CONCEPT-LEARNING IN THE ABSENCE OF COUNTER-EXAMPLES: AN AUTOASSOCIATION-BASED APPROACH TO CLASSIFICATION

BY NATHALIE JAPKOWICZ

A dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Computer Science

Written under the direction of
Stephen José Hanson and Casimir Kulikowski
and approved by

____

____

____

____

____

New Brunswick, New Jersey
October, 1999

ABSTRACT OF THE DISSERTATION

Concept-Learning in the Absence of Counter-Examples: An Autoassociation-Based Approach to Classification

by Nathalie Japkowicz
Dissertation Directors: Stephen José Hanson and Casimir Kulikowski

The overwhelming majority of research currently pursued within the framework of concept-learning concentrates on *discrimination*-based learning, an inductive learning paradigm that relies on both examples and counter-examples of the concept. This emphasis, however, can present a practical problem: there are real-world engineering problems for which counter-examples are both scarce and difficult to gather. For these problems, *recognition*-based learning systems are much more appropriate because they do not use counter-examples in the concept-learning phase.

The purpose of this dissertation is to analyze a connectionist recognition-based learning system—*autoassociation*-based classification—and answer the following questions raised by a preliminary comparison of the autoassociator and its discrimination counterpart, the Multi-Layer Perceptron (MLP), on three real-world domains:

- What features of the autoassociator make it capable of performing classification in the absence of counter-examples?
- What causes the autoassociator to be significantly more efficient than MLP in certain domains?
- What domain characteristics cause the autoassociator to be more accurate than MLP and MLP to be more accurate than the autoassociator?

The dissertation concludes that 1) autoassociation-based classification is possible in a particular class of practical domains called *non-linear* and *multi-modal* because the autoassociator uses a multi-modal specialization bias to compensate

for the absence of counter-examples. This bias can be controlled by varying the capacity of the autoassociator. 2) The difference in efficiency between the autoassociator and MLP observed on this class of domains is caused by the fact that the autoassociator uses a (fast) *data-driven* generalization strategy whereas MLP has recourse to a (slow) *hypothesis-driven* one, despite the fact that the two systems are both trained by the backpropagation procedure. 3) The autoassociator classifies more accurately than MLP domains requiring particularly *strong* specialization biases caused by the *counter-conceptual* class or particularly *weak* specialization biases caused by the *conceptual* class. However, MLP is more accurate than the autoassociator on domains requiring particularly *strong* specialization biases caused by the *conceptual* class.

The results of this study thus suggest that recognition-based systems, which are often dismissed in favor of discrimination-based ones in the context of concept-learning, may present an interesting array of classification strengths.

Acknowledgements

I would like to thank all the people who, in one way or another, helped me complete my Ph.D.

First, I am grateful to my advisors, Stephen José Hanson and Casimir A. Kulikowski, for their guidance and advice during the development of the research presented in this dissertation. I appreciated Steve's willingness to listen to my ideas and help me transform them into interesting research questions and I am thankful to Professor Kulikowski for patiently helping me turn them into a coherent whole.

I would also like to thank my other committee members. In particular, I am grateful to Mark A. Gluck for offering me the chance to work on an exciting problem—which subsequently turned into a dissertation topic—, for his generous financial support and for his friendly encouragement. I would also like to thank Robert Vichnevetsky for his kindness and support both during the early and later years of my Ph.D. studies. As well, I would like to thank Craig Nevill-Manning for agreeing to join my thesis committee at a later date and for his thoughtful insights about the project.

Although Vasek Chvatal was not part of my thesis committee, I am deeply indebted to him for the role he played as a Graduate Chair. He took an active part in helping me put my thesis committee together and facilitating communication among myself and the members of my committee. As well, I would like to thank Andrew Gelsey for acting as an interim advisor from January to August 1995.

I have also benefited from interaction with people outside my department. In particular, I received very useful advice on how to formalize my ideas from Nathan Intrator of Tel-Aviv University. Gary Cottrell of UCSD together with an anonymous reviewer provided very insightful comments on the study reported in Chapter 5. Catherine Myers of Rutgers-Newark provided me with a lot of help and advice during my first few years in Mark Gluck's Lab and Evangelia Micheli-Tzanakou of Rutgers-New Brunswick provided a warm learning environment in her department.

I would also like to thank the various people who provided me with exciting opportunities during my years as a graduate student. In particular I would like to thank Stu Zweben for hiring me as a lecturer at Ohio State University during the 1999 Winter and Spring quarters; Nathan Intrator for offering me a research fellowship at Tel-Aviv University during the 1997-1998 school year; Mark Gluck for hiring me as a research assistant in his Rutgers-Newark Lab from April 1994 to September 1997; William Cohen for hiring me as a Research Assistant at Bell Laboratories during the 1992 Spring semester; Jean-Gabriel Ganascia for providing me with space in his research lab at l'Universite Pierre et Marie Curie in Paris in the 1995 Spring quarter; John Taylor for doing the same at his King's College Lab in London in the 1995 Winter quarter; and Christiane Fellbaum, George Miller and Randee Tengi for welcoming me in the Cognitive Science Lab of Princeton University where I conducted the greatest part of my research.

My life as a graduate student would not have been the same without the support of many friends both at Rutgers and elsewhere. I am thankful to Stephen Kwek for his friendship, advice, and support during both my undergraduate and graduate years and despite the many miles often separating us. Since I stepped foot at Rutgers and until the last minutes I spent there, Dawn Cohen was a constant source of encouragement and advice helping me avoid the obstacles of a graduate career and cope with those I didn't avoid. Inna Stainvas of Tel-Aviv University made the year I spent there a very pleasant and instructive one as we spent many hours discussing various aspects of our projects while working side by side on our dissertations. I also very much enjoyed the terms I spent TA'ing Data Structures with Sesh Venugopal as well as our many "philosophical" conversations. I am thankful to my friends Frederique Alexandre, Rodrigo Arias, Florence Buzeyne, Tim Cooley, Claudia Goldmann, Peter Mones, Hava Siegelmann, Zehra Venugopal as well as office-mates Phil Bohannon, Rick Kaye and Fan Min for contributing to making my years as a graduate student most rewarding. Sadly, I can only wish that Paula Bonato whose friendship and kindness I have much missed since her untimely passing in July 1992 were here today to share in this moment.

Finally, I would like to thank my family for their patience and support throughout my studies. My parents, Michel and Suzanne Japkowicz as well as my sister, Florence Japkowicz-Ricouvier, and my brother, Maxime Japkowicz, always showed great interest in my work. Their constant encouragement, their pride in my successes and their deep concern for my difficulties are some of the major forces that helped me focus on my work and carry it through successfully. Robert and Hedwin Naimark, my cousins, were also very supportive and I much appreciated their advice on various aspects of the academic world. Toba and Michael Ripsman, my in-laws, were most understanding and supportive of my efforts and I appreciated them sharing my joys and disappointments as they would have for their own daughter. Last but not least, I would like to thank Norrin Ripsman, my husband, for all the help, support, and love, without which I could not have completed my degree. Despite the demands of his own Ph.D. and Postdoctoral studies, Norrin invested a lot of his time and energy helping me solve the problems I encountered during my graduate career and getting through its most trying problems. I am thankful for his patience, his understanding and his caring.

Dedication

I dedicate this dissertation to
my parents, Michel and Suzanne Japkowicz,
and
my husband, Norrin M. Ripsman

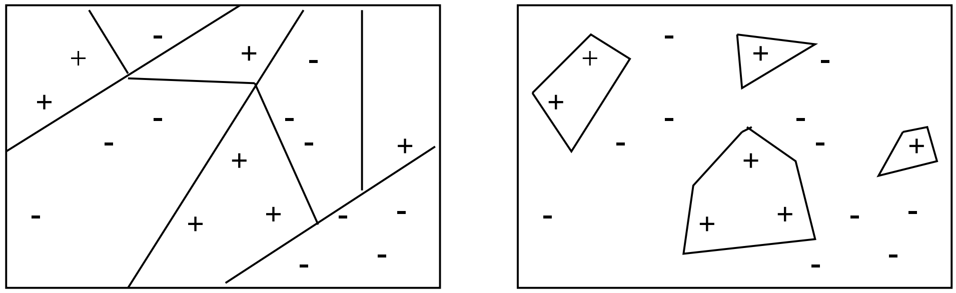
Table of Contents

Chapter 1
Introduction

Learning from examples or *induction* is an important focus of research in statistics, neural network technology and artificial intelligence. It consists of acquiring statistical or logical rules for *describing* a set of observed data points and extrapolating from them. One practical purpose of such rules is to *classify* novel data—data that was never seen by the inductive system—that is, to assign these data to the class in which they fit best. As a classical illustration of the classification problem, consider the task of diagnosis of whether a patient is sick or not. In this task, patients are described in terms of medical test results, such as blood pressure, blood tests, body temperature, and so on, and are labeled as either "sick" or "healthy". The goal of the classifier is to induce a set of rules based on these medical test results that can not only describe the medical characteristics of sick or healthy patients, but also predict whether a new patient having been subjected to the same tests is sick or healthy. Although the general problem of classification consists of assigning the data to any number of different classes, this dissertation restricts its attention to the problem of *binary learning* which takes into consideration only two classes: the class representing the concept and the class representing counter-examples of the concept. The simple medical diagnosis illustration constitutes an example of a binary learning problem.

Binary learning can be approached in one of two ways. The first way consists of considering *both* examples and counter-examples of the concept at hand and learning how to *discriminate* between these two classes. The second way consists of considering *only* instances of the concept and learning how to *recognize* those examples. More generally, discrimination-based learning falls in the category of *supervised learning* since it requires labeled examples of both the concept and counter-concept, and derives a discrimination function based on this information. Recognition-based learning, on the other hand, falls in the category of *unsupervised learning* since unlabeled data are fed to the learning system which is left to choose an internal organization on its own.

Discrimination- and recognition-based learning are illustrated in Figure 1.1. In discrimination-based learning (Figure 1.1(a)), boundaries are drawn *in-between* the examples and counter-examples of the concept. Conversely, in recognition-based learning (Figure 1.1(b)), boundaries are drawn *around* the examples of the concept with no knowledge about the counter-concept class. While the goal of the learning process in discrimination-based learning is classification, classification is only a by-product of the learning process in recognition-based learning.



(a) Discrimination (b) Recognition

Figure 1.1: The two approaches to concept learning: Discrimination versus Recognition. Note that the purpose of these figures is simply to illustrate the difference in learning strategy of the discrimination versus the recognition based approach to classification. They do not intend to illustrate the functioning of any particular classifier.

For example, in the medical diagnosis task, discrimination-based learning consists of finding rules that are sufficient to differentiate healthy from sick patients whereas recognition-based learning consists of finding rules for describing, say, healthy patients. The first approach returns a signal indicating whether the patient is "healthy" or "sick" whereas the second approach returns a signal indicating whether the patient is "healthy" or not. Since, in boolean logic and with the conventional meanings, "not healthy" = "sick", classification is performed by both schemes.

My dissertation is concerned with the problem of concept-learning in the absence of counter-examples and, thus, concentrates on the recognition-based learning paradigm. More precisely, I study in great detail *autoassociation-based classification*, a particular recognition-based technique originally introduced by [Hanson and Kegl1987] in the context of a natural language acquisition task. The purpose of my dissertation is to test this method on engineering applications, to explain its mechanisms, and to explain what makes it particularly efficient and accurate on a certain class of practical domains.

This chapter is divided into four sections. Section 1.1 presents the motivation for this dissertation. Section 1.2 discusses the challenges encountered when learning a concept in the absence of counter-examples. Section 1.3 then describes the purpose of the dissertation in more detail. And Section 1.4 is a Reader's Guide to the subsequent chapters.

1.1 Motivation

The overwhelming majority of research currently pursued within the framework of concept-learning concentrates on discrimination-based learning (Binary Hypothesis Testing [Fukunaga1990], Multi-Layer Perceptrons (MLP) [Rumelhart *et al.*1986], C4.5 [Quinlan1993], CART [Breiman *et al.*1984], Nearest-Neighbors [Fukunaga1990], and IB4 [Aha *et al.*1991]). The imbalance between the two approaches is well illustrated by the fact that in the 1997 set of proceedings of the National Conference on Artificial Intelligence, a representative conference, eighteen papers were listed in the "Learning" category, with twelve of them considering discrimination-based approaches to learning, three of them considering recognition-based approaches to learning, and three of them not belonging to either class of papers, since they discussed topics which did not fit in this simple categorization.[1]

This emphasis can present a practical problem: there are real-world engineering problems for which counter-examples are both scarce and difficult to gather. In machine fault diagnosis, for example, examples of the concept are plentiful and typically involve recording from the machine during normal operation. Counter-examples, on the other hand, involve causing the machine to break down in each manner in which future failure is possible so that a recording can be made of each failure type. Monitoring tasks which consist of examining a system's readily available signals and issuing an alarm when a potential problem is detected fall in this category of problems. The need to monitor a system's operation arises frequently and the development of reliable techniques for doing so would provide important benefits for critical military and commercial systems (e.g., helicopter gearboxes, shipboard fire pumps, motors, and generators).

For these problems, recognition-based learning systems such as Single Hypothesis Testing [Fukunaga1990], Signal Detection [Van Trees1968], Example-based Learning [Sung and Poggio1994], View-Based Eigen-Spaces [Pentland *et al.*1994], [Turk and Pentland1991], PARSNIP [Hanson and Kegl1987], are much more appropriate than discrimination-based learning systems because they do not use counter-examples in the concept-learning phase and thus require fewer examples altogether.[2] Studying recognition-based systems is thus an important endeavor since it will allow us to improve our ability to handle an important class of real-world classification tasks that cannot be tackled by discrimination-based approaches.

---

[1]One of the three papers that did not fit the classification mold discussed representation issues that could apply to both approaches, and the other two discussed solutions to optimization and modeling problems. Even the three papers which I categorized as fitting in the recognition-based approach class do not clearly perform classification. Nevertheless, their methodology could easily be adapted to a classification task, and I thus chose to consider them in my categorization.

[2]Binary learning is comprised of three parts: *preliminary setting*, *concept-learning* and *threshold-determination*. Preliminary setting refers to the process of gathering all the information required by the concept-learner and/or threshold-determiner to perform classification (such information includes choosing a distribution to which the data can be fitted, gathering specialized knowledge about the task, choosing a threshold-determiner, and setting the concept-learner and/or threshold-determiner's parameters, though different binary learners require different types and amounts of preliminary setting). Concept-learning is the procedure that acquires a concise description of the concept class (relative to the counter-concept class in the case of discrimination, and in isolation, in the case of recognition) while threshold-determination is the procedure that sets a boundary between the concept and counter-concept classes. Although concept-learning by autoassociation takes place in the absence of counter-examples, preliminary-setting and threshold-determination may require some of them.

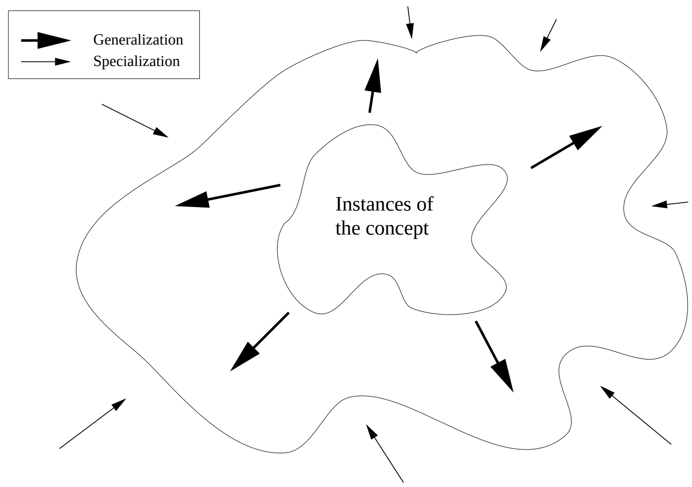


Figure 1.2: The Generalization/Specialization Framework: Inductive processes seek a balance between their generalization thrust and their specialization strive.

1.2 Challenges associated with Learning a Concept in the Absence of Counter-Examples

The marked preference for discrimination- over recognition- based methods that can be noted within the field of inductive learning is not fully unexpected. It can be explained by the fact that discrimination-based learning is a much more "natural" approach than recognition-based learning, given the task at hand of discriminating between the two classes.

Indeed, a useful framework for understanding the induction process of a generic classifier is to decompose it into two opposing processes in search of an equilibrium. The first process seeks to *generalize* from the positive examples of the concept given to the classifier; while the second one seeks to *specialize* these examples. Generalization amounts to seeking a concept description which characterizes all the positive examples of the concept while specialization consists of seeking a concept description which excludes all the counter-examples. This framework is illustrated in Figure 1.2 and it will be formalized in Chapter 2, following [Mitchell1982] in using the notion of "Version Spaces".

Within this generalization/specialization framework, discrimination-based classifiers learn concepts by generalizing from the examples of the concept while specializing using the counter-examples. Although recognition-based learners are also able to generalize from examples of the concept, they do not, however, have access to counter-examples for specialization. Assuming this framework, an important question then becomes: how can inductive systems learn how to discriminate between the concept and the counter-concept (or complementary concept) class in the absence of counter-examples? Since [Mitchell1982]'s framework suggests that, in the absence of counter-examples, an inductive learning system will never stop generalizing (because it is not aware of any boundaries), a recognition-based system is expected to over-generalize the concept class and thus be an inappropriate classification tool.

In fact, as will be clarified in the next section, this is not the case since there is a way in which classification is possible in the absence of counter-examples. In particular, the dissertation will show that the autoassociator's ability to specialize and thus, to classify domains accurately, follows from a particular *internal bias that gets implicitly applied to the generalization process.* This bias, it will be shown, is very powerful, at least for a certain type of domain of practical interest.

1.3 Purpose of the Dissertation

As already mentioned, the purpose of this dissertation is to analyze the recognition-based learning system of [Hanson and Kegl1987] in detail, with the following aims in mind: to determine what types of domains it is well-suited for; to extract the particular bias that makes it effective at classification on those domains; and, to ascertain whether it might be fruitful to employ even on certain tasks that might appear better suited for discrimination-based learning. These three aims touch upon important questions which have not been addressed before but which bear investigation if we are to employ recognition-based learning effectively and optimize our classifiers.[3]

The [Hanson and Kegl1987] approach to classification consists of training a one-hidden layer, nonlinear autoassociator to reconstruct the concept class of a domain at the output layer (the autoassociator architecture is illustrated in Figure 1.3 and its use for classification tasks will be discussed in more detail in Chapters 2 and 3). Once training is achieved, the autoassociator is used for classification, relying on the idea that if the network can generalize to a novel instance, then this instance must belong to the concept class; but that if generalization fails, then the instance must be a counter-example of the concept. As I explain in Chapter 2, I selected the autoassociation-based system of [Hanson and Kegl1987] because it presents two key advantages over the other recognition-based approaches. First of all, it requires a minimum of preliminary setting and second, it is a nonlinear system.[4]

[3]Most of the research on autoassociators deals with different questions: data compression ([Cottrell *et al.*1987], [Kramer1991]) and independent component analysis ([Oja1995]). Furthermore, the current research which deals with autoassociation-based classification ([Schwenk and Milgram1995]) does not concentrate on learning in the absence of counter-examples, since it uses one autoassociator per class and does not attempt to explain the inner-workings of the system. [Hanson and Kegl1987] and its successor [Petsche *et al.*1996], themselves, who do focus on learning in the absence of counter-examples, though they demonstrate that autoassociation-based classification works, do not explain why or how it works and for what type of domains it works.

[4]Nonlinearity is important since it potentially permits the application to any type of domain, whereas linearity restricts the use of the classifier to *linearly separable* domains, i.e., domains in

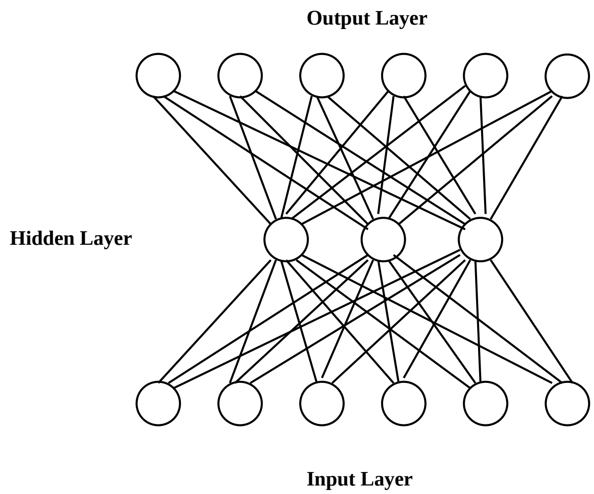


Figure 1.3: The Autoassociator Architecture. In this particular network, the input and output layers contains 6 units and the hidden layer contains 3 units. While the output layer is linear, the hidden layer is nonlinear and contains sigmoidal activation functions.

Prior to studying this system, however, a preliminary question that we must confront is: do recognition-based systems entail a significant loss of performance (both in terms of accuracy or time efficiency) compared to discrimination-based ones? If they do, then the advantage of studying such systems would be merely of academic interest.

To answer this question, I have tested the autoassociation-based recognition classifier of [Hanson and Kegl1987] against four leading discrimination-based systems (MLP, C4.5, IB4 and Predictive Autoencoding) on three real-world domains (helicopter gearbox fault monitoring, DNA Promoter recognition, Sonar Detection). My results indicate that not only does recognition-based learning entail no decrease in performance, but it actually outperforms these classifiers in some cases. I verify this conclusion by testing two of the systems—the autoassociator and MLP, the two systems of interest in the remainder of the dissertation—on an idealized artificial abstraction of the real-world domains. These results are encouraging for they indicate that for a large class of real-world problems—*nonlinear* and *multimodal* problems—where a wealth of counter-examples may be unavailable, we can effectively employ a recognition-based classifier without sacrificing accuracy or efficiency.

The remainder of the dissertation is driven by the results obtained in this preliminary set of experiments. In particular, it explains why these results were obtained by answering concrete questions elucidating the general concerns raised earlier. In more specific terms, the dissertation seeks to study the performance of the autoassociator by answering the following three questions:

- What features of the autoassociator make it capable of classifying domains

---

which classification can be achieved using a single line or hyperplane in the input-space.

which are multi-modal and nonlinearly separable?

- What causes the autoassociator to learn the artificial domain significantly more efficiently than MLP on these domains?

- What further domain characteristics cause autoassociation-based classification to be more accurate than MLP? What further domain characteristics cause MLP to be more accurate than autoassociation-based classification?

The answers to these questions constitute the core of this dissertation which concludes by discussing the implications of this study and proposing future extensions of the research.

1.4 Reader's Guide

The remainder of this dissertation is composed of seven chapters.

Chapter 2 introduces the terms and concepts used throughout the dissertation and ties the study into the broader automated-learning literature.

Chapter 3 describes the autoassociation-based classifier in great detail, noting its relation to neuroscientific and cognitive processes. In addition, it introduces a series of threshold-determination methods compatible with the autoassociative process and subsequently used in the study.

Chapter 4 assesses experimentally whether recognition-based systems entail a significant loss of performance (both in terms of accuracy or time efficiency) compared to discrimination-based ones. It reports the preliminary results obtained using both real-world domains and artificial abstraction of these domains.

Chapters 5, 6 and 7, each, respectively, answer one of the questions that arose from the results of the experiments of Chapter 4.

In particular, Chapter 5 investigates the operation of the autoassociator. More specifically, starting from [Mitchell1982]'s framework for inductive learning which decomposes the binary-learning task into a *generalization* and a *specialization* process, Chapter 5 identifies the specialization bias of the autoassociator that allows it to learn a concept in the absence of counter-examples. After identifying this specialization bias on the idealized artificial domain of Chapter 4, Chapter 5 tests its flexibility in order to establish some of the advantages and limitations of autoassociation-based classification.

Chapter 6 focuses on explaining what causes the large difference in time efficiency between MLP- and autoassociation- based classification on the idealized artificial domain of Chapter 4. The observation is explained by the fact that on this particular domain, the two systems use different learning strategies: while MLP performs a hypothesis-driven search of the generalization space which incurs a long initial latency period, the autoassociator learns the domain in a data-driven fashion which is practically instantaneous. While the difference in strategies adopted by the two systems is not surprising given the fact that the direct tasks that each network is required to solve are very different, this study suggests that the backpropagation procedure is flexible enough to select diametrically opposed strategies and suggests a number of further experiments geared at explaining the various learning modes to which it can resort.

The purpose of Chapter 7 is to explain why the autoassociation-based classification system is at times more accurate than, at times as accurate as, and at times less accurate than the MLP network. This explanation is obtained by testing the two classifiers on a number of variations of the idealized artificial domain of Chapter 4. In more detail, Chapter 7 shows how the degree of relative accuracy of the two systems depends on the amount of specialization required by the domain at hand. In particular, autoassociation-based classification is more accurate than MLP on domains which require *strong* specialization caused by the *counter-concept* class and *weak* specialization caused by the *concept* class. On the other hand, MLP is more accurate than the autoassociator on domains that require *strong* specialization caused by the *concept* class. These results are subsequently used to explain the observations on accuracy that were made on the real-world domains of Chapter 4 and they are generalized by a technique similar to Mathematical Induction to domains of higher dimensionality.

Finally, Chapter 8 concludes the dissertation by summarizing its results, describing several avenues for future work, and discussing the broader implications of the dissertation.

Chapter 2
Background

As suggested by the introduction, this dissertation is grounded in a number of different—though related—areas of automated induction including pattern recognition and statistics, detection theory, machine learning, and connectionist systems. The purpose of this chapter is to review the main concepts borrowed from these fields which will be used in the development of the dissertation, as well as to fit the study in a broader context. In particular, this chapter serves three functions: it defines the problem of binary learning formally, it reviews previous work on recognition-based learning and identifies some of its limitations, and it introduces the various technologies used in this dissertation.

In more detail, the chapter is divided into five sections. Section 2.1 defines binary learning in the framework of hypothesis testing and reviews its classical discrimination- and recognition- based solutions. Section 2.2 reviews previous work on concept-learning in the absence of counter-examples from the areas of signal detection and pattern detection in images, and considers its limitations. Section 2.3 introduces elements of the field of Neural Networks, focusing on the autoassociative feedforward scheme, the method of interest in this study, and its heteroassociative counterpart, the MLP Network. Section 2.4 discusses important theories of induction both from the fields of Neural Networks and Machine Learning that will be used to frame the dissertation. Finally, Section 2.5 discusses the practical issues that arise when using and comparing Neural Networks the way this work does.

2.1 Hypothesis Testing

In its most basic form, the problem of binary learning can be discussed in the framework of hypothesis testing. There are different types of hypothesis testing problems, but in this dissertation, I am only concerned with the problems of *single* and *binary* hypothesis testing. For both these problems, we are given two hypotheses: H_0, the null hypothesis, and H_1, the alternative hypothesis. Given an observation variable Y and a particular value y of Y, the two hypotheses H_0 and H_1 are defined as follows:

H_0: y belongs to class A
H_1: y belongs to class B

where $ClassB = \neg ClassA$. The purpose of single hypothesis testing is to find out whether H_0 holds (in this problem, H_1 is not explicitly considered) whereas the purpose of binary hypothesis testing is to find out which of H_0 or H_1 holds.

Like generic binary problems, single and binary hypothesis testing problems can be further decomposed into three subproblems: *preliminary setting*, *concept learning* and *threshold determination*. In the context of hypothesis testing which is a *parametric* approach to binary learning[1], preliminary setting consists of choosing a distribution for representing the data. Concept-learning, then, consists of evaluating the distribution of the available training data (as is, in the case of binary hypothesis testing, and in relation to its mean, in the case of single hypothesis testing). Threshold-determination, at last, consists of setting a boundary between the data of Classes A and B. In the binary hypothesis testing problem—the most commonly used approach to hypothesis testing—both instances of classes A and B are provided to the concept-learner. Binary hypothesis testing belongs to the class of *discrimination-based* binary learning problems. In the single hypothesis testing problem, only instances of class A are given to the concept-learner. Single hypothesis testing belongs to the class of *recognition-based* binary learning problems. The following is a description of the statistical solutions to the binary hypothesis testing and single hypothesis testing problems. This material constitutes the foundations of the field of pattern-recognition.

2.1.1 Binary Hypothesis Testing

As mentioned previously, binary hypothesis testing consists of finding out which of H_0 or H_1 holds. A solution to the concept-learning problem is presented, followed by two solutions to the threshold-determination problem. A simple implementation of this approach is to assume that the data can be accurately modeled by a Gaussian distribution.

Concept Learning: Bayes' Theorem

In binary hypothesis testing, concept-learning consists of calculating the probabilities $P(H_0|y)$ and $P(H_1|y)$ that y satisfies hypotheses H_0 and H_1, respectively. While $P(H_0|y)$ and $P(H_1|y)$ are not known, it is assumed that $P(y|H_0)$ and $P(y|H_1)$, the conditional probabilities of an observation given a specific hypothesis, $P(H_0)$ and $P(H_1)$, the prior probabilities of each hypothesis, and $P(y)$, the prior probability of observation y, can all be estimated. If this is the case, then $P(H_0|y)$ and $P(H_1|y)$ can be estimated as well using Bayes' Theorem:

---

[1] Binary learning can be performed by *parametric* or *non-parametric* approaches. In parametric approaches, the data is assumed to fit a pre-determined statistical distribution and learning consists of determining the parameters of this distribution. In non-parametric approaches, no statistical distribution is assumed and learning takes into consideration the true distribution of the data. Learning using a parametric approach has the advantage of being simple once a distribution has been selected, but it has the disadvantage of being inaccurate if the chosen distribution does not fit the data. Non-parametric approaches, on the other hand, are more complex but have the advantages of modeling the data more closely.

$$P(H_x|y) = \frac{P(y|H_x)P(H_x)}{P(y)}$$

with $x = 0$ or 1.

Threshold Determination

Estimating the probability density function of the data constitutes the principal part of solving the binary hypothesis testing problem, but it does not solve the problem per se. In addition, a *decision rule* must be designed to choose one hypothesis over the other. Associated with Bayes' Theorem are Bayes' criterions, two possible decision rules, which include the *simple criterion* of maximum a-posteriori probability and the *criterion for minimal cost.*

Bayes' simple criterion is straightforward. It consists of choosing:

H_0: if $P(H_0|y) > P(H_1|y)$ or
H_1: if $P(H_1|y) \geq P(H_0|y)$

While straightforward, however, this criterion does not take into consideration the possibly different costs of misclassifying a pattern of class A (Type I Error) or misclassifying a pattern of class B (Type II Error). Yet, such costs can be very important. Consider, for example, the medical application where class A corresponds to "cancer" and class B corresponds to "Healthy". In this case, Type I errors are much more "costly" than Type II errors (as far as the patient is concerned) since it is more dangerous for a patient to be misdiagnosed as healthy when sick with some dangerous disease than it is to be misdiagnosed as having the disease when healthy. Using Bayes' simple criterion is therefore not the best criterion to use when faced with the possibility of dangerous diseases.

Bayes' criterion for minimum cost takes this issue into consideration by generalizing the simple criterion as follows. Defining

c_{ij} = Cost of choosing Hypothesis H_i when Hypothesis H_j holds and
$r_i(y) = c_{i0}P(H_0|y) + c_{i1}P(H_1|y)$

with $i, j = 0, 1$, this criterion consists of choosing and choosing:

H_0 : if $r_0 > r_1$ or
H_1 : if $r_1 \geq r_0$

With this criterion, a patient faced with cancer, now, has the possibility of increasing the cost of a Type I error while decreasing that of a Type II error in his or her decision rule. Bayes' criterion for minimal cost is thus a more flexible technique worth using when the cost of different alternatives are different and can be assessed.

The two Bayes' decision criteria just presented illustrate the notion that there are different ways to choose a threshold once concept-learning has occurred. As a matter of fact, these two criteria are only a few of a large number of methods that can be used for setting a boundary between classes A and B. Other well-known methods include the Neyman-Pearson test which minimizes one of the type of errors (Type I or II) subject to the other being constant and the Minimax test which is independent of the prior probabilities of each hypothesis. Several other simple threshold-determination methods have been designed in this dissertation and will be described in Sections 3.3.1 and 3.3.2 of Chapter 3. These methods will be used in the experiments of Chapters 4 and 5.

Furthermore, there exists an additional method of a completely different nature which consists of plotting the relation between the Type I and type II errors by changing the value of the threshold continuously. This method is called the *Receiver Operating Characteristics (ROC)* Analysis technique and is useful for comparing different probability density estimation or other concept-learning techniques over all possible thresholds. It is an important method because it eliminates the bias inherently introduced by the choice of a decision criterion. ROC Analysis will be described in more detail in Section 3.3.3 of Chapter 3 and it will actually be used in the experiments of Chapter 7.

2.1.2 Single Class Hypothesis Testing

Single class hypothesis testing consists of finding out whether H_0 holds, independently of H_1. A solution to the concept-learning problem is presented followed by a solution to the threshold-determination problem.

Concept-Learning: Mahalanobis Distance

In the case of single class hypothesis testing, it is often not the probability density function that is estimated per se during the concept-learning phase, but rather, the probability function of the distances between each point of Class A and the mean of the distribution. A simple measure of distance is the Mahalanobis distance which is the distance of a point to the mean of the data in class A, normalized by the inverse of the covariance matrix. This distance is defined as:

$$d^2(y) = (y - M)^T \Sigma^{-1} (y - M)$$

where M represents the mean of the distribution of Class A and Σ is its covariance matrix. The distances can be modeled by a Gaussian distribution. The function $P(d^2(y)|H_0)$ being computed, threshold-determination must then be performed in order to obtain a classifier.

Threshold Determination

In single hypothesis testing, classification is achieved by assuming that if $d^2(y)$ is small, then $y \in$ class A, whereas if $d^2(y)$ is large, then $y \in$ class $\bar{A}$ or B. The question of what constitutes a "small" or a "large" variance is left to the threshold-determination component. In other words, the goal of threshold-determination is to find a threshold, $Thresh$, such that one can choose:

H_0: if $d^2(y) < Thresh$ or
H_1: if $d^2(y) \geq Thresh$

If an estimate of the noise level present in the concept class is provided, then $Thresh$ is set so as to reject a percentage of examples equal to that noise level. For example, if it is known that 5% of the people diagnosed with a dangerous disease are actually healthy, then $Thresh$ could be chosen such that only 95% of the concept-class training examples provided to the system satisfy H_0.

If, on the other hand, the noise level of the concept class is not provided, then, unless the distribution of distances from M to the concept class presents natural breaks allowing a user to guess the value of this noise level, then binary hypothesis testing techniques will need to be applied to the distances from M to both classes A and B. In this case, counter-examples are thus used in the binary-learning process of a recognition-based system, but they serve only as "sparse boundary markers". As discussed in Section 2.2.4, the number of counter-examples thus necessary by this approach is much smaller than the number of counter-examples necessary in a discrimination-based approach. Finally, another approach to threshold-determination is the ROC technique mentioned in Section 2.1.1.

2.1.3 Limitations of Hypothesis Testing Approaches

Although the approaches just presented are theoretically sound—the Bayes' error, obtained by applying Bayes' theorem followed by Bayes' simple criterion, is, theoretically, the lowest possible error that can be reached by a classifier—they present certain practical disadvantages. While $P(y|H_x)$, $P(d(y)|H_x)$, $P(H_x)$ and $P(y)$ ($x = 0$ or 1) can all be estimated directly, in the case of finite training samples, functions of these estimates can be severely biased, yielding an unacceptably high error for $P(H_x|y)$ (see [Fukunaga1990]). For this reason, as well as the fact that the statistical methods presented above are *parametric* and thus, are inappropriate when the data does not follow the distribution assumed by the preliminary setting phase, the fields of Pattern Recognition, Machine Learning and Neural Networks have developed a large number of techniques geared at reducing the amount of estimation error associated with $P(H_x|y)$ and at learning a classification *non-parametrically*, [See, for example, [Fukunaga1990], [Weiss and Kulikowski1991], [Hertz *et al.*1991], [Bishop1995]].

The main technique studied in this dissertation—Autoassociation-based learning—and its discrimination-based counterpart—MLP—also depart from pure statistical hypothesis testing techniques since they are connectionist non-parametric methods. These methods will be described in Section 2.3, Chapter 3, and Chapter 4. Before that, the next section surveys practical applications of single-class recognition-based concept-learning technique which use both statistical hypothesis testing and non-statistical approaches to the problem.

2.2 Recognition-Based Learning in Real-World Applications

As was seen in the previous section, concept-learning in the absence of counter-examples, or recognition-based learning, is not a novel enterprise since it has been considered previously in the context of hypothesis testing. Practically speaking, techniques for recognition-based learning were used in the fields of "Signal Detection Theory" [Van Trees1968] and "Pattern Detection in Images" which rely on hypothesis testing and other techniques that require a different type of preliminary setting phase [Sung and Poggio1994]. Various methods designed within these two fields are now described, followed by a discussion of their limitations with respect to the autoassociation-based classification method of [Hanson and Kegl1987] studied in this dissertation.

2.2.1 Signal Detection

In signal detection, a "receiver" observes a waveform for T seconds and decides, on the basis of this waveform, which symbol was transmitted during time interval T. In certain cases, the problem consists of distinguishing between two different signals, but in others, the problem consists of estimating whether or not a signal has occurred and been detected. In the second case, the problem is one of detection in the absence of counter-examples since the only class of interest is the class of signal occurrence.

Signal detection, has been typically addressed in one of two ways:

- By "creating" an artificial counter-concept class representing the absence of signal,
- By learning directly from the concept data—the data representing signal occurrence—with certain assumptions regarding the data distribution in that class.

In more detail, the general problem of signal detection attempts to establish which of the two hypotheses

H_0: Signal A
H_1: Signal B

has occurred. In the case where Signals A and B are two different signals, then the problem reduces to a typical binary hypothesis testing problem and can be solved using the methods discussed in Section 2.1.1 or more sophisticated ones (E.g., see [Fukunaga1990], [Weiss and Kulikowski1991], [Hertz *et al.*1991], [Bishop1995])

If, however, *Signal A* corresponds to "some signal A" and *Signal B* corresponds to "not signal A", then the problem reduces to the case of single hypothesis testing problem. As mentioned before, this problem has been approached in one of the following two ways.

First, by considering that the single hypothesis testing problem could actually be reduced to a binary hypothesis testing problem as follows ([Van Trees1968]): Let X be the actual signal A, then observation Y can in fact be expressed as $Y = X + N$ where N represents the noise present in the channel. Assuming that N follows a fixed distribution, then hypotheses H_0 and H_1 can be expressed as:

H_0: Signal Y has occurred
H_1: Signal N has occurred.

At this point, the problem can be treated as a regular binary hypothesis testing problem, and solved using classical methods as in the previous case.

The second alternative to dealing with a single hypothesis testing problem is to tackle it directly. In this case, the technique used to solve the single hypothesis testing problem discussed in Section 2.1.2 can be used. Note, however, that this direct technique has been reported to work well when the dimensionality of the data is very low (i.e., 1 or 2); but that its error increases significantly as the dimensionality increases [Fukunaga1990].

2.2.2 Pattern Detection in Images

Pattern Detection in images is a classical computer vision problem. Considering as input an arbitrary image in the form of a scanned photograph, for example, it consists of determining whether or not a given object is present in that image. Again, this problem is one of detection in the absence of counter-examples, or recognition-based learning, and in addition to having been attempted in the two ways described in the context of signal detection (E.g., [Sung and Poggio1994]), it was attempted in a third series of ways:

> By having a preliminary setting phase consisting of gathering contextual information about the particular task at hand, such as
>
> - *pictorial templates*, divided into the categories of
>   - *Fixed Correlation Templates*
>   - *Deformable Templates*
> - or *image invariants*.

A description of these three approaches is given following [Sung and Poggio1994].

Fixed correlation templates

Fixed correlation template approaches compute a difference measurement between a fixed target pattern and observed images. Like in the single hypothesis testing approach, this difference is subsequently thresholded for matches, however, the difference between these two techniques is that, rather than summarizing the concept data by its mean and covariance matrix, in the fixed correlation template approach, a few actual images are stored as reference and compared to test images. These images must be carefully selected according to contextual knowledge about the task, during the preliminary setting phase.

Deformable templates

Deformable template approaches are similar to fixed correlation ones except that they have some built-in variability for the templates. This allows certain components of a target image to be located within a range of relative positions to other components rather than to a fixed relative position. Such a scheme is especially useful for problems of face detection, for example, where diverse face expressions can be modeled with the same template. Again, this scheme requires contextual knowledge about the task, that must be gathered during the preliminary setting phase.

Image Invariants

Image invariants are an extreme case of deformable templates in which, rather than storing an image template (with some freedom to deform this image), only a few aspects of the image are memorized. In this respect, this approach is close to the single hypothesis testing approach except that instead of memorizing some general statistical information about the class of images, only a few domain-specific relations are memorized. In the case of face recognition, for example, it has been observed that the eye region of a human face is almost always darker than the cheeks' and the forehead's. Similarly, the bridge of the nose is always brighter than the two flanking eye regions. Such observed regularities can be encoded during the preliminary setting phase as ratio templates used to match faces [Sinha1994].

2.2.3 Limitations of Common Recognition-Based Approaches

The various concept-learning schemes described in the previous two subsections all present a common feature: if they cannot be transformed into a binary learning problem, they require a significant amount of preliminary information gathered during the preliminary setting phase. In particular, they require either information regarding the distribution of the concept data or contextual information about the task. As mentioned in the introduction, counter-examples are not always available. Furthermore, gathering and coding statistical or contextual knowledge about the task is not necessarily possible, simple, or efficient. For these reasons, the question asked in this section is: what is the minimum amount of preliminary setting necessary when counter-examples are absent?

Although all the methods described thus far use either counter-examples during the concept-learning phase or gather distributional or contextual information during the preliminary setting phase, the "view-based eigen-spaces" approach described by [Turk and Pentland1991] and [Pentland *et al.*1994] learns a concept from concept data alone and does not require the gathering of any distributional or contextual information during the preliminary setting phase.

The approach considers both the problems of face detection (distinction between faces and non-faces) and face recognition (distinction between the faces of different people), and represents the space of all faces as a linear combination of several orthonormal eigen-images. The eigen-images are computed by performing Principal Component Analysis on a database of face shots. The modeling scheme assumes that all faces lie within the orthonormal basis formed by the eigen-images, and each face shot corresponds to a set of coefficients that describes its coordinates in this eigen-image basis. As just mentioned, this approach can solve two problems at once. However, the problem we are principally interested in, given the topic of this dissertation, is the problem of face detection since it is a natural recognition rather than discrimination problem. As for other recognition-based systems, this problem is solved by computing and thresholding $Thresh$, a measure of how badly the eigen-image basis reconstructs the particular face.

The preliminary setting necessary for this approach is the setting of an optimal number of principal components for the particular task considered. This preliminary setting can be done by projecting the data onto a number of components equal to the dimensionality of the domain and keeping only the principal components which carry a significant amount of information. The view-based eigen-spaces method thus suggests that concept-learning in the absence of counter-examples and minimum preliminary setting is possible. However, the advantages and limitations of this method have not been discussed. In particular, since Principal Component Analysis is a linear procedure, the view-based eigen-spaces method is likely to classify only linearly separable domains.

The [Hanson and Kegl1987] autoassociation-based classification system studied in my dissertation and also used in [Petsche *et al.*1996] is very similar to the recognition-based system described by [Turk and Pentland1991] and [Pentland *et al.*1994], except that it implements Principal Component Analysis using a connectionist device rather than by applying the conventional Karhunen-Loeve statistical decomposition. Despite many beliefs to the contrary, such a move permits the introduction of nonlinearities in the transformation process (see Chapter 5) and thus yields a more powerful classification tool than the one introduced by [Turk and Pentland1991] and [Pentland *et al.*1994] which is limited to linear projections.

Similarly to the case of the "view-based eigen-space" method, the only preliminary setting necessary for this approach is the setting of an optimum number of hidden units. Many techniques have been designed for doing so in connectionist systems. In this dissertation, I have used the simplest method which consists of training the network on different capacities and testing the results on a testing set (see Section 2.5.4). Since the testing set is composed of both examples and counter-examples of the concept, in its current implementation, the preliminary setting phase of the autoassociator does not take place in the absence of counter-examples. However, it could have done so if I had integrated the parameter setting phase in the concept-learning phase using a pruning and weight-decay technique (e.g., [Hertz *et al.*1991], [Bishop1995]).

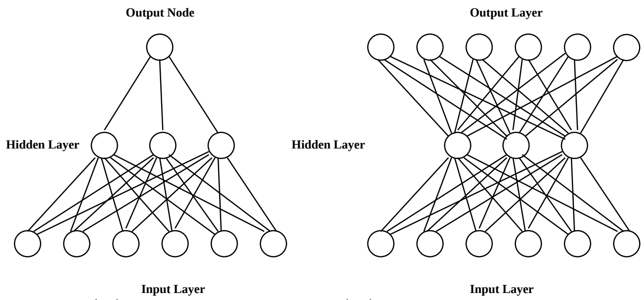
2.2.4 Current State of the Art in Recognition-Based Techniques

To summarize, the current state-of-the-art in recognition-based learning methods usually involves the transformation of the problem into a discrimination-based learning problem, or the gathering of statistical information about the concept class or contextual knowledge about the task during the preliminary-setting phase. In a few cases ([Hanson and Kegl1987], [Petsche *et al.*1996], [Turk and Pentland1991], [Pentland *et al.*1994]), recognition-based learning has been achieved without transforming the problem into a discrimination-based one, and with minimum prior setting.

The advantage of such methods is that, despite the fact that their preliminary setting and threshold-determination phases may use counter-examples, their concept-learning phase takes place in the absence of counter-examples and the complete recognition-based binary learning process requires fewer counter-examples than the complete discrimination-based one.[2] This property makes recognition-based systems such as those of [Hanson and Kegl1987], [Petsche *et al.*1996], [Pentland *et al.*1994], and [Turk and Pentland1991] *potentially* more useful than discrimination-based ones in problems for which counter-examples are either difficult or expensive to gather and prior information cannot be obtained easily.

The disadvantage of such systems, however, is that the robustness of their performance has never been established. This is the reason that they are only *potentially* more useful than discrimination-based ones, having been tested in only a few domains with no comprehensive explanation of their operation having been provided. This state of affairs, actually, constitutes the most crucial current disadvantage of these methods. This is why this dissertation concentrates on one such method—the [Hanson and Kegl1987] autoassociation-based classifier which has the advantage over the

[2]This is because even when counter-examples are required by the preliminary setting and/or threshold-determination phase, recognition-based approaches use fewer counter-examples than discrimination-based ones since discrimination-based approaches also make use of counter-examples in their preliminary setting and threshold-determination phases. Furthermore, there are indications that preliminary setting and threshold-determination could be achieved with fewer or no counter-examples, in which case, recognition-based methods could use even fewer counter-examples, or probably, no counter-examples whatsoever.



(a) MLP (b) Autoassociator

Figure 2.1: Examples of Feedforward Neural Networks: (a) a Discrimination Network; (b) an Autoassociator

[Turk and Pentland1991] and [Pentland *et al.*1994] methods of being nonlinear—and studies its performance and operation in great detail.

2.3 Concept-Learning using Feedforward Neural Networks

Since, as mentioned previously, the recognition-based technique studied in this dissertation uses a connectionist scheme and will be compared to its connectionist discrimination counterpart, I now describe how feedforward neural networks can be used to implement such binary learning approaches. Starting from Statistical Regression, I first describe the kind of functions that feedforward neural networks can represent and I then explain how these functions can be approximated using the backpropagation procedure, the training procedure used throughout this dissertation. The material discussed in these subsections comes from [Bishop1995] and [Hertz *et al.*1991].

2.3.1 Neural Networks and Statistical Regression

A formal definition of the binary learning problem typically involves an input vector x, and a response vector y which are such that the pair (x, y) belongs to some unknown joint probability distribution, P. The goal of concept learning is to induce a function $f(x)$ from a set $(x^1, y^1), (x^2, y^2)$, ... (x^N, y^N) of training examples, so that $f(x)$ approximates y.

In this section, I introduce a type of neural network that is capable of approximating, as closely as desired, any such continuous functional mapping f. This general class of networks is called the class of multi-layer Perceptrons and throughout my dissertation, I restrict my attention to multi-layer Perceptrons with a single hidden layer, i.e., with two layers of weights. Two examples of such feedforward neural networks are given in Figure 2.1. In figure 2.1(a), the input layer contains 6 units, the hidden layer consists of 3 units, and the output layer, of a single unit. Figure 2.1(b) is similar except for the output layer which contains 6 units.

Let us first consider the network of figure 2.1(a). Assuming that both the hidden and output layer have sigmoidal activations, then $f_k(x)$, the function implemented by the network at the output unit $k = K = 1$, is:

$$f_K(x) = g(\Sigma_{j=0}^{M} w_{Kj} g(\Sigma_{i=0}^{d} w_{ji} x_i))$$

where $g(x) = \frac{1}{1+e^{-x}}$, is the sigmoid function, d is the dimensionality of the input vector x ($d = 6$) and M is the capacity of (or number of hidden units in) the network ($M = 3$). w_{ji} is the weight going from input unit i ($i = 1..d$) to hidden unit j ($j = 1..M$) and w_{Kj} is the weight going from hidden unit j ($j = 1..M$) to output unit $k = 1..K$ ($K = 1$).[3]

In the network of figure 2.1(b), the same description holds, but rather than a single function, the output layer holds the result of several such functions $f_k(x)$ ($k = 1..K(= d)$) which all share the same hidden units but combine them in different ways through the layer of weights w_{kj} linking the hidden to the output layer.

There exist many ways other than neural networks to implement and represent functions. However, the usefulness of neural networks comes from the adaptability of their weights which can be *learned* using a simple optimization procedure for minimizing some chosen cost function. One such procedure is the *backpropagation procedure* and a possible cost function is the *sum of squares error*, both of which will be described in the next section. Prior to that, however, I discuss how both the networks of figure 2.1(a) and 2.1(b) can be used for classification, once learning has occurred.

The network of figure 2.1(a) is the most commonly used network in the field of connectionist systems. It is typically used to implement a discrimination-based approach to classification. Using this scheme, each element y_i of y, the response vector, takes on a value of "1" or "0" which is interpreted as to mean that x_i, the input vector associated with y_i belongs or does not belong to the concept class of the problem. After training the network to approximate the function $f_k(x)$ from which the training examples are believed to have been generated, it is expected that, if the training set was appropriately designed, the network will be able to compute the appropriate label ("1" or "0") for any input vector of size d, even if that vector did not appear in the training set. This is called *generalization* and will be discussed in great depth in Section 2.4.

The network of figure 2.1(b) is not as universal as the previous network, but it has been used previously to implement a compression scheme ([Cottrell *et al.*1987], [Elman and Zipser1988]). As well, and this is the application of interest here, it can be used to implement a recognition approach to concept learning as proposed

---

[3]Note that in the formula, both summations start at 0 rather than 1. This represents a convenient way to include the *biases* of the network—treated like regular adjustable weights w_{j0}, w_{k0} linking an extra virtual input or hidden unit fixed at value 1 to all the hidden units j and all the output units k—which are typically added to compensate for the difference between the average values of two successive layers of units.

by [Hanson and Kegl1987]. In this scheme, each response vector y_i of y is set to x_i, the corresponding input vector. In other words, the network is trained to reproduce the input at the output layer. Such a network is called an autoassociator and was originally introduced in [Rumelhart *et al.*1986]. For concept-learning to take place, the network is trained to reconstruct concept data only and classification is performed on a new vector x^{Test} by comparing its *reconstruction error* to a *threshold*, as described in Section 2.1.2 on single hypothesis testing, and assigning it to the concept class if the reconstruction error is smaller than this threshold and to the other class otherwise. In other words, once all the w_{ji} and w_{kj} have been learned from the concept data available for training, classification is performed as follows:

- x^{Test} is assigned to the concept class, if $\Sigma_{i=1}^{d}[x_i^{Test} - f_i(x^{Test})]^2 < Thresh$

- x^{Test} is assigned to the counter-concept class, otherwise.

$\Sigma_{i=1}^{d}[x_i^{Test} - f_i(x^{Test})]^2$ is called the reconstruction error of the test example and $Thresh$ represents the threshold. The idea behind this recognition-based classification scheme is that since the autoassociator is trained to compress and decompress examples of the concept class only, when tested on a novel data point, it will compress and decompress it appropriately if this example belongs to the concept class, but it will not do so appropriately if the example does not belong to the concept class. The system will be discussed in more detail in Chapter 3. The next section describes the backpropagation procedure used by the two feedforward schemes just described.

2.3.2 Weight Optimization by Backpropagation

As mentioned in the previous section, neural networks are useful since, rather than having to be hand-coded, their weights can be computed adaptively, given a set of training examples. This is done by minimizing a cost function. The particular cost function used throughout this dissertation is the sum of squares error defined as follows[4]:

$$E = 1/2 \times \Sigma_{n=1}^{N}\Sigma_{k=1}^{K}[y_k^n - f_k(x^n)]^2$$

where N is the total number of training examples and K, as before, the total number of output units.

Since the sum of squares error is a differentiable function of the network outputs, then it is also a differentiable function of the weights. The derivatives of

[4]In this formula, the cost function is multiplied by the constant $c = 1/2$ in order to simplify its derivation.

the error with respect to the weights can, therefore, be evaluated and used to find weight values which minimize the error function, by using either gradient descent or a more powerful optimization method (see [Bishop1995]). In this dissertation, the optimization method used is simple gradient descent implemented via the error backpropagation algorithm [Rumelhart *et al.*1986]. This procedure is now described in detail.

Backpropagation: The Delta Rule

The backpropagation algorithm[5] works by applying the gradient descent rule to a feedforward network. The algorithm is composed of two parts that get repeated over and over until a pre-set maximal number of *epochs*, EP_{Max}, has been reached. During the first part of the algorithm—the *feedforward* pass—, the activation values of the hidden units followed by those of the output units are computed. During the second part—*the backpropagation* pass—, the weights of the network are updated—starting with the hidden to output weights and followed by the input to hidden weights—with respect to the sum of squares error through a series of weight update rules called the *delta rule.*

For the hidden to output connections, the delta rule is very simple since the error made at the output layer—the forward layer adjacent to the hidden layer—is known directly. The formula for these weight updates is given below:

$$\begin{aligned}\Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}} \\ &= \eta \Sigma_{n=1}^{N} [y_k^n - f_k(x^n)] g'(h_k^n) V_j^n \\ &= \eta \Sigma_{n=1}^{N} \delta_k^n V_j^n\end{aligned}$$

with

- η corresponding to the learning rate, which is an extra parameter of the network,
- $h_k^n = \Sigma_{j=0}^{M} w_{kj} V_j^n$,
- $V_j^n = g(\Sigma_{i=0}^{d} w_{ji} x_i^n)$, and
- $\delta_k^n = g'(h_k^n)(y_k^n - f_k(x^n))$.

For the input to hidden connections, the rule is not as simple since the activations of the hidden units—contained in the forward layer adjacent to the input layer—cannot be compared to any pre-fixed expected values—as the output units can. Therefore, although the updates are estimated in the same way as in the previous case, this time, the weights are more deeply embedded and the chain rule must be used in the derivative formula. The formula for these weights is thus the following:

---

[5] Please note that the notational conventions used in this section are the same as the ones used in the previous section.

$$
\begin{aligned}
\Delta w_{ji} &= -\eta \, \frac{\partial E}{\partial w_{ji}} \\
&= -\eta \Sigma_{n=1}^{N} \, \frac{\partial E}{\partial V_j^n} \frac{\partial V_j^n}{\partial w_{ji}} \\
&= \eta \Sigma_{k,n} [y_k^n - f_k(x^n] g'(h_k^n) w_{kj} g'(h_j^n) x_i^n \\
&= \eta \delta_k^n w_{kj} g'(h_j^n) x_i^n \\
&= \eta \Sigma_{n=1}^{N} \delta_j^n x_i^n
\end{aligned}
$$

with

- $h_j^n = \Sigma_{i=0}^{d} w_{ji} x_i^n$,
- $\delta_j^n = g'(h_j^n) \; \Sigma_{k=1}^{K} w_{kj} \delta_k^n$,
- and all the other quantities already defined.

Backpropagation: The Algorithm

Having stated the formulae for weight updating, the backpropagation algorithm can be defined, step by step, as:

1. Initialize the weights to small random values, create a random pool of all the training patterns, and set EP, the number of epochs of training to 0.

2. Pick a training pattern μ from the remaining pool of patterns and propagate it forward through the network.

3. Compute the deltas, δ_k^μ for the output layer.

4. Compute the deltas, δ_j^μ for the hidden layer by propagating the error backward.

5. Update all the connections such that

   $$
   \begin{aligned}
   w_{ji}^{new} &= w_{ji}^{old} + \Delta w_{ji} \text{ and} \\
   w_{kj}^{new} &= w_{kj}^{old} + \Delta w_{kj}
   \end{aligned}
   $$

   using the formulae for Δw_{ji} and Δw_{kj} stated above.

6. If any training pattern remains in the pool, then go back to Step 2. If all the training patterns in the pool have been used, then set $EP = EP + 1$, and if $EP < EP_{Max}$, then create a random pool of patterns and go to Step 2. If $EP = EP_{Max}$ then stop.

Backpropagation: The Momentum

As just presented, the backpropagation procedure has the disadvantage of being too slow if η is small and can oscillate too widely if η is large. In order to palliate these problems, a common technique consists of adding a momentum term [Plaut *et al.*1986]. The idea of the momentum is to give each connection (between the hidden and output layer and between the input and hidden layer) some inertia, forcing it to change in the direction of the average downhill "force". This scheme is implemented by giving a contribution from the previous time step to each weight change so that the Δ formulae from the above rules and algorithm are replaced by

$$\Delta w_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \alpha \Delta w_{pq}(t)$$

p and q are any input and hidden, or, hidden and output units; t is a time step or epoch; and α is the momentum parameter of the network which regulates the amount of inertia of the weights. Using a momentum makes it possible to use a larger learning rate without leading to divergent oscillations.

2.4 Generalization in Inductive Systems

At the end of Section 2.3.1, it was mentioned that no inductive scheme could be guaranteed to work, and this is because, ultimately an inductive system's solution is nothing other than a guess as to how the training data was generated. Of course, this guess is educated in that it relies on past information to predict the value of future patterns. While in certain cases, such an approach does work, it does not necessarily do so always since on the one hand, the future may happen to diverge from the past, and, on the other hand, there are many different ways of extrapolating from the past and no guarantee that a function consistent with past data will predict the value of future data effectively. As far as generalization is concerned, this dissertation is not concerned with the first possibility since there is no way to guard against the eventuality that a common occurrence stops taking place or starts happening in a different fashion. Rather, it is concerned with the second point: how can an inductive system choose a good approximation scheme from the multitude of ways in which one can extrapolate or generalize from a given set of observations? In the first part of this section, I discuss generalization in neural networks and in the second part, the discussion is extended to more general inductive schemes. The generalization framework laid out in this section will be used throughout the dissertation to analyze the autoassociation-based classification scheme and to compare it to MLP.

2.4.1 Generalization in Neural Networks

Generalization, the fundamental property of a learning agent, can be thought of as the ability to extract, from a large number of observations, the most relevant

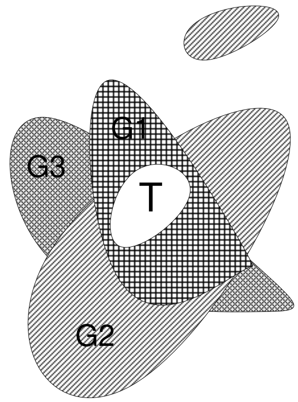


Figure 2.2: This figure shows that given a target concept T, many generalizations consistent with this target are possible, including $G1, G2$, and $G3$.

common features that characterize the different classes of these observations. As illustrated by [Denker *et al.*1987], given a set of observations, there exist a large number of generalizations (see figure 2.2). More precisely, [Denker *et al.*1987] points out that for every problem, the number of possible generalizations can be computed as:

$$\text{Number of Generalizations} = \frac{(2^k)^{2^d}}{(2^k)^N}$$

where k is the number of output units, d is the number of input units and N is the number of training examples. For example, they calculated that for a reasonable task involving a feedforward network of 30 input bits, a single output bit and 1,000 training examples, there are $2^{(10^9)}$ possible generalizations!

In order to deal with the problem of picking an appropriate generalization efficiently from the set of all possible generalizations that can be inferred by a Neural Network, [Denker *et al.*1987] proposes to restrict its attention to the class of *Network Efficiently Representable Functions*, or NERFs, which can represent a certain class of simple boolean functions using a total number of hidden units that grows only polynomially in the dimensionality d of the data. In other words, they propose to restrict (or *bias*) the search for the approximating function of the observed data to a small subset of the space of all possible such approximations. Noting that a large number of useful functions do belong to the class of NERFs, [Denker *et al.*1987] conclude that Neural Networks are appropriate for many applications.

While the above discussion is important for pointing out the idea of restricting the generalization language of an inductive system, it does not say how the set of all functions that can be represented by this restrictive language is to be searched for the best approximation by a neural network. The next section discusses the issue of generalization search, expanding the focus of this discussion to a more general class of inductive systems. The study reported in Chapter 6 shows how a neural network can go about searching the generalization space.

2.4.2 Mitchell's Framework for Generalization

This section introduces [Mitchell1982]'s work on Version Spaces focusing on the parts of his theory that are relevant for this dissertation. In order to formalize the concept of inductive learning, like [Denker *et al.*1987], [Mitchell1982] uses the idea of a restricted (or *biased*) pool of possible generalizations to be searched in the form of a restricted language in which generalizations can be expressed. In addition, he assumes that this generalization language implements a partial ordering of all the possible generalizations possible so that the induction problem reduces to a search through the "ordered" space of all possible generalizations defined within the particular language considered.

In more detail, [Mitchell1982] defines the notion of Concept Learning as follows[6]:

Definition (from [Mitchell1982])

"*Given:*

(1) A language in which to describe instances.

(2) A language in which to describe generalizations.

(3) A matching predicate that matches generalizations to instances.

(4) A set of positive and negative training instances of a target generalization to be learned.

Determine:

Generalizations within the provided language that are *consistent* with the presented training instances."

In this definition, a generalization is *consistent* with a set of training instances if and only if it matches every positive instance and no negative instance in the set.

The central contribution of [Mitchell1982]'s work is that it provides a rigorous framework to describe how the determination of a generalization consistent with a training set can be achieved. In particular, Mitchell starts by suggesting that the partial ordering of all possible generalizations could be based on the *more-specific-than* relation defined as follows:

[6] In the field of machine learning from which [Mitchell1982]'s work stems, a positive example refers to an example of the concept whereas a negative example corresponds to a counter-example. These terms will be used interchangeably in the remainder of the dissertation

> Given two generalizations, $G1$ and $G2$, $G1$ is *more-specific-than* $G2$ if and only if $G1$ matches a proper subset of the instances that $G2$ matches.

Using this definition, he describes several data-driven generalization methods, two of which prompted him to define an important framework for generalization which formalizes the concept class in a universal manner. It is this framework that will be used throughout the dissertation.

Mitchell's framework defines the space of all generalizations consistent with a training set using the notion of a *version space*. A version space is defined in terms of two sets S and G which represent the limits of the space of all generalizations. In more detail, S represents the set of all maximally specific hypotheses and is defined as follows (from [Mitchell1982]):

> "$S = \{s|s$ is a generalization that is consistent with the observed instances, and such that there is no generalization which is both more specific than s and consistent with the observed instances.$\}$"

G is the dual of S in that it represents the set of all maximally general hypotheses and is defined as follows:

> "$G = \{g|g$ is a generalization that is consistent with the observed instances, and such that there is no generalization which is both more general than g and consistent with the observed instances. $\}$"

A generalization, x, is said to belong to the version space represented by S and G if and only if

> "(1) x is more specific than or equal to some member of G, and
> (2) x is more general than or equal to some member of S."

The search for an appropriate generalization within the version space associated with a given data set and delimitated by S and G is performed bi-directionally and was depicted in Figure 1.2 of Chapter 1. This bi-directionality illustrates very well the sort of competition taking place during the learning process: a learning agent has to balance its search so as to find a generalization that is consistent with both the positive and the negative instance sets.

It is important to note that in this framework, a strong emphasis is given to the negative data set without which the version space theory could not hold since the specialization mechanism of the generalization process is driven by the counter-examples. This means that within [Mitchell1982]'s framework, the mere idea of concept-learning in the absence of counter-examples seems incongruous since this framework suggests that in such systems generalization would be the only thrust of the process and no specialization could take place.

However, as was already mentioned and as will be shown in Chapter 4, this dissertation demonstrates that concept-learning can occur in the absence of counter-examples. The mismatch between [Mitchell1982]'s framework and the observations regarding autoassociation-based classification is resolved by refining the framework, realizing that specialization is not necessarily caused by counter-examples only, but that it can be caused by other sources of prior statistical or contextual information about the domain, such as in the recognition-based techniques introduced in section 2.3, or by certain biases of the inductive process, as in the case of "view-based eigen-spaces" ([Turk and Pentland1991], [Pentland *et al.*1994]) and autoassociation.

The questions regarding the nature of the biases used by "view-based eigen-spaces" technique and the autoassociator in order to perform classification is important as confirmation that a system's bias can be sufficient for induction to take place, as a first step towards understanding the search through the generalization space performed by the autoassociator, and as a first step towards understanding the domain characteristics for which the autoassociator is most appropriate for classification. Since these questions constitute the core questions asked in this dissertation, [Mitchell1982]'s framework is central to this work.

2.5 Experimental Strategy

This section discusses the experimental strategy used in this dissertation. For the most part, the experiments that were conducted in the study consist of comparing the concept-learning phases of different classifiers, and, more specifically, the concept-learning phases of autoassociation-based classification and MLP. Such comparisons, however, entail three sets of concerns that must be addressed prior to running the experiments. First, while autoassociation-based classification is a recognition-based classifier, MLP is a discrimination-based one; is it thus possible to compare two classifiers that approach the same problem in such different ways? Second, as discussed previously, concept-learning, while essential, is not the only phase that constitutes the binary learning process: it is usually preceded by a preliminaries-setting phase, followed by a threshold-determination phase, and its performance is usually assessed in view of these three phases. How then can the concept-learning phase of a binary learning process be evaluated and compared to that of another binary learning process independently of the other phases? Is it useful to focus on such comparisons? Finally, there are different ways to evaluate the performance of classifiers. What then constitutes a reliable performance estimation measure for the binary learning processes of this dissertation? These three sets of concerns are addressed in this section which concludes the chapter on background issues.

2.5.1 Comparing Recognition- versus Discrimination- Based Concept-Learning

The question of whether recognition-based and discrimination-based concept learners can be compared meaningfully is easy to address once the distinction between preliminary setting, concept-learning and threshold-determination has been established for both recognition- and discrimination- based binary learning, as it was done both for hypothesis testing and neural networks in this chapter. Indeed, after having been trained, recognition-based concept-learners issue series of different signals that can be interpreted as "recognized" or "unrecognized". Similarly, trained discrimination-based concept-learners issue series of signals that can be interpreted as "Class A" or "Class B". By unifying the "recognized" category with "Class A" and the "unrecognized" category with "Class B" (or vice-versa), we are ensuring that the two approaches to binary learning are functionally equivalent, even though they use different learning strategies and require different types of training data. Comparing them thus becomes possible.

2.5.2 Evaluating Concept-Learning independently of Preliminary Setting and Threshold-Determination

The questions of whether concept-learning phases can be compared independently of the other phases of the binary learning process, and of whether such comparisons are useful, are more difficult to address. I will begin this section by discussing how concerns about the preliminary-setting phase can be eliminated. I will continue by addressing the question of how to eliminate the bias introduced by the threshold-determination component. Finally, I will conclude the section by discussing the issue of whether limiting the comparison of two classifiers to the comparison of their concept-learning phase is useful.

2.5.2.1 Eliminating Preliminary Setting Biases

The approach I use to eliminate the bias introduced by the preliminary-setting phase of the binary learning process follows a best-case strategy. Indeed, in order to compare two concept-learners fairly, it is necessary that they be compared according to the same criteria. Since setting the parameters of different concept-learners to the same values is not an option because these parameter values may be optimal for one device but not the other, my strategy consisted of finding the optimal settings for each system within the same space of possibilities. The search space was restricted to the same set for the different systems to ensure the equivalence of the parameter searching process and this policy was implementable in my experiments since the systems I compared belong to the same family of connectionist systems.

I begin this discussion by overviewing the nature of the parameter setting process necessary for neural networks. In particular, I introduce the Bias/Variance Dilemma and explain how it is typically handled when dealing with neural networks. I subsequently discuss the actual strategy that was used in my experiments.

The Bias/Variance Dilemma and Neural Networks

Feedforward neural networks belong to an important class of estimators called non-parametric. As discussed in Section 2.1, such estimators do not make any assumption regarding the density being estimated, but instead, "let the data speak for itself".[7] Many nonparametric approaches, including feedforward neural networks, have the advantage of possessing the *universal approximation property* which allows them to approximate as closely as desired any function, as long as a sufficient number of parameters is included in the model. While useful because of this property, such approaches, nevertheless, also suffer from a problem known as the *Bias/Variance Dilemma*.

I will now describe the Bias/Variance Dilemma following [Geman *et al.*1992], but in order to do so, it is first necessary to introduce a notation slightly different from the one used in Section 2.3.1. In this notation, I refer to $D = (x^1, y^1), (x^2, y^2)$, ... (x^N, y^N), as the training data set, and I refer to the induced function as $f(x; D)$ rather than simply $f(x)$ so as to underline the dependence of f on D.

It can be easily seen that $E[y|x]$, the regression of y on x (also expressed as $Prob(y =< SomeValue > |x)$) is, among all functions of x the best predictor of y given x in the mean squared sense[8]. Indeed,

$$\begin{aligned} E[(y - f(x))^2|x] &= E[((y - E[y|x]) + (E[y|x] - f(x)))^2|x] \\ &= E[(y - E[y|x])^2|x] + (E[y|x] - f(x))^2 \\ &\quad +2E[(y - E[y|x])|x].(E[y|x] - f(x)) \\ &= E[(y - E[y|x])^2|x] + (E[y|x] - f(x))^2 \\ &\quad +2(E[y|x] - E[y|x]).(E[y|x] - f(x)) \\ &= E[(y - E[y|x])^2|x] + (E[y|x] - f(x))^2 \\ &\geq E[(y - E[y|x])^2|x] \end{aligned}$$

It follows from this observation that the mean squared error of f as an estimator of the regression $E[y|x]$:

$$E_D[(f(x; D) - E[y|x])]^2$$

is a good measure of the effectiveness of f as a predictor of y. (E_D is the expectation with respect to the training set, D, that is, the average over the ensemble of possible Ds, for a fixed sample size N).

In addition to being representative, this measure is particularly expressive since it can be decomposed into two terms: the *bias* and *variance*, as follows (the derivation uses the same trick as the one above of adding and subtracting an extra term and regrouping the expressions):

[7] Note that neural networks can be thought of as non-parametric before their number of hidden units is set. Once it is set, they should be considered parametric. This is why they are sometimes referred to as *semi-parametric*. In this section, neural networks are considered prior to the setting of a fixed number of hidden units and are thus considered as non-parametric.

[8] The mean squared error is simply the sum of squares error defined in Section 2.3.2 divided by the number of observations, N.

$$\begin{aligned}
E_D[(f(x;D)-E[y|x])^2] &= E_D[((f(x;D)-E_D[f(x;D)]) \\
&\qquad +(E_D[f(s;D)]-E[y|x]))^2] \\
&= E_D[(f(x;D)-E_D[f(x;D)])^2] \\
&\qquad +E_D[(E_D[f(x;D)]-E[y|x])^2] \\
&\qquad +2E_D[(f(x;D)-E_D[f(x;D)]) \\
&\qquad\qquad (E_D[f(x;D)]-E[y|x])] \\
&= E_D[(f(x;D)-E_D[f(x;D)])^2] \\
&\qquad +(E_D[f(x;D)]-E[y|x])^2 \\
&\qquad +2E_D[f(x;D)-E_D[f(x;D)]]. \\
&\qquad\qquad (E_D[f(x;D)]-E[y|x]) \\
&= (E_D[f(x;D)]-E[y|x])^2 \qquad\qquad \text{“bias”} \\
&\qquad +E_D[f(x;D)-E_D[f(x;D)])^2] \quad \text{“variance”}
\end{aligned}$$

In other words, it can be said that the mean squared error of an estimator f is minimized if it has both a small bias and a small variance. This remark applies to a large number of estimators, and in particular, to the neural networks used in this dissertation since they too use the mean squared error function (times N) as the criterion to minimize.

Unfortunately, there is a typical tradeoff between the bias and variance contributions to the estimation error: in general, a decrease in variance yields a large bias whereas a decrease in bias yields a large variance ([Geman *et al.*1992]) . While parametric methods often suffer from a large bias term, they typically have the advantage of not having a large variance. Conversely, nonparametric approaches including feedforward neural networks do not have a large bias but suffer from large variance.

Nevertheless, the bias/variance tradeoff can be *regularized* in different ways. In the context of Neural Networks, the two simplest ways are by[9]:

- Controlling the capacity of the network
- Controlling the number of epochs for which the neural networks are trained.

Controlling the capacity of the network consists of increasing or decreasing the number of hidden units present. While reducing the number of hidden units reduces the variance of the system and increases its bias, increasing the number of hidden units causes the opposite effect. Similarly, reducing the network's training time might cause an increase in the system's bias, while increasing it might cause an increase in variance. Note that these phenomena are also known under the terms *underfitting* and *overfitting* where underfitting corresponds to the case of a large bias and small variance while overfitting corresponds to the case of a small bias and large variance.

[9]More sophisticated ways include the addition of bias constraints ([Geman *et al.*1992]) such as the addition of a reconstruction constraint ([Stainvas1999]), as well as other unsupervised constraints seeking some structure in the data ([Intrator1993]).

As will be discussed next, both the number of hidden units and the number of training epochs of the connectionist systems considered have been regulated in all the experiments conducted in this dissertation. This regularization was done as part of the preliminary setting phase prior to concept-learning.

Setting the Networks' Parameters

As discussed earlier, my preliminary-setting strategy for tuning the parameters (number of hidden units and stopping times) of the concept-learners of my experiments consisted of finding the optimal parameters of each concept-learner within the same range of possibilities, and only with those parameters, comparing and contrasting the performance of the different systems. This general preliminary-setting strategy was repeated for each experiment (i.e., for each system and each domain) so that the comparisons would be fair.

Since the neural networks tested in this thesis are closely related, searching for optimal parameters within the same range is reasonable. In particular, I started by looking for an optimal capacity by running the networks with 1, 2, 4, 8, 16, 32 and 64 hidden units. In the experiments involving real world domains, I refined my search further by searching the hidden unit interval that had yielded the best performance more deeply. For the artificial domain, I restricted my search to these values. The stopping point of each network was selected by running the networks past their point of convergence and choosing a stopping point located within a stable region of the error curve and prior to the time when overfitting started.

2.5.2.2 Eliminating Threshold Determination Biases

In order to eliminate the biases introduced by the use of a threshold-determination component, it is best not to use the best-case strategy of the previous section. Indeed, such a strategy would be able to assess the comparative performances of overall classifiers (including their parameter setting, concept-learning and threshold-determination phases), but would not be able to assess the comparative performances of their concept-learners alone. This is because, if a best-case strategy were used, the information concerning the relative degrees of involvement of the concept-learners and their attached threshold-determiners in the classification process would be unavailable. Instead, the strategy for comparing several concept-learners consists of eliminating the need for a threshold-determination altogether, using the ROC Analysis technique. If such a strategy is impractical, then, the same theshold-determination component should be used with all the compared systems. This section discusses these two strategies.[10]

[10]Please, note that in one case (See Section 4.5.3 of Chapter 4), I purposely selected not to compare the concept-learners of my systems alone, but the overall classifiers. The classifiers in question were the autoassociator together with the parametric recognition threshold-determination component and the MLP network together with a parametric discrimination-based threshold-determination technique. The purpose of this experiment was to show that,

ROC Analysis

The optimal strategy for comparing the performance of two concept-learners is to perform a series of ROC Analyses. ROC Analysis proceeds by comparing the different concept-learners over the range of all possible thresholds. Since all possible thresholds are considered by the different concept-learners, no bias is introduced by this method. ROC Analysis will be described in more detail in Section 3.3 of Chapter 3.

This method can be used in one of two ways. Either, as an overall unbiased performance measure of different concept-learners or as an actual (but biased) threshold-determination component. In this work, it was used in the first way, leaving the issue of threshold-determination to the two components discussed in the next paragraph. ROC analysis was thus primarily used in Chapter 7 where comparisons of the concept-learners are more interesting than the actual classification rates of each classifier.

Using the *Same* Threshold-Determination Component

When classification rates were sought rather than overall performance measures, a slightly less optimal, but still acceptable solution consisted of comparing the classification accuracy results of binary learning systems using *the same* threshold-determination component.[11] In such cases, one of the three threshold-determination methods designed in this dissertation—*Limit Search*, *Binary Search*, and *Parametric Recognition*—was selected and used by the different systems compared. Limit search is a heuristic method that requires data of a single class. It has the advantage of not requiring any prior knowledge about the data distribution or the task at hand; binary search is also a heuristic method, but it requires both examples and counter-examples of the concept class. Parametric recognition is a statistical method that requires data of a single class. Its advantage is to be conceptually straightforward, but its disadvantage is to be parametric.

Note that each of these techniques serves a different purpose. Binary search was designed to deal with cases where data of a single class did not appear sufficient to set an accurate boundary between the concept and counter-concept data. Limit search was designed to demonstrate that, in certain cases, data of a single class were sufficient to find such a boundary. And, parametric recognition was

---

even when the recognition-based learning system has the disadvantage of using only examples of the concept class in *both* the concept-learner and the threshold-determination component, it obtains a classification performance comparable to that of its discrimination counterpart which uses counter-examples in both its concept-learning and threshold-determination phase.

[11]This strategy is less optimal than ROC Analyses in cases where the threshold-determination component used by the concept-learners being compared is biased in favor of one of the concept-learners and not the others. This situation, however, is unlikely to have much effect in this dissertation, since, as discussed in the next section, when using connectionist concept-learners, the role of the threshold-determination component is minimal.

designed as an easy way to deal with the threshold-determination issue in the absence of counter-examples. Limit search and parametric recognition can be seen as a first step towards the complete elimination of counter-examples from the recognition-based binary learning process. The three techniques will be described in great detail in Section 3.3 of Chapter 3.

2.5.2.3 On the Fruitfulness of Comparisons between Concept-Learners Alone

The strategy used in most of the experiments of this dissertation consisted of comparing the concept-learning phase of different classifiers. However, as discussed earlier, the classification process of a classifier depends not only on its concept-learning phase, but also on its preliminary setting and threshold-determination phases. Given this observation, it is reasonable to ask whether there is a point in comparing the performance of concept-learners alone or whether it would be more useful to concentrate on the overall classifiers. The answer to this question is that there is an interest in comparing concept-learners alone, and that, for two reasons.

First, the design of a concept-learner and of a threshold-determination component[12] are two distinct and complex problems that would be difficult to treat simultaneously. Second, in the particular case of neural networks, most of the classification process is carried out by the concept-learning phase rather than by the threshold-determination phase. This is different from other types of classifiers for which the threshold-determination phase may fulfill a more crucial role. For example, while in a connectionist system, the purpose of the threshold-determination component is simply to *interpret* the value of the output ("0", "1", "a value greater than 0"), in other systems, such as hypothesis testing classifiers, its purpose is to *analyze* the outcome of the concept-learning phase and actually *choose* a boundary. The lack of importance of the threshold-determination component in neural networks is illustrated by the fact that researchers in the field do not usually design a threshold-determination component to accompagny their classification method, but instead, inspect the result of their network manually.

Given these two facts, it was natural for my dissertation to focus on the concept-learning component of the autoassociation-based classifier and its counterpart, the MLP network, leaving the careful design and optimization of their threshold-determination components to a future time.[13]

---

[12]The preliminary-setting phases of the concept-learners are currently taken into consideration since the parameters get optimally set. Therefore, only the lack of consideration of the threshold-determination component is necessary to address in this discussion.

[13]The different threshold-determination components designed in this dissertation were not optimized.

2.5.3 Reliable Error Estimation

An additional issue to consider when comparing two classifiers is the evaluation of the classification error. Following [Fukunaga1990], let us consider how to bound the error of a classifier. The reason why this error has to be bounded rather than calculated directly is that, for direct calculation, the entire distribution of the data would have to be known. However, this distribution is rarely fully known. Therefore, only a finite (and often small) data set is usually available for bounding or estimating the error. In addition, of course, this same data set needs to be used to design (or train) the classifier. The formula for bounding the error of a classifier is:

$$E_R \leq E_{True} \leq E_H$$

where E_{True} is the true error of the classifier, E_R is the *resubstitution* error and E_H is the *holdout* error. This formula states that the true error of a classifier can be bounded from below by the resubstitution error and from above by a holdout error. The resubstitution error is obtained by testing the classifier on the set it was trained on, while the holdout error is obtained by testing the classifier on a data set it has never seen before.

In practical applications, the resubstitution error is rarely used because it is much too optimistic. The holdout method, on the other hand, is appropriate. However, while this error can be calculated directly if the data sets are generated artificially by a computer—since arbitrarily large test sets can then be constructed—, the situation is not as straightforward in real-world domains. In such domains, typically, a single set of data is available, and often, this data set is rather small and difficult to monitor. More specifically, when dividing a finite real-world domain data set into two independent groups, three types of problems may occur:

- The number of samples in the training and testing sets may be too small
- Creating two data sets (one for training and one for testing) with close distribution is not easy
- Selecting an appropriate size for the training and testing set is problematic

For all these reasons, procedures other than the holdout method are often used to alleviate these problems: the *leave-one-out* and *cross-validation* methods.

In more detail, in the leave-one-out method, the entire set minus one sample is used to train the classifier which is then tested on this sample. The sample is thrown back in the pool and the next sample is excluded. The operation is repeated N times until all N samples have been tested. Then, the number of misclassified samples is counted to obtain the estimate of the error. A disadvantage of this method, however, is that it requests the training of N different classifiers. This is very time consuming, especially when dealing with Neural Networks which incur a large design time. The problem has been overcome, however, by using a method called *cross-validation*. Cross-validation works in the same way as leave-one-out, except for the fact that instead of being removed one by one for testing, samples are removed in groups of typical size N/10 or N/5.

In the experiments conducted on real-world domains in this dissertation, the five-fold cross-validation method (N/5) was selected. It was applied by dividing the training set into five subsets, four of which being used for training and the last one being used for testing in every case. The training sets were further divided into a concept-learning training set and a threshold-determination training set. The threshold-determination training set was typically very small, since it contained either 5 concept-class and 5 counter-concept-class points or 10 counter-concept-class points. In all case, the same partitions and subpartitions were used to test all the systems.

In the artificial domain, since it is possible to generate as many training examples as desired, the holdout method could be used directly without recourse to five-fold cross-validation. To increase the reliability of the results, however, each experiment was repeated five times on different training and testing sets generated from the same source.

Chapter 3
Autoassociation-Based Classification

The autoassociation-based classification scheme was briefly introduced in Section 2.3.1 of Chapter 2. This chapter presents a more thorough description of the process which constitutes the focus of this dissertation. It begins by discussing the cognitive science and neuroscience works from which it emerged in an attempt to illustrate the cross-fertilization between the natural and the artificial sciences. It then offers a detailed description of the system's two components—the concept-learning and the threshold-determination components—and the system's overall organization. In particular, the four implementations of threshold-determination introduced in Section 2.5.2 and used in the dissertation will be described in much detail.

3.1 Neuroscience and Cognitive Science Considerations

Although the primary goal of artificial intelligence is not necessarily to emulate human beings or animals, research in artificial intelligence can benefit greatly from advances made in the Cognitive Sciences. For instance, it is often possible for artificial intelligence researchers to get their inspiration from what is known about human or animal cognition. Similarly, cognitive scientists can benefit from studies made in artificial intelligence. The purpose of this section is to underline the bi-directional connection of the artificial intelligence study of this dissertation to the cognitive sciences. In more detail, by describing the natural science genesis of autoassociation-based classification, this section attempts to show both how the cognitive and neuro sciences inspired artificial intelligence, and how, in return, the results obtained in artificial intelligence studies such as those in this dissertation should be of interest for cognitive scientists.

The section begins by summarizing Hanson and Kegl (1987)'s use of the autoassociator for learning a natural language grammar and then describes Gluck and Myers (1993)'s neuroscience theory of the hippocampus. The actual classification device is described in the following sections.

3.1.1 Hanson & Kegl's Natural Language Learning System

The work of [Hanson and Kegl1987] takes as point of departure the linguistic theory which suggests that simple exposure to language is not sufficient to acquire a natural language grammar but that, instead, the language acquisition process is facilitated by a large innate component whose parameters can be set during the acquisition process. Although their work does not contradict this nativist point of view, it refines it in an attempt to determine the aspects of syntax that can actually be learned inductively. In particular, [Hanson and Kegl1987] built an autoassociator-based system, PARSNIP, which learns how to reconstruct correct sentences at the output layer using the back-propagation procedure [Rumelhart *et al.*1986], just as described in Section 2.3.1 and as will be described in more depth below, in section 3.2. PARSNIP was exposed to sentences from the Brown Corpus in which each word of text is associated with a tag which indicates its syntactic category. The results obtained by PARSNIP indicate that the system is able to generalize to correct sentences to which it was not exposed during the training period, as well as to reject sentences that violate natural language constraints. In effect, the system acts as if it has acquired syntactic rules during the training period.

Although the model is not a plausible model for language acquisition since it is not reasonable to assume repeated parallel exposure to a sentence with enforced production of that sentence, there are some similarities between a child's exposure to language and PARSNIP's. In particular, both learning "systems" are exposed to grammatically correct natural language sentences only (except for bad grammatical usage that can be thought of as noisy data) and asked to induce general rules and larger constituents from the regularities to which they are exposed. This similarity to cognition was reinforced by the Neuroscience theory of [Gluck and Myers1993] which seems to indicate that the computations taking place in the autoassociator during the learning phase are actually related to certain procedures observed in the brain.

3.1.2 Gluck & Myers' Neuroscience Theory

Although the hippocampus and adjacent cortical region are generally acknowledged to play a fundamental role in learning and memory, little consensus has emerged as to the precise specification of this role. Lesion data from humans and animals suggest that some learning tasks may not be solvable without an intact hippocampal region [Gluck and Myers1993, Squire1987].

Gluck and Myers (1993) study this region, starting with a computational theory of discrimination learning in intact animals and then seeking to identify a subcomponent of this theory that depends on the hippocampal region. The theory assumes that the hippocampal region develops new stimulus representations that enhance the discriminability of differentially predictive cues while compressing the representation of redundant cues. To develop their theory, they

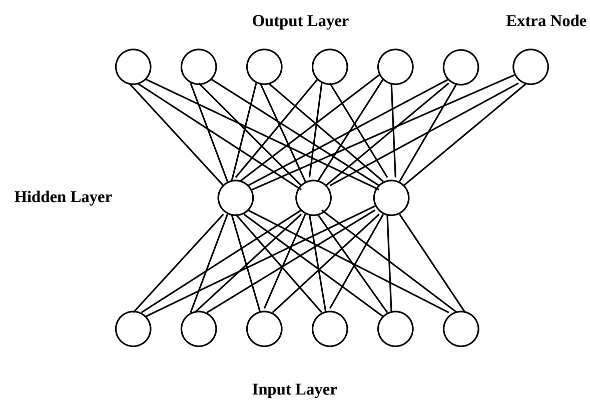


Figure 3.1: A Predictive Autoencoder with $I = 6$ and $H = 3$.

used connectionist network models as a formal framework for characterizing theories of associative learning. Within this framework, they developed and tested a trial-level connectionist theory of cortico-hippocampal interaction in classical conditioning.

Gluck and Myers built a model of the hippocampus that is a small variation of the autoassociator. Their model is called a *predictive autoencoder* (or *predictive autoassociator*)[1] and consists of an autoassociator augmented with the constraints that after having been trained on examples of different classes, together with its reconstruction, it must output a classification for the input pattern. This augmentation is implemented with an extra node on the output layer that is connected to all the nodes of the hidden layer and whose function is to indicate the class of the example under consideration. Figure 3.1 presents a predictive autoassociator with $I = 6$, $O = 7$, and $H = 3$. Like the autoassociator studied in this work, the predictive autoencoder architecture has linear input units and nonlinear hidden and output units, where nonlinearity is implemented using the logistic function, $f(x) = \frac{1}{1+e^{-x}}$.

Gluck and Myers chose to model their cortico-hippocampal theory using a predictive autoassociator because such a network implements the two aspects of their theory. Indeed, a predictive autoassociator has a narrow hidden layer which forces it to generate an internal representation that compresses redundancies in the input pattern in order to reproduce itself at the output layer, thereby matching the part of the theory that assumes compression of redundant cues in the new stimulus representation. As well, the narrow layer of the predictive autoassociator must also differentiate representations of input features that are especially useful in predicting the outcome, thereby matching the part of the theory that assumes enhancement of the discriminability of differentially predictive cues in the new stimulus representation.

[1]The terms "autoassociator" and "autoencoder" have been used interchangeably in the literature although the first one is commonly linked to cognitive theory while the second was introduced in the context of information theory.

By matching the available experimental data on learning in hippocampal lesioned animals, [Gluck and Myers1993]'s computational model allowed them to argue that their theory provides a simple and unified interpretation of the functional role of the hippocampus region in a wide range of conditioning paradigms [Gluck and Myers1993, Myers and Gluck1994]. The theory also makes novel predictions regarding the effects of hippocampal lesions for several additional training paradigms.

Faithful to the interplay between the natural and the artificial sciences, it is interesting to note that a similar architecture has been tested in the context of the engineering problem of face detection in [Stainvas1999]. Rather than neuroscience-inspired, the architecture of [Stainvas1999] is derived from purely statistical considerations, though those considerations might turn out to be valid from a neuroscience point of view as well. As mentioned in Section 2.5.2 (Footnote 9) of Chapter 2, the addition of a reconstruction constraint to a regular feedforward classifier is one possible way to deal with the Bias/Variance Dilemma discussed by [Geman *et al.*1992] in the context of neural networks. This consideration allowed for the design of a predictive autoencoder similar to the one of [Gluck and Myers1993]. Further statistical considerations, however, allowed [Stainvas1999] to go further than [Gluck and Myers1993]. In particular, she added a regularization parameter λ that controls the tradeoff between reconstruction and discrimination. Furthermore, rather than running single instances of her classifier, she used them within ensembles ([Wolpert1992]) which are known to improve performance of single experts by reducing the contribution of the variance term of the error, as long as the different experts in the ensemble are independent. Some independence was shown to be achieved by using systems with different values for λ within the same ensemble. It would be interesting to find out whether these statistical constraints are biologically valid as well.

The cognitive science and neuro-science theories of [Hanson and Kegl1987] and [Gluck and Myers1993] suggest that processes similar to those taking place in the autoassociator may take place in the brain. The study of this dissertation which explains the operation of the autoassociator in great detail should, therefore, be of interest to scientists interested in explaining the mechanisms of the brain even though its focus is primarily on engineering issues.

3.2 Concept-Learning by Autoassociation

The remainder of this chapter describes the system that will be tested and analyzed throughout this dissertation: the autoassociation-based classifier. This description is composed of three parts: the description of the concept-learning component, the description of the threshold-determination component, and the description of the overall functioning of the system. The concept-learning component was introduced formally in Section 2.3.1. In this section, I focus on a more functional description. The various threshold-determination components attached to this concept-learning component will be described in the next section followed by the overall description of the system.

As mentioned previously, the concept-learning component consists of a single hidden-layered autoassociator such as the one illustrated in Figure 2.1(b) of Chapter 2. In the past, autoassociators have been used for achieving dimensionality-reduction [Saund1989] and in particular, image compression [Cottrell *et al.*1987] and speech recognition [Elman and Zipser1988], for estimating learning algorithm reliability [Pomerleau1993], and for addressing the catastrophic inference problem—the problem of unlearning what was previously learned when new information is presented [Kortge1990]. In this dissertation, I discuss how such devices can also be used for classification, and more specifically, recognition-based classification in the way it was used by [Hanson and Kegl1987] for learning natural language grammars.

In order to be used for recognition-based classification, the autoassociator is trained on positive instances of the concept, using the back-propagation procedure already described in Section 2.3.2. Once trained, the autoassociator can be fed new instances that it tries to reconstitute at its output layer. The quality of reconstruction is evaluated by computing the sum of squares error at each corresponding input and output node. If this error is small, then the instance is labeled "positive" since it is likely to be an instance of the concept; otherwise, it is labeled "negative" since it is likely to be a counter-example. In this dissertation, the part of the system responsible for evaluating the size of the error and labeling the new instances is the automated module called the threshold determination component. Several instances of threshold-determination components are described below.

The intuition behind the learning strategy of the autoassociation-based classifier is that when the autoassociator is tested on novel positive examples, it is expected to reconstruct them well since they involve a pattern similar to those of the positive examples in the training set (i.e., the inputs that are redundant in the positive training data are also redundant in the positive testing data). Since counter-examples present patterns that are different from those of the positive data, the compression which allows the network to reconstruct the positive examples will not suffice for good reconstruction of novel negative patterns. At testing time, therefore, reconstruction of positive data will succeed, whereas reconstruction of negative data will fail, and this success or failure is the criterion used in classifying new instances. The classification technique is thus similar to the one used in "view-based eigen spaces" ([Turk and Pentland1991], [Pentland *et al.*1994]), but the difference between this work and [Hanson and Kegl1987]'s or mine is in the fact that rather than projecting the input data on the nonlinear hidden units of an autoassociator, they project them on the Principal Components of the concept-class data. While it has been sometimes claimed that even with nonlinearities in their hidden units, autoassociators do nothing other than to compute the principal components of the data in a domain ([Bourlard and Kamp1988], [Cottrell and Munro1988]), Chapter 5 will show that this claim is incorrect.

3.3 Threshold Determination

In order to determine whether a novel test example is a positive or a negative instance of the concept, its reconstruction error must be compared against some threshold, dividing the positive from the negative class. As discussed in the context of hypothesis testing (Section 2.1 of Chapter 2), the idea of a threshold-determination component is to learn a boundary between the positive and the negative class from the observation of a number of training data. This section presents the four different strategies implemented by the threshold-determination components used in this dissertation: Limit Search, Binary Search, Parametric Recognition, and ROC Analysis.

3.3.1 Limit and Binary Search

The first and second (related) methods used for threshold-determination that I describe are limit and binary searches through the reconstruction error space of the autoassociators. The goal of these searches is two-fold. On the one hand, they seek an optimal separation between positive and negative examples of the concept and, on the other hand, they try to ensure that this optimal solution is a stable one. The first method presented uses only one class of data while the second uses both concept-class and counter-concept-class examples. The single class method was designed both to demonstrate that threshold-determination can be carried out with a single class of data and no prior statistical information, and because one of the test domains considered did not include a large enough number of examples of the concept class and thus, could not devote a large enough number of positive data to threshold-setting.[2] The second method was used on the other two domains which contained a well-balanced set of concept-class and counter-concept-class data. The two implementations are described next.

Limit Search

In the case where only positive (or negative) training instances of the concept are provided to the threshold-determination component, the procedure simply computes the upper- (or lower-) bound of the reconstruction error of all the positive (or negative) training instances at every epoch considered and then relaxes this bound sequentially by augmenting (or decreasing) it by a fixed percentage, $RelaxationRatio$, at every iteration. New instances are subsequently classified by checking whether the reconstruction error of the new instance is higher (or lower) than that of the relaxed boundary in at least a certain acceptable proportion, $AcceptableProportion$, of the epochs considered (epochs occurring after $MinimumEpoch$). In such a case, the new instance is classified as negative (or

[2] As will be described in Chapter 4, the data used to set a threshold are separate from the data used to train the network.

positive); otherwise, it is classified as positive (or negative). Figure 3.2(a) illustrates the single-case threshold-setting case and shows the boundary that was derived when using only positive data.[3]. The following is the pseudo code for the *LimitSearch* procedure which takes as input

- *TestDatum*, a 1-dimensional array representing the output of the neural network at each epoch recorded, for the testing point, P,
- *PositiveSet*, a 2-dimensional array representing the output of the neural network at each epoch recorded for each point in the positive threshold-determination data set.

It returns the label "positive" or "negative" assigned to the test datum, P.

```
***************************************************************
LimitSearch(TestDatum, PositiveSet)

  For each epoch recorded, E                % Compute Boundary
     Boundary[E]= (1+RelaxationRatio)*
                  max(PositiveSet)
  end %For%

PosCount= 0
NegCount= 0
For every epoch E > MinimumEpoch     % Compute classification
   if TestDatum[E] < Boundary[E]     % at different epochs
       PosCount= PosCount + 1
   else
       NegCount= NegCount + 1
   end %If%
end %For%

If PosCount >= AcceptableProportion * NegCount   % Classify
   return(positive)
else
   return(negative)
end %If%
***************************************************************
```

Binary Search

In the binary case, the threshold determination component processes both positive and negative instances of the concept and uses a binary search strategy in order

[3]The values of all the parameters: *RelaxationRatio*, *AcceptableProportion* and *MinimumEpoch* will be specified in Section 4.3.1

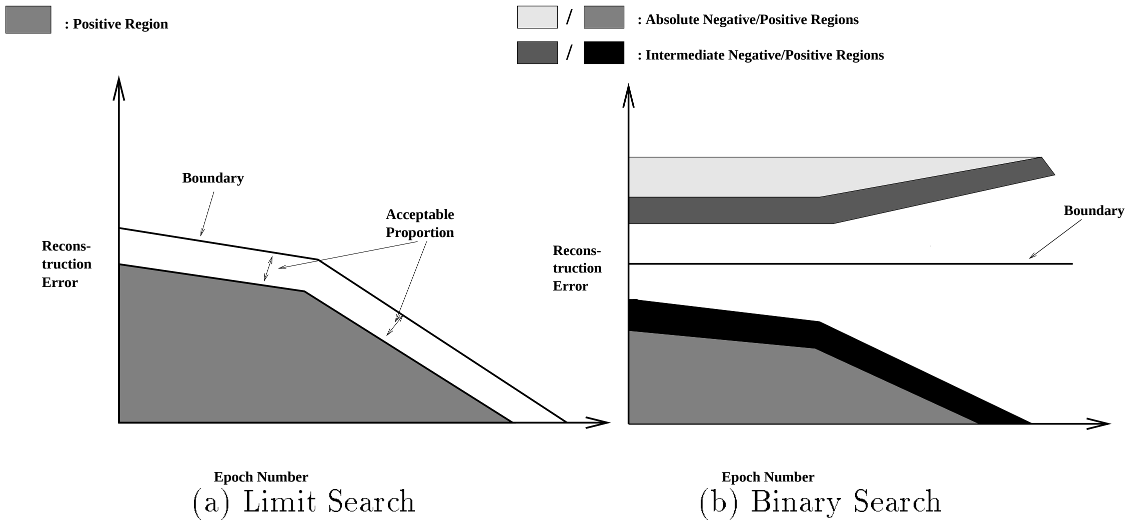


Figure 3.2: The two cases of threshold determination. In the limit search case, only positive or negative data are required while in the binary search case both positive and negative data are needed. In figure (a), the white border separating the gray zone from the boundary constitutes the acceptable proportion.

to establish a boundary. The procedure in this case tries to find the epoch at which best separation between the reconstruction errors of positive and negative instances of the concept is achieved, among all the epochs considered and at the same time, considers how stable this separation is.

In order to find the best separation, the procedure begins by constructing the boundaries of the *absolute* and *intermediate* positive and negative regions of the epoch versus reconstruction error space, at every epoch considered. Instances that belong to a given class with great certainty have reconstruction errors that fall in the absolute region of this class while instances that belong to this class with less certainty have reconstruction errors that fall in its intermediate region. Absolute and intermediate regions differ from *actual* regions which span the entire negative and positive instance sets respectively. Absolute and intermediate regions are illustrated in Figure 3.2(b). To construct the negative intermediate region in particular, this procedure uses a binary search technique that begins by stating the lower and upper boundaries of the actual negative region, and then proceeds by repeatedly shrinking this region by manipulating its boundaries until it finds the most accurate intermediate negative region. Accuracy depends on the fraction of the entire data set located in this region of the space. The parameter defining this fraction is called $AcceptableLowerRatio$ (see pseudo-code below) and will be specified in Section 4.3.1 The region located above the upper boundary of the final intermediate negative region defines the absolute negative region. The following is the pseudo code for the procedure that determines the boundaries of the negative intermediate region, $low[E]$ and $high[E]$, at a given epoch, E. This procedure takes as input

- $NegativeSet$, a 2-Dimensional array representing the output of the neural network at each epoch recorded of every point in the negative threshold-determination data set.

- E, a particular epoch belonging to the set of recorded epochs.

It returns low[E] and high[E], the lower and higher bound of the intermediate negative region, respectively, at point E.

```
****************************************************************
DetermineIntermediateBoundaries(NegativeSet, E)

  SortedNegSet= sort(NegativeSet)  % Determine lowest, highest
                                   % and mid positions on the
  low= SortedNegSet[1, E]          % entire negative set.
  high= SortedNegSet[MAX, E]
  mid= (low+high)/2

  ok= false
  while not ok do                  % Repeatedly move
    UpperCount=count(mid, high)    % the boundaries of the
    LowerCount=count(low, mid)     % intermediate region
                                   % until the number of
                                   % examples in the lower
                                   % part is smaller than
                                   % (acceptable-lower-ratio*
                                   % upper-count) within the
                                   % negative intermediate region

    if LowerCount >= AcceptableLowerRatio % if the lower part
                    * UpperCount          % contains more than an
       high=mid                           % acceptable lower ratio
       mid=(low+high)/2                   % of the number of data
                                          % in the upper part, then
                                          % move the upper bound to
                                          % the mid point and reset
                                          % the mid point.

    else while LowerCount > 0       % else, while the lower part
            low= median(low, high)  % is not empty, keep on raising
         end %While%                % the lower boundary.
         ok= true
    end %If%
  end %While%
  return(low[E]=low; high[E]=high)
****************************************************************
```

The intermediate positive region is constructed in a similar fashion and the boundary for classifying positive and negative examples is established as the midpoint between the lower boundary of the negative intermediate region and the upper boundary of the positive intermediate region. Note that the decision of placing the boundary at the midpoint between the positive and negative examples was completely arbitrary as it assumes equal prevalences of the positive and the negative data. More flexibility in this placement should be introduced in order to account for the cases in which these prevalences are not equal.

In order to find the epoch with the most stable separation, the program calculates how stable the separation is at each recorded epoch. The stability of a given epoch is defined as the slope of the line that goes through the separation of this epoch and the next epoch recorded. The separation that is selected for classification is the best separation whose stability is higher than $1/AcceptableSlopeRatio$ times the greatest stability encountered.[4] In more detail, picking the best epoch and classifying is done as follows in function $BinarySearch()$. The inputs to the function—$TestDatum$, $PositiveSet$ and $NegativeSet$—were defined previously and its outputs are the "positive" or "negative" label assigned to the data point P being classified.

```
***************************************************************
BinarySearch(TestDatum, PositiveSet, NegativeSet)

  For each recorded epoch, E   % Calculates the separation
                               % between positive and negative
                               % intermediate regions
    [LowNeg[E], HighNeg[E]]=
        DetermineIntermediateBoundaries(NegativeSet, E)
    [LowPos[E], HighPos1[E]]=
        DetermineIntermediateBoundaries(PositiveSet, E)
    Separation[E]= LowNeg[E] - LowPos[E]
  end %For%

  For each recorded epoch, E, (except the last one)
    Slope[E]=(Separation[E+1]-Separation[E])/2
  end %For%                 % Calculates the slope of two
                            % consecutive separations

  BestSep= -Inf                  % Calculates the epoch at
  BestSlope= +Inf                % which the separation is
  For each recorded epoch, E     % largest and somewhat stable
    if Separation[E] > BestSep
       if Slope[E] < AcceptableSlopeRatio
```

[4]Once again, the value of the parameter *AcceptableSlopeRatio* and all other parameters will be listed in Section 4.3.1.

```
                  * BestSlope
          BestSep=Separation[E]
          BestSlope=Slope[E]
          BestEpoch=E
       end %If%
     end %If%
  end %For%

  Boundary= (LowNeg[BestEpoch]+HighPos[BestEpoch])/2 % Classify
  If TestDatum(BestEpoch) <= Boundary
       return(positive)
  else return(negative)l
  end %If%
****************************************************************
```

As will be described in Section 4.3.1, the same threshold-determination component is also used for the other connectionist procedures of this dissertation.

3.3.2 Parametric Recognition

The two threshold-determination techniques described in Section 3.3.1 are heuristic in nature. Some of the autoassociation experiments conducted on the artificial domains of this dissertation use a threshold determination technique different from the ones just described. In particular it uses positive data only and relies exclusively on a simple technique from statistical pattern recognition. This technique, which was designed for its simplicity and for the fact that it does not require counter-examples of the concept, assumes that the distribution of the reconstruction errors is Gaussian and that the class noise ratio of the conceptual class is available or can be guessed. Note, however, that neither of these assumptions are justified and that, as a matter of fact, it is frequent for ratios, unlike sums, not to be normally distributed. Nevertheless, the experiments of this dissertation that use parametric recognition are based on worst-case analyses which question how well an autoassociation-based classification system can perform in the complete absence of counter-examples even if the threshold-determination component is severely biased. The quality of the results obtained using parametric recognition (See figure 4.6 in Chapter 4) suggest that autoassociation-based classification in the complete absence of counter-examples is an accurate classification method worth considering further since its results are acceptable even though the overall system uses a poorly adapted threshold-determination component. Research on how to design threshold-determination components better adapted to the autoassociator have been left for future work.

Parametric Recognition begins by gathering the results of applying the trained device to each positive data point of the threshold-determination data set. Each value thus computed is divided by the average of the values obtained by applying that same device to the entire training set, yielding an *error ratio* for each point in the threshold-determination data set. In other words, for each point (x, y) of the positive threshold-determination data set, the error ratio $z(x, y)$ is calculated as[5]:

$$z(x,y) = \frac{((A(x)-x)^2+(A(y)-y)^2)}{\Sigma_{i=1}^{N}((A(x_i)-x_i)^2+(A(y_i)-y_i)^2)/N}$$

In the above formula, x and y represent the two dimensions of each data point, $A()$ is the function realized by the trained autoassociators, and N is the total number of training points. Once computed, these error ratios are fitted to a Gaussian distribution and a boundary is set so as to allow for a fixed miss/fit rate of 3%. This boundary is subsequently used for classification of the test data. Parametric Recognition will be used for a few experiments in Chapters 4 and 5. In these experiments, Parametric Recognition stands as a very simple threshold-recognition method performant enough to underline, in a preliminary fashion, two interesting trends (1) the autoassociator's efficiency with respect to MLP; and 2) its accuracy with respect to PCA and linear autoassociation) that are then discussed in greater depth.

3.3.3 ROC Analysis

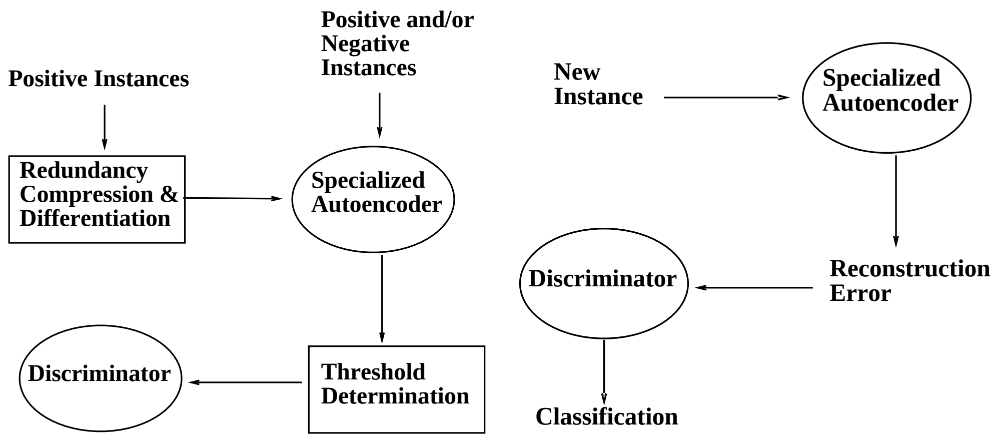
The last threshold-determination technique used in this dissertation is called Receiver-Operating Characteristic Analysis (ROC). ROC Analysis is a testing procedure used in the field of signal detection [Van Trees1968], which consists of considering the two types of errors—Type I and Type II errors—that can be made by a concept-learner and plotting them against one another. In effect, this corresponds to changing the threshold of the classifiers continuously. The curves obtained by such a procedure are commonly used by designers of classifiers who can use them to select a proper operating point and its corresponding threshold, depending on his/her needs in terms of a Type I/Type II error balance. In studies where the purpose of the experiments is not the definition of a single optimal operating point, but rather, the comparison of two systems according to all possible thresholds, drawing the ROC curves and analyzing them globally is a useful technique. It is this technique that was used in Chapter 7.

In more detail, the ROC curves I display in this chapter are generated for each classifier by plotting the percent of positive instances of the concept correctly characterized versus the actual number of false positives. If two classifier are represented in a plot, then the curve located above the others represents the most accurate classifier.

This concludes the description of the threshold-determination components designed in this dissertation. The next section concludes the chapter with a description of the overall functioning of the system.

---

[5]Please, note that the artificial domains used in this dissertation are two-dimensional, except for one domain in Chapter 5, which is three-dimensional.



(a) Training Phase (b) Testing Phase

Figure 3.3: Overall Functioning of the System.

3.4 Overall Functioning of the System

This section describes the overall functioning of the system which integrates the concept-learning and threshold-determination components together in order to present the full classifier.

The training of the autoassociation-based classifier is carried out in two phases. In the first phase, the inductive component is trained with positive instances of the concept. This phase results in the computation of a *specialized autoassociator* which is expected to differentiate between a positive and a negative instance by showing a small reconstruction error in the positive case and a large one otherwise. The second phase of training consists of training the threshold determination component. As seen above, in this phase, the specialized autoassociator is used with positive and/or negative instances. For each instance, the reconstruction error is recorded and fed into the threshold determination component which analyzes the reconstruction error of all the instances and issues a *discriminator*. The discriminator can be interpreted as a boundary between positive and negative instances. Figure 3.3(a) illustrates the functioning of the overall concept learner.

Once the two components have been trained, the classifier can be used as follows. First, an unlabeled instance can be input to the specialized autoassociator which will issue a reconstruction error. The reconstruction error can then be input to the discriminator which will issue a classification. Figure 3.3(b) illustrates the system's functioning during presentation of novel data.

Having now described the classifier under study in this dissertation, I can describe the various experiments that were conducted in order to explain its inner workings and analyze the conditions under which it is most effective.

Chapter 4
Evaluation on Practical Domains

Chapter 3 has described the autoassociation-based classifier used by [Hanson and Kegl1987] to acquire syntactic rules of English. This classifier, however, has not been tested on other domains and it is currently not known whether the autoassociation-based classification scheme has any practical engineering value or whether it is merely interesting for modeling certain cognitive processes. If our only interest is of a cognitive nature, then a study of the inner-workings would only be of academic interest for a computer scientist. If, however, it can be shown to be useful for practical tasks, then its study may be of practical interest to a computer scientist since it can provide some hints as to how to apply the system appropriately and how to optimize its performance.

The purpose of this chapter is to test the performance of the autoassociation-based classifier, both in terms of accuracy and time efficiency, in order to find out whether a deeper analysis of its operation is warranted. The evaluation was first performed on three real-world domains and the results obtained were subsequently verified on an abstraction of these three domains.

More specifically, the autoassociator's performance was established by comparing its accuracy and efficiency to MLP, Predictive Autoencoding, C4.5 and IB4 on three different practical domains of application: helicopter gearbox fault monitoring, DNA promoter recognition, and sonar target detection. Some aspects of these domains were then abstracted in a two-dimensional nonlinear and multi-modal domain. The accuracy and time efficiency of the autoassociator and MLP were compared on the abstracted domain in order to formalize the results obtained on the real-world domains. The autoassociator and MLP were the only two systems retained for the last experiment since MLP is the closest discrimination counterpart of the autoassociator, the recognition-based classifier of interest in my dissertation. The chapter concludes by posing three core questions aimed at explaining the preliminary results of this chapter and thus gathering a more complete and accurate assessment of the autoassociator's operation and performance. Each of these questions is subsequently answered in Chapters 5, 6, and 7.

The chapter is organized in six sections. Section 4.1 introduces the four systems contrasted to the autoassociator for the real-world domain data. Section 4.2 describes the three real-world domains considered. Section 4.3 describes the real-world experiments including their preliminary settings and the actual experiments. Section 4.4 lists the results obtained. Section 4.5, first, analyzes the real-world domains, and second, builds an artificial abstraction of these domains. It then tests the autoassociator and MLP network on these domains. Section 4.6, finally, lists the three questions whose answers represent the core of this dissertation.

4.1 Learning Systems Compared

As just mentioned, the accuracy of autoassociation-based classification was tested by comparing its performance to that of four different systems which include two connectionist learners: *MLP* and PREDICTIVE AUTOENCODING and two symbolic systems: *C4.5* and *IB4*. These four systems are conventional discrimination-based learners that use a roughly equal number of positive and negative data for training. The autoassociator is the only learner considered in this chapter that performs recognition-based learning and uses only positive instances of the concept in the concept-learning component. This section describes each learner in turn and points to some of the differences between the various schemes.

4.1.1 MLP

As mentioned earlier, MLP refers to the standard connectionist method which consists of applying the back-propagation procedure [Rumelhart *et al.*1986] to the classical discrimination-based feedforward network architecture. A discrimination-based feedforward network was illustrated in figure 2.1(a) of Chapter 2. In MLP, such a network is trained on positive and negative instances of the concept and by convention is made to output a “one” in the case of positive data and a “zero” in the case of negative data, using the backpropagation procedure described previously in Section 2.3.2. Learning in MLP can actually be seen as a two stage process. The first stage occurs while going from the input layer to the hidden layer in the backpropagation feedforward pass. During this stage, a representational shift occurs that tends to separate the positive from the negative data. Practically, this effect can be seen in the hidden layer where the positive data appear in some sense further away from the negative data than they were in the input layer ([Munro1991]). This type of behavior by the first stage of the learning process is actually caused by the computations that take place in the second stage of the process which occurs while going from the hidden layer to the output node in backpropagation's feedback pass: during this second stage, a discrimination function is computed that tries to discriminate between positive and negative data.

The threshold-determination component used by the MLP classifier for the real-world domain experiments are the same ones used by the autoassociator and described in Section 3.3.1: Limit and Binary Search. However, instead of the reconstruction error, the value on which threshold-determination is based for MLP is the value of the output unit. This means that the result of the threshold-determination components described in Section 3.3.1 have to be reversed since in the autoassociator, a positive instance of the concept is expected to obtain a smaller reconstruction error than a negative instance whereas for MLP's output unit, the opposite holds.

4.1.2 Predictive Autoencoding

The architecture used in PREDICTIVE AUTOENCODING [Gluck and Myers1993] was described in section 3.1.2 (Figure 3.1). Again, the network is trained using the back-propagation procedure [Rumelhart *et al.*1986]. PREDICTIVE AUTOENCODING can be seen as a mix between the autoassociator and MLP. In PREDICTIVE AUTOENCODING, the system is trained on both positive and negative instances of the concept and is expected, first, as was previously described for the autoassociator, to reproduce the input data at the output layer, and second, like in MLP, to discriminate between positive and negative data using its extra output node. Of all the nodes in the output layer, the only one that is relevant at classification time is the one that is used for discrimination: the other ones are present only for the purpose of constraining the representation in the hidden layer.

As in MLP, the learning process can be thought of as a two stage process during the first stage of which a representational shift occurs and during the second stage of which, a discrimination function is computed. During the second stage of learning, PREDICTIVE AUTOENCODING computes a discrimination function exactly in the same way as MLP (since only the discrimination node of the output layer is relevant for classification). The representational shift used by PREDICTIVE AUTOENCODING in the first stage of learning, however, is different from that of MLP. This time, not only is the hidden layer used to hold information that discriminates between positive and negative data, but it is also used to reconstitute the input layer at the output level. In order to reconstitute the input layer at the output level, the hidden layer has to compress redundancies in the data. This means that the data in the hidden layer not only need to save the information that discriminates between the data of both classes, but also that they need to save the information that underline the commonalities in the data. In effect, PREDICTIVE AUTOENCODING simultaneously separates positive from negative examples of the concept and groups the examples of each class together ([Gluck and Myers1993]). In other words, PREDICTIVE AUTOENCODING can be seen as performing a discrimination and a recognition task simultaneously.

In PREDICTIVE AUTOENCODING, the threshold determination components are used exactly in the same way as in MLP.

4.1.3 C4.5

C4.5 is a decision tree learning algorithm [Quinlan1993]. Figure 4.1 is an example of a decision tree that can be learned by C4.5. A decision tree is a formalism that allows to classify examples of a domain according to the value taken by some or all of its attributes. A decision tree learning system proceeds by recursively

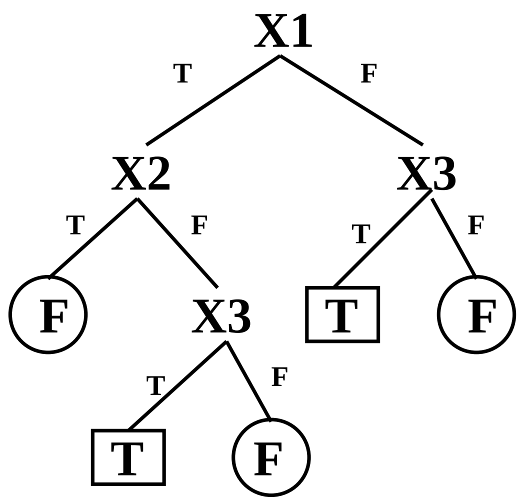


Figure 4.1: A Decision Tree that could have been Learned by C4.5 in the boolean domain. Positive leaves are indicated by a square and negative ones, by a circle. Each branch is labeled with the value "true" or "false". A novel example is classified by following the branches that correspond to the values of its various attributes, until a leaf is reached. The class of the example is the value at that leaf.

selecting the attribute with greatest discrimination value and dividing the training set according to the values taken by this attribute until all subsets of the training set are homogeneous (i.e., of the same class). To avoid overfitting the data, C4.5 prunes the tree after it is built by merging the children of some nodes, thereby allowing non-completely homogeneous subsets to remain unsplit. Note, also, that rather than merging the children of some nodes, the construction of the tree can be halted prior to the obtention of homogeneous sets. The amount of homogeneity permitted can be varied in a way similar to the way in which thresholds are set in the connectionist systems once concept-learning has taken place. Once the tree is constructed, new examples are classified by going down the branches of the tree according to the attribute values of the example, until a leaf is reached. Leaves represent classes of examples.

Like the two previous systems, C4.5 is a discrimination-based classifier, however, unlike the other two systems, C4.5 does not perform any representational shift prior to discriminating between examples: discrimination is performed on the original attribute space that was chosen by the user for describing the data.

No separate threshold-determination component is necessary since boundaries between positive and negative data are implicitly erected during the concept-learning process.

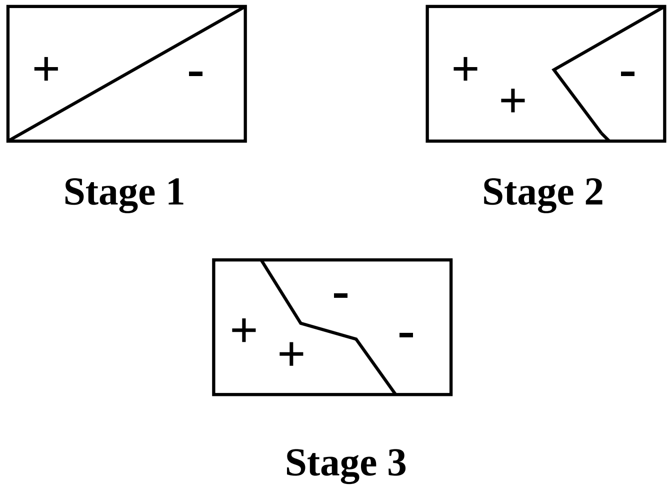


Figure 4.2: Learning with IB4. At every stage of learning, IB4 creates boundaries that divide the space into several regions that are such that any point within a positive region is closer to at least one positive example than to any negative example, and similarly for negative regions. The three figures above show the change in the boundaries as additional examples are presented to the system.

4.1.4 IB4

IB4 is an Instance Based Learning system[Aha *et al.*1991]. Instance based learning systems are similar to nearest neighbor systems. During training, IB4 memorizes all the significant training examples it encounters and at classification time, it returns the class of the example that is closest to the one in the process of being evaluated. Overfitting the data is avoided by keeping a classification record of each training example. Training examples are dropped if their classification record is too low. In addition, IB4 drops irrelevant attributes prior to classification. Figure 4.2 illustrates the learning strategy used by IB4.

IB4 is similar to the other three systems just described in that it is a discrimination-based system. The way it discriminates between positive and negative instances of the concept is by constructing discrimination lines into the attribute space which consist of mid-points between positive and negative data. As well, like in MLP and Predictive Autoencoding, IB4 uses a representational shift. IB4's representational shift is close to that of the autoassociator and aspects of that of Predictive Autoencoding since IB4 attempts to group data of the same class together rather than separate the data of one class from that of the other. It does so by dropping the features that it finds insignificant for the classification task at hand. Like in C4.5, overfitting avoidance is done by discarding training data since IB4 discards examples with too low a classification record and like in C4.5, threshold-determination is implicit in the IB4 process.

4.2 Domains Tested

The autoassociator's robustness, i.e., its ability to perform well across domains in terms of classification was evaluated by the comparisons between the systems described in the previous section for different domains. This section describes the three domains that were chosen for this study. These domains were selected because they all present challenging problems to the machine-learning community.

4.2.1 CH46 Helicopter Gearbox

The CH46 Helicopter Gearbox data was obtained from NRaD [Kolesar and NRaD1994]. The CH46 Helicopter problem is a monitoring problem that consists of discriminating between faulty and non-faulty CH46 helicopter gearboxes, according to the whining sound they emit during their operation. The sudden, unexpected failure of CH46 helicopter gearboxes is currently very costly both in terms of lives and equipment. The development of a monitoring system that can identify imminent failures before takeoff or when in flight is of paramount importance. The data for this problem was obtained by pre-processing the vibration time signal of the gearboxes of various faulty and non-faulty helicopters. The complete data set is composed of 18 non-faulty instances and 46 faulty ones which come in the form of 256 dimensional vectors of real numbers. In this particular problem, I chose the non-faulty examples to represent the conceptual class. Given the high dimensionality of each example, 18 training data constitutes a small sample. However, these are the only data that were provided by the U.S. Navy (in the first stage of our project) which, nevertheless, expected us to design a classifier and classify a blind test data set.

4.2.2 DNA Promoter

The DNA Promoter problem takes as input segments of DNA, a subset of which represent promoters. A promoter is a subsequence that signals the beginning of a gene to the chemical processes acting on the DNA. The goal of the problem is to train a classifier to be able to recognize promoters, which are taken to be the concept class. The training set is composed of 100 examples (47 promoters and 53 non-promoters), each of which is composed of a set of 51 nucleotides, where each nucleotide can take one of four values {a, c, g, or t}. The promoter data was obtained from the U.C. Irvine Repository of Machine Learning and was modified in response to Norton's critique of the biological flaws underlying the original formulation of the data [Norton1994]. In addition, as is usual for this problem, each example was converted into a 204-bit long vector where each nucleotide was represented with 4 bits when used by the connectionist systems.

4.2.3 Sonar Target Recognition

The Sonar Target Recognition problem takes as input the signals returned by a sonar system in the cases where mines and rocks were used as targets. The sonar data was obtained from the U.C. Irvine Repository of Machine Learning though only a subset of 100 instances (47 mines and 53 rocks) from this data was used in this particular case study. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles. Each instance of this data is represented as a 60-bit long vector. In this particular case study, we chose the signals returned by the mine targets to constitute the concept class.

4.3 Experimental Methodology

As mentioned in Section 2.5.2, in order to compare two neural network concept-learners reliably, their parameters have to be set optimally and their threshold-determination component has to be applied uniformly. Furthermore, when comparing different systems, and not necessarily, neural networks, the error of all the classifiers has to be estimated reliably.

4.3.1 Preliminary Settings

The preliminary settings for the experiments of this chapter were carried out as follows. First, an optimal capacity and stopping time were determined for each connectionist system for each domain problem considered. For the helicopter gearbox application, autoassociation-based classification and predictive autoencoding were shown to perform best with 32 hidden units, and MLP was shown to perform best with 50 hidden units. All three systems took about 200 epochs to converge. The results were recorded every 10 epochs for the first 190 epochs and every epoch subsequently.[1] For the promoter problem, all systems were shown to perform optimally with 153 hidden units while for the sonar target recognition problem, their optimal performance was achieved with 20 hidden units. In both applications 100 epochs recorded every 10 epochs were shown to be sufficient. The learning rate and momentum for the autoassociator, MLP, and PREDICTIVE AUTOENCODING were set to the standard values of 0.05 and 0.9, respectively, and held constant for all three case studies.

Once the optimal network parameters were established, the parameters of the threshold-determination component were also searched for on the same random set. For the helicopter gearbox problem, the Limit Search threshold-determination technique of Section 3.3.1 was selected and applied to negative examples of the

[1]Please note that finding a specific stopping criterion was not necessary in this work since the threshold-determination components used (Limit and Binary Search) are able to find one themselves.

concept, since the scarcity of the conceptual examples did not permit a two-class approach. In this procedure, the parameters were set as follows:

- $RelaxationRatio = .25(= 25\%)$
- $MinimumEpoch = 150$
- $AcceptableProportion = 1/2$

For both the promoter and the sonar target recognition problems, the Binary Search threshold-determination method of Section 3.3.1 was selected. In this procedure, the parameters were set as follows:

- $AcceptableLowerRatio = 2/5$
- $AcceptableSlopeRatio = 2$

In all cases, the same threshold-determination implementation with the same parameter settings was used across connectionist systems.

4.3.2 Learning Sample Selection

As mentioned previously, the systems were evaluated using 5-fold crossvalidation. At every fold of every experiment, the training set used by C4.5 and IB4 was divided into a training set for concept-learning and one for threshold-determination for the connectionist systems. Ten counter examples were selected at random for threshold-determination in the cases of the helicopter gearbox domain, and 5 examples of the concept and 5 counter examples were selected at random for threshold-determination in the case of the promoter and sonar domains. Since the autoassociator learns a concept from positive instances of the concept only, the counter-examples were eliminated from the concept-learning training set of this system while they were kept for MLP and Predictive Autoencoding. The testing sets of all the systems always corresponded and included both positive and negative instances of the concept, no matter whether negative data were used or not during the inductive learning process. Note that at every fold of every experiment, autoassociation-based classification uses significantly fewer negative data for overall training than the other four systems, although it is tested on the same number of negative data as the other systems: for the CH46 helicopter gearbox problem, it uses 10 negative data while MLP, Predictive Autoencoding, C4.5, and IB4 use between 35 and 39 such data (depending on the fold). For the other two case studies, autoassociation-based classification uses 5 negative instances while MLP, Predictive Autoencoding, C4.5, and IB4 use between 40 and 44 such instances.[2]

[2]It is possible that the discrimination-based classifiers could perform adequately with fewer counter-examples, but it is extremely unlikely that they could do so with as few as 10 or 5 of them, as suggested by the CCDD experiment of Chapter 7. On the other hand, only 5 or 10

4.4 Results

The error rates obtained by autoassociation-based classification, C4.5, MLP, Predictive Autoencoding, and IB4 in the three case studies considered are listed in Table 4.1. The best results in each case study are printed in boldface characters. Numbers after each "±" are standard deviations for each of the five-fold averages.

Case Study	AUTOASS. *Error (%)*	*C4.5* *Error (%)*	MLP *Error (%)*	*IB4* *Error(%)*	PRED. AUT. *Error(%)*
Helo	**3.125**$\pm$0.89	15.625 ± 1.87	10.9 ± 1.67	16.7 ± 2.95	7.8 ± 1.0
Promo	**20**$\pm$0.71	35 ± 1.41	**20**$\pm$1.41	39 ± 2.59	**20**$\pm$1.22
Sonar	**20**$\pm$2.65	29 ± 1.79	32 ± 3.21	24 ± 2.39	30 ± 2.83

Table 4.1: Summary of the Results for the three case studies.

Table 4.1 shows that in both the CH46 Helicopter and the Sonar Target recognition case studies, Autoassociation-Based Classification performed much better than either MLP, PREDICTIVE AUTOENCODING, C4.5, or IB4. In the Promoter case study, Autoassociation-Based classification, MLP and PREDICTIVE AUTOENCODING—all three connectionist systems— performed equally well and better than C4.5 and IB4.[3] Altogether, these preliminary experiments demonstrate that in addition to requiring a much smaller number of counter-examples of the concept for training than other systems[4], Autoassociation-Based Classification is capable of classifying novel instances more accurately than all the systems in all case studies except for the Promoter data, where its performance is matched by MLP's and PREDICTIVE AUTOENCODING's. In addition, AUTOASSOCIATION-BASED CLASSIFICATION is relatively robust, at least in the context of the three real-world domains considered and of the four classifiers against which it was tested.

---

counter-examples were sufficient to set the threshold since, as mentioned in Section 2.5.2, in neural networks, the concept-learning phase takes care of most of the learning and threshold determination is very simple.

[3]In the helicopter gearbox monitoring domain, the comparisons are all statistically significant with $p < .05$, except for the comparison with PREDICTIVE AUTOENCODING where $p = .1$. In the promoter recognition domain, the comparisons with C4.5 and IB4 are statistically significant with $p < .02$ while the significance of the comparison with MLP and PREDICTIVE AUTOENCODING were not established since the three systems performed equally well. In the sonar target recognition domain, the comparisons of MLP and PREDICTIVE AUTOENCODING with Autoassociation-Based Classification both achieved a significance level of $p < .05$, but the comparisons with C4.5 and IB4 could not be deemed significant. The significance levels were established using a T-test for coupled experiments that computes the probability that the better data is not in fact better.

[4]More counter-examples could be used by the autoassociation-based classification procedure, yielding as accurate or more accurate a threshold. However, were fewer counter-examples used during the MLP process, as shown in the CCDD experiment of Chapter 7 (figure 7.13), classification would not succeed.

In terms of time efficiency, as described in Section 4.3.1, all three connectionist systems[5] take about the same number of epochs: fewer than 200 in the helicopter gearbox fault monitoring domain and fewer than 100 in the DNA promoter recognition and sonar target recognition domains. This shows that, as far as time efficiency is concerned, autoassociation-based classification does not present a liability with respect to MLP and Predictive Autoencoding.

Because they are each other's true counterpart as far as recognition- versus discrimination-based learning is concerned, it is more meaningful to compare the autoassociator to MLP than to the other discrimination-based classifiers. Therefore, in the remainder of this dissertation, comparisons between discrimination- and recognition- based systems will be restricted to contrasting autoassociation-based classification to MLP.

4.5 Formalization

The results presented in the last section are interesting in three respects. First, they suggest that learning in the absence of counter-examples of the concept is possible even in the absence of statistical or contextual knowledge about the conceptual class. Second, they show that autoassociation-based classification is as time efficient as MLP, at least on the three domains considered. Finally, they suggest that, rather than being a liability, the absence of counter-examples can present an accuracy advantage in the learning process.

These observations, however, are very general and cannot be stated reliably without further investigation. The particular question that needs to be answered to refine these observations is whether the three domains tested in Section 4.4 present specific characteristics that are particularly favorable to autoassociation-based classification. This question is answered in this section which analyzes the CH-46 helicopter failure detection data, the DNA promoter recognition data, and the sonar detection data, isolates and abstracts their common characteristics and tests to what extent these characteristics are responsible for the results of Section 4.4.

In the remainder of this section, I first describe the analysis of the three domains; I then explain how the common characteristics of the domains were abstracted away; and I then show how these characteristics were tested to determine to which extent they are responsible for the results of Section 4.4. Finally, I conclude the chapter by stating the three core questions tackled by the balance of this dissertation.

[5]The question of learning efficiency is not relevant in C4.5 and IB4 since learning is practically instantaneous.

4.5.1 Real-World Domain Analyses

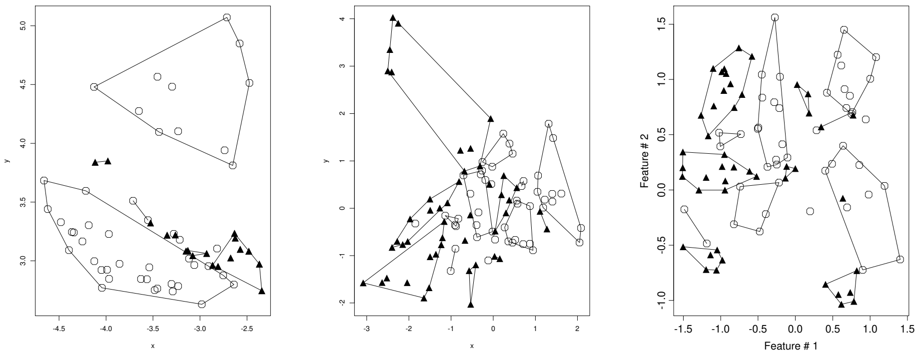
In this section, the three real-world domains presented in Section 4.2 were analyzed by plotting an approximation of the data based on Principal Component Analysis (PCA). More specifically, PCA was performed on each domain separately and the rotated data thus obtained were plotted along the first two principal components. The three plots were then analyzed using the graphical cluster analysis method described in Appendix A. The result of this process is shown in Figure 4.3.[6] In all the plots, black triangles represent positive instances of the concept and white circles represent negative ones. In the helicopter gearbox domain, positive instances correspond to non-faulty gearboxes and negative instances, to faulty ones. In the DNA promoter domain, positive instances correspond to promoters and negative ones, to non-promoters. And in the sonar detection domain, positive instances correspond to mines and negative ones, to rocks. The graphical clustering analysis of Appendix A yielded clusters of positive and negative instances of the concept distributed throughout the domain. These clusters are indicated by the convex hulls depicted in the plots of Figure 4.4.

Two main observations can be made about these plots. First, none of the domains are *linearly separable*; i.e., in none of these plots is a single straight line sufficient to separate the black triangles from the white circles. Second, it is not rare for two data clusters of a same class to be separated from one another by a data cluster of the opposite class; i.e., classes are not represented by a single homogeneous cluster, but rather, by a series of disconnected and class-intertwined sub-clusters. In other words, the three real-world domains of Section 4.2 can be characterized by both their *non-linearity* and their *multi-modality*. In order to assess whether these two properties are indeed those responsible for the results obtained in Section 4.4, they were abstracted away in an artificial domain and tested by the autoassociator and MLP. This operation is discussed in the next section.

4.5.2 Abstracting the Common Characteristics Away

The characteristics noted in the previous section were abstracted away by building an artificial domain. The two main considerations used to build this domain are that all the real-world domains used in Section 4.2 are nonlinear and multi-modal. An additional consideration is the observation that the average number of positive data clusters (of size greater or equal to three) is about four: the helicopter

[6] The first two principal components used in all the plots account for 39.4 % of the entire variance of the training set in the helicopter gearbox problem, 4.22% of the entire variance of the training set in the DNA problem, and 23.4% of the entire variance of the training set in the Sonar problem. This suggests that the domains are even more complex than they appear to be in two-dimensions. (Note that, in the case of the DNA problem, as will be discussed in Chapter 7, the two-dimensional plot is not significant since it represents a particularly low amount of information).



(a) Helicopter (b) Promoter (c) Sonar

Figure 4.3: 2D Plot and Analysis of the Three Real-World Domains of Chapter 1. Black triangles represent positive data and white circles represent negative data. The convex hulls surrounding various sets of data points represent the boundaries of the positive and negative components computed by the graphical clustering method described in Appendix A.

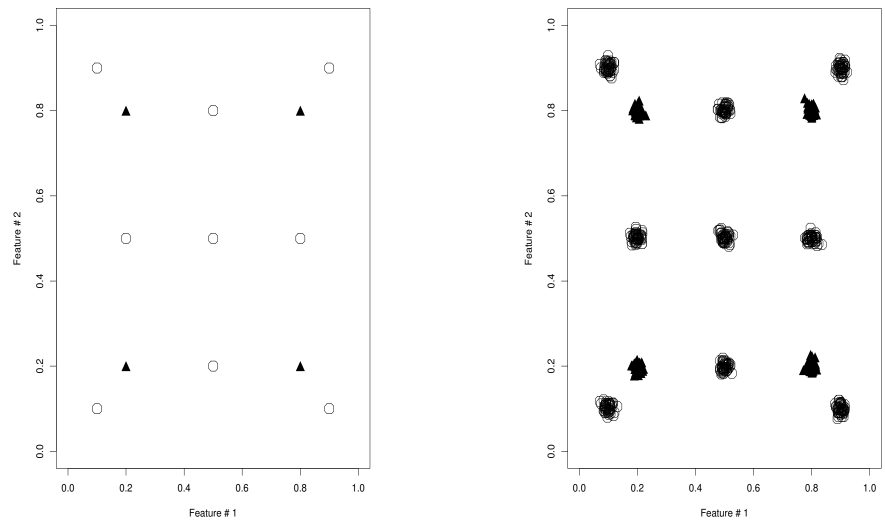
gearbox domain has two such positive clusters; the DNA promoter domain has five of them; and the sonar detection domain, six.

With this in mind, I generated a nonlinearly separable multi-modal domain composed of four disjoint concept-class clusters. The exact number of counter-concept components used in the domain and its exact configuration were derived using analytical considerations. In particular, I decided to configure the domain in a way that would be most "telling". More specifically, I opted for a domain with a maximum number of counter-concept areas of interest, each of a different nature. The way to maximize this number was to forbid any subset of size greater than two concept components to be aligned. This constraint guaranteed that the artificial domain would have three different counter-concept types of areas of interest: an interior area, a side area, and an exterior area, where the four positive components are viewed as the four corners of a quadrilateral. Following this layout for the data, I placed one or several negative component in each of the different areas of interest.

The artificial domain is displayed in Figure 4.4.

It is composed of four positive and nine negative components. More specifically, the figure shows that the means of the positive components are located at points: $(.2, .2)$, $(.2, .8)$, $(.8, .2)$, and $(.8, .8)$. The means of the negative components are located at points: $(.5, .5)$, for the negative "central" component; $(.1, .1)$, $(.1, .9)$, $(.9, .1)$, $(.9, .9)$, for the four negative "exterior" components; and $(.5, .2)$, $(.2, .5)$, $(.5, .8)$, and $(.8, .5)$, for the four negative "side" components. Each mean actually stands for a cluster of data points normally distributed around that mean and of variance $\sigma^2 = 0.01$.

Another more theoretical way to view the artificial domain of figure 4.4 is in terms of *X-OR* and *linear independence* problems. Indeed the domain can be



(a) Means Only (b) Complete Distribution

Figure 4.4: The nonlinearly separable, multi-modal Artificial Domain.

seen as a super-imposition of the four X-OR and the two linear independence problems depicted in Figure 4.5. Such problems are often used as a basis for assessing the difficulty of a domain, and it might be useful, for certain types of experiments not considered in this dissertation, to express our abstracted domain in terms of these common problems.

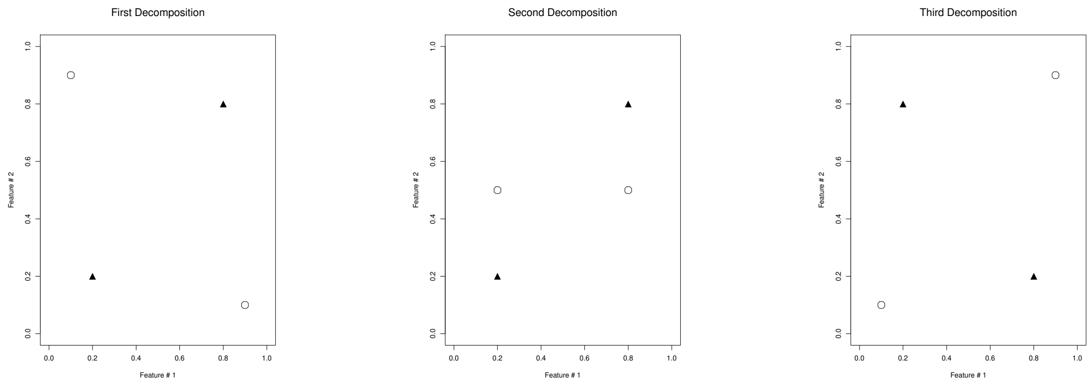
The following subsection describes the experiment conducted in order to establish whether nonlinear-separability and multi-modality are indeed the characteristics that could have been responsible for the success of the autoassociator in the real-world experiments of Section 4.4.

4.5.3 Testing the Abstracted Domain Characteristics

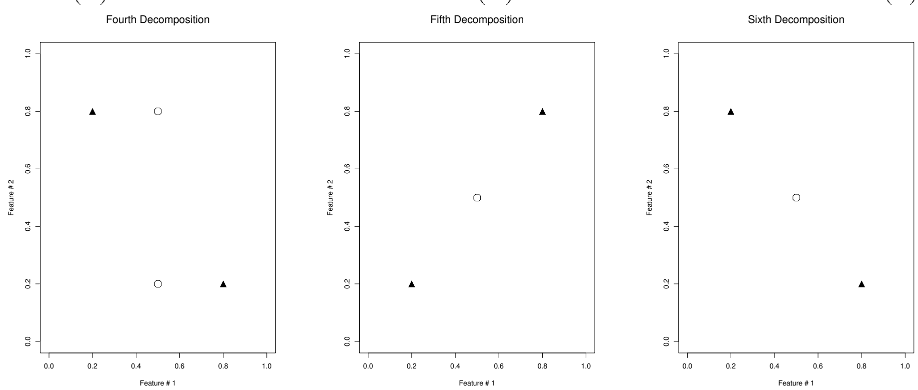
Testing the abstracted domain characteristics consisted of attempting to classify the data of the artificial domain just described using both the autoassociator and MLP. In these experiments, I used the Parametric Recognition threshold-determination technique described in section 3.3.2 for the autoassociator and set the boundary to

$$M = mean(NegativeSet) + (mean(PositiveSet) - mean(NegativeSet))/2$$

for MLP. The MLP threshold-determination technique is more robust than the autoassociator's parametric recognition threshold-determination technique and universally used. As discussed in footnote 10 of Section 2.5, this choice was made to find out whether acceptably accurate accuracy and time efficiency can be obtained in the absence of counter-examples both during the concept-learning and threshold-determination phases.



(a) X-OR 1 (b) X-OR 2 (c) X-OR 3



(d) X-OR 4 (e) Lin. Indep. 1 (f) Lin. Indep. 2

Figure 4.5: A Decomposition of the artificial domain of figure 2.3, based on the X-OR and Linear Independence (Lin. Indep.) Problems.

Parameter Settings

Like in all connectionist experiments, a capacity and maximum number of epochs had to be determined prior to running the experiments. It was shown that an optimal number of hidden units for the autoassociator and MLP for the artificial domain is 16. Convergence was attained when the autoassociator was run for 2000 epochs while MLP, which does not converge by 2000 epochs, was ran for 4000 epochs. Unlike the previous experiments, for which only a limited number of data are available, experiments in artificial domains allow an arbitrary large number of data to be generated. Thus, the holdout error estimation method rather than cross-validation can be used and in each of the training, threshold-setting and testing sets represented each cluster by 50 data points normally distributed around its mean. Furthermore, each experiment was repeated five times and the results reported are the averages of these five runs.

Results

The results obtained by both methods are illustrated in Figure 4.6 which displays the percent classification error rate obtained by both systems at various epochs.

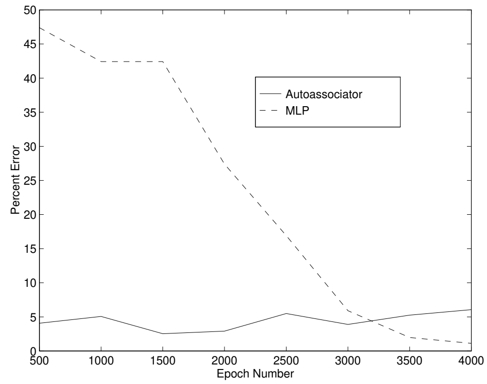


Figure 4.6: Classification error rates at different epochs for MLP and the autoassociator on the Artificial Domain. While the autoassociator has already learned the domain by epoch 500 (actually, it learns the domain as early as by epoch 100), MLP takes over 3000 epochs to learn it.

While confirming that the autoassociator is capable of classifying a nonlinear and multi-modal domain, the results do not confirm that such properties are the sole ones necessary for the autoassociator to outperform MLP as far as accuracy is concerned. They do, however, suggest that autoassociators are not simply as time efficient as MLP, but that on certain domains, they are actually much more efficient.

More specifically, Figure 4.6 suggest that nonlinear separability together with multi-modality are indeed favorable characteristics for autoassociation-based classification since the system displays a classification error of only around 6% at epoch 4000 on the artificially generated domain. However, since MLP is also capable of classifying this domain, and since the classification error rate it obtains is lower than that of the autoassociator, at about 1% at epoch 4000, then the observation that rather than being a liability the absence of counter-examples represents an accuracy advantage is not verified on this domain. This point will be addressed in Chapter 7 where the domain characteristics most favorable to autoassociation-based classification in relation to MLP will be extracted. Nevertheless, the useful additional observation that can be made based on the results of Figure 4.6 is that rather than simply not being a time efficiency liability, the autoassociation-based classifier can present a significant advantage. Indeed, while the autoassociator appears to learn the domain almost instantaneously—since by epoch 500, the error rate is around 4 percent and keeps on oscillating between the value of 5 subsequently (with a slight increase past epoch 2000 due to some overfitting)—, MLP on the other hand, learns the domain more slowly: at epoch 500, its error rate is at about 47%; at epoch 2000, it is still above 25%; and it is only after epoch 3000 that it drops below 5%.

The various observations made in this section and in this chapter represent the basis for the research conducted for this dissertation. The three core questions induced by my observations are stated in the next section. These questions will be answered in each of Chapters 5, 6, and 7.

4.6 The Three Core Questions Asked in this Dissertation

In summary, the observations made on the artificial domain and described in Section 4.5 as well as the results obtained on the three real-world domains and described in Section 4.4 raise three questions that need to be answered in order to explain the feasibility of autoassociation-based classification and discuss its liabilities and advantages. Answering these questions is useful both for indicating in which circumstances the autoassociator is a useful classification method and how it can be optimized. The three questions are stated below:

1. What features of the autoassociator make it capable of classifying domains which present nonlinearly separable and multi-modal characteristics?

2. What causes the autoassociator to learn the artificial domain at least 40 times more efficiently than MLP?

3. What domain characteristics cause autoassociation-based classification to outperform MLP or MLP to outperform autoassociation-based classification?

The first question will be partly answered in Chapter 5 which studies carefully the reconstruction error surface obtained by the autoassociator on the artificial domain of Section 4.5. Answers to the second question are given in Chapter 6 which studies the evolution of the hidden layer of both the autoassociator and MLP during learning on the artificial domain of section 4.5 as well as on a version of the simpler X-OR domain in an attempt to explain the difference in efficiency between the two systems. The last question is answered in Chapter 7 which generates and tests new artificial domains representing variations of the artificial domain of this section. The modifications are geared at strengthening or weakening the definition of the concept of the current artificial domain in an attempt to contrast the accuracy of the autoassociator and MLP under such conditions.

Chapter 5
Specialization in the Autoassociator

As discussed in Section 2.4 of Chapter 2, concept-learning, the second phase of the binary learning process can be decomposed into two subprocesses: *generalization* and *specialization*. Generalization seeks a concept description which characterizes all the examples of the concept while specialization seeks a concept description which excludes all its counter-examples. In this framework, counter-examples of the concept are important for training classifiers since their presence helps circumscribe the desired conceptual class by bounding the generalization thrust of the inductive process, thus enhancing its specialization strive.

Despite the importance of counter-examples, the results of Chapter 4 demonstrate that autoassociators with non-linearities in their hidden layer are capable of learning a concept *in the absence of* counter-examples.[1] Although it is usually expected that if counter-examples are absent, a concept-learner will use a statistical model of the conceptual data (e.g., Single hypothesis testing [Fukunaga1990]) or other domain-specific information (e.g., Fixed-Correlation or Deformable templates and Image Invariants [Sung and Poggio1994]), the autoassociator does not make use of either. The purpose of this chapter is consequently to identify which principles of the autoassociator make it capable of learning a concept without taking into account the information provided by counter-examples, a statistical model of the conceptual data or task-specific knowledge. In particular, since such information has an impact on specialization (rather than generalization), I seek to identify what *specialization mechanism* is used by the autoassociator to replace the missing information. In addition, since, as mentioned in Section 2.5.2.1 of Chapter 2, the bias of a neural network is controlled to a certain extent by the number of hidden units it uses, I survey the flexibility of the specialization mechanism by discussing the classification performance of the network as a function of its capacity.[2]

The chapter is divided into three sections. Section 5.1 reviews the literature on autoassociators in an attempt to identify the inner workings of the autoassociator from previous work. This literature indicates that autoassociators are

[1]As discussed in Chapter 2, although counter-examples are currently used for preliminary-setting and threshold-determination, concept-learning takes place in their absence.

[2]Please note that the "bias" of a classifier (see Section 2.5.2.1 of Chapter 2) and its "specialization" (see Section 2.4 of Chapter 2) capability are closely related concepts even though the first one is derived from statistical considerations whereas the second one follows from cognitive/machine learning concepts.

practically equivalent to Principal Component Analysis (PCA) and thus suggests that in order to identify the specialization mechanism of the autoassociator it is almost sufficient to explain that of PCA. This claim, however, is contradicted by my observations from Chapter 4 which can be interpretated to demonstrate that the autoassociator is not necessarily equivalent to PCA and thus, that its specialization mechanism must be identified independently of PCA. This is the goal of Section 5.2 which identifies the specialization mechanism of the autoassociator using the artificial domain of Section 4.5. This mechanism is also contrasted to that of PCA. Finally, Section 5.3 tests the flexibility of the autoassociator's specialization mechanism by studying the effects of varying the capacity of the autoassociator on its reconstruction performance of non-conceptual data in both the domain of Section 4.5 and on a 3-dimensional generalization of this domain.

5.1 Literature Review

As already discussed, the autoassociator used in the classification scheme analyzed in this dissertation is a single hidden layer network which contains nonlinearities in both its hidden and output layers. It is important to realize, however, that this particular architecture is not the only one possible: previous work has surveyed different variations of this network.[3] This section reviews the existing literature on both linear and nonlinear autoassociation. It begins by introducing a few of the autoassociative architectures discussed previously, summarizing previous work in the field, and then discussing some conventional wisdom from the connectionist community regarding nonlinear autoassociators. It concludes by showing why this conventional wisdom is erroneous.

5.1.1 Various Autoassociative Schemes

Prior literature has considered a certain number of autoassociative architectures, some of which are illustrated in Figure 5.1 in which circles crossed by a line symbolize linear layers whereas circles crossed by an elongated "S" represent nonlinear layers. In more detail, Figure 5.1(a) symbolizes a simple autoassociator with no hidden layer. Figure 5.1(b) symbolizes a one-hidden-layer autoassociator whose hidden and output layers are both linear. Figure 5.1(c) symbolizes a one-hidden-layered autoassociator whose hidden layer is nonlinear but whose output layer is linear. Finally, Figure 5.1(d) symbolizes a one-hidden-layered autoassociator whose hidden layer and output layer are both nonlinear. In all cases the input layer is linear and the function used in the nonlinear units is the usual logistic function, $\frac{1}{1+e^{-x}}$. The four connectionist systems are subsequently referred to as "L", "L-L", "NL-L" and "NL-NL", respectively, where "L" stands for "linear",

[3] In this work, I restrict my attention to single-hidden layer networks or networks with no hidden layer. However, architectures with additional layers have been considered as well (e.g., [Kramer1991], [Schwenk and Milgram1995]).

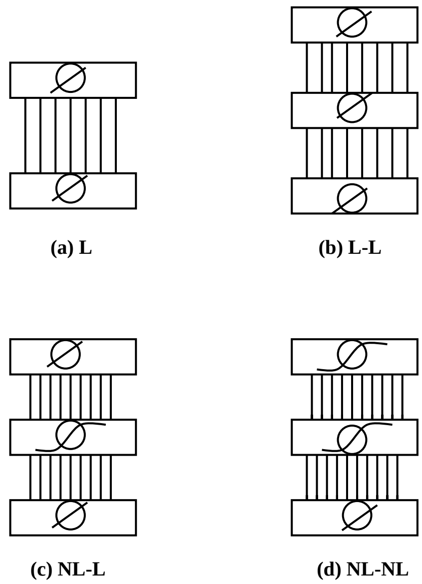


Figure 5.1: The four connectionist systems. Circles crossed by a line symbolize linear layers whereas circles crossed by an elongated "S" represent nonlinear ones.

"NL" stands for "nonlinear" and the symbol preceding the hyphen specifies the nature of the hidden layer while the symbol succeeding it specifies the nature of the output layer. If no hyphen is present, then the system does not have a hidden layer and the only symbol present specifies the nature of the output layer.

5.1.2 Summary of Previous Work

The literature on autoassociation has been concerned with two issues:

- What do autoassociators compute?
- How can they outdo existing techniques?

Since this literature can clarify the specialization mechanism of autoassociation, it is reviewed briefly. The first papers to address these issues are the experimental works by

Hanson and Kegl [Hanson and Kegl1987], Cottrell, Munro, and Zipser [Cottrell *et al.*1987] and Elman and Zipser [Elman and Zipser1988]. As discussed earlier, Hanson and Kegl showed that autoassociators could deal with a complex natural language processing problem. They used a one-hidden layer nonlinear autoassociator for concept-learning in the context of natural language syntax learning based on simple exposure to English sentences. Their system was shown to generalize to previously unseen sentences that are grammatically correct but to reject sentences which violate natural language constraints. However, the mechanism of the autoassociator on this task was not analyzed. At around the same time, Cottrell *et al.* used the NL-L autoassociator of figure 1(c) to compress a digitized image. Similarly, Elman and Zipser used the NL-NL autoassociator of figure 1(d) and the NL-L autoassociator of figure 1(c) to compress pre-processed and raw speech, respectively. Furthermore, [Cottrell *et al.*1987] compared the technique used by their autoassociator with the standard Principal Component Analysis (PCA) technique and showed that the two methods are very close since the hidden units of the NL-L autoassociator span the space of the first several principal components. They repeated the same experiment, using the L-L autoassociator of figure 1(b) and obtained similar results. Their work thus offers a first view on the two issues of interest in this area of research, namely (1) that one-hidden-layered autoassociators extract the first few principal components of a domain, or some close approximation to them and (2) that linear and nonlinear autoassociators are equivalently powerful on the task of image compression.

These two results were theoretically confirmed by Baldi and Hornik (1989) and Bourlard and Kamp (1988). In regards to the first issue, Baldi and Hornik conducted a theoretical study of the L-L autoassociator of figure 1(b) and showed that such networks actually project their input onto the subspace spanned by the first H principal components, where H corresponds to the number of hidden units.[4] The second issue was partly answered by Bourlard and Kamp who, short of discussing how autoassociators can outdo existing techniques in some general framework, demonstrated that the NL-L autoassociator of figure 1(c) does not outdo linear analytical methods, at least in the context of data compression, even if their hidden layer is nonlinear. In particular, they suggested that nonlinearities in the hidden layer are useless since the optimal parameter values of the NL-L autoassociator can be derived directly by purely linear techniques.

These results, nonetheless, did not completely discourage people from using autoassociators for the tasks of image compression, dimensionality reduction, or feature discovery. Such follow-up work was done by either modifying the presentation of the problem [Saund1989], or by using a different autoassociator architecture with three hidden units including two nonlinear ones ([Kramer1991], [Oja1991], [Usui *et al.*1991], [DeMers and Cottrell1993]). A third technique consisted of piecing together *local* linear PCA patches, rather than computing a *global* linear or nonlinear solution [Kambhatla and Leen1994]. In all three cases, the modifications were experimentally shown to improve on the linear analytical solutions obtained previously since they were able to derive nonlinear solutions to the compression problem.

---

[4] As an aside, it is interesting to note that autoassociators are not the only networks capable of extracting the principal components of a domain. Oja's rule (1982), a normalized version of the Hebbian rule, was also shown to compute the first principal component. This rule was extended to multiple principal components by Oja himself (1989) and Sanger (1989). However, since this line of research is only remotely connected to our topic, it will not be discussed further in this thesis.

5.1.3 A Conventional Wisdom

As a consequence of the findings discussed in the previous section, the connectionist community has generally accepted the fact that one-hidden layer autoassociators are often not capable of performing nonlinear operations, as suggested by the following quote from the popular textbook by [Hertz *et al.*1991]:

> "An interesting aspect of this problem is that nonlinearity in the hidden units is theoretically of no help [Bourlard and Kamp, 1988], and indeed Cottrell et al. found that nonlinearity conferred no advantage in their simulations."

and by the following quote from [Kramer1991]:

> "(Therefore) the performance of an autoassociative network with only one internal layer of sigmoidal nodes is often no better than linear PCA."

If correct, the almost-equivalence between autoassociation and PCA would entail that the specialization mechanism of the autoassociator is closely related to that of PCA. It would also suggest that single-hidden-layered autoassociators should not outperform PCA on classification tasks, and that the high quality of the results obtained by the autoassociator in the previous chapter are understandable only under the assumption that the domains on which the systems were tested are all linearly separable.

However, as demonstrated by the results of Section 4.5.3 of Chapter 4, the single hidden layer nonlinear autoassociator (at least NL-NL) is capable of classifying nonlinearly separable multi-modal domains that cannot be tackled by linear methods such as PCA. Furthermore, nonlinear autoassociators are also famous for their capacity to solve the encoder problem ([Rumelhart *et al.*1986]), a problem which cannot be solved by PCA because of the singularity of the principal components. These observations, thus, suggest that the conventional wisdom embodied in the above quotes is erroneous and that in fact, nonlinear autoassociation is not equivalent to PCA. The specialization mechanism of nonlinear autoassociators is therefore different from that of PCA and needs to be identified in order to explain how autoassociators perform classification. Prior to this, however, some flaws in the conventional wisdom are discussed in the next section.

5.1.4 Limitation of the Conventional Wisdom

When considering the conventional wisdom sparked by the results of [Bourlard and Kamp1988] and [Cottrell and Munro1988] and the evidence that nonlinear autoassociators can solve problems that cannot be solved by PCA ([Rumelhart *et al.*1986], [Japkowicz *et al.*1995], [Petsche *et al.*1996]), an important question comes to mind:

> Does the [Bourlard and Kamp1988] claim hold in all cases or does it depend on certain assumptions concerning the underlying domain or the autoassociator?

A careful look at the proof laid out in [Bourlard and Kamp1988] reveals that the theorem actually does depend on a particular condition. More specifically, let g be the nonlinear function present at the output of the hidden units of a nonlinear autoassociator (i.e., g is the function that makes the autoassociator nonlinear). The assumption made by [Bourlard and Kamp1988] in order to carry out their demonstration is that for small values of x, $g(x)$ can be approximated as closely as desired by the linear part of its Taylor power series expansion, such that:

$$g(x) \sim x_0 + \alpha_1 x$$

where x_0 and α_1 are two appropriately selected constants. However, this means that $x = h$, the vector of pre-synaptic hidden unit activations (i.e., the hidden unit activations prior to their transformation by g), must be very small.

Since,

$$h = W_1^T i + w_1$$

where i is the input vector to the network, W_1, the matrix of weights linking i to h and w_1, the vector of biases associated with h, h will be small if and only if W_1 and w_1 are small. So in other words, [Bourlard and Kamp1988]'s claim that nonlinear autoassociators are equivalent to PCA is only guaranteed to hold if a restriction on the amplitude of the weights and biases going from the input to the hidden layer and/or on the input applies.

This remark suggests that nonlinear autoassociators are not necessarily equivalent to PCA and that, therefore, their specialization mechanisms are not necessarily equivalent either. The next section identifies the specialization mechanism of the autoassociator and demonstrates experimentally that there are indeed differences between this mechanism and that of PCA and linear autoassociation. This result is shown to have practical consequences for the task of classification.

5.2 Specialization Mechanisms of Autoassociators

The purpose of this section is to analyze the differences between PCA and linear and nonlinear autoassociators and to derive the specialization mechanism of the nonlinear autoassociator. In particular, I contrast the performance of linear and nonlinear autoassociators relative to PCA on the nonlinear multi-modal problem of section 4.5 and, rather than considering the behavior of the hidden layer of the autoassociators—like previous work did—, concentrate on their output layer and compare the reconstruction errors they obtain to that of PCA. This method allows the viewing of the effect of nonlinear hidden units which, although possibly negligeable at the hidden layer, might be magnified at the output layer.

In more detail, this section presents the result of an experiment which demonstrates that, in the context of the reconstruction problem, *neither* linear *nor* nonlinear single-hidden layer autoassociators are equivalent to PCA. Although the equivalence between linear autoassociation and PCA can easily be recovered, the difference between the linear and the nonlinear schemes is more significant. In particular, I demonstrate that while linear autoassociators use a *uni-modal* interpolation bias (given a set of points S in a N-dimensional space, an m-variate continuous function f (with $m < N$) is said to interpolate the points in S if it is such that, for each point in S, $(x_1, x_2, ...x_n)$, we have $f(X) = Y$ where X is an m-variate vector composed of x_i values with $i \in I \in \{1, 2, ...N\}$ and Y is a $N-m$-variate vector composed of x_j values with $j \in \{1, 2, ...N\} - I$. As discussed in Section 2.4, there exists many functions f that satisfy this constraint and each of these functions is said to obey a different *bias* since it makes different guesses as to the Y values of the X's for any point not in S), the interpolation bias used by nonlinear autoassociators is *multi-modal.* It is this multi-modal interpolation bias that constitutes the specialization mechanism of the nonlinear autoassociator and makes classification of nonlinearly separable multi-modal domains possible.

This section is composed of four subsections. Section 5.2.1 describes the experimental methodology I used. Section 5.2.2 presents my experiments and the results I obtained . Section 5.2.3 is an explanation of these results, while section 5.2.4 discusses the implication of these results on the task of classification.

5.2.1 Experimental Methodology

I now describe the experiments which compare five different PCA or autoassociation-based recognition systems on the artificial domain of Section 4.5. The five devices considered and compared in this study are those illustrated in Figure 5.1, plus PCA[5], i.e., I am comparing the performance of: PCA, the L Autoassociator, the L-L Autoassociator, the L-NL Autoassociator, and the NL-NL Autoassociator. From an analytical point of view, the five systems differ in that three of them are purely linear (PCA, L, L-L) while the other two (NL-L, NL-NL) have nonlinear capabilities. As mentioned earlier, PCA stands for "Principal Component Analysis". The four autoassociators are connectionist methods which differ from one another with regard to the type of autoassociator they use.

The experiment I conducted consisted of computing and plotting the reconstruction error surface obtained by PCA and the four connectionist systems on an instance of the artificial problem of figure 4.4 in Chapter 4. In this instance, the training data set is composed of 200 data. More specifically 50 points were drawn from the distribution of each of the four positive components. The connectionist systems were trained using the backpropagation procedure with the standard learning rate and momentum of 0.05 and 0.9; The preliminary parameter-setting

[5]The technique used by PCA is the same as the one used in "view-based eigen-spaces" of [Turk and Pentland1991] and [Pentland *et al.*1994]

experiments of Chapter 4 determined that the three single hidden-layer autoassociators can be trained with 16 hidden units and for 2000 epochs in order to reach an optimal classification performance. For the same reason, PCA was used with two principal components.[6]

5.2.2 Results

The results I obtained on this experiment are presented in Figure 5.2 which displays 3-D plots of the error ratio surfaces constructed by L, L-L, NL-L and NL-NL, respectively. The plot for PCA is not displayed since the reconstruction error is zero for all points, ignoring the roundoff error. In the graphs of figure 5.2, all the plots are drawn along the x-, y-, and z-axes where x corresponds to the x-axis of the input domain, y, to its y-axis and z is the error ratio at every point considered in the space such that:

$$z(x,y) = \frac{((A(x)-x)^2+(A(y)-y)^2)}{\Sigma_{i=1}^{N}((A(x_i)-x_i)^2+(A(y_i)-y_i)^2)/N}$$

In the above formula, x and y represent the two dimensions of each data point, $A()$ is the function realized by the trained autoassociators, and N is the total number of training points.[7] Please note that for presentation purposes, the figures were scaled. In reality, the plots for the L and L-L autoassociators fall in the range $[1.3\times10^{-4}, 2.81]$, the plot for the NL-L autoassociator falls in the range $[.047, 420]$, and the plot for the NL-NL autoassociator falls in the range $[.065, 1113]$.

The plots of figures 5.2 are particularly helpful for understanding the nature of the solutions computed by the five methods. In particular, they allow for interesting insights regarding the correspondence between linear autoassociation and PCA, the correspondence between linear and nonlinear autoassociation, and the specialization mechanisms of different autoassociators.

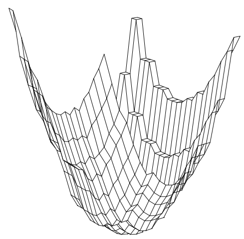
More specifically, the result for the comparison between linear autoassociation versus PCA is obtained by comparing Figures 5.2(a) and (b) to the flat reconstruction surface obtained by PCA.[8] This comparison shows that PCA computes different domain reconstructions from L and L-L. While this result

---

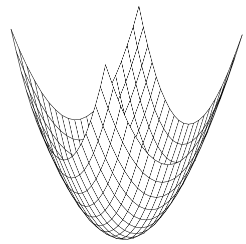
[6]The reason why more hidden units than (or as many principal components as) the number of input units need be used is because the classification task requires good interpolation of the training data and that, for the artificial domain—which is very complex relative to its dimensionality—, full interpolation necessitates more than two principal components or hidden units. In a higher-dimensional, but relatively simpler problem, the number of hidden units necessary to interpolate the training set might be smaller than the number of input units (e.g., see the experiments on real world domains of Chapter 4).

[7]The error ratio used in these experiments is the same as the one already mentioned in Section 3.3.2 of Chapter 3.

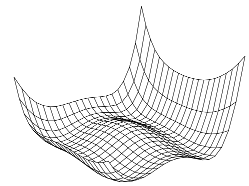
[8]The reconstruction error obtained by PCA on the artificial domain of Chapter 4 is "zero" for every point because two principal components are sufficient to reconstruct perfectly a two-dimensional domain.



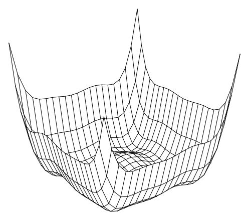
(a) L



(b) L-L



(c) NL-L



(d) NL-NL

Figure 5.2: 3-D Plots of the relative error ratio surfaces. While the nonlinear methods construct several local valleys, the linear methods build a single global valley.

might seem a little surprising in light of [Baldi and Hornik1989]'s demonstration that linear autoassociation is equivalent to PCA, it is important to bear in mind that [Baldi and Hornik1989]'s discussion pertained to ideal autoassociators, i.e., autoassociators that have reached full convergence. According to [Baldi and Hornik1989], if a linear autoassociator has not fully converged, then it will get stuck in a saddle point and its solution will not be equivalent to that of PCA. As can be seen from Figures 5.2(a) and (b), this is precisely what happened in the case of the two linear autoassociators I considered in this study. Indeed, the parabola shape of the reconstruction surface of these linear devices suggests that they both output a constant vector corresponding to the coordinates of the mean of the four concept class training clusters.[9]

[9]Nevertheless, full convergence of the two linear autoassociators can be obtained by increasing the learning rate of the backpropagation procedure from 0.05 to 0.1. When such a step is taken, the reconstruction surface of these autoassociators represents, as expected by the results of [Baldi and Hornik1989], a flat surface, just like PCA's. For classification purposes, however, the reconstruction error surfaces obtained by a linear autoassociator that has only partially converged is more interesting than the flat reconstruction error surfaces obtained by

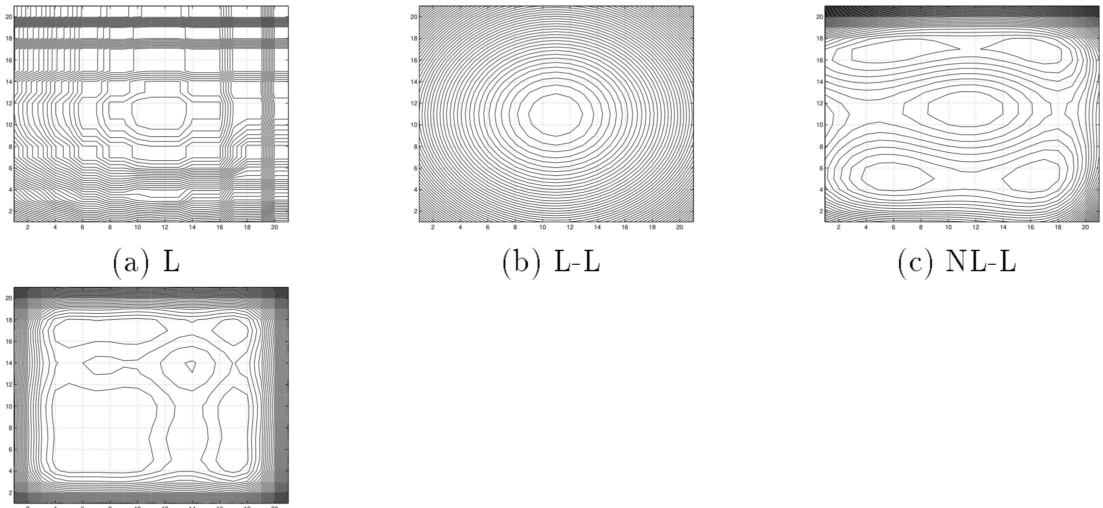
The second result concerns the comparison between nonlinear autoassociation versus linear autoassociation (prior to full convergence) as well as the specialization bias of the various autoassociators. In particular, the comparison of Figures 5.2(a) and (b) versus Figures 5.2(c) and (d) reveals that the error surfaces computed by the two nonlinear methods are qualitatively different from the ones computed by the linear autoassociators: while the nonlinear autoassociators are capable of building multiple-local-valley representations of the underlying domain data (Figures 5.2(c) and (d)), the linear autoassociators can only use a single global valley in their representation (Figure 5.2(a) and (b)). This suggests that the solution computed by the linear autoassociators for the internal counter-concept cluster assumes a uni-modal interpolation bias, whereas the solutions computed by autoassociators with nonlinearities in their hidden layer assume a multi-modal interpolation bias. These interpolation biases identify the specialization mechanisms of the different autoassociators which are thus shown to be different according to whether linear or nonlinear units are used in the hidden layer.

The results of this section have identified the type of specialization mechanism used by nonlinear autoassociators to circumscribe the instances of the conceptual-class. In particular, they have shown that isolated conceptual clusters are separated from each other and bounded by concentric circular boundaries constituting the contour lines of each cluster (more generally, these boundaries are ellipsoidal, depending on the dispersion pattern of the data [Bourassa1999]. This interpretation of the results of Figure 5.2 can be better explained in the equivalent representation of Figure 5.3 which illustrates the contour lines (50 of them) for the L and L-L autoassociators prior to full convergence as well as the contour lines for the NL-L and NL-NL autoassociators. The reconstruction error of a particular point depends on which of those contour lines it is located on, which in turn represent distances from the clusters' means.[10]

In summary, the important results of this section is the realization that the specialization mechanisms used by nonlinear autoassociators are different from those used by linear autoassociators (prior to full convergence) which, rather than separating them, build contour lines around a single surface comprising an average of the four concept-clusters of the domain. Furthermore, they are different from those used by PCA or linear autoassociators which have converged fully, since those systems do not perform any type of discrimination on the domain

the linear systems once they have reached full convergence. This is because, while no discrimination can occur in the case of a flat surface, a subclass of the counter-concept class (the external counter-concept examples) gets discriminated against the concept class in the case of a parabolic surface. For this reason, the remainder of this chapter considers the solutions obtained by linear autoassociators that have only partially converged, rather than those obtained by the linear autoassociators that have had a chance to converge fully.

[10] Thus, in this representation, setting a threshold corresponds to selecting one level of contour lines and considering as instances of the concept any point contained within these lines and rejecting as non-conceptual, any point outside these boundaries.



(a) L (b) L-L (c) NL-L (d) NL-NL

Figure 5.3: 2-D Contour Plots of the relative error ratios. While the nonlinear methods build several clusters, the linear method builds a single one.

considered. The next section is devoted to resolving the disagreement between the conventional wisdom regarding nonlinear autoassociation and PCA. It is followed by a discussion about the effect of the nonlinear and linear autoassociative interpolation biases on the classification problem.

5.2.3 Discussion

Analyzing the results obtained in the previous section provides an explanation for the departure of my results from the expectations derived from the past literature.

The discrepancy between the results obtained by linear autoassociation and PCA were already explained in the previous section. In particular, I pointed out that [Baldi and Hornik1989]'s results assume that the linear autoassociator they study have fully converged. This is not the case in the linear autoassociators considered in this chapter.

As far as nonlinear autoassociation versus linear autoassociation and PCA is concerned, the difference between the linear and nonlinear systems can be explained by the fact that, for the particular test domain considered, [Bourlard and Kamp1988]'s assumption has been violated. Indeed, as discussed in section 5.2.1, [Bourlard and Kamp1988]'s proof was shown to not apply to the case where the inputs to the nonlinear activation function of the hidden units are large. An observation of the pre-synaptic activations of the 16 hidden units of NL-NL reveals that on average, these values are equal to $\mu = 0.6953$ with variance $\sigma^2 = 0.1919$. Similarly, for NL-L, I found that $\mu = .3222$ and $\sigma^2 = .0548$. Although for input values in the [-1,1] interval, the sigmoid function is close to linear, my results seem to suggest that a small violation of the [Bourlard and Kamp1988] assumption can make a big difference in certain contexts: while this difference might be marginal and thus overlooked when considering the hidden layer alone, it is magnified at the output layer, as suggested by the plots of figures 5.2(a) and (b) versus those of figures 5.2(c) and (d).

5.2.4 Consequences for Classification

I now describe the consequence of the results of the two previous sections on the task of classification. In particular, I report on the average positive and negative percent classification error obtained by the five different systems on the artificial test domain of section 4.5 of Chapter 4. Like in the previous experiments, the connectionist systems are trained for 2000 epochs and those with a hidden layer use 16 hidden units. The linear autoassociators are trained with a learning rate of 0.05, thus, once again, not fully converging to the same solution as PCA. PCA is still used with two principal components. The threshold-determination method used by the five systems is the Parametric Recognition technique described in Section 3.3.2 of Chapter 3. The average results obtained on the five experiments are reported in the graph of Figure 5.4.[11]

The graph indicates clearly the difference between the linear and nonlinear classification methods: PCA, L and L-L are all shown to obtain an average error rate in the upper 20 to 50 percent while NL-L and NL-NL both obtained an average error rate much below 10 percent. Note that there is a small difference in accuracy between NL-L and NL-NL which suggests that NL-L might be more appropriate than NL-NL for classification. Nevertheless, this difference is minimal and does not affect the qualitative results of this dissertation which is devoted to the study of the [Hanson and Kegl1987] NL-NL classifier.

As discussed in the previous section which displayed the reconstruction error surface of the different systems, the surface of the nonlinear systems are qualitatively different from those computed by the linear ones prior to convergence: while the linear approaches build a unique global valley, the nonlinear systems are able to construct four local valleys that correspond to each positive component. This difference can explain the accuracy results reported in Figure 5.4. In the linear systems, since only a single valley is allowed, its lowest point has to be located inside the square spanned by the positive components (in order for the positive zone to include all four positive components most efficiently). This means that the negative data in the center and on the sides of this square get wrongly classified as positive. In the nonlinear systems, since the error surface may include several local valleys, the relative error ratios of the negative components can all be appropriately high relative to their positive counterparts whether they are located in the interior, the sides, or the exterior of the square underlying the positive components. Thus all the negative data get appropriately differentiated with respect to the positive data.

---

[11]Despite the fact that the positive and negative testing sets have different cardinalities, the positive classification error and the negative classification error were each made to account for 50% of the classification error.

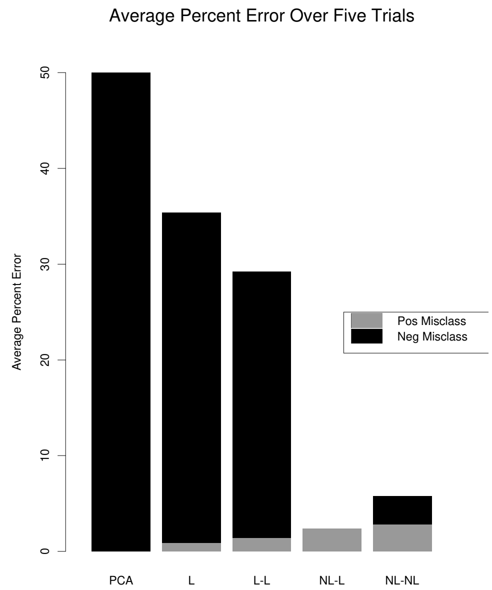


Figure 5.4: The Results of the Learning Task for the five different systems considered: PCA, L, L-L, NL-L, and NL-NL. The results are averaged over five trials.

Note that in this unsupervised and recognition-based approach to classification, the autoassociator's role is to perform non-parametric density estimation. It is important to keep in mind that the same approach could be used in conjunction with non-parametric density estimation techniques other than the one implemented by the autoassociator. For example, the Parzen density estimation technique (see [Fukunaga1990]) could be used to classify a point x as follows: if the number of (concept) training examples present in the small local region $L(x)$ (of fixed volume v) surrounding x is smaller than a given threshold T, then x is a counter-example of the concept. Otherwise, it is a positive example of the concept. The disadvantage of the Parzen density estimation method over the density estimation method implememnted by the autoassociator, however, is that it requires storage of all the training examples. On the other hand, it has the advantage of being much simpler to analyze than the technique used by the autoassociator and its study could be of great interest for a more theoretical approach to the comparison of discrimination versus recognition approaches to classification. Other non-parametric density estimation methods could also be used that do not require storage of the training data (e.g., radial basis function estimation).

5.3 Specialization and Network Capacity

The previous section has identified the type of specialization mechanism used by nonlinear autoassociators. In particular, it has established that nonlinear autoassociators circumscribe each conceptual data cluster of the domain separately. These results suggest that classification is possible in the autoassociator despite the absence of counter-examples and prior statistical and contextual information because the autoassociator possesses an internal bias which provides the necessary specialization. Since, as discussed in Section 2.5 of Chapter 2, capacity monitoring is a way to regularize the Bias/Variance Dilemma, it is interesting to find the effect of such regularization in the context of autoassociation-based classification, i.e., to assess the flexibility of NL-NL's specialization mechanism. In order to reach a greater generalization level of the results, the experiments of this section are performed on both the artificial domain used throughout the previous section and its generalization to 3 dimensions.

This section is composed of two subsections. Section 5.3.1 introduces the 3-dimensional domain extension and Section 5.3.2 describes the effect of varying the capacity of the network for the reconstruction error of counter-conceptual clusters in the two domains.

5.3.1 3-D Domain Description

The artificial domain I have tested so far is two-dimensional. While it is already known from the results of Chapter 4 on real-world domains that the NL-NL autoassociator is capable of classifying high dimensional domains, the purpose of this 3-dimensional extension is to test how the specific results obtained on the 2-dimensional artificial domain can generalize. The domain I generated is illustrated in Figure 5.5. Black stars correspond to instances of the concept and are located at the eight corners of the cube (displayed for clarity purposes). White circles correspond to negative instances of the concept.

In more detail, the 3-D domain used in this section is composed of eight positive components whose means are located at points: (.2,.2,.2), (.2,.8,.2), (.8,.2,.2), (.8,.8,.2), (.2,.2,.8), (.2,.8,.8), (.8,.2,.8), (.8,.8,.8). The means of the 27 negative components are located at points: (.5,.5,.5), (.5,.5,.2), (.2,.5,.5), (.5,.5,.8), (.8,.5,.5), (.5,.2,.5), (.5,.8,.5), (.1,.1,.1), (.1,.9,.1), (.9,.1,.1), (.9,.9,.1), (.1,.1,.9), (.1,.9,.9), (.9,.1,.9), (.9,.9,.9), (.2,.5,.2), (.5,.2,.2), (.5,.8,.2), (.8,.5,.2), (.2,.5,.8), (.5,.2,.8), (.5,.8,.8), (.8,.5,.8), (.2,.2,.5), (.2,.8,.5), (.8,.2,.5), (.8,.8,.5). Each mean actually stands for a cluster of fifty data points normally distributed around that mean and of variance $\sigma^2 = 0.01$.

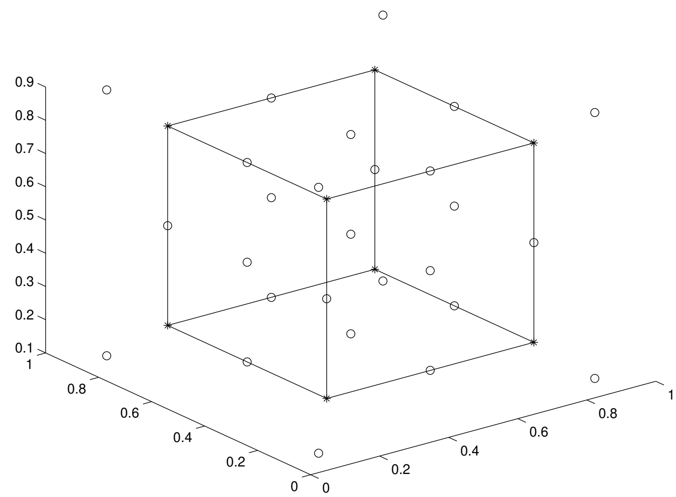


Figure 5.5: A three-dimensional extension of the Artificial domain generated in Chapter 2.

5.3.2 Effect of Capacity Changes on Reconstruction

The purpose of this section is to observe the performance of the NL-NL autoassociator—as far as reconstruction error of counter-conceptual data is concerned—with different numbers of hidden units on the 2- and 3- dimensional domains already described. The goal of these experiments is to study the flexibility of the multi-modal interpolation bias of the NL-NL autoassociator. Figure 5.6 presents the results obtained on the 2-dimensional domain whereas Figure 5.7 presents those obtained on the 3-dimensional domain.

In more detail, Figure 5.6 plots the sum of the error ratios (as defined in Section 5.2.2 above) of the "internal" negative data (points centered at (.5, .5), (.2, .5), (.5, .2), (.5, .8), and (.8, .5)) for 1, 2, 4, 8, 16, 32, and 64 hidden units on a single trial. The "external" negative data (points centered at (.1, .1), (.1, .9), (.9,.1) and (.9,.9)) were ignored since their reconstruction error is very high and thus discrimination against them is less sensitive than in the case of the "internal" negative data. Each capacity is represented by a pair of parallel bars. The left gray-filled bars indicate the error ratio of the "central" negative component (centered at (.5, .5)) and the right black-filled bars represent the average error ratio of the four side components (centered at (.2, .5), (.5, .2), (.5, .8) and (.8, .5), respectively). Figure 5.6 shows that as the number of hidden units grows towards an optimal value (4 hidden units)[12], the error ratios of the "side" and "central" negative data increase, but that past this point, they keep on decreasing (except for the slight increase of the central negative component in going from 16 to 32 hidden units. This increase, however, is negligeable).

[12]Please note that this result is obtained on a single trial. The selection of an optimal capacity (which amounted to 16 hidden units) was obtained on the average of several runs of the system.

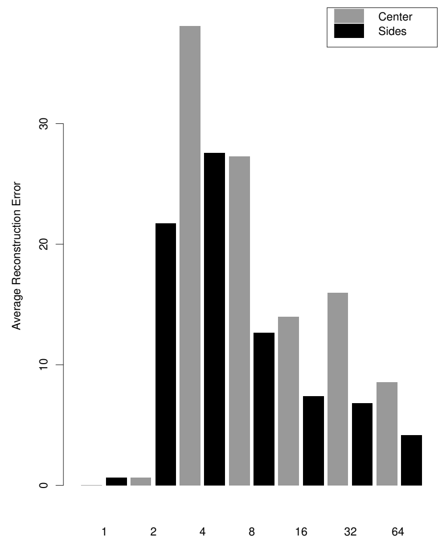


Figure 5.6: The Reconstruction Errors of the "Sensitive" Negative Data in the 2-D domain. Past a sufficient capacity, the reconstruction error decreases as the capacity increases. The largest reconstruction error for both the side and central components is reached when using H=4 hidden units.

Figure 5.7 presents similar results but in the case of the 3-dimensional domain. There, I differentiated between three data types: the "central" negative component (centered at (.5, .5, .5)) represented by a dark gray bar, the six "face" components (centered at each points containing ".5" twice in the location triplet) represented by a clear gray bar, and the twelve "side" components (centered at every point containing a single ".5" in the location triplet) represented by a black bar. Once again, the error ratio of the "internal" negative data is shown to increase until optimal values are reached and to decrease past these values. This time, the optimal capacity, as far as discrimination against the central component is concerned is reached only with 16 hidden units, while it is reached with 4 hidden units, again, for both the faces and side components. [13]

These results are useful for two reasons. First they suggest an advantage of the NL-NL autoassociator not observed in the previous section. Namely, they show that the specialization mechanism of the autoassociator can be regularized by means of controlling the size of the hidden layer. The highest degree of specialization is obtained by using the number of hidden units yielding the largest

[13]Note that the best classification rate obtained for the 3-D domain is obtained with 4 hidden units and reaches 6.9 % error (2.8% of positive misclassifications and 4.1 % of negative misclassification). This can be expected in this domain since the set of face and side components well discriminated against with 4 hidden units includes 18 subsets of 50 points each, whereas the set of center components well discriminated against with 16 hidden units includes a single subset of 50 points.

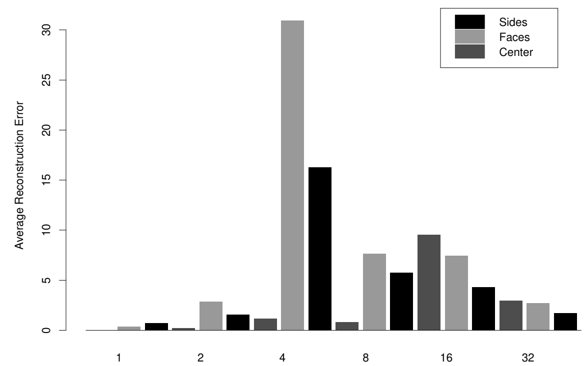


Figure 5.7: The Reconstruction Errors of the "Sensitive" Negative Data in the 3-D domain. Past a sufficient capacity, the reconstruction error decreases as the capacity increases. The largest reconstruction error for the face and side components are reached when using H=4 hidden units, whereas the largest reconstruction error for the central component is reached when using H=16 hidden units.

reconstruction error, while the amount of specialization is decreased by using a larger capacity. Controlling the degree of specialization is practical since certain domains require high degrees of specialization: For example, in the problem of discriminating between a dog and a wolf, the concept of the dog must be highly specialized so as not to include wolves; conversely, in order to include penguins in it, the concept of a bird must be weakly specialized.

A second effect of these results, however, is to display a limitation of the nonlinear autoassociation scheme. Since different degrees of embedding of the negative data (central component (embedded in a 3-D structure), face components (embedded in a 2-D structure) and side components (embedded in a 1-D structure)) reach their optimal performance—as far as reconstruction is concerned—at different capacities, it is expected that classification using the NL-NL autoassociator cannot yield perfect accuracy in domains composed of different degrees of embedding in the negative data.

Chapter 6
Efficiency and Generalization Considerations

The experiments of chapter 4 suggested that autoassociation-based classification differs from MLP in two respects. First, it was shown that the two systems do not necessarily display the same degree of accuracy, and second, it was shown that they do not necessarily display the same efficiency. More specifically, the autoassociator was shown to be more accurate than MLP and other discrimination-based systems on two of the three real-world domains tested (it is as accurate as the most accurate discrimination-based system on the third) and to learn the artificial domain of Section 4.5 significantly faster than MLP. The next chapter will explore the difference in accuracy between the two approaches. The purpose of this chapter is to characterize the learning mechanisms of the autoassociator and of the MLP network in order to explain their difference in efficiency and to show that, despite the difference in learning strategies, like MLP, the autoassociator does perform a satisfactory amount of generalization.

I will begin by demonstrating experimentally that, indeed, the two systems use different learning strategies and by explaining how this difference in strategies affects the efficiency of the two approaches. I will then demonstrate experimentally that although the learning strategy of the autoassociator is data-driven, the autoassociator does not use all its ressources to characterize the training data and thus must generalize from them. I will conclude by showing that the amount of generalization performed by the autoassociator is restricted enough so as to avoid over-generalization and to permit accurate classification. Aside from explaining the difference in efficiency between MLP and the autoassociator, this chapter thus explains more deeply than Chapter 5, the mechanisms underlying autoassociation-based classification.

More specifically, the experimental strategy used in this chapter consists of monitoring the changes in activation of the hidden units of the two networks both as a function of time alone and in relation with one another. The point of departure of this study is the observation that although both the MLP network and the autoassociator are ultimately used to solve the same classification problem, the two networks are not *directly* applied to the same task: while the MLP network is expected to learn how to group all the training data into two categories (concept and counter-concept classes), the autoassociator is expected to assign each training example to a distinct output (itself). In other words, the MLP network is constrained to deal with a binary classification task while the autoassociator must deal with a recognition problem, i.e., a multi-class classification problem for which the number of classes corresponds to the number of training examples.

The chapter is divided into three sections. Section 6.1 introduces the experimental methodology used in all the experiments of this chapter. Section 6.2 uses some of this methodology to explain the difference in efficiency between the autoassociator and the MLP network. In particular, it discusses whether the difference in direct task definition is what causes the difference in efficiency observed between the two approaches in our experiment of Section 4.5. Section 6.3 demonstrates that generalization does take place in the autoassociator and that the amount of generalization that takes place is restricted enough to enable classification. More specifically, I begin by evaluating the type and amount of generalization performed by the autoassociator and the MLP network on the domain of Section 4.5 in order to show that, despite their different learning strategy, they are both capable of generalization; I follow this study by showing what mechanism enables the nonlinear autoassociator not to overgeneralize and thus to perform accurate discrimination between concept and counter-concept data on the artificial domain of Section 4.5.

6.1 Experimental Methodology

The results of Chapter 4 showed that autoassociation and MLP perform differently on the task of classification. In particular, they showed that while MLP learns the artificial domain of Section 4.5 in about 3500 epochs, the autoassociator is able to learn how to classify this domain in less than 500 epochs. Yet, the two systems are feedforward networks optimized by the same optimization technique: backpropagation. The only difference between the two approaches is the type of problem to which each network is applied, and consequently, their architectures. While MLP is expected to solve a discrimination type of problem in which some instances are associated with an output value of "one" and the others are associated with an output value of "zero", the autoassociator is expected to solve a recognition problem in which the input must be reconstructed at the output layer. As discussed above, the main purpose of this chapter is to explain how this difference affects the learning strategy taking place in the networks in an attempt to shed light on the observed difference in efficiency. A secondary purpose is to show that even though the learning strategy of the autoassociator is different from that of MLP, generalization does take place and is restricted enough to allow for classification between concept- and counter-concept data. These two purposes will be achieved by monitoring the backpropagation procedure.

Monitoring the backpropagation procedure, however, is not trivial because of the large number of parameters present in feedforward neural networks. Indeed, a feedforward network composed of n input units, h hidden units and o output units has $h \times (o + n)$ weights, or free parameters that need to be monitored at various epochs of the process. In the MLP network of Section 4.5 composed of 2 input units, 16 hidden units, 1 output unit, and ran for 4000 epochs recorded every 500 epochs, this means that the evolution of 48 variables must be monitored at 8 different points in time. For the autoassociator used in this same section, the number of output units is 2 instead of 1 while all the other parameters are the same, and therefore, the evolution of 64 variables must be monitored at 8 different points in time. Such monitoring cannot usually be carried out by human beings manually since, in addition to the size of the problem, these variables are dependent upon one another. For these reasons, several experimental strategies have been proposed to help the monitoring process. Four of these methods were selected for the experiments of this chapter.

The four particular methodological tools used in this chapter, Plotting HUAPs, Plotting Boundaries, Variance Monitoring and Effectiveness Monitoring, were proposed in part by [Munro1991] and in part by [Weigend1994]. In both cases, rather than focusing on the weights of the networks which are typically very numerous, the strategies propose to monitor the activation values of the hidden units as time progresses. While not as specific as monitoring the weights, monitoring the hidden units is more tractable and has proven useful for shedding light on the learning and generalization strategy of neural networks. The next two subsections describe these different methodologies.

6.1.1 Munro's Monitoring Methodology

Munro's monitoring methodology will be useful for exploring the learning strategies employed by the autoassociator and MLP. It will also be useful for explaining how classification is enabled by the autoassociator by showing the means by which the generalization process of the autoassociator gets bounded. The strategy is composed of two parts: the first part consists of building the Hidden Unit Activation Planes (HUAPs) of the networks and plotting them. The second part consists of plotting the domain boundaries computed by the hidden-to-output layers of the networks on top of the HUAP obtained at a given epoch.

The HUAP plotting methodology was inspired by the study of natural dynamical systems (e.g., the behavior of biological organisms) and uses elements of *Stability Analysis Theory* [Hazlett and Bach1977]. The advantage of this methodology is that it permits a global analysis of the networks' learning strategies as well as an explanation of their classification abilities. Its disadvantage, on the other hand, is that, being based on visualization, it can only be used on simple problems whose solution requires only two hidden units. The boundary plotting strategy results from a direct interpretation of the networks' computations. It allows for a visualization of the networks' solutions within the HUAP. However, it too is restricted to networks of two hidden units.

I now discuss the strategy used to plot the HUAPs, followed by the strategy used to plot the boundaries associated with them. I then discuss some of the previous uses of these strategies and what I expect them to achieve in the context of this study.

Plotting HUAPs

As indicated above, plotting a neural network's HUAP is a useful monitoring method except for the fact that it is restricted to neural networks of two hidden units. The method consists of plotting the activation values of the hidden units for each input pattern as a function of one another. If the HUAP is plotted at a single epoch, then each input pattern is represented by a point, but if it is plotted over several epochs, then each input pattern is represented by a path which reveals the evolution over time of the relative activation values of the two hidden units for a given input. Convergence of the network is observed when each path has reached its final destination. Like in stability analysis theory, global properties of the system can be established by observing the paths formed in a HUAP. A classical example in stability analysis is the case of an inward spiral type of trajectory which reveals that a dynamical system is subjected to a phase of stable (or damped) oscillations. The same spiral going outward would reveal that the system is subjected to unstable (or undamped) oscillations ([Hazlett and Bach1977]). Although such trajectories will not be observed in the experiments of this chapter, other aspects of stability analysis theory are useful to bear in mind. Those include, in particular, the notions of stable and unstable fixed-points, where stable fixed-points refer to points towards which a trajectory converges and does not diverge from at any points in time and unstable fixed-points refer to points towards which a trajectory converges but from which it can diverge (abruptly) after some point in time. In the connectionist parlance, fixed-points are also known as *attractors* [Hertz *et al.*1991].

Plotting Boundaries

In a HUAP representing the state of the network at a single epoch (rather than over time), in addition to plotting the points corresponding to the representation in the hidden layer of each input pattern, the boundaries computed by the hidden-to-output layer of the network can also be plotted. Boundaries in the HUAP are represented by the weights linking the two hidden units to the output units. As a result, each output unit can be expressed as a boundary or as a family of parallel boundaries of the form:

$$w_1 s_1 + w_2 s_2 + \beta = \delta$$

where δ can assume different values corresponding to the different values taken by the output unit (thus creating a family of lines rather than a single line), w_1 and w_2 correspond to the weights linking hidden unit 1 and 2 respectively to the output unit considered, and s_1 and s_2 correspond to the activation values of hidden units 1 and 2.

Usefulness of the Plotting Strategy

Together, HUAPs and their associated boundaries can clarify greatly the learning, generalization and classification capability of a MLP network or an autoassociator. For example, considering tasks requiring only two hidden units, [Munro1991] was able to monitor the change of representation taking place in going from the input to the output layer of a MLP network on the X-OR problem as well as the organization of information taking place at the hidden layer of an autoassociator while learning the encoder problem. His observations were useful in practice in that they revealed that the addition of constrained output units can radically influence the nature of the hidden unit representations and can, in the case of MLP, speed up learning. Furthermore, his HUAP observations over time revealed the existence of near-equilibrium (or unstable attractor) states. His methodology thus allowed him to draw certain conclusions regarding the efficiency of the MLP network and to find ways in which to speed up the system. The goals of this chapter are very similar to those of [Munro1991]'s in that I too expect to observe characteristics of the learning strategies of the MLP network and of the autoassociator, to be able to explain why the autoassociator learns the artificial domain of Section 4.5 or the X-OR domain faster than the MLP network, and what enables classification to take place in the autoassociator. I thus expect [Munro1991]'s positive experience with the HUAP and boundary plotting methodology to transfer successfully to my experiments as well.

6.1.2 Weigend's Monitoring Methodology

Weigend's monitoring strategy is also composed of two parts: *variance* and *effectiveness* monitoring. This methodology can be useful to find out what sort of generalization is performed by the autoassociator and the MLP network as time progresses. In particular, it can enlighten us as to how many hidden units participate in the construction of a solution and how much of this participation is actually effective. These methodologies were developed in order to test experimentally [Moody1992]'s theoretical results, stating that the effective number of parameters of a feedforward neural network can be smaller than the actual number of parameters present in that network.

I first present the variance monitoring strategy, followed by the efficiency monitoring strategy, and I then discuss the usefulness of these monitoring methods.

Variance Monitoring

Weigend's variance monitoring strategy consists of recording and plotting the variance of each hidden unit over time as data from the entire training set are presented to the network. Low variance indicates that the hidden unit's activation does not change much as different types of data are presented, whereas high variance indicates the unit's sensitivity to different types of data. Monitoring the hidden units' variance over time is thus a good way to find out how many hidden units participate in the construction of a solution and the pattern in which this participation takes place, as far as whether the hidden units start participating in the solution at the same time and when their participation is first detected.

Effectiveness Monitoring

Although useful for indicating the variance of each hidden unit, variance monitoring is not indicative of the number of degrees of freedom actually used by the network to solve a task. The reason why variance is not capable of providing such information is because several hidden units could react strongly to some input, yet, at the same time, duplicate each other's effect, and therefore not each work effectively. To remedy this problem, Weigend proposed a measure of effectiveness of the hidden units of a network, which proceeds by computing and plotting the standard deviation associated with each of the principal components of the hidden unit space at different points in time. In more detail, Weigend used principal component analysis to return the square-roots of the significant non-zero eigenvalues of the covariance matrix of the network's hidden units. The returned values correspond to the spread of the data along the principal directions in which the hidden units are best and more concisely described. Thus, the effectiveness of the network can be monitored over time without the problem of hidden units duplicating one another's effect.

Usefulness of these two Measures

Using the variance monitoring measure, Weigend was able to find out that in discrimination tasks, the hidden units of a network do not immediately differentiate, but that rather, they remain highly correlated during the first phase of training. Furthermore, using effectiveness monitoring, he confirmed experimentally [Moody1994]'s claim that the effective number of parameters—i.e., the number of parameters actually active during the training process—is smaller than the actual number of parameters present in the network. These two measures permitted a keener understanding of the computations taking place in neural networks.

These two measures should be useful in the present study, as well, since they should allow us to get some insight as to whether the MLP network and the autoassociator generalize the data of the domain on which they are trained. Since, despite the fact that the two systems are exposed to different tasks they are both expected to generalize in order to be useful classifiers, finding out that such generalization is indeed achieved by each of them is particularly relevant.

The remainder of this chapter presents the experiments that were conducted using the methodological tools just introduced and their conclusions.

6.2 Explaining the Differences in Efficiency

The purpose of this section is to find out whether the difference in task definition for the autoassociator and the MLP network is what affects the learning efficiency of the two methods, and if so, how.

These questions are answered using the HUAP plotting methodology over time of [Munro1991] which can effectively contrast the learning approach used by the autoassociator to that used by the MLP network. Indeed, such a methodology permits a direct observation of the dependency relations between the hidden units of the two networks and, thus, allows for an explanation of what the learning strategy of each network is.

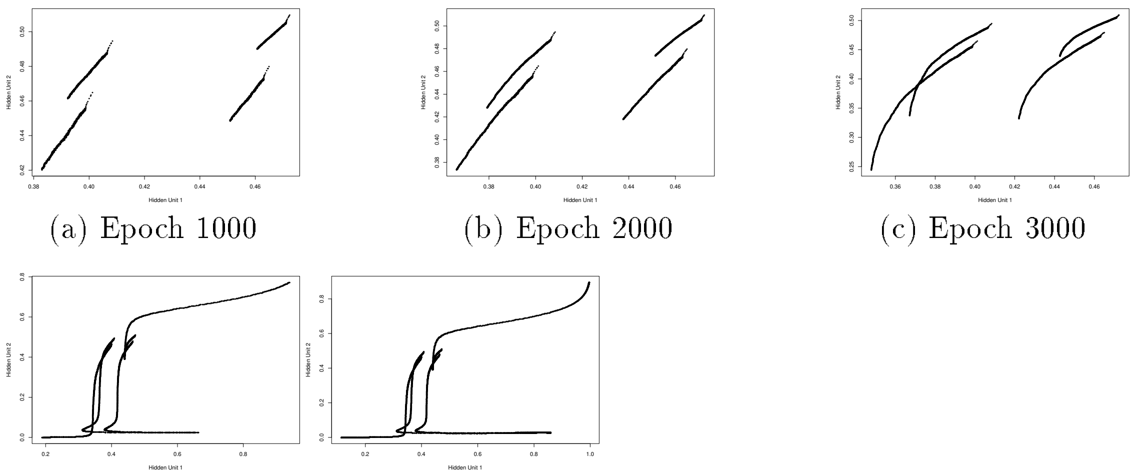
6.2.1 Applying Munro's HUAP Monitoring Strategy

The experiments of this section consist of plotting the HUAPs of both the MLP network and the autoassociator over time, and observing the trajectories followed by the HUAP representation of each input to the network. The purpose of these experiments is to observe the differences between the MLP network and the autoassociator in order to explain what strategic goal they, each, pursue.

In this study, the learning strategy of the MLP network and the autoassociator are contrasted with respect to the XOR problem since the artificial domain of Figure 4.5 is too complex to be completely solved by a network of only two hidden units. In the XOR problem, each data point is composed of two boolean features ("zero" and "one") and an input is considered positive if one of its features is a "one" and the other a "zero". It is negative otherwise.

The results of these experiments are displayed in Figures 6.1 and 6.2. Figure 6.1 shows what happened on a 5000 epoch run of the MLP network on this 4-input domain. The figure is composed of 5 different graphs each illustrating the state of the attractor space at a given epoch. Figure 6.2 is similar but for the autoassociator and on a run of 2000 epochs which, like in the artificial domain of Section 4.5, proved sufficient. Note that on each graph displayed within each figure, the scaling is different.

In each of these graphs, the trajectories displayed correspond to the change in state of the network in relation to a single data point, as time progresses. Each input datum is responsible for a different trajectory in that space. The graphs of Figure 6.1 show that for at least 3000 epochs, the trajectories caused by each of the four input data to the MLP network are parallel and tend towards the same general area. In other words, all input values were originally located in the basin of attraction (i.e., the area influenced by a single stable fixed-point or attractor) of the same single attractor and thus follow the same direction. By epoch 4000, however, the picture is very different. At this point, the single trajectory previously observed divides into three separate trajectories, one tending towards the upper right corner of the HUAP, and the other two directed toward its lower left and right corners. In other words, these plots show that after epoch 3000, the



(a) Epoch 1000 (b) Epoch 2000 (c) Epoch 3000

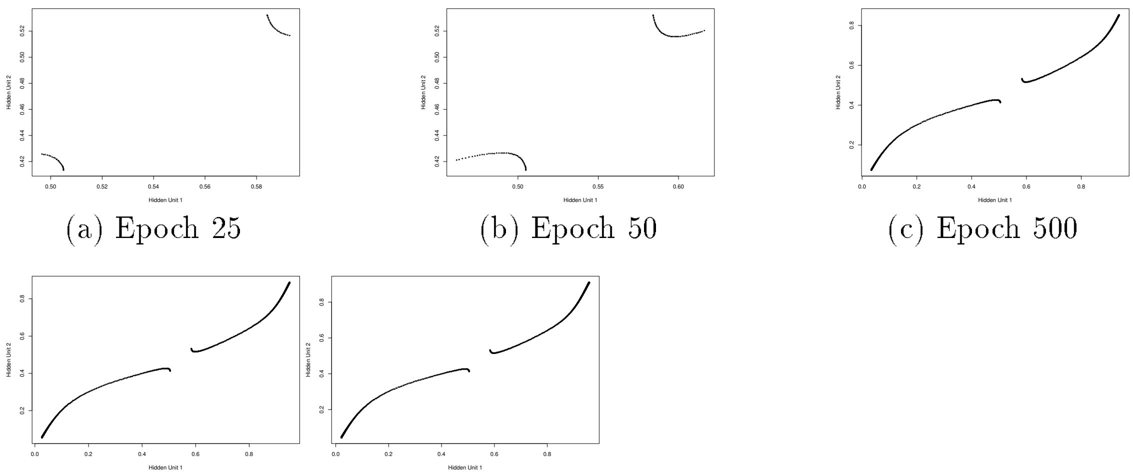
(d) Epoch 4000 (e) Epoch 5000

Figure 6.1: The trajectories followed by the MLP network at 5 different points in time and in function of the two positive and two negative examples of the X-OR problem. Each trajectory corresponds to the change in time of the activation values of the two hidden units of the system for a single example. It is interesting to see how, at first, the four examples draw the network into the same direction towards the same attractor and that by epoch 4,000, two of the four trajectories merge and follow the same attractor while the other two follow different ones.

input data to the MLP network have been divided into three parts: two of the same class containing a single input pattern each and another one of the opposite class, containing two input patterns. The figure displayed for epoch 5000 is very similar to the one displayed for epoch 4000, showing that the network has reached a stable solution by epoch 4000.

Translated into dynamical system jargon for neural networks, these plots suggest that the HUAP of the MLP network contains both unstable and stable attractors. It contains a single unstable attractor which corresponds to the point towards which all the trajectories were directed until some time between epoch 3000 and 4000 and it contains three stable attractors which correspond to the three separate points towards which the trajectories tend after epoch 4000.

The graphs of figure 6.2 display a very different pattern. They show that from the beginning, the trajectories of the two patterns input to the autoassociator are differentiated. They both begin in the vicinity of the center of the HUAP and start by following an inward direction, but then quickly reverse course and tend towards the upper right and lower left corners of the HUAP. The two trajectories are perfectly symmetrical and they tend, respectively, towards two of the furthest points in the HUAP. Convergence occurs very quickly since by epoch 500, the trajectories are close to having reached their final destinations. Again, these plots suggest that the HUAP of the autoassociator contains both unstable and stable attractors. It contains one unstable attractor towards which the two trajectories



(a) Epoch 25 (b) Epoch 50 (c) Epoch 500

(d) Epoch 1000 (e) Epoch 2000

Figure 6.2: The trajectories followed by the autoassociator at 4 different points in time and in function of the two positive examples of the X-OR problem. Each trajectory corresponds to the change in time of the activation values of the two hidden units of the system for a single example. It is interesting to see how, almost from the beginning, the two trajectories are drawn in the opposite direction and that the attractor points of each are located as far as possible from each other.

gravitate during the first few epochs of the learning process and two stable attractors that are reached by the two trajectories as early as epoch 500. The unstable attractor of this figure is very weak (in terms of attraction capability), however, since it is abandoned after only a few epochs.

6.2.2 Discussion

The purpose of the HUAP experiments just performed was to explain the learning strategies of the MLP network and the autoassociator so as to derive an explanation for their difference in efficiency. Two observations can be made from these experiments:

1. The attractor space of the MLP network contains a strong unstable attractor and three weaker stable attractors.

2. The attractor space of the autoassociator contains a weak unstable attractor and two strong stable attractors.

The mentions of strong or weak attractors are introduced in this description to give a qualitative sense for the number of epochs spent by the systems in reaching an attractor regardless of any other attractor. If the system spends many epochs trying to reach an attractor without being disturbed from its course, then this attractor can be described as strong. If, on the other hand, the course of action of the system is quickly diverted from an attractor or if the attractor is temporarily invisible to the system, then this attractor is weak.

The results I have obtained can help to qualitatively differentiate the learning strategies of the two networks. In particular, based on the observations, it can be seen that the MLP network uses a *hypothesis-driven* strategy in which, during the first phase of learning, the network attempts to group all the inputs (whether instances of the concept or counter-examples) into the same cluster. This can be seen from the fact that for up to epoch 3000, the HUAP representation of every input point follows the same trajectory. Because the destination of these trajectories is an unstable attractor, however, the trajectories separate and tend towards three different stable attractors. This corresponds to the creation of a new theory in the hypothesis-driven search of the generalization space. Although the autoassociator also starts by searching the space using a hypothesis-driven strategy (in the first phase of learning which is directed towards a weak attractor), this strategy is quickly abandoned in favor of a *data-driven* strategy in which, rather than trying to group all the input data together, the system uses its resources to specify all the input data separately, as necessary.

When considering the difference in nature of the tasks tackled by the MLP network and the autoassociator, these results are not surprising. Indeed, the task requested of the MLP network is a task of discrimination between two classes. Since there are more training data than classes, such an approach requires a hypothesis-driven strategy because a theory has to be created when none existed before, in order to classify all the data into the two abstract classes. The task requested of the autoassociator, on the other hand, is one of recognition. Since, in this case, there are as many training data as classes, such an approach is better tackled by a data-driven strategy that does not necessary attempt to abstract the training data away.

The interesting and novel aspect of these two different approaches, however, is that the hypothesis-driven strategy can take significantly more time than the data-driven strategy since, as shown in Figures 6.1 and 6.2, the autoassociator converges in no more than 500 epochs while the MLP network takes between 3000 and 4000 epochs to converge. Although novel, these observations can be explained as follows: the hypothesis formulated by the MLP network at the onset is not necessarily related to the sort of final hypothesis sought by the network that will be able to explain the data. While the training data input to the network attempts to divert the MLP network away from its original (erroneous) hypothesis, this process can take a very long time, probably because of the small incremental effect applied by the data to the network at each epoch. The learning strategy of the autoassociator, on the other hand, allows the system to converge rapidly and this is because the network does not need to formulate a "correct" (previously unavailable) hypothesis like MLP, but rather, it needs to find a way to recall each training data point, possibly by rote learning. Such a process is simpler than the process of abstracting the data, and thus, it is also quicker.

The results of this section are, therefore, useful for explaining the difference in efficiency observed between the MLP network and the autoassociator. They suggest that this difference is caused by the difference in task assigned to each network. However, these results also raise a concern regarding the autoassociator: although the task directly assigned to the autoassociator does not require any amount of generalization, the overall problem of classification to which the autoassociator is applied does require generalization, in order to avoid overfitting the data. Thus, it would be interesting to find out 1) whether the autoassociator performs any sort of generalization despite its data-driven strategy; and 2) whether, if so, we can be sure that it does not overgeneralize, and thus that it classifies the data correctly. These two questions will be studied in the next section.

6.3 Accounting for Generalization and Classification

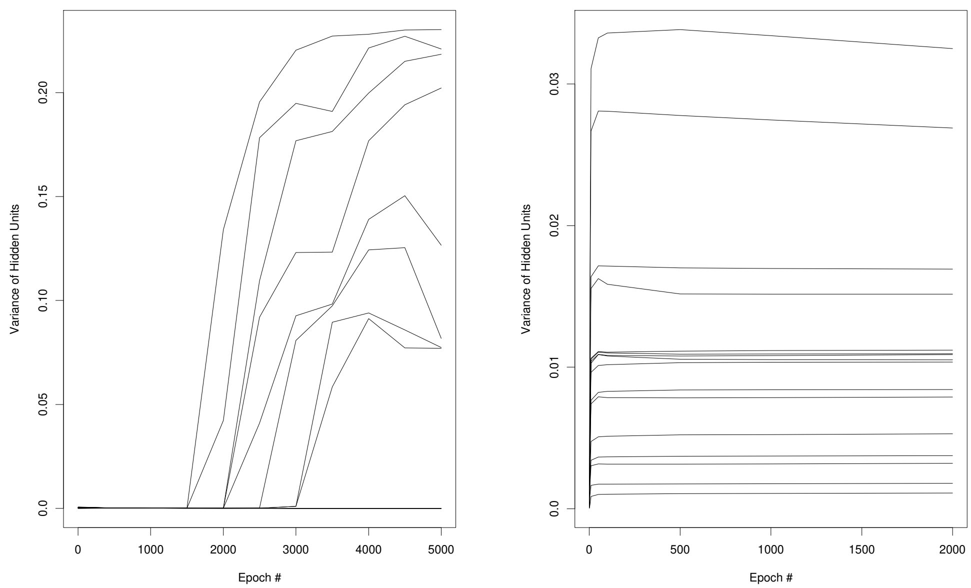
As just mentioned, the purpose of this section is to address the problem of generalization in the autoassociator. In particular, I investigate the role of the hidden units in the learning process, so as to find out the extent of the generalization process and the classification mechanisms of the autoassociator. I first compare the generalization strategy of the autoassociator to that of MLP and I then investigate the mechanisms by which the autoassociator manages not to overgeneralize and, therefore, to classify the data correctly.

6.3.1 Generalization in the MLP Network and the Autoassociator

In this section, I compare the generalization strategies of the autoassociator and MLP network using two different hidden unit capacities: 16 hidden units, the capacity shown in Chapter 4 to be optimal for the two systems applied to the artificial domain of Section 4.5, and 64 hidden units, a much larger capacity tried to test the effect of capacity increase on generalization. The section is divided into two parts. The first part describes the results obtained using Weigend's variance monitoring method while the second describes the results obtained using his effectiveness monitoring method. All the experiments of this section are conducted on the artificial domain of Section 4.5.

Applying Weigend's Variance Monitoring Strategy

The study reported in this section was conducted on single trials of the artificial domain of Section 4.5. The autoassociator was ran for 2000 epochs while MLP was ran for 5000 epochs since those were times observed, in Chapter 4, to be more than sufficient to solve the classification problem posed by the artificial domain of Section 4.5. All the experiments were repeated four more times each



(a) MLP (b) Autoassociation

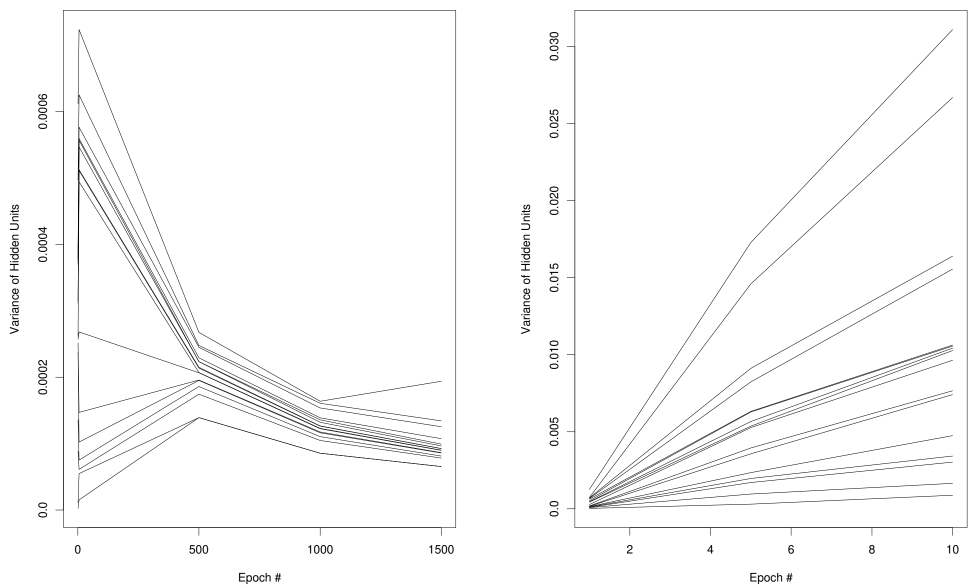
Figure 6.3: The variances of the hidden units of the MLP network and the autoassociator on the artificial task with 16 hidden units.

with different initial weight conditions in order to confirm the findings. These additional results are reported in Appendix B.[1]

The results of applying Weigend's variance monitoring strategy to a single trial of the problem are shown in Figure 6.3, 4 and 5. The values plotted in the graphs of figure 6.3 show the variances of the hidden units of the two systems over their respective training sets for a model size of $h = 16$ hidden units. Figure 6.4 magnifies these plots by displaying only the first 1500 epochs for the MLP network and the first 10 epochs for the autoassociator. Figure 6.5 displays the same results as figure 6.3 but for $h = 64$ hidden units. In all graphs, each curve represents the changes in variance of a single hidden unit as a function of time.

The results of figure 6.3, 6.4 and 6.5 show that for both the cases of a model size of $h = 16$ and $h = 64$, the hidden units of the MLP network and the autoassociator evolve differently. While the variances of the hidden units of the MLP network decrease or remain constant for the first 1500 epochs, those of the autoassociator start increasing as soon as learning begins and remain constant once a certain (small) number of epochs has been reached. After 1500 epochs have been reached in the MLP network, the variance of eight of its hidden units start increasing one after the other. The capacity of the MLP network is shown to influence the operation of the system—though not in terms of efficiency—since in the case of

[1]In one case—the case of the autoassociator with 64 hidden units—, one of the additional results is qualitatively different from the other four including the one reported in this chapter. This divergence will be discussed in the discussion section, below.



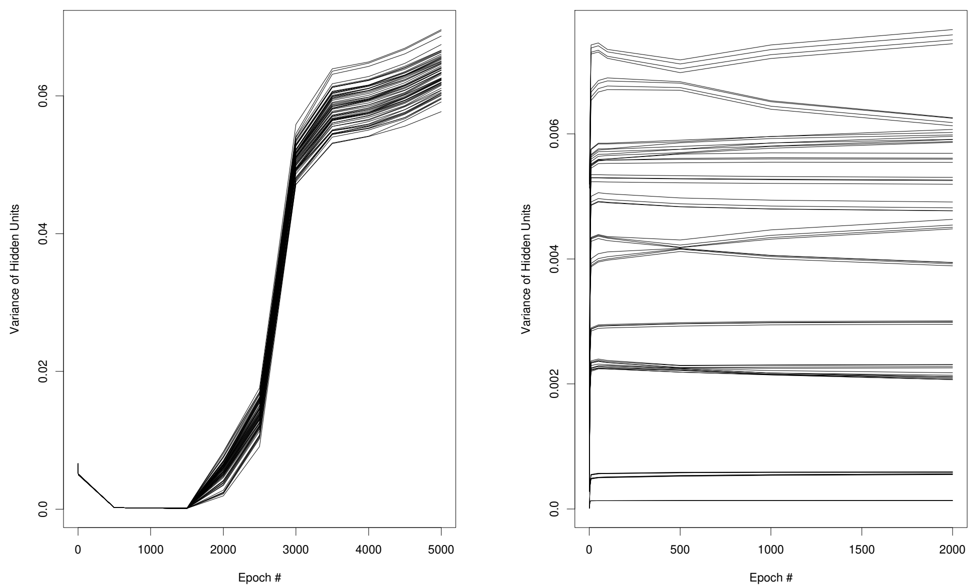
(a) MLP (b) Autoassociation

Figure 6.4: The variance of the hidden units of the MLP network and the autoassociator on the artificial task with 16 hidden units before effective learning has started.

$h = 16$, only half of the hidden units exhibit a variance greater than zero after 3000 epochs have been reached, whereas in the case of $h = 64$, all of them do so. In the autoassociator, on the other hand, no qualitative difference is exhibited between the cases of $h = 16$ and $h = 64$ since in both cases all the hidden units exhibit almost immediate positive variance and sustain it throughout the learning process. All these experiments have been duplicated four times each in Appendix B. They all came out qualitatively equivalent, except for one which will be discussed below. The results of this section will be analyzed in greater depth in the discussion section, below.

Applying Weigend's Effectiveness Monitoring Strategy

Rather than limiting my experiments to the ones just described, I followed Weigend's approach [Weigend1994] and assumed that although in some cases all the hidden units display a non-negligible level of variance, the true effective dimension (also called "intrinsic" dimension, in the pattern recognition community) used by the networks does not correspond to this activity since the units could be duplicating each other's work. Thus, the *true* effective number of hidden units must be extracted by using a specialized dimensionality-reduction method such as *Principal Component Analysis*. Using this method and following Weigend, I computed and plotted the principal components of the hidden unit ensemble against time. The graphs are displayed in figures 6.6 and 6.7, and represent, again, the results for



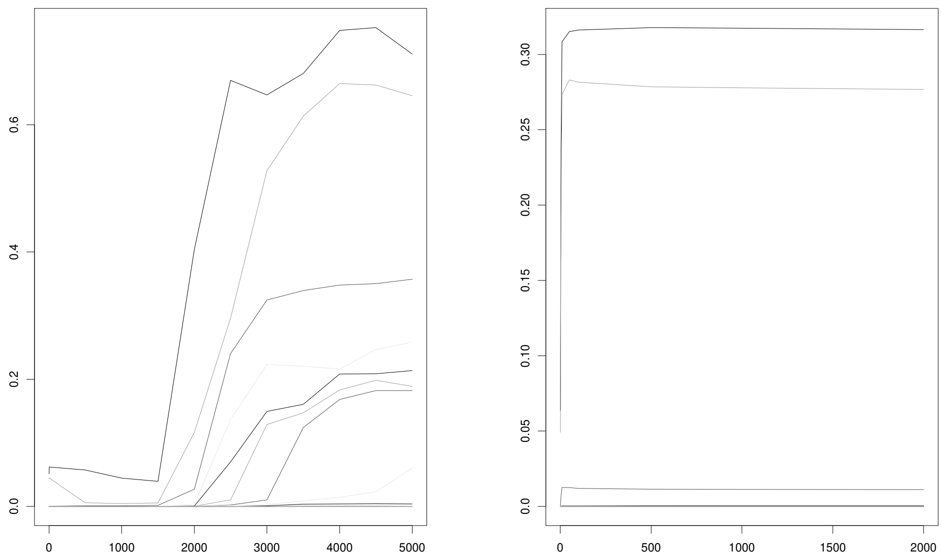
(a) MLP (b) Autoassociation

Figure 6.5: The variances of the hidden units of the MLP network and the Autoassociator on the artificial task with 64 hidden units.

16 and 64 hidden units respectively.

The results of figure 6.6 show that, in the case of $h = 16$, for both the MLP network and the autoassociator, the effective number of hidden units used by the networks is smaller than the actual number of hidden units present. The plots display a difference between the MLP network and the autoassociator, however, in that the autoassociator uses effectively only three out of the sixteen hidden units available, whereas the MLP network uses effectively eight of them. This difference can also be expressed by the fact that, while the number of effective hidden units in the MLP network matches the number of hidden units shown to display high variance in plot 6.3(a), in the autoassociator, this number goes down from sixteen to three. This suggests that, at least when using a moderate number of hidden units, while the MLP network uses a very conservative strategy, avoiding perturbation of any unnecessary hidden unit, the strategy of the autoassociator is much more expansive since all the hidden units vary but many duplicate the others' effect. Note that the notion of a conservative versus an expansive strategy concords with the results of Section 6.2, but the fact that the hidden units of the autoassociator duplicate, in great part, each other's job was not previously observed.

The results of figure 6.7 show that a drastic increase in the number of hidden units present in the network does not seem to affect their effective behavior in the case of the autoassociator. Despite the increase in the actual number of hidden units, like in the case of $h = 16$, only three of them remain effective in the autoassociator. Such an increase, however, does affect the MLP network



(a) MLP (b) Autoassociation

Figure 6.6: The principal components of the hidden units of the MLP network and the autoassociator as a function of the epoch number. The model size is $h = 16$.

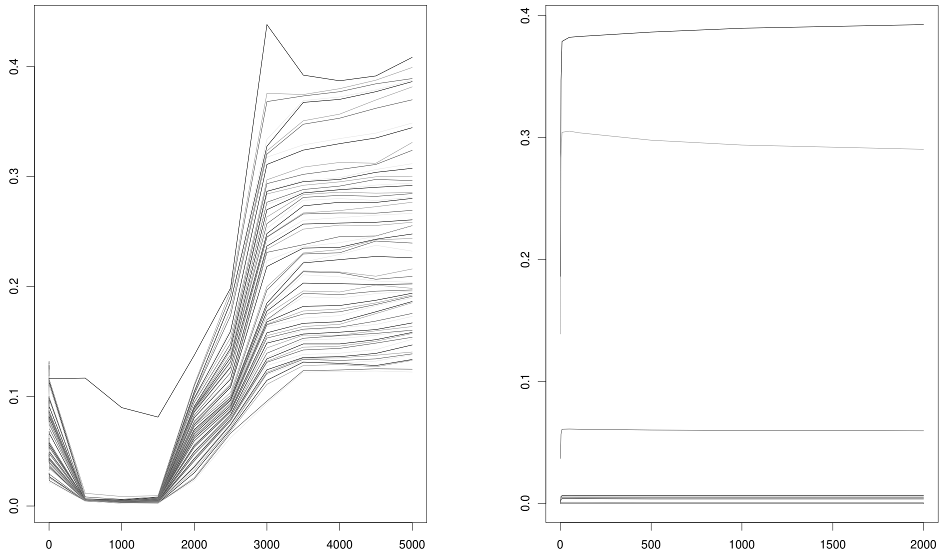
greatly—though, once again, not in terms of time efficiency. Indeed, figure 6.7(a) shows that in the case of $h = 64$, all the hidden units present in the MLP network are effective. This means that none of these hidden units duplicates each other's work but, rather, that they display different patterns of variations. An increase in the number of hidden units thus causes a change of learning strategy in the MLP network.

As mentioned earlier, these two experiments have been duplicated four times each in Appendix B. This time, again, all plots came out qualitatively equivalent except for one case in which the autoassociator with a capacity of $h = 16$ also used all the hidden units available effectively (Figure B.14 of Appendix B). This deviation from usual behavior will be discussed in the discussion section below as well as the other results obtained in the experiments of this section.

Discussion

Monitoring the variance and effectiveness of the two systems with 16 or 64 hidden units allowed me to make three observations:

1. Whether using 16 or 64 hidden units, the hidden units of the autoassociator display the same variance pattern: as soon as learning starts, the variance level of each hidden unit increases until a given early epoch (around Epoch 100), after which no variation occurs. However, no matter how many hidden



(a) MLP (b) Autoassociation

Figure 6.7: The principal components of the hidden units of the MLP network and the autoassociator as a function of the epoch number. The model size is $h = 64$.

units display some variance, only three of them are (generally) actually effective.[2]

2. Whether using 16 hidden or 64 hidden units, during the first 1500 epochs, the variances of the hidden units in the MLP network are all close to zero and tend to decrease even further. While all the hidden units are originally effective, only one of them remains effective between epochs 500 and 1500.

3. Past 1500 epochs in the MLP network, when using 16 hidden units, 8 of these units display a large amount of variance, whereas when using 64 hidden units, all of them do. In both cases, all the hidden units displaying some amount of variance are effective.

The results obtained from the variance and effectiveness experiments demonstrate that even though the MLP network and the autoassociator use different learning strategies, both systems are capable of generalizing. As a matter of fact, the autoassociator generalizes no matter what its capacity is. This is apparent from the fact that only three of the 16 or 64 hidden units used in the autoassociator are actually effective. MLP, on the other hand, generalizes when a capacity of 16 hidden units is used since only 8 of these 16 hidden units turn out to vary

[2]In one case displayed in Appendix B, all the autoassociator's hidden units are effective. This, however, is the result of the random initial conditions of the network.

and to vary efficiently. When 64 hidden units are used, however, MLP does not generalize at all since all 64 hidden units become effective.

In summary, these results suggest that no matter how large the capacity of the autoassociator used to perform a recognition (i.e., a many class multiclass classification) task, although it learns the domain using a (fast) data-driven strategy, it is always capable of generalizing from the training data. On the other hand, the MLP network which learns the domain using a hypothesis-driven strategy, generalizes from the data only if it has a small capacity. If its capacity is large, while its learning strategy is still hypothesis-driven, no generalization takes place. In both cases, the MLP learning process will be more time consuming than the autoassociator's, and in the second case, it should cause a lot of overfitting, and thus yield a large number of misclassifications.

Note that the results obtained in this section affect the field of neural networks beyond the discovery of the fact that the autoassociator is capable of generalization despite the fact that it uses a data-driven strategy. First, the observation that in one case (Plot B.14 of Appendix B), the autoassociator departed from its usual learning strategy—which consists of using effectively only a few hidden units—in order to use the learning strategy of the MLP network when trained with a large number of hidden units—which consists of using effectively all the hidden units available in the network—suggests that the difference between the two approaches is not hard set. This result, indeed, suggests that a simple change in initial conditions can (although rarely, since this occurred only once out of five trials) affect the generalization space search strategy. Such flexibility suggests that, in a different context, speeding up the learning process of a feedforward neural network may be possible. As a matter of fact, this question was considered by [Munro1991] who discovered that if the unstable attractors present in the hidden unit activation plane of MLP are turned into weak attractors, then MLP converges to a solution more quickly. One way of weakening these attractors—at least the strategy used by the autoassociator or by [Munro1991] in the MLP network—is to constrain the learning task by running the backpropagation procedure on a network containing many output units.[3] This result is interesting because it suggests that the flexibility of neural networks can be exploited in order to speed up the learning process. However, it also suggests that neural networks are very volatile and, thus, not always fully reliable.

Second, the observation that all the hidden units of the MLP network are effective when using 64 hidden units (and that such a phenomenon can also occur with autoassociators) is also interesting because it suggests that although, as [Moody1992] and [Weigend1994] showed theoretically and experimentally, the effective number of parameters or hidden units of a network can be smaller than its actual number—suggesting that MLP networks are self-regularized and do not overfit the training data too much—, this is not necessarily the case. Indeed, my

[3]This strategy is also reflected by the neural network community's common practice of designing binary learning networks of two output units rather than restricting their output layer—as was done in this dissertation—to the sufficient size of one unit.

results demonstrate that in the case of the MLP network with 64 hidden units (and for the 64-hidden unit autoassociator, given certain rare initial conditions), the effective number of hidden units is equal to the actual number of hidden units present in the network. In the autoassociator, on the other hand, some sort of self-regularization does take place systematically since, in most cases, whether using 16 or 64 hidden units, only a few hidden units are actually effective.

6.3.2 Classification in the Autoassociator

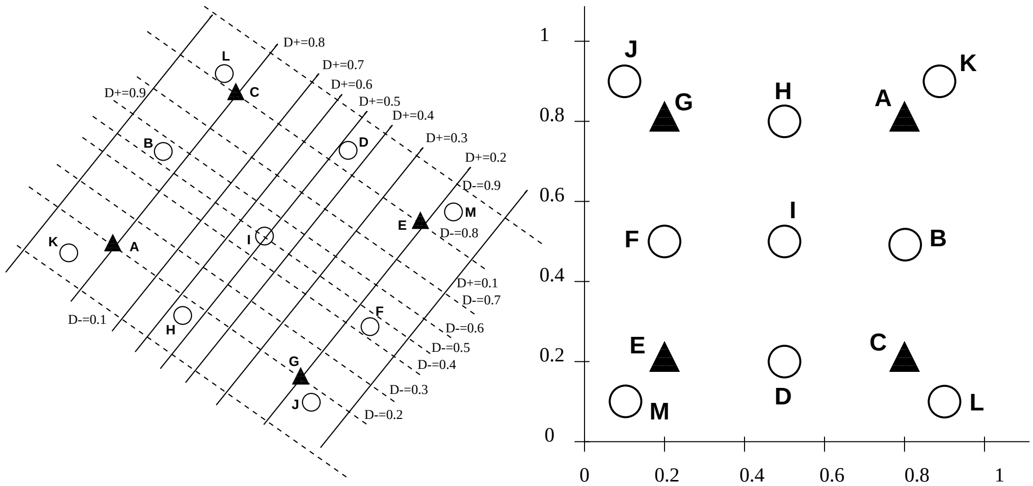
Having demonstrated that the autoassociator and MLP use different learning strategies and that, despite these differences, the autoassociator is capable of generalizing, I now turn to the question of whether the autoassociator generalizes beyond the training set to the counter-concept data or whether its generalization process is appropriately bound.

More specifically, I consider the artificial domain of Section 4.5 again, plot its HUAP at a fixed epoch (once the network has converged) and draw the boundaries associated with this HUAP. This methodology is intended to reveal the extent of the autoassociator's generalization and to explain a little bit more deeply its classification process. Although two hidden units are not sufficient for the autoassociator to solve this problem, because the partial solution it obtains is more revealing than the solution it would obtain on the simpler XOR problem used in the experiment of Section 6.2, the more complex domain of Section 4.5 was used for the experiments of this section.

Results of Munro's HUAP and Boundary Plotting Strategies

The plot of Figure 6.8(a) shows the HUAP resulting from training a 2-hidden unit NL-NL autoassociator on the artificial domain described in Section 4.5, together with its boundaries according to the description of Section 6.1.1. Note that, in addition to drawing the HUAP projection of the input patterns, I also drew the projection of the counter-examples through this network, and that, in order to simplify the HUAP, rather than considering the full training set, the autoassociator was trained on the four positive cluster means of the domain only, rather than on the four 50-point clusters. For comparison purpose, the input domain is reproduced in Figure 6.8(b).

Figure 6.8(a) displays the HUAP representation of the four concept means (black triangles) after 2000 epochs as well as those of the nine counter-concept means (white circles), although the counter-concept means were not used for training. As well, the two families of boundaries corresponding to the two output units of the network (one family for each output unit) are shown. One family corresponds to the positive-sloped lines (the family of full lines labeled D+ in figure 6.8(a)) while the other corresponds to the negative-slopped lines (the family of broken lines labeled D- in figure 6.8(a)). In both case, the slopes are drawn for $D+ = D- = 0.1, 0.2, .., 0.9$ (See formula in the Plotting Boundary Paragraph in



(a) Hidden Unit Space (b) Input Space

Figure 6.8: The HUAP and Boundaries of the Autoassociator on the artificial domain of Chapter 2.

Section 6.1.1 with the values of $D+$ and $D-$ corresponding to the value δ in the equation).

Figure 6.8(a) and (b) can be interpreted as follows: the points in the HUAP of figure 6.8(a) represent the reconstructed values of the points in the input domain of Figure 6.8(b). However, while the input coordinates can be read off the x- and y- axis of Figure 6.8(b), the output values represented by the points in Figure 6.8(a) must be read off the family of lines drawn on the figure. As mentioned before, one family corresponds to the reconstructed value obtained for the x-coordinate of the input points (in this case, the D+ family) while the other family corresponds to the reconstructed value obtained for the y-coordinate of the input points (in the case, the D- family). The reconstructed values for the x- and the y- coordinate of each point are equal to the D+ and D- values of the lines (belonging to the families corresponding to the respective coordinates) on which the point lies.[4]

As an example, consider the point represented by a black triangle and labeled "A" in Figure 6.8(a). The reconstructed value of this point is 0.8 for the x-coordinate and 0.2 for the y-coordinate since it lies at the intersection of lines $D+ = 0.8$ and $D- = 0.2$. Since, as shown in Figure 6.8(b), point "A" actually corresponds to point $(0.8, 0.2)$ in the input space of Figure 6.8(b), Figure 6.8(a) shows that point $(0.8, 0.2)$ is perfectly reconstructed by the autoassociator. For the point represented by a white circle and labeled "B", however—which corresponds to the reconstruction of point $(0.8, 0.5)$ in the input space of Figure 6.8(b)—, the autoassociator does not perform perfect reconstruction since

[4]Note that only the 9 lines corresponding to the D+ and D- values of $0.1, 0.2, \ldots$ 0.9 are shown in Figure 6.8(a). Nevertheless, each family contains an infinite number of lines since there is a line for every value between 0 and 1 (since each output unit can assume any value comprised between 0 and 1).

"B" lies away from lines D+=0.8 and D-=0.5. Generalizing to all the points in Figure 6.8 (a) and (b), it can be seen that all the points represented by a white circle (the counter-examples) lie away from their expected location, except for the central white circle which lies at the intersection of $D+ = D- = 0.5$ and is thus perfectly reconstructed. Conversely, all the black triangles (the examples of the concept) lie at the intersection of lines carrying $D+$ and $D-$ values of either 0.2 or 0.8 in Figure 6.8(a) and corresponding to their input values in Figure 6.8(b), signifying that they are all perfectly reconstructed by the autoassociator. These observations are discussed in the following section.

Discussion

Figure 6.8(a) and (b) are very useful as they clarify the classification process taking place in the autoassociator and show how over-generalization to the counter-concept data is avoided. As just mentioned, the HUAP representations of the four concept cluster means are located on the positively and negatively sloped lines corresponding to $D+ = D- = 0.2$ and $D+ = D- = 0.8$, simultaneously, and there is a correspondence between the reconstructed values of Figure 6.8(a) and the coordinates of the input space of Figure 6.8(b). This means that the output values of the four concept points ((0.2, 0.2), (0.2, 0.8), (0.8,0.2), (0.8, 0.8)) are all perfectly reconstructed, as expected from our classification scheme. Furthermore, while the HUAP of Figure 6.8(a) shows a great similarity with the input domain of Figure 6.8(b), there are some significant differences affecting the counter-concept examples: rather than being aligned with pairs of concept cluster means, the counter-concept cluster means are shifted away from these lines (except for the central mean which remains right in the center of the HUAP). This shift causes the means of all the counter-conceptual clusters (except for the central one) to be badly reconstructed, since they are not located at the intersections of the positively and negatively sloped lines corresponding to $D+ = D- = .1, D+ = D- = .2, D+ = D- = .5, D+ = D- = .8$, and $D+ = D- = .9$. In our classification scheme, however, since bad reconstruction of the counter-concept class is the key to classification, the shift experienced by the counter-examples is useful. On the contrary, it is the well-reconstructed central counter-concept mean (Point I) that is a problem since it represents a counter-concept example mistakenly believed to be an example of the concept. Nevertheless, this mistake can be explained by the fact that only two hidden units were used to solve the problem which, as discussed in Chapters 4 and 5, requires 16 hidden units to be optimally classified.

In addition to explaining the classification mechanism of the autoassociator, it is worth noting that Figure 6.8(a) shows that the generalization process of the autoassociator is bounded since it does not generally extend to the counter-examples. This bounding is caused by the distortion experienced by the counter-examples (which were not used at training time) during the projection of the input data onto the hidden layer, and this distortion does not affect the concept examples.

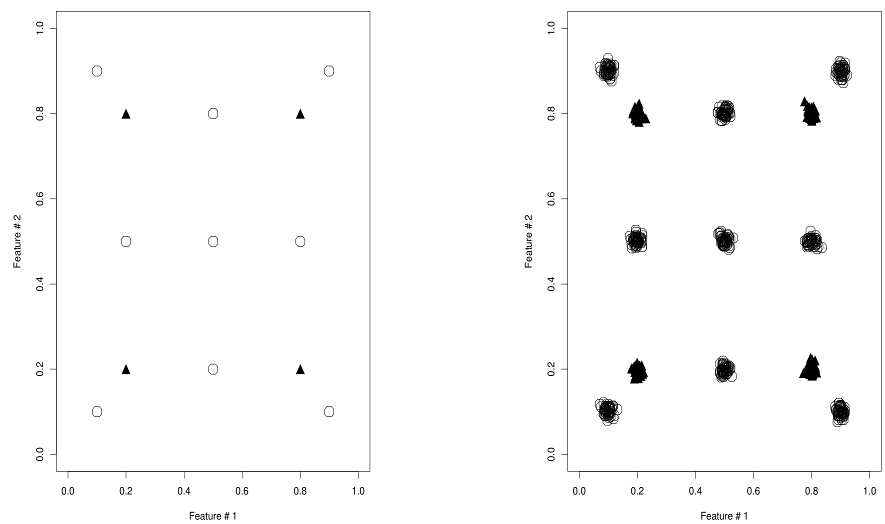
Altogether, this chapter thus demonstrated that the autoassociator can be more efficient than MLP on the domain of Section 4.5 because it uses a data-driven strategy that is faster than the hypothesis-driven strategy of the MLP network. In addition, it showed that despite this data-driven strategy, the autoassociator does perform some generalization and that the amount of generalization it performs is bounded enough to allow for classification to take place in the domain.

Chapter 7
Accuracy Considerations

The purpose of this chapter is to investigate the observation made in Chapter 4 about autoassociation-based classification being more accurate than MLP despite autoassociation-based classification using less information than MLP since its concept-learning phase ignores counter-examples and prior information about the task. This study is conducted within [Mitchell1982]'s generalization/specialization framework. In particular, I demonstrate experimentally that although the specialization bias used by the autoassociator (see Chapter 5) presents disadvantages in certain situations, it is sometimes more appropriate than the bias provided by counter-examples in several others. The results are subsequently used to explain the results obtained by the autoassociator and MLP on the real-world domains of Chapter 4.

In more detail, this chapter attempts to determine the accuracy shortcomings and advantages of autoassociation-based classification in contrast to MLP by comparing the classification accuracy of the two systems on a number of artificial domains derived from the artificial domain of Section 4.5. Because the principal difference between the autoassociator and MLP is their specialization bias—the MLP network uses counter-examples for specializing whereas the autoassociator does not—, this study focuses on testing how well the two systems adapt to various strains on this bias. More particularly, it investigates the effect of the two classifiers on domains which would benefit from a particularly strong specialization bias and on domains which would benefit from a particularly weak specialization bias. Such domain characteristics, actually, do occur in real-world settings as exemplified by the problem of discriminating between a dog and a wolf, where a classifier needs a particularly strong specialization bias since dogs and wolves have many features in common. Conversely, in the problem of discrimination between a dog and an elephant, although strong specialization might not have negative effects, weak specialization is sufficient. Nevertheless, there are cases in which strong specialization is harmful. Consider, for example, the following definition of a bird: "a bird is an animal that can fly". This definition is too specialized since it excludes the class of penguins which are birds even though they are not able to fly.

The chapter is divided into four sections. Section 7.1 describes the different domains on which the two systems are compared; Section 7.2 discusses the experimental methodology used for these comparisons; Section 7.3 displays the results obtained on these experiments; and Section 7.4 is a discussion of these results, including, in particular, their application to the real-world domains of Chapter 4



(a) Means Only (b) All Data

Figure 7.1: The Artificial Domain of Section 4.5.3. Examples of the concept are represented by black triangles while counter-examples are represented by white circles.

and their generalization to higher-dimensionality domains.

7.1 Domain Description

In order to test the flexibility of the specialization biases of the autoassociator and MLP network, I tested the two systems on a series of variations from the artificial domain of Section 4.5. The artificial domain of Section 4.5 is illustrated again in Figure 7.1, both as a display of the means of each cluster alone (Figure 7.1(a)) and as a display of the entire data set (Figure 7.1(b)).

For the experiments of this chapter, since the autoassociator and MLP networks yield classifications that are about equally accurate on the original domain (see Figure 7.8 of Section 7.3), my strategy consisted of declaring this domain "neutral with respect to the specialization bias it uses", and of generating new domains presenting a bias either stronger than that of the original domain or weaker. Following this domain generation strategy, evaluations of the performance of autoassociation-based classification and MLP were carried out relative to the results they obtained with respect to one another on the original domain, as will be discussed in Section 7.2. This section begins by overviewing the different domain modifications made to the original domain, and then describes each of them in greater detail.

7.1.1 Overview of the Domain Modifications

In order to create domains that require a strong specialization bias, I applied five different types of modifications to the original domain:

1. Concept Class Pattern Augmentations (CPA)
2. Concept Class Mean Proximity Increase (CMPI)
3. Concept Class Variance Increase (CVI)
4. Counter-Concept Class Variance Increase (CCVI)
5. Counter-Concept Class Density Decrease (CCDD)

The first three modifications affect the concept data while the last two affect the counter-concept data. The first three modifications can be thought of as void reduction methods in that they all—in one way or another—decrease the void occurring between the four concept clusters of the original domain. Decreasing this void calls for a stronger specialization effect since it creates a tendency for classifiers to integrate the various concept clusters in a single larger cluster, a step that would cause a large classification error to occur since internal counter-examples would be misclassified as concepts rather than counter-concepts (like when using PCA or a linear autoassociator). The last two modifications affect the counter-concept components. The purpose of the first one is to increase the amount of surface of each concept cluster surrounded by counter-concept data. This modification calls for greater specialization strength because the counter-concept data are not all located in a few localized spots, as in the original domain, but rather, they wrap around the concept clusters. The purpose of the second modification is to decrease the density of each counter-concept cluster, thereby decreasing the amount of external specialization provided to systems which take such information into consideration, and calling for increased internal specialization.

In order to create a domain that requires a weak specialization bias, I applied a single type of modification to the original domain:

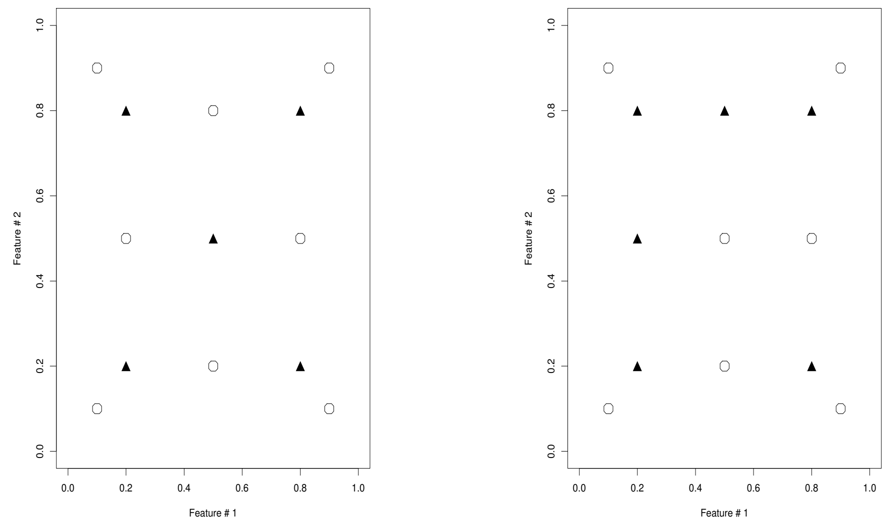
- Concept Density Decrease (CDD)

This modification was applied by decreasing the cardinality of one of the four concept clusters in order to test whether or not, in case of unbalance in the density of the concept clusters, the small cluster is correctly identified or misclassified as a noisy occurrence.

The next sections describe each of the domain modifications just introduced in more detail.

7.1.2 Concept-Class Pattern Augmentation

Concept-Class Pattern Augmentation was implemented in two ways, each creating a concave, but connected pattern. The first way consisted of turning the central



(a) CPA1 (b) CPA2

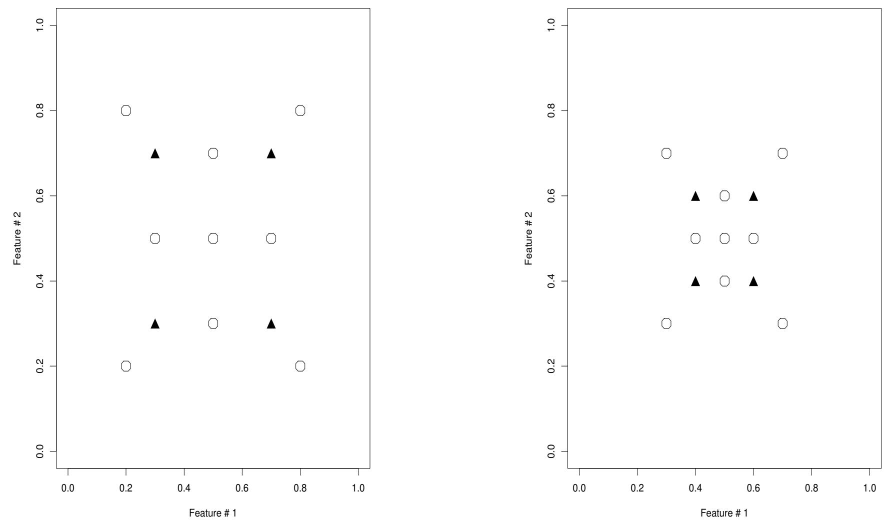
Figure 7.2: Concept-Class Pattern Augmentation (CPA). Examples of the concept are represented by black triangles while counter-examples are represented by white circles. In Figure (a), the central counter-concept cluster of the original domain (located at point $(.5, .5)$) was transformed into a concept cluster. In Figure (b), it is the counter-concept side components located at points $(.2, .5)$ and $(.5, .8)$ that underwent that transformation.

internal counter-concept cluster of figure 7.1(a) or (b) into a concept one; and the second,consisted of turning the side internal counter-concept clusters of two of its contiguous sides into concept clusters. The result of these modifications is displayed in Figure 7.2(a) and (b), displaying only the means of the clusters of the domain. For each cluster, 50 points are distributed around these means with variance $\sigma^2 = 0.01$.

The reason why these modifications call for greater specialization by classifiers is that they connect—at least some of—the subconcepts of the concept class, making the remaining counter-concept internal clusters more prone to misclassification as positive clusters than they were in the original domain.

7.1.3 Concept-Class Mean Proximity Increase

Concept-Class Mean Proximity Increase was implemented in two ways. First, moving the concept clusters to a distance of 0.4 units from each other; Second, by moving them to a distance of 0.2 units from each other (we recall that the original distance between each concept cluster was of 0.6 units). The counter-concept clusters are moved accordingly in order for the structural configuration of the original domain to be preserved. The result of these modifications is displayed in Figure 7.3(a) and (b), displaying only the means of the clusters of the domain. For each cluster, 50 points are distributed around these means with variance



(a) CMPI1 (b) CMPI2

Figure 7.3: Concept-Class Mean Proximity Increase (CMPI). Examples of the concept are represented by black triangles while counter-examples are represented by white circles. In figure (a), the original domain has been shrunk by one space, while it was shrunk by two spaces in figure (b).

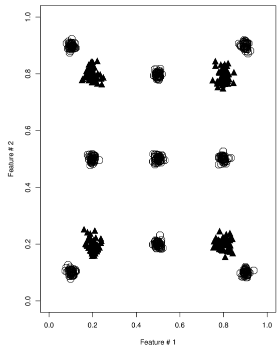
$\sigma^2 = 0.01$.

These modifications call for greater specialization by classifiers because they increase the within-class distance of the concept data making it more natural for a classifier to consider these data as belonging, again, to a single cluster than it was in the original domain.

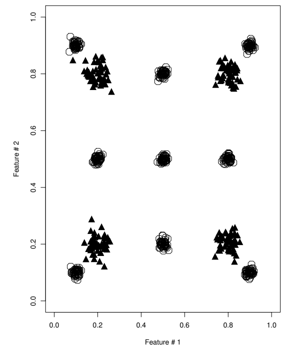
7.1.4 Concept-Class Variance Increase

Concept-Class Variance Increases were implemented in three ways: by increasing the variances of all the concept clusters to $\sigma^2 = 0.02, 0.03$ and 0.04, respectively (we recall that the variance was $\sigma^2 = 0.01$ in the original domain). These modifications are displayed in the graphs of figure 7.4(a), (b) and (c), which this time, display the domains in their entirety, rather than only the cluster means.

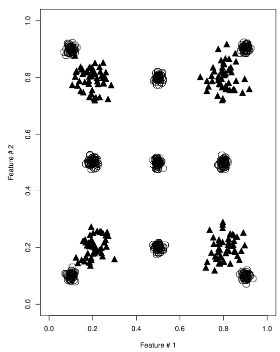
The reason why these modifications call for greater specialization by classifiers is similar to the reason stated in the case of Concept-Class Mean Proximity Increase. Namely, they increase the within-class distance of the concept data. However, in the present case, the increase in proximity of the concept cluster is implemented by variance increases rather than mean displacement.



(a) CVI1



(b) CVI2



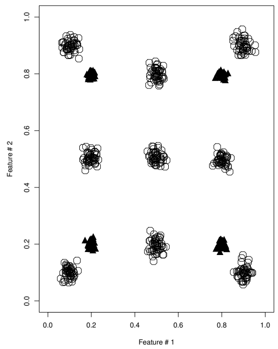
(c) CVI3

Figure 7.4: Concept-Class Variance Increase (CVI). Examples of the concept are represented by black triangles while counter-examples are represented by white circles. In figure (a), the variance of the concept clusters was increased by .01 with respect to the original artificial domain. It was increased by .02 and .03, respectively, in figures (b) and (c). Note, however, that although the variance of the concept clusters was increased, no between-class overlap occurs in any of the plots.

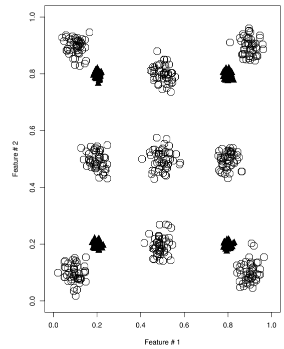
7.1.5 Counter-Concept-Class Variance Increase

Counter-Concept-class Variance Increase is the first of the two types of modifications geared at testing the strength of the specialization biases of the classifiers by means of modification of the counter-concept class. The next subsection discusses the second set of modifications of this type. Counter-Concept-class Variance Increase was implemented by increasing the variance of each counter concept cluster to 0.02, 0.03 and 0.04, respectively. These modifications are displayed in the graphs of figure 7.5(a), (b), and (c).

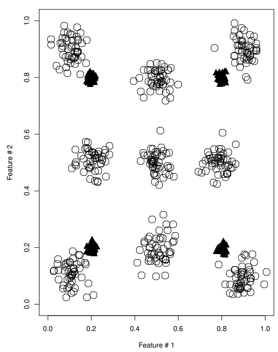
These modifications call for greater specialization by classifiers because they increase the amount of concept data directly exposed to counter-concept ones, thus increasing the amount of concept-specialization required. In other words, the concept data gets more and more surrounded from all parts in these domains whereas it was surrounded from only well-localized points in the original domain.



(a) CCVI1



(b) CCVI2



(c) CCVI3

Figure 7.5: Counter-Concept Class Variance Increase (CCVI). Examples of the concept are represented by black triangles while counter-examples are represented by white circles. In figure (a), the variance of the counter-concept clusters was increased by .01 with respect to the original domain. It was increased by .02 and .03, respectively in figures (b) and (c). Note that, again, although the variance of the concept clusters was increased, no between-class overlap occurs in any of the plots.

It thus requires more specialization.

7.1.6 Counter-Concept Class Density Decrease

The second counter-concept modification geared at increasing the demand for specialization was implemented by decreasing the size of the counter-concept clusters to 5 examples each (we recall that each cluster contained 50 data points in the original domain). This modification is displayed in Figure 7.6.

The reason why this modification calls for greater specialization by classifiers is because the reduction in the number of data by counter-concept clusters makes these clusters less statistically significant than they were in the original domain. Such a transformation can make certain classifiers mistake those data for noise and cause them to overgeneralize and label them as positive instances of the

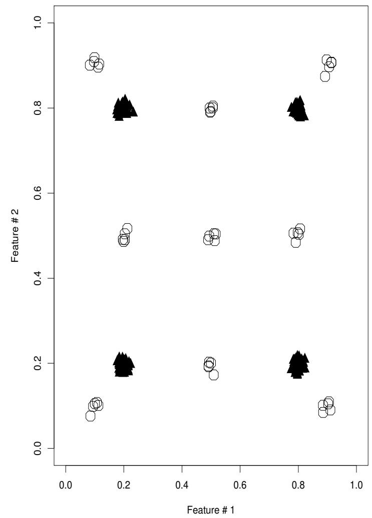


Figure 7.6: Counter-Concept Class Density Decrease (CCDD). Examples of the concept are represented by black triangles while counter-examples are represented by white circles. The cardinality of the counter-concept clusters was divided by 10 with respect to the original domain.

concept.

7.1.7 Concept-Class Density Decrease

Concept-Class Density Decrease is the only type of modification considered in this study whose aim is to create a domain requiring a weak specialization bias. This modification consisted of decreasing the cardinality of a single one of the concept clusters of the original domain from 50 to 5. This modification is illustrated in the graph of Figure 7.7 in which the size of the upper left concept cluster (centered at location (.2, ., 8)) is decreased.

The reason why this modification calls for a weakening of the specialization bias of classifiers is because, like for the previous modification, the decrease in cardinality yields a decrease in statistical significance of the concept cluster composed of 5 points. In order for a classifier to recognize the data of this cluster as positive instances of the concept, a weaker specialization bias is required of that classifier which, otherwise, would dismiss the cluster as a noisy patch in the input domain. Note, however, that this specialization weakening should be only local since the other components require the same amount of specialization as previously.

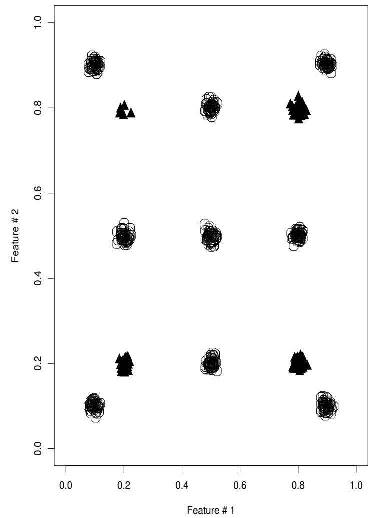


Figure 7.7: Concept-Class Density Decrease (CDD). Examples of the concept are represented by black triangles while counter-examples are represented by white circles. The cardinality of the concept cluster located at position (.2, .8) was divided by 10 with respect to the original domain.

7.2 Experimental Strategy

The parameter setting procedure used for the experiments of this chapter were carried out like those of section 4.5.3. Namely, the two systems were tested on each domain described in the previous section on networks of capacities 1, 2, 4, 8, 16, 32, and 64 hidden units and the two systems were compared using their optimal capacity (within this set range) on each of the domains considered in this chapter. The maximum number of epochs were set at 2000 epochs for the autoassociator and 5000 epochs for the MLP networks which are both more than sufficient (by about 1000 epochs, each) to solve the original artificial problem and, thus, should be sufficient for the closely related artificial domains considered in this chapter.[1] It was not necessary to stop the training process early since the artificial domains considered are noiseless, and thus, overfitting is minimal.

Like previously, for the autoassociator, the learning rate and momentum of the networks were fixed and set at 0.05 and 0.9, respectively. However, for MLP, while the momentum was set to 0.9, the learning rate was increased to 0.2 in order to speed up learning on the more difficult domains of this chapter. Every experiment was performed 5 times on different instantiations of the problems, and the

[1]If the autoassociator and MLP do, in fact, require longer training periods than the extended time period allocated to them, that would suggest that these systems are impractical for domains presenting the characteristics tested in this chapter. These classifiers would, therefore, not be worth considering, even if they are eventually able to solve the problems when given an infinite amount of training time.

graphs displayed in the next section record the average performance obtained by the autoassociator and the MLP networks. In these graphs, each system tested is represented by a ROC curve (See Section 3.3.3 of Chapter 3). The relative performance of these systems can be obtained by realizing that better performing systems have a larger area under their curves. For example, in Figure 7.8 which displays the results of the autoassociator and MLP network on the original artificial domain with 16 hidden units each, it can be seen that the MLP network is slightly more accurate than the autoassociator, since the curve representing the MLP results is located above the curve representing the autoassociator, when the two curves do not merge.

The optimal network capacity settings obtained for each system on each artificial domain considered are listed in table 7.1. The next section reports the results

Domain	AUTOASSOCIATOR	*MLP*
Regular	16	16
CPA-1	4	64
CPA-2	8	8
CMPI-1	16	16
CMPI-2	16	16
CVI-1	2	32
CVI-2	8	32
CVI-3	8	32
CCVI-1	32	64
CCVI-2	4	64
CCVI-3	64	16
CCDD	64	16
CDD	16	4

Table 7.1: Optimal Capacities for the autoassociator and MLP network on the different domains considered in this Chapter.

of the experiments comparing MLP to the autoassociator when using these parameters.

7.3 Results

The results obtained on the different domains of Section 7.1 are displayed in seven graphs. As discussed previously, the graph of figure 7.8 represent the results obtained on the original domain reproduced in Figure 7.1. The graphs of figures 7.9, 7.10 and 7.11 represent the results obtained on the domains that require a strong specialization bias due to modifications to the concept class. The graphs of figure 7.12 and 7.13 represent the results obtained on the domains that require a strong specialization bias due to modifications to the counter-concept class. The graph of Figure 7.14, finally, represents the result obtained on the domain that

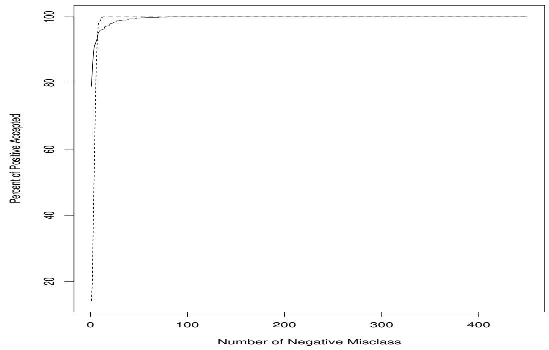


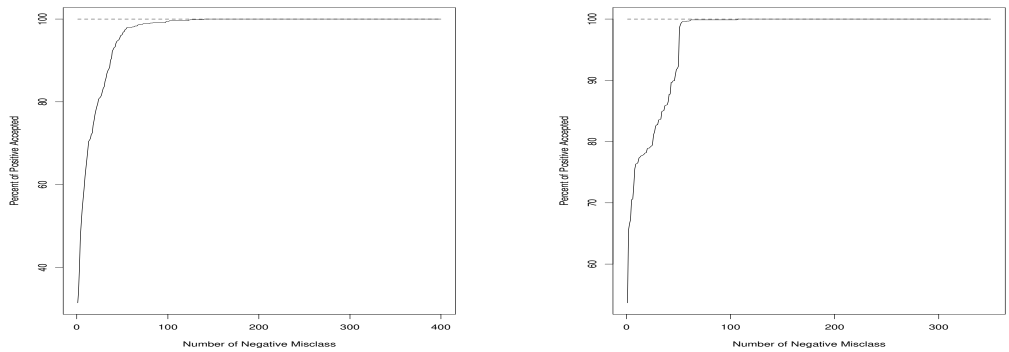
Figure 7.8: ROC Analysis of the autoassociator and the MLP network on the original artificial domain. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. The two systems are practically as accurate with MLP having a slight advantage over the autoassociator.

requires a weak specialization bias due to a modification to the concept class. In all the graphs, the accuracy performance of the autoassociator is represented by a full line while that of MLP is represented by a broken line. The graph of Figure 7.8 was already discussed in the previous section.

The graphs of figure 7.9 indicate that using an autoassociator is at a disadvantage with respect to using MLP since MLP can classify perfectly domains CPA-1 and CPA-2 (actually, even better than the original domain), whereas the autoassociator's performance deteriorates (with respect to its performance on the original domain). This suggests that the specialization bias of the autoassociator is too weak to deal with the strong specialization bias required to disconnect the concept-class subcomponents of the CPA-1 and CPA-2 domains. MLP, on the contrary, can learn how to disconnect these subconcepts using the counter-concept class.

The graphs of figure 7.10 show that, while the autoassociator gets affected by an increase in proximity of the concept clusters, MLP, once again, does not. On the contrary MLP is capable of classifying the two domains perfectly. This suggests that the specialization bias of the autoassociator is, again, too weak for dealing with the CMPI-1 and, more noticeably, the CMPI-2 domain. MLP, on the contrary, has no problem dealing with these domains since the counter-concept class creates a strong enough specialization bias to deal with this sort of domain characteristics.

The graphs of figure 7.11 show that although both the autoassociator and the MLP network are negatively affected by an increase in the variance of the concept class' clusters, MLP is less affected than the autoassociator. This suggests, once more, that the specialization bias of the autoassociator is too weak to handle increases in variance from the concept class. MLP, however, while somewhat weakened, remains adequate. In summary, the graphs of Figures 7.9, 7.10 and



(a) CPA-1 (b) CPA-2

Figure 7.9: ROC Analysis for domains requiring a strong specialization bias due to concept pattern augmentation. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. MLP is more accurate than the autoassociator on the two domains.

7.11 all suggest that domains that require a particularly strong specialization bias coming from the concept class are more accurately classified by MLP than by the autoassociator.

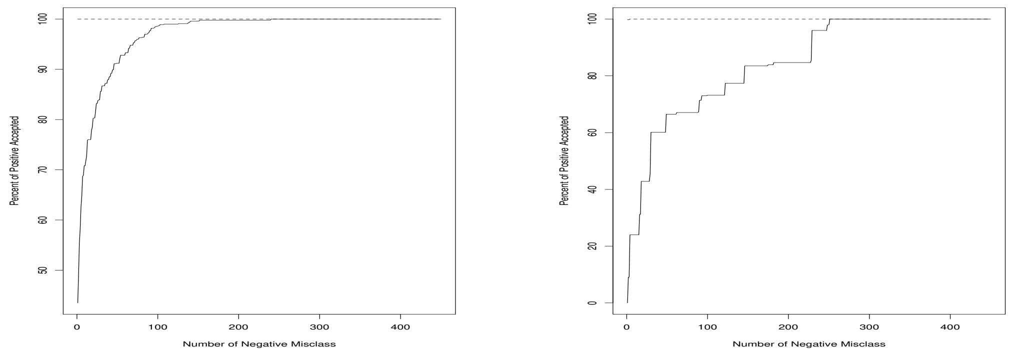
The graphs of Figure 7.12 show that while MLP is affected by an increase of variance in the counter-concept data, the autoassociator is not. This is not surprising in view of the fact that the autoassociator ignores the counter-concept data during training.

The graph of figure 7.13, similarly, shows the sensitivity of the MLP network to the statistical significance of the counter-concept clusters. In particular, it shows the strong detrimental effect of an unbalance in the cardinality of the concept and counter-concept clusters in the case of MLP. Again, the autoassociator is not affected since it ignores the counter-concept data altogether. Thus, the graphs of Figures 7.12 and 7.13 suggest that while MLP is negatively affected by domains requiring a strong specialization bias coming from the counter-concept class, the autoassociator's performance is not jeopardized.

Finally, the graph of figure 7.14 shows that the autoassociator is not affected by a decrease in concept density. Conversely, MLP is greatly affected by such a decrease. This can be explained by the fact that the autoassociator's operation is more local than MLP's (see the results of Chapter 6), and therefore, does not ignore statistically less significant concept data. Thus, the graph of Figure 7.14 suggests that domains with particularly weak specialization needs coming from the concept class are better classified by the autoassociator than by MLP.

7.4 Discussion

In summary, the results of the previous section suggest that for the classification task:



(a) CMPI-1 (b) CMPI-2

Figure 7.10: ROC Analysis for domains requiring a strong specialization bias due to concept-class mean proximity increase. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. While MLP is more accurate than the autoassociator on the CMPI-1 domain, the autoassociator is more accurate than MLP on the CMPI-2 domain.

- MLP is better suited than the autoassociator to domains which require a *strong* specialization coming from the *concept* class. This can be explained by the fact that the MLP network has the possibility of relying on counter-examples during the inductive process, whereas the autoassociator does not.

- The autoassociator is better suited than MLP to domains which require a *strong* specialization coming from the *counter-concept* class. This can be explained, conversely, by the fact that the autoassociator's strategy is independent of the counter-concept class, whereas MLP take into consideration both examples and counter-examples of the concept during the learning process.

- The autoassociator is better suited than MLP to domains which require a *weak* specialization coming from the *concept* class. This, again, can be explained by the fact that the autoassociator focuses on characterizing the concept class independently of the counter-concept class, whereas MLP, which considers both classes, may overlook a particularly sparse subconcept, especially if the counter-concept class is dense.

These results are interesting and useful for two reasons. First, they match certain domain characteristics to two different learning schemes making it possible for a user to select the more appropriate scheme for his/her data. Second, they provide detailed insight regarding the specialization bias of MLP and autoassociators. Such knowledge can help direct applied research aimed at improving these systems, and more theoretical research geared at explaining what the backpropagation procedure computes. In the remainder of this chapter, I focus on the practical aspect of these results. Their theoretical consequences are left for future work.

More precisely, regarding the practicality of the results just obtained, I propose to conduct two related studies geared at establishing their relevance. The first study is experimental and applied. The second study is also experimental, but more formal. In more detail, the first study consists of applying the results obtained in the previous section to the three real-world domains used in the preliminary experiments reported in Chapter 4, in order to explain their results. The second study, on the other hand, returns to the observation of artificial domains and discusses the type of characteristics necessary for a third dimension to possess in order 1) for the results obtained on two-dimensional domains *to hold* and 2) for the results obtained on two-dimensional domains *not to hold*. The purpose of this study is to demonstrate by mathematical-like induction that, as already suggested by the results of Chapter 4, the observations made in Section 7.3 are relevant for high dimensional domains provided that certain conditions apply. The next two sections describe these two studies and their results in detail.

7.4.1 Evaluating the Usefulness of the Accuracy Results by Application to Real-World Domains

The following study is an illustration of how the results obtained in the previous section can be used to explain (and thus predict) the results obtained on the real-world domains of Chapter 4. In the first step of this study, I have gathered statistics regarding domain characteristics of the projections of the three real-world domains of Chapter 4 on their two principal components (Figure 4.3 of Chapter 4). These statistics are:

- $PosNumber$ and $NegNumber$, the number of clusters (of size greater or equal to 3) of each class, respectively.
- $PosCompSize_x$, the size of $PositiveCluster_x$ ($x = 1..PosNumber$) and $NegCompSize_x$, the size of $NegativeCluster_x$ ($x = 1..NegNumber$). Here, "size" refers to the number of data points in each component.

From these statistics, the following domain characteristics were computed that match the different artificial domain variations studied in this chapter.

- $PosAvgVar$, the average variance of the various concept clusters
- $NegAvgVar$, the average variance of the various counter-concept clusters
- $AvgMeanDist$, the average mean distance between the means of the concept and counter-concept clusters
- $AvgDist$, the average actual distance between the concept and counter-concept clusters[2]

[2]The previous characteristic, $AvgMeanDist$, focuses on the distance between the means of the various clusters. This characteristic subtracts the average standard deviations of the concept and counter-concept components from the results of the previous question.

- $PosAvgSampleSize$, the average sample size of the concept clusters
- $NegAvgSampleSize$, the average sample size of the counter-concept clusters
- $PosAvgVarSize$, the variance in the sample size of the concept clusters
- $NegAvgVarSize$, the variance in the sample size of the counter-concept clusters

The formulas for the domain characteristics follow (with $Var(PosCluster_k)$ and $Var(NegCluster_l)$ corresponding to the variance of positive and negative cluster k and l, respectively):

- $PosAvgVar = \frac{\Sigma_{i=1}^{PosNumber} Var(PosCluster_i)}{PosNumber}$
- $NegAvgVar = \frac{\Sigma_{i=1}^{NegNumber} Var(NegCluster_i)}{NegNumber}$
- $AvgMeanDist =$ $\frac{\Sigma_{i=1}^{PosNumber} \Sigma_{j=1}^{NegNumber} distance(mean(PosCluster_i), mean(NegCluster_j))}{PosNumber * NegNumber}$
- $AvgDist = AvgMeanDist - sqrt(PosAvgVar) - sqrt(NegAvgVar)$
- $PosAvgSampleSize = \frac{\Sigma_{i=1}^{PosNumber} PosCompSize_i}{PosNumber}$
- $NegAvgSampleSize = \frac{\Sigma_{i=1}^{NegNumber} NegCompSize_i}{NegNumber}$
- $PosAvgVarSize = Var(PosCompSize_x)$, $x = 1..PosNumber$
- $NegAvgVarSize = Var(NegCompSize_x)$, $x = 1..NegNumber$

The values obtained for each of these characteristics on the three real-world domains of Chapter 4 are displayed in Table 7.2. The last two lines constitute a reminder of the percent classification error obtained by the autoassociator and MLP network on the three domains (these results first appeared in Table 4.1 of Chapter 4).

The observations that were made regarding the artificial domains of this chapter can be used to explain the results obtained on two of the three real-world domains of Chapter 4: the helicopter and sonar domains. The third domain, DNA promoter, cannot be explained. This, however, is not surprising given the fact that the domain characteristics just gathered depend on the projections of the two principal components of the real-world domains and that this projection accounts for only 4.22% of all the DNA promoter domain information whereas it accounts for 39.4% and 23.4% for the helicopter and sonar domains, respectively

Characteristics	Helicopter	Sonar	DNA Promoter
$PosAvgVar$	0.0151	0.027	0.476
$NegAvgVar$	0.21	0.0718	0.194
$AvgMeanDist$	1.149	1.217	2.085
$AvgDist$	0.568	0.785	0.955
$PosAvgSampleSize$	6.5	7.17	6.2
$NegAvgSampleSize$	20.5	7.2	5.8
$PosAvgVarSize$	12.5	18.97	5.7
$NegAvgVarSize$	180.5	7.7	1.7
Accuracy of Autoass.	3.125%	20%	20%
Accuracy of MLP	10.9%	32%	20%

Table 7.2: Domain Characteristics of the three real-world domains of Chapter 4.

(see footnote 6 of Chapter 4). The statistics gathered for the DNA promoter domain are thus not significant and cannot be used to explain the performance of the classifiers.

For the helicopter domain, however, these statistics are significant and the better performance of the autoassociator as compared to the MLP network can be explained by:

1. The large negative variance of the domain (See the value of $NegAvgVar$ for the helicopter domain and the results of the CCVI 1, 2, 3 experiments);

2. The large difference in average sample size of the concept and counter-concept clusters (See the values of $PosAvgSampleSize$ versus $NegAvgSampleSize$ for the helicopter domain and the results of the CCDD experiment in view of the fact that positive and negative data play symmetric roles and are interchangeable in the MLP network, whereas the autoassociator ignores the counter-concept data); and,

3. The large variance in the negative sample size (See the values of $PosAvgVarSize$ versus $NegAvgVarSize$ for the helicopter domain and the results of the CDD experiment, in view, again, of the symmetric role played by the two classes for MLP).

The statistics are significant for the Sonar domain as well and the results demonstrating that, again, the autoassociator is more accurate than MLP can be explained by:

1. The variance of the positive sample size (See the value of $PosAvgVarSize$ for the Sonar domain and the result of the CDD experiment).

In summary, Table 7.2 suggests that single-sided learning by autoassociation can be more accurate than dual-sided learning by MLP when, either, the positive class is tightly organized relative to the negative class, or when it contains elements that could easily be confused for negative ones.

Nevertheless, these a-posteriori and generalized conclusions are purely suggestive and require a substantial amount of further experiments prior to being fully validated. However, they are interesting for they suggest some practical aspects of the results of this chapter. Namely, when given a new domain to classify, a researcher could, perhaps, predict which of the autoassociator and MLP approaches to classification may be better to use on that domain by:

1. Projecting the domain onto its first and second principal components

2. Computing a few statistics from this projection ($PosNumber$, $NegNumber$, $PosCompSize_x$, $NegCompSize_x$, $Var(PosCluster_k)$, $Var(PosCluster_l)$, for all positive clusters k and negative clusters l)

3. Deriving the domain characteristics from these statistics

4. Matching these characteristics to the results of this chapter to decide which approach to classification may be more appropriate

Note, however, that if the amount of domain information summarized in the first two principal components is too low, such as in the DNA promoter case, this strategy will not be useful. If, however, the amount of information is significant, then, this strategy is likely to have a predictive value and this predictive value can actually be strengthened by considering more than just the first two principal components (i.e., if considering 3 principal components, the user can gather statistics from the plots of the projections of the first and second principal components, first and third principal components, and second and third principal components of the domain).

Once again, this is only a preliminary strategy and, in order to make this strategy more practical, it would be useful to quantify the study of this section. This can be done by turning the qualitative values (such as "large" and "small") of each variable used in the rules constructed to explain the results obtained on real-world domains into actual numeric values. Furthermore, a study should be performed to determine which rule takes precedence over the others in case where several rules with different conclusions were to apply to a given situation. Such an endeavor, however, is very extensive and is left for future work.

7.4.2 Establishing the Usefulness of the Accuracy Results by Mathematical-Like Induction

The purpose of the study reported in this section is to generalize the results of Section 7.3 to higher dimensionality domains. Indeed, if the observations made in Section 7.3 were only to apply to two-dimensional domains, then they would be of little practical use since real-world application domains are typically high-dimensional. The results obtained in section 7.4.1, by projecting the high-dimensional real-world domains of Chapter 4 onto their first two principal components are encouraging since they suggest that the conclusions of Section 7.3 can indeed apply directly to higher dimensional domains. However, they are not sufficient to suggest that they would always apply nor do they inform us as to when they do and when they don't. As a matter of fact, Section 7.4.1 discusses the fact that while the results of Section 7.3 do apply to the helicopter gearbox monitoring and sonar target detection domains, they do not apply to the DNA promoter recognition domain and the reason why they don't is the trivial fact that the information available to the first two principal components is not particularly representative of the domain. In this section, I discuss more significant ways of characterizing when these results will or will not apply.

The section is divided into two subsections. The first subsection describes the experimental methodology and setting up used in this study. The second section describes its results.

Experimental Methodology

Describing when the results of Section 7.3 will apply to high-dimensional domains and when they will not can obviously not be done enumeratively for every possible domain dimensionality. Therefore, I propose to use a mathematical-like method of induction whose goal is to describe 1) the characteristics of three-dimensional domains for which the performance of the autoassociator holds up with respect to that of MLP; and 2) the characteristics of three-dimensional domains for which its performance with respect to MLP does not hold up. The assumption is that the mechanisms at play in going from a two-dimensional to a three-dimensional domain are the same as those at play in going from an n-dimensional to an $n+1$-dimensional domain for any n, and that, therefore, it is sufficient to extract the mechanisms occuring at $n = 2$ in order to transfer them to any n.

As well, there is no need to run the experiments on every domain considered in Section 4.3. Instead, it is sufficient to extract the properties that make a third dimension "visible" or "invisible" from a single domain. The extra dimension will affect any two-dimensional domain in the same way.

The experiments conducted for this study, thus, consist of running the autoassociator and MLP on three-dimensional domains consisting of the original two-dimensional artificial domain of Chapter 4 (Figure 4.4) plus one extra bit that can assume different distributions. Different distributions were used for that extra bit so as to test the autoassociator's performance in different types of situations that can arise when introducing a third dimension to a two-dimensional domain. A description of the different distributions tested is listed below. In one case (experiment # 2), two experiments using a couple rather than a single additional bit(s) were added to the three-dimensional experiment in order to test that case more fully.

Experiment # 1: Extra bit Distribution = Class Distribution

Experiment # 2: Extra bit Distribution = Same Distribution as First bit Distribution

Experiment # 2': Extra *two* bit Distributions = Same Distributions as First and Second bit Distributions (4-D domain)

Experiment # 2": Extra *two* bit Distributions = Same Distribution ($\times 2$) as First bit Distribution (4-D domain)

Experiment # 3: Extra bit Distribution = Constant Distribution

Experiment # 4: Extra bit Distribution = Uniform Distribution over $[0, 1]$.

The first two sets of experiments test useful though redundant third and fourth bit distributions since although these distributions have discriminatory power, they repeat parts of or all the information already present in the first two bits of the original domain. The last two experiments test useless third bit distributions since these distributions do not contain any discriminatory information.

In more detail, in the first experiment, the extra bit is normally distributed (with a small variance of 0.01) around 1, for data points in the concept class, and around 0, for data points in the counter-concept class. Although the information contained in the third bit is redundant with the information contained in the first two (since class information can be deduced from the first two bits), it is of a different nature since the class information gets directly stated rather than needing to be deduced. Learning in this case is therefore very basic since it consists of merely reporting the information contained in the third bit without much concern for the information contained in the first two. The purpose of this experiment is to demonstrate that moving on to a higher dimension does not necessarily hinder the classification process. This case, however, is trivial and, thus, other experiments demonstrating that an increase in dimensionality can be of little harm have been conducted.

In the second experiment the value assumed by the extra bit was chosen to correspond to the value assumed by the first bit of the original domain. This does not add any explicit class information, like in Experiment # 1, but it may add implicit (and rather erroneous) information suggesting that the distribution of the first bit is, in some way, more significant than that of the second. To explore this situation more deeply, two additional experiments, this time, on 4-dimensional data sets were conducted. The first one (Experiment # 2') tests the autoassociator on a domain giving equal weight to the distributions of the first and the second original bits (i.e., it copies the distribution of the first bit onto the third, and the distribution of the second bit onto the fourth). The second one (Experiment # 2") tests the system on a domain emphasizing the distribution of the first bit only (it copies the distribution of the first bit onto the third and onto the fourth bit). The purpose of these experiments is to find out whether an imbalance in the weight given to the distribution of one of the two original domain dimensions over the other is a factor in the performance of the autoassociator versus that of MLP.

In the third experiment, back in three dimensions, a constant value of 0.5 is assigned to the extra bit of the domain no matter what class the example belongs to. This experiment is intended to test the performance of the autoassociator with respect to MLP under one possible type of useless information (i.e., information with no discriminatory power).

In the fourth experiment, a uniformly random value comprised between 0 and 1 is assigned to the extra bit of the domain, again, with no regard for the class of the data points. This experiment is intended to test the performance of the autoassociator with respect to MLP under yet another type of useless information.

For all these experiments, as previously, the autoassociator was trained for 2000 epochs with 1, 2, 4, 8, 16, 32, and 64 hidden units, and the optimal result it obtained are the ones reported in the next section. The threshold-determination method used to test the system is the parametric recognition method already used in Chapter 5. The learning rate and momentum were respectively set to the standard values of 0.05 and 0.9. The MLP network was trained for 5000 epochs with 1, 2, 4, 8, 16, 32 and 64 hidden units. Again, the results reported in the next section were obtained with the capacity established to be optimal by the parameter setting experiments. The momentum was set to the standard value of 0.9, but the learning rate was increased to 0.2 in order to increase the network's convergence speed (as the dimensionality of the test domain increases, the speed of convergence of the MLP network may decrease unless the learning rate is increased). Finally, as before, the results presented in the next section are the average results obtained after 5 runs of each system on each test domain.

Results

In all the experiments just described, MLP's classification record is similar to the record it obtained on the original two-dimensional domain. In other words, MLP's classification accuracy is close to perfect in all 3-D and 4-D tested variations of the original domain. This observation indicates that MLP is not sensitive to an increase of dimensionality, as far as accuracy is concerned, as long as the added information is either useful, though redundant, or useless.

In the case of the autoassociator, the results are more varied and worth studying in more detail. In particular, figure 7.15 presents the classification errors obtained by the autoassociator with optimal capacity on each of the domains considered in this study. The optimal capacities established are, themselves, listed in Table 7.3.

The results obtained in Figure 7.15 indicate that if the added dimension is fully informative in that it represents the class information directly (Experiment # 1), then this added dimension is helpful. While it is obvious that such would be the case for MLP which is a supervised classifier, in the case of the autoassociator, an unsupervised classifier, there was, a priori, no reason why the system would focus on the most informative bit (as far as discrimination is concerned), since the system is not aware of the importance of that bit. It is thus interesting to note the autoassociator's classification accuracy improvement on this problem with respect to the original problem.

Experiment	AUTOASSOCIATOR
Regular	16
1 (3-D)	1
2 (3-D)	2
2' (4-D, balanced)	16
2" (4-D, unbalanced)	2
3	16
4	4

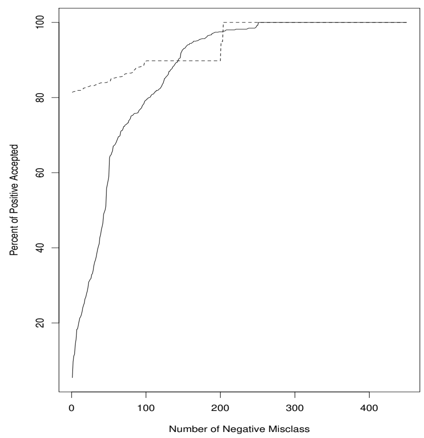
Table 7.3: Optimal Capacities for the autoassociator on the different domains considered in this Chapter.

The results obtained in Experiment # 2 show that the classification accuracy of the autoassociator on the 3-D domain obtained by copying bit 1 of the original domain onto bit 3 is slightly lower than that obtained on the original two-dimensional domain. This is partly caused by the introduction of an imbalance in the importance given to one of the original dimension over the other (as confirmed by the improvement of the results in Experiment # 2' which, in addition, copies bit 2 of the original domain onto bit 4, and their worsening in Experiment # 2" which copies bit 1 of the original domain onto bit 4); and partly caused by the fact that the added dimension(s) cause the reconstruction error of the concept class (which is close to zero) to increase proportionally more than that of the counter-concept class, thus creating a decrease in the difference between the two classes. Since this difference represents the criterion used for classification by the autoassociator, the observed decrease in classification accuracy could have been expected.

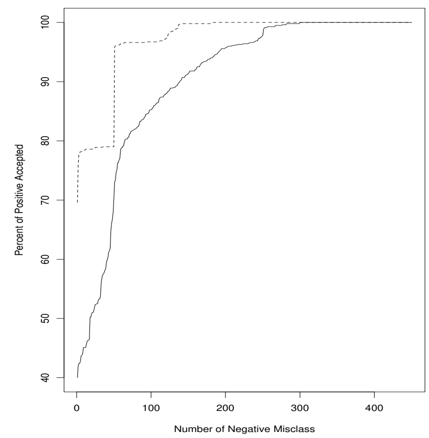
The results obtained on Experiment # 3, in which bit 3 is assigned a constant value, also indicate a decrease (slightly worse than in Experiments # 2 and 2') in the classification accuracy of the autoassociator with respect to the results it obtained on the original two-dimensional domain. This suggests that useless information is not invisible to the autoassociator (the way it is for MLP) and costs the system a loss in classification accuracy. This loss can be explained by the fact that, as far as the autoassociator is concerned, the domain of experiment # 3 is similar to the domain of experiment # 1 in which bit 3 represents class information. This suggests that, like in Experiment # 1, the autoassociator ascribed some importance to bit 3 assuming that its constant value is representative of the concept class. When it turned out, however, that bit 3's constant value is equally representative of the counter-concept class, classification accuracy necessarily decreased.

Finally, the results obtained for Experiment # 4, in which bit 3 is uniformly distributed over a large range, indicate a large decrease in accuracy of the autoassociator. This suggests that useless and highly varied, information is even more harmful for the autoassociator than useless and constant information. This can be explained by the fact that when the autoassociator cannot characterize the training data simply enough, it tends to revert to PCA's mode of action and simply copy the training data at the output layer. Note that although the autoassociator did largely emulate PCA in Experiment # 4, it did not completely do so since it retained enough discriminatory power to differentiate between the external negative data and all the other points.

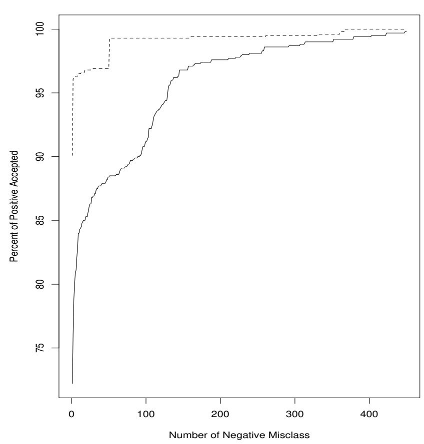
In summary, increasing the dimensionality of the domain will always somewhat hinder the performance of the autoassociator (unless it introduces explicit class information), but its effects will be small if either the information has discriminatory power though, is redundant (furthermore, an imbalance in redundancy creates a greater accuracy decrease than a balance in redundancy does) or if the information has no discriminatory power but is identical or close to identical for all data points. If, however, the addition of an extra dimension has no discriminatory power and its variance is high, then its effects will be strongly negative. MLP, on the other hand, remains unperturbed, except for a decrease in efficiency mitigated by increasing the learning rate of the system.



(a) CVI-1

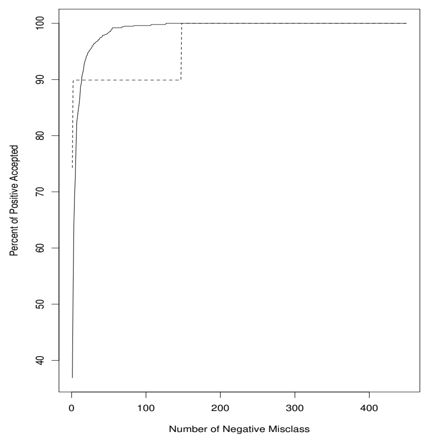


(b) CVI-2

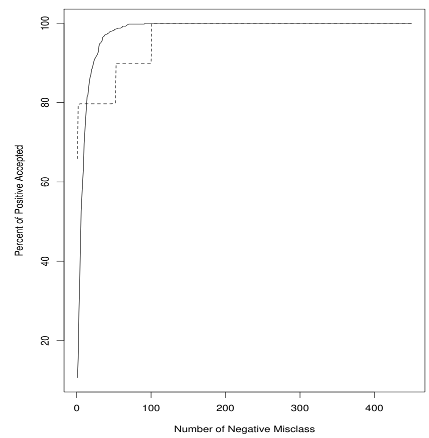


(c) CVI-3

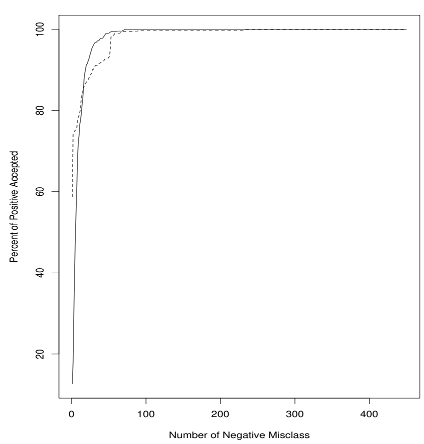
Figure 7.11: ROC Analysis for domains requiring a strong specialization bias due to concept-class variance increase. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. The autoassociator outperforms MLP on all three domains and the gap between the two systems increases as the variance of the concept class increases.



(a) CCVI-1



(b) CCVI-2



(c) CCVI-3

Figure 7.12: ROC Analysis for domains requiring a strong specialization bias due to counter-concept variance increase. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. The autoassociator outperforms MLP on all three domains. The autoassociator's performance remains the same as the variance of the counter-concept class increases, but MLP is more affected when this variance is small (CCVI-1).

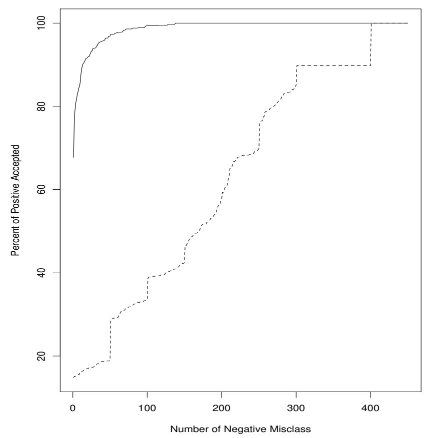


Figure 7.13: ROC Analysis for the domain requiring a strong specialization bias due to counter-concept density decrease. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. The autoassociator outperforms MLP significantly on this domain.

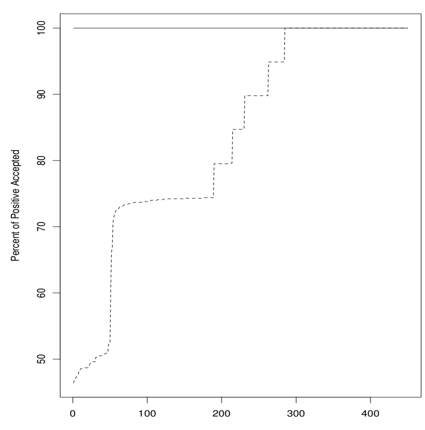


Figure 7.14: ROC Analysis for the domain requiring a weak specialization bias due to concept-class density decrease. The results of the autoassociator are represented by a full curve while those of the MLP network are represented by a broken one. The autoassociator outperforms MLP significantly on this domain.

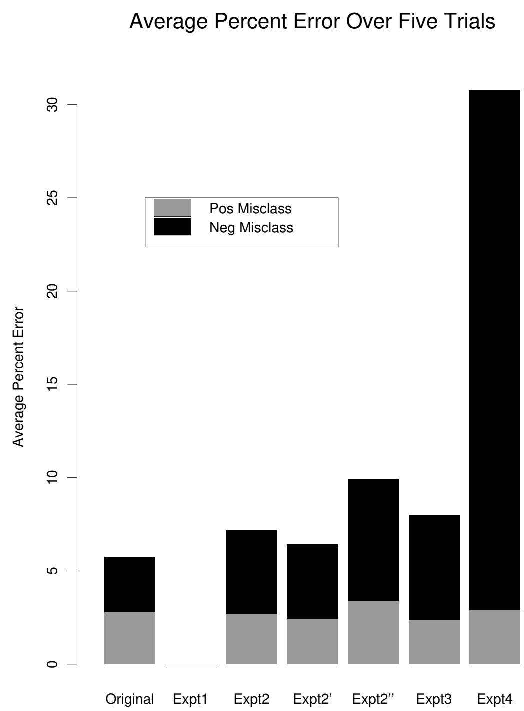


Figure 7.15: The classification error results obtained by Experiments 1, 2 (including 2' and 2" in 4-D), 3 and 4 compared to those obtained on the original two-dimensional artificial domain.

Chapter 8
Conclusions

This dissertation concludes by summarizing its main findings, overviewing the future work that would be interesting to pursue based on this research, and discussing the implications of the research.

8.1 Summary

This dissertation emerged from the observation by [Hanson and Kegl1987] that certain aspects of natural language could be acquired using an autoassociator—a feedforward connectionist network that learns how to reproduce its input at the output layer by way of a hidden layer—on grammatically correct instances of the language. The dissertation begins with a preliminary study which demonstrates that this autoassociative capability extends to domains other than natural language, and in particular, practical engineering domains. The primary interest of such a result is that autoassociators learn concepts using only positive instances of that concept, rather than both positive and negative instances of it. This can be an advantage in domains for which many instances of the concept are available but only a few counter examples can be found. In addition, unlike other paradigms which ignore the counter-examples, the autoassociator has the advantage of making use of a minimum of prior information.

The results of my preliminary study do not simply reveal that autoassociators are capable of learning how to classify practical engineering domains, but also, that they can, in certain cases, outperform classifiers that use positive and negative instances of the concept, both in terms of time efficiency and classification accuracy. Altogether, these preliminary results thus suggest three related questions which were used as a basis for my research and whose study constitutes the core of this dissertation. The purpose of these questions is to explain the preliminary results with the aim of learning how to use autoassociation-based classification effectively, and how to optimize its performance. These questions are:

- What characteristics of the autoassociator make it capable of performing classification tasks?
- What characteristic of the autoassociator make it more time efficient than MLP on certain domains?

- Which domain characteristics are more favorable—as far as accuracy is concerned—to recognition-based learning using an autoassociator than to discrimination-based learning using MLP; and which ones are more favorable to discrimination-based learning using MLP than to recognition-based learning using an autoassociator?

In the remainder of the dissertation, these questions are partially answered within [Mitchell1982]'s framework for binary induction which consists of decomposing the learning task into two opposing processes in search of an equilibrium. The first process seeks to *generalize* from the examples of the concept given to the classifier; while the second one seeks to *specialize* these examples. Generalization amounts to seeking a concept-description which characterizes all the examples of the concept while specialization consists of seeking a concept description which excludes all its counter-examples. Since within this framework, counter-examples of the concept are very important for training classifiers, in order to answer the first question asked in my dissertation, it is necessary to explain how concept-learning does occur in the absence of such examples when using an autoassociator. This is done by characterizing the specialization bias of the autoassociator. The second question asked in my dissertation is answered by studying the generalization strategies of MLP and the autoassociator in great detail while the third question is answered by testing the flexibility of the autoassociator's specialization bias relative to that of MLP.

The conclusions I have reached from the study of this dissertation are that:

- Autoassociation-based classification is capable of classifying domains of various dimensionality—at least *nonlinear multi-modal* ones—despite the absence of counter-examples and prior knowledge about the task, because they use a multi-modal interpolation/specialization bias. Furthermore, this bias presents some flexibility that can be regulated by means of increasing or decreasing the capacity of the network.

- Autoassociation-based classifiers can learn certain nonlinear multi-modal domains much more efficiently than their discrimination counterpart, MLP, because they are able to use a data-driven search of the generalization space rather than the time latency-bound hypothesis-driven search strategy used by MLP on these domains.

- Autoassociation-based classification is more accurate than MLP on two-dimensional domains that require strong specialization caused by the counter-concept class and on domains that require weak specialization caused by the concept class. However, MLP is more accurate than autoassociation-based classification on domains that require strong specialization caused by the concept class. These results also apply to higher dimensional domains for which the extra dimensions are invisible or close to invisible to the autoassociative process (e.g., if the extra dimensions contain discriminatory though redundant information or if they contain irrelevant information of low variance). They do not apply in cases where the extra dimensions are visible (e.g., if the extra dimensions introduce explicit class information or if they have no discriminatory power but high variance).

In the process of answering the particular questions asked in my dissertation, I have also made the following observations:

- I have shown that despite some conventional beliefs in the neural network community, autoassociators with nonlinearities in their hidden units do not have to yield results similar to Principal Component Analysis.
- I have shown that the backpropagation procedure has the flexibility of using different generalization strategies depending on certain structural and dimensional architectural features of the network to which it is applied.
- I have shown that, despite some previous statements, the effective number of hidden units used by the networks can be equal to the actual number of hidden units present, even if convergence could be reached using a number of hidden units smaller than the actual number of hidden units present in the network. This suggests that feedforward neural networks are not always self-regularized as far as their choice of an effective capacity is concerned.

The following section discusses several avenues of research worth investigating in the future.

8.2 Future Work

This section proposes eight different extensions of the studies conducted in this dissertation. These extensions are organized in the following groups:

- Additional Observations
- Simple Practical Extensions
- Practical Extensions by Combination
- Studies of Psychological Relevance

8.2.1 Additional Observations

Three extensions of my dissertation could be carved out that fall in the category of additional observations. The first one suggests the exploration of a deeper explanation of the processes taking place in the autoassociator studied in this dissertation, the second one proposes to explain what causes neural networks sometimes to use a subset of their hidden units and sometimes to use the entire set, and the third one proposes to study a related but different device.

My first suggestion for extending this work is to explain more specifically how classification occurs in the autoassociator by observing the hidden unit activation planes (HUAPs) that it builds more carefully. In particular, I propose to plot the projection of both concept and counter-concept data (so far, only the concept data were plotted), as well as the boundaries implemented by the hidden-to-output weights of the network, according to [Munro1991]'s strategy. This should be useful for clarifying why autoassociators discriminate between positive and negative instances of a concept the way they do. In addition, I propose to link the studies of Chapter 6 and 7 further, in order to find out how the different artificial domains tested in Chapter 7 cause different types of hidden unit activation plane (HUAP) representation. Such research should allow for a deeper understanding of where autoassociators are likely to succeed and where they are likely to fail and what type of external biases could be applied to make them more robust and accurate.

The second question I propose to study parallels the question studied in Chapter 6 on the efficiency of the autoassociator and of the MLP network. In that section, I discovered that the network's architecture constrained backpropagation to adopt a bottom-up or a top-down search of the generalization space. The reasons why a neural network will sometimes use all its hidden units to compute a solution when it is fully capable of finding such a solution with a smaller number of hidden units needs investigation. Similarly, it would be helpful to find out what triggers the network to use fewer hidden units than available to compute a solution. This study could be concluded by comparing the performance of the two systems in both cases in order to find out whether it would be useful to induce one of them artificially, whenever possible.

The third question of interest within the context of additional observations concerns the type of autoassociator used for classification tasks. My dissertation concentrated on the use of single hidden layer autoassociators with nonlinearities in either only their hidden layer or both their hidden and output layer. It would be interesting to find out how the 3-hidden layer autoassociator architecture introduced by [Kramer1991] would fare in the context of the same classification tasks as those considered in my dissertation (this architecture has already been tested in the context of multi-class digit recognition [Schwenk and Milgram1995]) and what sort of reconstruction error surface it yields. Similarly, it would be interesting to investigate the use of different functions in the nonlinear layers. For example, it would be interesting to find out what the use of a radial-basis function instead of the logistic function currently used would produce on my applications.

8.2.2 Simple Practical Extensions

While my dissertation demonstrates the practicality of the autoassociation-based classifier on several real-world domains, it would be useful to make the system more robust and to confirm its practicality on further domains. It would also be useful to quantify the strategy of Chapter 7 for predicting which of the autoassociator or MLP is more appropriate for a given domain.

Within this research framework, my first suggestion for future work is, therefore, to design and test reliable threshold-determination components. Particular emphasis should be given to devices that can choose a boundary in the absence of counter-examples. Furthermore, it would be interesting to apply weight-decay techniques to replace the preliminary setting phase of the autoassociator. Such research on both front could completely rid the autoassociation-based classification process of any counter-examples.

My second suggestion consists of testing the system on a variety of different real-world data sets each presenting different types of domain characteristics. In particular, I have not tested autoassociation-based classification on very large data sets or data sets containing noisy or missing attributes, and so on.

My third suggestion concerns the quantification of the predictive strategy of Chapter 7. This could be pursued in conjunction with the previous suggestion. In particular, after running the two classifiers on a large number of domains and finding out which of the two classifier is more appropriate for each domain, we could compute the different domain statistics described in Chapter 7 for each domain and apply a decision tree learning technique (e.g., CART or C4.5) on the statistical description of each domain, with, as class information, the name of the most accurate classifier on that domain. This study would result in a series of precise meta decision rules that could be subsequently used for selecting a classifier, given a new domain.

8.2.3 Practical Extensions by Combination

Two extensions of my work which consist of combining different autoassociation-based classifiers are also possible.

First, the results of Chapter 5 on the 3-dimensional extension of the original artificial domain suggest that different capacities of autoassociators perform differently on different types of data. It would be interesting to find a way to combine autoassociators of different capacities in order to optimize classification performance. The most natural scheme for such combinations is the mixture-of-experts schemes introduced by [Jacobs *et al.*1991] since it divides the space into different parts and applies the most appropriate expert on each of these parts. Another possibility could be to use the Boosting schemes of [Schapire1990] and [Freund1995].

The second idea for combination of different autoassociators consists of using an autoassociator to learn the concept class, using an autoassociator to learn the counter-concept class and merging the two results together using an adaptive voting procedure in which the weight of each classifier's vote would be determined during the training process. In addition, the classification scheme I used and analyzed in this dissertation could be expanded to multi-class problems (see [Schwenk and Milgram1995]).

8.2.4 Studies of Psychological Relevance

The last suggestion for future work, proposed in this section, consists of testing the validity of autoassociation-based classification as a psychological model. Autoassociation-based classification is based on the Gluck-Myers model of the hippocampus which is able to predict accurately a range of classical conditioning behaviors observed in normal and hippocampal-damaged animals. Furthermore, the system was shown to be useful for modeling certain aspects of natural language acquisition. It would be interesting to find out whether, in addition to the practical problems on which I have tested the system, it has any additional psychological relevance. In particular, it might be interesting to find out how well it can reproduce "Old-New" recognition data sets which test people's ability to distinguish objects previously presented to them from completely novel ones.

8.3 Implications of this Research

The research conducted in this dissertation has a couple of important implications for the fields of automated learning and neural networks. Parts of my work are also relevant to biological learning.

First, my dissertation suggests that concept-learning by recognition, and more generally, unsupervised learning may be advantageous in contrast to discrimination-based and supervised learning. Until now, comparisons of the two methodologies have been uncommon since the two approaches have been assumed to solve different types of problems. While supervised learning is usually aimed at classification tasks, unsupervised learning usually targets clustering problems. My dissertation represents a first step in the study of a comparative contrasting of recognition-based (or unsupervised) versus discrimination-based (or supervised) methods of classification, and the success of the autoassociation-based classification approach relative to that of MLP suggests that the field of automated learning could benefit from other studies of this sort and from the design of other recognition-based approaches. In particular, it would be interesting to find out whether there exist categories of problems for which any discrimination-based approach to classification will be outperformed by some good recognition-based approach, and whether, therefore, recognition-based learning should be used even if counter-examples are available in these domains.

The second implication of my dissertation concerns the discovery that the backpropagation procedure is flexible enough to support either a hypothesis-driven or a data-driven search of the generalization space, and, can spontaneously select different effective numbers of hidden units to solve a problem. While the flexibility of the backpropagation procedure was not foreign to researchers in the connectionist community, its concrete specification in terms of search concepts such as those I used can help us understand the operation of the backpropagation procedure better and can, perhaps, help us find principled ways to optimize its performance. Furthermore, since despite their complexity, artificial neural networks are still considerably simpler than their biological counterparts, my observations constitute a tractable step towards the explanation of the computations taking place in biological systems.

Appendix A
An Analysis of Real World Domains

This appendix describes a graphical method for analyzing the three real world domains used in some of the experiments of Chapter 4. The purpose of this analysis is to direct the construction of the artificial domain used throughout the dissertation. My analysis began by projecting the three multivalued real-domains onto their first and second principal components and displaying them graphically. A systematic method of analysis was then devised and applied to the projected domains. This method uses successively the graph theory concepts of *Minimum Spanning Trees* and *Convex Hulls* [Cormen *et al.*1990], and devises a classification geared at minimizing the degree of intersection of conceptual and counter-conceptual instances of these two constructs. In more detail, my method comprises four steps:

Step 1: Construction of a conceptual and counter-conceptual minimum spanning tree.

Step 2: Breaking of these trees into subtrees (First iteration).

Step 3: Construction of the convex hull of each subtree.

Step 4: Breaking of the subtrees into sub-subtrees (Second iteration).

The next paragraphs describe each of these steps in detail. At every step of the description, the method is illustrated on an artificial domain specially generated for this purpose and plotted in Figure A.1.[1]

Step 1

The first step of my analysis consists of building *minimum spanning trees (MSTs)* covering the positive and the negative data, respectively. The definition of a minimum spanning tree follows, adapted from [Cormen *et al.*1990]

> Let $G = (V, E)$ be a connected, undirected graph where V is the set of vertices and E the set of edges connecting these vertices. In addition, for each edge $(u, v) \in E$, there is a weight $w(u, v)$ specifying

[1]This domain is composed of 15 positive and 20 negative instances of the concept, all generated at random according to the uniform distribution. Positive data points are illustrated by dark triangles while negative ones are illustrated by white circles.

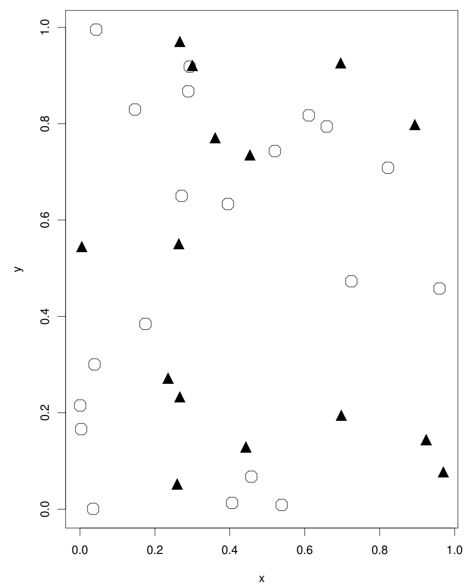


Figure A.1: Randomly Generated Domain for Illustration of the Graphical Analysis.

> the cost of connecting u to v. An MST for that graph is an acyclic subset $T \subset E$ that connects all of the vertices and whose total weight $w(T) = \S_{(u,v)\in T} w(u,v)$ is minimized.

From the set of conceptual and counter-conceptual data points, a positive and a negative MST are constructed. The cost function $w(u,v)$ is defined as the distance between u and v. The Minimum Spanning Tree formalism was chosen in this analysis because it is a simple way to organize the data that takes into consideration the within-class distance, an important feature in the context of classification. Figure A.2 illustrates the random domain of Figure A.1 after the two spanning trees have been generated.

Step 2

In the second step, the MSTs generated in the first step are broken into subtrees wherever edges of the two MSTs intersect. In such a case, the two edges are deleted, leaving 2 positive and 2 negative subtrees instead of a single positive and a single negative tree. The breaking of trees into subtrees in this manner was chosen because it is a simple way to divide the original data into positive and negative clusters. Figure A.3 illustrates the random domain of Figure A.1 after the completion of Steps 1 and 2.

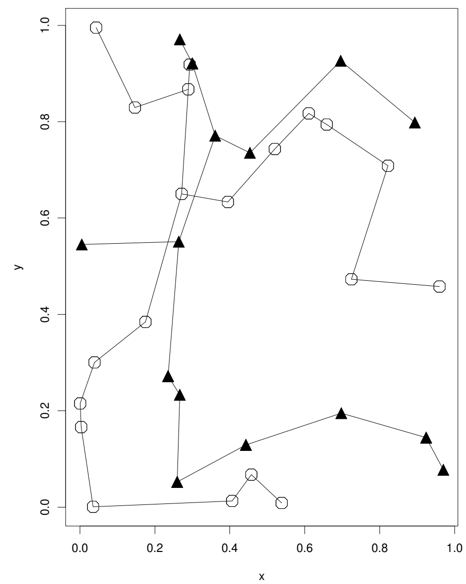


Figure A.2: The randomly generated domain after Step 1. At this point, the positive and negative MSTs have both been constructed.

Step 3

In the third step, *convex hulls* are generated for each of the subtrees that result from the previous step. The definition of a convex hull is given below from [Cormen *et al.*1990]:

> A convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Convex hulls were formed for each of the conceptual and counter-conceptual subtreed of Figure A.3. Once again, the reason for choosing the convex hull formalism for this analysis is that it allows for generalization of the conceptual and counter-conceptual data, much in the way in which commonly used inductive clusters (e.g., MLP, C4.5, CART) proceed. Figure A.4 illustrates the random domain of Figure A.1 after the completion of Steps 1, 2 and 3.

Step 4

In the fourth step, the subtrees that were generated in Step 2 are further dismantled in the locations where the convex hulls intersect. Convex hulls of the new resulting subtrees are then drawn. In locations where two convex hulls intersect, two sorts of actions are taken. First, if a boundary of a convex hull which is not an edge of the original MSTs intersects with the edge of an MST of the opposite class, then that edge is deleted. Second, if a convex hull is completely included in another convex hull of opposite class, and if that convex hull contains at least

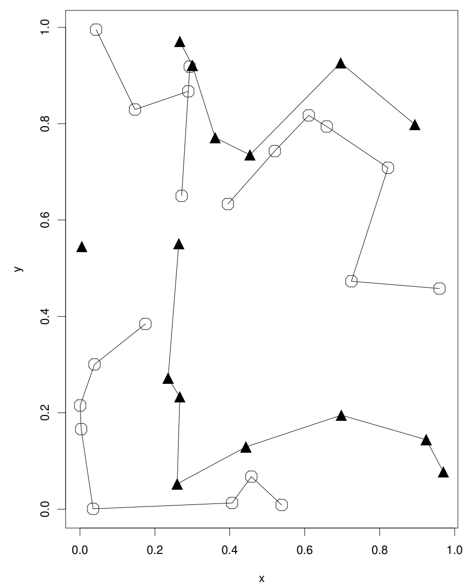


Figure A.3: The randomly generated domain after Steps 1 and 2. At this point, the positive and negative MSTs constructed in Step 1 have been broken according to Step 2.

two vertices, then the longest edge of the largest convex hull that will separate the two opposite hulls is deleted and two new hulls are formed from the original largest one. This step is again a formalism that will allow us to separate the two classes in a formal manner that seems reasonable. Figure A.5 & A.6 illustrates the random domain of Figure A.1 after the completion of Steps 1, 2, 3 and 4. Figure A.5 illustrates the further breaking of the subtrees while Figure A.6 illustrates the domain after new and old convex huls have been re-generated.

This graphical analysis was performed on the 2-Dimensional spaces representing the first and second principal components of the three real domains used in the preliminary experiments of Chapter 1: Helicopter gearbox monitoring, DNA promoter recognition, and Sonar target recognition. The results of this analysis on these domains are displayed in Figure 4.3 of Chapter 4.

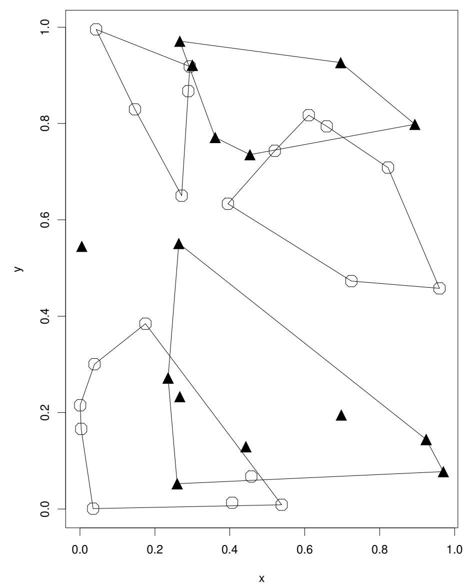


Figure A.4: The randomly generated domain after Steps 1, 2, and 3. At this point, a convex hull is generated around the positive and negative subtrees obtained after Step 2.

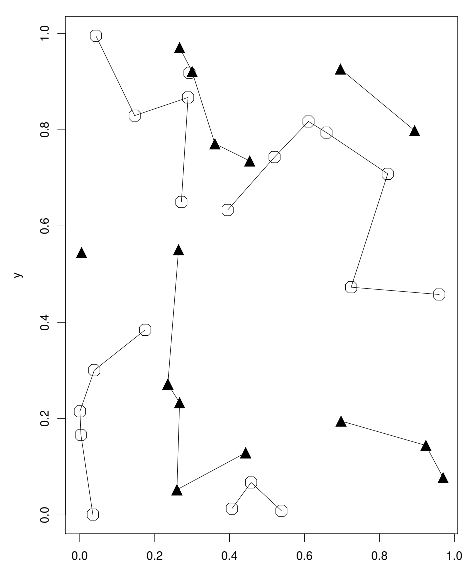


Figure A.5: The randomly generated domain after Steps 1, 2, 3, and 4 (Part I). At this point the subtrees generated at Step 2 are further dismantled at the locations where two convex hulls intersect.

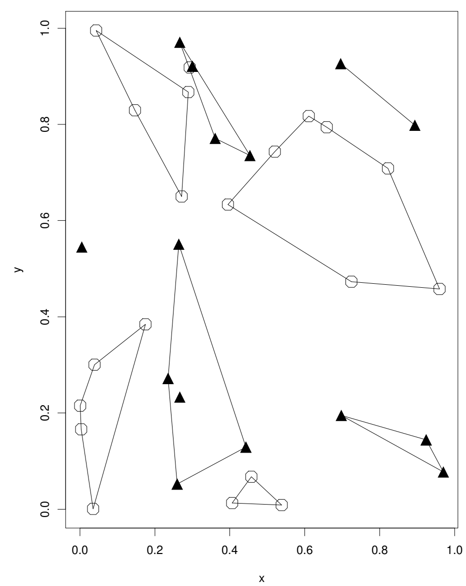
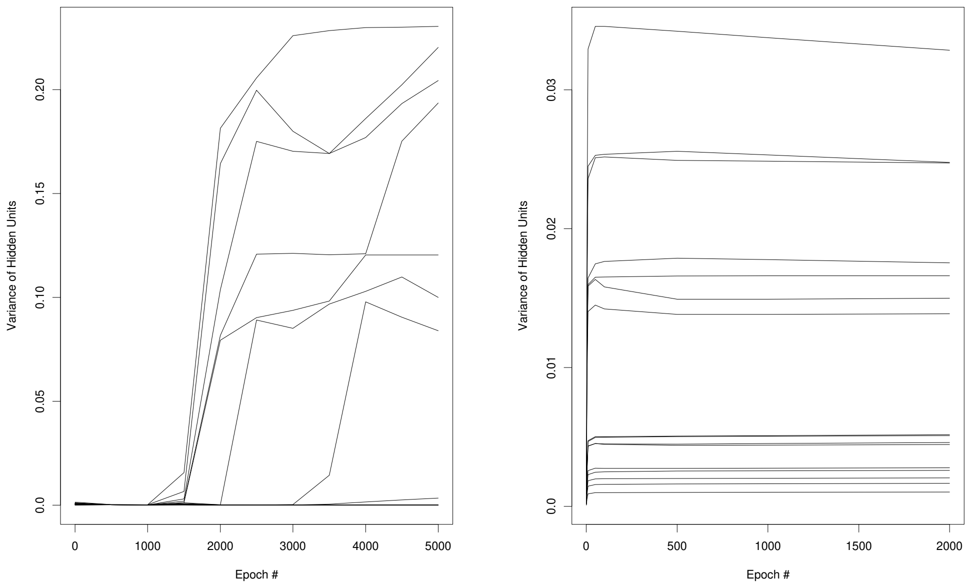


Figure A.6: The randomly generated domain after Steps 1, 2, 3, and 4 (Part II). At this point, new convex hulls are built around the final subtrees generated by our procedure.

Appendix B
Duplication of the Variance and Effectiveness Experiments of Chapter 6

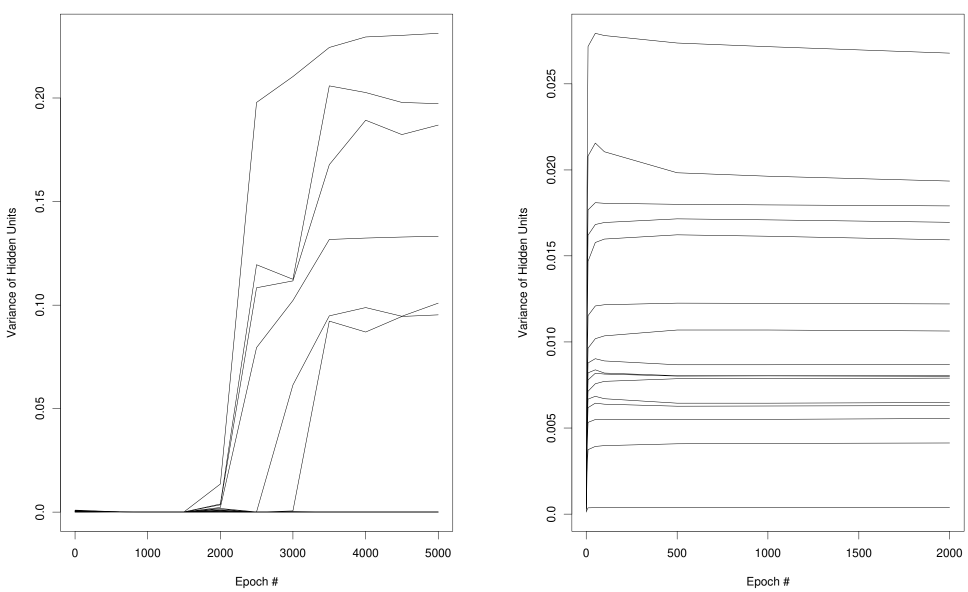
In this appendix, I display the results of the duplicated experiments of Section 6.2.1 and 2 of Chapter 6. Figure B.1 to B.4 are the result of the duplicated experiments that gave rise to Figure 6.1 of Section 6.2.1. These figures represent the results of Weigend's variance analysis of the hidden units of MLP and the autoassociator when 16 hidden units are used. Figure B.5 to B.8 are duplications of the experiments whose results were displayed in figure 6.3 of Section 6.2.1. These figures are the same experiments except for the fact that 64 hidden units were used. Figures B.9 to B.12 correspond to the experiments displayed in Figure 6.4 of Section 6.2.2 which correspond to the PCAs of the hidden unit activations of the two systems used with 16 hidden units while Figures B.13 to B.16 correspond to those of figure 6.5 of Section 6.2.2, which are the same experiments except that the sizes of the models are now 64.

The curves obtained in the experiments reported in this appendix corroborates those obtained in Chapter 6 except in one case. In the case where the autoassociator is used with 64 hidden units, results displayed in Figures B.6/B.14 of this appendix are different from those displayed in Chapter 6 and in figures B.5/B.13, B.7/B.15 and B.8/B.16. While in Chapter 6 and figures B.5/B.13, B.7/B.15 and B.8/B.16, the hidden units of the autoassociator with 64 hidden units all varied, but with an effectiveness of only about 3 hidden units, in figure B.6/B.14 all its hidden units are effective. This divergence is discussed in Section 6.3 of Chapter 6.



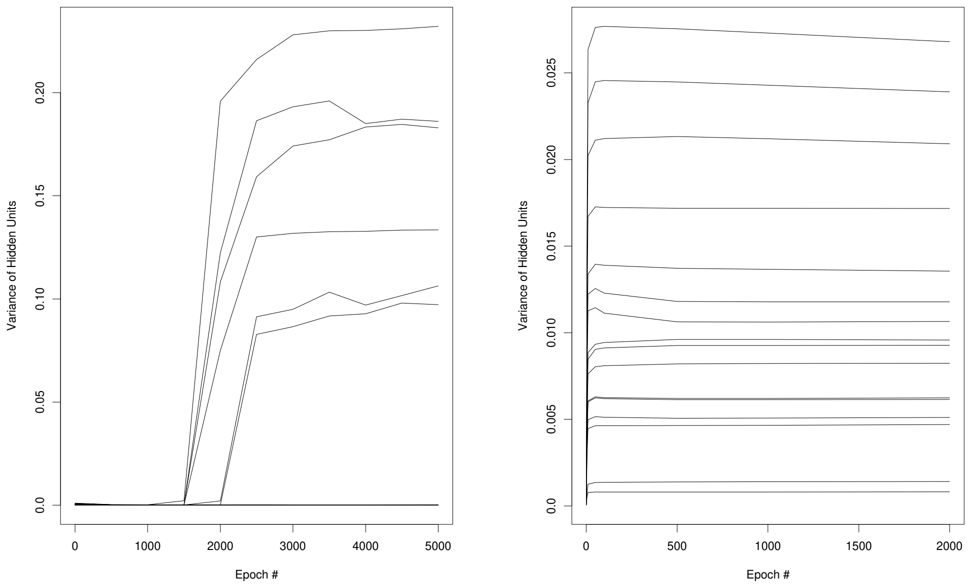
(a) MLP (b) Autoassociator

Figure B.1: The variances of the hidden units of MLP and the autoassociator on the artificial task with 16 hidden units.



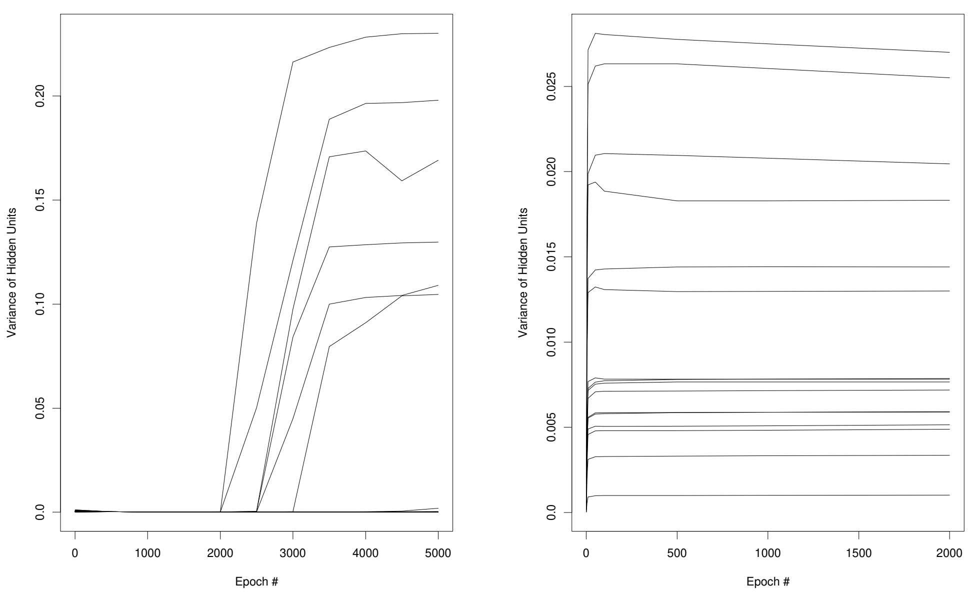
(a) MLP (b) Autoassociator

Figure B.2: The variances of the hidden units of MLP and the autoassociator on the artificial task with 16 hidden units.



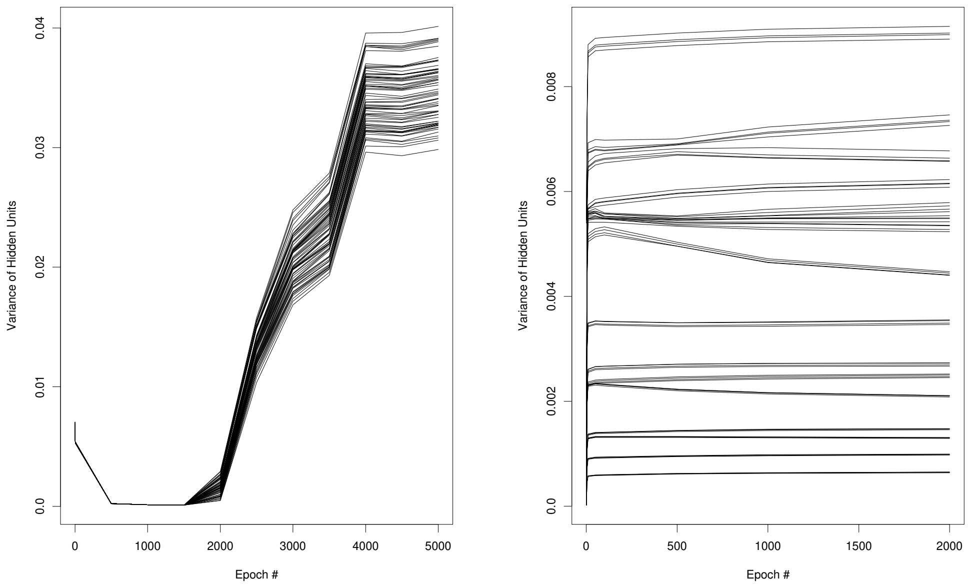
(a) MLP (b) AUTOASSOCIATOR

Figure B.3: The variances of the hidden units of MLP and the autoassociator on the artificial task with 16 hidden units.



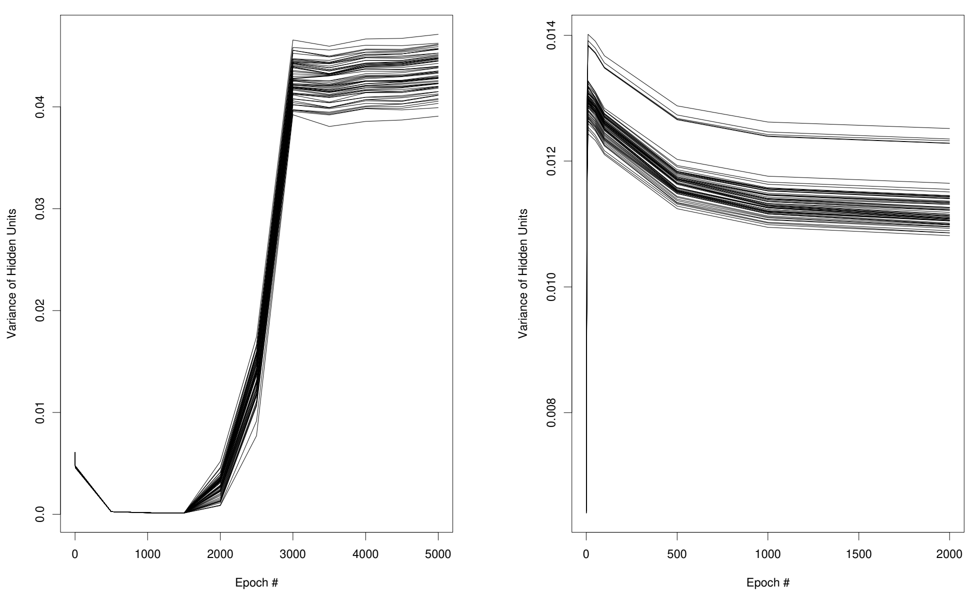
(a) MLP (b) AUTOASSOCIATOR

Figure B.4: The variances of the hidden units of MLP and the autoassociator on the artificial task with 16 hidden units.



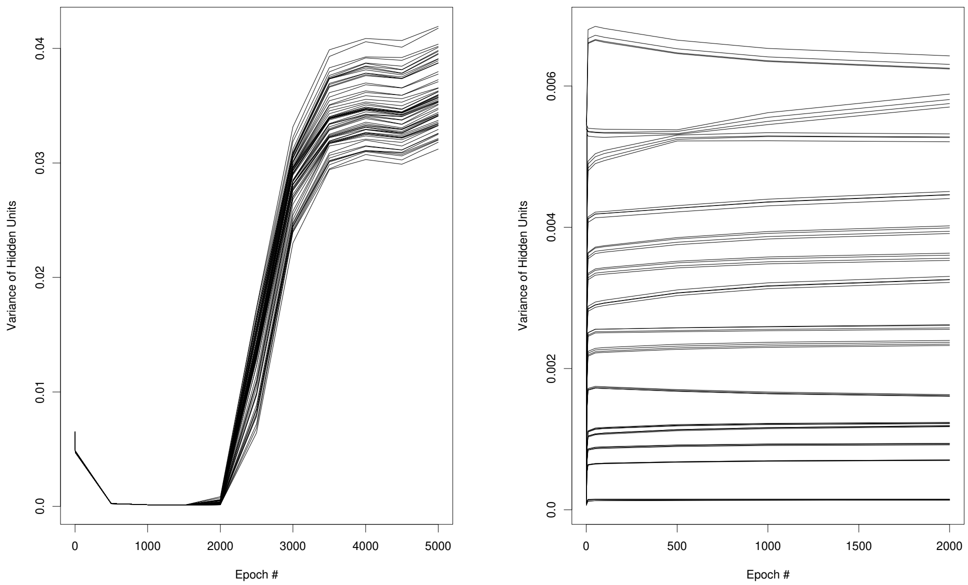
(a) MLP (b) Autoassociator

Figure B.5: The variances of the hidden units of MLP and the autoassociator on the artificial task with 64 hidden units.



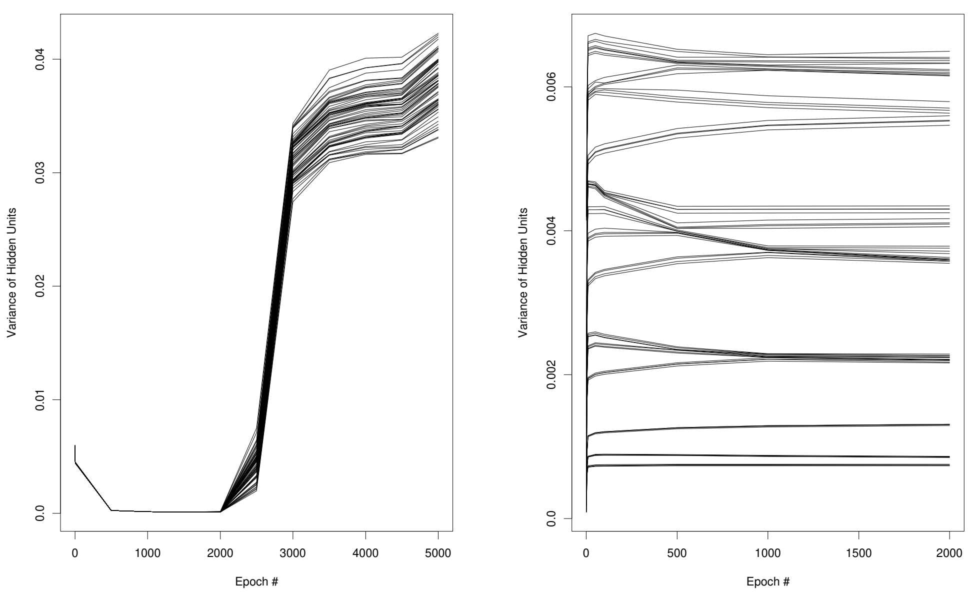
(a) MLP (b) Autoassociator

Figure B.6: The variances of the hidden units of MLP and the autoassociator on the artificial task with 64 hidden units.



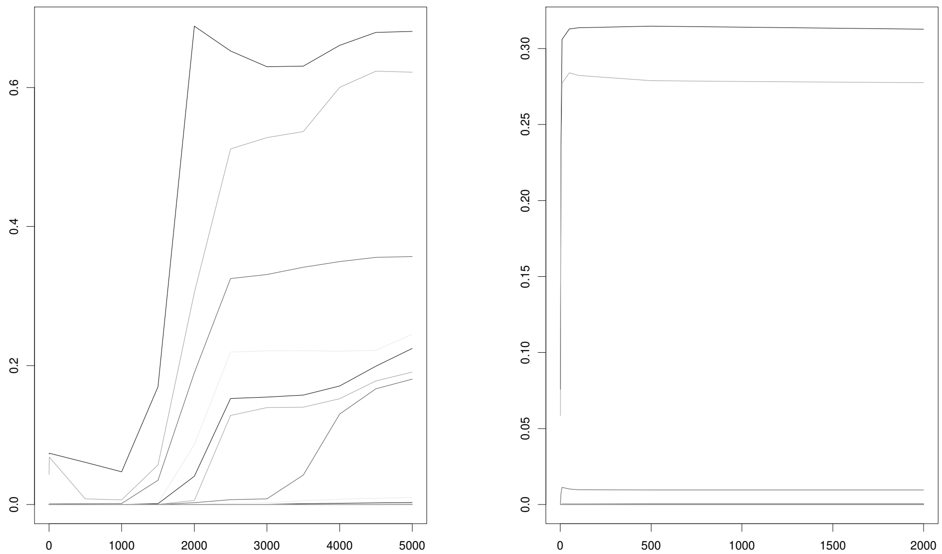
(a) MLP (b) AUTOASSOCIATOR

Figure B.7: The variances of the hidden units of MLP and the autoassociator on the artificial task with 64 hidden units.



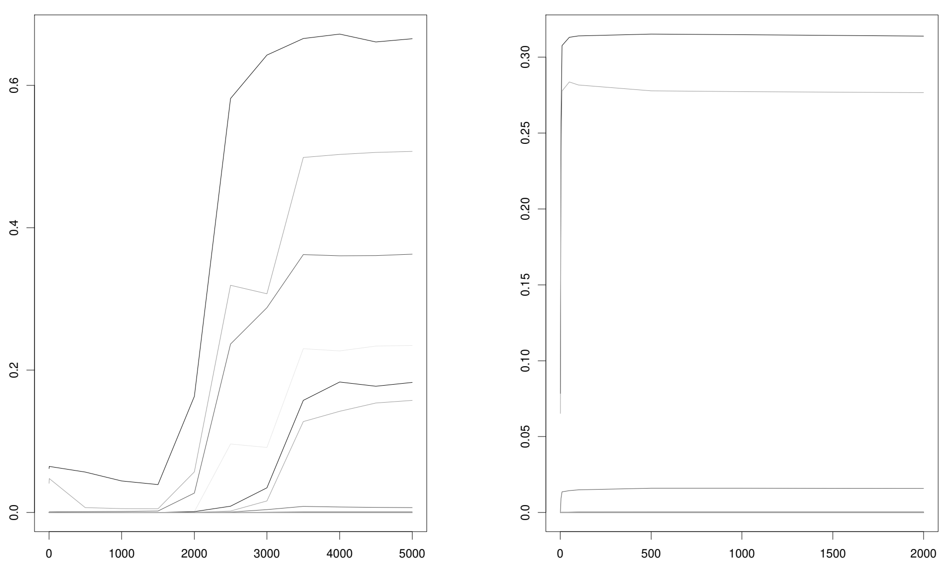
(a) MLP (b) AUTOASSOCIATOR

Figure B.8: The variances of the hidden units of MLP and the autoassociator on the artificial task with 64 hidden units.



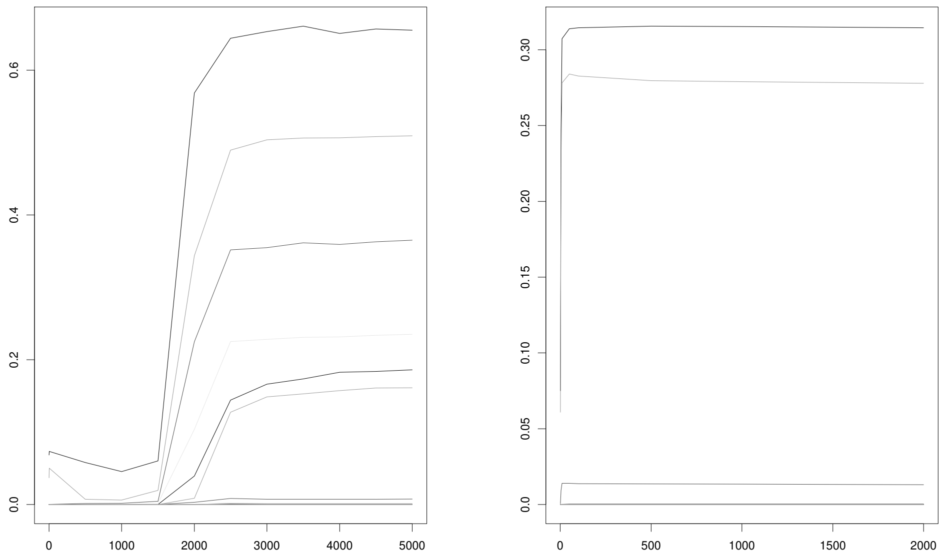
(a) MLP (b) AUTOASSOCIATOR

Figure B.9: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 16$.



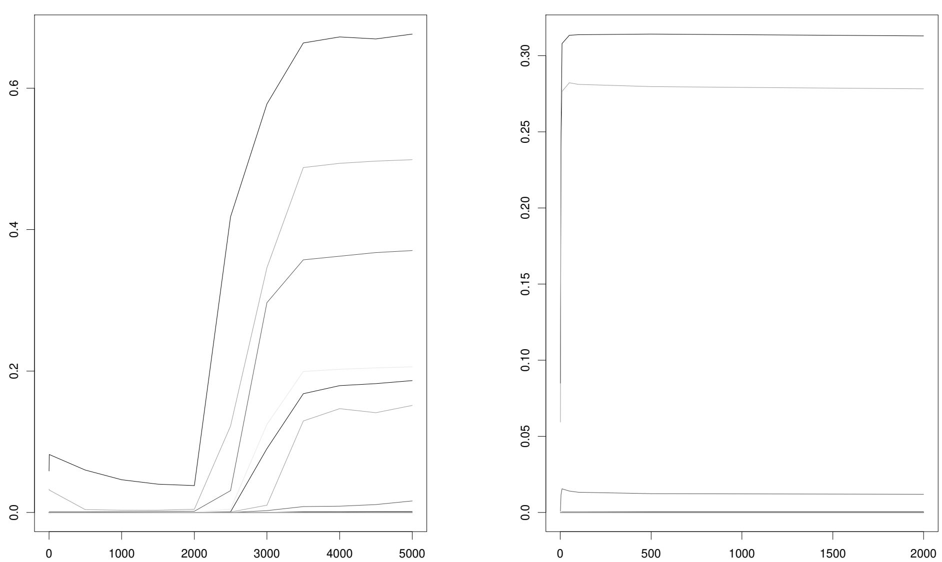
(a) MLP (b) AUTOASSOCIATOR

Figure B.10: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 16$.



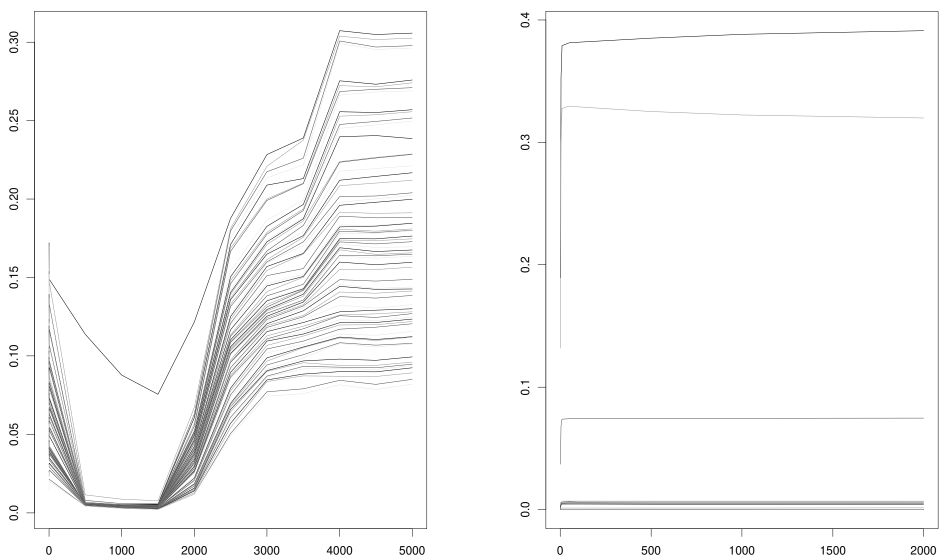
(a) MLP (b) AUTOASSOCIATOR

Figure B.11: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 16$.



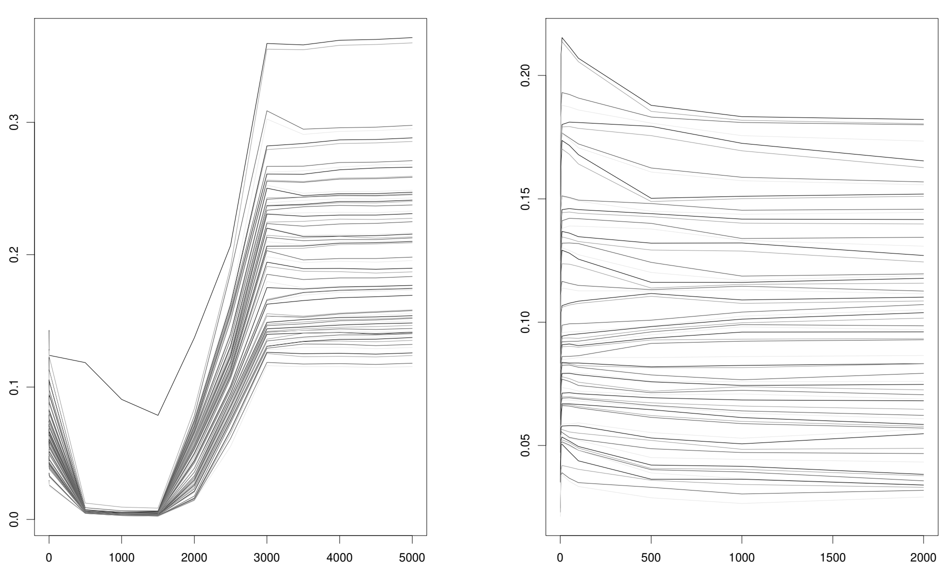
(a) MLP (b) AUTOASSOCIATOR

Figure B.12: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 16$.



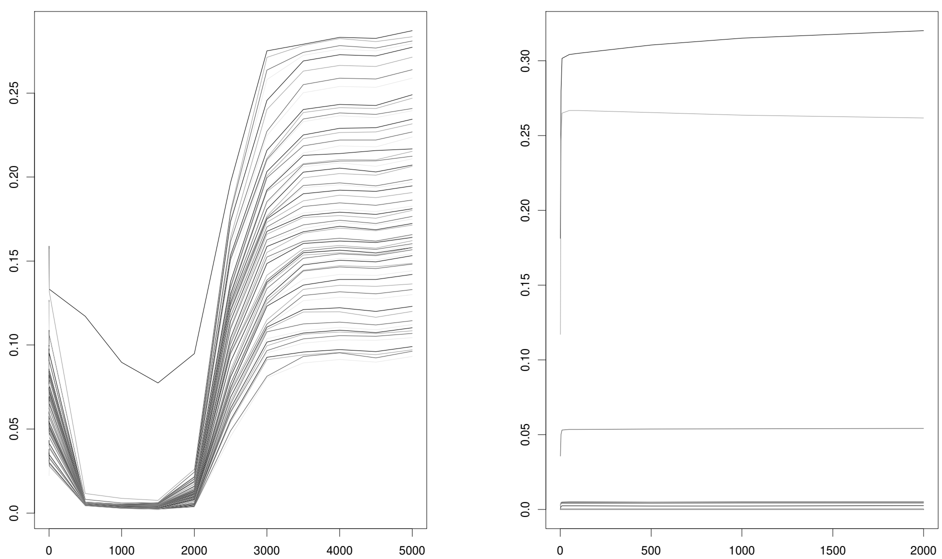
(a) MLP (b) AUTOASSOCIATOR

Figure B.13: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 64$.



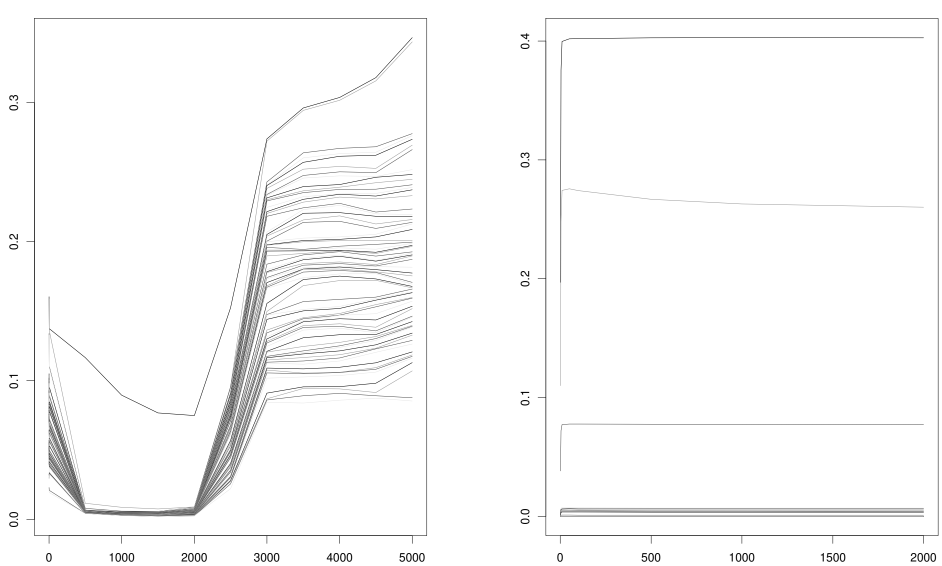
(a) MLP (b) AUTOASSOCIATOR

Figure B.14: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 64$.



(a) MLP (b) AUTOASSOCIATOR

Figure B.15: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 64$.



(a) MLP (b) AUTOASSOCIATOR

Figure B.16: The principal components of the hidden units of MLP and the autoassociator as a function of the epoch number. The model size is $h = 64$.

References

[Aha *et al.*1991] Aha, Kibler, and Albert. Instance-based learning algorithms. *Machine Learning*, 6(1), Jan 1991.

[Baldi and Hornik1989] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

[Bishop1995] Chris Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[Bourassa1999] Michael Bourassa. Personal communication, 1999.

[Bourlard and Kamp1988] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988.

[Breiman *et al.*1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[Cormen *et al.*1990] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw Hill, 1990.

[Cottrell and Munro1988] Garrison W. Cottrell and Paul Munro. Principal components analysis of images via back propagation. In *Invited Paper in Proceedings of the Society of Photo-Optical Instrumentation Engineers*, Cambridge, MA, 1988.

[Cottrell *et al.*1987] Garrison W. Cottrell, Paul Munro, and David Zipser. Learning internal representations from gray-scale images: An example of extensional programming. In *Proceedings of the ninth AnnualConference of the Cognitive Science Society*, pages 462–473, Seattle, WA, 1987.

[DeMers and Cottrell1993] David DeMers and Garrison Cottrell. Non-linear dimensionality reduction. In Hanson, Cowan, and Giles, editors, *Advances in Neural Information Processing Systems 5*. Morgan Kauffmann, San Mateo, CA, 1993.

[Denker *et al.*1987] J. Denker, D. Schwartz, B. Wittner, S. Solla, H. Richard, Jackel L., and Hopfield J. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

[Elman and Zipser1988] J. L. Elman and D. Zipser. Learning the hidden structure of speech. *Journal of the Acoustical Society of America*, 83:1615–1626, 1988.

[Freund1995] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[Fukunaga1990] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (2nd Edition)*. Academic Press, San Diego, CA, 1990.

[Geman *et al.*1992] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilema. *Neural Computation*, 4(1):1–58, 1992.

[Gluck and Myers1993] Mark A. Gluck and Catherine E. Myers. Hippocampal mediation of stimulus representation: A computational theory. *Hippocampus*, 3(4):491–516, October 1993.

[Hanson and Kegl1987] Stephen J. Hanson and Judy Kegl. Parsnip: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the Ninth Annual Conference on Cognitive Science*, 1987.

[Hazlett and Bach1977] Brian A. Hazlett and Catherine E. Bach. Predicting behavioral relationships. In Brian A. Hazlett, editor, *Quantitative Methods in the Study of Animal Behavior*, pages 121–144. Academic Press, New York, 1977.

[Hertz *et al.*1991] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Santa Fe Institute, Studies in the Sceinces of Complexity, Addison-Wesley, Redwood City, CA, 1991.

[Intrator1993] N. Intrator. Combining exploratory projection pursuit and projection pursuit regression with application to neural networks. *Neural Computation*, 5(3):443–455, 1993.

[Jacobs *et al.*1991] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[Japkowicz *et al.*1995] Nathalie Japkowicz, Catherine Myers, and Mark Gluck. A novelty detection approach to classification. In *Proceedings of the Fourteenth Joint Conference on Artificial Intelligence*, pages 518–523, 1995.

[Kambhatla and Leen1994] Nanda Kambhatla and Todd K. Leen. Fast nonlinear dimensionality reduction. In Cowan, Tesauro, and Alspector, editors, *Advances in Neural Information Processing Systems 6*. Morgan Kauffmann, San Francisco, CA, 1994.

[Kolesar and NRaD1994] Bob Kolesar and NRaD. Helicopter gearbox monitoring (presented). 1994.

[Kortge1990] Chris A. Kortge. Episodic memory in connectionist networks. In *The Twelfth Annual Conference of the Cognitive Science Society*, pages 764–771, 1990.

[Kramer1991] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[Mitchell1982] Tom Michael Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, March 1982.

[Moody1992] John E. Moody. The *effective* number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In S.J. Hanson J.E. Moody and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 NIPS Conference*, pages 847–854. Morgan Kaufmann, San Mateo, CA, 1992.

[Moody1994] John E. Moody. Prediction risk and architecture selection for neural networks. In J.H. Friedman V. Cherkassky and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*. NATO ASI Series F, Springer-Verlag, 1994.

[Munro1991] Paul Munro. Visualizations of 2-d hidden unit space. Technical Report LIS035/IS91003, School of Library and Information Science, University of Pittsburgh, 1991.

[Myers and Gluck1994] Catherine Myers and Mark Gluck. Context, conditioning and hippocampal representation in animal learning. *Behavioral Neuroscience*, 108(5):835–847, 1994.

[Norton1994] Steven W. Norton. Learning to recognize promoter sequences in E. coli by modeling uncertainty in the training data. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[Oja1991] Erkki Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 737–745. 1991.

[Oja1995] Erkki Oja. The nonlinear pca learning rule and signal separation – mathematical analysis. Technical Report A26, Helsinki University of Technology, 1995.

[Pentland *et al.*1994] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 84–91, 1994.

[Petsche *et al.*1996] Tom Petsche, Angelo Marcantonio, Christian Darken, Steve J. Hanson, Gary M. Kuhn, and Iwan Santoso. A neural network autoassociator for induction motor failure prediction. In *Advances in Neural Information Processing Systems 8.*, MIT, Cambridge, MA, 1996. MIT Press.

[Plaut *et al.*1986] D. Plaut, S. Nowlan, and G.E. Hinton. Experiments on learning by backpropagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, 1986.

[Pomerleau1993] Dean A. Pomerleau. Input reconstruction reliability estimation. In *Proceedings of the Fifth Neural Information Processing Systems Conference*, 1993.

[Quinlan1993] John Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[Rumelhart *et al.*1986] David E. Rumelhart, Geoff E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, pages 318–364. MIT Press, Cambridge, MA, 1986.

[Saund1989] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, mar 1989.

[Schapire1990] R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[Schwenk and Milgram1995] Holger Schwenk and Maurice Milgram. Transformation invariant autoassociation with application to handwritten character recognition. In David S. Touretsky, Gerald S. Tesauro, and Todd K. Leen, editors, *Neural Information Processing Systems (NIPS*7)*, pages 991–998. MIT Press, 1995.

[Sinha1994] P. Sinha. Object recognition via image invariants: A case study. *Investigative Ophtalmology and Visual Science*, 35:1735–1740, May 1994.

[Squire1987] L. Squire. *Memory and Brain.* OUP, New York, 1987.

[Stainvas1999] Inna Stainvas. *Trade-off between recognition and reconstruction: Application of Neural Networks to Robotic Vision.* PhD thesis, Tel Aviv University, 1999.

[Sung and Poggio1994] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. Technical Report A.I. Memo No. 1521, Artificial Intelligence Laboratory, M.I.T., 1994.

[Turk and Pentland1991] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[Usui *et al.*1991] S. Usui, S. Nakauchi, and M. Nakano. Internal color representation acquired by a five-layer neural network. In T. Kohonen, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 867–872. 1991.

[Van Trees1968] Harry L. Van Trees. *Detection, Estimation, and Modulation Theory.* Wiley, New York, 1968.

[Weigend1994] Andreas S. Weigend. On overfitting and the effective number of hidden units. In M.C. Mozer, P. Smolensky, D.S. Touretzky, J.L. Elman, and A.S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 335–342. Erlbaum Associates, Hillsdale, NJ, 1994.

[Weiss and Kulikowski1991] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn.* Morgan Kaufmann, San Mateo, CA, 1991.

[Wolpert1992] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

Vita

Nathalie Japkowicz

1984 Graduated from Lycée Louis-Le-Grand, Paris, France.

1988 B.Sc., McGill University, Montreal, Canada. First Class Honours.

1990 M.Sc. in Computer Science, University of Toronto, Toronto, Canada.

1999 Ph.D. in Computer Science, Rutgers, The State University of New Jersey, New Brunswick.

Publications:

1991 N. Japkowicz, J. Wiebe, "A System for Translating Prepositions from English into French", in *Proceedings of the Twenty Ninth Meeting of the Association for Computational Linguistics.*

1994 H. Hirsh, N. Japkowicz, "Bootstrapping Training-Data Representations for Inductive Learning: A Case Study in Molecular Biology", in *Proceedings of the Twelfth National Conference on Artificial Intelligence.*

1995 N. Japkowicz, C. Myers, M. Gluck, "A Novelty Detection Approach to Classification", in *Proceedings of the Fouteenth International Joint Conference on Artificial Intelligence.*

1999 N. Japkowicz, "Are We Better Off Without Counter-Examples?", in *Proceedings of the First Conference on Advances in Intelligent Data Analysis.*

1999 N. Japkowicz, S.J. Hanson "Adaptability of the Backpropagation Procedure", in *Proceedings of the 1999 International Joint Conference in Neural Networks.*