their properties It is also possible to develop a scheme to compare them by subdividing their "building blocks" into different categories, which include the topology, the type of dynamism displayed, the manner in which the neurons are considered to be neighbors, the additional counters utilized in the various types of SOMs, the states assumed by a neuron, and finally, the type of search procedure utilized for identifying the neurons to be associated with an input sample

The topological structure of a NN refers to the type of structure that is formed when the neurons are interconnected It is possible to identify five different types of topologies a grid, a tree, a (possibly unconnected) graph, an irregular grid, and a hierarchical structure built with SOM-layers

A grid topology is usually a two-dimensional array of interconnected neurons In a tree-based topology, the neurons are arranged in a hierarchical manner, starting from the root which is located at the top level of the hierarchy When the topology uses SOM-layers, the units being connected correspond to SOM grids themselves Moreover, an irregular grid is a partially complete grid where some of its nodes (and the corresponding connections) are missing When the NN employs none of these

topologies, we say that the topology pertains to the most general type of structure, i.e., a bi-directional graph.

Chapter 2 includes an in-depth description of the methods and properties listed above. We have examined methods by comparing their building constituents and by classifying them based on these "building blocks". We have also explicitly identified common properties, which have been presented in a comprehensive manner so as to describe how the various strategies are inter-related. Our aim is that the reader can gain as much insight as possible in the inner workings of the principal strategies presented in the literature, so as to explain why they perform certain tasks well, and why they fail in other cases. This approach has permitted us to determine possible research directions, such as the ones presented in Chapter 3 and Chapter 5, and their consequent PR applications in Chapter 4 and Chapter 6 respectively. The rationale for these are expressed below.

1.2.2 Motivations

A substantial amount of work has been done to overcome the difficulties encountered in the traditional SOM strategy. However, we have been able to identify areas that have been unexplored by the state-of-the-art. We can list the motivational aspects of our work as follows:

- The SOM, although possessing amazing properties, fails to represent the data correctly under certain constraints. We would like to reformulate solutions for situations of this kind.

- Even though the SOM is, in essence unsupervised, it relies on a fixed grid structure which is typically defined *a priori*. We would like to examine the consequences of permitting the user transfer his knowledge about the data set by imposing an arbitrary tree-like topology onto the set of neurons.

- We are also interested in a completely new and unexplored way of adapting the SOM. The principles behind this adaptation involve the *fusion* between the philosophies defined by the tree-structured families of SOMs and the field of

ADSs We would like to see if we can generate an asymptotically optimal tree based on the "importance" of the neurons, the quantification of which will be explained in detail

- In the reported literature, *all* the families of NNs work with the premise that the neurons are considered equal Our research queries this concept, and the reason for this is inspired by a phenomenon that we have introduced called "*Neural Promotion*" With Neural Promotion, a neuron is relocated in a more privileged position in the network with respect to the other neurons in the NN Thus, while all "all neurons are born equal", their importance in the society of neurons is determined by what they represent This is achieved by an explicit advancement in its rank or position We believe that this concept, itself, is pioneering

- From a computational perspective, we are interested in knowing if we can accelerate the task of identifying the neuron which best represents the stimulus, also known as the "winner" neuron We hypothesize that this can be achieved, by recursively employing hyperplanes which dichotomize the feature space into two halves

- We would like to see if we can impose a BST structure on the neurons to perform the adaptation

- The implications of imposing a tree with an arbitrary number of nodes on the neurons is also a fascinating unexplored avenue for research We would like to utilize such a model to generate the optimal tree based on the access probabilities of the neurons

This summarizes the motivations of our proposed thesis

1.2.3 Objectives

To render the goal of this research project more explicit, this section presents a series of sub-objectives that we plan to achieve We intend to

1 Devise a SOM-based scheme, i e the TTOSOM, whose topology is tree-based and which condenses the user's knowledge about the data distribution We hope that the algorithm will be able to learn a stochastic distribution using this user-defined perspective

2 Merge the fields of SOMs and ADSs by prescribing a framework, that permits the user to define a new family of NNs which are able to learn a stochastic distribution and simultaneously dynamically re-adapt the data structure so as to build an asymptotically optimal configuration, based on the accesses of the neurons

3 Utilize the SOM+ADS framework explained above to design a strategy that dynamically adapts a BST with the so-called Conditional Rotations (CONROT) strategy [38] Our hope is that this will asymptotically generate the optimal BST based on the access probabilities

4 Design a method that employs the TTOSOM for performing classification, by using both unlabeled and labeled instances

5 Design a method that employs the TTOCONROT This will extend the TTOSOM-based classifier so as to re-arrange the nodes of the tree as per the laws of ADSs

6 Devise a SOM-based scheme which recursively uses hyperplanes to divide the space into two halves Our hope is that by doing so, the time required to identify the "winner" neuron in each training step will be reduced

1.3 Organization of the Thesis Proposal

We shall now give a brief overview of the organization of the present manuscript, including the content as well the main contributions for each of the subsequent chapters

Chapter 2: This chapter concentrates on the basic notions that the reader must comprehend in order to best understand the contributions of the thesis We begin this

chapter by *briefly* discussing the more philosophical aspects concerning the paradigms of unsupervised learning and data clustering, and include a categorization of the different schemes and prominent algorithms reported in the state-of-the-art The next section focuses on the framework of ANNs and explains its main concepts and definitions We then direct our attention to a particular type of ANN, namely the Self-Organizing Map (SOM), which is the main focus of this thesis For this reason, we offer an in-depth description of the algorithm that covers its main properties and applications, and also list the known drawbacks reported in the literature The subsequent sections of the chapter concentrate on the most prominent families of these ANNs that take advantage of the original SOM strategy, sub-dividing them into two main categories, i e , those which use a tree as the underlying topology, and those that do not The chapter concludes by establishing a concise comparison between the different variants of the SOMs reported in the literature[2]

Chapter 3: The beauty of the Kohonen map is that it has the property of organizing the codebook vectors, which represent the data points, both with respect to the underlying distribution and topologically This topology is traditionally linear, even though the underlying lattice could be a grid, and this has been used in a variety of applications [97, 142, 149] The most prominent efforts to render the topology to be structured involves the Evolving Tree (ET) due to Pakkanen *et al* [143], and the Self-Organizing Tree Map (SOTM) due to Guan *et al* [82], among others In this chapter we propose a strategy, the Tree-based Topology Oriented SOM (TTOSOM) by which we can impose an *arbitrary*, user-defined, tree-like topology onto the codebooks Such an imposition enforces a neighborhood phenomenon which is based on the user-defined tree, and consequently renders the so-called Bubble of Activity (BoA) to be drastically different from the ones defined in the prior literature The map learned as a consequence of training with the TTOSOM is able to infer *both* the distribution of the data and its structured topology interpreted via the perspective of the user-defined tree The TTOSOM also reveals multi-resolution capabilities, which are helpful for representing the original data set with different numbers of points,

[2]Since this overview is comprehensive, we have been advised to submit a survey paper containing the material found here This paper is currently being reviewed [13]

and this can be obtained without the necessity of recomputing the whole tree. The ability to extract a skeleton, which is a "stick-like" representation of the image in a lower dimensional space, is discussed as well. These properties have been confirmed by our experimental results on a variety of data sets. A preliminary version of some of the results from this chapter appeared in the *Proceedings of CORES2009, the* 6^{th} *International Conference on Computer Recognition Systems*, Poland, in May 2009 [11]. This talk was a Plenary/Keynote Talk at the Conference. The journal version of these results is currently under review.

Chapter 4: Years of research in the field of PR has led to scores of algorithms which can achieve *supervised* pattern classification. Such algorithms assume the knowledge of well-defined training sets with a clear specification of the identity of *all* the training samples. However, in the last few years, a new stream has emerged, namely, the so-called "semi-supervised" paradigm, i.e., the one that uses a combination of labeled and unlabeled samples to perform classification [176]. Classifiers based on the latter, do not demand the specification of the class labels of every sample. Rather, a clustering-like mechanism processes the manifold, and attempts to distinguish the training samples into the separate classes, subsequent to which a supervised classifier is derived using a small subset of the training samples whose class identity is known.

In this chapter we will venture to use the TTOSOM in pattern classification. We will first train a TTOSOM in which the neurons collectively obey the stochastic, topological and structural distribution of *all* the classes. Subsequent to this juncture, we make use of the information provided in the labeled dataset. By using this information, we assign a class label to every single node in the NN, which, in turn, partitions the space into its Voronoi regions. On receiving the testing data, the task at hand is rather straightforward. One nearly determines the closest neuron to the testing sample and assigns the sample to the corresponding class. The complexity of the testing is linear, not in cardinality of the training set, but rather in the size of the TTOSOM tree.

Such an nearest-neighbor type testing is similar to the ones used by prototype reduction schemes [102, 116]. Such schemes represent the training data from each class by a small set of prototypes, which are subsequently used for classification. In

our case, however, the "prototypes" are the nodes of the TTOSOM which are not located in the feature space in a manner *unrelated* to each other Rather, they are constrained by the tree structure, the topology, and the distribution of *all* the classes

Our experimental results show that on average, the classification capabilities of our proposed strategy, even with a small number of neurons, are reasonably comparable to those obtained by some of the state-of-the-art classification schemes that only use labeled instances during the training phase The experiments also show that improved levels of accuracy can be obtained by imposing trees with a larger number of nodes **Chapter 5:** Numerous variants of Self-Organizing Maps (SOMs) have been proposed in the literature More recent embodiments involve SOMs which also possess an underlying structure [11, 28, 156, 71, 82, 105, 133, 143], and in some cases, this structure itself can be defined by the user [11, 105] Although the concepts of growing the SOM and updating it have been studied, the whole issue of using a self-organizing ADS to further enhance the properties of the underlying SOM, have been unexplored In Chapter 3, we had proposed a strategy by which we can impose an *arbitrary*, user-defined, tree-like topology onto the codebooks, which consequently enforced a neighborhood phenomenon and the so-called tree-based BoA In this chapter, we consider how the underlying tree itself can be rendered dynamic and adaptively transformed To do this, we present methods by which a SOM with an underlying Binary Search Tree structure can be adaptively re-structured using conditional rotations These rotations on the nodes of the tree are local, can be done in constant time, and performed so as to decrease the Weighted Path Length of the entire tree We introduce the earlier alluded-to concept refereed to as *Neural Promotion*, where neurons gain prominence in the NN as their significance increases The advantages of such a scheme is that the user need not be aware of any of the topological peculiarities of the specific input data set or of the stochastic data distribution Rather, the algorithm, referred to as the Tree-based Topology-Oriented SOM with Conditional Rotations (TTO-CONROT), converges in such a manner that the neurons are ultimately placed in the input space so as to represent its stochastic distribution, and additionally, the neighborhood properties of the neurons suit the best BST that represents the data These properties

have been confirmed by our experimental results on a variety of data sets. A preliminary version of some of the results from this chapter appeared in the *Proceedings of AI 2009, the 2009 Australasian Joint Conference on Artificial Intelligence,* Australia, in December 2009 [12]. This paper won the *Best Paper Award of the Conference.* The journal version of these results is currently under review.

Chapter 6: A natural extension to our investigation is to develop a study analogous to the one performed in Chapter 4, but now considering the effect of the neural rotations developed in Chapter 5. In this sense, our proposed methodology consists of deriving a classifier similar to the TTOSOM-based classifier, but using the TTOCONROT as a foundation.

Our experiments show that such a TTOCONROT-based classifier is able to sometimes outperform state-of-the-art classifiers that are unable to learn from unlabeled samples. This is consistent with the results of other researchers, in which semi-supervised schemes lead to performance levels that are comparable to the ones obtained by true supervised methods [73]. More specifically, in most of our experiments, trees whose sizes are only a small fraction of the cardinality of the dataset, are sufficient to obtain accuracies comparable to the ones provided by the best supervised classifiers.

Additionally, we have performed a meticulous analysis to identify the advantages of incorporating the neural rotations provided by the TTOCONROT. To do this, we compared the results with the TTOSOM-based classifier (presented in Chapter 4), using analogous parameter settings and using different tree sizes. Our results showed that regardless of the inclusion of the rotations, competitive accuracy rates can be obtained. Moreover, our experiments also suggest that in certain cases, the rotations lead to accuracy rates that increase in a smoother manner, in comparison to the ones obtained by the TTOSOM-based classifier, as more neurons are incorporated in the tree.

Chapter 7: The previous chapters essentially focused on how to accurately mimic the original properties of the stimulus, and how this can be advantageously utilized for tackling the problems of clustering and classification. In this chapter, we focus on a different perspective, i.e., on how to perform the required calculations more

efficiently Particularly, we concentrate in the most expensive problem associated with the original SOM strategy, namely the one involving the search for the so-called Best Matching Unit (BMU) As a first step in this direction, we show how a hyperplane-based partitioning scheme can be used to accelerate the TTOSOM These hyperplane are recursively used to divide the space into two halves The question of further utilizing this strategy to enhance the search for the BMU is a research topic in itself – which could lead to another doctoral thesis We have thus left it as an open problem

Chapter 8: This chapter concludes the manuscript

Chapter 2

Literature Review

2.1 Introduction

A SOM is a kind of ANN which learns the structure of an input sample set in an supervised/unsupervised manner, generating an output mapping in a (usually) lower dimensional space. Since its conception, numerous variants of the SOM have been proposed through the years so as to enhance the deficiencies of the basic SOM. Among these variants, we focus our attention on a particular subset, a family of NNs derived from the SOM, where the codebook vectors are governed by an underlying tree structure. In general, these approaches attempt to render the topology more flexible, so as to represent complicated data distributions in a better way and/or to make the process faster by, for instance, speeding up the task of determining the BMU. This Chapter explains, in a fairly comprehensive manner, the most relevant strategies that employ such an approach, categorizing them according to their principal properties and components.

The rest of the Chapter is organized as follows: We first, in a less formal manner, cover the concepts of Unsupervised Learning and Clustering in Sections 2.2 and 2.3 respectively. We then give in Section 2.4 an overview of the theory of ANNs. After that, the traditional SOM philosophy is described in detail in Section 2.5 We then review the most relevant SOM-variants in Sections 2.6 – 2.8.

2.2 Unsupervised Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that deals with the formal study of algorithms capable of improving their performance in an automatic manner, based on experience [136]. It considers the analysis, design and implementation of algorithms, which impart to computers the ability to learn. We further differentiate between two main categories in ML, according to the type of information that is provided to the system. These two main learning approaches are *Supervised* Learning and *Unsupervised* Learning.

Supervised Learning is the study of algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances [108]. In Supervised Learning a "teacher" provides the true category of samples or examples. This collection of instances for which the desired output is known *a priori*, is called the "training" set. Supervised Learning seeks the construction of a discriminant function from the training set. The goal is to predict the unknown labels of items that will be subsequently presented to the classification system. One important problem that presents itself when the system learns useful information from the samples is called over-fitting. A model is said to be "over-fitted" if even though it is able to perfectly predict the training items, it is unable to accurately predict the labels of new "testing" patterns. This issue leads to a trade-off between the complexity of the classifier (which usually captures more details about the data distribution of the classes being separated), and a simple discriminant function which is easily implemented, but may be unable to capture the subtle differences between the classes under scrutiny. This kind of learning is usually used in classification problems, and it is the most common way to train ANNs.

Sometimes, the true labels of the items are not available to the "teacher". This may occur because it is not possible to obtain these labels or because retrieving them is too expensive. This scenario corresponds to the paradigm known as Unsupervised Learning. As opposed to the supervised approach, in Unsupervised Learning the class labels are unknown, and one may not know the number of classes either. This poses a harder task, because the essential relationships between the data must be acquired

by merely examining the samples

Even though one works with labeled data, there are still several reasons to consider Unsupervised Learning Consider, for instance, the high cost of collecting and labeling large amount of sample patterns In some cases, the cost of extracting labels is itself prohibitive, restricting the possibility of solving the problem using a supervised technique Alternatively, the intrinsic nature of the categories of the patterns may be dynamic, and unsupervised learning may appear as a more suitable strategy to adapt the model according to the changes in the distribution of the data Furthermore, an unsupervised paradigm may help to gain insight about the underlying data distribution and its structure Last, but not less important, the unsupervised *modus operandi* seems to occur more common in the brain than its supervised counterpart

As explained earlier, sometimes the information about the labels may be missing for some reason, such as the associated cost As an example, consider the situation of examining pictures available on the web Pictures are usually perceived by computers as binary maps, i e , as "cold grids" of pixels without any particular meaning It is the human who is able to make the necessary abstraction to understand the diverse elements represented in the image These elements may correspond to different concepts like a family, a landscape or a car Furthermore, if labels that express these semantics are needed, one typically recruits a human-in-the-loop to tag them When the number of pictures are only a couple of dozens, the problem can be solved manually But when the number of pictures is in the order of millions, the task of labeling may become colossal

ANNs are usually educated using a training set that acts as a tutor However, a subset of neural systems are also capable of learning in an unsupervised manner through self-organization When NNs use an unsupervised approach, the weights are adjusted by merely examining the samples, and without the supervision of an external entity This may happen because the user does not know *a priori* what to expect from the network, forcing the ANN to determine a reasonable output The most prominent of these self-organized networks is the SOM [158], which is the primary focus of this Thesis

Apart from the supervised and unsupervised paradigms, there are other ways to

perform learning Among these alternatives are the semi-supervised learning and the reinforcement learning approaches The semi-supervised learning paradigm is the study of algorithms capable of learning by employing unlabeled *and* labeled data [176] Semi-supervised learning is differentiated from its unsupervised counterpart since the latter uses an entire manifold of unlabeled data Commonly, semi-supervised schemes work with a large amount of unlabeled samples (which are usually easier to obtain), together with a few, and more expensive, labeled data elements utilized to improve the quality of the classifier In reinforcement learning, it is often assumed that an agent learns in a constantly changing environment through trial-and-error attempts [94] Here, when an agent is in a certain state, it can take an action, which, when the so-called oracle observes, it returns the new state of the environment and also a reward (or a penalty) associated with the action performed by the agent Based on this iterative interaction, the agent is expected to learn the optimal behavior so as to maximize the reward

2.3 Clustering

In an increasing number of applications, researchers today encounter various situations which require the classification of patterns in large data sets Central to these methods are approaches which identify the most important *clusters* of the given data in an unsupervised manner A problem that arises in such situations is to capture the essence of the similarity in the samples, which implies that any given cluster should include data of a "similar" sort, while elements that are dissimilar are assigned to different subsets

The human brain possesses an amazing ability to find regularities in data by merely looking at it One way of expressing regularity is to gather the set of objects into groups that are similar to each other As an instance, in the field of natural sciences, biologists have assembled information about various species through the years They then make a distinction between "objects" in the natural world by distinguishing creatures with locomotion capabilities from other types of organisms that are unable to move on their own The first group is labeled as "animals", while

the second, as "plants" Further, individual plants and animals themselves do not possess an inherent label indicating the species they belong to, their *phyla* and so on Rather, humans have further categorized them according to their observable characteristics For example, in the case of the animal kingdom, some interesting characterizing features are the size of the organism, number of legs, hair, feathers and their ability to hunt, run, fly or swim The operation of grouping things together, as in this example, is referred to as Clustering

The SOM, which we shall study in the Thesis, is especially capable of achieving Clustering

2.3.1 Types of Classification

According to Jain *et al* [90], Clustering (also know as Data Clustering[1]) "is the process of classifying objects into subsets that have meaning in the context of a particular problem" These authors further assert that it is possible to view Clustering as a special kind of Classification Figure 2 1[2] shows a decomposition of the main classification schemes As a first subdivision, a classification scheme can be either considered to be overlapping or non-overlapping Subsequently, non-overlapping schemes can use unsupervised or supervised classification Finally, the unsupervised schemes are categorized as being either partitional or hierarchical

When an novel pattern is presented, the question of determining the best cluster that this pattern should fall into is not such a clear-cut decision This constitutes one of the most fundamental problems in Clustering [72] In certain cases, more than one cluster might appear to be the correct category As an illustration, consider the set of animals that are able to fly Although this category mostly includes birds such as hawks, eagles, doves, owls, etc , the group may include other animals that do not belong to the *Aves* class, such as bats, which are flying mammals As a

[1]Usually the term "Data Clustering" is used to differentiate it with a cluster of computers The latter refers to a group of computers linked together, so as to act like a single one In the context of this document the term "Clustering" refers to "Data Clustering"

[2]This figure is duplicated from [90], which constitutes a survey of clustering methods, and which is commonly-used standard reference in the field Given the progress of the field, we admit that there could be a slight overlap between some of these categories However, our interest here is to present a broad classification

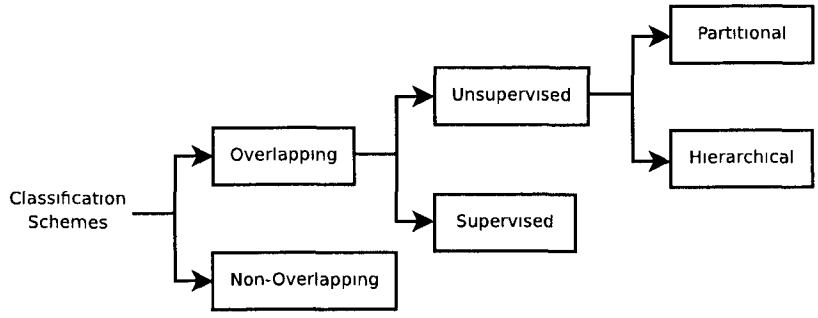


Figure 2.1: A categorization of Classification schemes.

second example, by examining the mammals in more depth, one observes that most animals have four legs and live on land. Here again, exceptions appear, as there are mammals without legs, and which further do not live on land, like whales and dolphins. Moreover, bats possess four legs and the ability to fly.

In non-overlapping schemes, each item belongs exclusively to one category An example of a non-overlapping scheme is the gender of a particular specie such as the duck, where the possibilities are whether the duck is a female or a male. Overlapping classification schemes are more complex and involve items belonging to more than one category. An example of a non-overlapping classification scheme is the Fuzzy k-Means algorithm [22, 23], where the condition of the traditional k-means philosophy that states [122] "each sample x_j is member of one and only one category" is relaxed; instead, each item is assigned to each cluster with some degree of certainty. Our study will focus on non-overlapping classification and clustering.

Non-overlapping (or exclusive) classification computes a *partition* of the set of items being examined. Formally [43], a collection $\mathcal{S} = \{S_i\}$ of non-empty sets forms a partition of a set S if

1. $S_i, S_j \in \mathcal{S}$ and $i \neq j \Rightarrow S_i \cap S_j = \emptyset$
2. $S = \bigcup_{S_i \in \mathcal{S}} S_i$

That is, each item of the input set appears in exactly one subset.

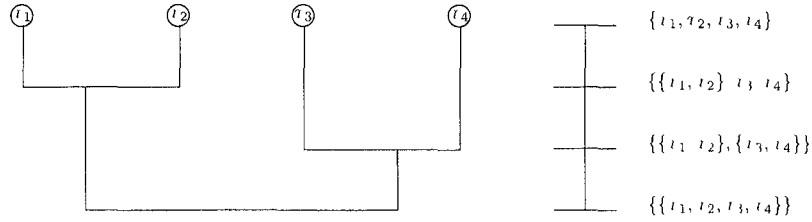


Figure 2.2: Example of hierarchical clustering.

As seen in Figure 2.1, non-overlapping classification schemes can be further categorized as being either supervised or unsupervised as explained earlier.

As a final subdivision, unsupervised classification can be subdivided into being either Hierarchical or Partitional. A Hierarchical Clustering scheme is one that generates a sequence of nested partitions, where each partition is nested into the next partition in the sequence. When the Hierarchical Cluster algorithm starts with each node constituting its own cluster, from which then new clusters are created by merging two clusters, it is called *Agglomerative*. On the other hand, when the Hierarchical Cluster algorithm starts with all the observations falling in a single cluster and it splits the clusters recursively to increase their number, it is called *Divisive*.

A *dendrogram* is a tree representation frequently used to illustrate the arrangement of the clusters produced by a hierarchical clustering. Figure 2.2 depicts a dendrogram showing how four samples are grouped. At the first level, each of the four elements x_1, x_2, x_3, x_4 constitutes a cluster by itself. At the second level of the sequence, the dendrogram shows how x_1 and x_2 are merged together to form the cluster $\{x_1, x_2\}$, and they will continue to be part of the same cluster in all the subsequent levels. In the third level, the clusters $\{x_3\}$ and $\{x_4\}$ are merged into the cluster $\{x_3, x_4\}$, thus producing two clusters, which are $\{\{x_1, x_2\}, \{x_3, x_4\}\}$. Finally the two remaining clusters are mixed together producing the cluster $\{x_1, x_2, x_3, x_4\}$ containing all the samples. A dendrogram is also able to give extra information about the clusters, for instance, the vertical axis may show the measure of similarity among the clusters.

2.3.2 Why to Perform Clustering?

There are several reasons why it is both interesting and necessary to perform clustering This section summarizes the most relevant issues

Clustering techniques can be used as predictive tools When a new item is encountered, the internal model generated previously can help to predict the class of this new observation In this sense clustering is a powerful tool for generating a classifier that can be trained with the available data, and that can be used to identify the label of novel patterns

Additionally, clusters can serve as a data compression technique Sometimes certain operations are difficult to perform due to the large size of the data set Compression of the data points into a few representative virtual points with lower dimensionality can make this possible Vector Quantization (VQ) [80] is a well-known technique used for such a purpose

There are neural systems that employ clustering as a learning engine The SOM, which will be described in detail in Section 2 5, is a prominent example of a NN philosophy that is trained in an unsupervised manner ANNs like this, perform clustering as a model of a learning process using a technique called Competitive Learning (CL), reviewed in detail in Section 2 5 2

Moreover, clusters may help in the detection of noisy patterns In statistics, an outlier is any observation from a collection of data which is inconsistent with the rest of the items in that collection [15] Even though outliers are sometimes considered as noise and are undesirable, this is not always the rule, and clustering techniques may be used for detecting them For example, if we pick pictures of the ocean, it may include elements that are classified as a school of fish, and others as rocks However, if we encounter an item not classified correctly, it could be a good reason to believe that we have located a submarine in the picture

2.3.3 Stages in Clustering

Although, there are different types of clustering methodologies, it is still possible to recognize some common processes involved in classical clustering algorithms [90, 89]

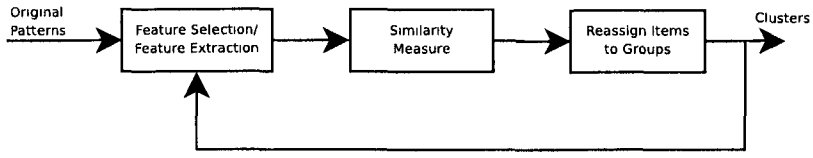


Figure 2.3: Stages involved a typical clustering solution.

Figure 2.3 depicts the most relevant processes performed in clustering.

The first activity in the clustering process is that of selecting a suitable representation for the patterns. This representation refers to the coding of the observations, usually in numerical form, and may involve the normalization of the values. Observe that this step is affected by variables that may be inherent to the problem being solved. For instance, in some clustering problems the number of classes are known *a priori*, while in others it is not. Other constraints include the types of the features that are being processed. Features usually encountered in clustering problems include numerical values such as integers and real numbers, binary-valued attributes or even qualitative values. The most appropriate clustering strategy is also closely related to the amount of data to be clustered. At this stage, it is advantageous to identify the most meaningful attributes. Here, one invokes processes for *feature selection*, which aim to identify the attributes which yield more valuable information, thus explicitly generating a ranking of the attributes and sorting them according to their predictive capabilities. Alternatively, it may also be possible to reduce the number of features by so-called *feature extraction*. Feature extraction is a dimensionality reduction technique achieved by creating (linear) transformations of the selected attributes, resulting in new attributes that condense the most valuable information from the original collection of features, and which permit the construction of a model which could be used to separate the items into clusters, using a lesser number of features.

A central task in the clustering process is to establish how similar the objects being examined are. Quantifying "similarity" is of paramount importance because it is desirable to assign those objects that are similar to the same cluster, while at the same time, separating the items that possess a high degree of dissimilarity into

different subsets

The next step consists of reassigning the samples to the identified groups This step utilizes the pre-determined inter-object similarity measure described earlier The ability to quantify the similarity of an object to an arbitrary element in a cluster offers a tool whereby one can establish the "correct" category of the item, which may, in turn, require a re-labeling of the sample

Finally, the clustering algorithm will output a series of clusters, which group the data items At this stage, the algorithm may terminate or may enter into a new refining iteration process of the same steps described above The decision concerning whether to iterate or terminate depends on the evaluation of the quality of the clusters This criterion is usually associated with a function to be optimized, and in this regard, the most common criterion function utilized is the so-called Sum-of-Squared-Error described in [57]

2.3.4 Similarity Between Objects

A central issue in any clustering methodology consists of quantifying the similarity between the objects to be grouped "Similarity" is a rather generic term that has been widely used and investigated in the fields of Psychology and PR From a psychological perspective, similarity can be understood to be the "closeness" of two mental representations In our research, where we are concerned with the PR perspective, the term "similarity" is quantitatively measured using a so-called distance/metric This metric, in turn, is a function whose inputs are the representation of the objects in question, and whose output is a scalar specifying the distance between them Formally, a metric on a set $\mathcal{X}$ is a function $D \quad \mathcal{X} \times \mathcal{X} \rightarrow I\!R$, satisfying the following conditions

Non-negativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$

Reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$

Symmetry: $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$

Triangular inequality: $D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z}) \geq D(\mathbf{x}, \mathbf{z})$

Examples of relevant distance functions are the binary distance, the Manhattan distance, the Euclidean distance, the graph distance, the Hamming distance and the Mahalanobis distance. The binary distance is a very simplistic measure in which the inter-item similarity between the elements is assigned a value of zero if they are identical, while non-identical items possess a distance of unity. The Manhattan or city-cab distance is equal to the sum of the absolute differences of their coordinates. The Euclidean distance is the length of the direct path connecting the given points. Some of the important properties of the Euclidean distance is that it is invariant to rotations and translation. One drawback of the Euclidean distance, however, is that it is sensitive to changes in the scale, i.e., if the objects are modified by a slightly non-linear transformation, the Euclidean distance between them will not be preserved. Observe though that the Manhattan and Euclidean distances are particular cases of a more general function called the Minkowski distance [57], defined for the d-dimensional space as:

$$L_k(\mathbf{x}, \mathbf{y}) = \{\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^k\}^{1/k}. \tag{2.1}$$

The cases of $k = 1$ and $k = 2$ lead to the Manhattan and Euclidean distances respectively.

In a graph, the so-called inter-node distance is defined as the shortest path connecting two nodes, where, by convention, it is assumed that the distance is infinite when the two given nodes are unconnected [34]. The Hamming distance between two vertices of a hypercube is the number of coordinates by which the two vertices differ [58]. Finally, a very important distance function used in statistical PR is the so-called Malahanobis distance [123], which not only takes into consideration the representation of the objects, but also the covariance associated with their assigned classes.

2.3.5 Measuring the Quality of the Clusters

Once the cluster algorithm returns a set of clusters as its output, one relevant question is that of knowing how "good" the clusters are. The answer to this question is

not unique and is typically dependent on the application domain. Usually, the user who possesses *a priori* knowledge about the data being analyzed, defines a suitable quality measure. This section summarizes the most relevant criteria for evaluating non-overlapping clusters, i.e., the cluster algorithm assigns an items to exactly one cluster.

The most common and simplest evaluation criterion is the so-called Sum-of-squared-error. The principle here is to measure how the points are spread out with respect to their centroid. If the center of mass of cluster $\mathcal{C}_i$ is given by $\mathbf{m}_i$, then the Sum-of-squared-error function is

$$e_{\mathcal{C}_i} = \sum_{\mathbf{x} \in \mathcal{C}_i} \|\mathbf{x} - \mathbf{m}_i\|^2. \tag{2.2}$$

Equation (2.2) provides meaningful information regarding a set of data points, however, it does not provide information relative to the spread of the data. An alternative family of error functions incorporate the so-called Scatter Matrix, which is a $d \times d$ positive semi-definite matrix (and is actually $n - 1$ times the sample covariance matrix). The Scatter Matrix is given in Equation (2.3) as follows:

$$S_i = \sum_{\mathbf{x} \in \mathcal{C}_i} \{\mathbf{x} - \mathbf{m}_i\}\{\mathbf{x} - \mathbf{m}_i\}^t, \tag{2.3}$$

where the term $\mathbf{m}_i$ corresponds to the sample mean given by:

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{C}_i} \mathbf{x}. \tag{2.4}$$

In the case of Equation (2.4), the n_i term is the number of samples for each class, respectively.

The literature also contain other scatter matrices related to Equation (2.3), including so-called Within-Cluster Scatter Matrix, Between-Cluster Scatter Matrix, and the Total Scatter Matrix [57].

The Within-Cluster Scatter Matrix, represented by S_W is the sum of all the Scatter Matrices for each class, i.e.:

$$S_W = \sum_{i=1}^{c} S_i \tag{2.5}$$

where c is the total number of clusters. The Within-Cluster Scatter Matrix is symmetric and positive semidefinite, implying that its eigenvalues are nonnegative [125]. It is also proportional to the sample covariance matrix.

The Between-Cluster Scatter Matrix, denoted by S_B, is defined by Equation (2.6) as follows:

$$S_B = \sum_{i=1}^{c} n_i \{\mathbf{m}_i - \mathbf{m}\}\{\mathbf{m}_i - \mathbf{m}\}^t, \tag{2.6}$$

where $\mathbf{m}$ is the so-called total mean vector given by:

$$\mathbf{m} = \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}, \tag{2.7}$$

and where n is the total number of samples.

The Total Scatter Matrix, S_T, can be obtained by adding the Within-Cluster Scatter Matrix and the Between-Cluster Scatter Matrix.

The reader must note that the matrices themselves do not provide a scalar index that is useful for measuring similarity. In other words, we need to specify error functions that transform the information contained in the scatter matrices so as to yield a scalar which quantitatively expresses how "good" the clusters are. This can be achieved by either computing the determinant or the trace of the corresponding matrix. If the latter is used as a criterion, one attempts to minimize the trace of the Within-Cluster Scatter Matrix, given by:

$$tr[S_W] = \sum_{i=1}^{c} tr[S_i] = \sum_{i=1}^{c} \sum_{\mathbf{x} \in \mathcal{C}_i} \| \mathbf{x} - \mathbf{m}_i \|^2 \tag{2.8}$$

2.3.6 k-means

The k-means is a very popular and easy-to-implement clustering algorithm proposed by MacQueen [122]. The main concept here is to place k centroids in the feature

space and thereafter, using a pre-defined distance function, the distance from each sample point to the centroids is calculated, whence each sample is assigned to its closest centroid. Subsequently, the position of the k centroids are recomputed, and the process is repeated until no change in the centroids is found (see Algorithm 1).

Algorithm 1 k-means($\mathcal{X}$,k)

Input:

i) $\mathcal{X}$, a set of unlabeled samples belonging to the $\mathbb{R}^d$ space.

ii) k, the desired number of clusters.

Output:

i) The centroids $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k$.

Method:

```
1 Place k points w_1, w_2, ..., w_k in the space represented by the objects that are
  being clustered.
2 repeat
3   for all x ∈ X do
4     Assign object x to the closest centroid w.
5   end for
6   for all i ∈ k do
7     recalculate the position of centroid w_i
8   end for
9 until centroids no longer move
```

End Algorithm

Recent studies shows that the worse case running time of the k-means is super-polynomial [9], with a lower bound $2^{\Omega(\sqrt{n})}$, where n is the total number of data samples. Although, in practice, the algorithm runs efficiently and it is often cataloged as an iterative optimization procedure, it does not necessarily find the optimal cluster configuration.

In spite of the above handicap, the k-means algorithm is still competitive due to its simplicity and fairly good performance. However, it possesses some fundamental drawbacks. For instance, one important problem associated with the approach, is that some clusters (centroids) might become abandoned, i.e., other centroids consistently become the nearest ones for *all* the samples. In practice, though, this handicap can be solved by running the algorithm repeatedly.

The algorithm is also significantly sensitive to the position of the initial randomly-selected cluster centers Formally, there is no specification about how this issue can be addressed One alternative is that of choosing a random set of k elements from the data set itself

The k-means approach has been demonstrated to be very effective in separating clusters coming from hyper-ellipsoidal-shaped clouds of points, as for example, random vectors from an underlying Gaussian distribution However, more complex problems are encountered when one attempts to separate real data where the input samples do not necessarily conform to hyperellipsoidal data clouds [84]

The k-means algorithm is an example of hard CL, where the winner (in this case the centroid closest to the input signal) takes "all" (in comparison to strategies like the SOM, which is the central NN discussed in this Thesis, where the winner take "most", as shown in section 2 5 2) It is important to note that the k-means algorithm requires that the number, k, of clusters must be known *a priori*, i e , it requires some prior knowledge about the data distribution – which may not always be available

An important issue associated with this simple strategy, though fast in convergence, is that it does not include a relationship between the different centroids themselves Rather, it has been reported that by explicitly representing the relationship between the centroids, one can glean more information about the underlying data distribution This additional capability can also be used to enhance the capability of the SOM, as we shall show in Chapter 3

2.3.7 Hierarchical Approaches

in Section 2 3 1 an overview of Hierarchical Clustering is given Here we focus on a specific algorithm [174], so as to provide more detailed understanding on how a Hierarchical Clustering method works In [174], Ward defines a procedure for grouping mutually exclusive subsets, based on the similarity between them More formally, the technique permits the reduction from k subsets to $k-1$ by measuring the similarity between all the $(k)(k-1)/2$ pair of clusters and merging the two possessing the maximum value for a given similarity function The process is repeated until all data

elements falls into the same partition

2.4 Artificial Neural Networks

Since the human being is, arguably, the "animal" which possesses the highest level of intelligence, researchers have been motivated by the following question Is it possible to "artificially" create an intelligent machine capable of reasoning in a manner similar to that of an average human being? Many theories have been proposed to respond to the above-mentioned query, but, probably few of them have been as interesting or impressive as ones which attempt to *mimic* the mechanism employed by the neurons in the brain itself This Section briefly presents some of the most fundamental aspects of one such paradigm, referred to as Artificial Neural Networks (ANNs)

The theory and applications of ANNs constitute an important branch of ML, thus formally providing a computer with the ability to learn from examples A general overview of the biological inspirations of this paradigm are given in [62][3] The field of ANNs is the most prominent area from among a family of philosophies collectively known as "connectionism", which focus on the study of mental (or behavioral) processes as being emergent as a consequence of interconnected units[4]

In this Thesis we shall use the following definition for NNs, taken exactly from [85]

A Neural Network (NN) is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experimental knowledge and making it available for use It resembles the brain in two respects

1 *Knowledge is acquired by the network from its environment through a learning process*

2 *Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge*

[3]We shall consistently use the term NN synonymously with the more strict term ANN

[4]The concepts of connectionism and ANNs are so closely interrelated, that in some cases they are used interchangeably, as in [74]

ANNs are composed by a set of architectural frameworks, allowing a designer to build a machine/program that is able to accept examples and learn from them The main component of an ANN is a set of nodes linked together through connections leading to a network of neurons Each node resembles the concept of a neuron in the brain, i e , each unit is simple in essence, providing only limited computational power However, several neurons are collectively able to perform complex tasks when they operate together

The way how the neurons are connected between themselves is referred to as the network's *topology*, and differs from one architecture to another The learning process is achieved by adjusting the internal representation of the NN, and this is dictated by a so-called *learning algorithm* Additionally, the patterns being learned are often presented in a numerical form, which, in turn, can have an impact on the internal representation of the NN From a PR perspective, the representation of an object in the real world is not unique It is typically determined by many factors, such as the type of classes that are being separated, as well as the type of sensors that are available This fact has an implication on the interpretation of the original object, which will, in turn, determine the representation of the patterns (by virtue of the weights) in the NN, and the associated output

ANNs are particularly powerful for solving problems in a vast diversity of domains, ranging from PR (e g , credit-risk assessment), data-processing (e g , adaptive signal processing), and the approximation of arbitrary functions (e g , time-series modeling and prediction) [93]

2.4.1 Neurons in the Brain

Biological species, such as human beings and animals, are equipped with powerful nervous systems composed of millions of interconnected neurons [158] The speed by which these neurons inter-communicate is very slow in comparison to the speed required for transmitting bits in an electronic computer However, even though the underlying communication is slow, natural NNs are capable of providing a response to complex tasks in small amounts of time with an amazingly impressive accuracy

In fact, natural NNs are able to "easily" solve numerous problems that their artificial counterparts are yet unable to solve. A central justification for the computational power of the nervous system lays in the way the neurons are connected. Since the field of ANNs originally found its inspiration from the class of biological NNs, we have chosen to include a *brief* summary of the way by which the nervous system addresses the problem of processing information.

The term *neuron* was introduced by the Nobel prize winner Ramón y Cajal [155], who was the first neuroscientist to explain the complex function of the brain through the interconnection of these much-simpler processing units. Since our goal is to compare the biological neuron model with the artificial model (subsequently proposed in [129] during the twentieth century), we will adopt the simplistic neural version based on four main elements described in [158], consisting of dendrites, synapses, the cell body and the axon.

The basic functionality of a neuron consists of perceiving excitatory signals and producing a corresponding response. Figure 2.4 shows the diagram of a biological neuron (adapted from [85]). The dendrites are responsible for perceiving chemical signals emitted by neighboring neurons. The extremes of the dendrites are connected to other neurons which possess specialized junctions that allow the flow of these chemical signals. In the human brain, the number of interconnections are much larger than the number of neurons. The component that is responsible for producing the chemicals necessary for the transmission and subsistence is the mitochondria, located in the body cell. Finally, the chemical signals produced by the mitochondria are themselves transmitted through the axon.

In neural systems the information is accumulated at the synapses. The question of whether some other forms of storage occur in the brain, is, as yet, both unknown and unexplained [158].

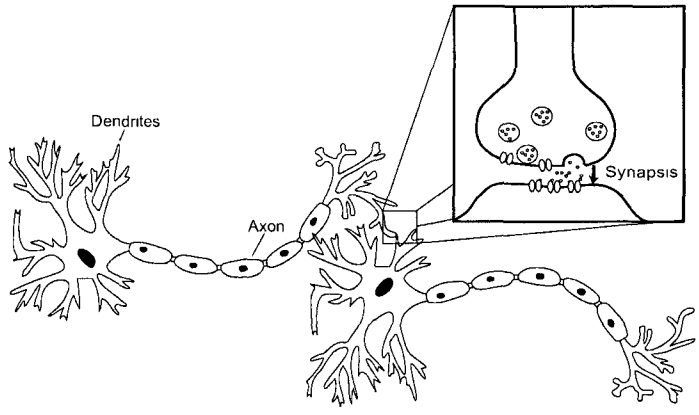


Figure 2 4 Model of the biological neurons in the brain

2.4.2 The Artificial Neuron Model

One important and fundamental step in the science of connectionism was the mathematical abstraction of the neuron due to McCulloch and Pitts [129][5] By using a simplified model, these authors tried to understand how the brain performs complicated tasks by interconnecting simpler processing units, and as a result of their study, they generated a simplistic model of the natural neurons in the brain, one which captured its essential biological properties

This simple but powerful model, allowed scientists to design networks and to adjust the weights so as to solve tasks such as resolving the elementary OR and AND functions in the logic domain This, in turn, indeed, provided a tool by which researchers could *design* a network topology with the appropriate weighted edges in order to solve certain classification problems

The artificial neuron model, depicted in Figure 2 5, contains some elements that are analogous to those found in the biological one First of all, the artificial model has a set of input sensors that perceive the external stimuli, thus mimicking the dendrites Each input branch captures a particular feature of a stimulus and associates with it a

[5]A more recent version of the original 1943 manuscript has been reprinted in [130]

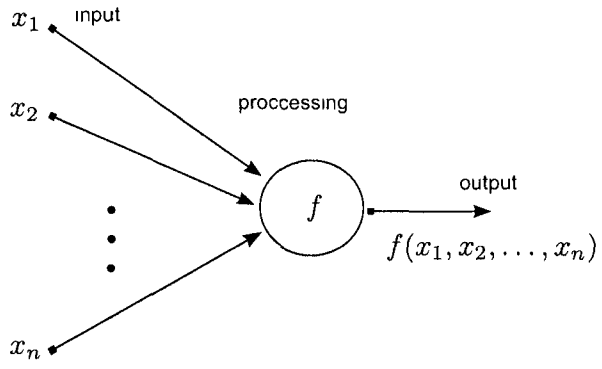


Figure 2.5: Model of an artificial neuron.

weight which will determine the influence of that input feature in the overall response of the neuron. Secondly, the model incorporates a processing unit that generates a response based on the stimulus. This unit reassembles the role of the mitochondria. Additionally, the output module transmits the response signals in a manner analogous to the axon. Finally, other neurons can be connected to the output of a given neuron, providing to them the capability of sensing the emitted signals. This connection between neurons is the abstraction of the synaptic connections between biological neurons.

2.4.3 The Perceptron Model

The classical Perceptron algorithm [159, 160], is probably one the oldest and simplest instantiations of an ANN. It is an example of supervised learning, in which the algorithm receives as input a set of labeled examples belonging to two different categories. The Perceptron algorithm generates a linear discriminant function, i.e., it attempts to create a hyperplane separating the instances according to their category. Given the nature of this philosophy, the instances can be successfully separated only if the categories are linearly separable, and when the latter is the case, the convergence of the algorithm is guaranteed [29, 135, 139].

The Perceptron model assumes that all the examples $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ are numerical, where each $\mathbf{x}_i$ belongs to the d-dimensional real space, i.e., $\mathbf{x}_i \in I\!R^d$. The

true category of each of the patterns is contained in the vector $\mathbf{y} = \langle y_1, y_2, \ldots, y_n \rangle$, where y_i specifies the corresponding true category of $\mathbf{x}_i$. For simplicity, it is often assumed that the labels are in $\{-1, 1\}$. In the Perceptron learning algorithm, described in Algorithm 2, the "prediction" weights $\mathbf{w} = \langle w_1, w_2, \ldots, w_d \rangle$ are initialized with arbitrary values, for example, randomly. The principal iteration of the algorithm estimates the predicted class of every pattern presented to the network as per Equation (2.9):

$$\hat{y} = \mathbf{w}^t \mathbf{x}. \tag{2.9}$$

Equation (2.9), corresponds to the inner product between the weight vector and the stimulus. When $\hat{y}$ does not have the same sign as the true category of the pattern, the pattern is misclassified, and a correction to the estimate $\mathbf{w}$ is necessary. If the misclassification occurred because the pattern belonged to class A (i.e., $y = 1$) but the estimate is negative, then the weight vector is adjusted by adding the pattern to it. A second type of misclassification occurs when the estimate is positive but the pattern belongs to class B (i.e., $y = -1$), in which case the weight vector is corrected by *subtracting* the pattern being examined. This process is repeated until either all the values are correctly classified (which occur in the linearly separable case), or when the user is satisfied with the classification error.

2.4.4 Neurons as Self-Organizing Machines

The perceptron explained above is an example of supervised learning, i.e., it allows a neural machine to learn from *labeled* examples. This section focuses on NNs that are able to learn from examples even when their true categories are unavailable, i.e., learn from examples without a teacher.

The Canadian neuropsychologist, Donald O. Hebb [86] postulated the first rule for self-organized learning. Hebb observed that two neurons that are simultaneously activated should have a stronger linkage in comparison to those neurons that are not related. Algebraically, the Hebbian rule can be specified as Equation (2.10):

Algorithm 2 `Perceptron`($\mathcal{X}$,$\mathbf{y}$)

Input:

i) $\mathcal{X}$, a set of patterns $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, where $\mathbf{x}_i \in I\!R^d$.

ii) $\mathbf{y}$, a sequence $\langle y_1, y_2, \ldots, y_n \rangle$, where $y_i \in \{-1, 1\}$ specifies the true category of $\mathbf{x}_i$.

Method:

```
1  Initialize the weight vector w ∈ IR^d, e.g., randomly.
2  repeat
3    for i = 1, 2, ..., |X| do
4      ŷ_i ← w^t x_i
5      if ŷ_i ≤ 0 and y_i = 1 then
6        w ← w + x_i
7      end if
8      if ŷ_i ≥ 0 and y_i = −1 then
9        w ← w − x_i
10     end if
11   end for
12 until all units are estimated correctly or when the user is satisfied with the
   classification error.
```

End Algorithm

$$\Delta \mathbf{w}_{ij} = \eta \mathbf{x}_i \mathbf{y}_j, \tag{2 10}$$

where the input is $\mathbf{x}_i$, the output is $\mathbf{y}_j$, $\mathbf{w}_{ij}$ measures the relationship between the input and the output, and the parameter, η, is a learning factor that determines the amount by which the interconnection is reinforced. This rule will increase the weight that connects both the input and the output, thus leading to a saturation point. This further suggests the need of alternate rules that dampen the connections from reaching a saturation level after repeated receptions of stimuli. Further, it is also possible to derive from his postulates, that the connection weights of the neurons that are not active decrease in strength. Hebb postulated that the limited resources forced neurons to compete for the energy available, and the "fittest" units would be "triumphant", in one way or another, at the expense of the unsuccessful neurons. Additionally, even though competition is predominant, there is also cooperation between neighboring neurons, because a single neuron by itself is simply not powerful enough to achieve results efficiently. Consequently, a neuron that is excited tends to strengthen its

connections to other units in its vicinity

Another important requirement that renders self-organization possible is the redundancy of the input data If the input data is uncorrelated, it possesses no structure that can be learned As related examples are presented to the NN, it is capable of capturing, in its synaptic relationships, the essence of the entire structure of the data distribution, and thus learning it

2.4.5 Neural Networks Architectures

The architecture of a neural network is the structure of the connection between the neurons The simplest of the architectural models is the single-layered NN As its name indicates, single-layered networks consist of a unique layer of neurons that simultaneously serve as output units In these settings, strictly speaking, there are two layers – the first generating the layer of input nodes which are connected to the layer of computational units The term "single-layered", however, refers to a single *layer* of neurons, corresponding to an abstraction made from biological neural neurons as explained in Section 2 4 1 The second architectural model corresponds to the so-called multi-layered network, where new layers of neurons are inserted in between the input and output layers These "inner" layers are also referred to as being "hidden" layers, because they are designed to perform only internal processing Multi-layer NNs are much more expressive than their single-layer counterparts Indeed the results proven by Kolmogorov [106] and refined by others, show that any continuous function (from input to output) can be implemented by a multi-layer network with only a single hidden layer, provided that there are a sufficient number of hidden units, appropriate nonlinearities, and the respective weights [57] Closely related to the multi-layer NN is the Convolutional Architecture Convolutional NNs also possess input, output and a series of hidden layers However, the difference when compared to the multi-layer architecture lies in the fact that each hidden unit receives inputs from a spatial portion of the previous layer and not from all of it Convolutional Networks have found particular applications in optical recognition systems [115]

Another important family of NNs is the so-called Recurrent NNs [57] The Recurrent networks are composed of a diversity of architectural layouts, but in general all of them incorporate a static multilayer NN which takes advantage of the non-linearities present in the mapping

2.5 The Self-Organizing Map (SOM)

In a considerable number of applications, researchers today face situations requiring the classification of patterns in large data sets As discussed in Section 2 3, determining the clusters using an unsupervised learning paradigm is essential to these schemes Moreover, an issue that appears in such situations is that of capturing the essence of the similarity possessed by the stimuli, further implying that any given cluster should include data of a "similar" sort, while elements that are dissimilar are assigned to different subsets

One important advantage of the families of ANNs introduced in Section 2 4, is precisely that they are capable of learning in an unsupervised way One of the most important families of ANNs used to tackle the above-mentioned problems is the well-known Self-Organizing Map (SOM) also referred to as the Kohonen network, granting credit to its inventor [103] The SOM is a single-layered NN designed for performing clustering and visualization Usually, it is trained using (un)supervised learning to produce a neural representation in a space whose dimension is usually smaller (typically, of magnitude two or three), than that in which the training samples lie Further, they seek to preserve the topological properties of the input space

The SOM concentrates all the information contained in a set of n input samples belonging to the d-dimensional space, say $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \quad, \mathbf{x}_n\}$, utilizing a much smaller set of neurons, $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \quad, \mathbf{c}_m\}$, each of which is represented as a vector Each of the m neurons contains a weight vector $\mathbf{w} = [w_1, w_2, \quad, w_d]^t \in I\!R^d$ associated with it These vectors are synonymously called "weights", "prototypes", "codewords" or "codebook" vectors The vector $\mathbf{w}_i$ may be perceived as the *position* of neuron $\mathbf{c}_i$ in the feature space During the training phase, the values of these weights are adjusted simultaneously so as to represent the data distribution and its structure In each

training step a stimulus (a representative input sample from the data distribution) $\mathbf{x}$ is presented to the network, and the neurons compete between themselves so as to identify which is the "winner", i e , one that best represents the current input signal After identifying the winner neuron, a subset of the neurons "close" to the winner are considered to be within the so-called Bubble of Activity (BoA), which further depends on a parameter specified to the algorithm, namely, the so-called radius Thereafter, this scheme performs a migration of the codebooks within that BoA so as to position them closer to the sample being examined The migration factor by which this update is effected, depends on a parameter known as the learning rate, which is typically expected to be large initially, and which decreases as the algorithm proceeds, and which ultimately results in no migration at all Algorithm 3 describes the details of the SOM philosophy In Algorithm 3[6], the parameters are scheduled by defining a sequence $S = \langle S_1, S_2, \quad , S_s \rangle$, where each S_i corresponds to a tuple (η_i, r_i, t_i) that specifies the learning rate, η, and the radius, r, for a fixed number of training steps, t_i

Algorithm 3 SOM($\mathcal{X}$,$\mathcal{S}$)

Input:

i) $\mathcal{X}$, the input sample set

ii) $\mathcal{S}$, the schedule for the parameters

Method:

1 Initialize the weights $\mathbf{w}_1, \mathbf{w}_2, \quad , \mathbf{w}_m$, e g , by randomly selecting elements from $\mathcal{X}$

2 **repeat**

3 Obtain a sample $\mathbf{x}$ from $\mathcal{X}$

4 Find the Winner neuron, i e , the one which is most similar to $\mathbf{x}$

5 Determine a subset of neurons close to the Winner

6 Migrate the closest neuron and its neighbors towards $\mathbf{x}$

7 Modify the learning factor and radius as per the pre-defined schedule

8 **until** no noticeable changes are observed

End Algorithm

[6]In [103], Kohonen defined two alternative specifications for the SOM The first of these considered a single stimulus selected in each iteration As opposed to this, the second operated in the so-called "batch" mode, which worked with an "epoch" of iterations An "epoch" in that context refers to a training step which includes *all* the instances For the purpose of this thesis, we will maintain the former specification

In the subsequent sub-sections we will discuss the various steps of Algorithm 3 in detail. After that, we will cover some related topics such as its visualization capabilities (Section 2.5.5), topology preservation properties (Section 2.5.6), the reported applications of the SOM found in the literature (Section 2.5.7) and also its most relevant drawbacks (Section 2.5.8).

2.5.1 Initializing the Weights

The initial values assumed by the weights $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ can be assigned by using different strategies. In this section we discuss the most relevant ones.

Random initialization is a simple scheme which is commonly utilized for demonstrating that the SOM is capable of learning using *arbitrary* initial values, i.e., even though the SOM starts with completely unordered weight vectors, the network is still capable of learning the properties of the data distribution in the long run.

An alternative approach, which is more practical than random initialization, involves the random selection of samples from the input sample set. Which such an initialization one can ensure that the initial weights fall into the zones where the input samples lie. The down side of random sampling is that it is vulnerable to the selection of outliers. Also, if, by coincidence, a weight vector assumes the position assigned *as per* random noise, it encounters the risk of falling too far from the data distribution, implying that other neurons will always win the competition, thus leading to the possibility of building ill-formed maps.

A more sophisticated strategy, that, according to the author of [103], gives good results, is the linear initialization strategy. In this mode, the user attempts to initially place the codebook vectors in a more meaningful position at the cost of some extra computation. One could, for example, compute the eigenvectors, and define a hyper-rectangle based on the eigenvectors of the two highest eigenvalues where the original SOM-grid is defined.

2.5.2 Finding the Best Matching Unit (BMU)

One of the most crucial and expensive tasks associated with the SOM algorithm is the one of identifying the most suitable neuron in a process called Competitive Learning (CL).

According to the principles of CL, when a sample $\mathbf{x} \in \mathcal{X}$ is presented to the network, the neurons compete so as to identify which of them best represents the input signal. As a result of this task, a single winner is selected and is usually known as the Best Matching Unit (BMU), denoted by $s(\mathbf{x})$ determined as per Equation (2.11):

$$s(\mathbf{x}) = \arg\min_{i} \| \mathbf{x} - \mathbf{w}_i \|, \tag{2.11}$$

where $\| \cdot \|$ denotes the appropriate norm. Moreover, the notation can be extended to refer to the closest unit $s_1(\mathbf{x})$, the second closest unit $s_2(\mathbf{x})$, etc.

Equation (2.11) is a key component in the phenomenon referred to by the term "competition" between the neurons. From this perspective, it is possible to classify ANNs into two main sub-categories: In the first family, only a single output neuron is activated, and this is referred to as *Hard* CL. As opposed to this, in the second approach, both the winner and other associated neurons are involved in the process, and this is referred to as *Soft* CL.

Finding the BMU in the SOM involves checking every neuron of the network, thus requiring $O(m)$ time, where $m = |\mathcal{C}|$ is the total number of neurons. Moreover, because determining the BMU is required at every training step, the overall time complexity associated with this task is $O(m \times T)$, where T corresponds to the total number of the training iterations.

2.5.3 The Neighborhood Function

Once the BMU is determined, both, it and some neighboring units are migrated toward the input presented to the network. This subset of neurons is known as the neighborhood of the BMU. This is a very important concept of the SOM algorithm, because it involves determining which neurons have to be updated at each learning

step So as to achieve convergence, the SOM requires that this neighborhood will be sufficiently large at the beginning, and decreasing to a limiting value of zero as the number of iterations increase

It is possible to distinguish between two basic types of neighborhood functions The first is the so-called neighborhood set, also known as the Bubble of Activity (BoA) The second, and more elaborated type, involves a Gaussian function as a smoothing kernel function In this Thesis, we focus on the former type of neighborhood function

Typically, to obtain the BoA, a subset of points around the neuron $\mathbf{c}$ is determined according to their neural distances to the BMU This definition requires the inclusion of a parameter known as the *radius* of the neighborhood The radius, $r \in \mathbb{N}$, is an integer value which is expected to be large at the beginning and approaches to zero as the data set is learned The size of the BoA varies with the value of the radius Larger values for the radius will usually be associated with more neurons in the BoA and vice versa Thus, for example, a radius equal to unity includes all the units next to the BMU in the lattice space, while a radius of zero involves *only* the BMU The BoA is defined as the subset of nodes within a neural distance of r away from the node currently examined, and can be formally defined as

$$\phi(\mathbf{c}_i, L, r) = \{\mathbf{c} | d_N(\mathbf{c}_i, \mathbf{c}, L) \leq r\}, \tag{2 12}$$

where $\mathbf{c}_i$ is the node currently being examined, and $\mathbf{c}$ is an arbitrary node in the lattice L, whose nodes are $\mathcal{C}$ Usually

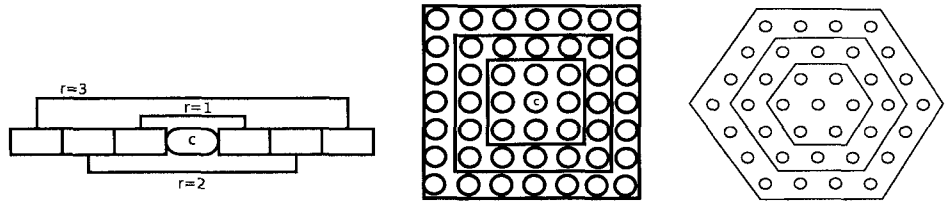
$$\phi(\mathbf{c}_i, L, 0) = \{\mathbf{c}_i\}, \tag{2 13}$$

$$\phi(\mathbf{c}_i, L, i) \supseteq \phi(\mathbf{c}_i, L, i-1), \tag{2 14}$$

and

$$\phi(\mathbf{c}_i, L, |\mathcal{C}|) = \mathcal{C} \tag{2 15}$$

Figure 2.6 illustrates different types of BoA used in the traditional SOM, including a 1-dimensional, a 2-dimensional and a hexagonal lattice. As shown in Figure 2.6a, when the neurons are arranged in a 1-dimensional grid, i.e. a list, and the radius is unity, the set of neighbors included in $\phi(\mathbf{c}_i, L, 1)$ is precisely the direct neighbors of $\mathbf{c}_i$, while a radius of 2 implies that all the direct neighbors of the neurons contained in $\phi(\mathbf{c}_i, L, 1)$ are considered as well. Also note that the BoA can be recursively defined in this manner.



(a) Linear lattice (b) Rectangular lat. (c) Hexagonal lat.

Figure 2.6: The concept of BoA, as applied to different types of lattices.

2.5.4 Migrating the BMU and its Vicinity

As explained in Section 2.5.3, once the BMU is determined, the SOM algorithm, instead of updating just the BMU, also calculates the BoA containing the neurons which are close to the BMU in the *neural* space. Once this neighborhood is calculated, the next step of the SOM consists on migrating the neurons included in the BoA. The weights $\mathbf{w}_i$ of the BMU and those in its vicinity are updated according to Equation (2.16):

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(t)(\mathbf{x} - \mathbf{w}_i(t)). \tag{2.16}$$

The idea behind this updating is that those units close to the winner will "absorb" some of the information provided by the sample, which is achieved by a translation (or migration) towards the stimulus, $\mathbf{x}$.

The amount by which the neurons are moved towards the input sample, is determined by the so-called learning rate, $\eta(t) \in [0, 1]$, which is dependent on the iteration

number, t, and which affects the speed of learning. Large values of the learning rate, (i.e., values close to 1) will cause the *momentum* of the neurons to be larger. Usually larger learning rates are associated with a more chaotic behavior for the neurons. The advantage of increasing the degrees of freedom of the codebooks is that it provides an improved capability for searching in the hyper-space. In spite of this advantage, larger values for the learning rate imply that the neurons absorb most of the information provided by the *current* sample presented to the network, and thus tend to "forget" the signals previously obtained. On the other hand, smaller values for η, (i.e., values close to 0), are responsible for more refined changes in the position of the weights, and as a result of this, one obtains smoother changes in the positions. Generally speaking, the initial use of small values for this parameter does not lead to a good convergence, and thus larger values in the initial phases is recommended. The SOM requires that these large values at the beginning, are gradually decremented so as to obtain good quality maps.

Figure 2.7 illustrates the case when the closest neuron, i.e. the BMU is moved in the direction of the sample presented to the network. In this example, the square represents an input signal and the small circles represent the neurons. A larger circle, that is concentric to the input sample explicit indicates the neuron that is closest to the sample. The final location of the BMU after the updating process is shown with a dotted circle. The figure also displays an arrow showing the direction of this migration, which is linearly towards the input sample. For simplicity, this example shows only how *a single* neuron is migrated, but, in general, this process is repeated for every neuron within the BoA.

2.5.5 Visualization Capabilities

Apart from the ability of preserving the properties of the input space, the SOM is also capable of visually displaying a human-readable map in the 2-dimensional (or 3-dimensional) space that seems to imply that the SOM "understands" the original data. When the SOM is properly trained, the information is compactly stored by the codebook vectors, and thus, it is possible to obtain a visual representation of the

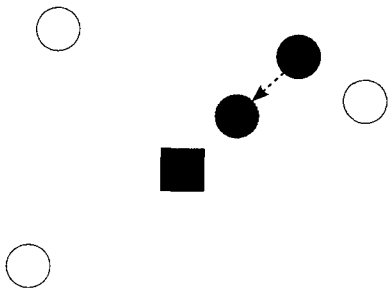
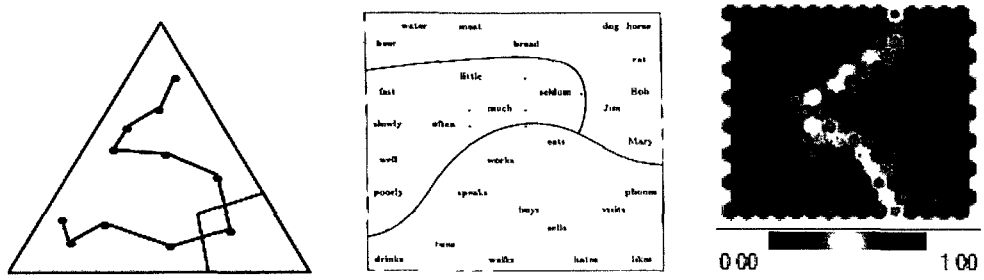


Figure 2.7: This figure shows how the BMU is migrated in the direction of the sample presented to the network.

structure of the data distribution. This mapping is possible even when the feature vectors belong to higher dimensional spaces. Although the SOM usually utilizes a 2-dimensional grid to visually display the mapping, it is not unusual to encounter a hexagonal neighborhood correspondence as well. Fig. 2.8 depicts three examples of how the SOM graphically displays the map being learned. Fig. 2.8a shows a linear grid, i.e., a list that learns the structure of a triangular-shaped data distribution. The neurons in this 1-dimensional chain are adjusted to mimic the properties of the 2-dimensional shape of the triangle, and as a result of this process, the list is twisted so as to conform a shape akin to a Peano curve [148]. Fig. 2.8b illustrates a typical case when the original data is composed of real-valued or binary data, which includes a label. In such a situation, the SOM uses unsupervised learning to train the data, after which it finally displays the labels for which each neuron is the BMU, meaning that one neuron can host more than one sample (which is not the case of the particular example of Figure 2.8b). The interesting phenomenon of such a map is that the final topology preserves the property that similar elements in the original feature space (in this case nouns, adjectives and verbs), will respectively tend to be together in the map. It is possible that more than one element is associated with the same codebook, in which case two labels will appear in a node of the grid. The figure also shows certain nodes in the grid that do not include any label. These nodes, correspond to those neurons that are not the BMU for any of the samples of the input data set. A third example, shown in Figure 2.8c involves an hexagonal lattice, like



(a) A linear map (b) A grid map with labels (c) A grid map with chromatic gradients

Figure 2.8: The figure shows how the SOM graphically displays the learned maps.

the one shown in Figure 2.6c. The original map includes nodes with different colors expressing a numerical quantity. Additionally, each node may also include a certain kind of special symbol in its center, useful for representing other extra features of the original data. In the examples shown here, some nodes include a black dot and the size of a dot is representative of the magnitude of the respective feature. Tools of this type are especially useful for visualizing and understanding the map learned from datasets of high dimensionality.

2.5.6 Topology Preservation

The SOM possesses the amazing ability of preserving the essential properties of the input vectors: similar vectors in the input space will be represented by neighboring units in the neural space. As mentioned earlier, this representation is achieved by mapping the original input space into an output space with a dimension that is usually smaller, typically, a two-dimensional or a three-dimensional space.

According to the authors of [99], there are three different criteria by which one can evaluate how good a map is. The first criterion indicates how *continuous* the mapping is, meaning that input signals that are close (in the input space) should be mapped to codebooks that are close in the output space as well. A second criterion involves the *resolution* of the mapping. Maps with high resolution possess the additional property that input signals that are distant in the input space are represented by distant codebooks in the output space. A third criterion involves the *accuracy* of the

mapping to reflect the *probability distribution* of the input set

A variety of measures for quantifying the quality of the topology preservation have been reported [8] The author of [150] surveyed a number of relevant measures for the quality of maps According to this author, the state-of-the-art measures include the Quantization Error, the Topographic Product, the Topographic Error and also the Trustworthiness and Neighborhood Preservation

The Quantization Error is a measure that quantifies the resolution of the map It is calculated by averaging the distance between each data vector and its BMU The Topographic Error [99] measures the topology preservation by calculating the fraction of input vectors for which the first and the second BMUs are not direct neighbors in the SOM lattice

The Topographic Product [18] is a measure which is sensitive to violations of the neighborhood ordering This measure is computed as follows First, the values $P1$ and $P2$, which quantify the distortion in the input and output spaces, respectively, are computed A value $P3$, which receives as parameters a codebook being evaluated and the number of s nearest neighbors, is obtained by calculating the geometric mean between $P1$ and $P2$ The Topographic Product, P, results by averaging the logarithm of the $P3$ values Values of P close to 0 express good topographic preservation, while $P < 0$ indicates that the number of codebooks is too small Lastly, $P > 0$ implies that the number of neurons is too large for the manifold being represented

The Trustworthiness [171] of the neighborhoods is quantified by measuring how far the new data points that enter a neighborhood are from the original neighborhood Its value is computed separately for every rank order s in a manner analogous to the one employed for the Topographic Product The Trustworthiness ranges from 0 to 1, where a value of 1 denotes a perfect preservation of the topology

The ordering of the weights (with respect to the position) of the neurons of the SOM has been proved for unidimensional topologies [158, 103, 42] Extending these results to higher dimensional configurations or topologies leads to numerous unresolved problems First of all, the question of what one means by "ordering" in higher dimensional spaces has to be defined Further, the issue of the "absorbing" nature of the "ordered state" is open Budinich, in [35], explains intuitively the problems

related to the ordering of neurons in higher dimensional configurations Huang *et al* [88] introduce a definition of the ordering and show that even though the position of the codebook vectors of the SOM have been ordered, there is still the possibility that a sequence of stimuli will cause their disarrangement Some statistical measures of correlation between the measures of the weights and the distances of the related positions, have been introduced in [17]

2.5.7 Applications

The applications of the SOM are numerous, a compendium with all the articles that take advantage of the properties of the SOM is surveyed in [97, 142] These survey papers classify the publications related to the SOM according to their year of release The report [97] includes the bibliography published between the year 1981 and 1998, while the report [142] includes the analogous papers published between 1998 and 2001 A forthcoming manuscript including the related work up to the year 2005 [149] will be published presently The publications mentioned in these surveys report a host of application domains, including Image and Video Processing [163], Pattern Recognition (PR) [3, 140, 144, 161, 164], Multidimensional Scaling (MDS) [30], Information Retrieval (IR) [110], Data Mining [120], Forecasting [169], Artificial Intelligence [109, 117], Engineering, Medicine, etc [97, 142, 149] It is fair to state that numerous researchers have attempted to produce enhanced variations of the basic algorithm since its pioneering introduction into the scientific domain, c f , [50, 91, 147]

We conclude this brief subsection by mentioning that the enhanced version of the SOM that we present in this Thesis, can hopefully be used, advantageously, for *all* these application domains

2.5.8 Known Drawbacks

Although the SOM has demonstrated an ability to solve problems over a wide spectrum, it possesses some fundamental drawbacks For instance, the user must specify the lattice *a priori*, which has the effect that the user must run the ANN a number

of times to obtain a suitable configuration Other handicaps involve the size of the maps, where a lesser number of neurons often represent the data inaccurately The rest of this section will focus on the details of the most relevant handicaps encountered in the traditional SOM algorithm

Although these drawbacks are tackled in various research studies (including ours), there is no universal solution that resolves them all

Complexity Associated to the BMU Search

An indispensable task required by the SOM philosophy is the one of identifying the BMU The BMU search process is described in Section 2 5 2 This process consists of establishing the neuron that best represents the input sample presented to the network As the CL process dictates, the whole set of neurons must be examined so as to determine the above-mentioned "most fitting" unit Assuming that the measurement of the distance between a neuron and a sample point is done in constant time, the total time required to determine the BMU is linear Another important issue is that the task of determining the BMU is required in every training step, implying that the BMU search process becomes one of the most expensive ones associated with the SOM strategy

The cost associated with the search for the BMU has led to research in the direction of creating new SOM variants with a "boosted" search procedure that decreases the time required for training the map [105] These "boosted" versions invariably involve sub-optimal solutions to the nearest neighbor problem, and often take advantage of a hierarchical arrangement of the neurons

Lack of Mathematical Foundation for the 2-Dimensional Case

The desirable feature of the SOM, namely, to yield prototypes which converge in distribution was a landmark, and the formal results which prove that the stochastic distribution of the prototypes follows the distribution of the underlying data points is available However, this formal result is true only if the prototypes are linearly arranged [103], implying that the concept of the direct neighborhood and the BoA

are also *linear* This statement, of course, needs clarification For example, if the nodes are arranged in a lattice, the neighbors of any node $\langle i, j \rangle$ are the nodes $\{\langle i - 1, j \rangle, \langle i + 1, j \rangle, \langle i, j - 1 \rangle, \langle i, j + 1 \rangle\}$, which, on a more critical inspection, can be seen to be those "linearly" close to $\langle i, j \rangle$

Convergence to No-density Areas

Although the topology of the SOM is able to mimic the distribution of the structured input vectors, it may also introduce several false representations outside of the distribution of the input space

According to the authors of [82], it is important to prioritize the formation of a model that effectively spans the data in problems involving high-dimensional feature spaces with sparse data structure

The SOM is sensitive to the original location of the codebook vectors If the neurons fall in areas of low density of data points, they risk not being selected as BMUs, not being migrated as part of the BoA of other units, and finally, when the radius becomes sufficiently small, some neurons may fall in areas with no data points, and are thus "abandoned" for the rest of the training process

Setting the Correct Parameters

The map learned by the SOM is quite dependent on the definition of the schedule of its parameters Usually, poorly-defined parameters will lead to ill-formed maps This handicap can be overcame by running the algorithm multiple times, utilizing different schedules for the parameters However, when the dataset is very large, training the map might be costly, and so it is desirable that the number of experiments required so as to reach good parameter settings, is minimal

Imposing a Pre-defined Structure

In [103], the book that first introduced the SOM (see page 86), Kohonen presented a map in which some of the codebook vectors did not fall in areas belonging to the data distribution possessed by the structure of the lattice itself He stressed the fact that

this should not be interpreted as an error since the SOM represents a non-parametric *regression* However, subsequent authors [69] have questioned the necessity of the SOM requiring a pre-defined lattice which remains static throughout the training phase These critics mainly focus on the very nature of the SOM algorithm, which, in essence, is unsupervised These arguments usually go in the direction that the topology of the network itself, should be learned through self-organization, which is a central theme of this thesis too!

2.6 Common Constituents of the SOM Variants

As seen in Section 2 5 7, the power of the SOM has been demonstrated by its numerous applications in a variety of fields However, as explained in Section 2 5 8, it possesses a number of known handicaps Various researchers have considered these disadvantages, and consequently invented a series of variants of the original SOM philosophy It is possible to identify a series of components (or characteristics) that are shared by all these variations This section describes these modules, which can be perceived to be the basic "building blocks" of the entire family of SOMs described in the literature

2.6.1 The BMU Counter

The so-called "conscience" [51] is a counter recording the number of times that a specific neuron has been reported to be the BMU Every time a unit is chosen to be the most representative one, the respective counter is incremented by unity

Formally, we define a vector $\tau = \langle \tau_1, \tau_2, \quad , \tau_m \rangle$, where τ_i is associated with the neuron $\mathbf{c}_i$ In general, when unit $\mathbf{c}_i$ becomes the BMU, the counter τ_i is updated by Equation (2 17)

$$\tau_i \leftarrow \tau_i + 1 \tag{2 17}$$

2.6.2 The "Age" Counter

Distinct from the BMU counter is the so-called "age" counter, which keeps track of the total number of iterations for which a particular neuron has been "alive". Formally, we have a sequence $a = \langle a_1, a_2, \ldots, a_m \rangle$, where a_i is associated with neuron $\mathbf{c}_i$. In general, a_i is initialized with a value of zero when the neuron c_i is created. Each time a training iteration is performed, the a values are increased as per Equation (2.18):

$$a_i \leftarrow a_i + 1. \tag{2.18}$$

In certain strategies, as recommended by the authors of [127, 71], rather than keeping record of the age of the neurons, one registers the age of the *connections*. In theses cases, the counter of "ages" is associated with the connection between two neurons, but the mechanism for increasing the age is similar to the one given by Equation (2 18). Further, Equation (2.18) is modified as follows:

$$a_{ij} \leftarrow a_{ij} + 1, \tag{2.19}$$

where, a_{ij} is the age counter associated to the edge between neuron i and j.

2.6.3 Sum of the Squared Distance

This special counter aims to keep track of how accurate a certain node is in representing the signals. When a stimulus is presented, the unit that becomes the BMU, increments this counter by adding the square distance between the position of the neuron in the features space and the current input signal.

Formally, this implies the definition of a sequence $\mathbf{e} = \langle e_1, e_2, \ldots, e_m \rangle$, where e_i is associated with the neuron $\mathbf{c}_i$. In general, when unit $\mathbf{c}_i$ becomes the BMU, according to Equation (2.11), the counter e_i is updated as follows:

$$e_i \leftarrow e_i + d^2(\mathbf{w}_i, \mathbf{x}), \tag{2.20}$$

where $d(\cdot, \cdot)$ measures the "distance" between a weight vector and the stimulus. If the metric is the Euclidean distance, Equation (2.20) can be rewritten as:

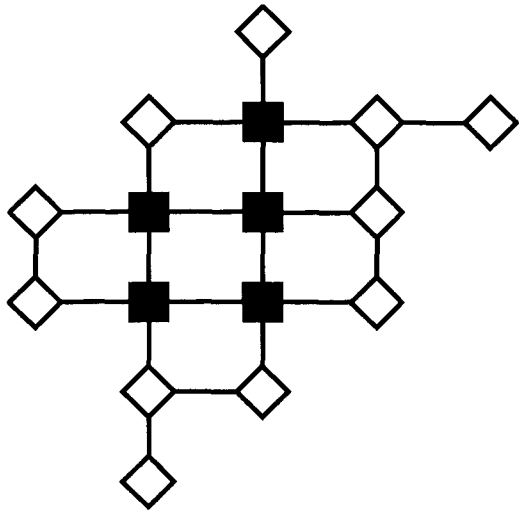


Figure 2.9: An irregular two-dimensional grid. Boundary units are specified with empty diamonds, while Inner units are denoted with filled squares.

$$e_i \leftarrow e_i + \sum_k (w_{ik} - x_k)^2, \tag{2.21}$$

where w_{ik} represents the k^{th} component of the weight $\mathbf{w}_i$.

2.6.4 Boundary Unit

A boundary node is defined in [27] as any node in the grid possessing at least one neighbor in its direct proximity in the two-dimensional lattice which is not already occupied by a neuron. Figure 2.9 depicts a 2-dimensional grid that includes boundary nodes which are specified with empty diamonds, as well as the so-called Inner-units indicated with filled squares. In contrast to the boundary units, Inner nodes have all their four neighboring spots occupied, and thus, are not allowed to be further expanded.

Observe that a node in the grid is allowed to have a maximum of four neighbors. Also, if the initial structure corresponds to a 2×2 grid, each neighboring unit will have at least one neighbor, and thus, also possess *at most* three empty neighboring

positions, as stated in [5]

2.6.5 Frozen Unit

Traditionally, strategies that utilize the SOM paradigm, utilize an updating process which involve the competition of all the units This can be altered by excluding units from the competition The authors of [105] introduced the concept of "*frozen*" nodes, which can be categorized as follows After a node is trained for a certain amount of time, it becomes static It then enters a "frozen" state during which it is not trained anymore

Strictly speaking, the fact that a neuron is static and that a criterion is used to decide when it should become static, are independent The definition provided in [105] implicitly depends on the BMU counter (described in Section 2 6 1) associated with each unit However, other criteria which also involve the "age" of the unit are also conceivable, or those that involve a function dependent on a Quantization Error (QE) measure

The concept of utilizing frozen nodes has found special application in strategies involving a tree-based structure, where the trees are trained starting from the root Whenever the unit becomes static, it allows subsequent units, i e , the direct children, to be trained Examples of strategies performing adaptations of this kind are found in [105, 143], and will be explained in a subsequent section

2.6.6 The Find-Best Search Procedure

The search for the BMU is one of the most expensive tasks in the SOM A remarkable amount of work has been invested in developing alternative solutions to the brute force method, one of which is the so-called Find-Best search procedure introduced in [105] The Find-Best, specified in Algorithm 4, is a heuristic search algorithm for rapidly identifying a winner neuron in a tree It starts from the root and recursively traverses the path towards the leaves, employing the concept of the frozen neuron (described above) If the unit currently being analyzed is frozen, the algorithm identifies the child which is closest to the stimulus, and performs a recursive call The algorithm

terminates when the node currently being analyzed is not a frozen node (i.e., it is currently being trained), and is returned as the BMU. In this way, frozen nodes serve as junction for performing the search. A relatively similar method is also utilized in [143], except in the case of the latter, the researchers use a dynamically growing tree.

Algorithm 4 find_best(**p**,**x**)

```
1 if p is frozen then
2    Identify c, the child of p that is closest to x.
3    find_best(c,x)
4 else
5    return p
6 end if
```

2.7 Non Tree-based SOM Variants

We shall now visit some of the most relevant variants of the SOM. As the focus of the present Thesis is on tree-based variants of the original philosophy, we have chosen to dichotomize our discussion based on two families, each possessing their own respective properties, namely by those strategies that do not use a tree as a their underlying data structure, and those that do. We start by describing the non tree-based SOM variants.

2.7.1 Growing Cell Structures (GCS)

In the Growing Cell Structures (GCS) method [69], the author addressed the difficulty of choosing a suitable imposed topology, and the necessity of defining the so-called decaying schedule for a set of parameters.

For each neuron, the strategy defines a BMU counter as specified in Section 2.6.1. Fritzke [69, 68] also noted an interesting phenomenon: Given that the neurons are in continuous movement, signals that are perceived more recently should be weighted more than those that triggered excitation in the past. Based on that reasoning, during each training step, the strategy decreases the counter by a fraction. Using such a counter, the *relative signal frequency* is built as per Equation (2.22):

$$f_i = \frac{\tau_i}{\sum_{j=1}^{m} \tau_j} \tag{2.22}$$

Algorithm 5 describes the GCS thoroughly. Initially, a pre-specified number of neurons are placed in the input space. At each step of the learning process, a stimulus is generated, and the BMU is determined as per Equation (2.11), after which the direct topological neighbors are moved towards the input sample. After a fixed number of iterations, an additional neuron is added between the unit possessing the largest relative signal frequency and its farthest direct neighbor. When a new unit is added to the system, an adjustment of the τ counters of all the direct neighbors of the newly inserted neuron are affected as per Equation (2.23).

$$\Delta\tau_c \leftarrow \frac{F_c^{(new)} - F_c^{(old)}}{F_c^{(old)}} \tau_c, \forall c \in N_r, \tag{2.23}$$

where F_c is the n-dimensional volume of F_c.

Lastly, the τ counter of the new unit is initialized using Equation (2.24).

$$\tau_c \leftarrow -\sum_{c \in N_r} \Delta\tau_c \tag{2.24}$$

Some advantages of the GCS include its ability to estimate the topology, as opposed to the properties of other approaches such as the basic SOM which require an imposed topology. The authors of [69] also remark that the algorithm uses parameters that are constant, i.e., the parameters do not decay during the execution of the algorithm.

2.7.2 Neural Gas (NG)

The Neural Gas (NG) [127] scheme is a NN that arranges itself in an unsupervised manner. It uses an update rule that affects not only the most representative neuron but also the nodes in its proximity using a "winner takes most" approach. As a result of this process, a series of connections between the nodes are obtained, where the final configuration of the connections between them is used to reflect the topology of the data distribution.

Algorithm 5 GCS($\mathcal{X}$,η_b,η_n,λ,α)

Input:

i) $\mathcal{X}$, the input sample set
ii) η_b, the learning factor for the BMU
iii) η_n, the learning factor for the direct neighbors of the BMU
iv) λ, number of iterations required before a new neuron is inserted
v) α, fraction by which the BMU counter, τ, is decreased in each training step

Method:

Initialize the codebook vectors
repeat
  Obtain a sample $\mathbf{x}$ from $\mathcal{X}$
  Identify the BMU as per Equation (2 11)
  Update the BMU using Equation (2 16), where $\eta = \eta_b$
  Update the direct neighbors of the BMU as per Equation (2 16), with $\eta = \eta_n$
  Increment the respective BMU counter as per Equation (2 17)
  if λ iterations have occurred since the last insertion **then**
    Identify the unit p, which is the one with highest relative signal frequency
    Insert a new neuron, r, between p, and q, its farthest direct neighbor
    Connect r with p and q
    Connect r with all the common direct neighbors of p and q
    Remove connection between p and q
    Adjust BMU counter for all the direct neighbors of r as per Equation 2 23
    Initialize BMU counter associated with unit r using Equation 2 24
  end if
  Decrease all the BMU counters by a fraction α
until end criteria

End Algorithm

In the NG approach, the number of codebook vectors must be specified from the beginning Also, the algorithm does not delete or add nodes during the training process, thus maintaining a constant number of neurons during its execution

When a stimulus is presented, the distance to every neuron is determined, thus implying an ordering of the elements and the required adjustments to the weights In this way, codebook vectors that are closer will be more attracted to the input signal

The creation of connections occurs between the two first elements of the ordering, and the "age" of these connections is then reset to zero (the "age" counter is mentioned in Section 2 6 2) When the connection between these two neurons already exists, the "age" of the old connection is rejuvenated A deletion occurs when the age of any given connection reaches a certain threshold, in which case the associated relationship is "forgotten"

In general, the resulting topology is neither a tree nor a grid, but a graph that is, possibly, not fully connected The authors of [127] mention that the most expensive task is the one that involves finding the ordering of the elements, requiring $O(N \log N)$ time They have suggested a scheme to mitigate this time, noting that codebook vectors which are farther apart require small modifications with respect to their weights, allowing the possibility of considering only a certain number of relevant codebook vectors to be involved in the sorting process, thus speeding-up the execution time The NG is described in detail in Algorithm 6

2.7.3 Growing Neural Gas (GNG)

The Growing Neural Gas (GNG) is a self-organizing NN proposed in [71] It successively adds new units to an initially small network by evaluating local statistical measures gathered during previous steps This strategy is similar to the GCS covered in Section 2 7 1, but unlike the later, a fixed topology is not imposed The GNG algorithm allows the deletion of the connections that are not used often, permitting the elaboration of isolated subgraphs The deletion of a connection occurs in a way similar to the NG (discussed in Section 2 7 2) It considers the so-called "age" counter (defined in Section 2 6 2) to determine when a given connection should be "forgotten"

Algorithm 6 NG($\mathcal{X}$,η_i,η_f,h_i,h_f,T,θ)

Input:

i) $\mathcal{X}$, the input sample set.
ii) η_i, the initial learning rate.
iii) η_f, the final learning rate.
iv) h_i, the initial value of the parameter h, computed in each step and which determines the number of neurons significantly changing their synaptic weights.
v) h_f, the final value for h, defined in an analogous manner as for h_i.
vi) T, the total number of training iterations.
vii) θ, the maximum lifetime for the synaptic connections.

Method:

$t \leftarrow 0$
Initialize the weights $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$, by randomly selecting elements from $\mathcal{X}$.
repeat
  Obtain a sample $\mathbf{x}$ from $\mathcal{X}$.
  Rank the neurons according to their proximity to $\mathbf{x}$, remembering the rank position k for each neuron. $R = \langle s_1(\mathbf{x}), s_2(\mathbf{x}), \ldots, s_m(\mathbf{x}) \rangle$
  $h \leftarrow h_i(h_f/h_i)^{t/T}$
  $\eta \leftarrow \eta_i(\eta_f/\eta_i)^{t/T}$
  for all $\mathbf{w} \in \mathcal{W}$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} + \eta e^{-k/h}(\mathbf{x} - \mathbf{w})$
  end for
  Rejuvenate the connection between $s_1(x)$ and $s_2(x)$, setting its age to 0.
  Increase the age of all the connections of $s_1(x)$
  Remove connections of $s_1(\mathbf{x})$ exceeding a lifetime θ
  $t \leftarrow t + 1$
until a stop criteria is reached

End Algorithm

A connection is deleted when its age reaches a pre-specified threshold Consequently, isolated neurons are removed from the network as well

Algorithm 7 describes the GNG in detail When a new stimulus is presented, the connection between the BMU and the second closest unit is strengthened, thus crystallizing the Hebb's principle of reinforcement learning (discussed in Section 2 4 4) The "age" of this connection is reset to zero When the connection already exists, its "age" is rejuvenated

The GNG strategy also incorporates an insertion mechanism which is triggered after a fixed number of learning iterations For this purpose, an error counter storing the accumulated squared error is associated to each neuron (as discussed in Section 2 6 3) During the insertion process a new unit is incorporated between the BMU and its neighbor with the largest accumulated error The newly inserted neuron is then placed in the middle of the path (in the feature space) separating the above-mentioned neurons The errors of the three neurons are then updated to reflect the changes (see Algorithm 7 for details)

At the end of each iteration, the errors variables for all the units are decreased by a constant factor, and the process is repeated again until a stop criterion is reached

The author of [71] noted the fact that in the GNG all the parameters are constant over time (as opposed to the original Kohonen Networks [103]) He also stressed that the GNG strategy eliminates the necessity of specifying the network size *a priori*

2.7.4 Growing Grid (GG)

The Growing Grid (GG) is a SOM variant that is able to grow as training proceeds Initially the GG starts with a 2×2 grid and dynamically adds entire rows or columns iteratively

During each adaptation step the BMU is identified, and a counter associated to that unit is increased by unity, according to Equation (2 17) This counter corresponds to the BMU counter defined in Section 2 6 1 Additionally, the GG utilizes the so-called city-block distance for determining the subset of neurons in the proximity of the BMU, i e , those which are to be migrated towards the stimulus

Algorithm 7 GNG($\mathcal{X}$,η_b,η_n,λ,θ,η_i,η_e)

Input:

i) $\mathcal{X}$, the input sample set
ii) η_b, the learning factor for the BMU
iii) η_n, the learning factor for the direct neighbors of the BMU
iv) λ, number of iterations required before a new neuron is inserted
v) θ, the maximum lifetime for the synaptic connections
vi) η_i, The constant factor by which are decreased the two direct neighbors of the node being inserted
vii) η_e, the multiplying factor by which the error variables, e_i , are decreased

Method:

1 Initialize the weights $\mathbf{w}_a$ and $\mathbf{w}_b$, e g , by randomly selecting samples from $\mathcal{X}$
2 **repeat**
3 Obtain a sample $\mathbf{x}$ from $\mathcal{X}$
4 Determine $s_1(\mathbf{x})$ and $s_2(\mathbf{x})$ using Equation (2 11)
5 Increase the "age" of connections emanating from s_1 using Equation (2 18)
6 Accumulate the error counter of s_1 as per Equation (2 20)
7 Update $s_1(\mathbf{x})$, i e , the BMU, by using Equation (2 16), where $\eta = \eta_b$
8 Update the direct neighbors of $s_1(\mathbf{x})$ using Equation (2 16), where $\eta = \eta_n$
9 Rejuvenate the connection between $s_1(x)$ and $s_2(x)$, setting its age to 0
10 Remove connections with age counter grater than θ
11 Remove neurons with no emanating edges
12 **if** number of generated input signals is multiple of λ **then**
13 Find the unit q with greater accumulated error
14 Find f the neighbor of q, with largest accumulated error
15 Create a unit r between q and f placing it in the halfway
16 Connect q with r, and also create a connection between r and f
17 Remove the connection between q and f
18 Decrease the errors of q and f by the constant factor η_e
19 Initialize the error of r, assigning the error value of unit q
20 **end if**
21 Decrease all error variables by a constant factor η_e
22 **until** stop criteria is fulfilled

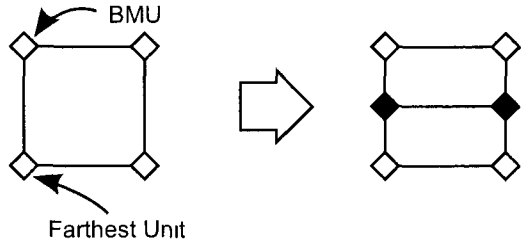


Figure 2 10 Insertion of a new row in the GG The BMU and its farthest neighbor determine where a new row must be inserted The initial grid is shown in the left, while the grid after the insertion is depicted on the right

An insertion takes place after a fixed number of training steps The unit that has been the BMU more often is identified, as well as its farthest neighbor in the *feature* space Then, the connection between the units determine whether to insert a new column or row, as illustrated in Figure 2 10 The position of a newly inserted unit will be at the point that equally divides the line segment connecting the two neurons between it

The training process is repeated until a stop criteria has been fulfilled, e g , the number of units in the grid reaches a certain threshold

2.7.5 Incremental Grid Growing (IGG)

The IGG [27] is a SOM-based NN that constructs its network incrementally by adapting its structure to the data distribution Initially, the IGG, starts with a small number of neurons (2×2) , and the structure is expanded using a insertion procedure

The insertion of a new node occurs in the so-called boundary node[7] having the largest QE Consequently, each neuron maintains an error counter which is updated as per Equation (2 20) Thus, as insertion occurs only on the free spots of the boundary nodes, the network preserves its 2-dimensional structure at all time, providing to the IGG visualization capabilities analogous to the ones presented in the SOM

Another important step of the algorithm consists of an examination of the direct

[7]Boundary nodes are described in Section 2 6 4

neighbors in the grid. Those units that are close neighbors in the feature space generate a connection, while those connections between distant neighbors in the feature space, are "forgotten".

2.7.6 Growing SOM (GSOM)

The Growing SOM (GSOM) [5], is a SOM-based NN capable of growing its structure as learning proceeds. The main training process is composed by three parts, namely the initialization, growing and smoothing phases.

The GSOM is initialized with four neurons, arranged in a 2×2 grid. Each of these units resembles the concept of a boundary node utilized in the IGG. The starting location of the neurons are computed by randomly selecting samples from the input data set. During this phase, the so-called Growth Threshold (GT) is computed. The GT depends on a parameter known as the spread factor, which controls the spread of the generated map. The spread factor is expected to be close to zero in the initial iterations, and, as time proceeds, its value is increased in order to explore new regions of the data distribution.

In each iteration of the growing phase, a signal is presented to the network, the BMU is computed, and the adaptation of the weights of the BMU and its vicinity is performed. It is in this phase where the grid in augmented. To determine whether to insert new nodes, an error measure is maintained for each neuron, which is updated each time a unit becomes the BMU according to:

$$e_i \leftarrow e_i + \sqrt{\sum_k (\mathbf{w}_{ik} - \mathbf{x}_k)^2}, \tag{2.25}$$

which differs from the error measure utilized in the IGG since it involves the *square root* of the summation.

The unit with the largest error is identified and if this error exceeds the GT, all its free positions will generate new nodes. The initialization of their respective weights depends on their position in the grid, and are determined by a series of cases which are discussed in detail in [5], but excluded here in the interest of brevity. The growing phase finishes when the growth of nodes decreases significantly.

In the smoothing phase, a finer weight adaptation is performed The learning factor is initialized at a lesser value in comparison to the growing phase and its decaying schedule is also reduced Moreover, the BoA in this phase is limited only to the direct neighborhood, and no addition of new units is permitted

2.8 Tree-Based SOM variants

While Section 2 5 reviewed the SOM algorithm, and Section 2 7 focused on variants of the original SOM strategy that do not employ a tree as the underlying structure, this section discusses the most relevant tree-based SOM philosophies

2.8.1 Tree-Structured VQ (TSVQ)

The Tree-Structured VQ (TSVQ) algorithm [105] is a SOM variant, where codebook vectors are arranged in a tree structure which is fixed in depth and in breath The training first takes place at highest levels of the tree The TSVQ incorporates the concept of a "*frozen*" node (discussed in Section 2 6 5), which implies that after a node is trained for a certain amount of time, it becomes static The algorithm then allows subsequent units, i e the direct children, to be trained, considering one level of the tree at a time The training process, which is essentially SOM-based, terminates when all the units are frozen In order to identify the unit to be frozen, the strategy utilizes a BMU counter (mentioned in Section 2 6 1) associated with each neuron When the respective BMU counter reaches a certain threshold, the neuron becomes static, allowing the training of its children It should be mentioned that in the case of the TSVQ, competition takes place by involving only a limited subset of the codebook vectors This process is repeated until all the nodes become static The algorithm outputs the values of the codebook vectors, which asymptotically represent the data distribution

Some advantages of the TSVQ algorithm are its topology preservation within the input space, its multi-resolution, and the fact that on convergence, no nodes are found in areas of zero-density As opposed to this, one disadvantage is the fact that

frozen units do not have the capability of being trained anymore The authors of [105] mention that the tree structure may be dynamic by stating that "*the dynamic hierarchy may expand in depth or in breath according to a certain criterion*", thus implying that the dynamism is addressed or quantified only in terms of expansion i e by adding elements to the tree structure However, the details of how this can be achieved have not been specified and are thus left open Certainly, other ways to incorporate dynamism can be conceived, for example, through the modification of the underlying connectionist structure between the existing neurons

The details of the TSVQ are given in Algorithm 8 The parameters of the algorithm are, η, the learning rate and, θ, the parameter specifying the number of steps necessary for "freezing" a unit

At the beginning of the training process, the codebooks are placed in the feature space by randomly selecting samples from the input set At first, only the root node is trained, migrating it towards the stimulus, according to Equation (2 16) Once the root becomes frozen, the children are placed in the same location as their parent, setting their BMU counter to zero The process is repeated recursively until all the nodes become static

To identify the BMU, the scheme incorporates Algorithm 4, which is a tree-search scheme described in Section 2 6 6

Koikkalainen and Oja, in the same paper [105] refine the idea of the TSVQ by defining the TSSOM, which inherits all the properties of the TSVQ, but redefines the search procedure and BoA In the case of the TSSOM, SOM layers of different dimensions are arranged in a pyramidal shape, as shown in Figure 2 11 (extracted from [105]) The search procedure is essentially dominated by Algorithm 4, but slightly differs from the TSVQ, in the sense that once a winner in a layer is found , the direct proximity is examined to check for the BMU On the other hand, the BoA differs in that, instead of considering only the BMU, its direct neighbors will also be considered

The main ideas included in the TSVQ strategy are important, because are further exploited by other strategies such as [143], which will, in turn, be described in detail in this Thesis

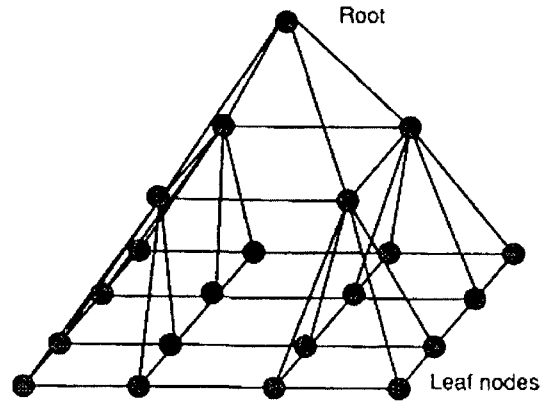


Figure 2 11. The structure of the TSSOM.

Algorithm 8 TSVQ($\mathcal{X}$,η,θ)

Input:

i) $\mathcal{X}$, the input sample set.
ii) η, the learning factor
iii) θ, the maximum value for the BMU counter. When this threshold is reached, the respective unit becomes frozen

Method:

Initialize the root, e.g., by randomly selecting samples from $\mathcal{X}$.
repeat
  Obtain a sample $\mathbf{x}$ from $\mathcal{X}$.
  Identify the winner neuron as per Algorithm 4, starting from the root
  Update the current BMU according to Equation (2.16)
  Increment the BMU counter of the winner neuron as per Equation (2.17).
  if the BMU counter of the winner neuron is greater than θ **then**
    The current BMU becomes "frozen".
    The children are initially located in the same position as the BMU.
  end if
until all units are frozen

End Algorithm

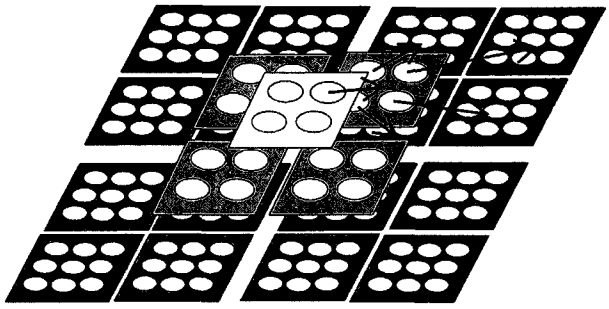


Figure 2.12: The structure of the HFM.

2.8.2 Hierarchical Feature Map (HFM)

The Hierarchical Feature Map (HFM) [134] is a strategy that defines a hierarchical structure of independent SOM layers. The HFM architecture operates with a fixed-structure topology which is defined *a priori*, including the size of the maps in each level of the hierarchy. Further, the algorithm attempts to learn the data distribution from *this* perspective.

The first layer of the HFM is a regular SOM, where each of its neurons are linked to a subsequent SOM layer. This process is recursively performed, defining a fully balanced tree, i.e., one where all the possible paths from the root down to the leaves are equidistant. As an instance, consider Figure 2.12 (adapted from [81]), that shows a HFM with a depth of three levels. The first layer of the structure is a 2×2 SOM grid. The second level of structure is composed by four 2×2 SOM grids, i.e., one layer for each unit from the first level, thus consisting of a total of sixteen units in the second level. The third level, consequently, is composed by sixteen SOM lattices, which in this example are 3×3 grids. The arrows show the correspondence between some of the neurons and their respective SOM-layers.

The training of the tree starts by first "educating" the highest level of the hierarchy. A small grid implies that only a general classification of the patterns is achieved. After the first level is self-organized, all the input signals represented by one of these units are further categorized by a subsequent SOM grid.

2.8.3 Growing Hierarchical SOM (GHSOM)

The Growing Hierarchical SOM (GHSOM) [156] is a dynamically growing NN that expands its structure according to the requirements of the input data. As opposed to the HFM strategy, the hierarchical architecture is dynamic, and not necessarily balanced, i.e., the tree is allowed to grow asymmetrically. The expansion of the structure is dual: The first type of adaptation is conceived by inserting new rows/columns to the SOM grid that is currently being trained, while the second type is implemented by adding layers to the hierarchical structure. Both types of dynamism depend on the verification of quantization measures. When the SOM grid currently being trained finishes its growing process, the terminating criterion is verified. Units that are still under-representing the data are further expanded, generating subsequent layers, and this process is continued recursively.

The quality of the map is measured in terms of the distance between the prototypes and its respective input signals represented by the BMUs. Specifically, the authors of [156] discuss the Mean Quantization Error (MQE) and the absolute value, i.e., the Quantization Error (QE) of a unit. Further, they recommend to use the QE as the basis for the GHSOM training. The QE is defined according to Equation (2.26), as follows:

$$q_i = \sum_{\mathbf{x}_j \in \mathcal{X}_i} \|\mathbf{w}_i - \mathbf{x}_j\|, \tag{2.26}$$

where $\mathcal{X}_i \subseteq \mathcal{X}$ is the set containing all the input signals for which the neuron i is the BMU. In particular, the QE for the single unit in the highest level of the hierarchy is computed as per Equation (2.27):

$$q_0 = \sum_{\mathbf{x}_j \in \mathcal{X}} \|\mathbf{w}_0 - \mathbf{x}_j\|. \tag{2.27}$$

Repetitively, while the current SOM layer is being trained, after a fixed number of training iterations, the unit with the largest QE and its most dissimilar neighbor are selected. The expansion mechanism consists of inserting a new row (or column) between those units, which is a process analogous to the one utilized in the GG and

depicted in Figure 2 10 Moreover, the selected units are expanded only if their QE exceeds a fraction of q_0, as expressed in Equation (2 28) The newly inserted units are initially positioned in the midpoint of their respective neighbors in the lattice

$$q_k < \theta_2 \times q_0, \tag{2 28}$$

After the insertion, the radius and learning factor are re-established to their original values and the training process continues As more neurons are inserted in the SOM, their respective QEs are decreased Consequently, when all units satisfy the required degree of granularity expressed by Equation (2 28), the map stops its expansion process

Now we shall discuss the second type of dynamism, which is the phenomenon of expanding the hierarchy

Once the growth of a map is completed, its units are verified The units whose errors exceed a certain threshold are further expanded, developing a subsequent 2×2 SOM layer The quality of a map is measured in terms of its MQE, which is defined by Equation (2 29)

$$Q_i = \frac{1}{|\mathcal{C}_i|} \sum_{c \in \mathcal{C}_i} q_c, \tag{2 29}$$

where $\bar{Q}_i$ is the MQE of the current *map* being trained, $\mathcal{C}_i \subseteq \mathcal{C}$ is the set of neurons belonging to the SOM i, and q_c is the QE of neuron c In words, Equation (2 29) represents the average of all the QEs of the units belonging to a given map

The GHSOM is described in Algorithm 9 The training process starts by defining a 2×2 SOM in the first layer, which is linked to the virtual unit mentioned above The first layer is then trained according to the traditional SOM strategy After a fixed number of iterations, new rows (or columns) are progressively added until the map attains the desired granularity Once the SOM converges, it may grow, adding new SOM layers to better represent some units of the SOM in more detail, or terminate the training of the current map if the data is well represented The resulting structure is an irregular tree, as shown in Figure 2 13

According to the authors of [145], usually an invariant condition between the

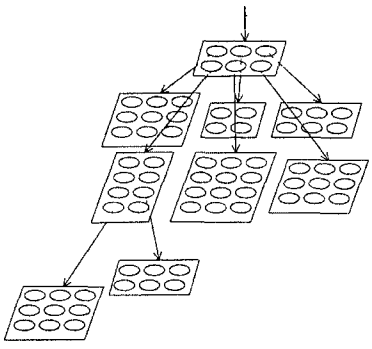


Figure 2.13: Structure of the GHSOM.

parameters θ_1 and θ_2, namely, $0 < \theta_2 \ll \theta_1 < 1$ applies. The authors [145] also state that the choice of the parameter θ_2 is not critical, since it is feasible to adjust it to small values, permitting a subsequent decision of how many layers are required. However, the parameter θ_1 leads to irreversible decisions Generally, adjusting θ_1 to small values leads to a large hierarchy of small SOMs, while setting it to a large value usually implies the generation of large SOMs with a flat hierarchical structure. A solution to this situation is also presented in [145], whose authors recommend that the GHSOM is trained multiple times and manually evaluated until an appropriate value for θ_1 is estimated. This task, as mentioned by other authors (e.g., [143]), may be very time consuming.

2.8.4 Self-Organizing Tree Map (SOTM)

The Self-Organizing Tree Map (SOTM) [82] is a SOM-based strategy designed for data clustering and visualization. It utilizes a tree-based structure that starts with a single node, and which is expanded by incrementally adding neurons throughout the training phase. The resulting tree attempts to reflect the properties of the input manifold.

The SOTM uses a concept known as a Vigilance Test, employed previously in Adaptive Resonance Theory (ART) [37]. According to the authors of [37], a degree of high vigilance enforces the exploration of new categories, as a response to small

Algorithm 9 GHSOM($\mathcal{X}$,η,r,θ_1,θ_2)

Input:

i) $\mathcal{X}$, the input sample set
ii) η, the learning factor
iii) r, the radius of the neighborhood function
iv) θ_1, Fraction of the MQE of the parent unit, used for hierarchical expansion
v) θ_2, Fraction of the QE of the virtual unit, used for inserting rows (or columns)

Method:

1 Create a 2 × 2 SOM associated to the parent layer Initialize the weights of its units by selecting samples from the subset of signals represented by the parent
2 **repeat**
3 Train the map using the SOM algorithm
4 **if** *lambda* iterations has been performed **then**
5 **if** quality of the SOM is not good **then**
6 Insert new row (or column) between the unit with largest QE and its most dissimilar neighbor
7 **end if**
8 **else**
9 Expand units possessing a high QE by creating a new 2 × 2 SOM as in step 2
10 For training this new SOM, consider only the signals represented by its parent unit
11 **end if**
12 **until** end criteria is reached

End Algorithm

differences between the input and what is expected As opposed to this, a degree of low vigilance permits large mismatches, and thus, it leads to a grouping of stimuli that presents a looser degree of similarity

In the SOTM, the range is controlled by the so-called "distance threshold", θ When the distance threshold is less than the distance between the stimulus and the BMU, a new child node is attached to the BMU, locating it in the same position as the stimulus Otherwise, i e , when the distance between the input sample and the BMU is within the tolerance threshold, the BMU is migrated according to Equation (2 16) The fact that *only* the winner neuron is moved towards the stimulus, makes the SOTM an example of *hard* CL

The above mentioned training process is repeated until a certain criterion is achieved The author of [82] leaves the question of which is the most appropriate criterion for finishing the training of the NN unanswered Algorithm 10 describes the SOTM in detail

---

Algorithm 10 SOTM($\mathcal{X}$,η,θ)

Input:

i) $\mathcal{X}$, the input sample set
ii) η, the learning factor
iii) θ, the distance threshold

Method:

```
1  Initialize the root node by randomly selecting samples from X
2  repeat
3     Obtain a sample x from X
4     Find the BMU as per Equation (2 11)
5     if distance between BMU and x is less than θ then
6        Update the BMU according to Equation (2 16)
7     else
8        Insert a new neuron, assigning the BMU as its parent, and locating it in the
         same position as the stimulus
9     end if
10 until (Convergence is achieved)
```

End Algorithm

2.8.5 Self-Organizing Tree Algorithm (SOTA)

The Self-Organizing Tree Algorithm (SOTA) [56] is a dynamically growing NN based on the SOM and which take some analogies from the GCS. The SOTA utilizes a binary tree as the underlying structure.

The algorithm considers the migration of the neurons *only* if they correspond to leaf nodes within the tree structure. Initially, the SOTA starts with a root node and its respective left and right children, whose weights are initialized by randomly selecting samples from the unit hyper-cube. In particular, the root of the tree is a non-leaf node since its "conception", and it continues in this "state" until the end of the training process and, consequently, it will never be selected as the BMU.

The SOTA possesses a particular definition for the BoA, which considers the topological neighbors in the *tree* space. This BoA phenomenon includes two different cases. The most general case occurs when the parent of the BMU is not the root, which is a situation where the BoA is composed by the BMU, its sibling and its parent node. In the contrary scenario, i.e., when the parent of the current BMU is the root node, the BoA constitutes the BMU *only*. The BoA of the SOTA is clarified in Figure 2.14.

The training of the tree is performed in a series of epochs. After each epoch, the SOTA triggers a growing mechanism that utilizes an error measure to determine the node of the tree to be split. This error is defined in [56] as the average value of the dissimilarity between a neuron and the stimuli it represents, which is equivalent to the definition of MQE for a particular *neuron*, defined by Equation (2.30):

$$\bar{q}_i = \frac{1}{|\mathcal{X}_i|} \sum_{\mathbf{x}_j \in \mathcal{X}_i} \|\mathbf{w}_i - \mathbf{x}_j\|, \tag{2.30}$$

where $\mathcal{X}_i \subseteq \mathcal{X}$ is the set containing all the input signals for which the neuron i is the BMU.

Further, the neuron possessing the maximum error is split into two new descendants, locating them in the same position as the parent neuron. After the split, the training of the tree is restarted for another epoch. The algorithm finalizes its growing process when all the neuron reach a certain QE. The criterion for terminating the

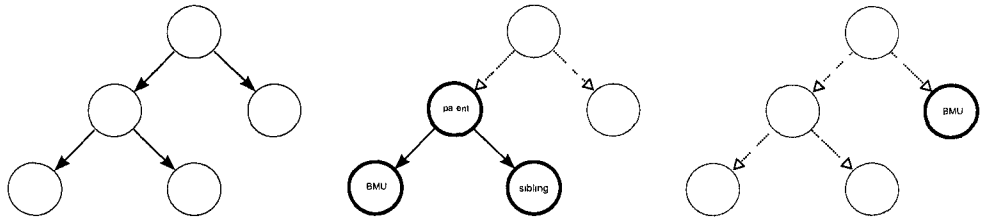


Figure 2.14: The BoA for the SOTA. Figure 2.14a shows a binary tree with five neurons. Figure 2.14b depicts the BoA when the parent of the BMU is different from the root of the tree. Figure 2.14c illustrates the BoA when the parent of the BMU is the root of the tree.

training process is the so-called *total* QE of the map, defined as the summation of the distances between each stimulus and its respective BMU, and given by Equation (2 31):

$$Q = \sum_{c \in \mathcal{C}} q_c, \tag{2.31}$$

where q_c is the QE of neuron c, calculated as per Equation (2.26)

2.8.6 Evolving Tree (ET)

The Evolving Tree (ET) is a tree-based self-organizing NN especially designed for learning large databases in an unsupervised manner [143]. In the ET, the neurons are arranged in a tree topology that is allowed to grow when any given node receives a certain number of hits from the training vectors, which is recorded by means of the BMU counter, similar to the GCS and the TSVQ explained earlier.

The ET starts with a single node, and after a certain number of training steps, the node is split. As a result of this process a pre-specified number of new nodes are created and designated to be the *Children* of the split node. The codebook vectors of the children are initialized to that of the parent. Once a node is split, it becomes "frozen" (discussed in Section 2 6.5). The authors of [143] refer to a node of this kind as a "trunk" node, which has the property that it is not eligible as BMU and does not constitute a part of the migration process either. In a manner analogous to

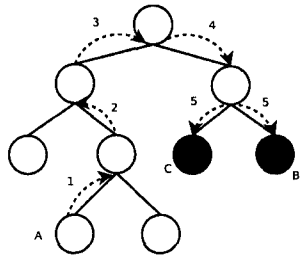


Figure 2 15 An example of the "neighborhood" for the ET, where nodes B and C are equidistant to A In the ET, the definition of the distance pertains only to leaf nodes

SOTA, competition only takes place among the leaf nodes, while the function of the trunk is to maintain the connections with the children The above process is repeated recursively, thus generating a tree capable of growing indefinitely

Each time the network receives an stimulus, the search for the BMU starts at the root node, examines the corresponding children, and selects the most suitable neuron When the unit that best represents the input sample is a leaf node, the search terminates, otherwise, its children are examined and the process is repeated, in a manner similar to the one employed by the TSVQ and described in Algorithm 4 The ET also incorporates a definition of the "neighborhood" which only considers leaf nodes, as shown in Fig 2 15 Although the search for a BMU is fast, it does not necessarily lead to the global optimum The issue of using static nodes which cannot be trained, eliminates the possibility of choosing certain nodes, even if they are the most representative solution points for an input sample, thus leading to the possibility of generating suboptimal solutions One of the known problems of the ET is that leaves belonging to different branches may move closer to each other after a certain number of iterations In this way, clusters that are identified at higher levels of the hierarchy of the tree, may not be well represented as one proceeds towards its lower levels

Algorithm 11 ET($\mathcal{X}$,η,r,θ,k)

Input:

i) $\mathcal{X}$, the input sample set
ii) η, the learning factor
iii) r, the radius of the BoA
iv) θ, the split threshold
v) k, the number of children to be created once a neuron is split

Method:

1 Initialize the root of the tree by, e g , locating it in the center of mass of $\mathcal{X}$
2 **repeat**
3 Find the BMU as per Algorithm 4
4 Update all the leaves within a radius r within the BMU using Equation 2 16
5 Increment the BMU counter of the winner according to Equation 2 17
6 **if** the BMU counter of the winner exceeds the threshold θ **then**
7 Split the winner neuron, generating k children and initialize their respective weights to that of the parent neuron
8 **end if**
9 **until** an end criterion is reached, such as a total number of training epochs

2.9 Comparison Between the Different SOM-Based Strategies

The previous sections has been dedicated to the description of the SOM algorithm, as well as its most relevant tree-based and non-tree based variations The constituting parts that are common to the different SOM-based strategies were described in a separate section We shall now discuss the properties of these SOM-based strategies from a higher level of abstraction, making explicit their most relevant similarities and differences

We start the comparison by subdividing these "building blocks" into six different categories, which include the topology, the type of dynamism displayed, the manner in which the BoA is calculated, the additional counters utilized, the states assumed by a neuron, and finally, the type of search procedure utilized for identifying the BMU

2.9.1 Topological Structure

The first subdivision corresponds to the so-called topology The topological structure of a NN refers to the type of structure that is formed when the neurons are interconnected From the strategies discussed in the previous section, we can extract five different types of topology a grid, a tree, a (possibly unconnected) graph, an irregular grid and a hierarchical structure built with SOM-layers

A grid topology is usually a two-dimensional (but may also be three-dimensional, or even unidimensional) array of interconnected neurons The traditional SOM and the GG are examples of strategies using this kind of topology

Distinct to the grid configuration is the tree topology In the SOM variants using a tree-based topology, the neurons are arranged in a hierarchical manner, starting from the root which is located at the top level of the hierarchy Examples of SOM-based variants that utilize a tree as their underlying topology are the TSVQ, the TSSOM, the HFM, the GHSOM, the SOTM, the SOTA, and the ET

Another topological structure is the one referred to as SOM-layers This type of topology is characterized by having an original topological structure, where the units being connected correspond to SOM grids themselves Among the strategies described in the previous sections, we can observe that some of them use a tree of SOM-layers, instead of using a tree of neurons Strategies possessing such a topology are the TSSOM, the HFM and the GHSOM

A different type of topology occurs when the grid maps are allowed to be irregular, that is, a partially complete grid In an irregular grid, the topology is "almost" a grid, but some of its nodes (and the corresponding connections) are missing An example of an irregular grid is illustrated in Figure 2 9 Strategies possessing this type of arrangement are the IGG and the GSOM

When none of the topologies mentioned earlier is employed, we say that the topology pertains to the most general type of structure, i e , a graph, which may not be fully connected In general, any type of topology may involve more than a single intra-connected subset, which, in turn, may not have any mutual inter-connections

2.9.2 Dynamism

The different SOM variants can incorporate the concept of dynamism in many forms We now attempt to categorize the different types of adaptation utilized by the strategies discussed in the previous sections

The most basic type of dynamism occurs when there is no dynamism at all, that is, the structure remains static throughout the learning phase Necessarily, strategies using static structures require the definition of the topology *a priori*, which can be supplied as a parameter The traditional SOM itself belongs to this family, since the grid structure is defined from the beginning and it is, thereafter, maintained "intact" until the termination of the learning phase Other strategies which utilize static structures are the TSVQ, the TSSOM, and the HFM

A basic type of adaptation is produced by adding and deleting connections to a set of neurons in the system NNs in this class maintain the number of neurons constant The sole example of the NNs visited in the previous section which possess this property is the NG

A different adaption process is the one that allows the addition of new neurons and connections When no deletions of nodes or edges is permitted, one obtains a structure which monotonically grows as the training proceeds Instances of this type of growing process can be found in the GG, the GSOM, the GHSOM, the SOTM, the SOTA and the ET

When, apart from the ability of adding neurons and connections, the strategy incorporates the deletion of both neurons and edges, the structure does not grow monotonically Rather, under certain circumstances, it is possible that the structure may be reduced depending on the distribution of the input manifold Since deletion corresponds to a more complex type of adaptation, it implies the incorporation of a module that is capable of identifying the units that over-represent a certain region A subset of non tree-based strategies that perform this type of adjustment include the GCS, the GNG and the IGG It is important to mention that *all* of the tree-based strategies discussed in the previous sections avoid such a deletion process

It is also worth mentioning that the TTOSOM incorporating the Conditional Rotations (TTOCONROT) strategy, which will be explained in depth in Chapter

5, explores a new type of adaptation which was not utilized by *any* of the strategies included in this survey This type of adaptation involves the techniques defined in the field of Adaptive Data Structure (ADS) It attempts to generate an asymptotically optimal tree with respect to the access probabilities of the neurons, i e , based on the number of times that they have been identified as the BMU

2.9.3 Type of Distance for Determining the BoA

When defining a variant of the original SOM strategy, the definition of the BoA constitutes an important module that must be carefully defined The different strategies discussed utilize a family of schemes to define the BoA We mainly identify three classes which are discussed in this section

The first type of BoA uses the concept of neural distance as in the case of the traditional SOM Once the BMU is identified, the neural distance is calculated by traversing the underlying structure that hold the neurons An important property of the neural distance separating two neurons is that it is proportional to the number of connections separating them Examples of strategies using the neural distance to determine the BoA are the GCS, the GG, the IGG, the GSOM, the TSSOM, the HFM, the GHSOM, the SOTA and the ET

A second subset of strategies employs a scheme for determining the BoA that does not depend on the inter-neural connections Instead, such strategies utilize the distance in the feature space In these cases, it is possible to distinguish between two types of NNs The simplest situation occurs when the BoA only considers the BMU, i e , it constitutes an instance of hard CL, as in the case of the TSVQ and the SOTM A more sophisticated and computationally expensive scheme involves a ranking of neurons according to their respective distance to the stimulus In this scenario, the BoA is determined by selecting a subset of the closest neurons An example of a SOM variant using this ranking is the NG

2.9.4 Additional Counters

A strategy that some researchers have followed in order to improve the performance of the NNs, is the one of adding extra counters to it. These counters correspond to additional variables that are included in the NN to perform certain tasks such as providing an estimator for the QE or determining how old a connection is. Such enhancements are primarily intended so as to alleviate the computationally expensive tasks needed to achieve self-organization. Counters that measure the QE of the neurons are useful for the identification of under-represented regions of the space which require further exploration. It is possible to differentiate a family of counters that store a QE measure using updating mechanisms such as the ones defined by Equations (2.20), (2.21), (2.25) and (2.30). These QE counters are usually dependent on the position of the stimulus and the previous value of the QE of the neuron currently being examined. Schemes using counters of this type are the GNG, the IGG, and the GSOM.

It is crucial that one makes the distinction between employing a QE counter and actually computing the QE. The latter requires the processing of all the input sample set, which is an avenue that is not always feasible or is too expensive. On the other hand, an estimation of the QE through the employment of these counters may not necessarily be coincident with the calculation obtained by examining each input sample separately. Rather, the results can be very approximate, particularly in the later stages of the training, where neural migration is diminished. The GHSOM is an example of a strategy that calculates the QE rather than using a counter that allows its estimation (see Equations (2.26), (2.27) and (2.29) to observe how this is achieved).

A counter that requires special attention due to its popularity among the different SOM-based strategies is the so-called BMU counter (explained in Section 2.6.1), that keeps track of how many times a neuron has been selected as the winner. Examples of strategies using the BMU counter are the GCS, the GG, the TSVQ, the TSSOM, and the ET.

Finally, a particularly interesting type of counter is the one that records the age of a connection, as utilized in the NG strategy. The NG is the only example from among

the SOM-variants in which a counter is associated with a *connection* as opposed to the neurons in the NN

2.9.5 Neuron Properties

Some strategies define special properties for the neurons These special properties allow an alternate type of distinction between the subsets of SOM variants studied in this Thesis

One such property is the so-called "Frozen" state, explained in detail in Section 2 6 5 In general, strategies using this approach train the neurons until they are themselves identified as BMUs a certain number of times, after which they become static and do not take part in the competition anymore Examples of NNs that invoke the concept of "frozen" neurons are the TSVQ, the TSSOM and the ET

A second type of neural property is the one which involves so-called "boundary" nodes, described in detail in Section 2 6 4 As the name suggests, neurons of this type are the ones which fall close to the borders of the underlying lattice This concept is complemented by the phenomenon of the so-called "inner" neurons A neurons is said to be "inner" if all of its 4 neighboring positions in the lattice are already occupied by a neuron Based on these definitions, boundary nodes are those that still have the possibility of generating new connections

This concept is utilized by strategies that dynamically grow a grid-like structure such as the IGG and the GSOM

2.9.6 Search Method

As mentioned in the introductory section on the SOM (Section 2 5), the search for the BMU is both crucial and one of the most expensive tasks associated with the traditional SOM strategy In general, any SOM-based strategy needs to identify a winner neuron in one way or another Researchers have tried to overcome this hurdle of using a brute-force approach by defining heuristic methods for determining a winner A particular heuristic discussed in this Thesis is the so-called "find-best" heuristic, discussed in Section 2 6 6, which recursively traverses a tree and identifies a

winner (that may not be the optimal one) in $O(\log n)$ time. The "find-best" algorithm was originally designed as part of the TSVQ philosophy, and has been adapted to also operate in conjunction with other tree-based schemes, e.g., the TSSOM and the ET.

2.10 Comparison Table

A table summarizing the main aspects of the different SOM-based strategies is given in Table 2.1. The columns of Table 2.1 specify a particular feature, such as the type of topology, while each row refers to a particular SOM variant. Check marks ("✓") are used to specify that a particular strategy possesses the respective feature.

For instance, the reader can easily determine from Table 2.1 that the NG approach, utilizes a graph topology and that it also considers the possibility of adding/deleting edges by taking advantage of the "Age" of the connections. It is also possible to glean knowledge of other strategies that share similar properties to a particular philosophy For example, Table 2.1 explicitly clarifies that the property of deleting edges observed in the NG is present only in a small subset of strategies (see Section 2.9.2).

We believe that Table 2.1 can be of great value for future researchers who require an understanding of the most relevant properties of the SOM variants and their mutual relationships specified in terms of their shared characteristics. We have deliberately included in the last rows, the strategies that we have currently developed, and which will be discussed in the following chapters. The reader can easily perceive the phenomena in which our novel methods are distinct from the previous strategies. In particular, one can see from Table 2.1 that we have ventured into a completely unexplored terrain by merging the tree structured family of SOMs with the field of ADSs, thus generating a type of dynamism that is both novel and unique.

Table 2.1: A comparison between different SOM strategies based on the criteria specified in Section 2.9.

Name	Topology					Dynamism						BoA		Counters			Neuron State		Search	
	(Unconnected) Graph	Tree (acyclic graph)	Grid	Irregular Grid	SOM-layers	Static	ADS	Add edges	Delete edges	Add nodes	Delete nodes	Feature space	Neural space	BMU	QE	Age of connection	Frozen	Boundary	Heuristic	Deterministic
SOM			✓			✓							✓							✓
GCS	✓							✓	✓	✓	✓		✓	✓						✓
NG	✓							✓	✓			✓				✓				✓
GNG	✓							✓	✓	✓	✓	✓			✓					✓
GG			✓					✓		✓	✓		✓	✓						✓
IGG				✓				✓	✓	✓	✓		✓		✓			✓		✓
GSOM				✓				✓		✓			✓		✓			✓		✓
TSVQ	✓					✓						✓		✓			✓		✓	
TSSOM	✓				✓	✓							✓	✓			✓		✓	
HFM	✓				✓	✓		✓					✓							✓
GHSOM	✓				✓			✓		✓			✓							✓
SOTM	✓							✓	✓	✓		✓								✓
SOTA	✓							✓		✓			✓							✓
ET	✓							✓					✓	✓			✓		✓	
TTOSOM	✓					✓							✓							✓
TTOCONROT	✓						✓						✓	✓						✓

2.11 Conclusions

In this chapter, we provided an overview of the fundamental SOM-based definitions and algorithms reported in the literature This was provided so as to survey the field and to also provide the background necessary to better comprehend the state-of-the-art Although many scores of contributions have been published (especially in the last three decades), we submit that this overview is quite comprehensive

The survey included a brief explanation of the concepts of machine learning, placing special emphasis on unsupervised learning and clustering After visiting the topics of ANNs and the SOM, we described, in greater detail, the families derived from the SOM, which, indeed, is the main focus of the present Thesis

Chapter 3

Imposing Tree-based Topologies onto SOMs

3.1 Introduction

In this Chapter[1] we will present a novel scheme by which the user can build, train and develop a Kohonen-like NN in which the topology is user-defined and tree-oriented As the reader can appreciate from the previous chapter, the beauty of the Kohonen map is that it has the property of organizing the codebook vectors, which represent the data points, both with respect to the underlying distribution and topologically This topology is traditionally linear, even though the underlying lattice could be a grid, and this has been used in a variety of applications [97, 142] The most prominent efforts to render the topology to be structured involves the Evolving Tree (ET) due to Pakkanen *et al* [143], and the Self-Organizing Tree Map (SOTM) due to Guan *et al* [82], among others – which have all been discussed in Chapter 2

As mentioned in Section 2 5 8, the proof that the stochastic distribution of the prototypes follows the distribution of the underlying data points is only available for the linear case This fact leads us to a host of other open issues The first issue is the

[1] A preliminary version of some of the results from this chapter appear in the Proceedings of CORES2009, the 6^{th} International Conference on Computer Recognition Systems, Poland, in May 2009 [11] This talk was a Plenary/Keynote Talk at the Conference The journal version of these results is currently under review

following Is it possible for a user to require the prototypes to follow any arbitrary topology? If this is possible, the implication is significant First of all, the question of finding the nearest neuron with such a topology (i e , the winner in the competition) has to be answered Secondly, the whole issue of how one describes the BoA, and of migrating neighbors of the winner is far more significant Indeed, the question of what we mean by a neighbor, how the list of neighbors is maintained and what the final convergence is, are unresolved issues This, indeed, is the goal of this Chapter

Our aim is to permit the user to specify any tree-like topology which can be entirely up to his imagination The reason why we prefer a tree-like topology is to prevent cyclic neighborhood correspondences Once this topology has been fixed, the concepts of the neighborhood and BoA are specified from *this* perspective The question is whether the prototypes can ultimately learn the stochastic distribution *and simultaneously* arrange themselves with the topology that mimics the one that the user hypothesized from the outset We show that this is indeed possible, as demonstrated by a set of rigorous experiments in which our enhanced ANN, the Tree-based Topology Oriented SOM (TTOSOM) is able to learn both the distribution and the desired structured topology of the data Furthermore, a consequence of this is the fact that as the number of neurons is increased, the approximation of the space will be correspondingly superior – both from the perspective of the distribution and of the user-defined topology

3.1.1 Contribution of the Chapter

The principal contributions of the present Chapter can be summarized as follows

1 We present a technique by which we can represent data points using prototypes, both with respect to the underlying distribution and *any* user-defined topology

2 In particular, we demonstrate how the user can impose an arbitrary tree-like topology onto the set of codebook vectors

3 Since the topology can be fairly arbitrary, we show that the resultant BoA is different from the ones defined in the prior literature, both structurally and

conceptually

4 The map learned as a consequence of the training process is able to infer both the distribution and the structured topology of the data as verified by extensive experiments

5 The strategy proposed reduces to the traditional 1-dimensional SOM when the tree is a linear sequence of nodes In other words, the traditional SOM is a special case of the family of ANNs which we propose

3.1.2 Organization of the Chapter

The organization of the Chapter is as follows In Section 3 2, we present the new SOM variant, namely the TTOSOM Thereafter, in Section 3 3, the experimental results which demonstrate its power are included Finally, Section 3 4 gives some concluding remarks and discussions

3.2 The Tree-based Topology Oriented SOM

The Tree-based Topology Oriented SOM (TTOSOM) is a tree-structured SOM which aims to discover the underlying distribution of the input data set $\mathcal{X}$, while also attempting to perceive the topology of $\mathcal{X}$ as viewed through the user's desired perspective The TTOSOM works with an imposed topology structure, where the codebook vectors are adjusted using a VQ-like strategy Besides, by defining a user-preferred neighborhood concept, as a result of the learning process, it also learns the topology and preserves the prescribed relationships between the neurons as per this neighborhood Thus, the primary consideration is that the concept of neurons being "near each other" is not prescribed by the metric in the space, but rather by the structure of the imposed tree

3.2.1 Declaration of the User-Defined Tree

The topology of the tree arrangement of the neurons plays an important role in the training process of the TTOSOM As detailed in Section 2 8, this concept is not new, and a few variations of SOMs take advantage of this approach

In general, the TTOSOM incorporates the SOM with a tree which has an arbitrary number of children Furthermore, it is assumed that the user has the ability to describe/create such a tree The user who presents it as an input to the algorithm, utilizes it to reflect the *a priori* knowledge about the *structure* of the data distribution[2]

The first task to be conducted is that of declaring, as an input, the user-defined tree We propose that this declaration is done in a recursive manner (see Alg 12), from which the structure of the tree is fully defined The input to the algorithm is an array that contains integers specifying the number of children for each node in the tree if the latter is traversed in a depth-first manner[3]

The algorithm works in a straightforward manner The input to the algorithm is a pointer to a node of a tree (which could be the root), and an associated array The position i in the array implicitly refers to the i^{th} node of the final tree if traversed in a depth-first manner The contents of the array elements in the i^{th} position refer to the number of children that node i has An example of this is given in Figure 3 1 where the input array is $\langle 2,3,4,0,0,0,0,1,0,2,0,0,2,0,0\rangle$, and the resulting tree is shown in the figure itself Observe that the same tree could have also been traversed in a breadth-first manner, in which case, the corresponding array for this tree would be the array $\langle 2,3,2,4,1,2,0,0,0,0,0,0,0,0,0\rangle$

3.2.2 Representation of the Tree

From the immense diversity of possible data structures capable of representing trees, choosing a data structure that best suits the purposes of our problem requires the specification of all the constraints that are to be satisfied First, each node in the

[2]The beauty of such an arrangement is that the data can be represented in multiple ways depending on the specific perspective of the user

[3]The tree could also easily be traversed in a breadth-first manner as we will see presently

Algorithm 12 `describe-topology`(A,p)

Input:

i) A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$, specifying the number of children for each node in the tree.

ii) p, a pointer to the current root of the tree.

Method:

```
1  if n = 0 then
2    return
3  else
4    number-of-children = head(A) {read and cut head of the sequence}
5    for i=1 to number-of-children do
6      create-node(x)
7      add-node(x,p)
8      describe-topology(A,x)
9    end for
10 end if
```

End Algorithm

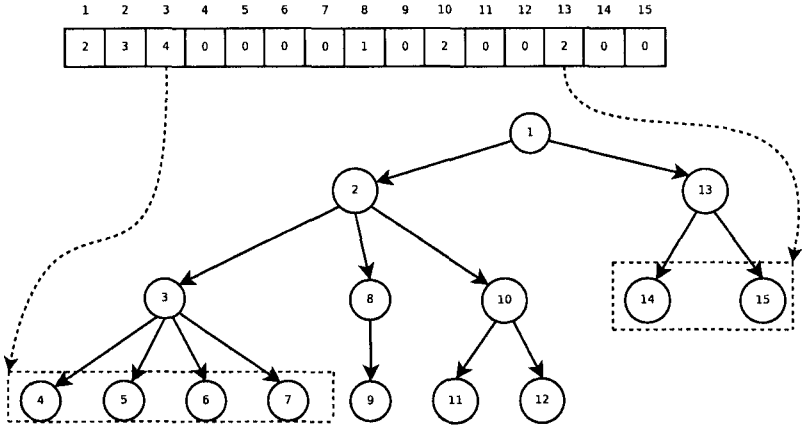


Figure 3.1: An example of the description of the original tree topology. The example shows an array containing the number of children for each node in the tree. If the latter is traversed in a depth-first manner, the index of the array is the position in the depth-first traversal, and the content of the array is the number of children of the node in question. The corresponding array, if the tree is traversed in a breadth-first manner, is $\langle 2, 3, 2, 4, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$.

tree could possibly have an arbitrary number of children, limited only by the user's "imagination" Secondly, the form in which the tree is represented must permit fast traversal, especially for the most expensive tasks of the algorithm, namely, the location of the current BMU, and the calculation of the BoA, i e , the neighborhood of units around it Fast identification of the parents is required, since our concept of neighborhood involves the traversal of the nodes, not only in the direction of the children of a given node, but also in the direction of its ancestors, as explained in Section 3 2 3, and pictorially described in Figure 3 3, which implicitly requires us to maintain a link to quickly reach the parent

An efficient way to represent the structure of the tree, is to use the Left-most-child, Right-sibling representation In this approach each node points only to its left-most child and not to all the direct children, and every node is linked only to its sibling immediately to the right (and not to all the siblings) Simultaneously, we also maintain a pointer from every node to its parent, with the exception of the root node which is pointing to the *NULL* pointer Interestingly, the Left-most-child, Right-sibling representation stores the data in $O(M)$ space, where M is the number of nodes Also, this strategy permits the representation of the tree with an arbitrary number of children, to be a *binary tree*

The implementation of the tree using the "Left-most-child, Right-sibling" is given in Figure 3 2 (for the tree of Figure 3 1), and will also be represented with the array $\langle 2, 3, 4, 0, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0 \rangle$

With regard to the data maintained in each node of the tree, we mention that, each node will contain the d-dimensional vector in the feature space migrated as per the position of the input samples and the update rule Note that in Figure 3 2 the details of the vectors in the feature space are omitted in the interest of simplicity

3.2.3 Neural Distance Between Two Neurons

In the TTOSOM, the *Neural Distance*, d_N, between two neurons depends on the *number* of unweighted connections that separate them in the user-defined tree, and is defined as the number of edges in the shortest path that connects the two given

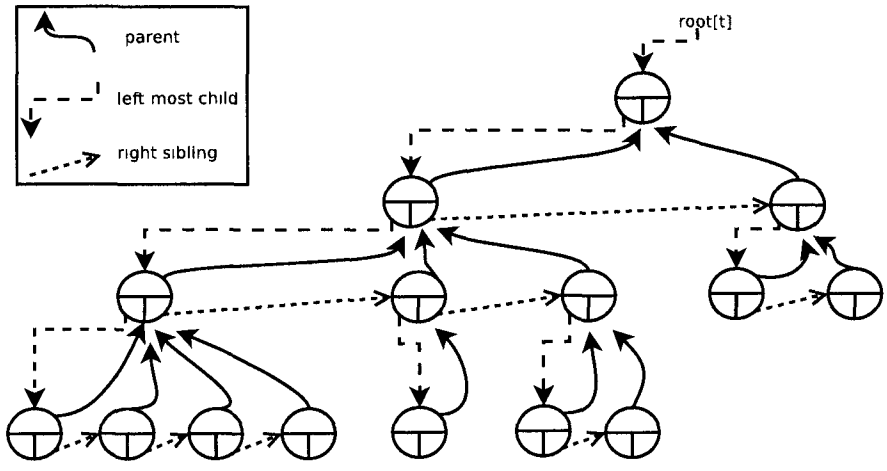


Figure 3 2 A suitable way to implement the user-defined tree is by using the Left-most-child, Right-sibling representation, which uses $O(n)$ space This figure illustrates the Left-most-child, Right-sibling representation for the tree of Figure 3 1 For simplicity, the details of the vector in the d-dimensional feature space, associated with each node, are omitted

nodes More explicitly, the distance between two nodes in the tree, is defined as the minimum number of edges required to go from one to the other In the case of trees (since this is the focus of the present work), there is only a single path connecting two nodes, implying the simplicity of enforcing the definition Additionally it is worth mentioning that this notion of distance is not dependent on whether nodes are leaves or not, as in the case of the ET (described in Section 2 8 6), thus permitting the calculation of the distance between *any* pair of nodes in the tree

More specifically, the distance between a neuron and itself is defined to be zero, and the distance between a given neuron and all its direct children and its parent is unity This distance is then recursively defined to farther nodes as shown pictorially in Figure 3 3a Clearly, if $\mathbf{c}_i$ and $\mathbf{c}_j$ are nodes in the tree, $d_N(\ ,\ )$ possesses the identity,

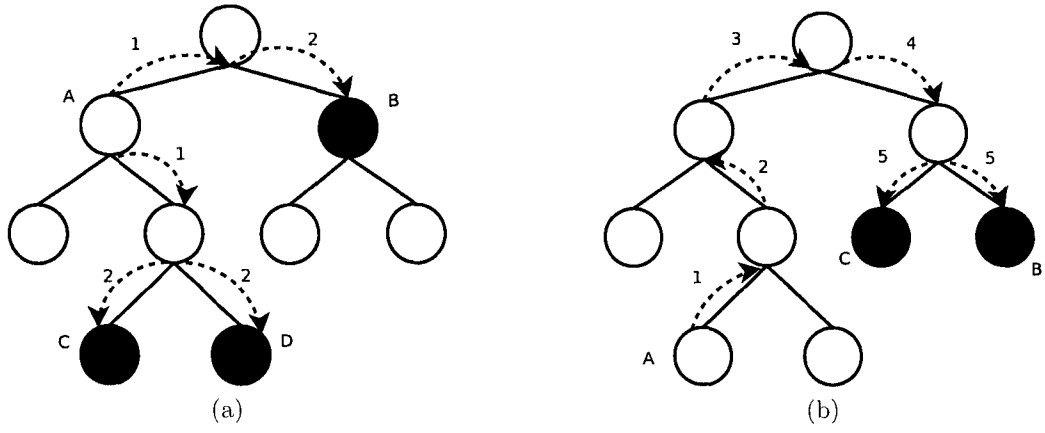


Figure 3.3: Figure 3.3a shows the neighborhood for the TTOSOM. Here nodes B, C and D are equidistant to A even though they are at different levels in the tree. Observe that non-leaf nodes may be involved in the calculation. Figure 3.3b shows the case for the ET, where nodes B and C are equidistant to A. In the ET, the definition of the distance pertains only to leaf nodes.

non-negativity, symmetry and triangular inequality properties:

$$d_N(\mathbf{c}_i, \mathbf{c}_j) \geq 0 \tag{3.1}$$

$$d_N(\mathbf{c}_i, \mathbf{c}_j) = 0 \quad \texttt{if and only if } \mathbf{c}_i = \mathbf{c}_j \tag{3.2}$$

$$d_N(\mathbf{c}_i, \mathbf{c}_j) = d_N(\mathbf{c}_j, \mathbf{c}_i) \tag{3.3}$$

$$d_N(\mathbf{c}_i, \mathbf{c}_k) \leq d_N(\mathbf{c}_i, \mathbf{c}_j) + d_N(\mathbf{c}_j, \mathbf{c}_k). \tag{3.4}$$

The reader should observe in Figure 3.3 that nodes at different levels could also be equidistant from any given node. Thus, in the case of Figure 3.3a, nodes B, C and D are all at a distance of 2 units away from A, while in Figure 3.3b nodes B and C are at a distance of 5 units from node A. It should also be mentioned that in the ET, one encounters only distances between the leaves (as in Figure 3.3b) and not between internal nodes, as in Figure 3.3a.

As in the case of the traditional SOM, the TTOSOM requires the identification of the BMU, i.e., the closest neuron to a given input signal. This is achieved by invoking Algorithm 13. To locate it, the distances, $d_f(\cdot, \cdot)$ are computed in the *feature* space and not in terms of the edges of the user-defined tree. The algorithm first calculates

the feature-based distance in the feature space between the input signal, $\mathbf{x}$, and every neuron in the network, and the index of the neuron with the smallest feature-based distance is returned In the case of a tie, the index of the first-found neuron with minimum distance is selected (although a random tie breaker is also possible) It is important to emphasize that the distance measured between neurons is completely distinct from the feature-based distance between a neuron and the input signal, since the latter is computed based on the coordinates of the neuron and the signal in the feature space, and the distance between neurons is computed in terms of the number of edges to be traversed in the user-defined tree A common choice for the distance is the Euclidean distance, however, it has been reported that for higher dimensional spaces, this particular definition of distance might be inaccurate [172] The TTOSOM attempts to tackle this issue by inheriting the SOM's update rule, which utilizes a distance function but does not necessarily specify one explicitly

Algorithm 13 `TTOSOM-Find-BMU`(p,$\mathbf{x}$,v)

Input:

i) p, a pointer to the node currently being examined
ii) $\mathbf{x}$, an input sample from $\mathcal{X}$
iii) v, a pointer to the best matching neuron found so far

Output:

i) v, a pointer to the BMU

Method:

```
1  if p=NULL then
2    return v
3  else
4    if getDistance(x, v) >getDistance(x, p) then
5      v=p {p is the new best matching unit}
6    end if
7    for all child ∈ p getChildren() do
8      v=TTOSOM-Find-BMU(child,x,v)
9    end for
10 end if
11 return v
```

End Algorithm

3.2.4 The Bubble of Activity

Intricately related to the notion of inter-node distance, is the concept referred as to the Bubble of Activity (BoA) which is the subset of nodes "close" to the unit being currently examined (see Section 2 5 3) These nodes are essentially those which are to be moved toward the input signal presented to the network This concept involves the consideration of a quantity, the so-called *radius*, which determines how big the BoA is, and which therefore has a direct impact on the number of nodes to be considered The BoA is defined as the subset of nodes within a distance of r away from the node currently examined, and can be formally defined as per Equation 2 12

Figure 3 4 depicts an example of how the number of neurons in the BoA is increased as the radius is increased In the case of the TTOSOM, the BMU can be *any* node in the tree, and this becomes the center of the BoA The question of whether or not a neuron should be part of the current bubble, depends on the number of connections that separate the nodes rather than the distance that separate the points in the solution space (for instance, the Euclidean distance)

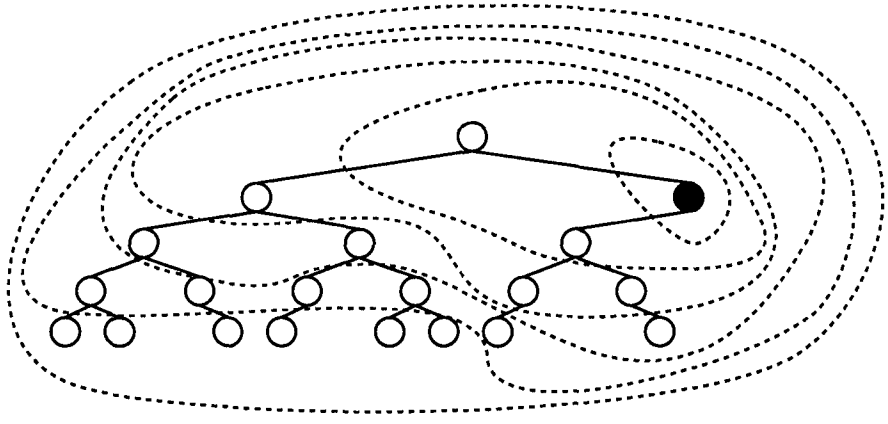


Figure 3 4 The figure shows how the BoA includes more nodes as the radius is increased The currently examined node is given in black

The concept of neighborhood utilized in the present work is distinct and different from the ones used in other approaches such as the ET (see Section 2 8 6) or the SOTM (detailed in Section 2 8 4) In the case of the TTOSOM, non-leaf units are

considered in the BoA As an example, in our case, nodes distant by a radius of zero will include, in the BoA, only the node being currently examined, while a radius of unity will consider the subset of all the direct children of the unit being examined as well as the direct parent node As in any SOM philosophy, theBoA is initially made large enough so as to include all the nodes in the structure, and subsequently, as the learning proceeds, gradually decreased to finally only include the node currently being examined There are reported works that consider only leaves in the BoA [143] and some of them use the distance of the neurons in the solution space [82]

Algorithm 14 dictates the calculation of the subset of neurons that are part of the neighborhood of the BMU This computation involves a collection of parameters, including ϕ, the current subset of neurons in the proximity of the neuron being examined, v, the BMU itself, and $r \in \mathbb{N}$ the current radius of the neighborhood When the function is invoked for the first time, the set ϕ contains only the BMU (which is previously estimated using Algorithm 13) marked as the current node, and through a recursive call, ϕ will end up storing the entire set of units within a radius r of the BMU The tree is recursively traversed for all the direct topological neighbors of the current node, i e , in the direction of the direct parent and children Every time a new neuron is identified as part of the neighborhood, it is added to ϕ and a recursive call is made with the radius decremented by one unit[4], marking the recently added neuron as the current node

3.2.5 Training the TTOSOM

The training process of the TTOSOM involves positioning the neurons which describe the user-defined tree in the feature space so as to capture the distribution and topology of the data points This process involves a loop of training steps which terminates when the convergence is acceptable to the user The *Training step* involves requesting an input sample from the dataset, locating the BMU, computing the nodes within the *current* BoA, and migrating those neurons toward the input signal using a SOM-like philosophy

[4]This fact will ensure that the algorithm reaches the base case when $r = 0$

Algorithm 14 TTOSOM-Calculate-Neighborhood(ϕ,v,r)

Input:

i) ϕ, the set of the nodes in the BoA identified so far

ii) v, a pointer to the node from where the BoA is calculated

iii) r, the current radius of the BoA

Output:

i) The set of nodes in the BoA

Method:

```
1  if r ≤ 0 then
2    return
3  else
4    for all child ∈ v getChildren() do
5      if child ∉ φ then
6        φ ← φ ∪ {child}
7        TTOSOM-Calculate-Neighborhood(φ,child, r − 1)
8      end if
9    end for
10   parent=v getParent(),
11   if parent ≠ NULL and parent ∉ φ then
12     φ ← φ ∪ {parent}
13     TTOSOM-Calculate-Neighborhood(φ,parent, r − 1)
14   end if
15 end if
```

End Algorithm

How the input sample is chosen may vary In the case of the TTOSOM, an input sample is drawn at random from the sample set Other mechanisms may involve the shuffling of the elements in the set, generating an order, and then at each step the samples are chosen as per the pre-established order Finally, when the last element in the sequence has been drawn, an *epoch* is said to be completed, and optionally a new shuffling takes place

The TTOSOM uses the SOM update rule to update the prototypes (described in Section 2 5 4), and requires the definition of a learning factor, η, which, in turn, specifies the fraction by which the BMU will move towards the input presented to the network The value of the learning rate is initialized to a high value, for example 0 9, and is gradually decreased so as to achieve convergence At first the updated positions of the BMU and neighbor neurons will be highly influenced by the input sample, but as time goes by, these will tend to move less, and at the end of the process only small changes in the positions of the neurons will be registered

Algorithm 15 gives the details on how the training step is performed Initially, the index of the BMU is obtained by invoking Algorithm 13, and will be the first neuron to be included in the BoA The next process involves the calculation of all the neurons in the bubble, which is obtained by calling Algorithm 14 Finally, for each neuron in the bubble, an update process take place, which is similar to the SOM update rule

Algorithm 15 `TTOSOM-Train`($\mathbf{x}$,p)

Input:

i) $\mathbf{x}$, an input sample from $\mathcal{X}$

ii) p, a pointer to root of the tree

Method:

```
1 v ← TTOSOM-Find-BMU(p,x,p)        {See Algorithm 13 }
2 φ ← {v}
3 TTOSOM-Calculate-Neighborhood(φ,v,radius)        {See Algorithm 14 }
4 for all b ∈ φ do
5    Update the neuron b according to Equation (2 16)
6 end for
```

End Algorithm

3.2.6 The Overall Procedure

In what we have covered, we have explained all the independent modules of the TTOSOM. Using these modules we shall show how all the pieces fit together to compose the overall TTOSOM algorithm. The main segments of the algorithm also include the initialization of the values and the main training loop.

The input to the algorithm consists of a set of samples in the d-dimensional feature-space, and, additionally, as explained in Section 3.2.1, an array by which the user-defined tree structure can be specified. Observe that this specification contains all the information necessary to fully describe the TTOSOM structure, such as the number of children for each node in the tree. Furthermore, the algorithm also includes parameters which can be perceived as "tuning knobs". They can be used to adjust the way by which it learns from the input signals. The TTOSOM requires a schedule for the so-called decay parameters, which is specified in terms of a list, where each item in the list records the value for the learning rate, the radius of the BoA, and the number of learning steps for which the latter two parameters are to be enforced.

The initialization phase constitutes the creation/description of the user-defined tree topology, and the value of the vectors in the feature-space associated with each node within the tree. At the beginning of this phase, the tree must be built as per Algorithm 12, implying the specification of the root of the tree. Initially the tree is not defined, and this is done by assigning the parameter p of Algorithm 12 to the *NULL* pointer. At this juncture, the starting values of the codebook vectors associated with each node of the tree, are set as well. One common way to accomplish this is by randomly choosing input vectors from the input sample set, and assigning their values to the codebook vectors. Alternate methods relying on a Principal Component Analysis (PCA) have also been reported in the literature [92] (weight initialization is described in Section 2.5.1).

The essential phase of the process involves the learning iterations, which is a sequence of steps as per Algorithm 15, and are monitored by the schedule of parameters mentioned earlier. Whenever a learning step takes effect, some of the nodes in the tree are moved towards the input sample. The number of nodes moved and the amount by which these neurons are moved, depends on the current parameters, such as the BoA

and the learning rate, η Algorithm 16 details the whole process, which terminates when the position of the neurons have converged satisfactorily

In practice, as we shall see presently, as a result of the execution of Algorithm 16, the codebook vectors will be arranged in such a way that they represent the structure and the distribution of the input data While the "spatial" distribution of the neurons follows the stochastic distribution of the data points, the relationships between the nodes of the TTOSOM will be as per the user-defined structure This preserves the topological information in the data points (as in the traditional SOM algorithm), but also the information involving the tree-based relationship between the neurons

Algorithm 16 `TTOSOM`($\mathcal{S}$,A,$\mathcal{X}$)

Input:

i) $\mathcal{S}$, the schedule of parameters
ii) A, the array containing the number of children for each node in the tree
iii) $\mathcal{X}$, the input sample set

Method:

```
1  T ← NULL
2  describe-topology(A,T)        {see Algorithm 12}
3  Initialize the codebooks by, for instance, randomly selecting samples from X
4  for all  s ∈ S do
5     Set the learning factor and radius according to the predefined schedule
6     for a fixed number of iterations pre-specified in the schedule do
7        Generate a stimulus x by randomly selecting a sample from X
8        TTOSOM-train(x,T)        {see Algorithm 15}
9     end for
10 end for
```

End Algorithm

3.3 Experiments and Results

To demonstrate the power of our method, and to obtain a better understanding on how the structure-oriented SOM works, we have implemented and tested it on numerous data sets[5] First of all, from a pedagogical perspective, the fact that the user

[5]These data sets have been made available to the public, and can be found at `http //www scs carleton ca/~castudil`

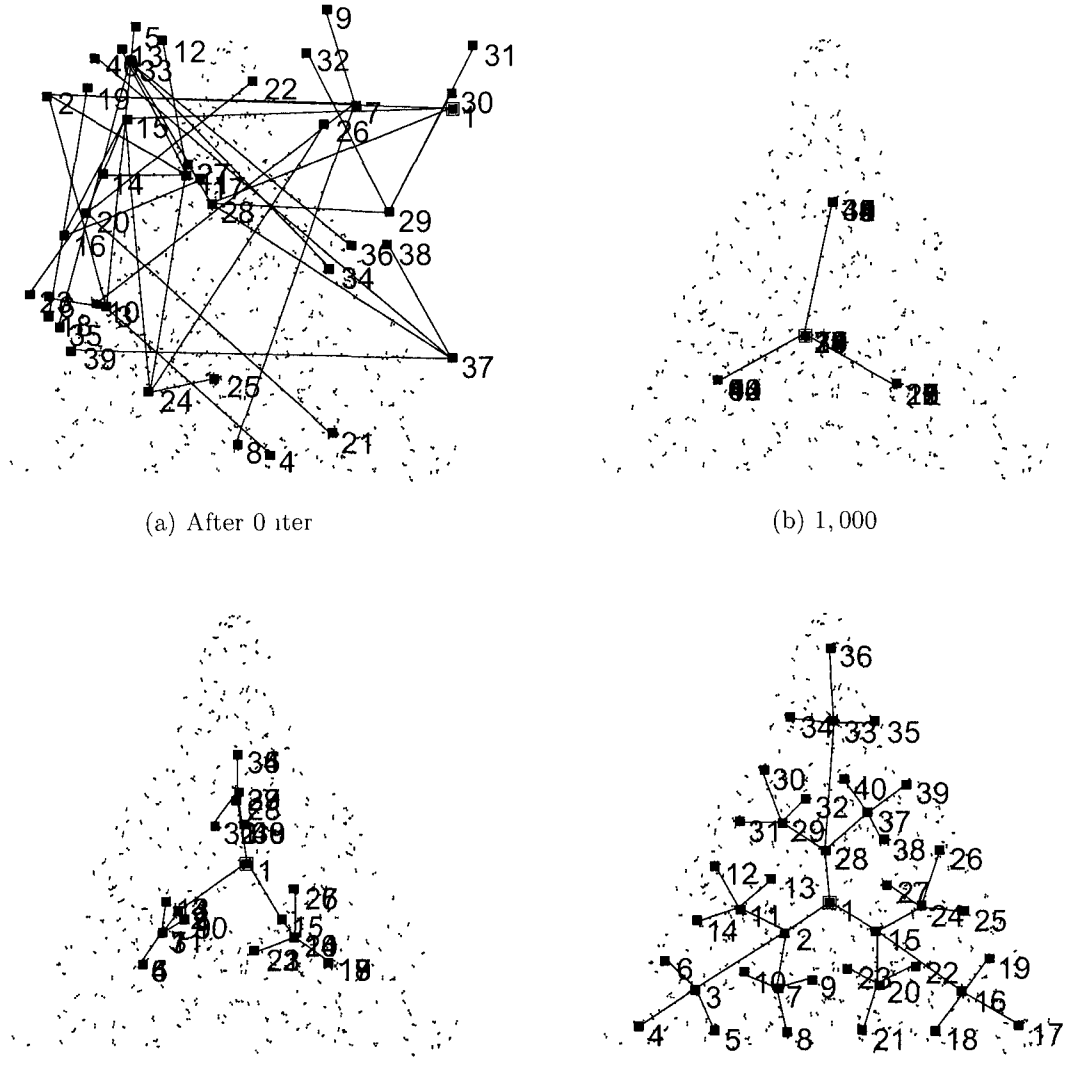
is capable of visualizing the resulting tree, is important, because, in this way, a human operator/user will be provided with a quick and intuitive tool for understanding the "structure" of the data Thus he will be able to comprehend what is happening to the tree that attempts to delineate the structure of the points from the data distribution This is achieved by virtue of the fact that the position of the nodes belonging to the tree correspond to the value of the codebook vector associated with each node, and which are also manipulated by the cloud of points that are randomly drawn from the input data set Therefore, to illustrate this philosophy, the experiments reported here, were done in the 2-dimensional feature space It is important to remark that the capabilities of the algorithm are also applicable for higher dimensional spaces, though, the visualization of the resulting tree will not be as straightforward Additionally, in all the cases presented here, the input samples were drawn from a probability distribution unknown to the algorithm While both the distribution and its structure were unknown to the TTOSOM, the hope is that the latter will be capable of inferring them through the learning process, without human intervention, thus, performing *Unsupervised Learning* Lastly, the schedule of parameters was specified so as to result in a rather "slow" convergence This was achieved by defining steady values for the learning rate for a large number of iterations, so that we could understand how the position of the nodes migrated towards their final configuration We believe that to solve practical problems, the convergence can be accelerated by appropriately choosing the schedule of parameters

3.3.1 Learning the Structure

Consider the data generated from a triangular-spaced distribution as per Figure 3 5 We assume that the *a priori* knowledge from the user permitted him to define a tree topology with a complete tree of depth 4, and where each node has exactly 3 children This implies that the user's tree consists of $\sum_{i=0}^{4} 3^i = 1+3+9+27 = 40$ nodes, which is initialized as per the procedure explained in Section 3 2 1 Figure 3 5a, depicts the position of the nodes of the tree after a random initialization Random initialization was used by uniformly drawing points from the unit square Observe that the original

data points do not *lie in the curve* Our aim here was to show how our algorithm could learn the structure of the data using *arbitrary* (initial and "non-realistic") values for the codebook vectors The reader must appreciate that the initial random positioning completely obfuscates the tree structure since the input signals conform to a cloud of points within the unit square However, once the main training loop becomes effective, the codebook vectors get positioned in such a way that they represent the structure of the data distribution, and simultaneously preserve the user-defined topology This can be clearly seen from Figure 3 5b and 3 5c which are snapshots after $1,000$ and $10,000$ samples, respectively At the end of the training process (see Figure 3 5d), the complete tree fills in the triangle formed by the cloud of input samples and seems to do it uniformly The final position of nodes of the tree suggests that the underlying structure of the data distribution corresponds to the triangle Observe that the root of the tree is placed roughly in the center (i e , the mean) of the distribution It is also interesting to note that each of the three main branches of the tree, cover the areas directed towards a vertex of the triangle respectively, and their sub-branches fill in the surrounding space around them

Another example is shown in Figure 3 6, where the data was generated from a "square-shaped" distribution Analogously, we presuppose that using the information at the disposal to the user, he defines a tree topology with a complete tree of depth 4, and where each node has exactly 4 children As a consequence, the user's defined tree will be composed by $\sum_{i=0}^{4} 4^i = 1+4+16+64 = 85$ nodes, and is again initialized utilizing the procedure detailed in Section 3 2 1 Figure 3 6a, portrays the location of the nodes within the tree after setting their values by randomly selecting points from the unit square As we can see, the initial position of the codebooks based on the structure of the user-defined tree is "confusing", because of the random nature of the samples drawn from the unit square distribution Convergence takes place by repeatedly executing the main training loop, and all the while the codebook vectors are moved until they represent the structure of the data distribution, and at the same time maintain the topology of the tree This can be clearly seen from Figure 3 6b and 3 6c which display the situation after 10,000 and 100,000 iterations, respectively Figure 3 6d depicts the situation after training Observe that the position of the



(a) After 0 iter

(b) 1,000

(c) 10,000

(d) 300,000

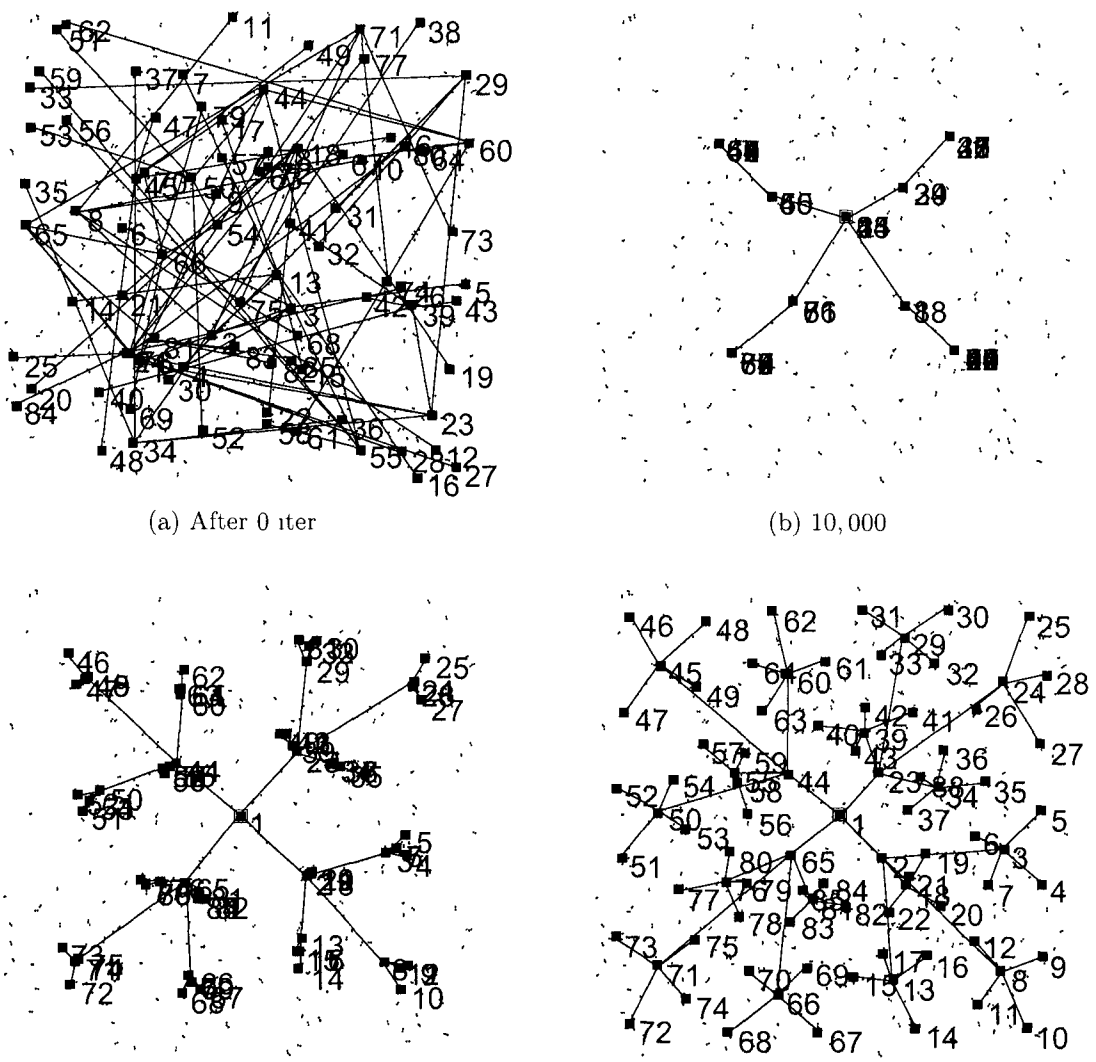
Figure 3.5: TTOSOM-based 3-ary tree topology learned from a "triangular" distribution.

nodes appears to be uniform inside the square formed by the cloud of samples In this case, the position of the nodes implicitly suggests that the underlying structure of the data distribution corresponds to the square Similar to the triangle example, note that in this case, the root of the tree is placed near the center of mass of the square formed by the cloud of input signals Also interestingly, the main branches of the tree cover the principal diagonals of the square, and their sub-branches spread through the space around these, respectively

With the aim of comparing our method with the traditional SOM, we considered a discrete data distribution of 500 data points[6] For this experiment an equivalent number of neurons where employed, The SOM utilized a 5×5 grid, i e , 25 neurons, while the TTOSOM employed a complete 4-ary tree of four level, i e , $1 + 4 + 16 = 21 \approx 25$ neurons The resulting maps demonstrate one of the known drawbacks of the SOM, namely the convergence of neurons in zero-density areas Additionally, although the root of the tree trained by the TTOSOM lays in a zero-density area, it is located roughly in the center of mass of the data distribution, and thus renders meaningful information about the *entire* data set In our opinion, the TTOSOM using a lesser number of connections yields a mapping which is more easy for a human being to "read" Note also that the nodes at second level of the tree lie in the centers of mass of the four biggest concentration of data points

To illustrate the properties of the TTOSOM by incorporating other structures, we consider the scenario when the topology is unidirectional Indeed, we obtain very impressive results when the tree structure is the 1-ary tree as seen in Figure 3 8 In this case, the user-defined a list (i e , a 1-ary tree) as the imposed topology Initially, the codebook vectors were randomly placed as per Figure 3 8a Again, the reader must observe that at the beginning, the linear topology is completely lost due to the randomness of the data points The migration and location of the codebook vectors after 10,000 and 100,000 iterations are given in Figure 3 8b and 3 8c respectively At the end of the training process the list represents the triangle very effectively, as also reported in [103]

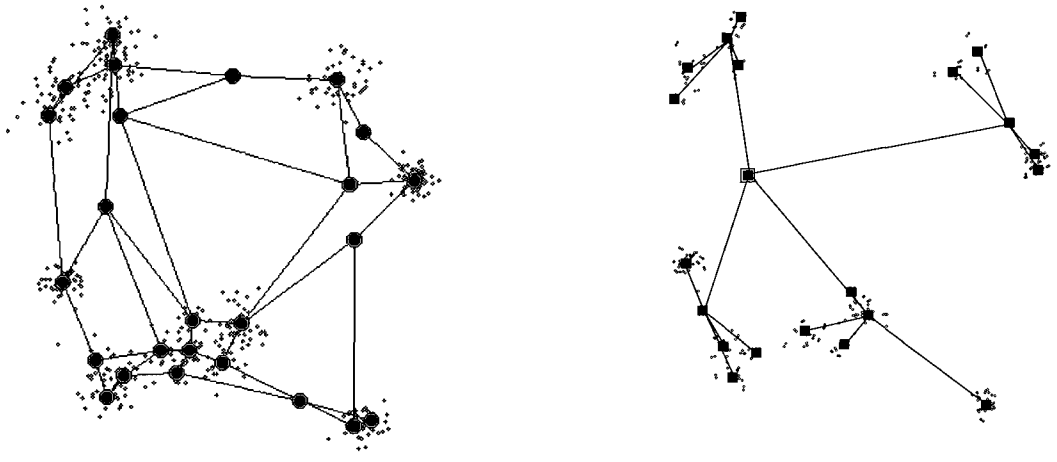
[6]The data set and the SOM training were extracted from the software DemoGNG version 1 5, available at the URL `http //www neuroinformatik ruhr-uni-bochum de/ini/VDM/research/gsn/DemoGNG/GNG html`



(a) After 0 iter (b) 10,000

(c) 100,000 (d) 300,000

Figure 3 6 TTOSOM-based 4-ary tree topology learned from a "square" distribution.



(a) SOM (b) TTOSOM

Figure 3 7: The result of invoking the SOM and TTOSOM on discrete data distribution using an almost identical number of nodes.

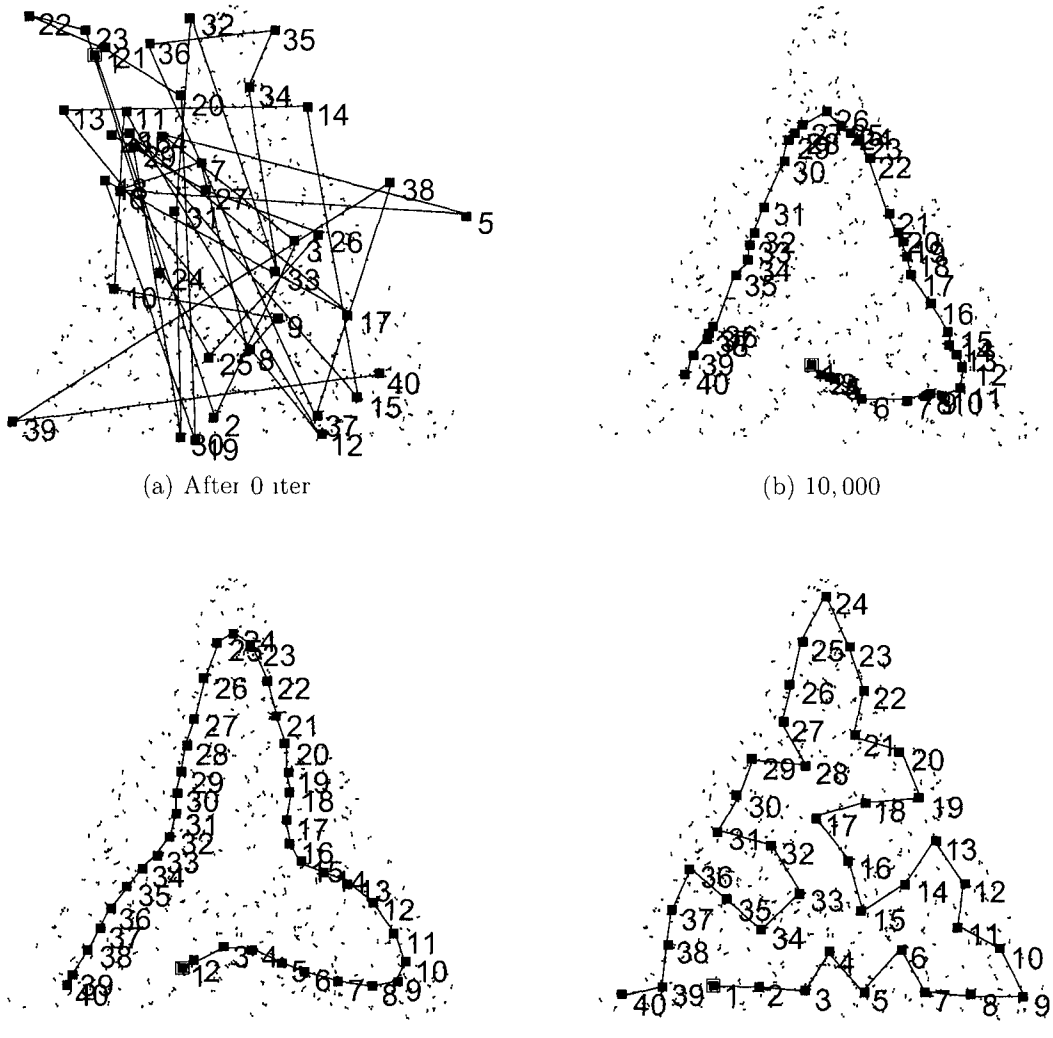
For the case of using a 1-dimensional tree (i e , a list) as per Figure 3.8, our algorithm is capable of not only representing the whole distribution of the input samples, but also of preserving *this* "tree-like" topology. Indeed, on termination, the indices of the codebook vectors are arranged in an increasing order as seen in Figure 3.8d In this case, the one-dimensional list of neurons is evenly distributed over the triangle, preserving the original properties of the 2-dimensional object and also presenting a shape which reminds the viewer of the so-called Peano curve [148]

As we can see, this one-dimensional representation is achieved by defining a tree with a single child per node. This example can be effective in distinguishing our method over the ET. If the same data was processed using the ET, the final configuration of the codebook vectors will not represent the input set accurately, even if the topology is preserved The reason for this is as follows: In the first stage of the algorithm, the ET's root is trained for a certain number of iterations, after which the root will be most likely close to the center of mass of the input data cloud. The ET then "freezes" the location of this unit, which implies that it is no longer trained, and

thus *this* codebook vector remains static It no longer participates in the competitive learning process since our assumption of using a 1-ary tree is in place After freezing this unit, the ET resorts to a splitting operation The children are created, and in this particular case only a single child is created and trained In this phase, the child does not compete with the parent because the latter is static Consequently, the random samples taken from the input set will again tend to place the child close to the center of mass of the data cloud As this process is repeated for the subsequent children, all of them will be placed near to the center of the data distribution

From this perspective, one of the best advantages of the ET is simultaneously a drawback Freezing the unit after a certain time and then splitting it lends to the algorithm the ability to only train the leaf-nodes This property is cleverly exploited by the authors of [143] so as to accelerate the search for the BMU in $O(\log n)$ time On the other hand, the frozen units are excluded from the subsequent competition, and open the possibility of being ill-formed codebook vectors Briefly put, the 1-ary tree case is an extreme one, and although for the general n-ary tree (for $n > 2$) the ET possesses a good performance, in the linear case our algorithm outperforms the ET

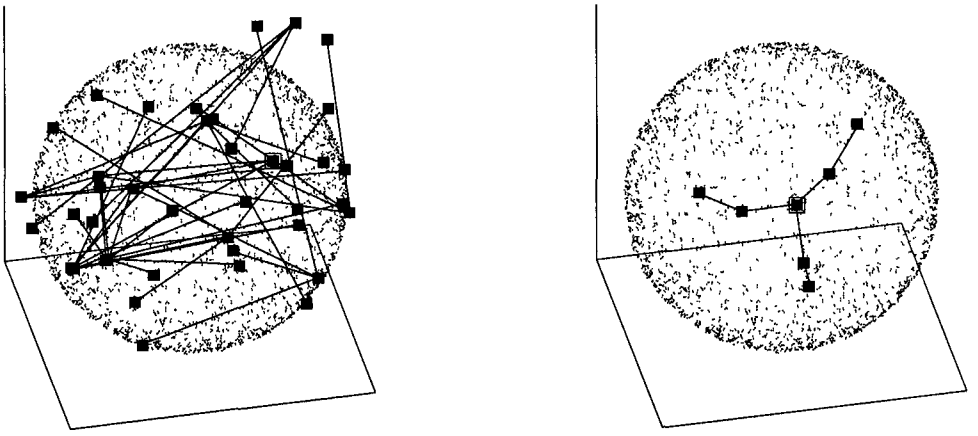
Figure 3 9 presents an example in a higher dimensional space, where the TTOSOM attempts to learn a unit *sphere* In this experiment, so as to display the capabilities of the algorithm, we decided to use the same 3-ary tree configuration that was utilized in the triangle-shaped distribution shown in Figure 3 5, i e , a complete 3-ary tree of four levels, thus containing 40 nodes At the beginning, as shown in Figure 3 9a, the initial codebook vectors are randomly distributed within the circumscribed unit cube As the training proceeds, Figure 3 9b depicts the situation where all the nodes converge into a single point which is located roughly in the center of the sphere Figure 3 9c, displays the case when the neurons belonging to lower levels of the tree begin to spread evenly inside the sphere The situation after training is shown in Figure 3 9d, where the tree expands uniformly in the interior of the sphere, suggesting the spherical shape of the original data manifold Given the symmetries of the sphere, other tree topologies can also be utilized to learn its structure



(a) After 0 iter (b) 10,000

(c) 100,000 (d) 300,000

Figure 3.8: TTOSOM-based 1-ary tree (list) topology learned from a "triangular" distribution



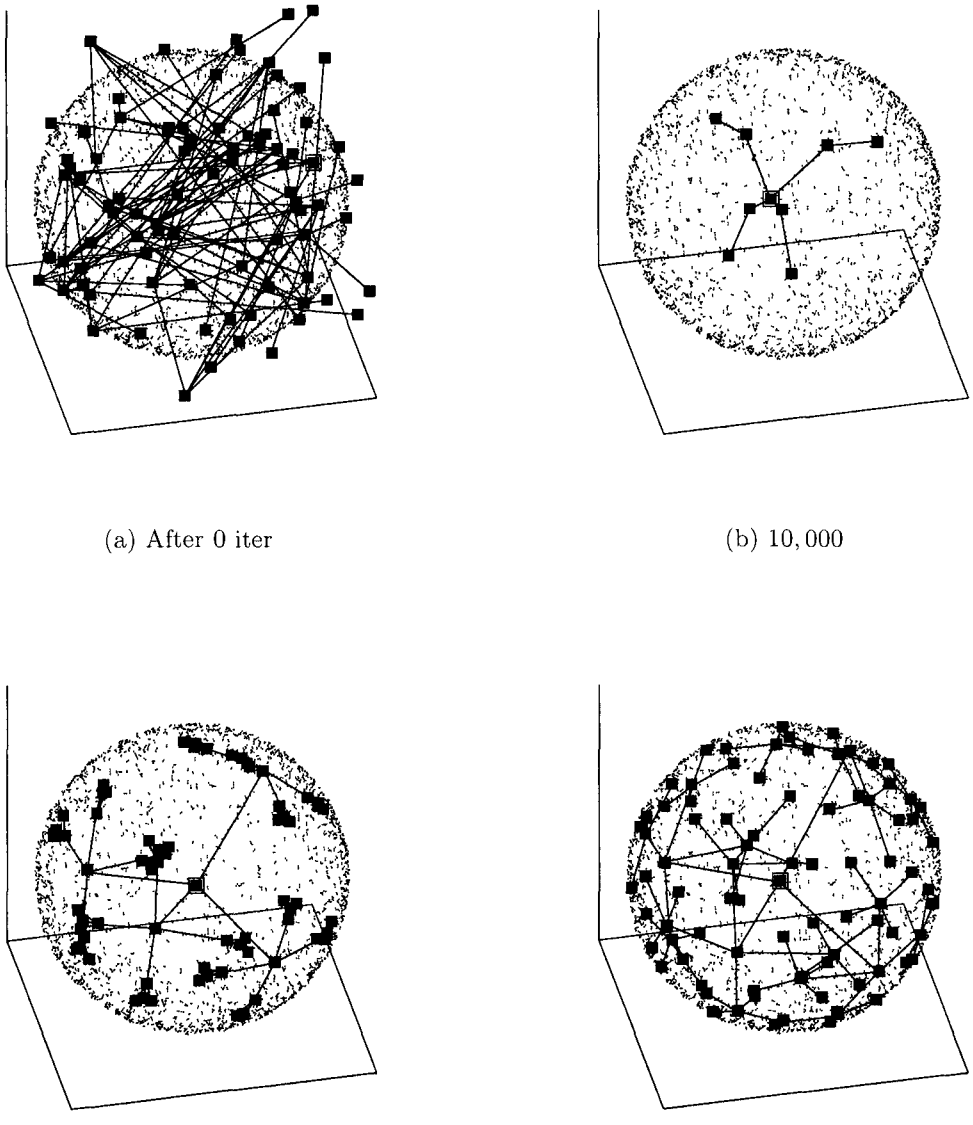
(a) After 0 iter.

(b) 10,000

(c) 100,000

(d) 300,000

Figure 3.9: The same 3-ary tree utilized for learning the triangle in Figure 3.5 now learns the unit sphere in 3-dimensions.



(a) After 0 iter

(b) 10,000

(c) 100,000

(d) 300,000

Figure 3.10: The same 4-ary tree utilized for learning the unit square in Figure 3.6, now learns the unit sphere in 3-dimensions.

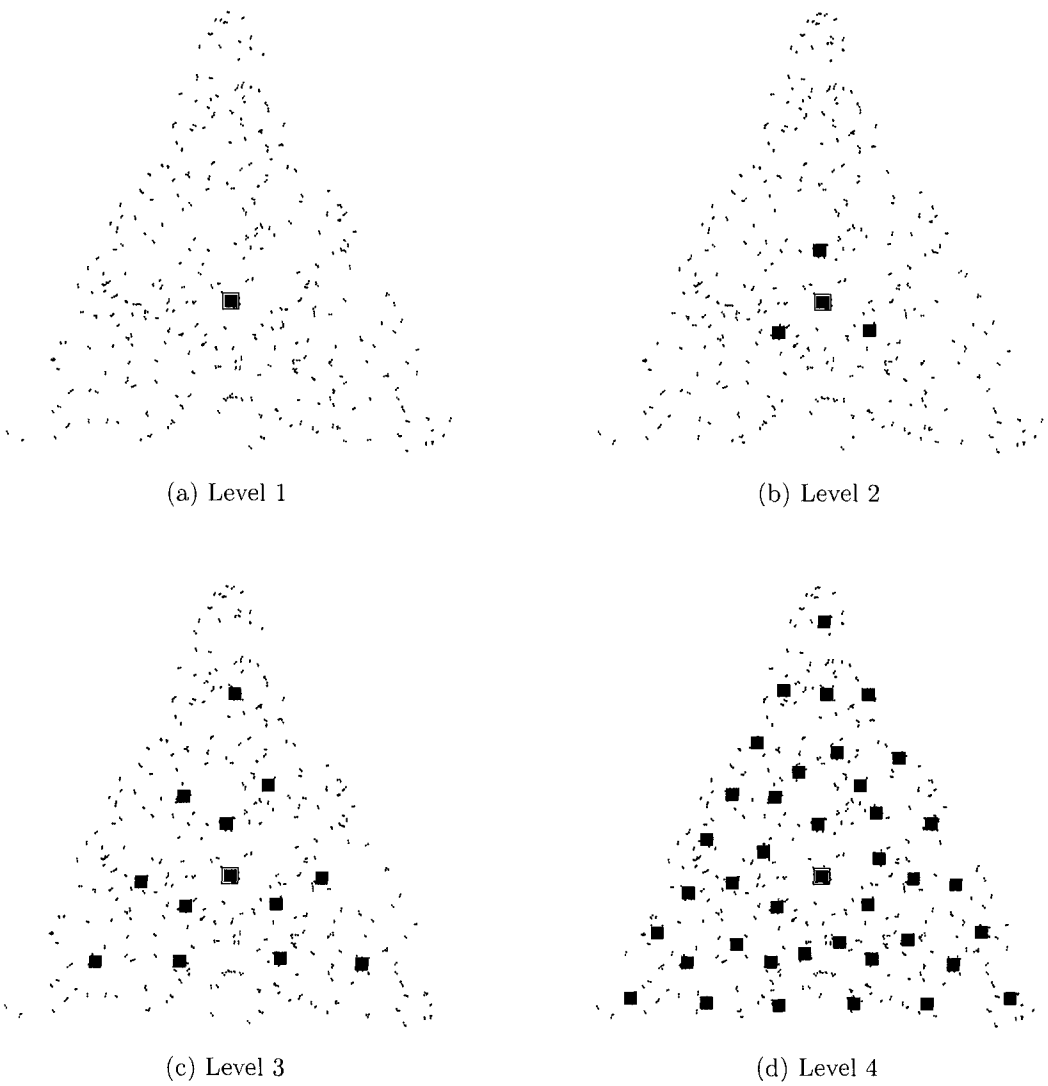
3.3.2 The Hierarchical Representation

Another distinct advantage of the TTOSOM, which is not possessed by other SOM-based networks, is the fact that it has hologram-like properties In other words, although the entire tree specified by the user can describe the cloud of data points as per the required resolution, the same cloud can be represented with a more coarse resolution by using a lesser number of points Thus, if we wanted to represent the distribution using a single point, this can be adequately done by just specifying the root of the tree A finer level of resolution will include the root and the second level, where these points and their corresponding edges, will represent the distribution and the structure Increasingly finer degrees of resolution can be obtained by including more levels of the tree We believe that this is quite a fascinating property

To clarify this, consider the triangular distribution in Figure 3 11, which is the same distribution of Figure 3 5 Figure 3 11a shows how the cloud can be represented by a single point, i e , the root In Figure 3 11b it is represented by 4 nodes which are the nodes up to the second level If we use the user defined tree of four levels, the finest level of resolution will contain all the 40 nodes, as displayed in Figure 3 11d

To demonstrate the analogous effect in 3-dimensions, we similarly, consider the spherical distribution depicted in Figure 3 12, which corresponds to the distribution of Figure 3 6 When the cloud is represented by only one point, i e , the root, a rough approximation of the data distribution can be obtained, as per Figure 3 12a, which shows the root of the tree located roughly in the center of the sphere In Figure 3 12b the entire distribution is represented by 4 nodes which corresponds to the nodes in the first and second levels of the tree If we include nodes up to the fourth level of the tree, the representation at this resolution will contain all the 40 nodes, as displayed in Figure 3 12d

From these results, we believe that the representations obtained for increasing resolutions are both intriguing and remarkable



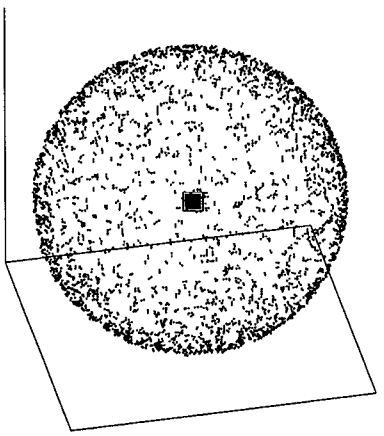
(a) Level 1

(b) Level 2

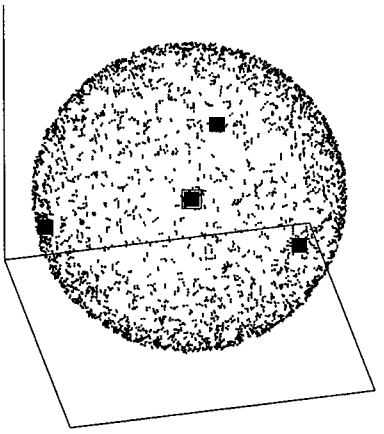
(c) Level 3

(d) Level 4

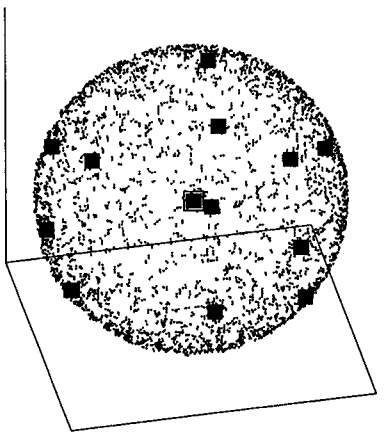
Figure 3 11: Multi-level resolution of the results shown in Figure 3.5.



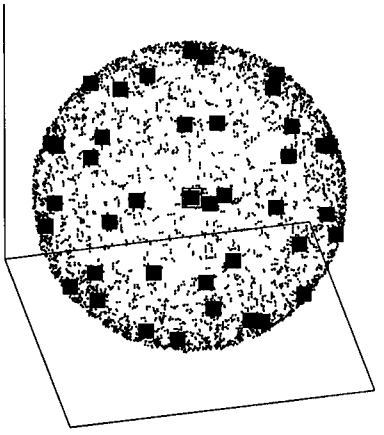
(a) Level 1



(b) Level 2



(c) Level 3



(d) Level 4

Figure 3 12 Multi-level resolution of the results shown in Figure 3 9 for a sphere in 3-dimensions

3.3.3 Skeletonization

Intuitively, the objective of skeletonization is to construct a simplified representation of the global shape of an object In general, such a skeleton is expected to contain much less points than the original image and should be a thinned version of the original shape According to the authors of [141], skeletonization in the plane is the process by which a 2-dimensional shape is transformed into a 1-dimensional one, similar to a "stick" figure In this way, skeletonization can be seen as a dimensionality reduction technique that captures local object symmetries and the topological structure of the object The applications of skeletonization are diverse, including the fields of Computer Vision and PR

There are different types of algorithms that attempt to solve the skeletonization problem Traditional skeletonization approaches assume the connectivity of the pixels that constitute the image Thus, they are inadequate when pixels in the image lack connectivity, as can happen all too often, as a consequence of an inappropriate manipulation of the image, noise, or the intrinsic nature of the data itself In such cases, traditional methods may not perform well, and advanced techniques that can also inherently process *structure*, are needed

SOM variations have been used to tackle this situation when points are sparse in the space [46, 165] In [46] the authors used a GNG-like approach, while in [165], the authors recommend the use of a Minimum Spanning Tree (MST) which is calculated over the *positions* of the codebook vectors, followed by a post-refinement phase that adds and deletes edges

We now advocate a completely different SOM-like strategy, namely the TTOSOM Indeed, as we perceive it, the structure generated by using the TTOSOM can be viewed as an endo-skeleton of the given data set, in the sense that it conforms to an internal framework that support the "soft parts" of the original object This skeleton meets at pseudo-joints, which are, in our representation, the nodes of the tree As the whole structure of these pseudo-bones are dependent on the position of the nodes in the feature space, a single learning iteration of the TTOSOM is capable of affecting the overall shape of the skeleton, and on convergence it will self-organize so as to assimilate the fundamental properties of the primary representation

In this context we propose that the edges can be seen as pseudo-bones Each pseudo-bone is defined by two codebook vectors in their extremes, and contrary to what happens with real bones, these pseudo-bones have a great measure of flexibility, and also the ability to contract or enlarge as a consequence of the movement of the nodes that define their respective extrema It is also worth mentioning that the movement of a joint will have the implication of the modification of at least one edge Thus, when a node is moved, all the edges associated with the children and parent will change accordingly, modifying the shape of the inferred skeleton The difference between using the SOM-like philosophy [165] and the TTOSOM lies exactly here A SOM-like algorithm will change the edges of the skeleton but *only* as the algorithm dictates as per the MST computed over the nodes and their distances in the "real" world, i e , the feature space As opposed to this, a TTOSOM like structure can modify the skeleton as dictated by the particular node in question, but also *all* the nodes tied to it by the BoA, as dictated by the *user defined tree*, i e , the link distances

The reader can appreciate, in Figure 3 13, the original silhouette of a rhinoceros, a 2-dimensional person as well as a 3-dimensional person All three objects were processed by the TTOSOM using exactly the same tree structure, the same schedule for the parameters, and without any post processing of the edges From the images at the lower level of Figure 3 13 we observe that, even without any specific adaptation, the TTOSOM is capable of representing the fundamental structure of the three objects in a "1-dimensional" way effectively The figures at the second level display the neurons without the edges In this case, it can be seen that our algorithm is also capable of granting an intuitive idea of the original objects by merely looking at the points

A potentially interesting idea is that of mixing the hierarchical representation of the TTOSOM presented in Section 3 3 2 with its skeletonization capabilities We propose that in this case, the user will be able to generate different skeletons with different levels of resolution, which we believe, can be used for managing different levels of resolution at a low computational cost for applications in the fields of geomatics, medicine and video games

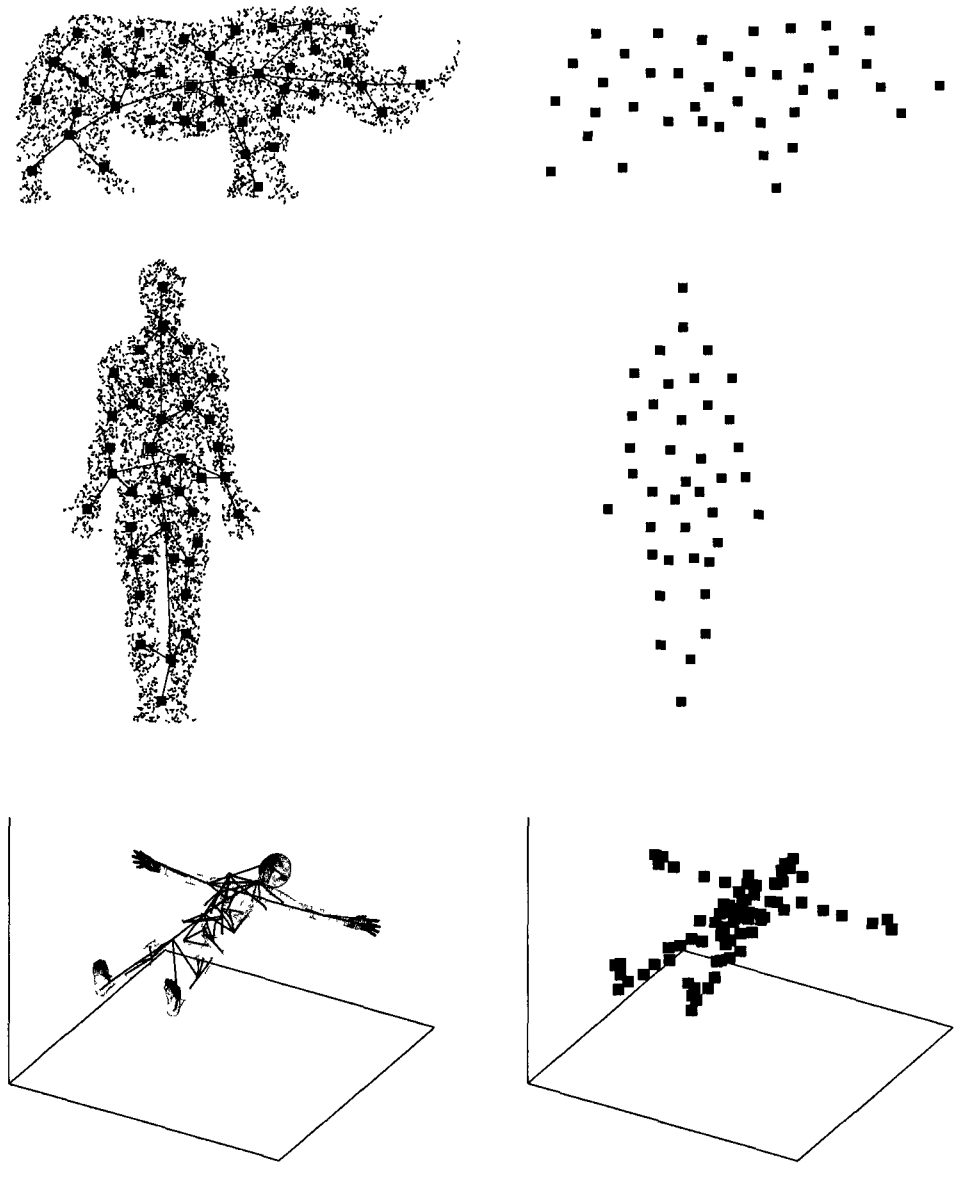


Figure 3 13. The result of a skeletonization process for the silhouettes of various shapes using the TTOSOM, namely, of a rhinoceros, a 2-dimensional person, as well as a 3-dimensional person.

3.3.4 Clustering and Pattern Recognition Capabilities

Additionally, we have examined the performance of the TTOSOM in clustering and PR applications for real-world data The well known Iris dataset[7] was chosen for this purpose The data set gives the measurements (in centimeters) of the variables which are the sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of the iris family The species are the Iris Setosa, Versicolor, and Virginica

Since the TTOSOM is an unsupervised learning algorithm, in order to show the power of our method, we decided to invoke it using exactly the same configuration employed to learn the triangle and sphere displayed in Figures 3 5 and 3 9 respectively, i e , the same underlying tree topology of a complete 3-ary tree of depth 4 By this we attempt to show examples of how exactly the *same* tree configuration can be utilized to learn the structure from data belonging to the 2-dimensional, 3-dimensional and also 4-dimensional spaces After executing the TTOSOM, each of the main branches of the tree were migrated towards the center of mass of the cloud of points in the hyper-space belonging to each of the three categories of flowers, respectively It is important to mention that the TTOSOM did not know that the data came from 3 different categories, rather this information was inferred by it We find this result quite fascinating, indeed!

After convergence, each of the samples were associated to the closest neuron, thus generating a set of clusters that are arranged in a hierarchical manner Moreover, if the true labels of the original data set are provided, the TTOSOM can be further utilized to perform PR, by identifying each of the neurons as a representative of a particular class The results of this classification scheme applied to the Iris data set are summarized in Table 3 1 The information about each category of the Iris families is given in the first row Analogously, each column considers one of the three main branches of the tree The table indicates how many elements of a particular kind of Iris were best represented by a particular branch of the tree For instance, all the data points which were best represented by neurons belonging to Branch1 happened

[7]The Iris dataset was obtained from The R Project for Statistical Computing, available at the URL `http //www r-project org/`

	Setosa	Versicolor	Virginica	Accuracy
Branch1	50	-	-	1 00
Branch2	-	48	2	0 96
Branch3	-	2	48	0 96

Table 3 1 The power of the TTOSOM for learning the characteristics of the Iris dataset The underlying tree structure was exactly the same one employed for learning the triangle and the spheres of Figures 3 5 and 3 9, respectively

to be of class Iris Setosa Moreover, all the 50 instances of Iris Setosa found in the data set are represented by Branch1 The last Column of Table 3 1 summarizes the accuracy of recognizing a particular category of Iris

The experimental results shown in Table 3 1, not only demonstrate the potential capabilities of the TTOSOM for performing clustering, but also suggest the possibilities of using it for pattern classification According to [57], there are several reasons for performing pattern classification using an unsupervised approach Moreover, due to the multi-resolution nature of our proposed strategy, different levels of the tree may be employed for performing increasingly accurate classification tasks We are currently investigating such a classification strategy

3.3.5 Theoretical Analysis

Although the applications of the new method have been demonstrated, we feel that it is appropriate to close this section by mentioning the hurdles that will be encountered in analyzing the asymptotic distribution of the TTOSOM[8] In all brevity, we believe that even the tools for such an analysis have not been fully developed, as we explain below

At the outset, we mention that Bauer *et al* [18] explain, in great detail, a tool called the Topographic Product, utilized for the measurement of the topology preservation These authors also show the power of this tool by applying it on different artificial

[8]We are very grateful to an anonymous Referee who pointed us to these references, and for the insight that he provided We totally concur with him in that it is difficult (if not currently impossible) to mathematically solve the explicit equations for the final distribution and topology for our present solution

and real-world data sets, and also compare it with different measures to quantify the topology [17] Their study concentrates on the traditional SOM, implying that the topologies evaluated were of a "linear" nature, with the consequential extension to 2-dimensions and 3-dimensions by means of grids only In [85], Haykin mention that the Topographic Product may be employed to compare the quality of different maps, even when these maps possess different dimensionality However, he also noted that this measurement is only possible when the dimensionality of the topological structure is the same as the dimensionality of the feature space Further, tree-like topologies were not considered in their study To be more precise, most of the effort towards determining the concept of topology preservation for dimensions greater than unity are specifically focused on the SOM [17, 18, 35, 42, 88, 103], and do not define how a tree-like topology should be measured nor how to define the order in topologies which are not grid-based Thus, we believe that even the tools to analyze the TTOSOM are currently not available The experimental results obtained in this chapter, suggest that the TTOSOM is able to train the NN so as to preserve the stimuli However, in order to quantify the quality of this topology, the matter of defining a concept of ordering on tree-based structure has yet to be resolved Although this issue is of great interest to us, this rather ambitious task lies beyond the scope of our present manuscript

3.4 Conclusions

In this chapter we have proposed a schema called the Tree-based Topology Oriented SOM (TTOSOM) by which the operator/user is able to impose an *arbitrary*, user-defined, tree-like topology onto the codebook vectors of a SOM This constraint leads to a neighborhood phenomenon based on the user-defined tree, and, as a result, the BoA becomes radically different from the ones studied in the previous literature The map learned as a consequence of training with the TTOSOM is able to determine *both* the distribution of the data and its structured topology, interpreted through the perspective of the user-defined tree In addition, we have shown that the TTOSOM revealed the ability to represent the original data set in multiple levels of granularity,

and this was achieved without the necessity of computing the entire tree again Additionally, we discussed the capability of the TTOSOM to extract an skeleton, which is a "stick-like" representation of the image in a lower dimension of space Finally, we also showed how the TTOSOM can be utilized to perform and enhance pattern classification by using a scanty structured representation of the original training sets All these properties have been confirmed by numerous experiments on a diversity of data sets

Chapter 4

Pattern Recognition using the TTOSOM

4.1 Introduction

In the last chapter we introduced the TTOSOM as a mechanism by which the information contained in any given dataset can be compactly represented by a set of representative neurons. The goal of the exercise was to have these neurons possess the stochastic distribution of the data points, and at the same time, preserve the topology and the structure of the data.

In this chapter we will venture to use the TTOSOM in pattern classification – in a matter that is quite novel.

As explained in Chapter 2, there are scores of algorithms which can achieve supervised pattern classification. Such algorithms assume the knowledge of well-defined training sets with a clear specification of the identity of each training sample. In the unsupervised model, as we have seen earlier, one does not provide the classifier with the class labels of the samples. Rather, the data is subjected to a clustering-like mechanism, which attempts to distinguish the training samples into the separate classes, subsequent to which a supervised classifier will be invoked.

The reader must observe that such a clustering action can be quite time consuming, especially if the dataset is large.

Our proposal to utilize the TTOSOM for classification attempts to bridge the two paradigms It is, rather, akin to the philosophy advocated by Zhu [176], which we will explain in a subsequent section What we propose is the following Using an unsupervised approach, we will first train a TTOSOM in which the neurons collectively obey the stochastic, topological and structural distribution of *all* the classes Subsequent to this juncture, we make use of the information provided in the labeled dataset By using this information, we assign a class label to every single node in the NN This can be done in numerous ways, but we have chosen to do it using a simplistic Euclidean criterion Observe that such an assignment partitions the space into its Voronoi regions, although the specification (or complete description) of these subspaces is unnecessary

On receiving the testing data, the task at hand is rather straightforward One nearly determines the closest neuron to the testing sample and assigns the sample to the corresponding class

Observe that once the TTOSOM has been computed, the complexity of the testing is linear, not in cardinality of the training set, but in the size of the TTOSOM tree

The reader will observe that such an nearest-neighbor type testing is similar to the ones used by prototype reduction schemes [102, 116] Such schemes represent the training data from each class by a small set of prototypes, which are subsequently used for classification In our case, however, the "prototypes" are the nodes of the TTOSOM which are not located in the feature space in a manner *unrelated* to each other Rather, they are constrained by the tree structure, the topology, and the distribution of *all* the classes

Our goal in this chapter is to see how such a philosophy will function One must, of course, realize that the accuracy that we obtain will never exceed that of the optimal Bayes classifier Indeed, we cannot expect an accuracy greater than that which a true nearest neighbor classifier yields, because we could be only using a small set, e g , about 25 representative points, instead of the entire set, which could consist of thousands of points Further, by starving the classifier from information of the class labels, one can expect the accuracy to be even less What is astonishing, however, is the fact that our "semi-supervised" TTOSOM-based classifier, achieves an accuracy

which is only marginally less than state-of-the-art supervised classifiers reported

With this as a background, we list below the contributions of the chapter

4.1.1 Contributions of the Chapter

1 We present a method that employs a tree-based NN for performing classification, by using both, unlabeled and labeled instances

2 Our method first learns the structure of the data distribution in an unsupervised manner, where the distribution involves all the classes collectively

3 After convergence, and once labeled data become available, our strategy tags each of the clusters according to the evidence provided by the instances

4 The classification strategy that we are proposing, unlike other neighborhood-based approaches, uses only a small set of prototype vectors whose cardinality can be much smaller than that of the input manifold

5 Our experimental results show that on average, the classification capabilities of our proposed strategy are reasonably comparable to those obtained by some of the state-of-the-art classification schemes that only use labeled instances during the training phase

6 The experiments also show that improved levels of accuracy can be obtained by imposing trees with a larger number of nodes

4.1.2 Organization of the Chapter

The remainder of the chapter is organized as follows In the next section we provide an a very brief explanation of the principal algorithms that are currently employed for supervised classification Subsequently, we present the details of the design and implementation of our TTOSOM-based classifier, and show how we can use it to perform pattern recognition Thereafter, we provide the experimental results that demonstrate the capabilities of the approach Finally, we conclude the chapter with a discussion of the results of our study

4.2 Classifiers

Typically, in classification tasks, one assumes that the samples in the dataset possess the Independently and Identically Distributed (IID) property, i.e., that the instances, which are viewed as random variables, are uncorrelated between each other, and that all are drawn from a state of nature possessing the same probability distribution. Of course, an additional assumption is that the data can be divided according to the different classes they belong to. Particularly, the problem of PR is to develop the so-called decision boundaries from the data, which define the portions of the space that separate the samples according to their "states-of-nature", and which will subsequently serve to label an instance whose class is unknown. The remainder of this section explains some of the state-of-the-art classifiers. In our description, we assume that we want to create a dichotomizer, i.e., to partition the data into *two* main subsets, each belonging to one of the two classes present in the dataset. However, the generalization to more than two classes is, in most of the cases, straightforward.

4.2.1 Bayesian Decision Theory

The core of the theory of Bayesian inference is based on the Bayes rule, which is one of the central theorems in learning theory [21]. This method quantifies a decision based on probabilities and the cost associated with the respective decisions, under the assumption that the probability values necessary to make the calculations are known (or can be learned).

If we must decide between two states of nature, ω_1 and ω_2, the model assumes that they occur with certain probabilities, $P(\omega_1)$ and $P(\omega_2)$, which are referred to as the *prior* probabilities. These probabilities encapsulate our previous knowledge about the chances of getting a sample of a certain category. In the hypothetical case that no extra information about the data has been provided, the prior probabilities represent the sole source of information helpful for taking a decision. Intuitively, if someone presents us with a sample, and nothing is told to us about it, we will choose to label it with the category which is more popular in the dataset. In other words, we will choose to label it as ω_1, if $P(\omega_1) > P(\omega_2)$, otherwise we categorize it as ω_2.

The latter is an example of a very simplistic decision rule In a Bayesian classifier, an instance $\mathbf{x}$ from the dataset, is considered to be a random variable, and its probability, $p(\mathbf{x}|\omega)$, known as the class-conditional probability or *likelihood*, conditions $\mathbf{x}$ based on the class to which it belongs If the prior probabilities are equal, then the largest value of the likelihood will serve to determine the decision about the category of a novel item

The Bayes rule, combines both, prior and likelihood, and from it, one can obtain the so-called *Maximum A Posteriori* (MAP) decision rule, which as follows

$$\Omega_{\mathsf{MAP}} = \arg\max_{\omega_i \in \boldsymbol{\omega}} \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})}, \tag{4 1}$$

where the denominator corresponds to the so-called total probability, which is common to all the different classes, and which plays a "normalization" role Thus, for decision purposes, it can be safely omitted, yielding to the following simplified (and equivalent) version [57]

$$\Omega_{\mathsf{MAP}} = \arg\max_{\omega_i \in \boldsymbol{\omega}} p(\mathbf{x}|\omega_i)P(\omega_i) \tag{4 2}$$

Equation (4 2), considers the prior and the likelihood criteria in the decision If the class-conditional probabilities are equal, the priors will determine the decision On the other hand, if we know that the categories are equally likely to be present, then the largest class-conditional probability will determine the best decision so as to minimize the probability of making an erroneous decision

4.2.2 The Naïve Bayes Classifier

In particular, the Naive Bayes (NB) classifier [57] simplifies the original Bayes theorem by assuming that the random samples not only possess the IID property, but additionally, that the features that describe the instances are also independent between each other Examining the scenario in greater detail, if the input vector is denoted by $\mathbf{x} = [x_1, x_2, \quad , x_d]$, the probability that $\mathbf{x}$ occurs given that it belongs to class ω_j is given as follows

$$p(\mathbf{x}|\omega_j) = \prod_{i=1}^{d} p(x_i|\omega_j) \tag{4.3}$$

By replacing this term in Equation (4.2), one obtains the most common form of the NB classifier, which is explicitly given by

$$\Omega_{\mathsf{NB}} = \arg\max_{\omega_j \in \boldsymbol{\omega}} P(\omega_j) \prod_{i=1}^{d} p(x_i|\omega_j) \tag{4.4}$$

The interesting feature of the NB method, is that even though the assumption that it is based on is very simplistic (and hence its name), in practice, the classifier performs surprisingly well in many real-world domains, offering competitive (and sometimes better) results when compared to more sophisticated learning paradigms

4.2.3 The Bayesian Belief Network

Another classifier rooted in Bayesian foundations, is the so-called Bayesian Networks (BNs), also known as Belief Networks The family of BNs are classifiers based on a Directed Acyclic Graph (DAG), in which each node of the graph corresponds to a random variable, and where the respective edges connecting the nodes represent the dependences between the different variables Given some evidence about the world, the BN estimates the conditional dependencies by employing statistical methods Since the theory of BNs is both mathematically rigorous and easy to understand, the PR paradigm is popular in the ML community [19]

More specifically, if $\mathcal{V}$ represents the set of variables of the system $V_1, V_2, \quad , V_k$, we label the corresponding variable at each node by the respective lowercase letter In this sense, even if the elements that variable $S = V_1$ can assume are discrete values (e g , s_1 ="smoker" and s_2 ="non-smoker") , they may be associated to real-numbered probabilities (e g , $P(s_1) = 0\,25$ and $P(s_2) = 0\,75$) If two nodes are connected, we say that they are related, and this is denoted by a directed edge emanating from one of them and which is incident on the other In this case, we say that the child node depends on the parent Viewed from the opposite perspective, we can state that the parent influences the child For example, if $H = V_2$ is a variable

that denotes the health of a person with h_1 ="healthy" and h_2 ="sick", we can link the health of the person with his smoking habits by formulating the conditional probability $P(h|s)$ In general, many parents can influence a particular child, and one child can be influenced by many parents However, once a node becomes a child it cannot influence its parent, since it will violate the definition of a DAG, which is essential for the computation of the conditional probabilities Additionally, when a node has no parents, its probability is assumed to be unconditional

The BN makes the assumption that each variable is independent of its non-descendants in the graph given the state of its parents [19], which is a simplification of the equation that dictates the Joint Probability Distribution (JPD) of the variables, and which provides an efficient way to compute the *a posteriori* probabilities If π_i denotes the set of parents of node i, the JPD is given by

$$P(V_1, V_2, \quad , V_k) = \prod_{i=1}^{k} P(V_i|\pi_i) \tag{4 5}$$

Equation (4 5) can be used to obtain a direct application of Bayes Theorem, making it possible for us to determine any configuration of the variables in the JPD However, in order to proceed, one has to specify the so-called Conditional Probabilities Table, which yields the probability of a particular variable occurring conditioned on its parents [57]

4.2.4 Instance-Based Classification

A different family of classifiers are the ones referred to as Instance-Based (IB) learners [2] This type of learning assumes that instead of building a probability model from the training samples, they are encapsulated and used to classify a novel item based on the similarity to its neighboring instances This family is characterized as being "lazy", since the computational calculations are deferred until a query is requested, and therefore it cannot be postponed anymore

Among this family of methods, a particularly powerful, yet simply strategy is the one referred to as the k-Nearest Neighbors (k-NNs) algorithm [57] The k-NN uses the Nearest Neighbor rule which assigns to $\mathbf{x}$ the label of its closest instance

The Nearest Neighbor rule is a suboptimal procedure However, it can be shown that given an unlimited number of points, its error will never be greater than twice the one produced by the Bayes decision rule [57] In the k-NN approach, the query instance is labeled according to the class that is most "popular" among the k closest instances positioned in the neighboring areas where the query instance is located To avoid ties, k, the single parameter of the algorithm, is chosen so as to be an odd number, for the 2-class problem

4.2.5 Decision Trees

Up to this juncture, the present thesis focuses on feature vectors that are positioned in the domain of the real numbers, where a natural notion of measurement is defined Although most of the problems in pattern classification strategies fall into this class, there is a realm that also considers the so-called "nominal" features A nominal feature, as opposed to a numerical feature, does not necessarily include the notion of numerical measurement or even ordering among the diversity of the values

A Decision Tree is a particular type of tree that classifies instances by sorting them based on its features [57] Here, each node in the tree represents a feature, and the branches emanating from it correspond to the different values that can be assumed The problem of constructing the optimal decision tree is NP-complete and therefore, researchers have attempted to devise strategies that are able to built almost optimal trees [107] Ideally, the feature that best divides the data should be the root of the tree The literature contains many strategies for finding the "most valuable" attributes [107], but according to the authors of [138], this is still an open problem Decision trees naturally classify datasets that include nominal features [57]

If two trees offer the same prediction accuracy, the one with a lesser number of leaves will be preferred For this and other reasons, to improve comprehensibility and to avoid over-fitting, researchers have employed different strategies to make the tree smaller [33], and one of the best technique is known as pre-pruning which consists of not permitting the tree grow too much [32]

One of the most popular decision tree methods is the C4 5 [151] scheme The

C4 5 algorithm utilizes a special type of pruning called *subtree raising* whose goal is to identify a node (and its respective subtree) that can be replaced by one of its descendants (and its respective subtree) In general, in decision trees, the pruning decision is made according to an estimate of the error at the node, which is obtained during a validation stage Particularly, the error estimation followed by C4 5 consists of analyzing the whole training data, considering the class of each sample at a particular node, and assigning to the node a class based on a voting scheme

4.2.6 LVQ1

Closely related to VQ [119] and the SOM [103] (earlier discussed in Section 2 5) is the method called Learning Vector Quantization (LVQ) [103, 102] As opposed to VQ and the SOM, the LVQ family uses the *supervised* learning paradigm Distinct from the SOM, the LVQ does not include the concept of the BoA, implying that it does not consider neural topology In other words, the LVQ is an example of hard-competitive learning where no topological preservation is expected According to its architect [103], the LVQ is primarily intended for PR, and its only purpose is to identify class regions in the input space

The traditional approach in statistical PR is to develop an approximation for $p(\mathbf{x}|\omega_i)P(\omega_i)$, and using this information, the most likely class for the testing pattern is obtained by invoking Equation (4 2) LVQ operates in a different manner Once the codebook vectors are placed, each of them is assigned a class label When an instance is presented to the model, it is tagged with the label of its closest codebook vector in the Euclidean space

In general, the migration of the codebook vectors in LVQ are a result of a reward-punishment philosophy In particular, the LVQ1 will move the sample closer to the codebook if their labels match, as expressed in Equation (2 16) Otherwise, i e , if the category of the sample and the prototype are not equal, the sample is migrated in the same amount as in Equation (2 16), but in the opposite direction (the second term of Equation (2 16) has a negative sign)

4.3 Performance Metrics for Comparing Classifiers

In this section we will briefly discuss different metrics for analyzing the performance of a classifier For simplicity, the discussion focuses on binary classification, where the first class is labeled as "positive" (P) and the second class is labeled as "negative" (N) A classifier generates a mapping from the so-called testing instances to their respective predicted classes Further, the predicted class may match the true label of the instance or not When a single instance is provided to the classifier, four possible outcomes are possible [59]

1 The predicted class is P and the true label was P $\Rightarrow$ Accurate Prediction

2 The predicted class is N and the true label was P $\Rightarrow$ Misclassification

3 The predicted class is P and the true label was N $\Rightarrow$ Misclassification

4 The predicted class is N and the true label was N $\Rightarrow$ Accurate Prediction

When a set of testing instances are provided to the classifier, its outcome can be summarized in the so-called Confusion Matrix (CM), which is 2-dimensional matrix that shows the predicted and actual classifications In general, the CM is of size $C \times C$, where C is the number of different label values [101] In particular, for the problem of deciding between two classes, the 2×2 CM associated with it is depicted in Figure 4 1 The numbers in the diagonal of the matrix represent the correct decisions made by the classifier, and the non-diagonals represent the inaccurate outcomes, i e , when the classifier is "confused"

From the CM, it is possible to obtain many of the most frequently-used metrics [59], some of which are briefly explained below

The most common metric is the so-called **accuracy** (acc), which is defined as the summation of all the elements in the diagonal of the CM, divided by the total sum of the elements of the matrix For the binary case, it is as follows

$$acc = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}} \tag{4 6}$$

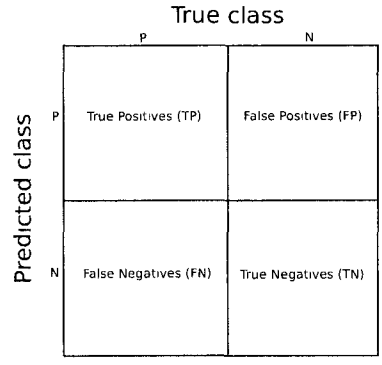


Figure 4.1: The Confusion Matrix (CM) for a binary classification problem.

The **true positive rate** (*tpr*) (also known as **recall**, **hit rate** or **sensitivity**) measures the proportion of actual positives that are correctly identified, and is defined by:

$$tpr = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{4.7}$$

Analogously, the **false positive rate** (*fpr*) (also known as **false alarm rate**) measures the proportion of negatives incorrectly classified, i.e.:

$$fpr = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \tag{4.8}$$

Additionally, the **specificity** (*spec*) measures the proportion of negatives which are correctly identified, and it is given as follows:

$$spec = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}}. \tag{4.9}$$

Observe that the specificity is equal to $(1 - fpr)$. Further, the **precision** (*prec*) is defined as the number of items that the classifier identified as being positive, divided by the total number of elements that truly belong to the positive class. It is given as follows:

$$prec = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{4.10}$$

Even though, the classification accuracy is still the most common used measurement in the ML community [48], in some fields, such as in Information Retrieval (IR), the data is extremely unbalanced, and the relevant class can be found in as few as 1% of the documents [124]. In such circumstances, a more informative measurement is required. A single measure that yields a trade-off between the precision and the recall is the so-called **F-measure**, defined as the weighted harmonic mean of these indices, and is given as follows [124]:

$$\text{F-measure} = \frac{2}{\alpha\frac{1}{\text{precision}} + (1-\alpha)\frac{1}{\text{recall}}} = \frac{(\beta^2+1)\text{precision} \times \text{recall}}{\beta^2\text{precision} + \text{recall}}, \tag{4.11}$$

where $\beta = \frac{1-\alpha}{\alpha}$. The default balanced F-measure equally weights the precision and the recall, and occurs when $\beta = 1$ (or equivalently when $\alpha = \frac{1}{2}$). In that particular case, Equation (4.11) becomes:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}. \tag{4.12}$$

An additional concept utilized to measure the quality of classifiers is the Receiver Operating Characteristics (ROC) curve, which plots the true positive rate (or sensitivity) against the false positive rate (which is equivalent to $1 - spec$) [124]. An ROC curve is useful for organizing classifiers and also for analyzing their performance [31]. It always goes from the bottom-left to the top-right of the graph. A good classifier is expected to climb quickly on the left side. An ROC curve is a *graphical* representation of classier's performance, when, to compare it with other classifiers, one needs to report a single scalar value. A common method is to calculate the Area Under the ROC Curve (AUC). Theoretically, the AUC ranges form 0 to 1, however, since a "random guess" will produce an AUC=0.5 (a straight line from the bottom-left to the top-right in the ROC graph), no realistic classifier should go below this quantity [59].

The literature also includes a number of other metrics for quantifying the quality of a classifier. For a comprehensive examination, the interested reader is requested to consult [124, 14].

In this study, we shall utilize the most simple and widely-used approach, i e , the accuracy of the classifier given by Equation (4 6) The rationale for this is the following In the strictest sense, we are not so interested in the performance of the TTOSOM *per se* Rather, we are interested in evaluating the various algorithms *comparatively*, and to demonstrate that by utilizing the TTOSOM we are able to achieve a comparable performance (whatever the metric used), even though the data is represented much more compactly than if we used the original datasets Thus, even though the accuracy measure is, in many contexts, inadequate, our experience is that the inferences gleaned from using it are identical to those obtained by using a more elaborate measure like the AUC

4.3.1 Stochastic Sampling

Stochastic Sampling is a family of methods that deal with the random selection of a subsets of individuals from a population From a PR perspective, and according to Dara *et al* [45], in real-world applications, it is not uncommon that the so-called testing samples only become available once the ML system is already in operation For this reason, often researchers utilize some kind of validation to measure the performance of a particular classifier One of the most popular validation techniques is the one referred to as "Stratified 10-fold cross-validation" In 10-fold cross validation the training samples are roughly divided into ten equal partitions Each fold is further used for testing the classifier, while the remainder nine are employed for training The process is then repeated for each of the folds The term stratified, comes from the statistical concept known as "stratified sampling", which is a sampling method that draws items from the different categories so as to obtain relatively homogeneous subgroups

4.4 The TTOSOM-Based Classifier

As stated earlier, the TTOSOM, described in the previous chapter is central to the classifiers designed presently The remainder of this section concentrates on a "wrapper" method known as Cluster-and-Label, as well as the details of the design of the TTOSOM-Based classifier that we propose

4.4.1 Cluster-and-Label

Zhu, in [176], proposed the concept that clustering algorithms could be employed to perform pattern classification As per his solution methodology, one alternative is to perform classification by applying the so-called Cluster-and-Label method explained below Prior research to the latter approach includes [45, 47, 73], among others

Algorithm 17 `Cluster-and-Label`($\mathcal{A}_C$,$\mathcal{A}_S$,$\mathcal{X}_U$,$\mathcal{X}_L$)

Input:

i) $\mathcal{A}_C$, the clustering algorithm
ii) $\mathcal{A}_S$, the supervised learning algorithm
iii) $\mathcal{X}_U$, the set of unlabeled instances
iv) $\mathcal{X}_L$, the set of labeled instances

Output:

i) A set of labels $\mathcal{Y}_U$ of the unlabeled samples $\mathcal{X}_U$

Method:

1 Cluster the unlabeled samples in $\mathcal{X}_U$ using the clustering algorithm $\mathcal{A}_C$
2 For each cluster C_i determine the subset $\mathcal{X}_L^i$ of labeled samples falling into C_i
3 Predict the label $\mathcal{Y}_C^i$ of each cluster C_i based on $\mathcal{X}_L^i$ and the algorithm $\mathcal{A}_S$
4 Assign the label $\mathcal{Y}_C^i$ to all the unlabeled samples falling into the cluster C_i

End Algorithm

Given a clustering algorithm $\mathcal{A}_C$, a set of labeled instances $\mathcal{X}_L$, a set of unlabeled instances $\mathcal{X}_U$, and a supervised learning algorithm $\mathcal{A}_S$, the Cluster-and-Label method, described in Algorithm 17, works as follows First, we identify the clusters of the input manifold using the clustering algorithm $\mathcal{A}_C$ Secondly, we determine which of the labeled samples fall in each cluster For each cluster we determine a decision boundary based on the supervised algorithm $\mathcal{A}_S$, and the labeled samples assigned to

that cluster, which, in turn, allows the prediction of the label of every cluster Finally, each uncategorized item is labeled according to the predicted class of the cluster in which it is contained

According to the author of [176], the performance of this approach is dependent on the capabilities of the clustering algorithm to mimic the properties of the original data distribution

4.4.2 The Algorithm

As stated earlier, our aim is to devise a classifier that works in two stages First of all, we will learn the data distribution and its structure in an unsupervised manner Thereafter, and in a second phase, we use some labeled items to tag the decision regions created previously The resultant TTOSOM-based classifier is described in Algorithm 18

In order to learn the decision boundaries, the TTOSOM algorithm is employed to train a tree structure so as to mimic the properties of the distribution of data points of *all* the classes, which is done without the necessity of providing the actual class labels of the items This corresponds to line 1 of Algorithm 18 The output of this initial phase is a TTOSOM tree structure, where each of the neural nodes are optimally placed in the feature space so as to glean the properties of the data distribution Our hypothesis is that these neurons represent regions of the hyper-space belonging to the same taxonomy, whose label is unknown The problem then is to accurately guess the actual label of that taxonomy

In the subsequent phase (see line 2 of Algorithm 18), our classifier determines which subset of the labeled instances are represented by each neuron In an ideal scenario, where a neuron is the BMU of instances belonging to the *same* category, the decision of tagging the unlabeled instances falling into the region will be trivial Unfortunately, as the authors of [45] point out, the latter does not occur necessarily For this reason a general mechanism is required which permits the *a posterior* decision about the class to be assigned to each neuron We thus maintain a statistical record of the number of instances belonging to each category that fall in a particular region

where a neuron is the BMU

The next phase (see line 3 of Algorithm 18), consist of a *supervised* phase in which class labels are assigned to each neuron in the tree From a statistical perspective, when the functions that dictate the probability of finding an item in a certain region of the hyperspace are known, the problem of deciding the category of a particular sample in the area, can be optimally determined by the function which maximizes its probability where the query item is positioned However, as per our problem statement, these probability density functions are not known, and so if one employs an approach like the one described above, we must have an "approximation of sorts" of such functions Fortunately, there is a simple way to have a rough estimation of the probability functions, i e , by using the information provided by the labeled training set Each neuron in the tree is thus assigned a label based on the k-NN rule [57] On closer inspection, the label of each neuron will be the one which occurs more frequently among the k nearest samples, where k is the number of data points for which the particular neuron is the BMU

Algorithm 18 `TTOSOM-Build-Classifier`($\mathcal{X}_U$,$\mathcal{X}_L$)

Input:

i) $\mathcal{X}_U$, the set of unlabeled instances

ii) $\mathcal{X}_L$, the set of labeled instances

Output:

i) A set of labels $\mathcal{Y}_U$ of the unlabeled samples $\mathcal{X}_U$

Method:

1 Train a TTOSOM tree using $\mathcal{X}_U \cup \mathcal{X}_L$

2 Determine the subset $\mathcal{X}_L^i \in \mathcal{X}_L$ for which the neuron i is the BMU

3 Label each neuron using $\mathcal{X}_L^i$ and the k-nearest neighbors rule, where $k = |\mathcal{X}_L^i|$

4 Label each sample in $\mathcal{X}_U$ *as per* the label of its respective BMU

End Algorithm

The final step (line 4 of Algorithm 18) consist in predicting the class label of each of the unlabeled instances In our method, this is done by taking a particular instance referred to as the "query" instance and finding its, BMU, i e , the closest neuron in the feature space, which is basically the notion of a VQ query The class label of the query instance will be same as the class label of the neuron which is "representing"

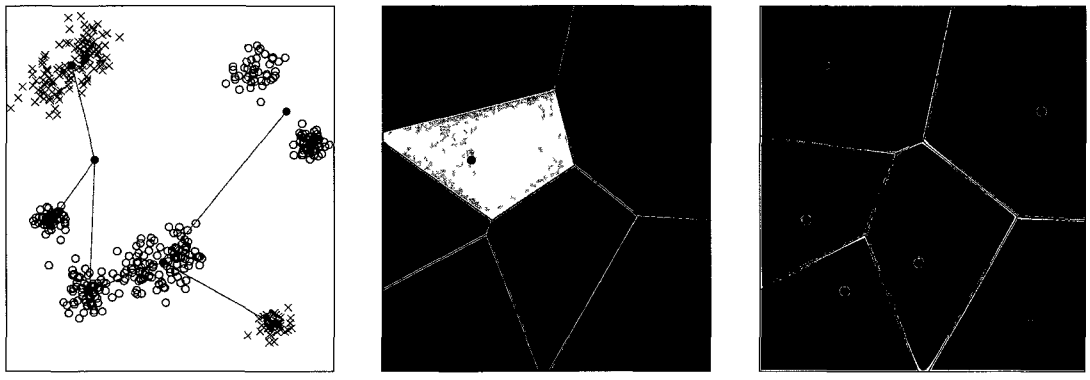
it Given the nature of the TTOSOM, some of the neurons act as a "joint" within the tree, reflecting the concentration of other smaller clusters in its vicinity It is likely, that these joints may not represent any sample in particular, and therefore, one needs an additional assumption in order to define its class label In our case, we have simply decided to exclude them from the competitive learning process In that sense, the search for the BMU in the classifier is slightly different from the one utilized by the TTOSOM (and inherited from the SOM) In this case, the label of the neuron is examined, and when it is undefined, the respective neuron is excluded from the "competition" process, which is a phenomenon that we call *supervised BMU search*

An example of classification using this scheme is depicted in Figure 4 2 In the example, 500 items belonging to two different classes are placed in such a way that it is not possible to devise an effective rectilinear dichotomization Figure 4 2a shows the samples and a complete binary tree of depth 3 which was trained with the TTOSOM Even though Figure 4 2a shows the original position of the samples and its labels, the respective class labels are unknown to the TTOSOM, which is essentially obtained in an unsupervised manner Figure 4 2b depicts the Voronoi regions induced by the neurons, i e , those areas of the space for which a particular neuron is the closest one The different colors (or shades) of the regions represent the majority class, which are determined by using the information in an additional set of labeled items Additionally, in this particular example, one of the neurons does not represent any labeled category As mentioned earlier, we have opted to exclude such a "joint" neuron, and as a result the Voronoi regions are modified as illustrated in Figure 4 2c

4.5 Experimental Setup

4.5.1 The Datasets

To test the ability of the TTOSOM for classifying items belonging to the real world domain, we have chosen a variety of datasets from the UCI Machine Learning repository [64]



(a) (b) (c)

Figure 4.2: TTOSOM-based classifier using *only* 7 nodes perfectly learns a dataset which is non-linearly separable. The learning is performed without the necessity of providing the actual class labels of the items. In Figure (a), the TTOSOM tree mimics the distribution and the structure of the points. Figure (b) shows the respective Voronoi tessellations, and Figure (c) displays the Voronoi regions excluding the so-called "joint" neurons.

The data sets used in these experiments have different numbers of output classes, ranging from two to ten. Additionally, their features pertain primarily to the continuous domain, whose dimensions varies from four up to thirty. The six data sets chosen for this experiment are described in Table 4.1.

Datasets	**Instances**	**Attributes**	**Classes**	**Problem Type**
Iris	150	4	3	classification
WDBC	569	30	2	classification
Wine	178	13	3	classification
Yeast	1,484	8	10	classification
Wine_Quality (red)	1,599	11	6	classification/regression
Glass	214	9	6	classification

Table 4.1: Datasets selected for the comparison of the classifiers.

The first dataset is the well-known Iris [61] set[1]. The features include the measurements (in centimeters) of the variables which are the sepal length, sepal width,

[1]Even though the Iris data set was utilized and introduced in an earlier chapter, we repeat some

petal length and petal width, respectively, for 50 flowers from each of 3 species of the Iris family The species are the Iris Setosa, Versicolor, and Virginica

The second dataset is the Wisconsin Diagnostic Breast Cancer (WDBC) set which contains information regarding breast cancer, and whose goal is the prediction of benign and malign tumors (i e , leading to a binary classification problem) The features are computed from digitized images of a Fine Needle Aspirate of a breast mass, which describe characteristics of the cell nuclei present in the image [64] The original attribute *ID number* was ignored in our experiments

The third dataset considered in this study is the Wine dataset The data was obtained as the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars This dataset was originally part of the system called PARVUS [170], and contains 13 features, all of them belonging to the continuous domain The goal consists of predicting the quality of the wine based on its chemical composition

The fourth dataset is the Yeast dataset, and represents a more challenging environment for the classifiers It contains 1,484 instances of yeast classified into 10 different categories [87] The objective is to predict the cellular localization sites of the proteins in question

The fifth dataset is the Wine_Quality dataset, which aims to predict the quality of red wine based on its chemical information [44] The samples come from the north of Portugal Given that the categories range from 3 to 8 in an increasing order, this dataset can be studied from a regression perspective as well

The final dataset considered in this study is the Glass dataset, originally provided by the USA Forensic Science Service The study was motivated by criminological investigations in which the glass left at the scene of a crime could be used as evidence [64] The dataset contains 6 types of glass, defined in terms of their oxide content (i e , Na, Fe, K, etc)

---

of these descriptions (in the context of PR and of the other data sets) so as to render this chapter complete

4.5.2 The Parameters

The respective parameters for the algorithms were rendered to be the same across all the different datasets, and no algorithm possessed parameter values that were tuned for the datasets used for the test. In particular, the three strategies based on VQ, i.e., the TTOSOM, the SOM and the LVQ1 utilized the same number of iterations (50,000). Additionally, they all used the same initial learning rate (0.5), and the radius of the BoA was chosen in such a way that initially, all the neurons were considered as part of the BoA, i.e., twice the depth of the tree in the case of the TTOSOM, and the width plus the height in the case of the SOM. Observe that LVQ1, as defined in [102], does not consider a BoA. As well, the three schemes utilized the same (linear) decaying schedule for its parameters.

4.6 Results

In this section, we present a general overview of the performance of each of the classifiers investigated for the different datasets mentioned in Section 4.5.1. Specifically, we present an assessment of the classifiers in terms of their *acc*, i.e., the percentage of correctly labeled instances.

The classifiers considered in this study are five supervised classifiers, BN, NB, C4.5, k-NN and LVQ1, and two "semi-supervised" classifiers, namely the TTOSOM and the SOM. The sampling method utilized to measure the *acc* was the stratified 10-fold cross validation.

4.6.1 Comparison to Other Classifiers

The results of the performance of the different classifiers (columns) across all the dataset (rows) is summarized in Table 4.2. Specifically, we are interested in the performance of our classifier on problems across a diversity of domains in which labeled and unlabeled data is available[2]. For example, Table 4.2 shows that the

[2]Our hypothesis is that one should use as much labeled data as is available. Since the datasets mentioned above are all composed of labeled instances, we have opted to use *all* this information in

TTOSOM classifier, using *only* 15 neurons, is able to accurately predict with an *acc* of 89 33% the correct label of the instances belonging the wine dataset On the other hand, the SOM classifies correctly the same dataset with an *acc* of only 67 98%

One possibility by which we can quantify the quality of our method is to consider the family of classifiers inheriting the VQ mechanism One such strategy that belongs to the supervised family is the LVQ1, while the SOM and the TTOSOM primarily learn the distributions using the unsupervised learning paradigm The three classifiers utilized the same parameters, which are described in Section 4 5 2 Besides, while the LVQ1 and the SOM utilized 128 neurons, the results shown for the TTOSOM include *only* 15 neurons As per our results, the TTOSOM, using only a small percentage of the neurons used in the SOM and LVQ1 (almost 10%), outperforms their recognition capabilities in *all* six datasets

Apart from the above, observe that the classification results offered by the TTOSOM are comparable to the ones obtained by the k-NN However, both approaches present important differences in how they perform learning First of all, the k-NN, being a supervised classifier, requires *all* the instances to be properly labeled Secondly, due to its "laziness", the computations for the k-NN are left until a query is performed, which implies that the *whole* manifold is visited so as to create the ordering of the samples, as per their proximity to the query samples On the other hand, the TTOSOM only requires a small subset of the tagged labels, and is able to learn from unlabeled samples Also, the query is done by using the TTOSOM tree and the respective labels of the neurons, and only requires the comparison with the total number of neurons, which is usually significantly smaller than the entire dataset Even though our method internally uses the k-NN to tag the neurons, we note that this is done only once, i e , when the tree is being learned, and furthermore, the computations are performed only for each neuron instead of the whole dataset

Another perspective by which we can compare the schemes is to consider the "most" competitive supervised classifiers In this case, except for the LVQ1, they outperformed the accuracy produced by the unsupervised strategies This is an expected behavior, because the supervised classifiers had access to the class labels of

the "supervised" phase of our algorithm

all the instances However, in environments where only few tags are available, traditional supervised classifiers struggle to extract useful information from unlabeled instances Indeed, experiments performed by Gabrys *et al* [73], showed that when a sufficiently large number of labeled instances were utilized, the semi-supervised schemes included in their study achieved levels of accuracy that were comparable to the ones obtained by the supervised classifiers that incorporated a much higher number of labeled samples

Dataset	TTOSOM15	BN	NB	C4.5	k-NN	LVQ1	SOM
iris	96 67	92 67	96 00	96 00	95 33	96 00	84 67
wdbc	93 32	95 08	93 15	93 15	96 66	92 09	90 51
glass	67 29	71 96	49 07	67 76	67 76	61 22	63 08
wine	89 33	98 88	97 19	93 82	94 94	74 16	67 98
yeast	54 18	56 74	57 61	55 86	54 78	24 33	46 16
wine_quality	51 91	57 72	55 03	62 91	57 79	44 15	49 59

Table 4 2 General classification results of the methods investigated, reported in terms of accuracy

4.6.2 Effect of the Number of Neurons

Another interesting question that we advocated is the effect of the *number* of neurons on the classification accuracy To test this, we trained the TTOSOM using the configuration presented in Section 4 5 2, and systematically increased the size of the tree In order to retain the desired property that, initially, all the neurons are considered as part of the BoA, in each case we adjusted the radius to be twice the depth of the tree Even though the size of the tree was increased, we decided to maintain the number of training iterations to be unchanged

Table 4 3 shows the results of these experiments Each column displays the results of the TTOSOM using the specific *number* of neurons, which increases from 15 neurons up to, for these particular datasets, an "unrealistic" number of 4,096 neurons Networks of this size are, in one sense, "unrealistic", because they contain *more* points than the original datasets themselves Thus, these represent scenarios in which

we artificially create data points to populate the space, and yet preserve the overall characteristics of the original datasets

If one analyzes the average behavior of the results, one identifies a significant increase in the overall performance as the number of neurons is increased For example, for the wine dataset, the accuracy was 64 61% when we used 15 nodes, and increased to 76 40% when the number of nodes was 1023 Similarly, for the glass dataset, we obtained an accuracy of 69 16% when we used 15 nodes, which increased to 71 96% when the number of nodes was 127

Observe that in the extreme case, when the number of neurons exceeds the number of training samples, we can obtain an improvement in the accuracy, when compared to smaller tree configurations Additionally, we note that a lesser number of neurons, which implies a lower computational requirement, outputs a fairly good approximation to the one offered by the reported supervised classifiers To be more specific, if we consider the results in more detail, we note that the accuracy obtained for the glass dataset using a high number of neurons (1,023 and 4,095, respectively), were the best accuracies obtained for the particular problem set (76 40% and 75 84% respectively) This result is quite promising and demonstrates the potential of the scheme for certain types of domains, given that the *supervised* classifiers chosen for this benchmark are among the most competitive supervised classifiers available in the literature

Dataset ↓ Neurons →	**15**	**127**	**1023**	**4095**
iris	94 67	95 33	96 00	96 00
wdbc	91 39	92 79	91 21	91 56
glass	69 16	71 96	74 30	73 36
wine	64 61	69 66	76 40	75 84
yeast	53 98	53 50	51 21	51 68
wine_quality	48 84	48 66	53 35	56 79

Table 4 3 The accuracy of the TTOSOM as the number of neurons increases

4.6.3 Changing the Distance Measure

In all the results presented so far, we had assumed that the data was normalized before being analyzed To be more specific, the classifiers which we used utilized the so-called Local Normalization [60], in which the range of every dimension was scaled to be between 0 and unity so as to have them equally weighted In this section we consider how the techniques can be invoked if we maintain all the parameters at their original values, and simultaneously not perform any type of normalization prior to the training process

Table 4 4 presents the accuracy of the TTOSOM using a binary tree to represent the structure, with the further constraint that the data is not normalized

As a general remark we note that one observes differences with respect to the case when the data was normalized For example, in the *glass* dataset it was possible to obtain an accuracy of 71 96% when using 127 neurons which is an index equivalent to the one provided by the best supervised classifier (BN) for this specific problem domain It is even more interesting to see that when the number of neurons was increased to 1,023, the accuracy obtained was 74 30%, which is the *best* reported accuracy obtained for the glass dataset, when one includes *all* the supervised classifiers displayed in Table 4 2

However, we have noticed as well, that in some problem sets, as in the case of the *wine* dataset, the classification accuracies shown in Table 4 4 are inferior to those given in Table 4 3, where an *a priori* normalization was invoked

Our explanation for this phenomenon is that, when we do not normalize the feature vectors before processing them, the classifier weights those features with larger ranges for its values, more, and in certain cases it happens that these features are exactly the ones that help to advantageously discriminate between the different categories This reasoning also explains the scenario when poorer results are obtained This is apparently a consequence of weighting certain features (i e , those which possess a high variance) more, i e , those which offer inadequate discriminating aspects Those features do not provide information that is too useful for effectively building the discrimination regions

Dataset ↓ Neurons →	15	127	1023	4095
glass	69 16	71 96	74 30	73 36
iris	94 67	95 33	96 00	96 00
wdbc	91 39	92 79	91 21	91 56
wine	64 61	69 66	76 40	75 84
winequality	48 84	48 66	53 35	56 79
yeast	53 98	53 50	51 21	51 68

Table 4 4 The accuracy of the TTOSOM using a binary tree when the data is not normalized *a priori*

4.6.4 Using Trees Other Than Binary Trees

All the experiments presented previously in this section have employed a binary tree structure To further investigate the power of the TTOSOM, we performed another set of experiments so as to test the effect of using trees with a higher branching factor, i e , the number of children that a particular node had Table 4 5 shows the accuracy rates obtained by training the TTOSOM using a 3-ary tree, where the data was previously normalized

Dataset ↓ Neurons →	13	121	1413	3876
glass	52 34	68 22	68 22	67 76
iris	96 00	94 67	94 00	94 00
wdbc	90 69	95 61	95 08	95 78
wine	96 63	97 19	95 51	95 51
winequality	52 22	55 97	59 72	62 79
yeast	51 68	54 04	49 93	50 61

Table 4 5 The accuracy of the TTOSOM using a full 3-ary tree in which the training data was normalized *a priori*

As far as we can see, there are no noticeable changes in accuracy when the branching factor per node is increased from 2 to 3

In Chapter 3, when we focused on the clustering properties on the TTOSOM, we showed how different branching factors led to a "better representation" of certain shapes By better representation, in this case, we meant that the basic properties

of some objects were preserved, so that the human eye could perceive the essential characteristics of the original object by merely looking at the learned structure Examples of these were the triangle illustrated in Figure 3 5d, the rectangle presented in Figure 3 6d, etc , which were represented in a superior manner using specific branching factors (c f , the representations in Figure 1 1a and Figure 3 8d, which correspond to neural structures for the triangle using a grid and a line, respectively)

The clustering property mentioned above suggests that the symmetry presented in some data sets could be better exploited by a TTOSOM-based classifier using the adequate branching factor However our preliminary evidence shows us that at least for the *real-world dataset* that we tested, the classifier is not noticeably affected by incrementing the number of branches in the tree Instead, the number of neurons utilized, regardless of the branching factor of the tree, seems to be more pertinent when it concerns the resultant accuracy This certainly is an avenue for further research

4.7 Conclusions

The purpose of this chapter was to design and present an experimental analysis of a novel PR scheme based on the TTOSOM This scheme constructed a classifier by combining the information provided by labeled and unlabeled samples of the classes simultaneously

Our experimental results showed that the TTOSOM classifier possesses an improved classification accuracy in comparison to other VQ-based classifiers Moreover, we have obtained accuracies that are comparable to the one attained by the state-of-the-art classifiers, even when the number of neurons utilized is only a small fraction of the cardinality of the dataset

It was also found that increasingly superior recognition capabilities could be obtained by representing the data using a TTOSOM that involved a tree with a larger number of neurons In particular, the results presented here indicate a "monotonic" improvement of the mean classifier performance as the size of the tree was increased We believe that this occurs because of the desirable properties of the TTOSOM to

mimic the underlying distribution of the points, and its capability to represent the stochastic and structural characteristics more accurately by utilizing a larger tree.

Chapter 5

Learning the SOMs Topology with Adaptive BSTs

5.1 Introduction

In this Chapter[1] we dedicate our attention to a completely new and unexplored way of adapting the SOM, which, to the best of our knowledge, has not been utilized by *any* of the strategies reported in the literature The principles behind this adaptation involves the *fusion* between the philosophies defined by the tree-structured families of SOMs and the field of ADSs Our novel scheme attempts to generate an asymptotically optimal tree with respect to the access probabilities of the neurons, i e , based on the number of times that they have been identified as the BMU

As shown in Chapter 2, there exist numerous variants of the SOM which also possess an underlying structure, and in some cases, this structure itself can be defined by the user, as in the case of the TTOSOM presented in Chapter 3 Although the concepts of growing the SOM and updating it have been studied, the whole issue of using a self-organizing *adaptive data structure* to further enhance the properties of the underlying SOM, have never been explored In Chapter 3, we proposed a strategy by

[1] A preliminary version of some of the results from this chapter appear in the *Proceedings of AI 2009, the 2009 Australasian Joint Conference on Artificial Intelligence,* Australia, in December 2009 [12] This paper won the *Best Paper Award of the Conference* The journal version of these results is currently under review

which we can impose an *arbitrary*, user-defined, tree-like topology onto the codebooks, which consequently enforced a neighborhood phenomenon and the so-called tree-based BoA In this chapter, we consider how the underlying tree itself can be rendered dynamic and adaptively transformed To do this, we present methods by which a SOM with an underlying BST structure can be adaptively re-structured using conditional rotations These rotations on the nodes of the tree are local, can be done in constant time, and performed so as to decrease the Weighted Path Length (WPL) of the entire tree We introduce a pioneering concept referred to as *Neural Promotion*, where neurons gain prominence in the NN as their significance increases We are not aware of any research which deals with the issue of Neural Promotion The advantages of such a scheme is that the user need not be aware of any of the topological peculiarities of the specific input data set or of the stochastic data distribution Rather, the algorithm, referred to as TTOCONROT, converges in such a manner that the neurons are ultimately placed in the input space so as to represent its stochastic distribution, and additionally, the neighborhood properties of the neurons suit the best BST that represents the data These properties have been confirmed by our experimental results on a variety of data sets

This Chapter is a pioneering attempt to merge the areas of the Kohonen's NNs with the theory of ADS Put in a nutshell, we can describe the goal of this chapter as follows Consider a SOM with n neurons Rather than having the neurons merely possess information about the feature space, we also attempt to *link* them together by means of an underlying data structure This data structure could be a singly-linked list, a doubly-linked list or a BST, etc The intention is that the neurons are governed by the laws of the SOM *and* the underlying data structure Observe now that the concepts of "neighborhood" and BoA are not based on the nearness of the neurons in the feature space, but rather on their proximity in the underlying data structure Having accepted this premise, we intend to take this entire concept to a higher level of abstraction and propose to modify this data structure *itself* adaptively using operations specific to it As far as we know, the combination of these concepts has been unreported in the literature

Although initial studies of using singly-linked lists and doubly-linked lists to constrain the codebooks are underway, our aim in this chapter is to constrain the neurons so that they are related to each other based on a BST relationship We endeavor to explore this novel integration, with the overall goal of obtaining an enhancement of the capabilities of the original SOM algorithm so as to represent the underlying data distribution and its structure in a more accurate manner Furthermore, as a long term ambition, we also anticipate methods which can accelerate one of the most time-consuming phases in the SOM philosophy, i e , that which corresponds to the task of locating the nearest neuron during the CL phase Our intention is to investigate how an adaptive process applied to the BST, can be integrated into the SOM mechanism so as to render the latter possible This chapter will present the details of the design and implementation of such a methodology

Although a lot of attention has been dedicated to deriving variants of the original SOM strategy, few of them possess the ability of modifying the underlying topology (see Section 2 9 2) Moreover, as detailed in Section 2 8, only a small subset of these use a *tree* as their underlying data structure From these reported works, it is possible to distinguish, as a common denominator, an attempt to dynamically modify the nodes of the SOM, for example, by adding and/or deleting nodes However, our hypothesis is that it is also possible to attain to a better understanding of the unknown data distribution by performing *structural* tree-based modifications on the tree, which although they preserve the general topology, attempt to modify the overall configuration, i e , by altering the way by which nodes are interconnected, and yet continue as a BST We accomplish this by dynamically adapting the edges that connect the neurons, by rotating[2] the nodes within the BST that holds the whole structure of neurons In other words, we place emphasis on the self-organization achieved by means of *restructuring* the tree (and not merely on the location of the points in the feature space), where this restructuring is performed over the tree-based connections between the neurons As we will explain later, this is further achieved by local modifications to the overall structure in a constant number of steps Thus, we attempt to use rotations, tree-based neighbors *and* the feature space to improve the

[2]The operation of rotation is the one associated to BSTs, as will be presently explained

quality of the SOM.

One of the primary goals of the area of ADS is to achieve an optimal arrangement of the elements, placed at the nodes of the structure, as the number of iterations increases. This reorganization can be perceived to be both automatic and adaptive, such that on convergence, the data structure tends towards an optimal configuration with a minimum average access time. In most cases, the most probable element will be positioned at the root of the tree, while the rest of the tree is recursively positioned in the same manner. The solution to obtain the *optimal* BST is well known when the access probabilities of the nodes are known beforehand [100]. However, our research concentrates on the case when these access probabilities are *not known a priori*. In this setting, the most effective solution is due to Cheetham *et al.* and uses the concept of conditional rotations [38]. The latter paper introduced the new philosophy of reorganizing the BST so as to asymptotically produce the optimal form. It also specified how an accessed element can be rotated towards the root of the tree so as to minimize its overall cost. Additionally, unlike most of the algorithms that are otherwise reported in the literature, this move is not done on every data access operation. It is performed if and only if the overall WPL of the resulting BST decreases. It should be mentioned that the best reported adaptive BST methods use the "conditional rotation" concept due to Cheetham *et al.* in one form or the other.

We can conceptually distinguish the TTOCONROT from two perspectives, namely from that of the components it possesses and from the characteristics that it displays. In terms of components, we detect five elements. First of all, the TTOCONROT has a set of neurons, which, like all SOM-based methods, represents the data space in a condensed manner. Secondly, the TTOCONROT possesses a connection between the neurons, where the neighbor of any specific neuron need not necessarily be another that lies in its proximity in the *feature* space. Rather, the neighbors are based on a learned nearness measure that is tree-based. The third and fourth components involve the migration of the neurons. Similar to the reported families of SOMs, a subset of neurons closest to the winning neuron are moved towards the sample point using a VQ rule. However, unlike the reported families of SOMs, the identity of the neurons that are moved is based on the tree-based proximity and not on the

feature-space proximity Finally, the TTOCONROT possesses tree-based mutating operations, namely the above-mentioned conditional rotations

With respect to the characteristics of the TTOCONROT, we mention the following First of all, it is adaptive, with regard to the migration of the points Secondly, it is also adaptive with regard to the identity of the neurons moved Thirdly, the distribution of the neurons in the feature space mimics the distribution of the sample points Finally, by virtue of the conditional rotations, it turns out that the entire tree of neurons is optimized with regard to the overall accesses, which is a unique phenomenon (when compared to the reported family of SOMs) as far as we know

5.1.1 Contributions of the Chapter

The present work develops a tree-based SOM strategy, referred to as TTOCONROT, which combines the philosophies of two areas, the first one being in data representation (i e , the SOM), and the second being in mainstream computer science, i e , adaptive data structures The above-mentioned components and characteristics distinguish it from the SOM-based state-of-the-art schemes Thus, the contributions of the chapter can be summarized as follows

1 We present an integration of the fields of SOMs and ADS

2 The neurons of the SOM are linked together using an underlying tree-based data structure, and they are governed by the laws of the TTOSOM tree-based paradigm, and simultaneously the restructuring adaptation provided by conditional rotations

3 The distance between the neurons, as applicable for the task of determining the neighborhood phenomena, are based on the distances associated with the underlying tree structure and not their distance in the feature space This is valid also for the BoA, rendering the migrations distinct from the state-of-the-art

4 The adaptive nature of TTOCONROT is unique because adaptation is perceived in two forms The migration of the codebook vectors in the feature space

is a consequence of the SOM update rule, and the rearrangement of the neurons within the tree as a result of the rotations

5.1.2 Organization of the Chapter

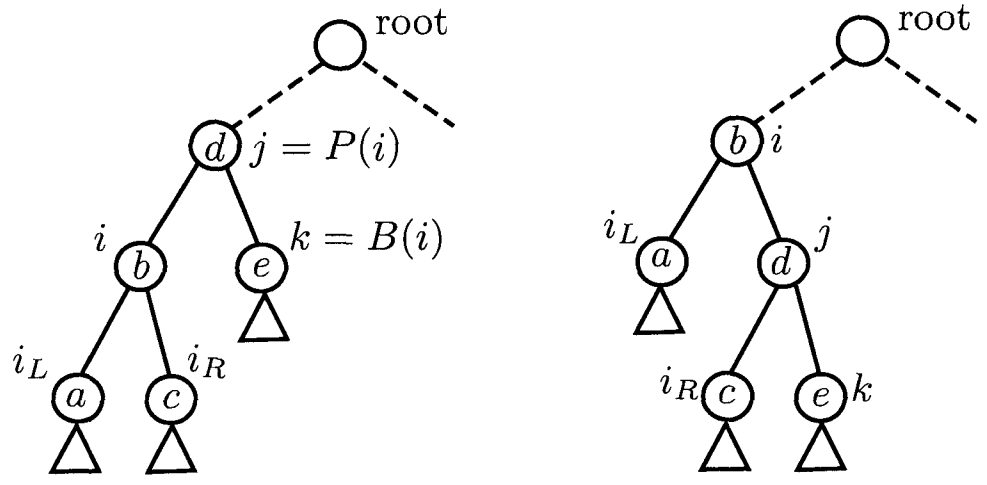
The rest of the chapter is organized as follows The next section surveys the field of BSTs with special attention being given to rotations and conditional rotations After that, in Section 5 3, we provide an in-depth explanation of the TTOCONROT solution The subsequent section demonstrates the capabilities of the approach through a series of experiments Finally, Section 5 5 concludes the chapter

5.2 ADS for BSTs

A BST may be used to store records whose keys are members of an ordered sequence $\mathcal{A} = \langle A_1, A_2, \quad , A_m \rangle$ The records are stored in such a way that a symmetric-order traversal of the tree will yield the records in an ascending order If we are given $\mathcal{A}$ and the sequence of access probabilities $\mathcal{S} = \langle S_1, S_2, \quad , S_m \rangle$, the problem of constructing efficient BSTs has been extensively studied The optimal algorithm due to Knuth [100], uses dynamic programming and produces the optimal BST using $O(m^2)$ time and space In this chapter, we study the problem in which $\mathcal{S}$, the access probability vector, is not known *a priori* We seek a scheme which dynamically rearranges itself and asymptotically generates a tree which minimizes the access cost of the keys

The primitive tree restructuring operation used in most BST schemes is the well known operation of Rotation [1] We describe this operation as follows Suppose that there exists a node i in a BST, and that it has a parent node j, a left child, i_L, and a right child, i_R Consider the case that i is itself a left child (see Figure 5 1a) A rotation is performed on node i as follows j now becomes the right child, i_R becomes the left child of node j, and all the other nodes remain in their same relative positions (see Figure 5 1b) The case when node i is a right child is treated in a symmetric manner This operation has the effect of raising (or promoting) a specified node in the tree structure while preserving the lexicographic order of the elements (refer again

to Figure 5 1b)



(a) The tree before a rotation is performed The contents of the nodes are their data values, which in this case are the characters $\{a, b, c, d, e\}$

(b) The tree after a rotation is performed on node i

Figure 5 1 The BST before and after a Rotation is performed

A few memory-less tree reorganizing schemes[3] which use this operation have been presented in the literature among which are the Move-to-Root and the simple Exchange rules [6] In the Move-to-Root Heuristic, each time a record is accessed, rotations are performed on it in an upwards direction until it becomes the root of the tree On the other hand, the simple Exchange rule rotates the accessed element one level towards the root

Sleator and Tarjan [166] introduced a technique, which also moves the accessed record up to the root of the tree using a restructuring operation called "Splaying", which actually is a multi-level generalization of the rotation Their structure, called the Splay Tree, has an amortized time complexity of $O(\log m)$ for a complete set of tree operations which included insertion, deletion, access, split, and join

The literature also records various schemes which adaptively restructure the tree with the aid of additional memory locations Prominent among them is the Monotonic

[3]This review is necessary brief A more detailed version is found in [43, 113]

Tree (MT) [24] and Mehlhorn's D-Tree (DT) [131] The MT is a dynamic version of a tree structuring method originally suggested by Knuth [100]

In spite of all their advantages, all of the schemes mentioned above have drawbacks, some of which are more serious than others The memory-less schemes have one major disadvantage, which is that both the Move-to-Root and Splaying rules always move the accessed record up to the root of the tree This means that if a nearly-optimal arrangement is reached, a single access of a seldomly-used record will dis-arrange the tree along the entire access path, as the element is moved upwards to the root

As opposed to these schemes, the MT rule does not move the accessed element to the root every time However, as reported in [38], in practice, it does not perform well The weakness of the MT lies in the fact that it considers only the frequency counts for the records, which leads to the undesirable property that a single rotation may move a subtree with a relatively large probability weight downwards, thus increasing the cost of the tree

This chapter focuses on a particular heuristic, namely, the CONROT-BST [38], which has been shown to reorganize a BST so as to asymptotically arrive at an optimal form In its optimized version, the scheme, Algorithm CONROT-BST (see Algorithm 19), requires the maintenance of a single memory location per record, which keeps track of the number of accesses to the subtree rooted at that record The CONROT-BST algorithm specifies how an accessed element can be rotated towards the root of the tree so as to minimize the overall cost of the entire tree Finally, unlike most of the algorithms that are currently in the literature, this move is not done on every data access operation It is performed if and only if the overall WPL of the resulting BST decreases

In essence Algorithm 19 attempts to minimize the WPL by incorporating the statistical information about the accesses to the various nodes *and subtrees* rooted at the corresponding nodes

The basic condition for the rotation of a node is that the WPL of the entire tree must decrease as a result of a single rotation This is achieved by a so-called *Conditional* Rotation To define the concept of a Conditional Rotation, we define

$\tau_i(n)$ as the total number of accesses to the subtree rooted at node i One of the biggest advantages of the CONROT-BST heuristic is that it only requires the maintenance and processing of the values stored at a specific node and its direct neighbors, i e , its parent and both children, if they exist

Algorithm 19, formally given below, describes the process of the Conditional Rotations for a BST The algorithm receives two parameters, the first of which corresponds to a pointer to the root of the tree, and the second which corresponds to the key to be searched, which is assumed to be present in the tree When a node access is requested, the algorithm seeks for the node from the root down towards the leaves

The first task accomplished by the Algorithm 19 is the updating of the counter τ for the present node along the path traversed After that, the next step consists of determining whether or not the node with the requested key has been found When this occurs, the following equations are computed to determine the value of a quantity referred to as Ψ, where

$$\Psi_j = 2\tau_j - \tau_{jR} - \tau_{P(j)} \tag{5 1}$$

when j is a left child of $P(j)$, and

$$\Psi_j = 2\tau_j - \tau_{jL} - \tau_{P(j)} \tag{5 2}$$

in case that j is a right child of $P(j)$

When Ψ is greater than zero, an upward rotation is performed The authors of [38] have shown that this single rotation yields to a decrease in the WPL of the *entire* tree This occur in line No 9 of the algorithm, in which the method `rotate-upwards` is invoked The parameter to this method is a pointer to the node j The method does the necessary operations required to rotate the node upwards, which means that if the node j is the left child of the parent, then this is equivalent to performing a right rotation over $P(j)$, the parent of j Analogously, when j is the right child of its parent, the parent of j is left-rotated instead Once the rotation takes place, it is necessary to update the corresponding counters, τ Fortunately this task only involve the updating of τ_i, for the rotated node, and the counter of its parent, $\tau_{P(i)}$ The last

Algorithm 19 CONROT-BST(j,k_i)

Input:

i) j, A pointer to the root of a BST T

ii) k_i, A search key, assumed to be in T

Output:

i) The restructured tree T'

ii) A pointer to the record i containing k_i

Method:

```
1  τ_j ← τ_j + 1
2  if k_i = k_j then
3    if left-child(j) then
4      Update Ψ_j according to Equation 5.1
5    else
6      Update Ψ_j according to Equation 5.2
7    end if
8    if Ψ_j > 0 then
9      rotate-upwards(j)
10     recalculate-tau(j)
11     recalculate-tau(P(j))
12   end if
13   return record j
14 else
15   if k_i < k_j then
16     CONROT-BST( left-child(j) , k_i )
17   else
18     CONROT-BST( right-child(j) , k_i )
19   end if
20 end if
```

End Algorithm

part of the algorithm, namely lines Nos 14-19, deal with the further search for the key, which in this case is achieved recursively

The reader will observe that all the tasks invoked in the algorithm are performed in constant time, and in the worst case, the recursive call is done from the root down to the leaves, leading to a $O(h)$ running complexity, where h is the height of the tree

5.3 Merging CONROT-BST and the TTOSOM

This section describes the details of the integration between the fields of ADS and the SOM, and in particular, the TTOSOM Although merging ADSs and the SOM is relevant to a wide spectrum of data structures, we focus the scope of our discussions by considering only tree-based structures More specifically, we shall concentrate on the integration of the CONROT-BST heuristic (explained in the previous section) and the TTOSOM (detailed in Chapter 3)

The general dynamic adaptation of SOM lattices was earlier studied in the literature by essentially adding and/or deleting edges (as presented in Sections 2 7 and 2 8) However the concept of modifying the underlying structure's *shape* itself has been unrecorded Our hypothesis is that this is advantageous by means of a repositioning of the *nodes* and the consequent *edges*, as seen when one performs rotations on a BST In other words, we place our emphasis on the self-arrangement which occurs as a result of restructuring the data structure representing the SOM In this case, as alluded to earlier, the restructuring process is done between the connections of the neurons so as to attain an asymptotically optimal configuration, where nodes that are accessed more frequently will tend to be placed close to the root We thus obtain a new species of tree-based SOMs which is self-arranged by performing rotations **conditionally**, **locally** and in a **constant number of steps**

The primary goal of the field of ADS is to have the structure and its elements attain an optimal configuration as the number of iterations increases Particularly, among the ADSs that use trees as the underlying topology, the common goal is to minimize the overall access cost, and this roughly means that one places the most frequently accessed nodes close to the root, which is also what CONROT-BST moves

towards Although such an adaptation can be made on any SOM paradigm, the CONROT-BST is relevant to a tree structure, and thus to the TTOSOM This further implies that some specific settings/modifications must be applied to achieve the integration between the two paradigms

We start by defining a Binary Search Tree SOM (BSTSOM) as a special instantiation of a SOM which uses a BST as the underlying topology An Adaptive BSTSOM (ABSTSOM) is a further refinement of the BSTSOM which, during the training process, employs a technique that automatically modifies the configuration of the tree The goal of this adaptation is to facilitate and enhance the search process This assertion must be viewed from the perspective that for a SOM, neurons that represent areas with a higher density, will be queried more often

Every ABSTSOM is characterized by the following properties First, it is **adaptive**, where, by virtue of the BST representation this adaptation is done by means of rotations, rather than by merely deleting or adding nodes Second, the connections of the NN corresponds to a BST The goal is that the NN simultaneously maintains the essential stochastic and topological properties of Kohonen's Self-Organizing Maps (explained in Chapter 2)

5.3.1 Neural Distance

As mentioned in Chapter 3, in the case of the TTOSOM, the *Neural Distance*, d_N, between two neurons depends on the *number* of unweighted connections that separate them in the user-defined tree It is consequently the number of edges in the shortest path that connects the two given nodes More explicitly, the distance between two nodes in the tree, is defined as the minimum number of edges required to go from one to the other In the case of trees, the fact that there is only a *single* path connecting two nodes implies the uniqueness of the shortest path, and permits the efficient calculation of the distance between them by a node traversal algorithm Note however, that in the case of the TTOSOM, since the tree itself was *static*, the inter-node distances can be pre-computed *a priori*, simplifying the computational process The situation changes when the tree is dynamically modified as we shall explain

below

The implications of having the tree which describes the SOM to be dynamic, are three-fold First of all, the siblings of any given node may change at every time instant Secondly, the parents and ancestors of the node under consideration could also change at every instant But most importantly, the structure of the tree itself could change, implying that nodes that were neighbors at any time instant may not continue to be neighbors at the next Indeed, in the extreme case, if a node was migrated to become the root, the fact that it had a parent at a previous time instant is irrelevant at the next This, of course, changes the entire landscape, rendering the resultant SOM to be unique and distinct from the state-of-the-art An example will clarify this

Consider Figure 5 2, which illustrates the computation of the neural distance for various scenarios First, in Figure 5 2a, we present the scenario when the node accessed is B Observe that the distances are depicted with dotted arrows, with an adjacent numeric index specifying the current distance from node B In the example, prior to an access, nodes H, C and E are all at a distance of 2 from node B, even though they are at different levels in the tree The reader should be aware that non-leaf nodes may also be involved in the calculation, as in the case of node H Figures 5 2b and 5 2c show the process when node B is queried, which in turn triggers a rotation of node B upwards Observe that the rotation itself only requires local modifications, leaving the rest of the tree untouched For the sake of simplicity and explicitness, unmodified areas of the tree are represented by dashed lines Finally, Figure 5 2d depicts the configuration of the tree after the rotation is performed At this time instant, C and E are both at distance of 3 from B, which means that they have increased their distance to B by unity Moreover, although node H has changed its position, its distance to B remains unmodified Clearly, the original distances are not necessarily preserved as a consequence of the rotation

Generally speaking, there are four regions of the tree that remain unchanged These are, namely, the portion of the tree above the parent of the node being rotated, the portion of tree rooted at the right child of the node being rotated, the portion of tree rooted at the left child of the node being rotated, and the portion of tree rooted

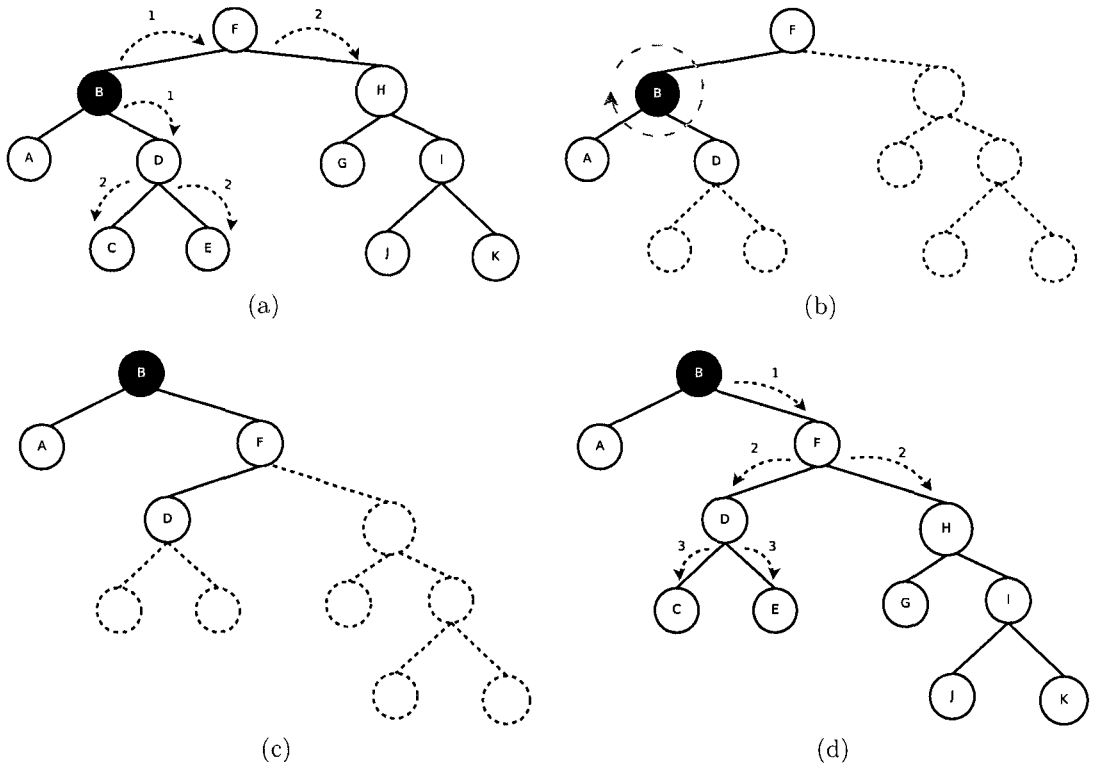


Figure 5.2: Example of the neural distance before and after a rotation. In Figure 5.2a nodes H, C and E are equidistant from B even though they are at different levels in the tree. Figures 5.2b and 5.2c show the process of rotating node A upwards. Finally, Figure 5.2d depicts the state of the tree after the rotation when only C and E are equidistant from B, and their distance to B has increased by unity. On the other hand, although H has changed its position, its distance to B remains the same.

at the sibling of the node being rotated. Even though these four regions remain unmodified, the neural distance in these regions are affected, because the rotation could lead to a modification of the distances to the nodes.

Another consequence of this operation that is worth mentioning is the following: The distance between any two given nodes that belong to the same unmodified region of the tree is preserved after a rotation is performed. The proof of this assertion is obvious, inasmuch as the fact remain that every path between nodes in any unmodified sub-tree remains with the same sub-tree. This property is interesting because it has

the potential to accelerate the computation of the respective neural distances

The Bubble of Activity

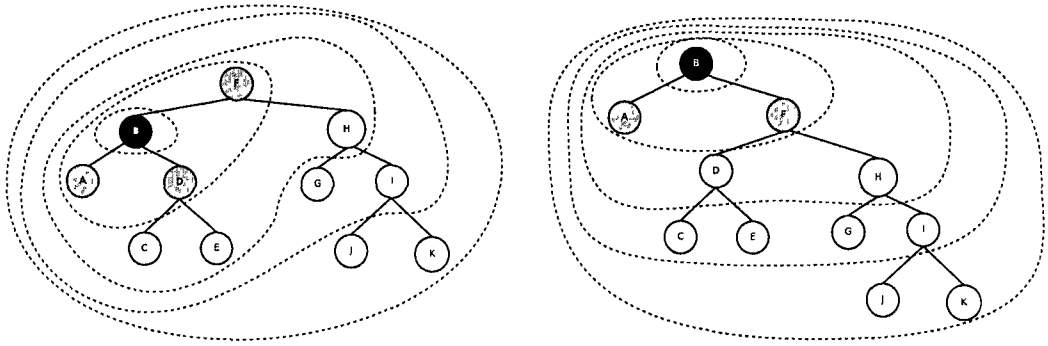
As defined in Section 3 2 4, the BoA corresponds to the subset of nodes within a distance of r away from the node that is currently being examined Those nodes are in essence those which are to be migrated toward the signal presented to the network This concept is valid for all SOM-like NNs, and in particular for the TTOSOM We shall now consider how this bubble is modified in the context of rotations

To clarify how the bubble is modified in the context of rotations, we begin by describing the context when the tree is static As presented in Chapter 3, Algorithm 14 specifies the steps involved in the calculation of the subset of neurons that are part of the neighborhood of the BMU This computation involves a collection of parameters, including ϕ, the current subset of neurons in the proximity of the neuron being examined, the BMU itself, and $r \in \mathbb{N}$ the current radius of the neighborhood When the function is invoked for the first time, the set ϕ contains only the BMU marked as the current node, and through a recursive call, ϕ will end up storing the entire set of units within a radius r of the BMU The tree is recursively traversed for all the direct topological neighbors of the current node, i e , in the direction of the direct parent and children Every time a new neuron is identified as part of the neighborhood, it is added to B and a recursive call is made with the radius decremented by one unit[4], marking the recently added neuron as the current node

The question of whether or not a neuron should be part of the current bubble, depends on the number of connections that separate the nodes rather than the distance that separate the networks in the solution space (for instance, the Euclidean distance) Figure 5 3 depicts how the BoA differs from the one defined by the TTOSOM as a result of applying a rotation Figure 5 3a shows the BoA around the node B, using the same configuration of the tree as in Figure 5 2a, i e , before the rotation takes place Here, the BoA when $r = 1$ involves the nodes $\{B, A, D, F\}$, and when $r = 2$ the nodes contained in the bubble are $\{B, A, D, F, C, E, H\}$ Subsequently, considering a radius equal to 3, the resulting BoA contains the nodes $\{B, A, D, F, C, E, H, G, I\}$ Finally,

[4]This fact will ensure that the algorithm reaches the base case when $r = 0$

the $r = 4$ case leads to a BoA which includes the whole set of nodes. Now, observe the case presented in Figure 5.3b, which corresponds to the BoA around B *after* the rotation upwards has been effected, i.e., the same configuration of the tree used in Figure 5.2d. In this case, when the radius is unity, nodes $\{B, A, F\}$ are the *only* nodes within the bubble, which is different from the corresponding bubble before the rotation is invoked. Similarly, when $r = 2$, we obtain a set different from the analogous pre-rotation case, which in this case is $\{B, A, F, D, H\}$. Note that coincidentally, for the case of a radius equal to 3, the bubbles are identical before and after the rotation, i.e., they invoke the nodes $\{B, A, D, F, G, I\}$. Trivially, again, when $r = 4$, the BoA invokes the entire tree.



(a) (b)

Figure 5.3: The BoA associated with the TTOSOM before and after a rotation is invoked at node B.

As explained in Chapter 2, Equation (2.12) describes the criterion for a BoA calculated on a *static* tree. It happens that, as a result of the conditional rotations, the tree will be dynamically adapted, and so the entire phenomenon has to be revisited. Consequently, the BoA around a particular node becomes a function of time, and, to reflect this fact, Equation (2.12) should be reformulated as:

$$\phi(\mathbf{c}_i; T, r, t) = \{\mathbf{c} | d_N(\mathbf{c}_i, \mathbf{c}; T, t) \leq r\}, \tag{5.3}$$

where t is the discrete time index.

The algorithm to obtain the BoA for a specific node in such a setting is identical

to Algorithm 14, except that the input tree itself dynamically changes Further, even though the formal notation includes the time parameter, "t", it happens that, in practice, the latter is needed only if the user/application requires a history of the BoAs for any or all the nodes Storing the history of the BoAs will require the maintenance of a data structure that will primarily store the changes made to the tree itself Although storing the history of changes made to the tree can be done optimally by utilizing a persistent data structure [95], the question of explicitly storing the entire history of the BoAs for all the nodes in the tree remains open

5.3.2 Enforcing the BST Property

The CONROT-BST heuristic [38] requires that the tree should possess the BST property [43]

Let x be a node in a BST If y is a node in the left subtree of x, then $key[y] \leq key[x]$ Further, if y is a node in the right subtree of x, then $key[x] \leq key[y]$

To satisfy the BST property, first of all we see that, the tree must be binary[5] As a general TTOSOM utilizes an arbitrary number of children per node, one possibility is to bound the value of the branching factor to be 2 In other words, the tree trained by the TTOSOM is restricted to contain at most two children per node Additionally, the tree must implicitly involve a comparison operator between the two children so as to discern between the branches and thus perform the search process This comparison can be achieved by defining a unique key that must be maintained for each node in the tree, and which will, in turn, allow a lexicographical arrangement of the nodes

This leads to a different, but closely related, concept, which concerns the preservation of the topology of the SOM During the training process, the configuration of the tree will change as the tree evolves, positioning nodes that are accessed more often closer to the root This probability-based ordering, will hopefully, be preserved by the rotations

A particularly interesting case occurs when the imposed tree corresponds to a *list*

[5]Of course, this is a severe constraint But we are forced to require this, because the phenomenon of achieving conditional rotations for arbitrary k-ary trees is unsolved This research, however, is currently being undertaken

of neurons, i e a 1-ary tree If the TTOSOM is trained using such a tree where each node has at most two children, then the adaptive process will alter the original list The rotations will then modify the original configuration, generating a new state, where the non-leaf nodes might have one or two child each In this case the consequence of incorporating ADS-based enhancements to the TTOSOM will imply that the results obtained are significantly different from those shown in Chapter 3

As shown in [100], an optimal arrangement of the nodes of the tree can be obtained using the probabilities of accesses of the neurons If these probabilities are not known *a priori*, then the CONROT-BST heuristic offers a solution, which involves a decision of whether or not to perform a single rotation towards the root It happens that the concept of the "just accessed" node in the CONROT-BST is compatible with the corresponding BMU defined for the CL model In CL, a neuron may be accessed more often than others and some techniques take advantage of this phenomenon through the inclusion of strategies that add or delete nodes (e g , the GCS, the GG and the ET, discussed in detail in Chapter 2)

The CONROT-BST implicitly stores the information acquired by the currently accessed node by incrementing a counter for that node This is (in a philosophical sense) akin to the concept of a BMU counter defined in competitive networks (see Section 2 6 1), which is a counter that records the number of times that a specific neuron has been identified as the BMU

During the training phase, when a neuron is a frequent winner of the CL, it gains prominence in the sense that it can represent more points from the original data set This phenomenon is registered by increasing the BMU counter for that neuron We propose that during the training phase, we can verify if it is worth modifying the configuration of the tree by moving this neuron one level up towards the root as per the CONROT-BST algorithm, and consequently explicitly recording the relevant role of the particular node with respect to its nearby neurons CONROT-BST achieves this by performing a *local* movement of the node, where only its direct parent and children are aware of the neuron promotion

Neural Promotion is the process by which a neuron is relocated in a more

privileged position[6] in the network with respect to the other neurons in the NN Thus, while all "all neurons are born equal", their importance in the society of neurons is determined by what they represent This is achieved, by an explicit advancement in its rank or position Given this premise, the nodes in the tree will be adapted in such a way that neurons that have been BMUs more frequently, will tend to move towards the root if an only if a reduction in the overall WPL is obtained as a consequence of such a rotation The properties of CONROT-BST guarantee this

Once the SOM and BST are "tied" together in a symbiotic manner (where one enhances the other and vice versa), the adaptation can be achieved by affecting the configuration of the BST This task will be performed every time a training step of the SOM is performed Clearly, it is our task to achieve an integration of the BST and the SOM, and Figure 5 4 depicts the main architecture used to achieve this It transforms the structure of the SOM by modifying the configuration of the BST that holds the structure of the neurons

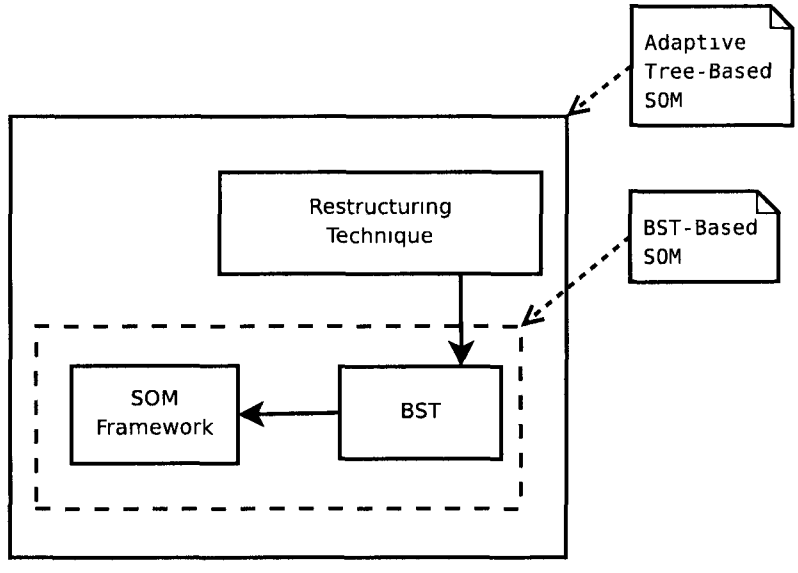


Figure 5 4 Architectural view of an Adaptive Tree-Based SOM

[6] As far as we know, we are not aware of any research which deals with the issue of Neural Promotion Thus, we believe that this concept, itself, is pioneering

5.3.3 Initialization

Initialization, in the case of the BST-based TTOSOM, is accomplished in two main steps which involve defining the initial value of each neuron and the connections among them The initialization of the codebook vectors are performed in the same manner as in the basic TTOSOM The neurons can assume a starting value arbitrarily, for instance, by placing them on randomly selected input samples On the other hand, a major enhancement with respect to the basic TTOSOM lays in the way the neurons are linked together The basic definition of the TTOSOM utilizes connections that remain static through time The beauty of such an arrangement is that it is capable of reflecting the user's perspective at the time of describing the topology, and it is able to preserve this configuration until the algorithm reaches convergence The inclusion of the rotations renders this dynamically

5.3.4 The Required Local Information

In our proposed approach, the codebooks of the SOM correspond to the nodes of a BST Apart from the information regarding the codebooks themselves, each neuron requires the maintenance of additional fields to achieve the adaptation Also, besides the codebook vectors themselves, each node inherits the properties of a BST node, and it thus includes a pointer to the left and right children, as well as (to make the implementation easier), a pointer to its parent Each node also contains a label which is able to uniquely identify the neuron when it is the "company" of other neurons This identification index constitutes the lexicographical key used to sort the nodes of the tree and remains static as time proceeds Figure 5 5 depicts all the fields included in a neuron of a BST-based SOM

5.3.5 The Neural State

The different states that a neuron may assume during its lifetime is illustrated in Figure 5 6 At first, when the node is created, a unique identification is assigned to it and the rest of the data fields are filled with their initial values Here, the codebook vector assumes a starting value and also the pointers are configured so as to

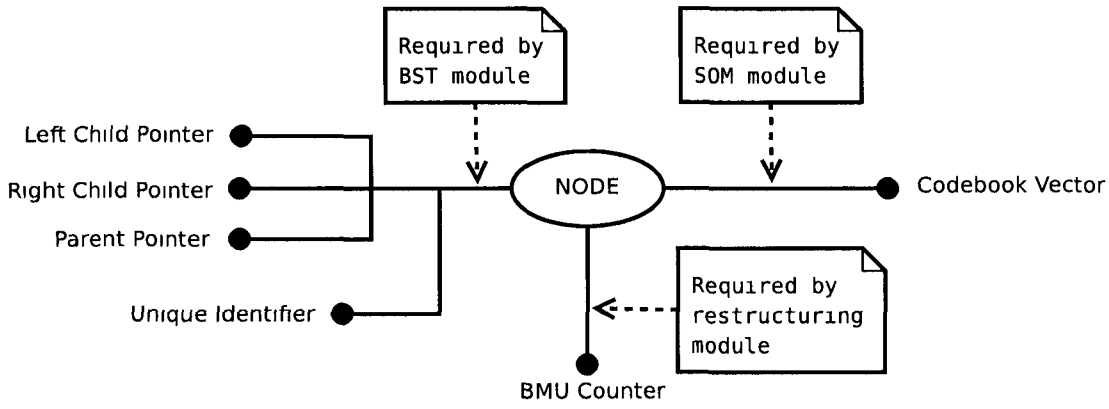


Figure 5 5 Fields included in a BST-based SOM Neuron

appropriately link the neuron with the rest of the neurons in the tree Next, during the most significant portion of the algorithm, the NN falls in a main loop, where training is effected This training phase, involves the adjusting of the codebooks and also may trigger optional modules that affect the neuron Once the BMU is identified, the neuron might assume the "restructured" state, which means that a restructuring technique, such as the CONROT algorithm, will be applied alternatively, the neuron might be ready to accept queries, i e , be part of the CL process in the mapping mode Additionally, we are currently investigating the case when a neuron is no longer necessary and thus, it may be eliminated from the main structure of neurons, this, the so called "deleted" state, is marked in dashed lines Lastly, an alternative "frozen" state, where the neuron does not longer form part of the CL during the training mode (but it does compete during the mapping mode), also needs further revision

5.3.6 The Training Step of the TTOCONROT

The training module of the TTOCONROT is responsible of determining the BMU, performing restructuring, calculating the BoA and migrating the neurons within the BoA Basically, what it has to be done, is to integrate the CONROT algorithm in the sequence of steps of the training step defined by the TTOSOM Algorithm 20

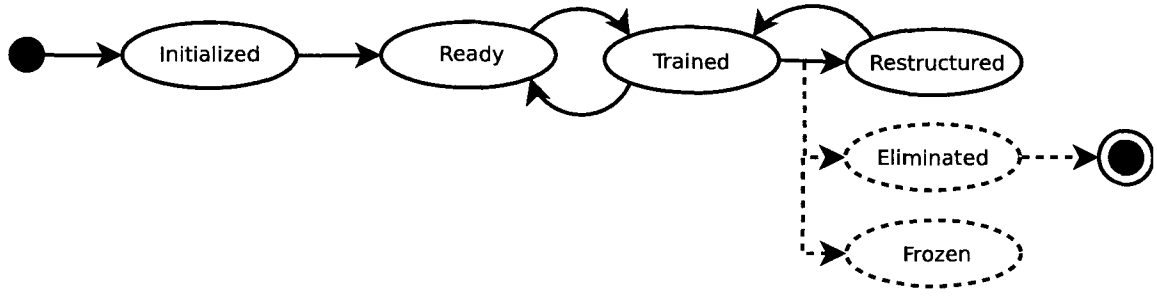


Figure 5.6: Possible states that a neuron may assume.

describes the details of how this integration is fulfilled. Line No. 1, performs the first task of the algorithm, which is the establishment of the BMU. After that, line No. 2, deals with the call to the CONROT algorithm. The reason to follow this sequence of steps is that the parameters needed to perform the conditional rotation, as specified in [38], includes the key of the element queried, which, in the present context, corresponds to the key of the BMU. At this stage of the algorithm, the BMU may be rotated or not, and the BoA is determined *after* this restructuring process, which is performed in lines No. 3 and 4 of the algorithm. Finally, lines No. 5 to 7, are in charge of the neural migration, involving the movement of the neurons within the BoA towards the input sample.

Algorithm 20 `TTO-CONROT-BST_train`(x,p)

Input:

i) x, a sample signal.

ii) p, the pointer to the tree.

Method:

1: $v \leftarrow$ `TTOSOM_Find_BMU`(p,x,p)
2: `cond-rot-bst`(p,v.`getID`())
3: $B \leftarrow \{v\}$
4: `TTOSOM_Calculate_Neighborhood`(B,v,*radius*)
5: **for all** $b \in B$ **do**
6: `update_rule`(b.`getCodebook`(),x)
7: **end for**

End Algorithm

5.3.7 Alternative Restructuring Techniques

Even though we have explained the advantages of the CONROT algorithm, the architecture that we are proposing allows the election of an alternative restructuring module Candidates to perform the adaptation are the ones mentioned in Section 5 2 and include the splay and the monotonic-tree algorithms, among others

Figure 5 7 offers the structural design to connect the functionality of alternative restructuring techniques into a BST-based SOM The diagram utilizes the so-called Strategy Pattern [75] and allows the exchange of restructuring techniques even at runtime

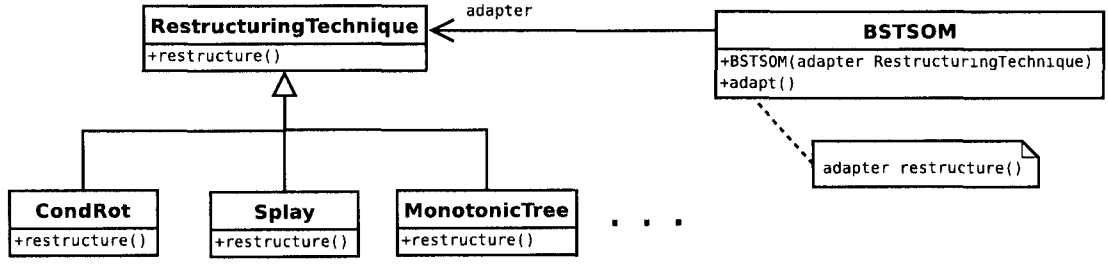


Figure 5 7 Different restructuring techniques can be plugged into a BST-based SOM

5.4 Experimental Results

To illustrate the capabilities of our method, the experiments reported in the present chapter are limited to lower dimensional feature spaces (just as in Chapter 3) However, it is important to again remark that the algorithm is also capable of solving problems in higher dimensions, although a graphical representation of the results will not be as illustrative We know that, as per the results obtained in Chapter 3, the TTOSOM is capable of inferring the distribution and structure of the data However, in this present setting, we are interested in investigating the effects of applying the neural rotation as part of the training process To render the results comparable, the experiments in this chapter use the same schedule for the learning rate and radius, i e , no particular refinement of the parameters has been done for any specific data

set Additionally, the parameters follow a rather "slow" decrement of the so-called decay parameters, allowing us to understand how the prototype vectors are moved as convergence takes place When solving practical problems, we recommend a further refinement of the parameters so as to increase the speed of the convergence process

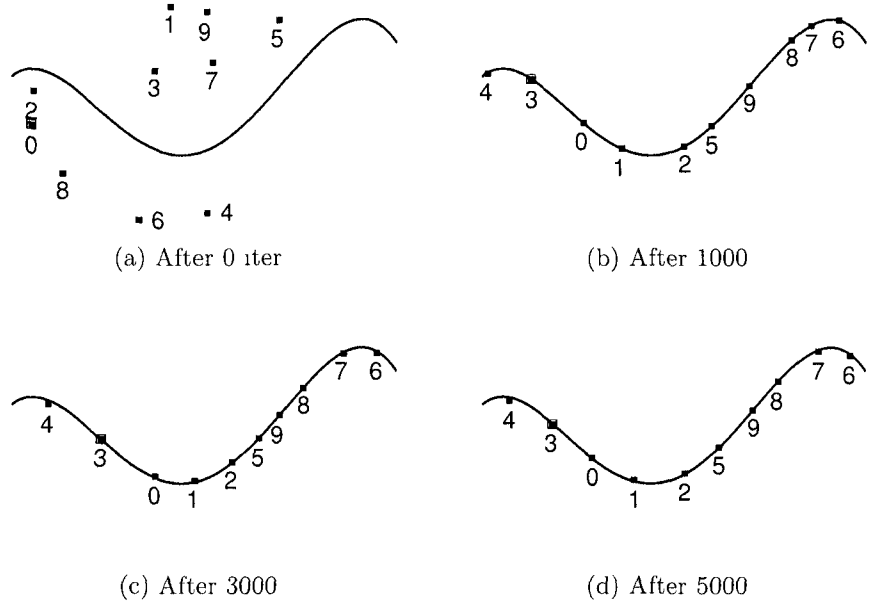
5.4.1 TTOCONROT's Structure Learning Capabilities

Contiguous Data Points

We shall describe the performance of TTOCONROT with data sets in 1, 2 and 3 dimensions The specific advantages of the algorithm for various scenarios will also be highlighted

One dimensional objects: Since our entire learning paradigm assumes that the data has a tree-shaped model, our first attempt was to see how the philosophy is relevant to a unidimensional object (i e , a curve), which really possesses a "linear" topology Thus as a *prima facie* case, we tested the strength of the TTOCONROT to infer the properties of data sets generated from linear functions in the plane Figure 5 8 shows different snapshots of how the TTOCONROT learns the data generated from a curve Random initialization was used by uniformly drawing points from the unit square Observe that the original data points do not *lie in the curve* Our aim here was to show how our algorithm could learn the structure of the data using *arbitrary* (initial and "non-realistic") values for the codebook vectors Figures 5 8b and 5 8c depict the middle phase of the training process, where the edges connecting the neurons are omitted for simplicity It is interesting to see how, after a few hundred training steps, the original chaotic placement of the neurons are rearranged so as to fall within the line described by the data points The final configuration is shown in Figure 5 8d The reader should observe that after convergence has been achieved, the neurons are placed almost equidistantly along the curve Even though the codebooks are not sorted in and increasing numerical order, the hidden tree and its root, denoted by two concentric squares, are configured in such a way that nodes that are queried more frequently will tend to be closer to the root In this sense, the algorithm is not only capturing the essence of the topological properties of the data set, but at the

same time rearranging the internal order of the neurons according to their importance in terms of their probabilities of access



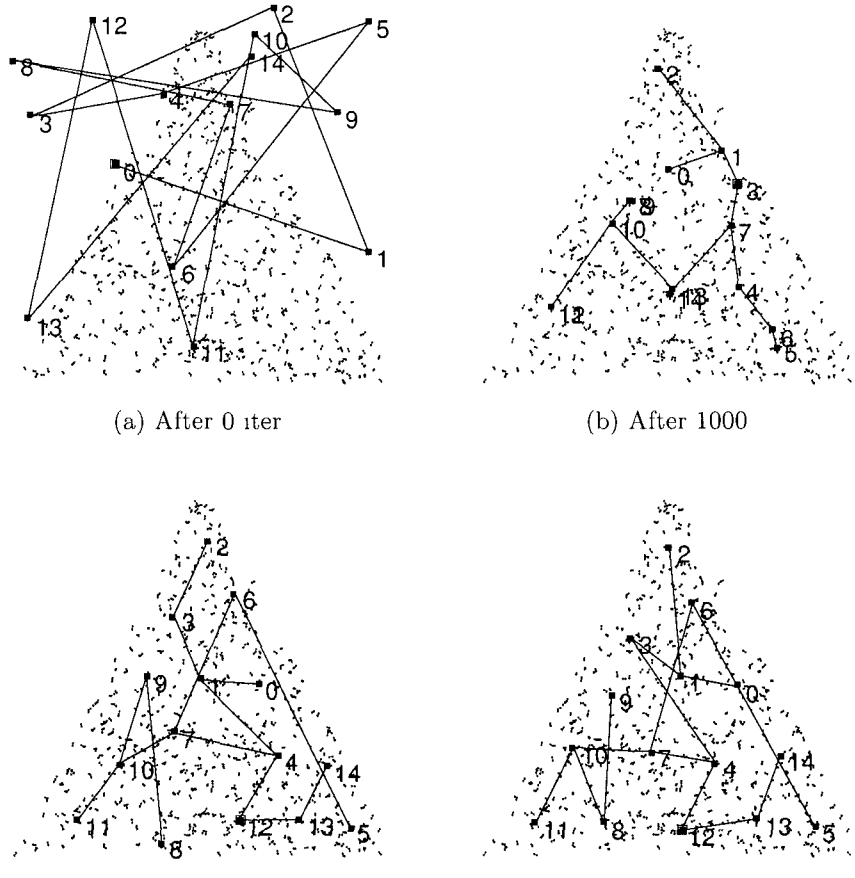
(a) After 0 iter (b) After 1000

(c) After 3000 (d) After 5000

Figure 5 8 A 1-ary tree, i e a *list* topology, learns a curve For the sake of simplicity, the edges are omitted

Two dimensional data points: To demonstrate the power of including ADS in SOMs, we shall now consider the same two-dimensional data sets studied in Chapter 3 First we consider the data generated from a triangular-spaced distribution, as shown in Figures 5 9a-5 9d In this case, the initial tree topology is unidirectional, i e , a list, although, realistically, this is quite inadvisable considering the true (unknown) topology of the distribution In other words, we assume that the user has *no a priori* information about the data distribution Thus, for the initialization phase, a 1-ary tree is employed as the tree structure, and the respective keys are assigned in an increasing order Observe that in this way we are providing minimal information to the algorithm The root of the tree is marked with two concentric squares, i e , the neuron labeled with the index 1 in Figure 5 9a Also, with regards to the feature space, the prototype vectors are initially randomly placed In the first iteration, the

linear topology is lost, which is attributable to the randomness of the data points As the prototypes are migrated and reallocated (see Figures 5 9b and 5 9c), the 1-ary tree is modified as a consequence of the *rotations* Such a transformation is completely novel to the field of SOMs Finally, Figure 5 9d depicts the case after convergence has taken place Here, the tree nodes are uniformly distributed over the entire triangular domain. The BST property is still preserved, and further rotations are still possible if the training process continues



(a) After 0 iter (b) After 1000

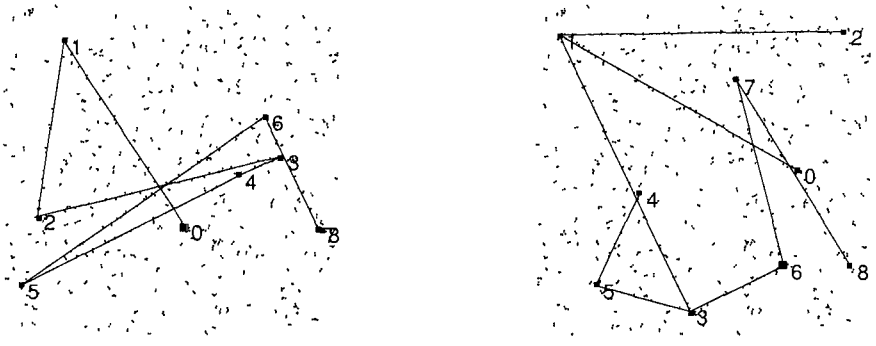
(c) After 3000 (d) After 5000

Figure 5 9 A 1-ary tree, i.e , a *list* topology, learns a triangular distribution The data structure is self-adapted so that nodes accessed more frequently are moved closer to the root conditionally The BST property is also preserved

This experiment serves as an excellent example to show the differences between our current method and the original TTOSOM algorithm (described in Chapter 3), where the same data set with similar settings was utilized In the case of the TTOCONROT the points effectively represent the entire data set However, the reader must observe that we do not have to provide the algorithm with any particular *a priori* information about the structure of the data distribution – this is learned during the training process, as shown in Figure 5 9d Thus, the specification of the initial "user-defined" tree topology (representing his perspective of the data space) required by the TTOSOM is no longer mandatory, and an alternative specification which *only* requires the number of nodes in the initial 1-ary tree is sufficient

A second experiment involves a square-shaped distribution, also studied in Chapter 3 Here a unit-square in the 2-dimensional space is learned using the TTOCONROT algorithm To start with, the randomness of the point selection process forces the codebooks to be positioned in a chaotic manner, as depicted in Figure 5 10a From the next snapshots of the learning process, shown in Figures 5 10b and 5 10c, it is possible to perceive that the codebook vectors represent the structure of the data distribution more accurately Finally, the convergence of the entire training execution phase is displayed in Figure 5 10d Here, it is possible to see that the codebooks represent the structure accurately, suggesting a square structure for the data distribution We agree that the original SOM, taking advantage of a grid-like neural configuration is quite capable of representing such a square-shaped distribution, but we emphasize that the TTOCONROT infers the same structure without receiving any *a priori* information about the distribution or its structure

The experiment shown in Figures 5 11a-5 11d considers data generated from an irregular shape with a concave surface Again, as in the case of the experiments described earlier, the original tree includes 15 neurons arranged unidirectionally, i e , as in a list As a result of the training, the distribution is learned and the tree is adapted accordingly, as illustrated in Figure 5 11d Observe that the random initialization is performed by randomly selecting points from the unit square, and this points thus do not necessarily fall within the concave boundaries Although this initialization scheme is responsible of placing codebook vectors outside of the irregular



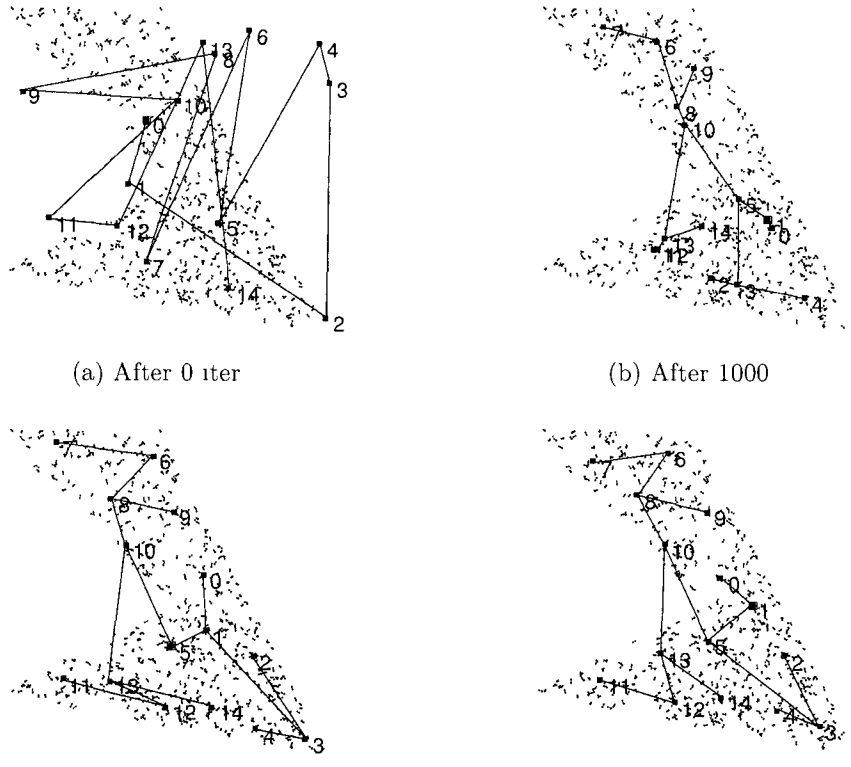
(a) After 0 iter (b) After 1000

(c) After 3000 (d) After 5000

Figure 5 10: A 1-ary tree, i e., a *list* topology, learns a square distribution. The neurons accessed more frequently are promoted closer to the root conditionally, as per the CONROT algorithm.

shape, the reader should observe that in a few training steps, they are repositioned inside the contour. The final distribution of the points is quite amazing!

Three dimensional data points: We will now explain the results obtained when applying the algorithm with and without CONROT. To do this we opt to consider three-dimensional objects. The experiments utilize the data generated from the contour of the unit sphere as in Chapter 3. It also initially involves an *uni*-dimensional chain of 31 neurons Additionally, in order to show the power of the algorithm, both cases initialize the codebooks by randomly drawing points from the unit cube, which thus initially places the points outside the sphere itself. Figure 5.12 presents the case when the basic TTOSOM algorithm (without CONROT) learns the unit sphere



(a) After 0 iter

(b) After 1000

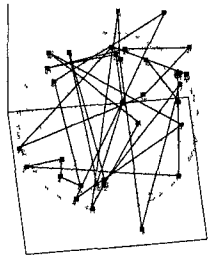
(c) After 3000

(d) After 5000

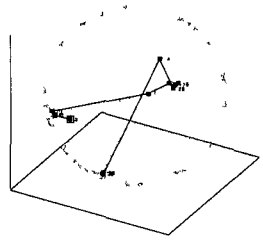
Figure 5.11 A 1-ary tree, i e , a list topology, learns different distributions from a concave object using the TTOCONROT algorithm The set of parameters is the same as in the other examples

without performing conditional rotations. The illustration presented in Figure 5 12a show the state of the neurons before the first iteration is completed. Here, as shown, the codebooks lie inside the unit cube, although some of the neurons are positioned outside the boundary of the respective circumscribed sphere, which is the one we want to learn. Figures 5 12b and 5 12c depict intermediate steps of the learning phase. As the algorithm processes the information provided by the sample points and the neurons are repositioned, and the chain of neurons is constantly "twisted" so as to adequately represent the entire manifold. Finally, Figure 5 12d illustrates the case when the convergence is reached. In this case, the one-dimensional list of neurons is evenly distributed over the sphere, preserving the original properties of

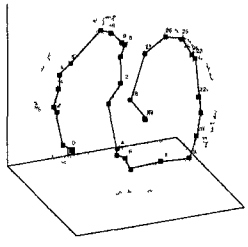
the 3-dimensional object and also presenting a shape which reminds the viewer of the so-called Peano curve [148].



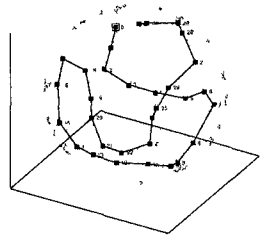
(a) After 0 iter



(b) After 1000



(c) After 3000

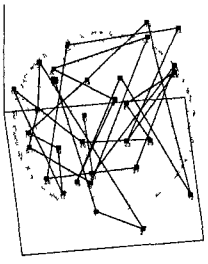


(d) After 5000

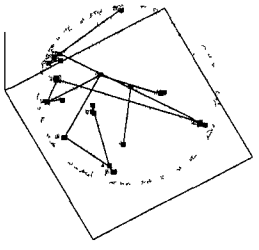
Figure 5.12: A 1-ary tree, i e. a list topology, learns a sphere distribution when the algorithm does not utilize any conditional rotation.

A complimentary set of experiments where the learning of the same unit sphere with the TTOSOM augmented by conditional rotations (i.e., CONROT) was also conducted Figure 5.13a shows the initialization of the codebooks. Here, the starting positions of the neurons fall within the unit cube as in the case displayed in Figure 5.12a. Figures 5.13b and 5.13c show snapshots after $1,000$ and $3,000$ iterations respectively. In this case the tree configuration obtained in the intermediate phases differ significantly from those obtained by the corresponding configurations shown in Figure 5.12, i.e , those that involved no rotations. In this case, the list rearranges

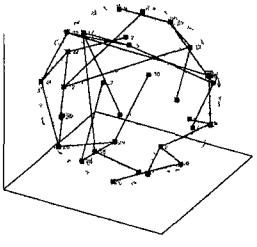
itself as per CONROT, modifying the original chain structure to yield a more-or-less balanced tree. Finally, from the results obtained after convergence, and illustrated in Figure 5.13d, it is possible to compare both scenarios. In both cases, we see that the tree is accurately learned. However, in the first case, the structure of the nodes is maintained as a list throughout the learning phase, while, in the case when CONROT is applied, the configuration of the tree is constantly revised, promoting those neurons that are queried more frequently. Additionally, the experiments show us how the dimensionality reduction property evidenced in the traditional SOM, is also present in the TTOCONROT. Here, an object in the 3-dimensional domain is successfully learned by our algorithm, and the properties of the original manifold are captured from the perspective of a tree.



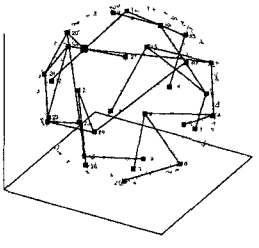
(a) After 0 iter



(b) After 1000



(c) After 3000



(d) After 5000

Figure 5.13: A 1-ary tree, i.e., a list topology, learns a sphere distribution.

Non-Contiguous Data Points

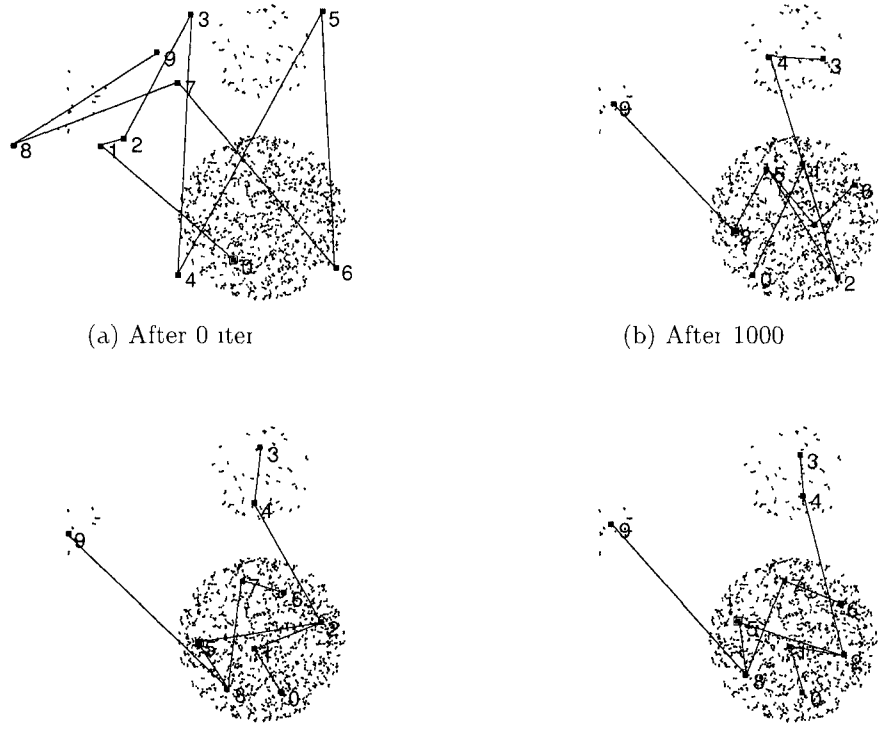
We now move our attention to a fascinating scenario, which involves the power of the scheme in *disjoint* spaces. An example of such a space is shown in Figure 5.14. In this particular data set, we deal with data generated from a distribution involving three circular-shaped "sub-distributions". For simplicity, each "cloud" considers a density of points proportional to its diameter. In this experiment, again, in the first iteration, the original 1-ary tree structure is lost because of the random selection of the codebook vectors. The transformation of the codebooks as the experiment evolves is shown in Figures 5.14b and 5.14c respectively. Interestingly, after convergence, and as depicted in Figure 5.14d, the algorithm places a proportional number of codebook vectors in each of the three circles according to the density of their data points. Note too that no codebook vector falls outside of the respective clouds, which in our opinion, is quite fascinating.

5.4.2 Skeletonization

In general, the main objective of skeletonization consists of generating a simpler representation of the shape of an object. The authors of [141] refer to skeletonization in the plane as the process by which a 2-dimensional shape is transformed into a 1-dimensional one, similar to a "stick" figure, as explained in Chapter 3.

The traditional methods for skeletonization (see Section 3.3.3) assume the connectivity of the data points, and when this is not the case, more sophisticated methods are required. Previous efforts, involving SOM-based variants which achieve skeletonization, has been proposed in [11, 46, 165]. We remark that the TTOSOM (discussed in Chapter 3) is the only one which uses a tree-based structure. The TTOSOM assumes that the shape of the object is not known *a priori*. Rather, this is learned by accessing a single point of the entire shape at any time instant. Our results reported earlier confirm that this is actually possible, and we now focus on how the conditional rotations will affect such a skeletonization.

Figure 5.15 shows how the TTOCONROT learned the skeleton of different objects in the 2-dimensional and the 3-dimensional domain. In all the cases the same



(a) After 0 iter (b) After 1000

(c) After 3000. (d) After 5000

Figure 5.14: A 1-ary tree, i.e., a list topology, learns a distribution which include 3 clouds of points. Again, the data structure is learned while preserving the BST property.

schedule of parameters were used, and only the number of neurons employed was chosen proportionally to the number of data points contained in the respective data sets. It is important to remark that we did not invoke any post-processing of the edges, e.g., MST, and that the skeleton observed was exactly what our BST-SOM learned. Firstly, Figures 5.15a-5.15d illustrate the shapes of the silhouette of a human, a rhinoceros, a 3d representation of a head, and a 3d representation of a woman. The figures also show the trees learned from the respective data sets. Additionally Figures 5.15e-5.15h display only the data points, which in our opinion are capable of representing the fundamental structure of the four objects in a 1-dimensional way effectively.

As a final comment, we stress that all the shapes employed in the experiments involve the learning of the "external" structure of the objects. For the case of solid objects, if the internal data points are also provided, the TTOCONROT is able to give an approximation of the so-called endo-skeleton, i.e., a representation in which the skeleton is built inside the solid object.

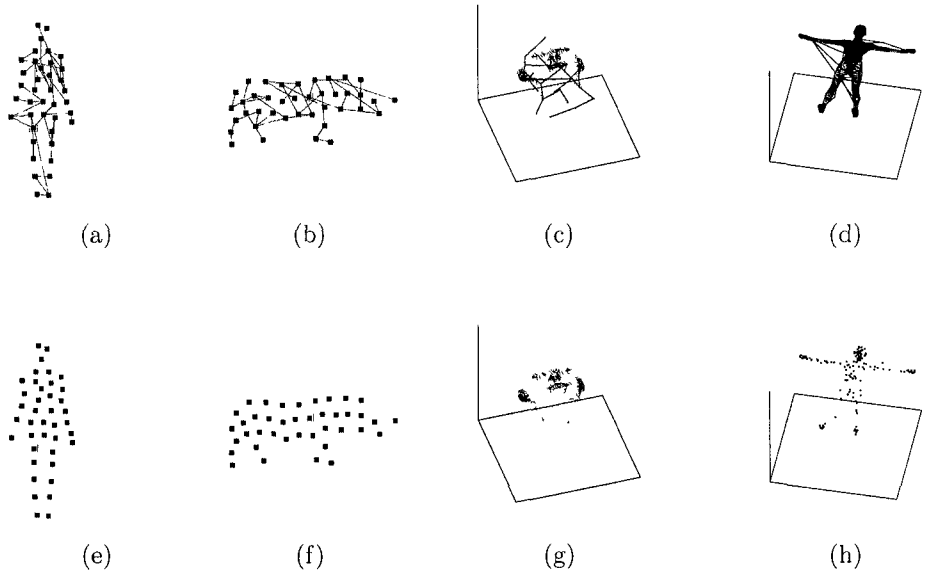


Figure 5.15: TTOCONROT effectively captures the fundamental structure of four objects in a 1-dimensional way.

5.5 Conclusions

In this chapter, we have proposed a novel integration between the areas of Adaptive Data Structures (ADSs) and the Self-Organizing Maps (SOMs). In particular we have shown how a tree-based SOM can be adaptively transformed by the employment of an underlying Binary Search Tree (BST) structure, and subsequently, restructured using rotations that are performed conditionally. These rotations on the nodes of the tree are local, can be done in constant time, and performed so as to decrease the Weighted

Path Length (WPL) of the entire tree. One of the main advantages of the algorithm, is that the user does not need to have any *a priori* knowledge about the topology of the input data set. Instead, our proposed method, namely the TTOCONROT, infers the topological properties of the stochastic distribution, and at the same time, attempt to build the best BST that represents the data set.

Chapter 6

Pattern Recognition using the TTOCONROT

6.1 Introduction

In Chapter 5, we presented a clustering algorithm that combined the philosophies defined by the tree-structured families of SOMs and the field of ADSs. The pioneering manner in which we perceived clustering, attempted to generate asymptotically optimal trees based on the access probabilities of the neurons. In this chapter we design a classifier based on these principles, and show the effect of the classification accuracies obtained on different real-world domains.

To report the contribution of this chapter in the context of the work done in Chapter 4, we mention that in Chapter 4, we designed a classifier based solely on the TTOSOM algorithm, and showed that it was able to learn the decision boundaries based on labeled and unlabeled samples simultaneously. This so-called "semi-supervised" learning classifier utilized the strategy presented by Zhu [176], which identified clusters by using a possibly large number of unlabeled samples, and subsequently, associating each neuron with a label by utilizing a small number of labeled instances. We showed that this approach, indeed, can produce accuracies that are reasonably comparable to the ones achieved by state-of-the-art classifiers.

A natural extension to our investigation is to develop a study analogous to the

one performed in Chapter 4, but now considering the effect of the rotations in the tree. In this sense, our proposed methodology consists of deriving a classifier similar to the TTOSOM-based classifier, but using the TTOCONROT as a foundation.

As most of the necessary background regarding the SOM, the TTOCONROT and the state-of-the-art supervised classifiers has already been discussed in the previous chapters, to avoid repetition, we briefly explain the design of the classifier and directly summarize the results obtained.

6.1.1 Contributions of the Chapter

1. We present a method that employs a tree-based NN for performing classification. The novel mechanism, apart from incorporating the information provided by unlabeled and labeled instances, re-arranges the nodes of the tree as per the laws of ADSs.

2. Particularly, we investigate the PR capabilities of the TTOSOM when Conditional Rotations (CONROT) [38] are incorporated into the learning scheme.

3. The learning methodology inherits all the properties of the TTOSOM-based classifier designed in Chapter 4. However, we now augment it with the property that frequently accessed nodes are moved closer to the root of the tree.

4. Our experimental results show that on average, the classification capabilities of our proposed strategy are reasonably comparable to those obtained by some of the state-of-the-art classification schemes that only use labeled instances during the training phase.

5. The experiments also show that improved levels of accuracy can be obtained by imposing trees with a larger number of nodes.

6.1.2 Organization of the Chapter

The remainder of the chapter is organized as follows The next section explains how the TTOCONROT is used to perform classification[1] Section 6 3 focuses on the experimental results, and finally, Section 6 4 contains the conclusions

6.2 The TTOCONROT-Based Classifier

In Chapter 4, we designed a classifier based on the TTOSOM algorithm that was able to learn from labeled and unlabeled samples This so-called "semi-supervised" learning classifier utilized the strategy presented by Zhu [176], and consisted of clustering the instances using a "massive" number of unlabeled samples, and subsequently, identifying the label of each neuron based on a possibly small number of labeled instances We showed that this approach can, indeed, produce accuracies that are comparable to the ones achieved by state-of-the-art classifiers

Our goal in this chapter is to devise a classifier analogous to the one presented in Chapter 4, but this time based on the foundation of the TTOCONROT instead of the TTOSOM In order to clarify the way in which this classifier is built, we will first summarize the main properties of the above-mentioned clustering technique

The reader will recall that in Chapter 5, we had merged the fields of SOMs and ADS The adaptive nature of the strategy presented, namely the TTOSOM with Conditional Rotations (TTOCONROT), is unique because adaptation is perceived in two forms The migration of the codebook vectors in the feature space is a consequence of the SOM update rule, and the rearrangement of the neurons *within* the tree is a result of the ADS-related rotations This reorganization can be perceived to be both automatic and adaptive, such that on convergence, the DS tends towards an optimal configuration with a minimum average access time In most cases, the most probable element will be positioned at the root (head) of the tree (DS), while the rest of the tree is recursively positioned in the same manner

[1] To render the chapter to be self contained, the next section is marginally repetitive with respect to what has been done earlier It describes the highlights of the TTOSOM, CONROT and the TTOCONROT

The TTOCONROT is a further enhancement of the TTOSOM (described in Chapter 4) which considers how the underlying tree itself can be rendered dynamic and adaptively transformed To do this, we presented a method by which a SOM with an underlying BST structure can be adaptively re-structured using Conditional Rotations [38] These rotations on the nodes of the tree are local, can be done in constant time, and performed so as to decrease the WPL of the entire tree In Chapter 5, we also introduced the concept referred to as *Neural Promotion*, where neurons gain prominence in the NN as their significances increase The advantages of such a scheme is that the leaned tree learns the topological peculiarities of the stochastic data distribution, and at the same time recursively positions neurons accessed more often close to the root As a result, the TTOCONROT, converges in such a manner that the neurons are ultimately placed in the input space so as to represent its stochastic distribution, and additionally, the neighborhood properties of the neurons suit the best BST that represents the data

Even though, we explained the advantages of the CONROT algorithm, the proposed architecture allows the inclusion of alternative restructuring modules other than the CONROT Potential candidates which can be used to perform the adaptation are the splay and the MT algorithms, among others

Analogously to the classifier devised in Chapter 4, our aim is to design a classifier that works in two stages First of all, the data distribution and its structure is learned in an unsupervised manner using the TTOCONROT scheme In a second phase, we utilize some labeled samples to categorize the decision regions that have been previously created

The TTOCONROT-based classifier essentially utilizes the cluster-and-label paradigm, explained in Section 4 4 1, which leads to an algorithm similar to the one described in Algorithm 18, with the difference that the TTOCONROT scheme is used as the unsupervised learning algorithm (on line 1)

6.3 Experimental Results

6.3.1 Experimental Setup

In order to verify the capabilities of the TTOCONROT for classifying items belonging to the real-world domain, and for making the results comparable to the ones obtained by the TTOSOM-based classifier, we have chosen the same datasets described in Table 4 1 For a detailed explanation of each of the six datasets included in our study, we refer the reader to Section 4 5 1

Analogous to the experiments performed in Chapter 4, the classifiers considered in this comparison include five supervised classifiers, namely, BN, NB, C4 5, k-NN and LVQ1, and three "semi-supervised" classifiers, namely the TTOCONROT, the TTOSOM and the SOM The sampling method utilized to measure the *acc* was the stratified 10-fold cross validation

6.3.2 Comparison to Other Classifiers

We started our experimental analysis by comparing the accuracies of the TTOCONROT with the rest of the classifiers mentioned above, using the parameter settings specified in Section 4 5 2 The results obtained are presented in Table 6 1, which shows the accuracy of the classifiers obtained for the various datasets For example, Table 6 1 shows that the TTOCONROT classifier, using *only* 15 neurons, accurately predicts, with an accuracy of 96 07%, the correct label of the instances belonging the *wine* dataset, which is outperformed only by the BN and the NB schemes On the other hand, the SOM classifies correctly the same dataset with an *acc* of only 67 98%!

We have also compared the TTOCONROT-based classifier with other VQ-based methods One such strategy that belongs to the supervised family is the LVQ1, while the SOM and the TTOSOM and the TTOCONROT primarily learn the distributions using an unsupervised learning paradigm All four classifiers used the same values for their parameters (see Section 4 5 2) In addition, the LVQ1 and the SOM used 128 neurons, and the results shown for the TTOSOM and the TTOCONROT include *only* 15 neurons, respectively As per our results, the TTOCONROT, using only a small

Dataset	ROT15	TTO15	BN	NB	C4.5	KNN	LVQ1	SOM
iris	94 00	92 00	92 67	96 00	96 00	95 33	96 00	84 67
wdbc	93 32	92 09	95 08	93 15	93 15	96 66	92 09	90 51
glass	53 74	52 34	71 96	49 07	67 76	67 76	61 22	63 08
wine	96 07	95 51	98 88	97 19	93 82	94 94	74 16	67 98
yeast	51 08	51 82	56 74	57 61	55 86	54 78	24 33	46 16
winequality	53 60	53 41	57 72	55 03	62 91	57 79	44 15	49 59

Table 6 1 General classification results of the TTOCONROT and other methods investigated, reported in terms of the accuracy, *acc*

percentage of the neurons used in the SOM and LVQ1 (almost 10%), outperforms their recognition capabilities in almost *all* six datasets[2] The differences with respect to the TTOSOM are more subtle and are analyzed in a subsequent section

Similar to the TTOSOM, we observe that the TTOCONROT offers accuracies comparable to the ones obtained by certain supervised classifiers For instance, the results are similar to what one obtains from using the k-NN However, as stated in Chapter 4, even though the k-NN is internally used for the labeling of the neurons, this is done only once, and applies to only a small subset of the neurons, which represent a small fraction of the total number of sample, i e , those which are involved in the computations of the k-NN every time a query is performed

Another advantage that the scheme presents is its "semi-supervised" nature, that allows it to associate the neurons with a class label using only a minimum number of tagged instances In cases when these samples are scarce (but when the unlabeled samples are abundant), it has been shown that other schemes that belong to the same "semi-supervised" family, yield competitive results as pointed out in [73] and as our results in Chapter 4 demonstrate

[2]The table show the results for the case when we have only 15 neurons But if the number of neurons is increased to 127, the accuracy is superior in all the VQ-based algorithms

6.3.3 Effect of the Number of Neurons

We now consider the effect of varying the number of neurons involved in the TTOCONROT tree To test this, we trained the TTOCONROT with the configuration presented in Section 4 5 2, and steadily augmented the size of the respective tree Analogous to the experimental settings used in Chapter 4, we permitted the starting value for the radius to be twice the depth of the tree, so as to ensure that all the neurons are initially considered as part as the BoA

Table 6 2 shows the accuracies obtained by the TTOCONROT, where in the respective column, the specific tree size is systematically increased The respective graphical curves are illustrated in Figure 6 1 To cite an example, Table 6 2 shows that the TTOCONROT classifier, using 127 neurons, predicts with an accuracy of 55 60%, the actual category of the instances belonging to the *winequality* dataset The experiments use an analogous parameter configuration in which we systematically increase the size of a full binary tree with depths ranging from 3 to 12 Observe that the TTOSOM tree was restricted to be binary, even though it could have been of an arbitrary size The reason for this was to render the results comparable to the ones obtained by the TTOCONROT which is constrained by a BST structure Trees with small size were used to test the capabilities of the classifiers with a very condensed representation of the feature space On the other hand, trees with a larger size were utilized so as to observe the effect of adding artificial data points which, in some cases was even greater than the number of sample points themselves These artificial points attempted to preserve the original properties of the feature space

Dataset ↓ Neurons →	**7**	**15**	**31**	**63**	**127**	**255**	**511**	**1023**
glass	51 87	53 74	63 08	67 76	68 22	66 36	66 36	67 29
iris	92 00	94 00	92 67	94 67	93 33	94 67	94 00	94 00
wdbc	88 23	93 32	94 55	95 96	94 55	95 08	96 13	95 43
wine	91 01	96 07	97 19	94 38	97 19	97 19	96 07	95 51
winequality	53 85	53 60	53 28	54 72	55 60	56 85	56 66	58 79
yeast	50 74	51 08	53 23	55 73	55 32	50 27	51 21	52 29

Table 6 2 The accuracy of the TTOCONROT as the number of neurons increases

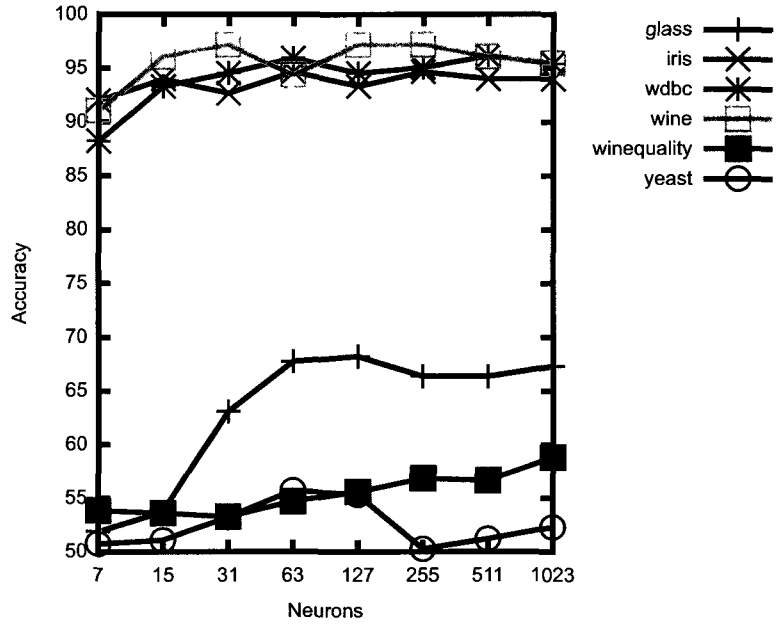


Figure 6.1: The accuracies for the different datasets as obtained by using the TTOCONROT-based classifier and an increasing number of neurons.

6.3.4 Difference of classifying with and without Conditional Rotations

We have also investigated the effect of the rotations and studied how the accuracies vary as the number of neurons is increased.

The results for the *wine* dataset when the number of neurons is increased are shown in Table 6.3. In the table, each row presents the accuracies obtained by using a specific tree size, and each column indicates the accuracies obtained by the TTOCONROT and the TTOSOM, respectively. In order to verify if one strategy performs better than the other, we have computed the average accuracy and ranking indices. As per our observations, both algorithms possess a similar pattern. The maximum accuracy obtained in both cases is 97.19%. However this "peak" is reached by the TTOSOM only once (when using 127 neurons), while the TTOCONROT-based classifier achieve this in 3 instances, i.e., when using 31, 127 and 255 neurons respectively. In this regard, it is worth mentioning that based on this results, only the BN (explained in

Section 4.2.3 and which belongs to the "supervised" family) could outperform this accuracy, obtaining in that case, 98.88%, i.e., only a fraction better than the results obtained by our proposed methods. It is remarkable that the TTOCONROT was able to provide almost the highest accuracy possible, in comparison to the state-of-the-art classifiers included in our study, using only a limited number of 31 neurons. The fact that this result can be replicated by using a larger tree, further demonstrates the consistency of the method. From our perspective, this evidence suggests that the user does not need to know *a priori* the exact number of neurons required to train the tree effectively.

Neurons	TTOCONROT	TTOSOM
7	91.01	94.94
15	96.07	95.51
31	97.19	95.51
63	94.38	96.07
127	97.19	97.19
255	97.19	96.63
511	96.07	96.07
1023	95.51	96.63
2047	96.07	96.63
4095	96.07	96.07

Table 6.3: *Wine* dataset – Accuracy rate in % obtained by using the TTOCONROT and the TTOSOM as the size of the tree is increased.

Another dataset that we have considered belongs to the same problem domain, i.e., the *winequality* dataset. However, the latter presents a harder classification problem, in which the state-of-the-art supervised classifiers provide accuracy rates which are roughly between 50% and 60%. Figure 6.2 illustrates the differences in accuracy obtained by the TTOCONROT and the TTOSOM as the size of the tree is increased.

Observe that in both cases, the classifiers have a tendency to increase their recognition capabilities as the number of neurons is increased. However, we observe that the TTOCONROT presents an almost monotonic non-decreasing behavior. From this behavior we believe that when solving practical problems, it is worth training

the classifier with trees that possess even more neurons than the number of training samples. From our experiments, we infer that it is possible to improve the accuracy rates, if additional computational power, time and/or space are available. We believe that this occurs because the TTOCONROT tree also effectively covers those regions where no samples lie, and this is used by the classifier to accurately predict the class labels of those regions when labeled instances become available.

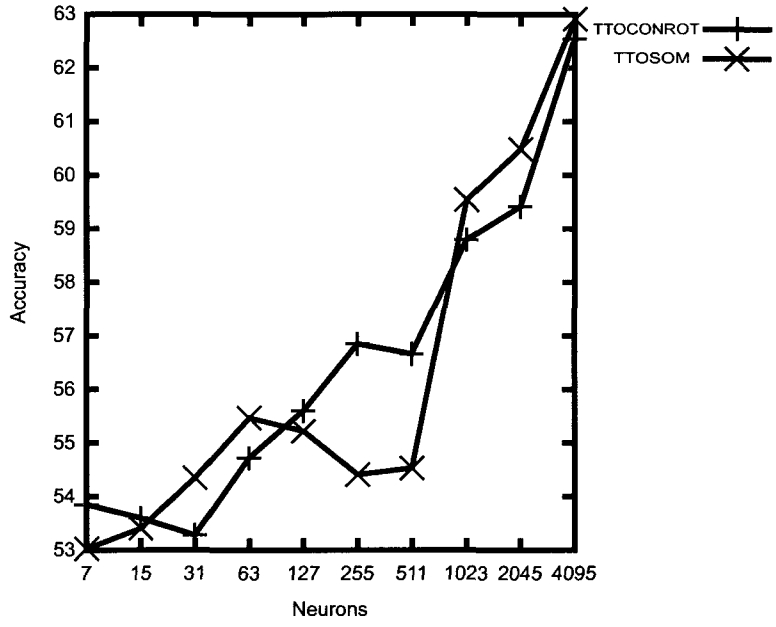


Figure 6.2: The *winequality* dataset is learned using the TTOCONROT-based classifier and the TTOSOM-based classifier. In each case the number of neurons is increased systematically.

6.4 Conclusions

This chapter has presented the design and experimental analysis of a PR scheme based on the TTOCONROT. The approach utilizes the tree-based neural network to learn the distribution of *all* the samples available, regardless of the fact that their labels are known, and then utilize a set of labeled samples (expected to be scarce), to categorize the regions of the feature space. In particular, the proposed scheme constrains the

neural tree as per the laws of the field of Adaptive Data Structure (ADS)

Our experiments demonstrated that, the TTOCONROT-based classifier is able to sometimes outperform state-of-the-art classifiers that use the supervised learning paradigm, i e , those which are unable to learn from unlabeled samples This is concurs with the results of other researchers who observe that under certain scenarios, semi-supervised schemes like the one presented in this chapter, can lead to performance levels that are comparable to the ones obtained by true supervised methods [73] Particularly, in most of our experiments, trees whose sizes are only a small fraction of the cardinality of the dataset, are sufficient to obtain accuracies comparable to the ones provided by the best supervised classifiers

Additionally, we have performed a meticulous analysis to identify the advantages of incorporating the neural rotations provided by the TTOCONROT To do this, we compared the results with the TTOSOM-based classifier (presented in Chapter 4), using analogous parameter settings and using different tree sizes Our results showed that regardless of the inclusion of the rotations, competitive accuracy rates can be obtained Moreover, our experiments also suggest that in certain cases, the rotations lead to accuracy rates that increase in a smoother manner, in comparison to the ones obtained by the TTOSOM-based classifier, as more neurons are incorporated in the tree

Chapter 7

Fast BMU Search Using Random Hyperplane Trees

7.1 Introduction

One task that recurrently appears in many areas of CS involves the Pattern Matching (PM) problem, where a given sample is compared with a set of stored items and the objective is to find the item that is most similar to the stimuli [40]. In particular, this problem arises in the self-organizing mechanism of the VQ algorithm, which is central to the SOM, and which consists of identifying a winner neuron during the so-called CL process. The most common way of determining the BMU in the map is by an exhaustive search, which was detailed previously in Section 2.5.2. However, under certain scenarios, such as, for example, in real-time applications [126], the handicap of the deterministic search is that it may necessitate a prohibitive amount of time to perform the required computations. Reducing the time required for identifying the BMU has become one of the major objectives of the SOM-variants reported in the literature [143], and its further investigation is the initial goal of the present chapter.

In this chapter we shall present a novel scheme that attempts to reduce the time required for identifying the so-called BMU. Our strategy utilizes a subset of the input manifold to recursively build a BST, in which each dichotomization is induced by a hyperplane that fits a number of points sampled *randomly* from the dataset.

As a direct consequence of building such a tree, the time required for identifying the BMU will no longer be linearly dependent on the total number of neurons Rather, the expected time turns out to be logarithmic with respect to the height of the randomly-generated hyperplane-based tree Our enhanced DS learns the static properties of the input manifold, and at the same time, keeps track of the current location of the codebook vectors To the best of our knowledge, the use of random hyperplanes to identify the BMU is novel, and what we submit in this chapter are our preliminary results Our focus is placed on the practical aspects of this family of search methods

In the field of PR, it is possible to encounter a number of classifiers that utilize hyperplane-based trees to recursively split the dataset (c f [7, 16, 65, 76, 79, 83, 118, 137, 132]) In particular, the authors of [79] utilize hyperplanes to derive decision boundaries that are invariant under strictly monotone transformations of the coordinate axes This fact concurs with the results of [53], where the authors mention that an important property of hyperplane trees is that their shapes are invariant under rotations and, indeed, under linear transformations, in general As a matter of fact, rotations do alter the form of *k-d* trees [20] and quadtrees [162]

Classifiers, such as the ones mentioned above, typically work under the assumption that the items posses a well-defined taxonomy However, our focus is rather different, as we intend to provide a scheme applicable to the VQ migration, which is unsupervised in nature Thus, we make no assumption regarding the category of the items One interesting aspect of building a VQ-search module like the one devised in the present chapter, is that it is general enough to be compatible with most of the SOM variants reported in the state-of-the-art For instance, this scheme can be "plugged into" the TTOCONROT scheme, keeping the rest of the algorithm untouched The rest of the chapter will focus on the BMU search problem from a VQ perspective, i e , with an attempt to be as generic as possible in order to make these results valid to other SOM-based methods, like the ones surveyed in Chapter 2

In neural systems like the ones investigated in the present thesis, one encounters a trade-off scenario, when one needs to determine the correct number of neurons needed so as to obtain high-quality clusters, and at the same time minimizing the time spent to train the network Accelerating the time required for identifying the BMU, will

not make such a method better *per se* However, the acceleration of the BMU search computation, implies that the system is able to handle more neurons by investing a time component similar to a NN which does not perform an enhanced BMU search scheme, and which thus possesses a lesser number of neurons Moreover, assuming that more neurons yield to a more detailed representation of the stochastic distribution, one might expect a better quality of the resulting clusters This assumption is founded on our observations in Chapters 3, 4, 5 and 6, where both the clustering and classification applications of the TTOSOM and TTOCONROT are improved when more neurons are incorporated into the NN

In the previous paragraph, we implicitly assumed that the result returned by the enhanced BMU search scheme is as good as the deterministic search Another situation appears when the BMU search module returns an approximation of the true BMU The state-of-the-art reports algorithms where a tree structure is utilized so as to get the closest codebook vector to a given sample [80], and which is further refined so as to be applicable to a tree-structured SOM [105, 143]

In general, the quality of the VQ process increases as the number of neurons is incremented However, the time required to deterministically find the codebook vector which is closest to an input sample increases with respect to the total number of codebook vectors in the system Even worse, the adequate number of codebook vectors that leads to satisfactory results usually grows exponentially with respect to the dimensionality of the input set [10] (due to the so-called "curse of dimensionality" [57]) The state-of-the-art includes mainly two types of families to tackle this problem The first category involves the preprocessing of unstructured codebook vectors so as to reduce the complexity of locating the BMU [40, 41, 66, 154, 175] Even though fairly good results are obtained when one deals with low dimensions, unfortunately, as the authors of [167] have shown, moderately large dimensions are associated with running times that are only marginally better than the brute-force approach A second family imposes a hierarchical structure onto the codebook vectors whose goal is to *rapidly* find a "good" approximation of the BMU [98] The Tree Structured VQ was originally proposed in [36] The author of [80] explained this mechanism in detail focusing on binary trees, and also stated that more general trees (e g multiway-trees

[26]), will provide better performance, while at the same time, lead to a significant decrease in complexity In general, such a hierarchy leads to a decay in the quality of the solution because it no longer perform an exhaustive search This scheme, provides a trade-off between the quality of the codeword identified as the BMU (i e , a sub-optimal answer) and an increase in the performance of the search method [80]

7.1.1 Contribution of the Chapter

Summarizing, the main contributions of the chapter are as follows

1 We design a DS that uses random hyperplanes trees to attempt to solve the BMU search problem in VQ without checking all the neurons

2 We show that, when one is willing to relax the requirement of identifying the true BMU, it is possible to reduce the expected running time and yet have a low degradation of the resultant QE

3 The performance of our proposed algorithm has been measured in terms of the resultant QE, and these results have been compared to the scheme that finds the true BMU

4 We show that increasing the complexity on the dimensionality implies a dramatic reduction in term of the running time, while the degradation in the performance remain small

5 Our result are supported by numerous experiments on datasets with different dimensionalities and distributions

7.1.2 Organization of the Chapter

The remainder of the chapter is organized as follows In Section 7 2 we start by describing the sate-of-the-art Section 7 3 describes in detail the RHST algorithm Thereafter, Section 7 4 explains our proposed data structure that uses the RHST scheme to locate the position of the codebook vectors as they dynamically migrate

during the training process After that, Section 7 5 presents the experimental results Finally, Section 7 6 includes the conclusions and potential future areas of research

7.2 Current Fast BMU Searching Schemes

As we are interested in BMU search schemes and their application to SOM-based schemes, we can observe the methods from a VQ perspective, where no topological relationships exist between the prototypes, or, on the other hand, from a more specific SOM-based approach, where a topological connection between the neurons exist

One mechanism proposed in [96] attempts to find an approximation for the BMU based on a VQ-like scheme The scheme presented in [96] is a modified version of the so-called Threshold Order-Dependent (TOD) algorithm described in [67] It dynamically adds new codebook vectors if the distance between the stimulus and the reported BMU is greater than a given threshold The authors of [96] successfully utilized the scheme in a Support Vector Machine (SVM)-like strategy, and showed that it is possible to obtain faster results in comparison to the traditional SVM algorithm In [96], the authors only focused on the VQ case, and did not address the situation when the codebook vectors were inter-connected forming a topology, as in the case of the SOM or the TTOSOM

Another technique that has been successfully utilized for finding approximations of the BMU for VQ, is the one presented in [112] The authors of [112] utilized two inequalities, one for terminating the searching process and the other for discarding impossible codebook vectors, respectively They showed that their method can outperform state-of-the-art strategies [146, 168] However, the authors of [39] compared the results obtained in [112] with the ideas devised in [121] and explained why the latter was more efficient Further, according to the experimental results presented in [39], it appears as if their proposed method, which also employs inequalities to reject impossible codebook vectors, is one of the most competitive schemes within the family of fast VQ schemes However, it is important to mention that all the search strategies mentioned above and their applicability to SOM-based methods still remain unreported, to the best of our knowledge

The literature also report attempts to accelerate the task of finding the BMU in SOM-based schemes. In [153], the authors discussed a method which takes advantage of the hierarchical structure of a NN called the SplitNet [152]. The basic strategy of SplitNet consisted of growing a 1D SOM which is further split into two halves based on a topology preserving criterion. The BMU search is accelerated by a fast searching procedure. The process basically utilized a local search mechanism by traversing the neurons through its topological connections with the hope of finding a closer neuron. The authors of [153] also mentioned that such a scheme considerably increases the speed of identifying a BMU in the NN that they were studying. Unfortunately, the details of how this is achieved was not fully described in [152, 153]. According to the authors of [152], the SplitNet shares various phenomena with other approaches, such as the TSSOM [105, 114], and thus also the undesirable property that the search might be misled by the upper levels.

In [111], the authors proposed a mechanism that takes advantage of the topological structure of the SOM to perform a non-exhaustive search of the BMU. They mentioned that regardless of the fact that the topology preserving property of the SOM provides advantages for fast codebook searching, this is an aspect of the SOM that has been largely unexploited. The scheme presented in [111] utilized a two-step procedure. In the first step they used a suboptimal search that included a Finite State Vector Quantizer [78] which depended on an acceptance threshold. When the threshold was not satisfied, "derailment" is said to have occurred which triggered a second phase. The second phase involved a smaller SOM trained with the codebook vectors of the original SOM, i.e., a scheme *à la* TSVQ [80, 105] and which used a two-level hierarchy. As the authors of [128] observed, the quality of the topology preservation obtained by the traditional SOM degraded if there was a significant difference between the dimensions of the data and the display lattice.

7.3 Random Hyperplane Search Trees (RHSTs)

From our experience, we have observed that, in general, many of the tree-structured SOMs reported in the literature that do not perform exhaustive search, rather execute

a scheme similar to the TSVQ in one way or another (c f , [54, 80, 105, 143]) As stated earlier, we attempt to investigate a different avenue, namely the one of using a RHST, which is described in detail in the present section

The authors of [53] define a Hyperplane Search Tree (HST) as a binary tree that is able to store a set $\mathcal{S}$ of elements belonging to the d-dimensional space Additionally, they define a RHST as a HST where the points that define each hyperplane are chosen *randomly* The hyperplane associated with the root node of a RHST is obtained by randomly selecting d points which further dichotomize the hyper-space into two halves The hyperplane divides the remaining samples (i e , excluding the points chosen for building the respective hyperplane) into two smaller subsets The entire tree is obtained by recursively applying this process

RHSTs are a particular instance of BSTs When one builds a BST from a permutation of $1, \quad , n$ and i is the first element to be selected from the permutation, i is positioned at the root of the tree, the corresponding left subtree contains the keys $1, \quad , i-1$, and its shape is only dependent on the order in which these keys appear in the permutation Analogously, the right-subtree contains the keys $i+1, \quad , n$ and *its* shape also depends only on the order in which *these* keys appear in the permutation [157]

Hierarchical DSs are an important representation technique by virtue of their applicability to a broad range of fields, such as PR, Computer Graphics, Image Processing, Computational Geometry, Geographic Information Systems, and Robotics [162] These methods are founded on the philosophy of recursive decomposition, which are analogous to the divide-and-conquer paradigm [4]

There are other types of hierarchical DSs that can serve to be alternatives to RHSTs In particular, special attention in the community has been placed on k-d trees [20] and quad-trees [162] The main difference between the RHST and the well-known family of k-d trees is that the latter uses hyperplanes which are orthogonal to one of the coordinate axis, and which divide the corresponding hyper-rectangles into two sub-rectangles From our perspective, an important advantage of RHSTs is that they are invariant to rotations In particular rotations do alter the form of k-d trees and quadtrees [53]

The simplest case for a RHST arises in the 1D space In this situation a single randomly selected point dichotomizes the real line into two halves the process is then repeated recursively until all the points are selected Thus, from the perspective of the original HST definition, all points in the dataset are utilized as internal nodes and, subsequently, all the leaves of the tree become abandoned

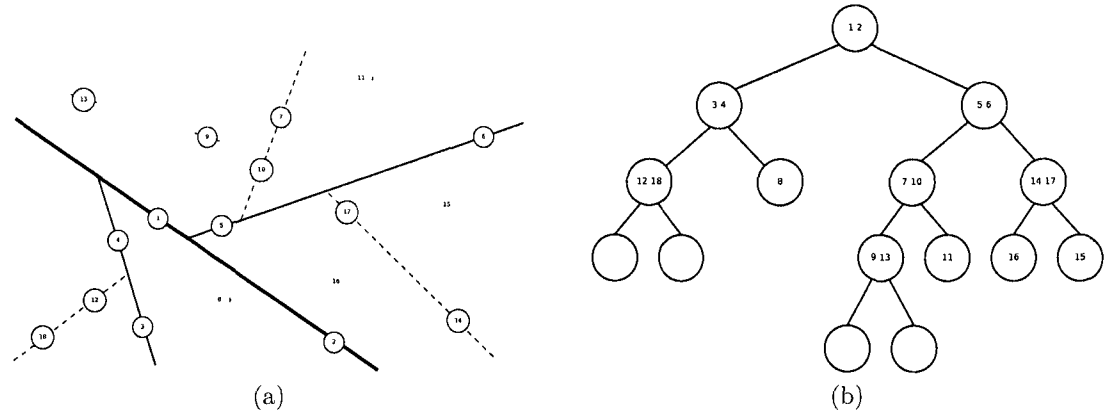
In the following subsections we shall provide all the algebraic details for the process of obtaining hyperplanes for dimensions greater than two

7.3.1 RHST in the 2-dimensional Space

For the 2D space, the points are located relative to the x-axis and y-axis of the Cartesian coordinate system, proposed by Decartes in [49], in a book which is considered the beginning of modern philosophy and mathematics

Figure 7 1 (duplicated from [53]) depicts a RHST in the 2-dimensional space and the partition of the plane into disjoint polygons that it defines As shown in Figure 7 1a, the partitions of the space are induced by the hyperplanes, which in the 2-dimensional case, correspond to lines Figure 7 1b details which points from the original manifold where utilized to create the respective hyperplanes For the 2-dimensional case, each dichotomization requires two points so as to define a line in the plane When the cardinality of the partitioned subset is less than two, it is not possible to create a line from the points Thus, this subset of points that remain are associated with the respective leaf node Observe that in the case of Figure 7 1b, the leaf nodes may store zero or a single point In general, according to the original definition given in [53], for higher dimensions, the number of points that can be stored in a leaf of the RHST ranges from zero to $d - 1$ We will modify this situation, as explained below

To render this chapter complete, we shall detail how these hyperplanes are obtained Given two points in the plane, (x_1, y_1) and (x_2, y_2), the hyperplane (i e , line) that separates the space in two halves is given by Equation (7 2)



(a) (b)

Figure 7.1: (a) The partitions of the plane induced by the RHST. (b) Each dichotomization (as per the internal nodes) force two points to create a line in the plane. Leaf nodes contain 1 or zero points.

$$x_1 + a_1 y_1 = -b_1 \tag{7.1}$$

$$x_2 + a_2 y_2 = -b_2. \tag{7.2}$$

The reader must note that, as the two points belong to the same line, both equations share the same "slope", which is given by constant a. Moreover, given that the lines represented by both equations are actually the same line, they cut the y axis at the same point b. With these facts, Equation (7.2) can be simplified to the following:

$$x_1 + a y_1 = -b \tag{7.3}$$

$$x_2 + a y_2 = -b. \tag{7.4}$$

Since a and b are the unknowns, Equation (7.4) can conveniently be reformulated as:

$$a y_1 + b = -x_1 \tag{7.5}$$

$$a y_2 + b = -x_2 \tag{7.6}$$

By using a matrix representation, Equation (7 6) can be represented as follows

$$\begin{bmatrix} y_1 & 1 \\ y_2 & 1 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -x_1 \\ -x_2 \end{bmatrix}, \tag{7 7}$$

and isolating the unknowns from Equation (7 7), we obtain

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 & 1 \\ y_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -x_1 \\ -x_2 \end{bmatrix} \tag{7 8}$$

Equation (7 8) shows that the solution to the coefficients of the function that determines the hyperplane in the 2 dimensional space, is determined by the inverse of the matrix whose first column is conformed by the y-values of the two given points, while its second column are unity, multiplied by the 2-dimensional vector containing the x-values of the given points

Example Consider the points $(1, 1)$ and $(4, 2)$ Replacing the given points in Equation (7 8) we obtain

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ -4 \end{bmatrix}, \tag{7 9}$$

and by solving a and b we obtain the following

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \end{bmatrix} \tag{7 10}$$

As per the values obtained, the hyperplane generated by the points $(1, 1)$ and $(4, 2)$ is defined by the following hyperplane (line),

$$H \quad x - 3y + 2 = 0 \tag{7 11}$$

Figure 7 2 depicts the situation, showing the two above-mentioned points along with the respective hyperplane induced by the two points

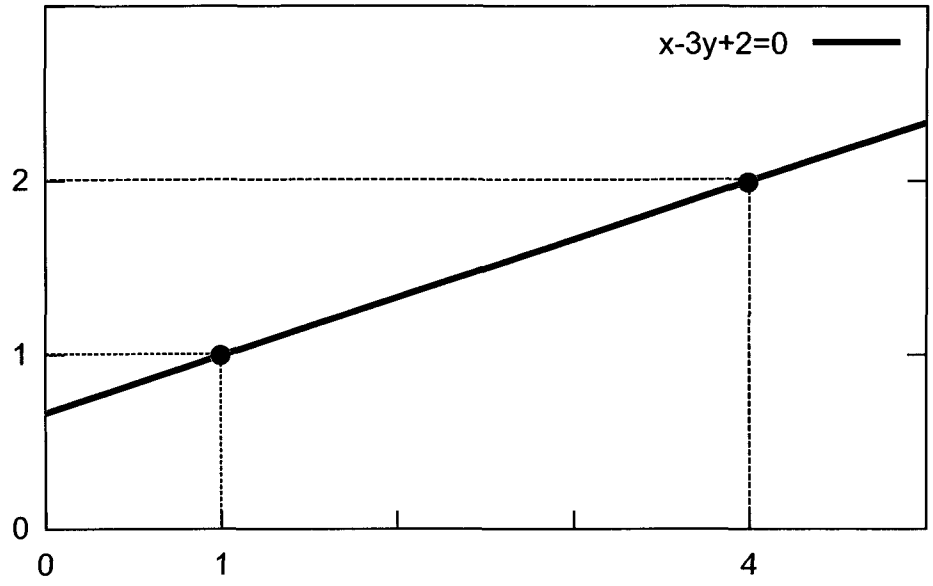


Figure 7.2: A two-dimensional hyperplane, obtained by using Equation (7.8), fits the points $(1, 1)$ and $(4, 2)$

7.3.2 Computing the Hyperplane in Higher Dimensions

Supported by the algebraic details given above in Section 7.3.1, we shall now briefly express the case for the 3D space and its generalization to higher dimensions.

A hyperplane in the 3-dimensional space is formed by specifying 3 points of the type $(x_i, y_i, z_i) \in I\!R^3$. The systems of equations that define the hyperplane is given by:

$$x_1 + ay_1 + bz_1 + c = 0 \tag{7.12}$$

$$x_2 + ay_2 + bz_2 + c = 0 \tag{7.13}$$

$$x_3 + ay_3 + bz_3 + c = 0 \tag{7.14}$$

Analogous to Equation (7.8), the unknowns can be solved, obtaining the following:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -x_1 \\ -x_2 \\ -x_3 \end{bmatrix} \tag{7 15}$$

Even though, providing a mechanism to quickly find a point in the 2D and 3D spaces finds numerous applications in Computational Geometry and GIS, from a PR perspective, where the dimensionality of the patterns are large, a more general version of equations (7 8) and (7 15) is necessary In order to derive a formula to obtain hyperplanes in higher dimensions, we first start by defining the form of a hyperplane in d-dimensions as follows

$$x_1 + a_1 x_2 + a_2 x_3 + \quad + a_{(d-2)} x_{(d-1)} + a_{(d-1)} x_d + a_d = 0 \tag{7 16}$$

Determining the hyperplane in the d-dimensional space can be achieved by specifying d-points Let $\mathcal{X}_h = \{\mathbf{x}_1, \mathbf{x}_2, \quad , \mathbf{x}_d\}$ be a set of d points from which the hyperplane is being derived, where each $\mathbf{x}_i = \langle x_{i1}, x_{i2}, \quad , x_{id} \rangle$ are points in $\mathbb{R}^d$, i e , each $x_i \in \mathbb{R}$ Assuming that the points are linearly independent, the hyperplane which divides the d-dimensional space into two halves is given by

$$\begin{bmatrix} a_1 \\ a_2 \\ \\ a_d \end{bmatrix} = \begin{bmatrix} x_{12} & x_{13} & & x_{1d} & 1 \\ x_{22} & x_{23} & & x_{2d} & 1 \\ & & & & 1 \\ x_{d2} & x_{d3} & & x_{dd} & 1 \end{bmatrix}^{-1} \begin{bmatrix} -x_{11} \\ -x_{21} \\ \\ -x_{d1} \end{bmatrix}, \tag{7 17}$$

where x_{ij} corresponds to the j^{th} component of $\mathbf{x}_i$, and where a_i is the i^{th} coefficient of the hyperplane function Without loss of generality, we further assume that the coefficient a_0, which is the one associated with x_1, is unity

7.4 Our Proposed Method

Even though the ideas presented in [53] are appealing from a theoretical perspective, when one decides to apply the above-mentioned DS to VQ-based clustering, a number

of open problems appear First, the examples presented in [53] only involve artificially generated data When working with real-world datasets, due to the nature of the samples, when one systematically divides the sub-spaces, it is possible to obtain an infinite number of hyperplanes that fit the selected d points, which is a scenario that is not fully addressed in [53] Even though the field of linear algebra and numerical analysis, provide ways to address this situation, such as re-selecting points, pseudo-inverse approaches, Cholesky decomposition [77], least-square minimization [25], etc , these issues are not discussed in detail in [53], and the authors do not clearly express if there exists an advantage of using one of these schemes over the other We believe that for a practical application of the RHST, these questions need further investigation Another important fact is that the data structure proposed in [53] implicitly assumes that the points to be searched are static As mentioned in Chapter 2, the task of finding the BMU involves the *migration* of the neurons, i e , the neurons are affected by a dynamic change in their location If one decides to apply the ideas presented in [53] to identify the location of the BMU, one must consider the effect of dynamism in the DS The question of identifying freely-moving points using a RHST, to the best of our knowledge, has remained unreported in the literature This chapter advocates the investigation of this avenue For this purpose, we first discuss the inner workings of the RHST and adapt the original algorithm to overcome the mentioned hurdles

There is a relationship between the expected depth of the tree and the sample size According to the orignal RHST definition, a recursive split is made if the sample set is greater or equal than d, the dimensionality of the dataset

In this connection, we observe that trees that are randomly constructed possess the desirable property of being well balanced with a high degree of probability For the case of randomly built BSTs, the expected height is logarithmically proportional to the cardinality of the collection of elements being processed [52, 157]

Given that the position of the codewords change over time, the sub-region where these codewords fall into, also varies In an ideal case, a single prototype will represent each of the above-mentioned sub-regions This means that one desires a number of leaves as equal as possible to $|\mathcal{W}|$, the cardinality of the set of codewords $\mathcal{W}$ The resulting RHST contains as many regions as the leaves in the tree

Once we understand the above fact, we see that many factors will influence the total number of nodes in the tree, such as the distribution of the points. Thus, it is very difficult to specify a precise number for the leaves *a priori*.

To resolve this, we have decided to define a parameter, called n_0, that relates the size of the samples in the leaves with the number of codebook vectors. In the original definition for the RHST, a set of cardinality less than d was not split. Rather, it occupies a position as a leaf in the tree. Leaves correspond, thus, to collections of points of cardinality between 0 and $d-1$ [53]. We propose to modify this definition, so that a set with cardinality less than θ, is no longer split. The value for θ is obtained by multiplying the factor n_0 by the cardinality of the original dataset. Again, the reason behind this is to maintain a total number of leaves approximately equal to the number of codewords.

An interesting point about controlling the depth of the tree based on the number of points (instead of fixing a maximum depth), is that the tree is able to grow in depth in those areas of the space where more points are concentrated, while regions with a lower density of points will necessitate a coarser resolution.

Our proposed method is described in Algorithm 21. The input of the algorithm are the sample set, and the above mentioned parameter θ, which controls the degree and depth of the tree. Algorithm 21 is essentially different to the one defined in [53] in two aspects. The first difference is that, as explained above, the algorithm stops growing the tree when the number of points is less than θ, instead of using d as the stopping criterion. The second difference is more subtle and occurs in line 5 of the algorithm. When the equation of the hyperplane is determined by computing the inverse of the matrix of the selected points, as in Equation (7.17), we observe that it is possible to obtain a non-invertible matrix. To efficiently tackle the situation mentioned above, we have proposed an inexpensive solution consisting of altering the original location of the points by adding a small quantity, for instance, by a random Gaussian noise. Although it still does not mathematically guarantee that the inverse of the resultant matrix exists, the probability of obtaining an invertible matrix will increase as the covariance matrix of the random noise is increased. For practical applications, one wants a variance of the noise to be as small as possible in order to

preserve the information contained in the samples

Algorithm 21 RHST-VQ-build($\mathcal{X}$, θ)

Input:

i) A set $\mathcal{X}$ of samples belonging to the d dimensional space

ii) θ, the minimum number of elements in the set so as to perform a split

Output:

i) a RHST

Method:

```
1  if |X| ≤ θ then
2     return
3  end if
4  Select randomly, and without replacement, d samples from X
5  Add random noise to the selected points (e.g. Gaussian, i.e. Z ~ N(0, σ))
6  Obtain the hyperplane h fitting the points previously computed
7  Create a new node and store h
8  Compute X⁻ and X⁺, the subsets of points X located in the lower and upper
   half-space of h, respectively
9  Create a left child by a recursive call using X⁻
10 Create a right child by a recursive call using X⁺
```

End Algorithm

The search mechanism is described in Algorithm 22. When a query point is presented to the network, the point is compared to the hyperplane H stored at the root, whence one determines the subregion of the dichotomization in which it falls. This operation is recursively executed and repeated until the number of codebook vectors in the respective subtree is zero. When this occurs, the BMU is chosen from among all the codebook vectors associated with its parent node in the RHST. If the search continues down to a leaf node, i.e., when one or more codebook vectors fall in the region induced by the leaf node, an exhaustive search is performed over the codebook vectors located in *that subregion* (only), and the most similar element is identified as the BMU. The complexity associated with the search operation is linear with respect to the height of the tree.

As the codebook vectors change their locations, we need to adapt the DS to support this migration. Our proposition utilizes a *deletion* of the codeword, considering its location *before* its migration, followed by an *insertion* of the codeword,

Algorithm 22 `RHST-VQ-search`($\mathbf{x}$,c)

Input:

i) $\mathbf{x}$, an input sample

ii) c, the current node being examined

Output:

i) The codebook vector identified as the BMU

Method:

```
1  if c is a leaf node then
2     return BMU from the list of codewords associated to c
3  end if
4  if the list of codewords associated with c is empty then
5     return BMU from the list of codewords associated to the parent of c
6  else
7     Evaluate x with respect to the hyperplane stored at c and determine in which
      half-space it is located
8     When x falls in H^-, recursively traverse the left child of c
9     When x falls in H^+, recursively traverse the right child of c
10 end if
```

End Algorithm

Algorithm 23 `RHST-VQ-insert`($\mathbf{w}$,c)

Input:

i) $\mathbf{w}$, a codebook vector

ii) c, the node of the RHST currently being examined

Output:

i) The codebook vector is located in the respective region of the RHST and a pointer to it is updated along the path up to the root

Method:

```
1 Add the current codebook vector to the list of codewords at node c
2 Evaluate w with respect to the hyperplane stored at node c, and determine in
  which half-space it is located
3 When w falls in H^-, recursively traverse the left child of c
4 When w falls in H^+, recursively traverse the right child of c
```

End Algorithm

which rather involves its location *after* the migration, i.e., once the VQ's update rule, defined in Equation (2.11), has been applied. For designing the delete and insert operations, we have modified the search operation, defined in Algorithm 22. Both operations are essentially a search, with the difference being that the respective list of codewords associated with each of the traversed nodes should be updated. In the case of the deletion (see Algorithm 24) as the algorithm traverse the tree, the codebook being deleted is removed from the list associated with the node currently being visited, which is a task that is recursively performed on each node until a leaf node is reached. The case for the insertion (see Algorithm 23) is analogous. When a codebook is inserted into the system, the codebook vector is added to the respective list of codewords.

By performing a deletion followed by an insertion of the same codebook, but at a different location, one ensures that the codebook vector falls into a unique region of the space at a given time. Additionally, after applying the update rule, it is possible that the codebook vector falls in a different sub-region when compared to the one it was in, prior to the update. The insertion mechanism takes care of this situation, since an entire re-positioning of the codebook vector is done from the root of the tree down to a single leaf. Similar to the search operation, the complexity of a deletion or an insertion is $O(h)$. Thus, we can take care of the migration of a codebook vector with an amount of work asymptotically equal to the time necessary to perform a search.

7.5 Experimental Results

7.5.1 2-dimensional data

Artificially-Generated Data

To better understand our submission, we conducted experiments so as to provide insight on the known behavior of RHSTs reported in [53]. These experiments involved randomly generated data sets in $\mathbb{R}^2$. The goal of the study was to test if we could successfully reproduce analogous results to the ones obtained in [53], and to yield a

Algorithm 24 `RHST-VQ-delete(`**w**,c`)`

Input:

i) **w**, a codebook vector

ii) c, the node of the RHST currently being examined

Output:

i) All references to **w** are erased from the lists of codewords in the RHST

Method:

1 Delete the reference to **w** from the list of codewords stored at node c

2 Evaluate **w** with respect to the hyperplane stored at node c, and determine in which half-space it is located

3 When **w** falls in H^-, recursively traverse the left child of c

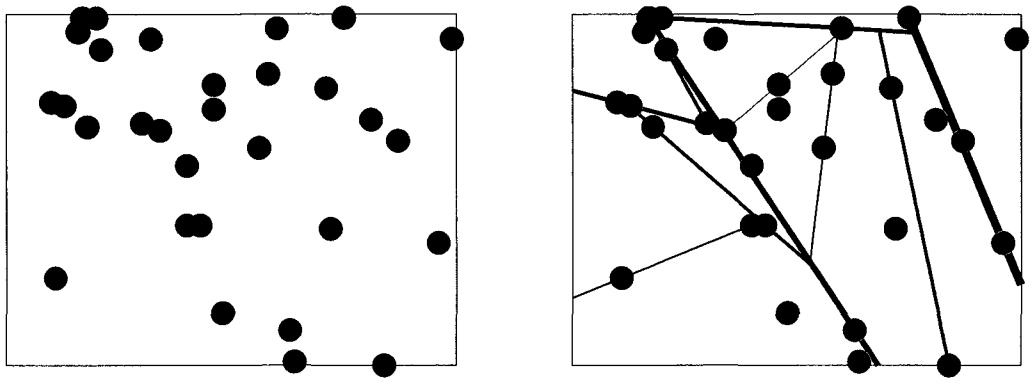
4 When **w** falls in H^+, recursively traverse the right child of c

End Algorithm

better insight in the inner-workings of the scheme

The first experiment involved random points from within a rectangle, as shown in Figure 7 3a The situation after the construction of the corresponding RHST is illustrated in Figure 7 3b The hyperplane associated with the root node of the tree is represented with bold line, and it is located at the top-right corner of the figure As the tree grows in depth, thiner lines are utilized to represent the hyperplanes Following the algorithm, each hyperplane is built by randomly selecting a pair of points Additionally, leaf nodes contain zero, one or two points One can observe that the hyperplanes divide the space in a manner which is analogous to the one reported in [53], and whose illustration is duplicated in Figure 7 1

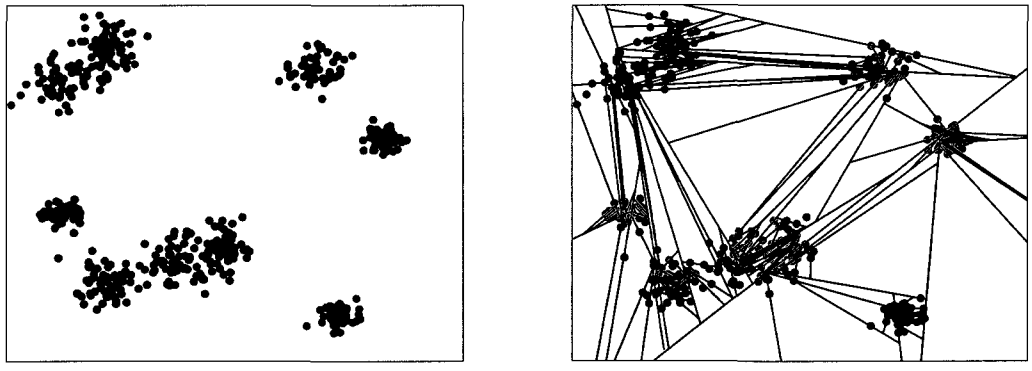
Additionally, we experimented with random data forming several ellipsoidal-shaped clusters, which is depicted in Figure 7 4a The points in the dataset belonged to two different categories shown in blue and red (or different levels of gray if printed in gray-scale printer mode), respectively The categories of the data points are unknown to the algorithm, as it essentially performs data clustering Figure 7 4b illustrates the RHST learned from the dataset The split threshold, n_0, was chosen adequately so as to mimic the behavior of the original RHST algorithm In this sense, a split will be performed if *two* or more points are available The resultant tree divides the plane in several sub-regions One can observe a more refined division of the space in those areas possessing a higher density of data points We also observe that in this



(a) (b)

Figure 7.3: Randomly selected points in $\mathbb{R}^2$ are utilized to show the behavior of the RHST algorithm.

case, some of the areas possessing no points are correctly identified, as in the case of the sub-region in the right-bottom corner of the figure.



(a) (b)

Figure 7.4: (a) Randomly generated data points forming ellipsoidal-shaped clusters. (b) The resultant RHST.

Real-World Data

We also conducted a series of experiments to observe the effect of the split threshold that determines the depth of the tree. An initial set of experiments considered the well-known Iris dataset [61], which is described in detail in Section 4.5.1. Figure 7.5

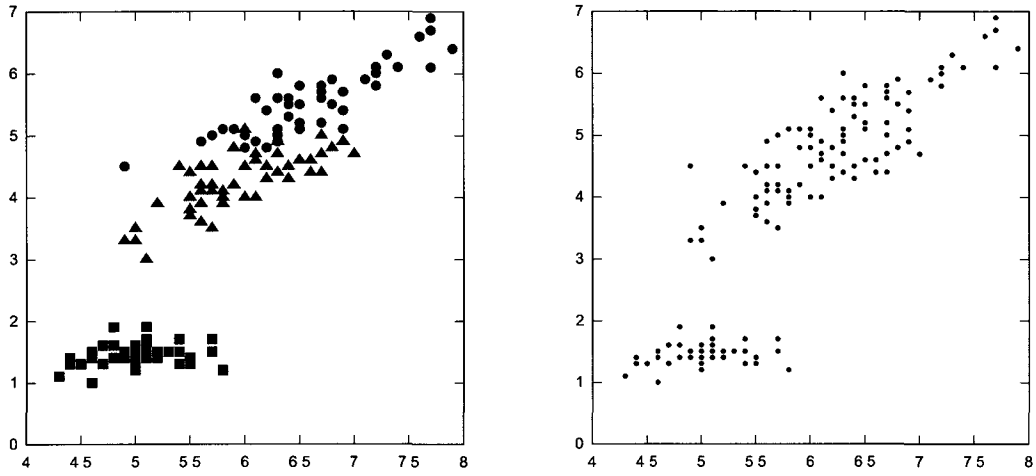


Figure 7.5: A projection of the Iris dataset in 2D. The X-axis corresponds to the attribute Sepal Length, and the Y-axis to the Petal Length, respectively. (a) The case when the categories are known. (b) The case when the classes are unknown.

illustrates the samples contained in the dataset, considering a 2D projection of two of the original attributes. To be more precise, the X-axis corresponds to the attribute Sepal Length, and the Y-axis to the Petal Length, respectively. The situation when the 3 classes are known is illustrated in Figure 7.5a. The flowers belonging to the Setosa category are represented with red squares, the Versicolor sub-family is represented with blue triangles, while the Virginica sub-specie is symbolized with green circles. Figure 7.5b depicts the situation when the class labels are not known, which is the case in which the items are represented with small red circles.

We first chose to use our proposed method to mimic the results generated by invoking the traditional VQ algorithm. We achieved this by setting the split threshold factor to be unity. In other words, by setting the split threshold to be equal to the size of the dataset, we hindered the growth of the depth of the RHST, and thus, one expects that the original data will fall into a single region. The situation after convergence is illustrated in Figure 7.6. The black rectangles show the location of the codebook vectors. Here, as expected, the samples are not divided at all, because the split threshold is high enough to impede *any* hyperplane dichotomization. As a consequence of the latter, *all* the codebook vectors fall into *a single* sub-region, where,

according to our proposed method, the search for the BMU becomes deterministic. Thus, the quantization scheme becomes equivalent to the traditional VQ algorithm. We observe that, after convergence, the locations of the codebook vectors is similar to the ones expected when executing the traditional VQ scheme. In the next section, we will explore this situation in a more detailed manner, when we experiment on several datasets possessing different dimensionalities, and compare the resultant QE to the ones obtained by invoking the traditional VQ scheme.

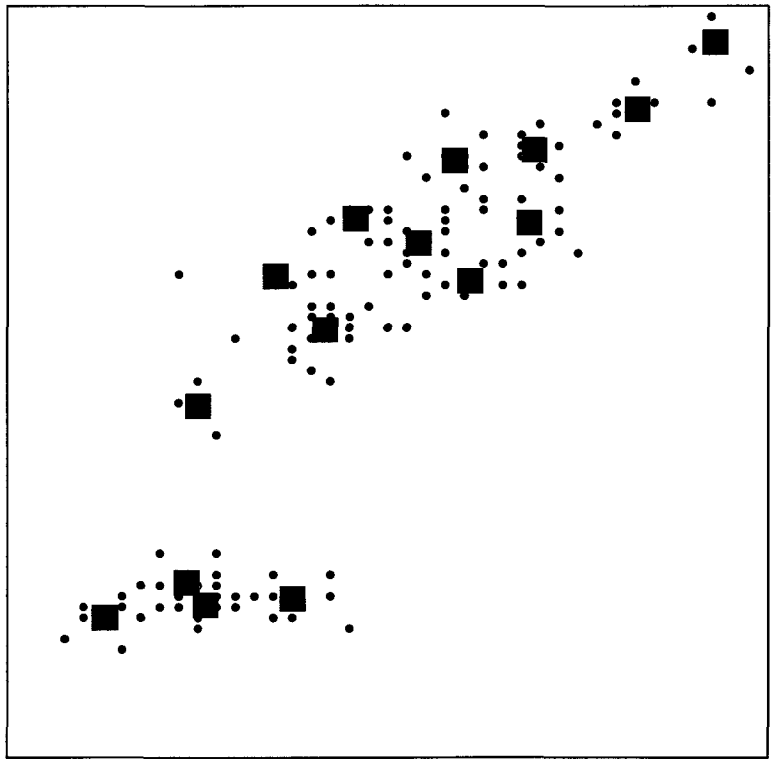


Figure 7.6: The extreme case when the split threshold factor is chosen so as to generate only a single node of the tree. As a consequence, the BMU search process becomes equivalent to one utilized by the traditional VQ algorithm.

The next step in our experiments consisted in modifying the split threshold systematically, so as to generate RHSTs with an increasing number of splits. Figure 7.7 illustrates the case when a single split is created (which was produced by using $n_0 = 0.9$). In this scenario, only a single hyperplane is created. In both half-spaces the quantization seems to occur in a manner similar to the one observed in Figure

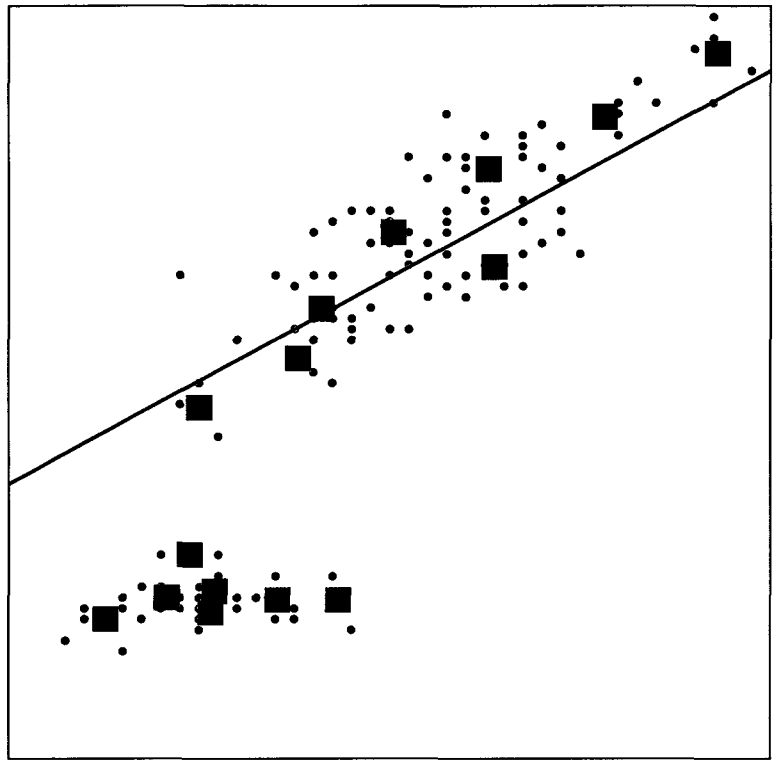


Figure 7.7: The case when $n_0 = 0.9$. The figure shows the situation when a single hyperplane divides the space. In both half-spaces the quantization seems to occur in a manner similar to the one observed in Figure 7.6.

7.6. The random nature of the algorithm places a fraction of the codebook vector in each side of the hyperplane, which are then affected by the VQ update rule. As the original position of codebook vectors are chosen by a uniform random sampling, we expect that the initial distribution of the codebook vector mimics the stochastic distribution of the whole sample set. The latter is an important property exploited by the algorithm, because we assume that it is very unlikely that a migration of codebook vectors occurs between the main branches of the tree.

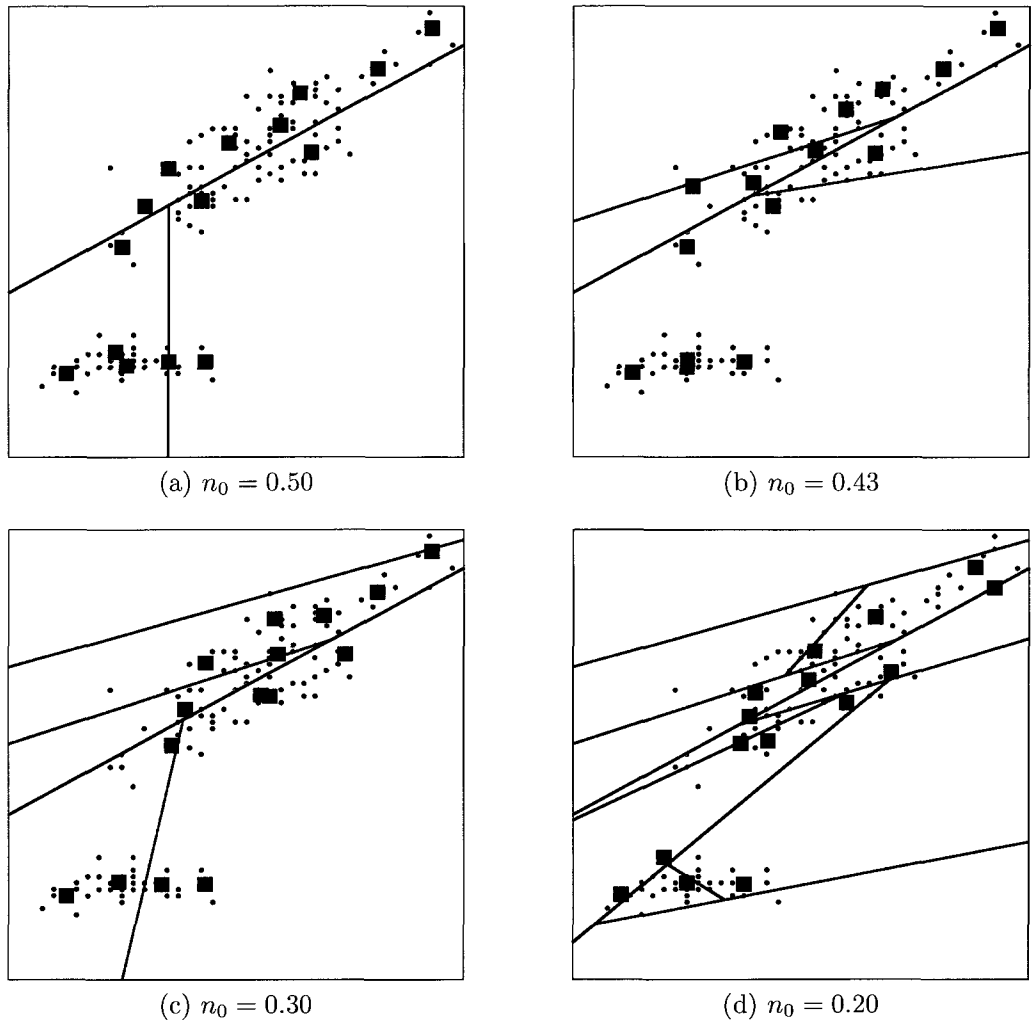
The resultant RHSTs obtained by systematically varying the value for the split threshold are illustrated in Figure 7.8. These different experiments utilized a split threshold of 0.5, 0.43, 0.3, and 0.2, respectively. In these experiments, different runs of the algorithm were produced with the same random seed. For the latter reason, the first dichotomization was the same in all the instances. However, further

dichotomizations vary across different experiments because of the effect of the split threshold currently chosen Figure 7 8a shows the case for $n_0 = 0\ 5$ Here, apart from the main dichotomization generated by the root node, one of the leaves of the RHST also splits We observe that in the second level of the tree, the quantization seems to occur in an analogous manner to the one observed in Figure 7 7, where only a single split was performed In other words, we observe that the quantization phenomenon occurs recursively in each subregion of the tree, as if a separate VQ algorithm was run by using the samples in each leaf of the RHST Figure 7 8b illustrates the case when both main branches of the tree are expanded (which is obtained by using $n_0 = 0\ 43$) In this case, again, an analogous behavior is observed, and, as a result, the codebook vectors seem to adequately represent the original distribution of points Figures 7 8c and 7 8d depict the situation when using a split threshold factor of 0 3 and 0 2, respectively These experiments produced HSTs of larger sizes, which again demonstrates that each sub-region is quantized in a VQ-like manner

7.5.2 Higher Dimensions

Different quantization schemes lead to maps with different quality One strategy by which we can estimate how good a quantization algorithm performs is by quantifying the "goodness" of the resulting mapping The literature reports a variety of measures for evaluating the quality of a map [17, 18, 85, 35, 42, 88, 103, 104] Indeed, one measurement that is extensively utilized is the Quantization Error (QE), which measures the average distance between each data vector and its BMU [8], and can be expressed, for example, as the Euclidean norm of the difference of the stimuli and the respective BMU [104]

The SOM, as well as many of its related variants, including the TTOSOM and the ones described in [5, 27, 55, 63, 69, 70, 71, 82, 105, 127, 134, 143, 156], utilize the VQ's update rule as part of their internal mechanism All these schemes possess, in common, the phenomenon that they represent a stochastic distribution using a finite set of representative points Even though the QE observed in VQ can be sometimes higher than the one observed by using the SOM [104], we believe that it is worth



(a) $n_0 = 0.50$ (b) $n_0 = 0.43$

(c) $n_0 = 0.30$ (d) $n_0 = 0.20$

Figure 7.8: RHSTs obtained for different numbers of splits. The results were obtained by using different values for the split threshold parameter.

measuring the QE when we use the HST proposed here, and comparing it to the classical VQ strategy defined in [80].

We tested our proposed methods with a set of real world datasets detailed in Section 4.5.1, which were extracted from the UCI Machine Learning repository [64]. These datasets are characterized by by real-valued attributes, whose dimensionality varies from four up to thirty dimensions. The six dataset are iris, wdbc, wine, yeast, wine-quality and glass, respectively.

	VQ	$n_0 = 1.0$	$n_0 = 0.9$	$n_0 = 0.8$	$n_0 = 0.4$	$n_0 = 0.3$	$n_0 = 0.2$	$n_0 = 0.1$
iris	3.41	3.38	3.36	3.91	3.79	4.84	9.55	11.32
wdbc	2829.82	2832.34	2832.34	2933.45	4447.39	4447.39	4875.46	13258.39
glass	10.42	10.43	10.42	12.86	12.90	12.90	13.01	19.65
wine	282 51	289.25	278.01	313.54	313.54	796.04	578.20	1202.53
yeast	14.75	14.85	14.68	15.31	16.38	16.85	19.52	21.84
wine_quality	613.79	613.79	613.79	703.51	832.71	832.71	1315.24	2053.35

Table 7.1: QE for different values of n_0.

We performed experiments on the six datasets, by systematically modifying the split factor by decreasing its value from 1.0 until 0.1. The resultant QEs for the different split factor values (columns) across all the datasets (rows) are summarized in Table 7.1. The results shown in Table 7.1 correspond to the average QE after 10 executions of the algorithm.

The first column of the table presents the resultant QEs when applying the traditional VQ scheme. As the reader can observe, these results are very close to those obtained by invoking our proposed algorithm and choosing $n_0 = 1.0$. This phenomenon is compatible to the one discussed in Section 7.5.1, and confirms our observations that, in general, defining a RHST with a single region will generate solutions similar to the ones obtained by executing the traditional VQ algorithm.

We also observe that in *all* six datasets there was no change in the QEs for the values of n_0 between 0.8 and 0.5. We explain this phenomenon by stating that there were no extra divisions of the space when using threshold factors in this range.

Moreover, in all the datasets, as the split threshold decreased, a degradation of the QE occurs. From this we conclude that even though more divisions of the space accelerate the BMU search process, too many sub-regions will lead to an inadequate quantization of the samples. In other words, we identify a trade-off between the speed of the BMU search, and the quality of the resultant quantization.

7.6 Conclusions

7.6.1 Concluding Remarks

In this chapter, we presented a novel DS that utilizes random hyperplanes trees to attempt to solve the BMU search problem in VQ Our proposed scheme was to, hopefully, achieve this without the necessity of checking *all* the neurons

In the course of our investigation we had to modify the original HST algorithm so as to accommodate the original philosophy presented in [53] so as to solve practical applications One of these situations occurred when the randomly selected points were linearly dependent, which yielded hyperplane equations possessing more than a single solution We also enhanced the DS to allow the handling of codebook vectors whose locations dynamically changed with time These migrations were efficiently updated in our proposed DS

Additionally, we showed that when one is willing to relax the requirement of identifying the true BMU, it is possible to reduce the expected running time and yet have a low degradation of the resultant QE The performance of our proposed algorithm has been measured in terms of the resultant QE, and these results have been compared to the scheme that finds the true BMU In this respect, the results obtained by using the VQ scheme are reproducible by defining our proposed strategy and using a "high split" threshold factor We also have shown that our modified version of the HST displays a behavior similar to the one observed by using the original HST defined in [53] when a very small split threshold is utilized

To conclude, on one hand, maintaining a sufficiently small HST results in a low degradation of the QE On the other hand, as we have observed, larger trees separate the sub-regions more accurately, making the search for the BMU faster, since more codebook vectors are discarded as the HST is traversed

7.6.2 Future Work

One possible improvement of the current algorithm might be to force the codebook vectors to initially be located in each of the sub-regions induced by the leaves of the

HST so as to ensure that every region is covered by at least one codebook vector This modification might alter the way in which each the codebook vector migrates from one region to another We believe that such a modification might result in an change in the QEs, and, from an optimistic perceptive, might improve the results presented here

The HSTs have shown to be successful in quickly identifying a BMU In spite of the positive properties possessed by the HSTs, they still need that the data belongs to the real domain From this perspective, the question of how we can use HSTs for performing VQ on datasets that contain nominal attributes, remains open

Another interesting avenue is to adapt the RHST so as to devise a classification algorithm We are currently investigating some modifications of the algorithm to accomplish this This new family of classifiers will possess the theoretical properties associated with HSTs, that might, in turn, lead to numerous interesting phenomena

Chapter 8

Conclusions

With the increasingly growing amount of data that humans beings have to deal with, the availability of mechanisms to discover knowledge in large datasets has become an important field in Computer Science (CS) At the same time, numerous organizations would like to take advantage of the volume of data they have about their businesses, such as clients, providers and products For these institutions, exploiting the data so as to discover patterns has been a priority, with the ultimate goal of obtaining interesting pieces of information that can be competitively used

This thesis has dealt with the issue of discovering knowledge in environments where the amount of data available is huge We have attempted to categorize data according to their similarities, and to generate partitions without the intervention of human beings Further, we have assumed that the true labels of the data items are unknown, or that due to financial constraints, it is unfeasible to tag *all* of them correctly

The focus of our research was concentrated on the Self-Organizing Map (SOM), which is a tool that has been applied to solve many practical problems [97, 142, 149] Even though the SOM possesses amazing properties, under certain constraints, it fails to represent the data correctly We have analyzed these known handicaps and proposed possible solutions to some of them

We shall now offer a summary of the work developed in this thesis, providing the overall conclusions for each chapter, individually

8.1 Summary of Work Done

Chapter 1: We began this manuscript by providing the main reasons that led us to study this field, and presented the motivations that inspired us, and the objectives that we intended to achieve

Chapter 2: In this chapter we covered the philosophical aspects concerning the paradigms of unsupervised learning and data clustering, including a categorization of the different schemes We also presented the most prominent algorithms reported in the state-of-the-art By describing the framework of an arbitrary Artificial Neural Network (ANN), we were able to explain its main concepts and definitions Special attention was placed on the SOM (which, as mentioned above, is the main focus of this work), and we thus offered an in-depth description of the algorithm, its main properties, as well as its applications and the known drawbacks reported in the literature We also surveyed the most prominent families of ANNs that take advantage of the original SOM strategy, sub-dividing them into two main categories, namely those which use a tree structure for their underlying topology, and those that do not Additionally, we also established a concise comparison between the different variants of the SOM reported in the literature

Chapter 3: In this chapter we proposed a schema called the Tree-based Topology Oriented SOM (TTOSOM) by which the user is able to impose an *arbitrary*, user-defined, tree-like topology onto the codebook vectors of a SOM This constraint leads to a neighborhood phenomenon based on the user-defined tree, and, as a result, the Bubble of Activity (BoA) becomes radically different from the ones studied in the prior art The map learned as a consequence of training with the TTOSOM is able to determine *both* the distribution of the data and its structured topology, interpreted through the perspective of the user-defined tree In addition, we showed that the TTOSOM possesses the ability to represent the original data set in multiple levels of granularity, and as per our results, this can be achieved without the necessity of computing the entire tree again Further, we discussed the capability of the TTOSOM to extract an skeleton, which is a "stick-like" representation of the image in a lower dimensional space Finally, we also showed, as a *prima facie*

case, how the TTOSOM can be utilized to perform and enhance pattern classification by using a scantily-structured representation of the original training sets – using a semi-supervised classifier All these properties have been confirmed by numerous experiments on a diversity of data sets

Chapter 4: The purpose of this chapter was to design and present an experimental analysis of a novel Pattern Recognition (PR) scheme based on the TTOSOM This scheme constructed a classifier by combining the information provided by labeled and unlabeled samples of the classes simultaneously Our experimental results showed that such a classifier possesses an improved classification accuracy in comparison to other Vector Quantization (VQ)-based classifiers Moreover, we have obtained accuracies that are comparable to the one attained by the state-of-the-art classifiers, even when the number of neurons utilized is only a small fraction of the cardinality of the dataset

Chapter 5: In this chapter, we proposed a framework that integrates the areas of Adaptive Data Structure (ADS) and the SOMs In particular, we designed a tree-based SOM that is able to self-adapt an underlying Binary Search Tree (BST) structure that constraints the neurons, and subsequently, restructures them using rotations that are performed conditionally In our model, the rotations lead to an asymptotically optimal neural tree with respect to the number of times that each neuron was identified as the Best Matching Unit (BMU) One of the main advantages of the algorithm is that the user is alleviated from the necessity of defining an initial tree configuration Instead, our proposed method, namely the TTOSOM with Conditional Rotations (TTOCONROT), infers the topological properties of the stochastic distribution, and at the same time, attempts to construct the best BST that represents the data set

Chapter 6: This chapter presented the design and experimental analysis of a PR scheme based on the TTOCONROT Analogous to the classifier developed in Chapter 4, the design invoked a tree-based neural network to learn the distribution of *all* the samples available, regardless of whether their labels were known or not, and then utilized a set of labeled samples (expected to be scarce), to categorize the regions of the feature space In particular, the proposed scheme constrained the neural tree as per the laws of the field of ADS These unique characteristics were analyzed and

contrasted to the results obtained by the TTOSOM-based and VQ-based classifiers described in Chapter 4

Chapter 7: This chapter considered the RHSTs to attempt to solve the BMU search problem in VQ According to our observations, in one hand, maintaining a sufficiently small RHST results in a low degradation of the QE On the other hand, as we aso have observed, larger trees separate the sub-regions more accurately, making the search for the BMU faster, since more codebook vectors are discarded as the HST is traversed

8.2 Future Work

Even though we believe that the work presented in this thesis is quite comprehensive, there are, undoubtedly, a number of avenues by which the research presented here can be extended The rest of this section concentrates on these potentially new research directions

- In Chapter 2 we presented a comprehensive survey of the SOM variants, and as mentioned earlier, we have submitted a paper summarizing the state-of-the-art in tree-based SOMs We foresee that the whole area of SOMs that use a graph-like topology, as described in Chapter 2, i e , those which are not of a tree or a grid-like topology, has yet to be fully surveyed and explored

- Chapter 3 advocated the design and analysis of the TTOSOM In that chapter, we empirically showed the advantages of using a tree-like topology as opposed to the standard grid-like topology utilized by the SOM We are currently investigating the other properties that the TTOSOM possesses, and hope to formalize it in the future In particular, we would like to quantify its topology preserving characteristics by invoking some of the sate-of-the-art methods such as the ones presented in [173]

- In Chapter 4 we had explained the design and implementation of TTOSOM-based *classifiers*, and experimentally showed their desirable classification capabilities The reader will recall that we recorded especially good results when the

size of the dataset was "large", and when few of the item were simultaneously associated with their true category This is indeed the situation in certain real-life domains Thus, a further area of research that we would like to develop, is the use of TTOSOM-based classifiers for predicting the quality of wine from its chemical attributes, as well as, identifying taste-trends based on geographic or ethnic origins

- As described in Chapter 3, Cheetham *et al* [38] presented an algorithm that involved Conditional Rotations for a BST (CONROT-BST), which is able to asymptotically infer the optimal tree that has the minimum Weighted Path Length (WPL) for the tree The optimization is achieved by performing conditional rotations These rotations depend only on local information and are performed in constant time The authors of [38] formally proved that the WPL of the entire tree decreased every time a rotation is performed In spite of all these advantages, the results obtained in [38] are limited only to the case when the tree is *binary*

  A k-ary tree is a tree containing at most k children per node k-ary trees are considered to be a generalization of multi-way trees which, by definition, are trees possessing an arbitrary number of children for each node We are currently investigating the generalization of the results obtained in [38] by defining k-ary conditional rotations The new CONROT-like heuristic will be applicable to trees with non-binary number of children We expect that the number of operations necessary to perform a promotion on this data structure will be constant, and that these operations will not be performed all the time Rather, a promotional operation will take place only if the WPL of the entire tree is guaranteed to decrease Analogous to the CONROT-BST, we seek a scheme that can verify if a promotion will lead to a superior tree configuration, i e , one with a lower WPL, and we hope to achieve this by merely checking local information Additionally, contrary to many rotational strategies that move the most recently accessed node all along the path up to the root (e g , [166]), we expect that the proposed rotation will only effect the immediate vicinity of the

node involved, while the rest of the tree will remain unmodified

- A potential generalization of the CONROT-BST algorithm, from the perspective of our research, is a direct consequence of the k-ary Conditional Rotations (CONROT) solution Indeed, we believe that the TTOCONROT can be enhanced by relaxing the constraint of operating with a binary tree Put in a nutshell, a possible integration of the TTOSOM with the CONROT for k-ary trees (say, k-CONROT), instead of the traditional CONROT-BST, could imply that the tree, could now be able to modify itself so as to build the optimal *k-ary* tree that best suits the data distribution based on the access probabilities We believe that if we succeed in this venture, it would constitute a significant but natural extension of the work that is currently the state-of-the-art

- Our results on Chapter 7, showed how a Random Hyperplane Search Tree (RHST) can be advantageously utilized for accelerating the time required for identifying the so-called BMU One research direction to expand this work could be to improve the way in which the hyperplanes estimate the BMU, by, for example, defining a general method that can be applied to any type of SOM-based Neural Network (NN), and not just the TTOSOM Other improvements may involve the use of the RHST to directly design a classifier, i e , independent from the TTOSOM philosophy, by guiding it with additional information extracted from the dataset

Bibliography

[1] M Adelson-Velskıı and E Landıs, M An algorıthm for the organızatıon of ınformatıon *Sov Math DokL*, 3 1259–1262, 1962

[2] D W Aha, D Kıbler, and M K Albert Instance-based learnıng algorıthms *Machıne Learnıng*, 6 37–66, 1991

[3] A Ahmadı, S Omatu, T Fujınaka, and T Kosaka Improvement of relıabılıty ın banknote classıfıcatıon usıng reject optıon and local pca *Informatıon Scıences*, 168(1-4) 277 – 293, 2004

[4] Alfred V Aho and John E Hopcroft *The Desıgn and Analysıs of Computer Algorıthms* Addıson-Wesley Longman Publıshıng Co , Inc , Boston, MA, USA, 1st edıtıon, 1974

[5] D Alahakoon, S K Halgamuge, and B Srınıvasan Dynamıc self-organızıng maps wıth controlled growth for knowledge dıscovery *IEEE Transactıons on Neural Networks*, 11(3) 601–614, 2000

[6] B Allen and I Munro Self-organızıng bınary search trees *J ACM*, 25(4) 526–535, 1978

[7] A C Anderson and K S Fu Desıgn and development of a lınear bınary tree classıfier for leukocytes Technıcal report, Purdue Unıversıty, West Lafayette, IN, 1979

[8] E. Arsuaga Uriarte and F. Díaz Martín. Topology preservation in SOM. *International Journal of Applied Mathematics and Computer Sciences*, 1(1):19–22, 2005.

[9] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, New York, NY, USA, 2006. ACM Press

[10] S. Arya and D M. Mount. Algorithms for fast vector quantization. In J. A. Storer and M. Cohn, editors, *Proc. of the Data Compression Conference (DCC'93)*, pages 381–390. IEEE Press, 1993.

[11] C. A. Astudillo and B. J. Oommen. A novel self organizing map which utilizes imposed tree-based topologies In *6th International Conference on Computer Recognition Systems*, volume 57, pages 169–178, 2009.

[12] C. A Astudillo and B. J. Oommen. On using adaptive binary search trees to enhance self organizing maps. In A. Nicholson and X. Li, editors, *22nd Australasian Joint Conference on Artificial Intelligence (AI 2009)*, pages 199–209, 2009.

[13] C. A. Astudillo and B. J. Oommen. Topology-oriented self-organizing maps A survey. *currently being reviewed*, 2010.

[14] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[15] V. Barnett and T. Lewis. *Outliers in statistical data.* Wiley, third edition, 1994

[16] L.A. Bartolucci, P.H. Swain, and Chialin Wu. Selective radiant temperature mapping using a layered classifier. *Geoscience Electronics, IEEE Transactions on*, 14(2):101 –106, 1976.

[17] H. U. Bauer, M. Herrmann, and T. Villmann. Neural maps and topographic vector quantization. *Neural Networks*, 12(4-5):659 – 676, 1999.

[18] H. U. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *Neural Networks*, 3(4):570–579, July 1992

[19] I. Ben-Gal. *Encyclopedia of Statistics in Quality and Reliability*, chapter Bayesian Networks. John Wiley & Sons, Ltd, 2007.

[20] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975

[21] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2 edition, July 2008. hardcover.

[22] J C. Bezdek. *Fuzzy Mathematics in Pattern Classification*. PhD thesis, Cornell University, Applied Mathematics Center, Ithaca, New York, 1973.

[23] J. C. Bezdek. *Pattern Recognition With Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[24] J R. Bitner. Heuristics that dynamically organize data structures. *SIAM J. Comput.*, 8:82–110, 1979.

[25] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[26] P. E. Black. "multiway tree". in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology, 27 October 2005. (accessed 15 December 2010) http://xw2k.nist.gov/dads/HTML/multiwaytree.html.

[27] J. Blackmore. Visualizing high-dimensional structure with the incremental grid growing neural network. Master's thesis, University of Texas at Austin, 1995.

[28] J. Blackmore and R. Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proc. ICNN'93, International Conference on Neural Networks*, volume I, pages 450–455, Piscataway, NJ, 1993. IEEE Service Center.

[29] H. D. Block. The perceptron: A model for brain functioning. *Rev. Mod. Phys.*, 34(1):123–135, Jan 1962.

[30] E. Bonabeau. Graph multidimensional scaling with self-organizing maps. *Information Sciences*, 143(1-4):159 – 180, 2002.

[31] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms *Pattern Recognition*, 30(7):1145–1159, 1997.

[32] M Bramer. Pre-pruning classification trees to reduce overfitting in noisy domains. In Hujun Yin, Nigel Allinson, Richard Freeman, John Keane, and Simon Hubbard, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2002*, volume 2412 of *Lecture Notes in Computer Science*, pages 247–258. Springer Berlin / Heidelberg, 2002.

[33] L. A. Breslow and D. W. Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(01):1–40, 1997.

[34] F. Buckley and F. Harary. *Distance in Graphs.* Addison-Wesley, Redwood City, CA, 1990.

[35] M. Budinich. On the ordering conditions for self-organizing maps. *Neural Computation*, 7(2):284–289, 1995.

[36] A Buzo, Jr. Gray, A., R. Gray, and J. Markel. Speech coding based upon vector quantization. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(5):562 – 574, October 1980.

[37] G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, 1988.

[38] R. P. Cheetham, B J. Oommen, and D. T. H. Ng. Adaptive structuring of binary search trees using conditional rotations. *IEEE Trans. on Knowl. and Data Eng.*, 5(4):695–704, 1993.

[39] S.X. Chen, F.W. Li, and W.L. Zhu. Fast searching algorithm for vector quantisation based on features of vector and subvector. *Image Processing, IET*, 2(6):275–285, 2008.

[40] D.-Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham. Fast search algorithms for vector quantization and pattern matching. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '84.*, volume 9, pages 372 – 375, March 1984.

[41] K. L. Clarkson A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, August 1988.

[42] P. L. Conti and L. De Giovanni. On the mathematical treatment of self organization· extension of some classical results. In *Artificial Neural Networks - ICANN 1991, International Conference*, volume 2, pages 1089–1812, 1991.

[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein *Introduction to Algorithms, Second Edition.* McGraw-Hill Science/Engineering/Math, July 2001.

[44] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547 – 553, 2009. Smart Business Networks: Concepts and Empirical Evidence.

[45] R. Dara, S.C. Kremer, and D.A. Stacey. Clustering unlabeled data with SOMs improves classification of labeled real-world data. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2237 –2242, 2002.

[46] A. Datta, S. M. Parui, and B. B. Chaudhuri. Skeletal shape extraction from dot patterns by self-organization. *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, 4:80–84 vol.4, Aug 1996.

[47] A. Demiriz, K. Bennett, and M.J. Embrechts. Semi-supervised clustering using genetic algorithms. In *Artificial Neural Networks in Engineering (ANNIE-99)*, pages 809–814, 1999.

[48] J. Demšar. Statistical comparisons of classifiers over multiple data sets *J. Mach Learn. Res* , 7:1–30, 2006.

[49] R. Descartes. *Discourse on Method, Optics, Geometry, and Meteorology.* Hackett Publishing, 2001. Oscamp, P. J. (trans).

[50] C. J. Deschenes and J. Noonan. Fuzzy kohonen network for the classification of transients using the wavelet transform for feature extraction. *Information Sciences*, 87(4):247 – 266, 1995.

[51] D DeSieno. Adding a conscience to competitive learning. *IEEE International Conference on Neural Networks*, 1:117–124, July 1988.

[52] L. Devroye. A note on the height of binary search trees. *J. ACM*, 33:489–498, May 1986.

[53] Luc Devroye, James King, and Colin McDiarmid. Random hyperplane search trees. *SIAM J. Comput.*, 38(6):2411–2425, 2009.

[54] M. Dittenbach, D. Merkl, and A Rauber. The growing hierarchical self-organizing map In *Neural Networks, 2000 IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 6, pages 15–19 vol.6, 2000

[55] J. Dopazo. *Fundamentals of Data Mining in Genomics and Proteomics*, chapter Clustering – Class Discovery in the Post-Genomic Era, pages 123–148. Springer US, 2007.

[56] J. Dopazo and J. M. Carazo. Phylogenetic reconstruction using an unsupervised growing neural network that adopts the topology of a phylogenetic tree. *Journal of Molecular Evolution*, 44(2):226–233, February 1997.

[57] R. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)* Wiley-Interscience, 2000.

[58] G. Exoo. A euclidean ramsey problem. *Discrete & Computational Geometry*, 29(2):223–227, 2003.

[59] T. Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006

[60] U. Fayyad, G. G. Grinstein, and A. Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[61] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[62] N. Forbes. *Imitation of Life: How Biology Is Inspiring Computing*. MIT Press, 2004.

[63] A. Forti and G. L. Foresti. Growing Hierarchical Tree SOM: An unsupervised neural network with dynamic topology. *Neural Networks*, 19(10):1568–1580, 2006.

[64] A. Frank and A. Asuncion. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2010.

[65] J. Friedman. A tree-structured approach to nonparametric multiple regression. In Th. Gasser and M. Rosenblatt, editors, *Smoothing Techniques for Curve Estimation*, volume 757 of *Lecture Notes in Mathematics*, pages 5–22. Springer Berlin / Heidelberg, 1979

[66] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, September 1977.

[67] M. Friedman and A Kandel. *Introduction to Pattern Recognition. Statistical, Structural, Neural and Fuzzy Logic Approaches*. Imperical College Press, 1999.

[68] B. Fritzke. Unsupervised clustering with growing cell structures. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume ii, pages 531–536 vol.2, Jul 1991.

[69] B. Fritzke Growing Cell Structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[70] B. Fritzke. Growing Grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, 1995.

[71] B. Fritzke. A growing neural gas network learns topologies In G Tesauro, D S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632, Cambridge MA, 1995. MIT Press.

[72] K. Fukunaga. *Introduction to statistical pattern recognition (2nd ed.)* Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[73] B. Gabrys and L. Petrakieva. Combining labelled and unlabelled data in the design of pattern classification systems. *International Journal of Approximate Reasoning*, 35(3):251 – 273, 2004. Integration of Methods and Hybrid Systems.

[74] S. I. Gallant. Perceptron-based learning algorithms. *Neural Networks, IEEE Transactions on*, 1(2):179–191, Jun 1990.

[75] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional, 1994.

[76] S.B. Gelfand, C.S. Ravishankar, and E.J. Delp. An iterative growing and pruning algorithm for classification tree design. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(2):163 –174, February 1991.

[77] J. E. Gentle. *Numerical Linear Algebra for Applications in Statistics*, chapter Cholesky Factorization. 3.2 2, pages 93–95. Springer-Verlag, Berlin, 1998.

[78] A. Gersho and R M. Gray. *Vector quantization and signal compression.* Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[79] L. Gordon and R. A. Olshen. Asymptotically efficient solutions to the classification problem. *Annals of Statistics*, 6:515–533, 1978.

[80] R. Gray. Vector quantization. *ASSP Magazine, IEEE*, 1(2):4 – 29, April 1984.

[81] D. Greene, P. Cunningham, and R. Mayer. *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval (Cognitive Technologies)*, chapter Unsupervised Learning and Clustering, pages 51–90. Springer Berlin Heidelberg, 2008.

[82] L. Guan. Self-organizing trees and forests· A powerful tool in pattern clustering and recognition. In *Image Analysis and Recognition, Third International Conference, ICIAR 2006, Póvoa de Varzim, Portugal, September 18-20, 2006, Proceedings, Part I*, pages I: 1–14, 2006.

[83] D. E Gustafson, S. Gelfand, and S. K. Mitter. A nonparametric multiclass partitioning method for classification. In *Proceedings of the Fifth International Conference on Pattern Recognition*, pages 654–659, Miami, FL, 1980. International Association for Pattern Recognition.

[84] J. Handl. Ant-based methods for tasks of clustering and topographic mapping: extensions, analysis and comparison with alternative methods. Masters thesis. Master's thesis, Chair of Artificial Intelligence, University of Erlangen-Nuremberg, Germany, November 2003.

[85] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3rd edition edition, 2008

[86] D. O Hebb. *The Organization of Behaviour*. John Wiley & Sons, New York, 1949.

[87] P. Horton and K. Nakai. A probabilistic classification system for predicting the cellular localization sites of proteins. In *Proceedings of the 4th International Conference on Intelligent Systems for Molecular Biology (ISMB96)*, volume 4, pages 109–115, 1996.

[88] G. Huang, H. A. Babri, and H. Li. Ordering of self-organizing maps in multidimensional cases. *Neural Computation*, 10:19–24, 1998.

[89] A. K. Jain and R. C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[90] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[91] H Jin, W.-H. Shum, and Leung K -S and Wong M.-L. Expanding self-organizing map for data visualization and cluster analysis. *Information Sciences*, 163(1-3):157 - 173, 2004. Soft Computing Data Mining

[92] I. T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, 1986.

[93] M. Tim Jones. *Artificial Intelligence: A Systems Approach.* Infinity Science Press, 2007.

[94] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial intelligence Research*, 4:237–285, 1996.

[95] H. Kaplan *Handbook of Data Structures and Applications*, chapter 31: Persistent Data Structures, pages 31.1 - 31.26. Chapman and Hall/CRC, 2004.

[96] Wataru Kasai, Yutaro Tobe, and Osamu Hasegawa. A fast bmu search for support vector machine. In *ICANN '09: Proceedings of the 19th International Conference on Artificial Neural Networks*, pages 864–873, Berlin, Heidelberg, 2009. Springer-Verlag.

[97] Samuel Kaski, Jari Kangas, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 1981–1997. *Neural Computing Surveys*, 1:102–350, 1998.

[98] I. Katsavounidis, C.-C.J. Kuo, and Z. Zhang. Fast tree-structured nearest neighbor encoding for vector quantization. *Image Processing, IEEE Transactions on*, 5(2):398 –404, February 1996.

[99] K. Kiviluoto. Topology preservation in self-organizing maps. In P IEEE Neural Networks Council, editor, *Proceedings of International Conference on Neural Networks, ICNN'96*, volume 1, pages 294–299, New Jersey, USA, 1996.

[100] D. E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[101] R. Kohavi and F. Provost. Glossary of terms. *Machine Learning*, 30:271–274, 1998.

[102] T. Kohonen. Improved versions of learning vector quantization In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, volume 1, pages 545–550, June 1990.

[103] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[104] T. Kohonen, I. Nieminen, and T. Honkela. On the quantization error in som vs. vq: A critical and systematic study. In Jos Prncipe and Risto Miikkulainen, editors, *Advances in Self-Organizing Maps*, volume 5629 of *Lecture Notes in Computer Science*, pages 133–144. Springer Berlin / Heidelberg, 2009.

[105] P. Koikkalainen and E. Oja. Self-organizing hierarchical feature maps. *IJCNN International Joint Conference on Neural Networks*, 2:279–284, June 1990.

[106] A. N. Kolmogorov. On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition. *Doklady Akademiia Nauk SSSR*, 114(5):953–956, 1957.

[107] S. Kotsiantis, I. Zaharakis, and P. Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26 159–190, 2006. 10.1007/s10462-007-9052-3.

[108] Sotiris B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica (Slovenia)*, 31(3):249–268, 2007.

[109] N. Kubota. Computational intelligence for structured learning of a partner robot based on imitation. *Information Sciences*, 171(4):403 – 429, 2005. Intelligent Embedded Agents.

[110] K. Lagus, S. Kaski, and T. Kohonen. Mining massive document collections by the websom method. *Information Sciences*, 163(1-3):135 – 156, 2004. Soft Computing Data Mining.

[111] A. Laha, B. Chanda, and N. Pal. Fast codebook searching in a som-based vector quantizer for image compression. *Signal, Image and Video Processing*, 2:39–49, 2008. 10.1007/s11760-007-0034-3.

[112] J.Z.C. Lai and Y.-C. Liaw. Fast-searching algorithm for vector quantization using projection and triangular inequality. *Image Processing, IEEE Transactions on*, 13(12):1554 –1558, dec. 2004.

[113] T. W. H. Lai. *Efficient maintenance of binary search trees.* PhD thesis, University of Waterloo, Waterloo, Ont., Canada, 1990.

[114] J. Lampinen and E. Oja. Fast self-organization by the probing algorithm. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 503 –507 vol.2, June 1989.

[115] S. Lawrence, C.L. Giles, Chung Tsoi Ah, and A.D. Back. Face recognition: a convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, Jan 1997.

[116] S. Lazebnik and M. Raginsky. Supervised learning of quantizer codebooks by information loss minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(7):1294 –1309, July 2009.

[117] R. Y. Li, G. L. Lebby, and H. Si. A modified approach for constructing the self-organized layer in a multilayer feedforward neural network. *Information Sciences*, 98(1-4):69 – 81, 1997.

[118] Y.K. Lin and K.S. Fu. Automatic classification of cervical cells using a binary tree classifier. *Pattern Recognition*, 16(1):69–80, 1983.

[119] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, 28(1):84–95, Jan 1980.

[120] P. Lingras, M. Hogo. M. Snorek, and C. West Temporal analysis of clusters of supermarket customers: conventional versus interval set approach. *Information Sciences*, 172(1-2):215 – 240, 2005

[121] Z.M. Lu, S.C. Chu, and K.C. Huang Equal-average equalvariance equal-norm nearest neighbor codeword search algorithm based on ordered hadamard transform. *International Journal of Innovative Computing, Information and Control*, 1(1):35–41, March 2005.

[122] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability*, pages 281–297, 1967.

[123] P. C. Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):4955, 1936.

[124] C. D. Manning, P Raghavan, and H Schtze. *An Introduction to Information Retrieval* Cambridge University Press, Cambridge, England, 2009.

[125] M. Marcus and H Minc. *Introduction to Linear Algebra.* Dover Publications, New York, 1988.

[126] J. Martín-Herrero, M. Ferreiro-Armán, and J. L. Alba-Castro. Grading textured surfaces with automated soft clustering in a supervised som. In Aurélio C. Campilho and Mohamed S. Kamel, editors, *ICIAR (2)*, volume 3212 of *Lecture Notes in Computer Science*, pages 323–330. Springer, 2004.

[127] M. Martinetz and K. J. Schulten. A "neural-gas" network learns topologies. In *in Proceedings of International Conference on Artcial Neural Networks*, volume I, pages 397–402, North-Holland, Amsterdam, 1991.

[128] T Martinetz and K Schulten Topology representing networks *Neural Netw*, 7 507–522, March 1994

[129] Warren McCulloch and Walter Pitts A logical calculus of the ideas immanent in nervous activity *Bulletin of Mathematical Biophysics*, 7 115 – 133, 1943

[130] Warren McCulloch and Walter Pitts A logical calculus of the ideas immanent in nervous activity *Bulletin of Mathematical Biology*, 52(1) 99–115, January 1990

[131] K Mehlhorn Dynamic binary search *SIAM Journal on Computing*, 8(2) 175–198, 1979

[132] W S Meisel and D A Michalopoulos A partitioning algorithm with application in pattern classification and the optimization of decision trees *Computers, IEEE Transactions on*, C-22(1) 93–103, 1973

[133] D Merkl, S Hui-He, M Dittenbach, and A Rauber Adaptive hierarchical incremental grid growing An architecture for high-dimensional data visualization In *In Proceedings of the 4th Workshop on Self-Organizing Maps, Advances in Self-Organizing Maps*, pages 293–298, 2003

[134] R Miikkulainen Script recognition with hierarchical feature maps *Connection Science*, 2(1&2) 83–101, 1990

[135] M Minsky and S Papert *Perceptrons An Introduction to Computational Geometry* The MIT Press, 1969

[136] Tom Mitchell *Machine Learning* McGraw Hill, 1997

[137] Riichiro Mizoguchi, Makoto Kizawa, and Masamichi Shimura Piecewise linear discriminant functions in pattern recognition *Systems-Comput -Controls*, 8(1) 62–68 (1978), 1977

[138] S K Murthy Automatic construction of decision trees from data A multi-disciplinary survey *Data Mining and Knowledge Discovery*, 2 345–389, 1998

[139] Albert B Novikoff On convergence proofs for perceptrons In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962

[140] M S Obaidat, H Khalid, and B Sadoun Ultrasonic transducer characterization by neural networks *Information Sciences*, 107(1-4) 195 – 215, 1998

[141] R L Ogniewicz and O Kubler Hierarchic voronoi skeletons *Pattern Recognition*, 28(3) 343–359, 1995

[142] Merja Oja, Samuel Kaski, and Teuvo Kohonen Bibliography of self-organizing map (SOM) papers 1998-2001 addendum *Neural Computing Surveys*, 3 1–156, 2003

[143] J Pakkanen, J Iivarinen, and E Oja The Evolving Tree — a novel self-organizing network for data analysis *Neural Processing Letters*, 20(3) 199–211, December 2004

[144] S K Pal and S Mitra Fuzzy versions of kohonen's net and mlp-based classification Performance evaluation for certain nonconvex decision regions *Information Sciences*, 76(3-4) 297 – 337, 1994

[145] E Pampalk, G Widmer, and A Chan A new approach to hierarchical clustering and structuring of data with self-organizing maps *Intell Data Anal*, 8(2) 131–149, 2004

[146] J -S Pan, Z -M Lu, and S -H Sun An efficient encoding algorithm for vector quantization based on subvector technique *Image Processing, IEEE Transactions on*, 12(3) 265–270, 2003

[147] G Patan and M Russo Distributed unsupervised learning using the multisoft machine *Information Sciences*, 143(1-4) 181 – 196, 2002

[148] G Peano Sur une courbe, qui remplit toute une aire plane *Mathematische Annalen*, 36(1) 157–160, 1890

[149] M. Pöllä, T. Honkela, and T. Kohonen. Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. *Neural Computing Surveys*, forthcoming, 2007.

[150] G. Pölzlbauer. Survey and comparison of quality measures for self-organizing maps. In Ján Paralič, Georg Pölzlbauer, and Andreas Rauber, editors, *Proceedings of the Fifth Workshop on Data Analysis (WDA'04)*, pages 67–82, Sliezsky dom, Vysoké Tatry, Slovakia, June 24–27 2004. Elfa Academic Press.

[151] J.R. Quinlan. *C4 5: Programs for machine learning*. Morgan Kaufmann, San Francisco, 1993.

[152] J. Rahmel. SplitNet: learning of tree structured Kohonen chains. In *Neural Networks, 1996., IEEE International Conference on*, volume 2, pages 1221 – 1226 vol.2, June 1996.

[153] J Rahmel, C. Blum, and P. Hahn. On the role of hierarchy for neural network interpretation. In *IJCAI'97: Proceedings of the Fifteenth international joint conference on Artifical intelligence*, pages 1072–1077, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[154] V. Ramasubramanian and K K. Paliwal. Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *Signal Processing, IEEE Transactions on*, 40(3):518 –531, March 1992.

[155] S. Ramón y Cajal. *Histologie du Systéms Nerveux de l'homme et des vertébrés*. Maloine, Paros, 1911.

[156] A. Rauber, D. Merkl, and M. Dittenbach. The Growing Hierarchical Self-Organizing Map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, 2002.

[157] Bruce Reed. The height of a random binary search tree. *J. ACM*, 50:306–332, May 2003.

[158] R. Rojas. *Neural networks: a systematic introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[159] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[160] Frank Rosenblatt. *Principles of Neurodynamics.* Spartan, New York, 1962.

[161] D. Roverso. Soft computing tools for transient classification. *Information Sciences*, 127(3-4):137 – 156, 2000. Intelligent Manufacturing and Fault Diagnosis. (II). Soft computing approaches to fault diagnosis

[162] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput Surv.*, 16:187–260, June 1984.

[163] U. Seiffer and B. Michaelis. Estimating motion parameters with three-dimensional self-organizing maps. *Information Sciences*, 101(3-4) 187 – 201, 1997. Advanced Neuro-Fuzzy Techniques and Their Applications.

[164] K. Shanmukh, C.N.S. Ganesh Murthy, and Y.V. Venkatesh. Applications of self-organization networks spatially isomorphic to patterns. *Information Sciences*, 114(1-4):23–39, 1999

[165] R. Singh, V. Cherkassky, and N. Papanikolopoulos. Self-Organizing Maps for the skeletonization of sparse shapes. *Neural Networks, IEEE Transactions on*, 11(1):241–248, Jan 2000.

[166] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[167] R F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(4):579–589, 1991.

[168] S.C. Tai, C.C. Lai, and Y.C. Lin. Two fast nearest neighbor searching algorithms for image vector quantization. *Communications, IEEE Transactions on*, 44(12):1623–1628, December 1996.

[169] A. Ultsch and F. Rske. Self-organizing feature maps predicting sea levels. *Information Sciences*, 144(1-4):91 – 125, 2002.

[170] B. Vandeginste. PARVUS: An extendable package of programs for data exploration, classification and correlation, M. Forina, R. Leardi, C. Armanino and S. Lanteri, Elsevier, Amsterdam, 1988, Price: US $645 ISBN 0-444-43012-1. *Journal of Chemometrics*, 4(2):191–193, 1990.

[171] J. Venna and S. Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *ICANN*, volume 2130 of *Lecture Notes in Computer Science*, pages 485–491. Springer, 2001.

[172] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *8th International Workshop on Artificial Neural Networks, IWANN 2005*, pages 758–770, 2005.

[173] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz. Topology preservation in self-organizing feature maps: exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266, Mar 1997.

[174] Jr. Ward, Joe H. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[175] A C Yao and F F Yao. A general approach to d-dimensional geometric queries. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 163–168, New York, NY, USA, 1985. ACM.

[176] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005