Combinatorial Optimization Algorithms Applied to Pattern Classification

Georgios Lappas

A thesis submitted in partial fulfilment of the
requirements of the University of Hertfordshire
for the degree of Doctor of Philosophy

The programme of research was carried out in the
Department of Computer Science,
Faculty of Engineering and Information Sciences,
University of Hertfordshire

January 2006

2

Abstract

Accurate classification by minimizing the error on test samples is the main goal in pattern classification. Combinatorial optimization is a well-known method for solving minimization problems, however, only a few examples of classifiers are described in the literature where combinatorial optimization is used in pattern classification. Recently, there has been a growing interest in combining classifiers and improving the consensus of results for a greater accuracy. In the light of the "No Free Lunch Theorems", we analyse the combination of simulated annealing, a powerful combinatorial optimization method that produces high quality results, with the classical perceptron algorithm. This combination is called LSA machine. Our analysis aims at finding paradigms for problem-dependent parameter settings that ensure high classification results. Our computational experiments on a large number of benchmark problems lead to results that either outperform or are at least competitive to results published in the literature. Apart from parameter settings, our analysis focuses on a difficult problem in computation theory, namely the network complexity problem. The depth vs size problem of neural networks is one of the hardest problems in theoretical computing, with very little progress over the past decades. In order to investigate this problem, we introduce a new recursive learning method for training hidden layers in constant depth circuits. Our findings make contributions to a) the field of Machine Learning, as the proposed method is applicable in training feedforward neural networks, and to b) the field of circuit complexity by proposing an upper bound for the number of hidden units sufficient to achieve a high classification rate. One of the major findings of our research is that the size of the network can be bounded by the input size of the problem and an approximate upper bound of $8 \cdot \sqrt{2^n/n}$ threshold gates as being sufficient for a small error rate, where $n := \log|S_L|$ and S_L is the training set.

4

Ackowledgements

First of all, I am sincerely grateful to my head supervisor Dr. Andreas Albrecht. He showed a deep concern for my professional and academic development and I will benefit greatly from his guidance, his encouregement, his advice, and his expertise that kindly was willing to share with me.

I would also like to thank my supervisor Dr. Ray Frank for his help, advice and for our very useful discussions. I like to acknowledge the guidance of Dr. Vivian Ambrosiadou during the first year of this work.

I would like to thank my institution, the T.E.I. of Western Macedonia, and especially Prof. Prodromos Yannas for persuading me to start this research, and for his sincere encouragement and support as a friend, colleague and the head of my department.

Mostly, I want to thank my wife Olga for supporting me all these years with all her love and patient. Moreover, just before completing this work she gave me ultimate happiness by giving birth to our daughter. How can I then forget to acknowledge my new born daughter for the happinnes she brought to my family and for making this thesis writing an unforgettable process. I would also like to thank my parents for all their love and their support. Finally many thanks to my colleagues and my friends for their help and support in any form.

Contents

List of Figures

List of Tables

6.2 Error Rates on Test Sets

Chapter 1

Introduction

Pattern classification is the process of inferring meaning (or else, assigning to *classes*) from observations, which are quantified by measurements called *features* or *attributes*, and the problem related with this process is called *the classification* or *the decision problem.* Since the foundation of artificial intelligence, pattern classification became an important field because of the challenge to mimic human recognition behaviour. The interest in pattern classification has been enormously increased and greatly appreciated, as humans *recognized* their limitations on making decisions when feature and domain complexity are increased. Pattern classification is now an excited field in computer science with ongoing intensive research, which is expected to continue for many years, as our ability to collect and store data grows with the explosion of technology.

1.1 Pattern Classification Essentials

Designing a pattern classification system is establishing a mapping from the measurement space into the space of *classes.* In a classification system one needs a function that rules the pattern. However, this function is never known in practice, and therefore one tries to approximately create such a function from sets of training samples, i.e. the *cases* drawn from the pattern source, and with an optimization criterion that adjusts the mapping of the measurements into the space of meanings in order to come closer to the true meaning. This *learning from examples* approach, which is the most distinguished feature of the *Machine Learning* field, enables the classification system to extract knowledge from the patterns. In pattern classification we design some kind of generic mapping containing a number of free parameters, which maps the observed measurement vector into the most probable meaning. With an optimization criterion we adjust the mapping to come as close as possible to the true meaning.

Neural networks from the early years have been widely used for pattern classification and consequently can be viewed as pattern classification devices. A wide variety of other approaches exists for undertaking the classification task. Two large classes of such other methods can be identified in the literature: *statistical approaches* and *Machine Learning approaches.* Statistical approaches focus on probabilities assigned to cases with respect to each of the classes. In the statistical pattern classification approach, all one needs are the probability functions ruling the pattern source. However, these statistical laws are never known but must be approximately reconstructed from sets of training samples drawn from the pattern source. *Bayes classifiers, support vector machines* are well known statistical approaches. Machine learning approaches like *decision trees, fuzzy* and *rule-based classifiers* try to generate classification expressions by learning a classification task from a series of examples.

Neural networks combine features of statistical techniques with Machine Learning methods. Neural networks provide a potential computing architecture with many useful applications, which have been designed, built and commercialised and much research continues in this field. The neural network approach has its origins in efforts to produce a computer model for the information processing that simulates the human nervous system. Neural networks can be viewed as a massively parallel computational device, composed of a large number of computational units, *neurons*, which are connected through interconnections with variable strengths, *weights.* Weights are trained and the learned information is stored in weights. Most neural networks perform a type of error minimization during the learning phase. The computing architecture and the learning philosophy behind it make neural networks very well suited for pattern classification. A number of complexity issues are in the focus of research on neural network classifiers for many years, like network architectures, connectivity topology, sample complexity and learning algorithm complexity. These are some of the parameters related to the performance of the neural classifier.

The most time-concerned process in pattern classification and in neural networks is the learning process ruled by *learning algorithms.* Supervised and unsupervised learning are two learning methods that rely on the help of a supervisor that knows the desired class of the pattern and thus is used for correcting the classifier's output class (supervised learning case); or the classifier tries to infers from the existing patterns without any supervisor's help. In supervised learning, the minimization procedure is closely related to the learning procedure of the network, which uses an iterative procedure of modifying the network weights and is guided by the *learning algorithm* that tries to correctly classify a pattern or some set of patterns.

The *perceptron algorithm* is such an iterative procedure that can find a solution on linearly seperable problems. Limitations of the perceptron algorithm apply to non-linearly seperable classes like the XOR problem. *Gradient descent* is another iterative procedure for minimizing the error by calculating the difference between desired and actual outputs by using error vectors . The learning method for a number of presented patterns follows the gradient on the average patterns by changing the weights by an amount proportional to the estimated error multiplied by the input weight. A generalization of the gradient descent procedure to multilayer feedforward networks is widely known as the *backpropagation method.* Multilayer networks and backpropagation have been used as methods to overcome the perceptron limitations for the XOR problem. The main consideration on the above learning methods is to take care of the degree of weight changes. If the weight changes are very large, then overcorrection occurs where the optimum will be overpassed, producing oscillations around it. If the changes are very small, then the convergence is too slow. We have seen limitations of the perceptron learning algorithm, but the gradient descent methods including backpropagation are not the ultimate learning methods for minimizing the error. The main two problems are: i) the methods are applicable only when the error can be defincd strictly as the length of an error vector, and ii) the learning method can be easily trapped to local minima. Anderson [24] refers also to other problems in gradient descent related to the fact that the method might be extremely slow in time with thousand and ten thousands of learning trials and it goes up rapidly in the size of the network. The error that we want to minimise in pattern classification is the *classification error*, i.e. the number of misclassified samples on training data as well as on unseen test samples. Minimizing the misclassification is equivalent to obtaining maximum generalization performance of the network. Therefore, a classification problem is also an *optimization problem*, where we seek for the maximum classification rate or the minimum misclassification.

1.2 Combinatorial Optimization Essentials

Optimization is the attempt to find the maximum or minimum of a certain function, called the *objective function*, which can be defined on some domain. The domain is often specified by a set of constraints, equalities or inequalities that the elements of the domain have to satisfy. In *combinatorial optimization* problems, the objective is to find the *best* or *optimal* solution among a finite or countable infinite number of alternative solutions [281]. Combinatorial optimization has been a research field for decades with the *traveller salesman problem (TSP)* being the most classical and representative problem of the

field. The TSP deals with finding the best journey for the salesman between a number of cities that can be visited only once, achieving the minimum cost. The problem is a very hard problem to solve due to the lack of an enumeration of all possible solutions and belongs to a special class of problems that describe the hardness of the problem, called *NP-complete* class. NP-complete problems have always the quest for designing efficient algorithms to cope with the hardness of the problem. A number of algorithms exist in the literature through this direction. Well known combinatorial optimization algorithms like genetic algorithms, simulated annealing, tabu search and Boltzmann machines have been inspired by nature and have been proved to be efficient algorithms for solving combinatorial optimization problems. Such problems cannot be solved in reasonable amount of time and only approximations of optimum solutions can be found. Such approximation algorithms are described in Section 3.4.1. Over the past decades, our ability to solve large combinatorial optimization problems has been dramatically improved due to the advent of powerful computer systems.

Simulated annealing is among the approximation algorithms that can find good solutions in reasonable amount of time. Although simulated annealing has been studied in combinatorial optimization since the early eighties [73, 213], the research on this method and its applications have been boosted by new findings and new application areas: Simulated annealing-based algorithms and associated techniques for landscape analysis, in particular variants based on inhomogeneous Markov chains, have been used to investigate problems from Computational Biology; cf. [150, 387] and the literature therein. Furthermore, recent advances in genetic algorithms research [326] have underpinned the importance of inhomogeneous Markov chains in the broader context of general local search methods: L.M. Schmitt proved that in order to ensure the convergence of genetic algorithms to optimum solutions, simulated annealing-based selection has to be employed in one way or another (see Section 10 in [326] for a summary of results).

Classification problems can be considered as a special class of combinatorial optimization problems [2]. In pattern classification, the requirement that the space of measurements maps into the correct classes determines an optimization problem, where the optimum among all conceivable mappings is one that in terms of maximization guarantees the maximum classification rate, and in terms of minimization guarantees the minimum error rate in the number of *misclassified patterns*, i.e. in the number of wrong mappings of the measurement space into the set of classes. Minimizing the error rate is closely related to approximating the function that rules the pattern and also to establishing an optimal weight configuration of the classification system that will maximize

the performance of the classification rate. Establishing an optimal weight or network configuration for the classifier and minimizing the error rate can be related to a combinatorial optimization problem that in most cases is hard and cannot be searched for optimal solution in efficient time. Thus, usually only approximations of solutions can be achieved.

Since searching the solution space for an optimum weight or network configuration is a minimization problem, combinatorial optimization algorithms can be applied to such tasks. The relation between neural networks and combinatorial optimization can be viewed as a two-fold relation with mutual benefits. Neural networks have been used for solving combinatorial optimization problems by approximating the best configuration that can describe the instance of a combinatorial optimization problem, and, on the other hand, combinatorial optimization algorithms can be used for obtaining an appropriate weight or network configuration of maximum classification performance. Looi's paper "Neural Network Methods in Combinatorial Optimization" [245] gives an overview of the former relation. Hopfield's and Tank's [184] work was one of the first approaches for using network energy minimization for solving the famous hard TSP combinatorial optimization problem (see Section 3.1.2). The network they proposed did not find the best solution but has approximated good solutions. Boltzmann machines [173] have been used for solving combinatorial optimization problems as well as pattern classification problems [106].

1.3 Motivation of this Research

In pattern classification, on non-seperable problems, finding the best function for minimizing the classification error is an NP-complete problem. Blum and Rivest [55] proved that even for a very small network to find weight and thresholds that learn any given set of training examples is an NP-complete problem. Unless $P = NP$, for any polynomial time training algorithm (see 3.3) there will be some set of training data on which the training algorithm fails to correctly classify the data. Höffgen and Simon [177] deal with the problem of learning a probably almost optimal weight vector for a neuron, finding that it is an NP-complete problem. Also finding an optimum network configuration for solving combinatorial optimization problems is not an easy task. Yannakakis [398] proved that the problem of finding a stable network configuration belongs to the categories PLS-complete, P-SPACE complete and P-complete problems, which in simple words are hard to solve problems. Thus, establishing an optimal weight configuration of threshold units (in order to minimize the error rate) cannot be executed in efficient time and therefore only approximations of optimum solutions can be achieved.

Combinatorial optimization is an establised method for minimization problems. Therefore, it is a promising method for pattern classification, where it is vital to minimize the number of misclassified examples. Despite the popularity of combinatorial optimization algorithms to solve hard problems, they rarely have been used in pattern classification (see Section 3.6). An example is simulated annealing combined with unsupervised neural networks producing the well known *Boltzmann machine* [173] introduced by Hinton and Sejnowski, which implements parallel simulated annealing. Albrecht and Wong [19] introduced a new learning machine called the *LSA Machine*, which is based on the combination of a specific type of simulated annealing with the perceptron algorithm for pattern classification. LSA machine was successfully applied to a number of problems and we like to further investigate the search strategies of this algorithm with respect to classification accuracy, to learning and convergence properties and to circuit complexity. The quest for learning algorithms that are efficient and capable to search the solution space for minimizing the classification error made the selection of good learning algorithms a competitive arena. However, in the light of the *No Free Lunch Theorems (NFLT)* [391], these learning algorithms cannot be solely viewed as universal classifiers and their performance is problem-dependent. Our motivation is to find rules for *a priori* setting the problem dependent parameters of the method based on properties of the classification problem for achieving high classification rates. Following these rules, a large number of experiments to identify good values for the parameters will be considerably limited.

Research on circuit complexity was motivated by our interest to work on larger depths and to investigate whether a small increase in depth obtains higher classification rates or at least as good as swallow depths but with considerably smaller network size. The number of hidden units and hidden layers of a network classifier form the circuit complexity problem of the classifier, a very hard and unsolved problem in theoretical computer science with only slow progress made over the past decades. The relevance of Kolmogorov's theorem [219] with the expressive power of depth-two neural networks, and the difficulty to find methods and topologies to train units at larger depths resulted in slow progress in the past in research on networks of large depth. Most researchers rely on the fact that depth-two neural networks are considered to be adequate as good classifiers. One of the very few existing works on larger depths is Lupanov's work [250], which gives an upper bound on the size of depth-four circuits as $size \leq X \cdot \sqrt{2^n/n}$. However, factor X remains an unknown constant. We are very interested to investigate the circuit complexity issue, trying to contribute on a very hard problem in computer science. We would like to provide a learning method for training units at larger depths

and to find any estimation for the network size required for achieving high classification rates.

1.4 Focus of this Research

We particularly focus on complexity issues with much effort placed on the size and depth complexity of the underlying architecture of the LSA machine. The research on larger circuit depths requires that we investigate the performance of our classification method for circuits of depth-two. According to NFLT, learning algorithms performance is problem-dependent. This changed the view for designing efficient learning algorithms from finding universal classifiers to the study of **when** a specific learning algorithm performs well. Problem dependent parameters and fine tuning of them is unavoidable through the NFLT. Since the NFLT focuses on finding when our method works best, we need to investigate the problem dependent parameters and fine-tuning them. Consequently, we like to investigate the performance of our selected method on a number of well known, hard datasets, where our aim is to find any rules for the problem dependent parameters that can be used as a general guideline on applying this method to new datasets.

Based on the above facts the main research questions of this work that will be investigated are:

- What is the impact of problem dependent parameters on the classification accuracy in the LSA machine?
- Is it possible to formulate paradigms that are helpful for a priori settings of these parameters and thus ensuring a high classification accuracy on new datasets?
- How can we train units located in circuit levels at depths larger than two?
- What is the impact of circuit complexity on the classification accuracy of the LSA machine?
- Does a small increase in circuit depth lead to better classification accuracy, or does it at least provide approximately the same classification accuracy, but with noticeably smaller circuit size compared to depth-two circuits?
- Is it possible to estimate a priori the size of classification circuits that provide a high classification accuracy on hard classification problems?

- If the paradigms for parameter settings are applied to new datasets, how do our experimental results perform if compared to results published in the literature?

In Section 4.9, we describe the methodology used to investigate the above research topics.

1.5 Findings of this Research

A major finding of this work is an upper bound of $8 \cdot \sqrt{2^n/n}$ threshold gates as sufficient for a small error rate, where $n := \log|S_L|$ is the number of input nodes which depends on the size of existing training samples S_L. This is an important finding as it contributes to a hard problem in computer science and allow us to a priori set the network size. Moreover, this finding is a new view in neural network theory that relates the network learning capacity with the number of existing training samples. In circuit complexity lower and upper bounds have been related to the number of input nodes that represent features of the problem, whereas in this work our proposed upper bound is related with the size of existing training samples. This means that if more training data emerge in a problem then a larger network is required. Consequently, this also proves that the number of trained units cannot be constant for all classification problems but is problem dependent, which can be related with a *No Free Lunch* for having constant size of network for all problems. Most researchers in the neural network field work with constant numbers of trained units for all problems, however, this work demonstrates that the learning capacity of a network depends on its available training examples, which should be taken into consideration for applications that require the best performance.

Another important contribution of this work is a new learning algorithm for training next depth layers. The idea of the new learning algorithm is based on generating new patterns from unseen existing ones based on the outcomes of the previous layers. These new generated patterns are then used for training the next depth. The new learning approach introduces the concept of recursively training next depths to adjust the significance of the training quality of the computational units from previous layers.

This work also demonstrates that this classification method is very competitive to existing classification methods. Almost all classification results on hard datasets are at least as good as the highest reported in the literature or even outperform them.

The results of our research have been presented at conferences [14, 15, 17, 229, 230, 231] and have been published in part already in scientific journals

[16, 18, 232]. A first response to our work can be found in [215, 216].

1.6 Thesis Overview

In Chapter 2, the theory behind pattern classification (Section 2.1) is covered, focusing on the neural approach model (Section 2.2.1), the perceptron (Section 2.2.6) and threshold circuits (Section 2.2.11), which are key issues in our work. Various existing pattern classifiers are presented and discussed with learning methods for minimizing the classification error. Complexity issues for threshold circuits are then analysed in Section 2.3. Finally the *No Free Lunch Theory* is presented in Section 2.4 with its consequences in Machine Learning. The theorem is used in Chapter 5 for parameter settings in our classification model.

In Chapter 3, we describe the basics of our combinatorial optimization approach. We describe combinatorial optimization, explaining its application in various type of problems (Section 3.1), the need for efficient algorithms (Section 3.2) for hard problems (Section 3.3), and the existing search methods (Section 3.4) focusing on local search. Details of simulated annealing, our selected combinatorial optimization method, are followed in Section 3.5. A combination of neural networks and simulated annealing called the Boltzmann machine model is also presented in Section 3.5.7. Finally, in Section 3.6 we review the existing relation of combinatorial optimization and pattern classification.

In Chapter 4, we describe the details of the LSA machine. We present the characteristics of the model in Section 4.1. The lines of the learning method, which our approach goes along are presented in Section 4.2. Details of the network topology (Section 4.3), distinguished key features of the LSA machine (Section 4.4) and the pseudocode of the algorithm (Section 4.5) provide a close inside on the LSA machine. In this Chapter we define also the experimental parameters under investigation for their relation with the classification accuracy (Section 4.6, Section 4.7 and Section 4.8). Finally our methodology for the rest of the thesis for investigating the research questions of Section 1.4 is presented in Section 4.9.

In Chapter 5, experimental results on investigations of depth-two circuits are presented for the setting of domain dependent parameters, as suggested by the NFLT. Our choosen datasets are described in Section 5.1. Details of experimental results for the investigated problem dependent parameters are presented from Section 5.2 to Section 5.7. The conclusions from setting *a priori* some of the problem dependent parameters are presented in Section 5.8. Finally our experimental results on the popular datasets are compared

to existing results from the literature in Section 5.9 showing that after fine-tunning of the parameters our approach on some of the datasets outperforms the best reported classification accuracy results in the literature, whereas in all our datasets it achieves classification accuracy at least as good as the reported accuracy in the literature.

In Chapter 6, we investigate the circuit complexity problem for larger depth classifiers. The depth vs size problem is presented in Section 6.1. We introduce a new learning algorithm for the investigation of the depth vs size problem when training larger depths (Section 6.2). We also present the structure of depth-4 circuits (Section 6.3), giving details on the methodology used for the computation of units in this structure (Section 6.4). We provide the experimental results on the circuit complexity problem followed by a detailed analysis for estimating the network size (Section 6.5). The general applicability of the new learning algorithm is tested in Section 6.6. After estimating an important parameter as the required network size for high classification accuracy and in accordance to results from Section 5.8, we provide rules for *a priori parameter* settings for tackling classification problems by the LSA machine (Section 6.7). New datasets described in Section 6.8 are used in Section 6.9 for the evaluation of the suggested *a priori parameter* setting rules, along with a comparison of our experimental results on the new datasets with existing results from the literature showing again that our approach achieves classification rates that are competitive or outperform reported rates.

In Chapter 7, we present the conclusions of this work and we suggest further research from this point.

Chapter 2

Pattern Classification Methods

Systematic research on solving problems in specialized areas such as speech recognition, optical character recognition, signal classification and pattern recognition in Neurobiology formed the foundation of early pattern classification theory in the seventies. Since then, the explosion of computer hardware, the increasing demand on "making decisions" along with the progress in algorithms that could learn, led to an explosion of activity on both the theory of pattern classification and the practical applications of it. Now, pattern classification is an immensely broad subject with innumerable practical applications in every science where decisions are to be made.

Pattern classification is vital in robotics for building machines that need to solve interface decision problems (automated pilots) based on environmental parameters, where the environmental inputs are collected from sensors and decisions are made based on learned experience to correctly interpret the inputs. Interesting applications come from the field of writing such as recognizing handwriting, performing document searching and archiving. Pattern classification is central to human-computer interface problems such as automated speech recognition, optical character recognition, camera images classification and pen-based computing. Face recognition, fingerprint identification, automated speech recognition and optical character recognition are research fields of pattern classification with many applications that need identification based on biological data, like security systems, e-commerce and e-banking applications. Sciences such as Control Theory, Biology, and Communication Technology have provided a large number of classification problems and have benefited from pattern classification. In biology, for example, the *Human Genome Project*, a multi-year international effort to sequence entire human DNA and to determine the biological functions of the genes, provided an im-

portant research field for pattern classification. Well known Biology problems for pattern classification are i) the *protein folding problem*, which seeks for the prediction of the structure of a protein given its amino acid sequence, ii) the *gene recognition problem*, which involves the identification of DNA functional elements with many databases that have been developed for DNA sequence analysis to assist such recognition, and iii) the *protein family classification problem*, which provides an effective method with several desirable impacts on the annotation of newly sequenced genes.

The above are only some of the innumerable applications of pattern classification. Such usefulness of pattern classification makes this field extremely exciting.

2.1 Pattern Classification

Recognizing patterns is a human ability that over the past million of years evolved as a sophisticated feature of the brain systems for categorizing patterns, which has been a crucial task for our survival. It is then natural that we want to build machines that are capable of correctly classifying patterns.

In pattern classification, an object is classified into one of the available classes by using features that describe this object. For example, to classify an image like Figure 2.1 into one of the letters or numerals, we first need to transform the image into features. These features might describe the curvature at some points, the number of holes, or the level of greys in subregions of the image. These features are usually extracted in a preprocessing step or usually are provided in pattern classification as a dataset with known classes called the *training samples* used for training a classifier. The image is then classified into one of the classes according to the values assigned to the features. The task in pattern classification is to train the classifier on a number of training samples and then test the classifier on classifying new data, called the *test samples*, in order to evaluate the *generalization ability*, i.e. the performance of the classifier on new data.

For the classification task a number of competing classification approaches exist in the literature, from statistical models, neural approaches, radial basis functions, tree based classifiers and support vector machines. In *statistical pattern classification* patterns either are expressed in probability densities or probability parameters that command the classification procedure. Pattern classification is a natural way to apply neural network, which is the case in *neural pattern classification*. Other pattern classification models focus on finding logical rules to describe the patterns or to apply Machine Learning approaches [266]. Our focus is on neural classification approaches and, more

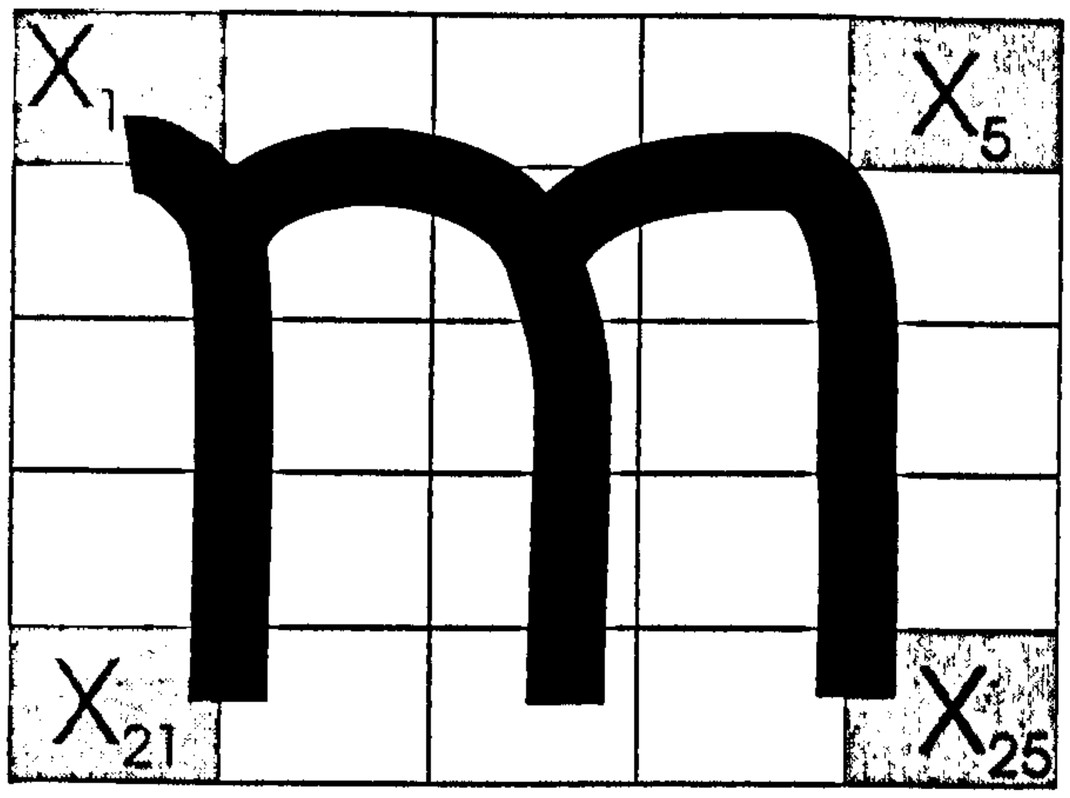


Figure 2.1: Example of Features.

specific, on threshold circuits.

2.1.1 Classification Process

Schürmann [332] defines a pattern as a pair $Pattern = [v, \omega]$ comprising a collection of observations v, which in the physical world is any kind of measurement, and ω is a concept or a meaning of v. The concepts ω comes from a finite set of concepts Ω, which includes all possible meanings.

Pattern classification is the process of inferring meaning from observation or measurements. Designing a pattern classification system is equivalent to establishing a mapping from the measurement space into the space of potential meanings (Figure 2.2). The main approach in pattern classification is concerned with hypothesizing the class of provided input data, then applying techniques that eliminate classification errors, and finally choosing the class that corresponds best according to the classification rules that govern the classifier.

When true classes are known, various names for the classification procedure exist in the literature like *pattern recognition, discrimination,* or *supervised learning*, whereas, when the classes are inferred from the data, the names *clustering* or *unsupervised learning* are used [266]. Henery [169] distinguishes two meanings of pattern classification: a) the *clustering* or *unsupervised* meaning, where given a set of measurements, the aim is to establish the existence of classes or clusters in the data; and b) the *supervised* meaning, where the aim is to establish a rule, whereby one can assign a new measurement to one of the known existing classes. Pattern classification and pattern recognition [358] are terms that are used in the literature almost identical. Other related terms that should be distinguished from pattern classification are *image processing,*

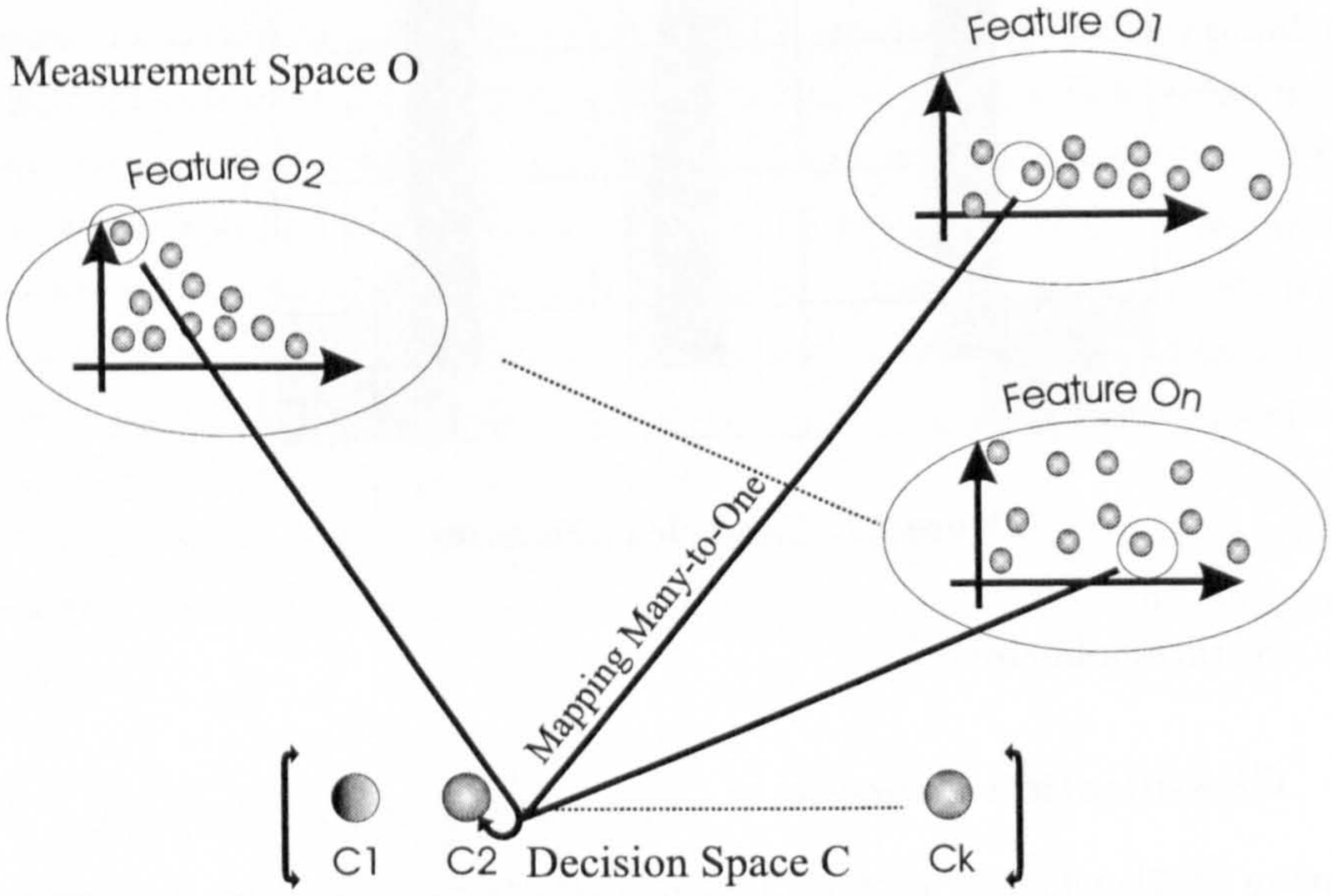


Figure 2.2: Mapping Measurement Space into Decision Space.

regression, *interpolation* and *density estimation*. In image processing, the input pattern and the output are images. In regression, data are tried to be described by a function that relates features between them, and the goal is to predict values for new inputs. In interpolation, we use a variety of interpolation functions (like linear, gaussian or polynomial) to infer the intermediate ranges between a finite number of inputs. In density estimation, the problem is to find the probability that input patterns of a specific class do have particular features.

2.1.2 Supervision and Learning

Three main learning methods are dominant in pattern classification: *supervised learning*, *unsupervised learning* and *reinforcement learning*. In supervised learning, all samples are labelled with their classes. Samples of the training set are iteratively provided to the classifier seeking to reduce the error. In *unsupervised learning* or *clustering* the classifier tries to form clusters or groupings of the input patterns. In *reinforcement learning* or *learning with a critic* the classifier gets only a signal about the correctness or not of the decision but is not guided by any specific feedback to which direction it will improve its decisions like in supervised learning.

In supervised learning the labels indicate the class of the data and therefore the desired response to the features accompany the training data. This is the more common case in classification tasks [266]. Examples of supervised

learning in neural network theory are the perceptron, the multilayer perceptron and radial basis function networks. In pattern classification, the aim of supervised learning is to find out to which pattern class of training data the classifier should most strongly. After training, unseen data provided to the classifier are processed according to the learned activations. Supervised learning will be in the centre of the present work.

On the other hand, unsupervised learning refers to training data that are not accompanied by class labels. Using a probability density function or finding the centre and width of clusters that represent the data, the aim is to model the structure of the measurement space. Example of such neural networks are the Kohonen network [217] and the Hopfield network [183]. In pattern classification, the aim of unsupervised learning is to calculate a set of class probabilities and determine subsets of the training data as points for centres of clusters. Hopfield suggested an important technique for analysing neural networks by viewing them as minimizing an energy function. He associated the minimization of an energy function E with interconnections in a network that uses threshold logic units that changes in time. If w_{ij} is the connection strength between the threshold logic unit i, j, and if $f(i)$ and $f(j)$ are the activity of the i, j neurons, then the energy function E is given by:

$$E = -\frac{1}{2}\sum\sum_{i \neq j} w_{ij} f(i) f(j). \tag{2.1}$$

The network evolves in time by a process called *asynchronous updating*, where only one randomly selected neuron at a time is changing its state according to disagreement with the input state. Changes in the state aim at decreasing the energy of the network. Hopfield networks have the problem of *trapping in local minima*: if a local minimum is found, then no further changes in the energy are possible.

2.1.3 The Supervised Learning Model

The basic supervised learning classification model can be partitioned into components. These components are described in the next subsections.

Preprocessing of data

Preprocessing of data involves procedures that simplify the input data without losing relevant information, and wherever possible reduces the input noise. All real world decision problems involve noise in some way. Apart from the noise problem that might be difficult for a classifier to cope with, the problem of *missing values* has no trivial answer. *Missing values* may occur because for

some samples measurements of some feature values were not possible. Dealing with the missing values problem is not always easy: we may discard the samples with missing values from our training set, but if the number of samples is important we may have to find a way to make the best classification using only available feature values. Substituting the missing values with zeros, or with the average of the values for the rest of the patterns, or with some other values may effect the performance quality.

Data Representation

This involves the selection of the data type used for the input patterns. A feature vector $\vec{x}$ describes the properties of a pattern in the N-feature space. Classes form regions in the N-feature space and our classification task is to find boundaries of the class regions called *decision boundaries.* In a two-dimensional space the decision boundaries might be a simple straight line, whereas for extremely complicated problems the decision boundaries are more complex than a straight line. A good data representation enables the classifier to naturally reveal the structural relationships among the patterns. Usually patterns are presented as binary vectors, or vectors of integers or even vectors of real numbers. Data might also be represented by a finite alphabet of attribute descriptions or characteristics of the features, like colour, shape, smell etc. These types of data are usually transformed into vector representations of binary, integer or real type. It is not always obvious how the transformation should be executed, as the way that we quantify "near" and "far apart" will determine the success of our classifier. Sometimes there are types of data that are not appropriate for a type of classifiers. For instance non-metric data are naturally described in decision trees but they are not appropriate in neural classifiers. The process of transforming an existing data representation into an equivalent form suitable for a classifier is called *encoding.*

Feature Extraction

Feature extraction is closely related to data reducibility where certain properties or features are selected as being more important than others for the classification task. Features are used for separating our patterns into classes. Therefore, the problem is to partition the feature N-space into subspaces of fewer features that perform equally or even better than the N-space features. The main questions in feature extraction are: Which features work best? Which features are redundant?

Some features might be important for the class separation, other might provide little improvement, other might be too expensive to measure, and

others might even degrade the performance. What actually is needed in practical applications are quick classifiers with good classification rates. Smaller numbers of features hopefully might lead to simpler decision boundaries and faster classifiers, as they are considerably easier in training, and less sensitive to noise.

Classifier Parameters

The design phase involves the selection of an appropriate classification model that executes the classification task. The main problem in the design phase is the *computational complexity* issue. Usually, the computational complexity of a classifier is related to the algorithmic complexity that scales as a function of the number of feature dimensions, the number of patterns, the number of classes, and a number of other parameters that all together increase the difficulty of the classification problem. Designing a classifier is an optimization procedure of finding the optimum parameters under constraints defined by the problem (for example, the number of classes) that implies a realistic computational complexity for solving the classification problem.

Define Training and Testing Data

Learning to correctly classifying a pattern might be interesting, however, we want from our classifier to accurately separate the different classes from a number of samples which can be either the whole set of available samples, or a subset of selected samples used for training the classifier called *training samples.* Subsets of selected samples that have not been used during the learning phase and are kept as unseen patterns for testing the classifier are called *testing samples.* The number of samples used for training the classifier to learn the pattern classes and the number of samples that will be used as the unseen patterns that will test the classifier's performance is a decision that may affect the overall performance of the classifier.

The Costs

Usually, decisions taken by the classifier have a cost measurement, which is the consequence of the decisions taken by the classifier. Cost minimization of misclassified training and testing samples is of major importance in pattern classification.

Training and Learning

The process of using existing samples from a specific domain to determine the classifier is called *training*. Training is an important task as it is closely related to the performance of the classifier. The most effective method for training involves a learning procedure from example patterns. Most of the time is spent on the learning task, which is an iterative procedure. Special algorithms are designed for learning the training patterns, i.e. reducing the errors on the training patterns, and estimating the parameters of the classifier. These algorithms are called *learning algorithms*. Many learning algorithms have been designed and developed for many classifiers since the perceptron was introduced.

Training protocols in supervised learning determine how the patterns will be presented to the classifier and when learning occurs. If patterns are randomly chosen and used for learning, then *stochastic training* occurs. If all patterns are presented before learning takes place, then *batch training* occurs. Finally, when patterns are presented one each time without storing them then *online training* occurs. Because training is an iterative procedure, we make several passes through the training data. The single presentation of all patterns in the training set during the training phase is called *epoch*, and we create several epochs during training. In some applications, apart of the training and testing sets, a *validation set* of independent samples is kept separate from the training data in order to use it as the stopping criterion of the iterative learning procedure. Training is stopped when the error on the validation set stops decreasing, i.e. when the error on the validation set reaches the first local minimum.

The learning task is to minimize the error on the training examples, which is called the *training performance* of the classifier. As we will see, high training performance does not guarantee the maximum performance of a classifier due to overlearning, a memorization process where the classifier is over tuned with respect to the training examples. *Overfitting* is a similar term in literature describing this process. Unfortunately, there is no universal training method that is capable to learn all example patterns of any domain.

Generalization

When a classifier takes decisions on new unseen patterns, we say that the classifier *generalizes*, and the process is called *generalization*. Usually, good generalization is provided by simple decision boundaries, whereas complex decision boundaries provide poorer generalization.

Performance

The most common measure of classifier performance is the classification error rate, i.e. the percentage of new patterns that have been misclassified in the test set. If ϵ is the error rate then the corresponding *classification accuracy* α is:

$$\alpha = 1 - \epsilon. \tag{2.2}$$

Minimization of the error-rate, and consequently maximisation of the classification accuracy, is the main goal in pattern classification. Both the error rate and the classification performance refer to the *generalization performance* ability, i.e. to use the prior knowledge acquired from the training set to the unseen test set. The generalization performance is not always proportional to the training performance. Suppose a classificr is learning all cases in the training set, which is equivalent with minimization of the error on the misclassified patterns. If the training set consists of noisy data, then the training performance has also learned wrong patterns and therefore the generalization performance decreases. This problem called the *overlearning* problem is a common trap in classifiers. Our efforts in trying to end up with a model that accurately fits the fine details of the training set might or might not cause overlearning, according to the type of data (noisy or not) and the specific type of classification problems (real world problem or artificial). Usually, we might be more satisfied with poorer performance on the training samples if this means that our classifier will have better performance on new patterns. However, the problems of choosing between complex and simpler decision boundaries and predicting how well our model generalizes are central problems in pattern classification. An optimal trade-off between good performance on the training set and simplicity of the classifier is giving the highest accuracy on new patterns [106]. So we need to adjust the complexity of the classifier so that it will be not too simple and cannot explain the differences between the classes, and that it will be not too complex to give poor generalization performance.

Unfortunately, in the light of the *No Free Lunch Theorems*, there exists no general-purpose pattern classification device tackling a wide variety of problems. Pattern classification depends very much on the specific type of the classification problem. Performance on real world pattern classification problems generally requires exploiting domain-specific knowledge.

2.1.4 Models of Classifiers

A number of different classifiers exist in the literature. *Bayesian models* are designed when the probability structure of the underlying classes is perfectly known. This is considered an ideal case but rarely occurs in real world problems. When the full probability structure is not known, but some general form of distribution is known, various models called *parametric models* try to estimate parameters that represent the uncertainty about a probability distribution. *Non-parametric* models are free of any kind of probability density and are based only on information provided by training samples alone. Types of non-parametric classifiers include *nearest neighbour classifiers*, where the information of "neighbouring" samples is used for classifying a current sample; *linear and non-linear discriminants* like the perceptron and multilayer neural networks, where the classifier can be viewed as a *machine* that computes a number of discriminant functions resulting in class partitions; *stochastic classifiers* like Boltzmann machines, *tree-based classifiers*, *rule based* classifiers, *fuzzy classifiers*, *kernel classifiers*, and *support vector machines* are some but not all of the variety of existing classifiers described in the literature. These classifiers are using a variety of methods mainly from two different fields a) Statistics with *statistical* and *probability methods* b) from Machine Learning, like fuzzy methods, tree-base methods, neural network methods, and kernel methods. These models will be briefly described in the next subsections, while neural networks, linear and non-linear classifiers will be extensively described in Section 2.2, as they are in the focus of our work.

2.1.5 Statistical Models

Statistical approaches are rather concerned with the probability of a case being in each of the classes than simply obtaining its classification. An explicit underlying probability model is used for this purpose. The approach is widely spread in the statistical community and many probabilistic models have been proposed from the early derivatives of Fisher's model [117] to more complex systems. Roughly, if a classifier is used under a probabilistic system then it is called a *Bayes classifier*, where it is assumed that the probability of a particular pattern to belong to a given classification is known, or else the distribution of the possible inputs is known.

Bayesian decision theory is the main statistical approach in pattern classification. Classification in the statistical approach is based on known probabilities of various classification decisions and the accompanied costs of such decisions. If $P(x|\omega_i)$ is the probability of a sample x being in class $\omega_i \in \Omega = \{\omega_1, ..., \omega_n\}$ then the *Bayesian decision rule* decides:

$$Class = \omega_m, \text{ if } \forall\, \omega_i \in \Omega,\ P(x|\omega_m) > P(x|\omega_i). \tag{2.3}$$

and the Bayesian decision rule is used to minimise the probability of error $P(\epsilon|x)$ for a particular x:

$$P(\epsilon|x) = min[P(x|\omega_1), ..., P(x|\omega_n)]. \tag{2.4}$$

As it is very difficult to have the above posterior probabilities, the *Bayes formula* allows to calculate them from the prior known probabilities $P(\omega_i)$ of the ω_i class, where $\omega_i \in \Omega = \{\omega_1, ..., \omega_n\}$ (also $\sum_{j=1}^{n} P(\omega_j) = 1$) and the *class conditional probability density function* $P(x|\omega_i)$, which is the probability density function for x being in class ω_i. Then the Bayes formula calculates the posterior $P(\omega_i|x)$ probability of selecting class ω_i for feature x by the equation:

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)}. \tag{2.5}$$

where

$$P(x) = \sum_{j=1}^{n} P(x|\omega_j)P(\omega_j). \tag{2.6}$$

The above equation can be used for features with continuous values, whereas for features with discrete values the probability density function can be substituted by the corresponding probabilities. A *Bayes classifier* calculates for any input x the corresponding Bayes formula and assigns the class with the maximum probability $P(\omega_i|x)$. So the structure of a Bayes classifier is determined by the prior probabilities $P(\omega_j)$ and the conditional densities $P(x|\omega_j)$. If we know the full probability structure of a problem, then we can construct an *optimal Bayes decision rule.* When the probability structure is known, then the Bayesian decision rule can also be used for solving the missing value problem [9, 313]. Bayesian models have been in the focus of research over many years [46, 47, 106, 124, 236].

The main problem in many pattern classification problems is that the conditional densities are not known. When some form of probabilities may be assumed, then other models can be used, called *parametric models*, where characteristic probability parameters like *means* and *covariance matrices* are calculated from training data. *Hidden Markov models*, introduced by Baum and Petrie [42], are using Bayesian parameter estimations along with *transition probabilities* of being in state s_{t+1} at time $t+1$ from state s at time t. These models are usually used when sequences of decisions are needed as data arrive in time (for example recognizing phonemes in speech). It consists of nodes

representing hidden states, while interconnected links represent the conditional probabilities of a transition between the states. The transition probabilities can be learned iteratively from sample sequences and classification proceeds by finding the most likely model.

Even when no information about the underlying densities is known, then statistical *non-parametric methods* can be used with arbitrary distributions for estimating the density functions $P(x|\omega_j)$ from pattern samples. Probabilistic neural networks [347] are such non-parametric methods, where the activation function represent *Parzen windows* [282], which are functions for density estimations from training samples and localized basis functions such as Gaussians.

2.1.6 Nearest Neighbour Classifiers

Pattern classification usually makes an assumption about the statistical structure of the world. It is often assumed that nearby patterns are likely to have the same classification. An exception from this assumption is the *XOR* problem described in Section 2.2.6 and the *parity problem*, where the allowable number of "ones" in a binary vector is even or odd. Figure 2.3 demonstrates the parity problem for a 3-bit binary vector $\vec{b}$. Such problems are called *complex* and are likely to be hard for a classification model to be learned.

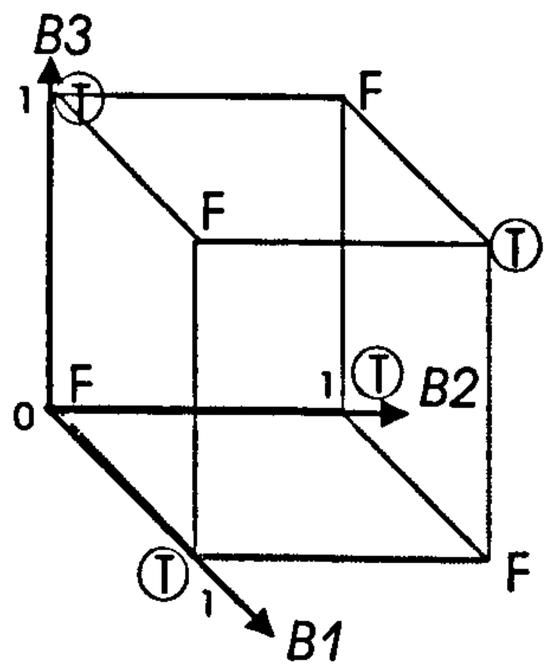


Figure 2.3: The Parity Separability Problem.

Cover and Hart [88] introduced in 1967 the "nearest neighbour pattern classification". In nearest neighbours classifiers, the assumption that nearby state points are likely to have the same classification is used. If a new pattern is to be classified, then the distance to existing patterns is calculated and the class is assigned to the new pattern to be the same with the class of the *nearest neighbour*, i.e. the closest pattern. A variation, where instead of assigning the class of the closest pattern the class of the majority of k-patterns is assigned, is called the *k-nearest neighbour classifier* and was explored by Patrick and Fisher

[283]. The weakness of the nearest neighbour classifiers lies in the boundaries of the region where uncertainty increases due to noisy patterns or strangely shaped classification regions. The algorithm is very slow as more patterns are learned, i.e. more distances between new patterns and old ones have to be computed. The latter problem might become less serious if a region is approximated by patterns lying somewhere in the middle of the region, called the *prototype patterns*. Unfortunately, how well this method works depends on the shape of the region. Detailed reference on nearest neighbour classification techniques can be found in [94].

2.1.7 Support Vector Machines

Support vector machines (SVMs) [58] are based on kernel methods and became popular as they are particularly competitive in problems where the input data are not giving any suggestion of the importance degree of its feature values. The input is mapped by a nonlinear function to a high-dimensional space, and the optimal hyperplane is the one that has the largest margin. The term *support vectors* is related to pattern vectors that represent the margin. Usually, patterns close to the margins are hard to classify and contain the most information needed for designing the classifier. Due to the close relation to finding separation margins, SVM are also called *margin classifiers*.

By using an appropriate mapping to a sufficiently high dimension, we can always have a hyperplane separating data from two classes. This method can solve the *XOR* problem as we will also see for multilayer networks in Section 2.2.6, where a similar idea of extending the inputs to a higher dimension holds. In SVMs, Gausian or polynomial or other basis functions are chosen to be the non-linear functions that map the input to a higher dimensional space. Support vector machine training is based on a modification of the perceptron learning algorithm (see Section 2.2.6). In small classification problems, where the search through the entire training set is feasible, the *worst classified patterns* are searched for. The modification to the perceptron learning algorithm is that these patterns are used to determine the proportion for updating the weights, instead of a randomly selected misclassified pattern as used in perceptron learning algorithms. For larger problems, the learning method tries to find the optimum margin by maximising the smallest possible *margin value* under the constraints that all patterns are well classified and that the weight vector is normalized.

SVMs employ also some multilayer networks ideas. Subsequently it is natural that SVMs and multilayer neural networks are competing in hard classification problems. The main idea is to construct the optimal separation

hyperplane, based on calculating a kernel function of the support vectors and applying a linear hyperplane with a maximum separation boundary. The architecture of SVMs consists of the support vectors, where the dot products of the support vectors and the input pattern is computed, and the result is combined with kernel weights to provide the output of the network.

The idea of SVMs is rather new and first appeared in [58, 327]. Extensive presentation of SVM methods can be found in [76, 78, 328, 339, 344, 371]. SVM gain their popularity by the advantage that the number of support vectors characterizes the complexity of the resulting classifier. Thus, the expected upper bound of the error rate linearly depends on the number of support vectors only. The approach is also less sensitive to the overfitting problem.

2.1.8 Fuzzy Classifiers

Fuzzy classification methods employ heuristics to find conjuction rules between features operating as discriminant functions. A set of fuzzy rules is defined for each class. For classification, the degree of membership of the input sample is calculated and the sample is classified into the class associated with the fuzzy rule according to the highest degree of membership. Shapes like hyperbox regions, ellipsoidal regions, or polyhedral regions are used for approximating class regions in fuzzy classifiers. A membership function is associated with the fuzzy regions. The problem in fuzzy classifiers is to determine the number of fuzzy rules that must be generated to realize sufficient recognition rates for both training and testing samples. A common method in fuzzy methods is generating iteratively fuzzy rules by using all or part of the training data forming regions that include a class until no training data remains to generate rules. If the generated regions are overlapping between them, then tuning of the fuzzy rules solves the overlapping problem. Tuning of the fuzzy rules is achieved by optimization of an objective function related to the improvement of the recognition rate.

Fuzzy methods, originated from the mid-sixties [401], can be used when there are few training data or even in absence of training data. Then main advantage of fuzzy classification is that knowledge in a syntactic form is transformed into discriminant functions. Limitations on fuzzy systems are that they are slow and inefficient when complex problems or problems with high numbers of features are used. Whenever the performance of a fuzzy system is unacceptable then traditionally practitioners employ neural methods leading to a combination of *neuro-fuzzy systems.* A reference for fuzzy techniques in pattern classification can be found in [204], whereas Abe [5] gives an in-depth analysis of neuro-fuzzy methods in pattern classification.

2.1.9 Decision Trees

Decision trees are classifiers where a pattern is classified through a graph of a directed tree. At each node, a decision is made according to different possible values of the features of a pattern, reaching at the final node, which is the corresponding class where no further choice is possible. Decision trees are build from training data by progressively splitting the training samples into smaller and smaller subsets according to the properties of features that can be used to discriminate patterns. A simple recursive algorithm for creating a decision tree is known as the CART decision tree. CART grows recursively a decision tree by the following process: If the node cannot find a property to split the data presented into subsets, then declare a category for this set of data, otherwise split the data into subsets. CART uses binary trees, while other decision tree methods like Quinlan's ID3 [293] and the successor C4.5 [294] have a branching factor $B_i, B_i \geq 2$, coming out from a node i. Most of the time in building a decision tree is spend on deciding which property should be analysed at each node. The property that leads to the simplest model that explains the data is favoured. The most popular measure for this simplicity is the *entropy impurity* factor that defines how impure the data from node N are reaching descent nodes for a property T.

$$i(N) = -\sum_j F(c_j) \log_2 F(c_j), \tag{2.7}$$

where $F(c_j)$ is the fraction of patterns at node N that are in category c_j for a binary tree. The task is to find a property T that minimizes the impurity entropy, i.e. the goal is to maximise the drop of impurity entropy, which is

$$\Delta i(N) = i(N) - F_L i(N_L) - (1 - F_L) i(N_R), \tag{2.8}$$

where N_L and N_R are the left and right descendent nodes, respectively, $i(N_L)$ and $i(N_R)$ their entropy impurity, and F_L are the patterns at node N that will go to N_L for the specific chosen feature property T. For properties where the data are discrete instance descriptions, called *nominal data*, like colour={red, blue, green, yellow}, it is easy to split the data according to each of the instance descriptions. We may need to exhaustively search over all possible subsets of the training sets to find the rule that maximizes (2.8). More complicated is splitting the data when the feature property has numeric values. Optimization methods like gradient descent or combinatorial optimization may apply in this case.

Tree based classifiers have the benefit that the classification result constitutes a tree, which can interpret the patterns as logical expressions. Therefore,

without prior knowledge of the relations between the feature properties, the decision tree gives meaningful relations. Once the decision tree is built, classification of new patterns is rapid. Another important benefit of decision trees is that they are particularly useful for nominal data, whereas for other classification methods nominal data need preprocessing steps to incorporate them into the problem. On the other hand, although decision trees yield comparable or better results with other classification methods for many problems, they seem to be poor in inferring simple concepts like solving the parity problem [106]. Readings on tree-based classification methods can be found in [63, 270].

2.1.10 Rule-based Classifiers

Rule-based methods use prepositional logic or first order logic to describe a category based on relationships among entities. Rules are of the *if-then* type like:

IF MALE(x) AND FEMALE(y) AND FATHER(x,y) THEN DAUGHTER(y)

Rule-based systems have formed in artificial intelligence the foundations of expert systems. Their use, however, in pattern classification has been modest and is mainly based on Michalski's work [264]. The design of learning rules must specify the predicates and the functions based on prior knowledge of the problem domain, like *expert knowledge*, which is the main drawback of this method for real world pattern classification problems that lack such expert knowledge. Artificial Intelligence textbooks like [195, 385] provide a classical overview of learning rule methods for classification, including also the first rule-based system DENDRAL [66] for inferring chemical structures from mass spectra and MYCIN for medical diagnosis [341].

2.2 Neural Networks and Threshold Circuits

2.2.1 Neural Networks

Neural networks are inspired by the human brain analogy. The brain is composed of a very large number of *neurons*. Signals between the interconnected neurons are propagated by neuron activation. Each neuron is activated (or fires) if the received signal exceeds a certain value called *threshold*. *Artificial neural networks (ANNs)*, or *connectionist models* [114], or simply neural networks are computational models that mimic the brain behaviour. ANNs are using the same representation of neurons, interconnection links, and computational units for calculating the inner product between the input of the neuron

and the *connection strengths* (or *weights*) and compare it with a threshold value to determine whether the neuron is firing or not. Since these models are emphasizing connections between the neurons (also called *units*), they are also called *connectionist models.* A neuron is considered to have memory as information is stored in connection strength and can be distributed over the network.

The research that investigates computational methods aiming at achieving human-like performance in computer systems via dense interconnection of very simple processing elements is called *neural computing.* The ANN model is described as to collect all inputs multiplied by the weight factors, to use a function for altering the weights, and to apply an output function that can be a *linear, sigmoid* or *threshold function* in order to determine the activation neuron. ANNs consist of layers of interconnected neurons, called *network nodes.*

The greatest potential of neural networks is that they work towards the general belief that massive parallelism is essential for high performance and fault tolerance. Moreover, the fascination of having a learning capability used for training data (that many neural network models incorporate) provides a robustness property for a powerful computational model that exploits massive parallelism in a natural way. Devijver and Kittler [98] express that for pattern classification neural networks seem to be a panacea, as other computational models have been a major limitation.

There exist a large number of textbooks and papers on neural networks [24, 160, 163, 171, 196, 301], whereas mathematical aspects for neural networks are in [27, 144]. As neural networks are networks of perceptrons, the history of neural networks is closely related to the origins of Rosenblatt's perceptron [310] presented in Section 2.2.6.

2.2.2 A Neuron Model

A neuron can be seen as a directed graph with units situated in the vertices. These units can be either *input units* or *computational units* or *output units* (Figure 2.4).

Input units: In some models, for each input feature x_i there is an associated neuron that accepts the input value x_i and produces the same input x_i as output of the neuron input unit. In most models, the input units are represented as directed arcs carrying the input value. The latter will be used in this work.

Computational units: These units are usually called neurons and are represented as a circle or a parallelogram where the input arcs are terminated

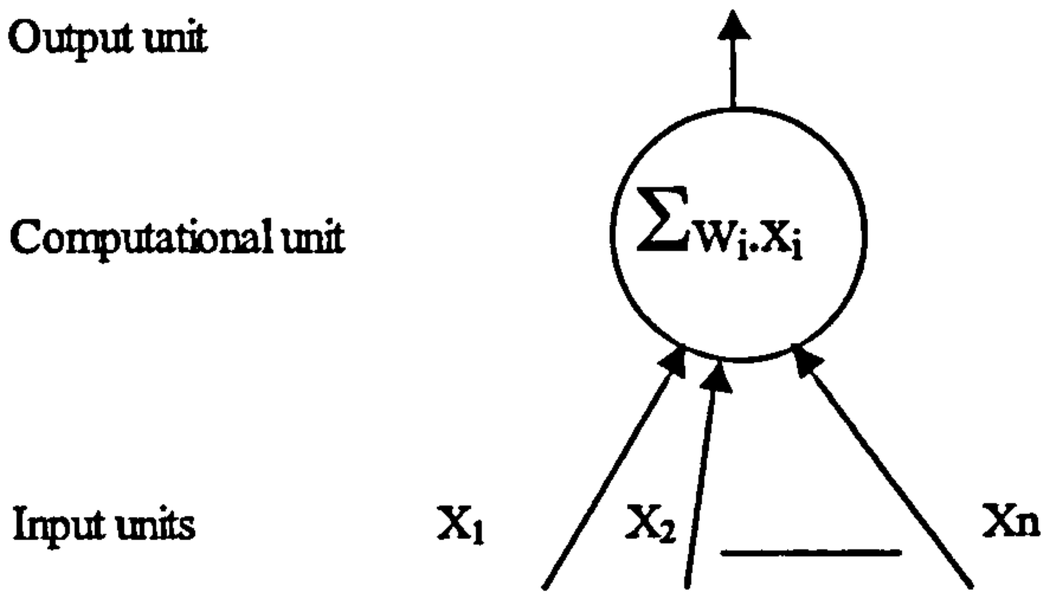


Figure 2.4: Model of Neuron.

and where the inner product is calculated and compared with an activation function. Activation units that are associated with computational units and determine the output of the network may also be found in literature.

Output units: ANNs usually have one output unit that represents the output of the network. Output units can also be found after each layer, however as they are input units to other computational units they will be considered as input units.

In our work, when we refer to neurons, we refer to computational units. For a number of n inputs the configuration state of the neuron α is described by the *weight* vector $\vec{w} \in \Re^n, \vec{w} = w_1, ...w_n$, and the activation function f_α.

The type of activation function determines whether a neuron will have linear threshold units, sigmoid units, or polynomial units.

For linear threshold units, which are in the focus of our work, the activation function is described by the equation

$$f_\alpha = \sum_{i=1}^{n} w_i \cdot x_i \geq \theta, \tag{2.9}$$

where θ is the threshold. The neuron outputs 1, if the inner product is at least equal to θ, otherwise it outputs 0.

For sigmoid units, the activation function is calculated by $f_\alpha = \frac{1}{1+e^{-a}}$. The behaviour of the sigmoid units is based on the fact that if the inner product is much larger than the threshold θ it outputs 1, and if it is much less than θ, it outputs 0. The values inbetween are smoothed by the activation function.

For polynomial threshold units of degree r we have instead of the linear combination $w_i \cdot x_i$ of the inputs to the unit the computation of polynomial activation functions. For example, if $r = 3$ we can imagine a unit which

computes the quadratic expression:

$$f_\alpha = w_1x_1+w_2x_2+w_3x_3+w_4x_1^2+w_5x_2^2+w_6x_3^2+w_7x_1x_2+w_8x_1x_3+w_9x_2x_3 \geq \theta. \tag{2.10}$$

Of course, if $r = 1$ we obtain again a linear threshold unit.

2.2.3 Problem Solving and Neural Networks

Although ANNs can be attached to almost any kind of computational problems, we shortly describe here three main categories where ANNs are successfully applied: pattern classification, combinatorial optimization, and time-series prediction.

The classification problem can be formalized as a pair (O, C) where O denotes a finite set of objects $O = \{o_1, ..., o_n\}$, the *input patterns*, and C denotes a collection of disjoint subsets $c_1, ..., c_m$ of C, the *classes*. The problem is to classify automatically a given object $o_i \in O$ as a member of one of the subsets $c_j \subset C$ [2]. Simple classification problems such as pattern classification can be easily solved (recognized) by human brain but are very hard for computers to solve [400]. Neural networks, in general belief, are the most suitable computational models that can be used for classification problems [172]. Humans can recognize objects by perception. The neurons of the brain receive external stimuli from the visual system and adjust their connection state according to the external stimuli and according to the current information stored in the strengths of the neuron connections [164].

Neural networks can classify objects in a similar way as the human brain. Given a connection network and the connection strengths, external stimuli can be considered to clamp the states of a subset of units and, therefore, the objective is either to adjust the remaining free units in such a way that the most probable configuration that recognizes most of the examples of the data set is achieved. Usually, in classification problems there exists a pre-processing step that concerns with noise and data reduction.

Pattern classification is a major field for ANNs. Pattern classification problems have the objective to recognize an object as a member of a given subset. Therefore classification problems can be viewed as trying to find a configuration of a neural network with an appropriate structure and appropriate connection strength such that it can correctly classify all objects of a given subset. Pattern classification is described in details in Section 2.1.

For a given set of input patterns, ANN learning tries to minimize a suitable error function. Combinatorial optimization algorithms can be used for this purpose. The training patterns are presented iteratively to the computational

unit with successive adjustment of the weights and backpropagation of the error [315]. When the error rate approaches an acceptable level, the weights are fixed and ready to be used for classifying unseen patterns. The ability of the network to correctly classify the *test patterns* is called *generalization performance* or *classification accuracy* of the network (cf. Section 2.1.3).

ANNs can be used to attack *combinatorial optimization problems* for finding good solutions [184]. This brought a new perspective to search methods. Unlike other combinatorial optimization methods, the ANN approach does not explore the different possible configurations and acts more as a statistical interpretation of the results [287]. The aim of ANN from a search perspective is to optimise a given global quantity or function by stabilising the connection strengths of the units. Combinatorial optimization problems can be viewed as assigning a neural network to each instance of a given combinatorial problem. Then each configuration corresponds to a solution of the optimization problem. Solving the combinatorial optimization problem is equivalent to choosing an appropriate structure with appropriate connection strengths where the data set can be applied producing a maximum consensus to the cost function. Applying neural networks to the most common combinatorial optimization problem namely, the travelling salesman problem (TSP, see Section 3.1.2), Peterson and Söderberg [287] found that the performance is comparable with the performance of simulated annealing for TSP. Anderson [24] states that an adaptive feedforward neural network can be viewed as an approximation device, because it approximates an output function as the weighted sum of individual functions of the input patterns. Combinatorial Optimisation is addressed in more details in Chapter 3.

Time-series prediction is another type of problems for ANNs, where the aim is to predict future values of a series given previous values [378]. This can be done by replacing the output unit that has threshold behaviour by an output unit representing a real value obtained by a computational unit that calculates a linear transfer function:

$$x(t) = F(x_{t-1}, x_{t-2}, ...), \tag{2.11}$$

where $x(t)$ is the real value of the output unit and $x_{t-1}, x_{t-2}, ...$ are the values of the series at previous times.

As described above, ANNs are not limited to the three types of problems only, as they can be applied to almost any kind of computational problems. Even for problems like natural language processing, where representing the structure of a parse tree seems to be hard to be carried out by neural networks, there has been research on special types of neural networks [154, 226, 227] that

successfully represent and solve the parsing task.

ANNs have been very popular among researchers due to successful applications to a large number of problem domains from different scientific areas. The power of ANNs comes from:

- the potential of modelling a large variety of problems, as models may vary from a simple neuron to very complex structures representing sophisticated functions;
- the easy training methods that allow a neuron to learn a set of data;
- the successful uncovering of hidden knowledge of the data;
- the applicability to a variety of problems such as prediction, optimization, classification or control;
- the highly parallel process as neurons continuously evaluate their output according to their input for activating inputs of other interconnected neurons.

Three design issues are considered when developing neural networks for solving any type of problems [2]. The issues are related to:

- the number of units needed;
- the way that the units should be interconnected;
- the way that the connection strengths should be chosen.

These issues, along with other aspects, determine the type of the neural network used for solving a problem. Such taxonomy and models are presented in the next section.

2.2.4 Taxonomy of ANN

With respect to a number of various aspects we consider the following taxonomy of neural networks models (the referred models are presented in Section 2.2.5):

Aspects of dynamics classify ANNs into synchronous and asynchronous ANN models: This classification is based on the different dynamics of the models. Synchronous models, which our work focuses on, are based on simple neurons with weighted connections, the *synapses*, while asynchronous models include mimics of biological models with complex dynamics and activation of neurons by more than one event at each time step.

Aspects of topology classify ANNs into models of changing and unchanging connectivity. According to whether the network connectivity topology changes during execution we have two classes. Our work focuses on ANNs with unchanging connectivity.

Aspects of loops in information propagation classify ANNs into feedforward and recurrent ANNs: According to whether there exist loops in the propagation of information or not, we classify the ANNs into feed-forward or recurrent nets. In feed-forward networks the signal is propagated from the input units (neurons) to the output of the network without any loop. In recurrent (or also feedback) networks information go both ways down and up. Feedforward networks are mainly used in pattern classification. The major representative of this class is the perceptron (see Section 3.5.7). Recurrent networks, where the activation of the neuron continues until a steady state is reached, are used in solving combinatorial optimization problems. Boltzmann machines (see 3.5.7), which incorporate simulated annealing methods (see Section 3.5), is an example for recurrent ANN.

Aspects of the learning procedure used, classify ANN models into supervised and unsupervised learning models: Supervised learning, as we have seen in Section 2.1.2 and Section 2.1.3, uses examples to learn and a teacher to return the feedback of the learning performance. This method is based on an error function, which is determined at the output of the network by comparing the output result with the desired activation. This difference forms the error vector, which is back-propagated to the predecessors to locally adjust their error. This learning procedure is also known as the *backpropagation learning rule* [315]. Examples of supervised learning are the perceptron model and the backpropagation model.

Unsupervised learning is usually used in problems where there is no prior knowledge of the desired output. ANNs using this learning method are also known as *self-organizing nets* due to absence of the desired output, i.e. they self-adjust their weights in a competitive constraint environment. Example of unsupervised learning are Kohonen networks and Boltzmann machines.

The above is a taxonomy commonly referred to in the literature. Furthermore, other aspects found in the literature may be used for taxonomy:

- aspects of data type classify ANN models into binary, rational and continuous value models. Well known models, where the states of the units may be restricted to binary values, are the *perceptron model* [311], the *Boltzmann machine* [173] and the *Hopfield model* [183].

- aspects of probability in search strategy classify ANN models into deterministic (like the perceptron and the Hopfield model) and probabilistic

models (like the Boltzmann machine).

- aspect of direction of connectivity classify ANN models into unidirectional models, where information flows in one direction (perceptron) and bidirectional models, where units influence each other in both ways (Boltzmann machine, Hopfield networks).

- aspects of network depth classify ANN models into models with hidden layers (perceptron, Boltzmann machines) or without hidden layers.

- aspects of the activation function classify ANNs into linear threshold networks (perceptron) and sigmoid networks.

According to the above taxonomy, when referring to an ANN, we also have to define a number of related aspects that can make more specific how the particular ANN is used in an application.

2.2.5 Neural Network Models

There exists a large number of ANN models in the literature, in accordance with differentiations and combinations of the aspects referred to in the previous section. The models most related to our work (threshold circuits, perceptron and multiplayer perceptron) will be analysed later in more details.

A number of famous models are listed in the current section. Some of the previous mentioned classification aspects are outlined, so that a general view on the arsenal of neural network models is stated.

The Perceptron Model

Rosenblatt's perceptron [311] is historically the first classifier that uses a network of binary threshold units connected by unidirectional links. In Section 2.2.6 we have a closer look at the perceptron.

Multilayer Perceptrons (MLP)

Multilayer perceptrons are supervised feed-forward networks, closely related to threshold circuits. MLP have most of the common ANN characteristics. In classification problems, MLP networks are defined by their weights and thresholds, which together give the equation of the separating hyperplanes.

Threshold Circuits

Threshold circuits are feed-forward linear networks trained with supervised learning. In Section 2.2.11 this model will be studied in more details.

Back Propagated Delta Rule Networks (BPDR)

Back propagated ANNs are synchronous feed forward networks that use a backpropagation learning rule. The back propagation algorithm performs gradient descent on the error surface by moving towards a minimum with a step proportional to the learning rate. Rumelhart et al. [315] made this model popular and called their algorithm generalized delta rule. The idea is based on the well-known supervised learning approach having added extra hidden layers. This network type is used in classification problems, time-series forecasting, and modelling problems.

Radial Basis Function Network (RBF)

RBF are also feedforward supervised learning networks, which are configured usually with hidden layers to model non-linear functions. The activation function is selected among a number of functions called basis functions. RBF networks are much faster than BPDRs and are similar to probabilistic networks. A hidden layer performs weight adjustments, which are set before the actual adjustment of first layer weights occurs. Compared to MLP networks, the main difference is in the way which hidden units adjust their weights. In MLPs the inner product of combining the values is used, while in RBFs, Euclidean distance is used to combine the input vector with the weight vector. Therefore, in RBF networks, the function that is to be approximated presents a radial symmetry being only a function of distance between learned and input patters. The Euclidean distance can be interpolated either by Gaussian, or linear, or polynomial functions. The data separation space is divided by using circles, instead of hyperplanes used in MLPs.

Kohonen Networks

Kohonen networks [217] use unsupervised learning and belong to feedforward networks. By using self-organising processes, it modifies the connection strengths according to the input patters. The most active unit is detected when a new pattern is presented and its connection weights along with that of close neighbours are adjusted to be closer to the input pattern.

Hopfield Networks (HN)

Hopfield networks [183] are networks that have neurons fully connected to each other with allowable output states being binary values of [-1,+1]. HNs perform error correction and energy minimization and therefore can be considered as networks for combinatorial optimization. The HN is evolving in time by a

process called asynchronous updating. In this iterative process, a neuron is randomly chosen and its weight is adjusted to agree with its inputs. This process is usually leading to local minima, however there is no guarantee for finding the global minima.

Boltzmann Machines

Boltzmann machines [173] are extensions of Hopfield networks that combine the idea of simulated annealing with neural networks. Boltzmann machines are also considered probabilistic networks as they make steps towards local minima and they have the ability to move away from local minima. Boltzmann machines are presented in Section 3.5.7.

Probabilistic Neural Networks (PNN)

PNNs [347] are feedforward supervised learning ANNs with backpropagation of the error. Used mainly in classification problems, PNNs employ a number of statistical methods to decide to which class the input pattern belongs. Important factors in PNN are the distribution of neighbouring patterns and the distance of the test patterns from them. The decision is carried out by probability density functions (PDF), which are calculated for each class by using a weight function, which is called kernel and is centred to the input pattern. Kernel functions are usually Gaussian functions forming a bell shape. The architecture of PNNs consists of three layers at least: the input, the radial (which models a kernel function centred at each candidate pattern) and the output layer. Buntine [67] provides a guide to probabilistic networks.

2.2.6 The Perceptron

The efforts to solve systems of linear equalities and inequalities by using fast and reliable algorithms dates back to the thirties. Fisher [117] introduced in 1936 the *linear discriminant* as a statistical procedure for classification. For the discrimination, Fisher uses sets of hyperplanes, where each hyperplane is defined by a linear combination of the attribute values. Rashevsky [299] proposed in 1938 binary logic operations, such as addition and subtraction operations, that can be used for implementing binary logic *XOR* operation. A model similar to Fisher's model and Rashevsky's circuits, was suggested by McCullogh and Pitts in 1943 [261]. McCullogh and Pitts introduced a binary device which consists of a weighted sum of the inputs, and a non-linear activation function, which is the original *threshold function*:

$$output_k = \begin{cases} 1 & \text{if } \sum_i w_{ik} x_i \geq \theta_k, \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

McCullogh and Pitts combined the neuron's threshold with binary logic action producing the so called *threshold logic*. The work by McCullogh and Pitts had a great impact on neural network theory, as the exciting result was that a McCullogh-Pitts neuron network could implement any finite logical expression. For example, a simple McCullogh-Pitts neuron with two binary inputs x_1 and x_2 and a threshold $\theta = 2$ is equivalent to the logical operation *AND*, i.e. in order to reach the threshold, both x_1 and x_2 input should have value of "one". If we change the threshold $\theta = 1$, then the neuron performs the logical operation *OR*, i.e. unless both x_1 and x_2 values are zero, they always reach the threshold value of 1. From the neurophysiology point of view. the importance of the McCullogh-Pitts neuron was that it demonstrated that the brain consists of powerful logic and "computational devices".

Hebb [164] introduced in 1949 a neuron model with the ability to learn by adjusting its weights, therefore, establishing that the connection strength between neurons defines their actual functionality. The simple Hebb's rule on weight adjustment was based on the neuron's response in such a way that the network adjusts the weights to increase the probability of desirable response to similar inputs and decreases the probability of a similar undesirable response to inputs.

The *relaxation method* for finding solutions of linear equations was introduced in 1939 by Temple [357]. Agmon [8] introduced in 1954 a similar relaxation method for solving systems of linear inequalities of the type $l^j(\vec{z}) = \vec{\alpha}^j \cdot \vec{z} + b^j \geq 0, j = 1, ..., m$. Associating the inequalities with half-spaces, Agmon's method proposes an iteration procedure that, starting with an arbitrary initial vector $\vec{z}_0$, finds feasible solutions by constructing a trajectory of straight line segments. When $\vec{z}_i$ does not represent a solution of the system, then $\vec{z}_{i+1}$ is taken as the orthogonal projection of the farthest hyperplane which corresponds to a violated linear inequality:

$$\vec{z}_{i+1} := \vec{z}_i + t \cdot \vec{\alpha}^{j_0}, \tag{2.13}$$

where $t = -\frac{l^{j_0}(\vec{z}_i)}{|\vec{\alpha}^{j_0}|^2}$ and $\vec{\alpha}^{j_0}$ maximizes $-\frac{l^j(\vec{z}_i)}{|\vec{\alpha}^j|^2}$.

Agmon's method later became known as the classical *perceptron algorithm*. Rosenblatt [310] introduced in 1958 the *perceptron* as a model for pattern detection. The original perceptron from 1958 was an unsupervised learning pattern classifier. In 1962, Rosenblatt introduced the *classical perceptron*, which turned to be a supervised learning classifier where the weights change

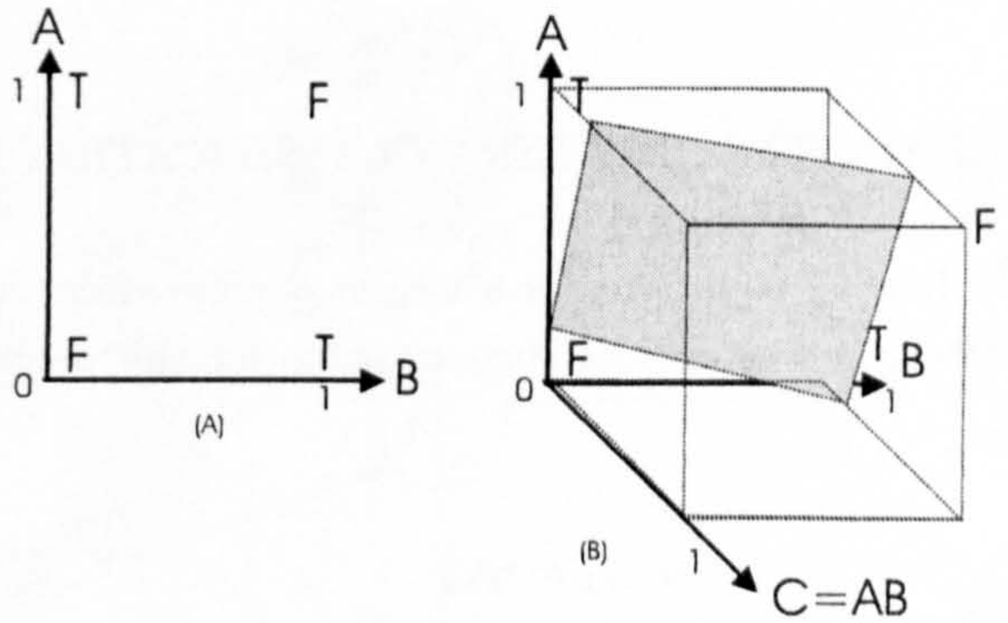


Figure 2.5: XOR Separability.

only in response to a misclassification. Unlike the original perceptron, which is pulled towards some defined goal, the classical perceptron is back-propagating the error so that it adjusts the weights to learn a known classification task.

Rosenblatt's classical perceptron model was used to solve a number of simple classification problems by networks of binary threshold units. The classical perceptron has an input layer, a processing layer, and an output layer. The basic processing unit of the perceptron is the *threshold logic unit.*

The perceptron model uses a learning algorithm called the *perceptron algorithm*, similar to Agmon's relaxation method to adjust the strengths of the connections in case the network provides a wrong classification answer. Rosenblatt proved the *perceptron convergence theorem*:

Theorem 2.1 *The perceptron learning algorithm will find in finite time a set of connection weights that correctly classifies a finite set of patterns if they are linearly separable and if such set of connection weights exists for a classification problem.*

A proof for the *perceptron convergence theorem* can be found in [24].

The main characteristic of perceptron algorithms is that they react to incorrect classifications only. If the perceptron outputs a correct response, then the feedback rule is *do nothing* with the connection weights. If the classification is supposed to be +1 and the perceptron unit responds with 0 (or -1), then the feedback is to add a positive value z to weights in such a way that strength of the connections is increased, and it will be more likely for the perceptron to exceed the threshold and provide an output of +1. Likewise, if the classification is supposed to be 0 (or -1), and the perceptron responds with +1, then the feedback is to add a negative value $-z$ to weights so that it overall weakens the connection strengths, and it will be more likely not to exceed the threshold and provide an output of 0 (or -1). This can be mathematically

formulated in the following equations for a weight change after trial L. If the classification of pattern y was correct then we have for the weight vector at step $L+1$

$$w_{L+1} = w_L; \tag{2.14}$$

if the classification of pattern y was incorrect, then we have for the weight vector

$$w_{L+1} = \begin{cases} w_L + z \cdot y & \text{if pattern } y \text{ is assigned } +1; \\ w_L - z \cdot y & \text{if pattern } y \text{ is assigned 0 or -1.} \end{cases} \tag{2.15}$$

By using this learning procedure, it is more likely to correctly classify the pattern y when provided as input again.

Proof: Suppose that the correct class of pattern y is +1. The inner product of new weights w_{L+1} is larger than previously as

$$\begin{aligned} F &= \sum_{j=1}^{n} w_{j,L+1} \cdot y_j && (2.16) \\ &= \sum_{j=1}^{n} (w_{j,L} + z \cdot y_j) \cdot y_j && (2.17) \\ &= \sum_{j=1}^{n} (w_{j,L} \cdot y_j) + (z \cdot (y_j)^2) && (2.18) \\ &> \sum_{j=1}^{n} w_{j,L} \cdot y_j. && (2.19) \end{aligned}$$

Thus weights are changed by increasing the previous value of the inner product so that the perceptron is guided to overcome the threshold value for the pattern y. The same applies to weakening the weights if patterns have been misclassified.

The learning method of the perceptron has limitations when dealing with noisy data, as the perceptron will never stabilize its weights, making changes in weights forever to classify cases that are wrongly labelled. A further problem of the perceptron method is that the more accurate it becomes, the more the learning slows down and the learning time increases as corrections are made only when misclassifications occur. Another problem deals with the *generalization quality* of the separating hyperplanes, i.e. the quality of correctly classifying unseen patterns after training and stabilization of the weights.

Limitations of the perceptron algorithm, which caused decrease of interest in this model for many years, were stated by Minsky and Papert [268]. Since the perceptron was widely used for recognizing images, Minsky and Papert

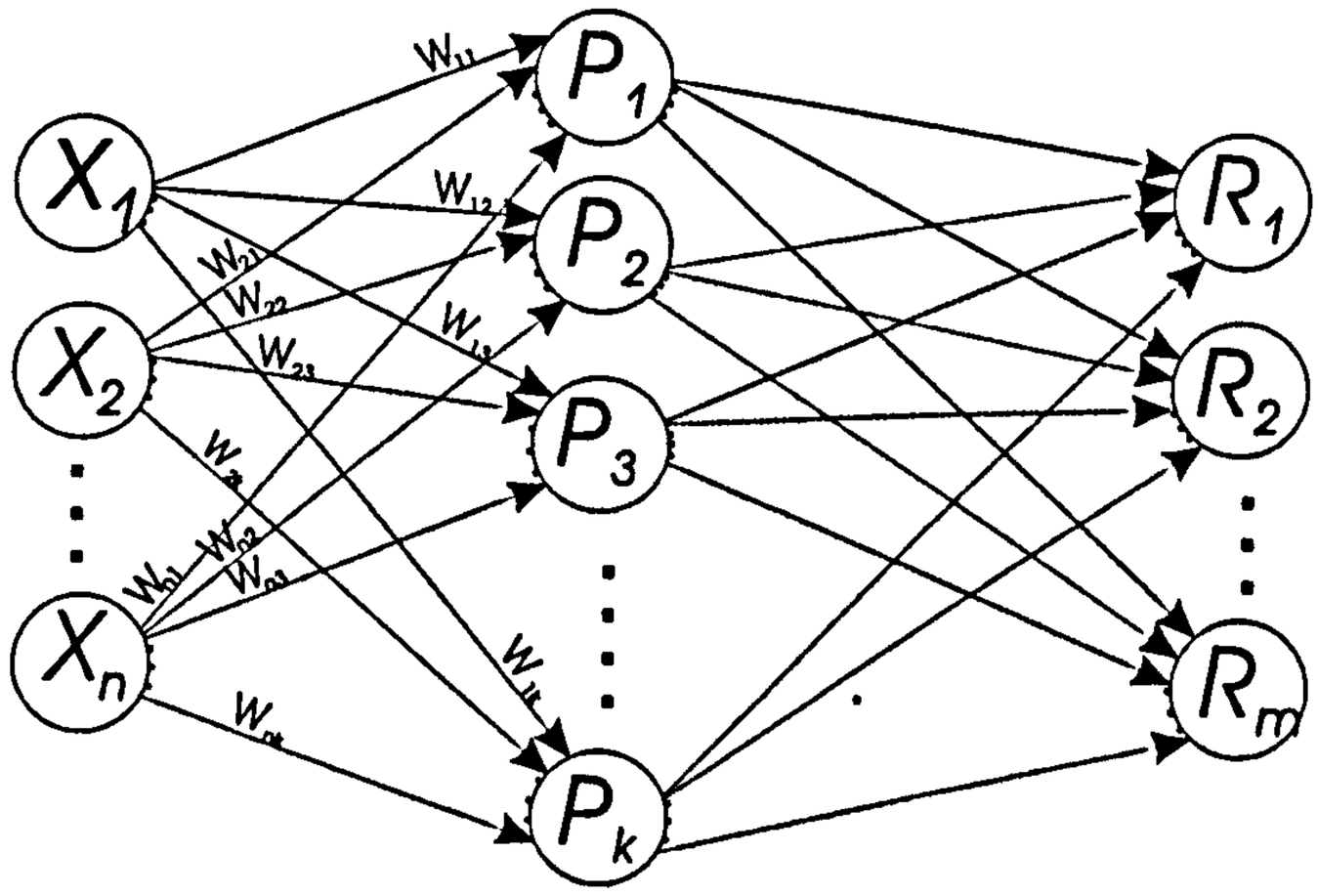


Figure 2.6: Multilayer Perceptrons.

showed that the perceptron has serious limitations to recognize correctly various objects. They also used the *XOR* problem to show that the problem is not linearly separable in two dimensional space, and only if the representation is extended to three dimensional space, we can solve the *((A XOR B) XOR C)* problem (Figures 2.5 (a) and (b)). This involves extra perceptron layers compared to the classical perceptron, which introduces in fact the *multilayer perceptron*, with the extra layer called *hidden layer*. In Figure 2.6, the inputs $X_1, X_2, ..., X_n$ are passed to the k processing units $P_1, P_2, ..., P_k$, where for each unit P_j calculates the inner product of inputs with the corresponding connection strengths $w_{ij}, i = 1, .., n, j = 1, .., k$. The result of unit P_j is compared to the corresponding threshold $\theta_j, j = 1, .., k$, and the output layer $R_1, R_2, ..., R_m$ provides the response to the specific input pattern.

The impact of Minsky's and Papert's work was so strong that the field of neural networks experienced a setback, which lastet until the eighties, when new important models and theories were presented by many researchers [114, 173, 183, 184, 217]. Moreover, research on pattern classification has shown [98, 114] that conventional computational methods turn out to be unsuitable for solving classification problems because massive parallelism is essential. Hardware implementations of neural networks with hundreds of thousands of neurons are possible nowdays, thus enabling faster convergence of the perceptron algorithm and therefore overcoming limitations of the past.

2.2.7 Pattern Classification with Linear Discriminants

Perceptrons and threshold units are *linear discriminants*, i.e. linear functions of a set of parameters as weights, that draw a hyperplane in a two class problem (or n–1 hyperplanes in n-class problem) for separating the classes in the feature space. Such *linear machines* define hyperplanes as decision boundaries. Linear classifiers are trained by minimizing a criterion function, which is usually the misclassification error, by using linear discriminant learning methods (see next Section). The discriminant function is of the following type:

$$f_j(\vec{x}) = \vec{w} \cdot \vec{x} + w_0 = \sum_{i=1}^{n} w_i \cdot x_i + w_0, j = 1, ..., c, \qquad (2.20)$$

where c is the number of classes, $\vec{x}$ is a feature vector, $\vec{w}$ the weight vector and w_0 the threshold weight.

Linear separability is shown in Figure 2.7, where in case (a) a hyperplane can be drawn such that the two classes are separated, whereas in case (b) no such hyperplane can be drawn and thus the two classes are not linearly separable.

Two-Class Problems

If $c = 2$, then we have a hyperplane separating the two classes. Equation 2.20 decides for pattern $\vec{x}$ class C_1 if $f(\vec{x}) \geq 0$, and decides class C_2 if $f(\vec{x}) < 0$. Usually, the decision is that a pattern $\vec{x}$ is a *positive sample* of class C_1, or it is a *negative sample* of class C_1 and therefore C_2 is the set of patterns that do not belong to C_1.

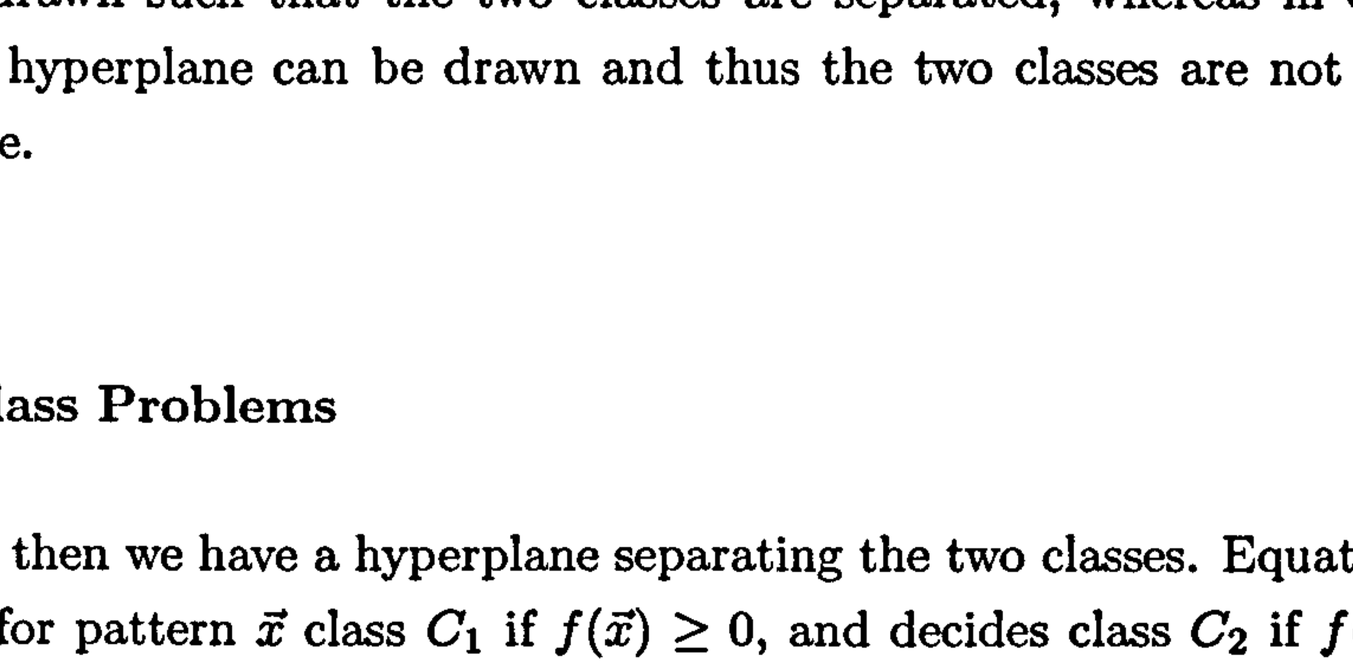


Figure 2.7: Linearly and Non-linearly Seperable Classes.

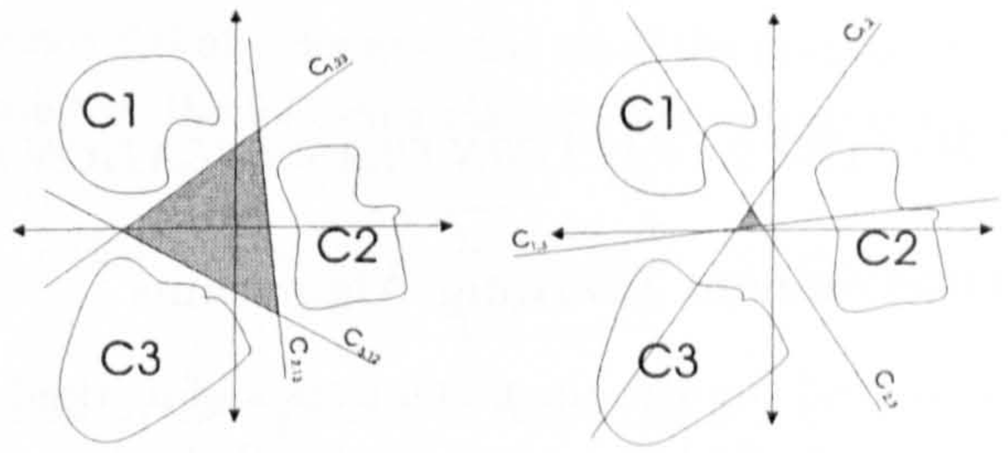


Figure 2.8: Two Approaches for the Multiclassification Problem.

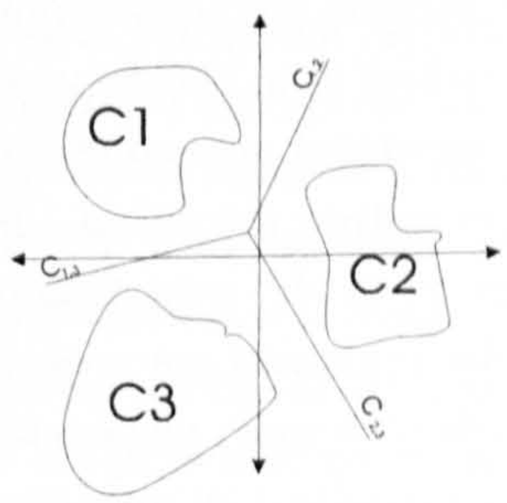


Figure 2.9: Linear Machine Decision Boundaries for a Three Class Decision Problem.

Multiclassification Problems

When $c > 2$, then the classification problem is called *multiclassification* problem and it can be tackled by linear discriminants by reducing it to c two-class problems [343]. A linear discrimant function separates points assigned to class c_j from those not assigned to c_j. Another approach to reduce the multiclassification problem is to consider one linear discriminant for each pair of classes, so that we will need $c(c-1)/2$ linear discriminants. Figure 2.8 shows the two approaches. The shaded area is an ambiguous area where no class can be assigned to patterns of this area. The task is to find for each pair a linear discriminant that implies a smaller ambiguous region, however, the most common approach is the former one, as the latter needs more linear discriminants for $c > 3$.

Linear machines are linear discriminants that overcome the ambiguous regions and divide the feature space into c decision regions, as shown in Figure 2.9, by following decision rule:

$$Class = C_i, \text{ if } \forall\ i \neq j,\ f_i(\vec{x}) > f_j(\vec{x}). \tag{2.21}$$

2.2.8 Linear Discriminant Learning Algorithms

A linear classifier is trained through the minimization of a criterion function Z, an approach that involves finding a set of weights that solve a set of linear inequalities $\sum_{i=1}^{n} w_i \cdot x_i > \theta$. A number of methods exist for minimizing the criterion function, like gradient descent, Newton descent algorithm, perceptron classical algorithm, and relaxation methods. Combinatorial optimization as an effective minimisation method is in the focus of our work, where the powerful optimization method simulated annealing is combined with a classical learning method like the perceptron algorithm. This method along with the results obtained are presented in the next Chapters. Here, alternative learning procedures for linear discriminants are presented.

Such procedures involve *gradient descent*, where starting from an arbitrary weight vector $\vec{w}$, the $\vec{w}(k+1)$ are obtained by moving apart from $\vec{w}(k)$ by the *rate of learning* ℓ, that sets the learning step size, and the gradient vector $Z(\vec{w}(k))$. The following equation illustrates this:

$$\vec{w}(k+1) = \vec{w}(k) - \ell(k)\nabla Z(\vec{w}(k)), \tag{2.22}$$

where $\nabla Z(\vec{w}(k))$ is the derivative or gradient of $Z(\vec{w}(k))$, i.e. $\nabla Z(\vec{w}(k)) = grad\ Z(\vec{w}(k)) = \frac{\partial Z(\vec{w}(k))}{\partial \vec{w}(k)}$
A gradient descent algorithm might have the following pseudocode:

procedure gradient descent
begin
initialize $\vec{w}, \theta, \ell, k$
do
$k := k + 1;$
$\vec{w}(k+1) = \vec{w}(k) - \ell(k)\nabla Z(\vec{w}(k));$
while $\nabla Z(\vec{w}(k+1)) < \theta$
return $\vec{w}(k+1)$
end

If we substitute ℓ by the inverse *Hessian matrix* H^{-1} of second derivatives $\partial^2 Z/\partial w_i \partial w_j$ estimated at $w(k)$, then we obtain the *Newton descent algorithm*. The Newton algorithm gives a greater improvement per step than gradient descent, however, the time to estimate matrix H^{-1} may overbalance Newton's algorithm advantages [106].

The classical perceptron algorithm employs the *perceptron criterion function* where the next weight vector $\vec{w}(k+1)$ is obtained by adding some weight correction towards or away from a class. This weight correction is proportional

to a pattern vector $\vec{x}$ that belongs to the set of the misclassified patterns $S\Delta f$. This is illustrated by the following classical perceptron update rule.

$$\vec{w}(k+1) = \vec{w}(k) - \frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sqrt{\sum_{i=1}^{n} x_i^2}} \cdot \vec{x_i}, \ \vec{x} \in S\Delta f. \tag{2.23}$$

This rule is similar to the Agmon's [8] relaxation methods for solving linear inequalities (see Section 2.2.6). Variants of the perceptron algorithm on samples that are consistent with linear separation are presented by Duda et al. [106]. Two examples of methods for finding a separating vector when the samples are linearly separable are the *batch perceptron algorithm* and the *fixed increment perceptron algorithm*. In the batch perceptron algorithm, the weight solution vector is based on a large group of patterns if computing weight updates, instead of a single sample. In the *fixed increment perceptron algorithm*, weight update occurs every time a pattern is misclassified.

Variants of the perceptron algorithm on sample sets that are inconsistent with linear separation are presented by Bylander in [68, 69]. Relaxation methods related to nonseparability are also presented by Duda et al. [106] and involve methods like the *minimum squared-error procedure*, where the sum of squared error criterion function $\sum_{i=1}^{n}(w_i \cdot x_i - b_i)^2$ is minimized, with b_i being some arbitrarily specified positive constants.

Linear discriminants are usually not sufficient for large pattern classification problems. Multilayer networks or threshold circuits are used to overcome this problem by employing multiple copies of single nonlinear functions of the input features and trying to find an appropriate nonlinear function.

2.2.9 Pattern Classification with Multilayer Neural Networks

Real world problems contain noisy data, which usually provide us with nonseparable data. This is the main problem for linear discriminant functions. The smaller the training set, the easier it is to find a weight vector that separates the data. This is the motivation for using multilayer neural networks, where we increase the complexity by employing a number of nodes working as linear discriminants, which have a subset of the overall training set to separate. The nodes are then combined to provide us with the classifier decision on test patterns. Therefore, instead of one weight vector, we have a number of weight vectors, which form the solution functions of the classification problem. What is achieved by this approach is to have a classifier that learns nonlinear functions that can cover arbitrary decision regions. In Figure 2.10, we see that a depth-1 threshold function (having two layers, one for the inputs x_1, x_2

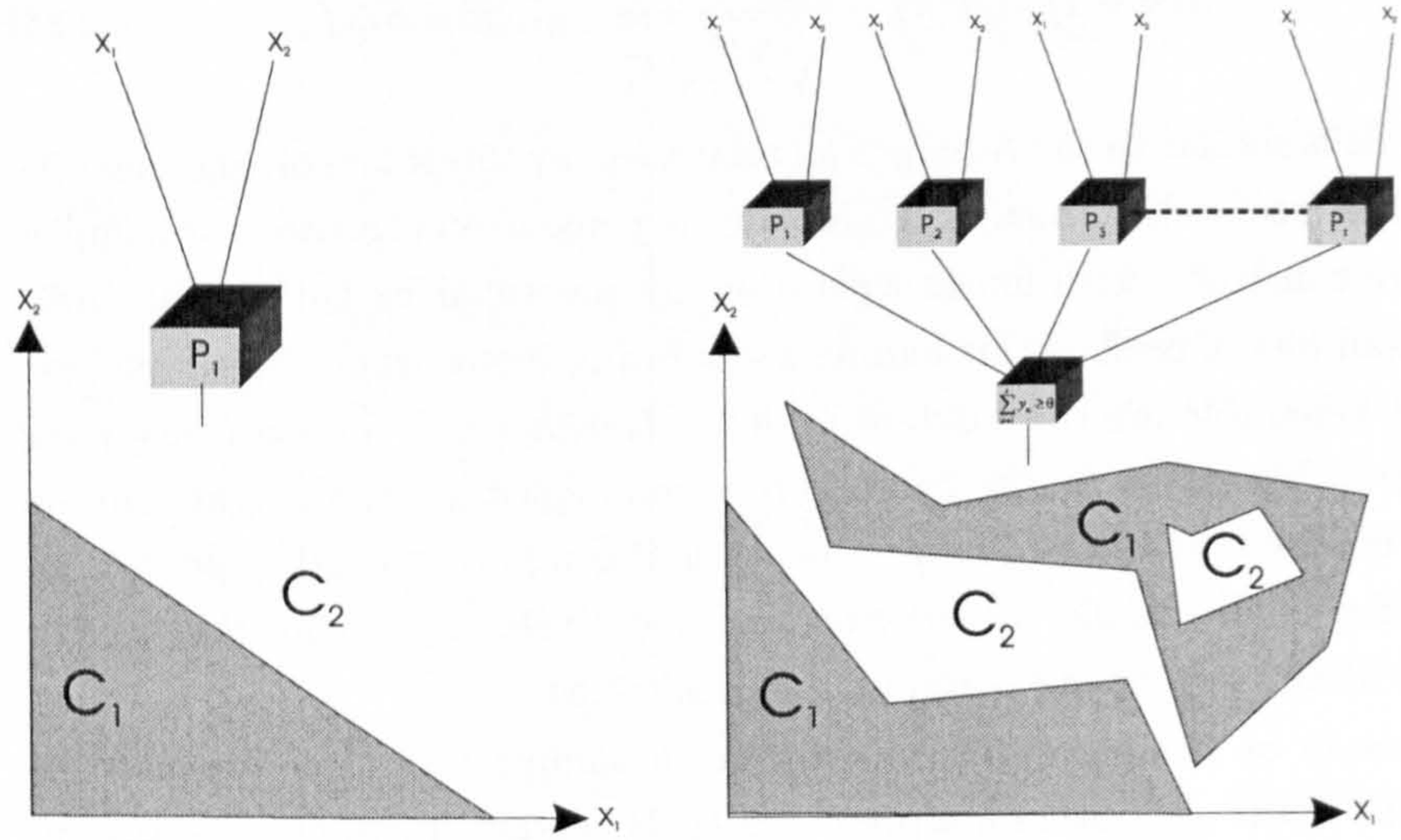


Figure 2.10: Single-Layer and Multilayer Discriminants.

and one for the computational unit P_1) can be a linear discriminant separating the feature space into two regions, one for each class. However, if the features produce complex regions that might be even not connected, a three layer depth-2 threshold circuit could theoretically implement such arbitrary decision boundaries.

The key power of multilayer neural networks in pattern classification is that the nonlinearity can be learned from training data. In this way, powerful classification machines can be built and applied to difficult real world classification problems. Kolmogorov [219] proved that

Theorem 2.2 (Kolmogorov) *For properly chosen functions Φ_j and Ψ_{ij}, any continuous function defined on the unit hypercube I^d $(d \geq 2)$ can be represented in a form:*

$$f(x_1, .., x_n) = \sum_{j=1}^{d} \Phi_j \left(\sum_{i=1}^{n} \Psi_{ij}(x_i) \right). \tag{2.24}$$

Equation 2.24 looks similar to the output of a three-layer network, where the n inputs x_i are passed with some weights w_{ij} to d hidden units. The output of the d units is calculated by using the d new weights and provides the output of the network $f(\vec{x})$. Unfortunately, the functions Φ_j and Ψ_{ij} are not the simple weighted sums passed through activation functions as in neural networks. The functions Φ_j and Ψ_{ij} can be extremely complex. There has been a debate among researchers [93, 138, 185, 222, 223, 228] as to whether or not Kolmogorov's theorem is relevant in neural networks theory. Moreover,

Hecht-Nielsen [165] used Fourier's theorem that any continuous function can be approximated by an infinite sum of harmonic functions to theoretically prove that a three-layer network can implement such continuous functions.

Kolmogorov's theorem and Hecht-Nielsen ideas are of great theoretical importance; however, in practice they do not provide us with a universal architecture as both ideas give us no information about a) the number of hidden units needed, b) the proper weight values, and c) the types of activation functions. The most important point is that both methods give us no answer to the question: how to find the nonlinear functions based on training samples. Consequently, a constructive proof of the universal expressive power of three-layer networks still remains a hard problem and very little work has been done into this direction. The practical impact of Kolmogorov and Hecht-Nielsen framework is that three-layer classifiers are confidently good classifiers, resulting in a general belief that given sufficient number of hidden units, proper activation functions and proper weights, any continuous function from input to output can be implemented by a three-layer neural network. However, the limitations are that we have no theoretical guidelines for the number of hidden units, the type of activation functions, and the construction of proper weights. These limitations are equivalent to major questions in pattern classification of how to design the topology, or how to define the architecture (both related to the network complexity issue), and how to select the training method.

2.2.10 Neural Networks and Supervised Learning Algorithms

The aim of supervised network learning is to set the weights based on the training samples and the desired output. Generally, network learning uses a feedforward operation to present a pattern to the input units, passing the signal to the output units to define an output value. Learning takes place by changing the network parameters to get closer to the desired output. Any difference of the current output to the desired output corresponds to an error. This error is expressed in terms of a function of the weights, called *criterion function* or *objective function*, and the aim is to reduce it by adjusting the weights. Learning procedures may depend on a *per pattern basis* correction or may depend on a *batch training* correction, where adjustment takes place iteratively after presenting all training samples.

When dealing with threshold units, the perceptron learning algorithm is used for adjusting the weights. We have already seen that one of the limitations of the classical perceptron is to solve the *XOR problem*. These limitations can be overcome by expanding the classical perceptron to multilayer perceptrons.

A combination of a powerful stochastic combinatorial optimization method like simulated annealing with the perceptron algorithm was introduced by Albrecht and Wong [19], forming the LSA machine. Our work will focus on this new combinatorial optimization supervised learning method, and therefore the LSA machine will be presented in details in Chapter 4.

For differentiable activation functions like the sigmoid function or a polynomial function, *backpropagation* [315] is the most widely used method for training multilayer neural networks. Generally, the term *backpropagation* is used in literature to mean a number of different things like to describe multilayer perceptron architectures, or to describe the training of multilayer perceptrons. In the original definition, the backpropagation term is used to evaluate the derivatives of the error function with respect to their weights and to utilize these derivatives to adjust the weights. Basically the backpropagation algorithm is based on gradient descent, where the weights are changed into a direction that will reduce the error:

$$\Delta\vec{w} = -\ell\frac{\partial Z(\vec{w})}{\partial\vec{w}}, \tag{2.25}$$

where the training error $Z(\vec{w})$ is

$$Z(\vec{w}) = \frac{1}{2}\sum_{i=1}^{n}(d_i - m_i)^2, \tag{2.26}$$

and $\vec{w}$ is the weight vector representing all weights in the network; d_i and m_i are the desired and the network output, respectively. The ℓ in (2.25) is the learning rate, and since the equation is always positive, the equation 2.26 moves into the reduced error direction. The backpropagation algorithm iteratively updates weights from step k as:

$$\vec{w}(k+1) = \vec{w}(k) + \Delta\vec{w}. \tag{2.27}$$

Thus, one has to calculate $\Delta\vec{w}$ for each input-output layer. Duda et al. [106] proved that, given layers i, j, their weights w_{ji}, and the input pattern x_i from units i to j, then $\Delta w_{ji} = \ell\delta_j x_i$ implements the backpropagation algorithm at each layer as

$$w_{ji}(k+1) = w_{ji}(k) + \ell\delta_j x_i. \tag{2.28}$$

Therefore, a simple backpropagation algorithm for a three-layer network by randomly selecting a pattern $\vec{x}$ for updating weights is given by:

```
procedure backpropagation
begin
  initialize w⃗, θ, ℓ, k,
 i = layer − one, j = layer − two, m = layer − three
  do
   k := k + 1;
   randomly select input pattern x⃗;
   w_ji(k + 1) = w_ji(k) + ℓδ_j x_i ;
   calculate outputs x_j at layer j ;
   w_mj(k + 1) = w_mj(k) + ℓδ_m x_j;
  while ∇Z(w⃗(k + 1)) < θ
  return w⃗(k + 1)
end
```

The function $Z(\vec{w})$ induces the error surfaces [187] on the weight space, consisting of minima, maxima and plateaus. From studying error surfaces we can gain information about if the error surface has large plateaus (this means that the error varies only slightly as a function of weights leading to slow convergence) or if many local minima exist in the landscape (i.e. the neural network is unlikely to find global minima). As the starting point, the question of weight initialisation has to be considered. The initialisation of weights may determine the uniformity or non-uniformity of learning, where *uniform learning* occurs when all weights are reaching their final equilibrium values in about the same time. In most cases, however, *non-uniform learning* occurs, either because weight initialisation may be poor or because some characteristics of the problem have weights that converge slower. In this case, the learning rate is important to be balanced and not to be very fast nor to be very slow, or it can be even different with respect to each weight as proposed in [106] for multilayer networks training with backpropagation.

Apart from using the square root error of (2.26) as the criterion for training in backpropagation, one can choose other training criterion functions. If d_{mk}, b_{mk} are the known probabilities of the desired output and the actual output of k units for pattern m, then the *Minkowski error* gives us such a criterion:

$$Z(\vec{w}) = \sum_{m=1}^{n} \sum_{k=1}^{c} |b_{mk}(\vec{x}) - d_{mk}(\vec{x})|^R, \tag{2.29}$$

where $R \geq 1$.

Another criterion is the *cross entropy* that measures a type of distance

between probability distributions:

$$Z(\vec{w}) = \sum_{m=1}^{n} \sum_{k=1}^{c} d_{mk} \ln (d_{mk}/b_{mk}). \quad (2.30)$$

Other criterion functions employ complexity measures and relate the complexity to the criteria that have to be minimized. In such methods, the new criterion function tries to balance between network complexity and error on the training set by penalizing highly complex models.

Other approaches to reduce the training error involve methods of eliminating weights. *Weight decay* [379] is a weight elimination method where after each weight update, weights are decayed according to :

$$\vec{w}(k+1) = \vec{w}(k)(1-\epsilon), \text{ where } 0 < \epsilon < 1. \quad (2.31)$$

Therefore, only weights that are needed to solve the classification problem are kept. The importance of each weight is utilized in [92, 349]. Following Wald's [374] idea for estimating the importance of a parameter in a model, one tries to predict how the training error depends upon a weight, eliminating weights that have small contribution to the problem's solution.

As seen for multilayer networks, much appreciation is given to units with activation functions that have derivatives. Backpropagation methods can be used for estimating weight updates, cf. Werbos [382] for an overview of backpropagation methods. Most of the neural network textbooks deal with pattern classification by multilayer neural networks [5, 50, 266, 307]. Relations of neural networks and statistical pattern classification are also discussed in [274, 305, 314, 335, 377].

2.2.11 Threshold Circuits

Perceptrons are depth-2 threshold circuits consisting of a single threshold gate at the root with AND gates at the next level. The basic computing unit is the *threshold logic unit* which was introduced by McCullogh and Pitts in 1943 [261]. The threshold unit forms the sum of the inner product between the input pattern, which represents $X_1, X_2, ..., X_n$ features, and the connection strengths $w_1, .., w_n$. The sum is compared to a threshold θ, and the threshold unit outputs only one of two values, which is either 0,1 or -1,+1. The threshold unit together with the perceptron learning algorithm can learn two classes, if they are linearly separable. Non-linearly separable classes, which are the most frequent in real world problems, require more complex decision surfaces and can be solved by threshold circuits of a more complex nature, see Section 2.3.3.

2.3 Complexity Issues of Threshold Circuits

The complexity of pattern classification problems can be characterized by the following questions:

- Sample Complexity: How large should the training set be so that one can expect good performance in the training and testing phase?
- Computational Complexity: How much computational effort is required to achieve good performance in the training and testing phase? Is there a learning algorithm that guarantees to find good weights and thresholds in polynomial time related to the sample size?
- Circuit Complexity: What topology or network architecture is required for expecting good performance on training and test data?
- Size Complexity: How many nodes are best describing a function that has to be learned?
- Depth Complexity: How many layers are best describing a function that has to be learned?

We briefly discuss these complexity issues. The sample complexity is closely related to the *VC-dimension* [370], defined by Vapnik and Cherkovski, while the combined size and depth complexity defines the *depth vs. size problem.*

2.3.1 PAC-Learning and VC-Dimension

One of the greatest perceived advantages of neural networks, which has been one of the primary reasons for the high degree of interest in the field, is the ability to generalize. Despite the interest to the field there is little research addressing the theoretical analysis of generalisation ability. Experimental results in networks tell us only about the generalisation performance of specific networks applied to specific problems. Theory on generalisation comes mainly from the work of Baum and Haussler [41], who introduced methods for the analysis of generalisation based on Valiant's work [367] on learnability theory of *probably approximately correct (PAC) learning*, and the work of Vapnik and Chervonenkis [370] on the theory of the uniform convergence of relative frequencies to propabilities, better known as VC-dimension.

PAC learning and VC-dimensionality are closely related to the size of training examples used for training a neuron unit, where each example is drawn independently according to a fixed distribution on the total available training data. A *learning function* for a class $\omega \in \Omega$ is a function f, which, given a

sufficiently large sample set, returns a hypothesis h that is a *good approximation* to ω with high probability [178]. Defining the sufficiently large sample set induces the *sample complexity* of the learning function, which is the smallest sample size guaranteed to achieve good approximation, and the class for which there is such a learning function is called *uniformly learnable.* While PAC learning is related to the learnability of a function, the VC-dimension influences the speed of convergence and hence is also related with the actual number of training examples required. The link between PAC learnability and VC-dimension is that a class ω is uniformly learnable, i.e. there exist a PAC learning algorithm, if and only if the VC-dimension of the class is finite [56, 178].

Theoretical analysis on lower and upper bounds on VC-dimension has been performed for some type of networks. Baum and Haussler [39, 41] show that in a feedforward network with W weights and P computational threshold units with n inputs and j units the VC-dimension is bounded

$$VC \leq 2 \cdot W \cdot log_2(e \cdot P). \tag{2.32}$$

Moreover, they showed that for $0 < \epsilon < 1/8$:

$$N \geq \frac{W}{\epsilon} \log_2 \left(\frac{P}{\epsilon} \right) \tag{2.33}$$

meaning that if N patterns can be learned by the network such that a fraction of $1 - \epsilon/2$ is correctly classified, then there is a high probability that the network on further patterns will correctly classify a fraction of $1 - \epsilon$

Maas [251] has obtained a lower bound $Wlog_2W$ for certain types of multilayer feedforward network of threshold units. Other tight bounds for specific type of functions have been obtained in [35, 37, 143, 209, 218, 345, 381]; see Anthony and Bartlett [28] for a review of the field.

Bounding VC-dimensions is a challenging task in mathematical investigations of neural networks which provides a number of sophisticated mathematical tools. The bounds, however, tend to be too large, since they provide such guarantees of generalisation for any probability distribution of training examples and for any training algorithm that minimizes the training error on the training examples. Therefore, the VC-dimension is a very general theoretical measure of pattern classification ability. The proposed bounds are usually unrealistic and with limited practical value for real world problems. Other alternative methods for calculating more realistic bounds should be considered, as in Haussler et al. [161], where a probability distribution was introduced for considering the performance of a classifier that implements a Bayes opti-

mal classification algorithm [106]. Empirical studies investigating the relation of classification ability to parameters of the problem are important for more realistic VC-dimension bounds.

2.3.2 Computational Complexity

Algorithmic complexity presented in Section 3.2 refers to the hardness that an algorithm can find an optimal solution to an optimization problem. The class NP, and specifically the class of NP-complete problems, denote classes of problems that are very hard to solve. In neural network theory, Yannakakis [398] proves that finding a stable configuration is NP-hard. Judd [202] shows that for a given neural network and a set of training examples it is NP-complete to find a set of weights that can correctly classify all of the training examples. Blum and Rivest [55] prove that even for a very small network to find weights and thresholds that learn any given set of training examples is a very hard problem. Specifically, given a two-layer network with three-threshold units of n inputs, it is NP-complete to decide whether there exist weights and thresholds that produce outputs consistent with a given set of training examples. Unless $P = NP$ (see Section 3.3.2), for any polynomial time training algorithm (see Section 3.3) there will be some sets of training data on which the training algorithm fails to correctly classify the data. Höffgen and Simon [177] deal with the problem of learning a probably almost optimal weight vector for a neuron, finding that the minimum error cannot even be approximated within a constant factor in polynomial time, thus it is an NP-complete problem. They conclude that their learning model might be too restrictive and that *"learning should use more powerful tools than random examples only. It is however not quite clear which additional tools are available for the purposes of neural learning"*. Thus, there is a need for new ideas about finding a probably almost optimal weight vector.

2.3.3 Circuit Complexity

Similarly to the importance of the learning methods is the importance of the *network architecture* or *topology*, where the optimal topology depends on the classification problem, as knowledge of the problem is tried to be incorporated into the network through the choices of input units, the number of hidden layers, selection of feedback connections etc. Selecting the optimal network configuration is a hard decision, which can be viewed as a heuristic network model selection method. The problem of selecting or adjusting the complexity of the network is called *regularization* and is a very hard problem, where little or no process exists in the literature.

How many computational units, how many layers, how many computational units in each layer, what type of connections between the units, what type of weights and threshold should be used in each connection? These are some questions related to the network complexity problem. Once a network topology is set, parameter estimation takes place for designing the classification network. If too many free parameters are used, then one may have a poor generalization and, conversely, if few free parameters are used, one may face a poor training quality. The problem of finding the smallest network that can realize an arbitrary function given a set of m vectors in n dimensions [44] defines the *circuit complexity* problem. The circuit complexity problem is of great importance for hardware implementations and tries to find tight bounds for the number of units used to realize an arbitrary function. Research on tight bounds already began in the fifties in relation to threshold logic. Beiu [44] gives a large number of bounds for a number of different network approaches. We present here a short list of circuit complexity results:

- Neciporuk (1964) [272]: $size \geq 2 \cdot (2^n/n)^{(1/2)}$ is the asymptotic lower bound on the size of a threshold circuit for "almost" all n-ary Boolean functions;

- Lupanov (1973) [250]: $size \leq 2 \cdot (2^n/n)^{(1/2)} \cdot (1 + \underline{o}[(2^n/n)^{(1/2)}])$ is the upper bound on the size of a threshold circuit for "almost" all n-ary Boolean functions realized in a depth-four circuit;

- Allender (1989) [21]: $size = n^{O(\log n)}$, stating that any Boolean function computable by a polynomial size constant-depth logic circuit with unbounded fan-in is also computable by a depth-3 threshold gate circuit of superpolynomial size;

- Razborov and Wigderson (1993) [300]: $size \geq n^{\Omega(\log n)}$, on the size of depth-3 threshold circuits which compute polynomial size functions on a depth-3 model with AND gates at the bottom level.

Of particular interest in computer science theory is how the increase in depth affects the performance. This is the depth vs size problem, which is described in the next section.

2.3.4 Size and Depth Complexity

The number of hidden units and hidden layers governs the expressive power of the network classifier and the complexity of the decision boundary. If patterns are linearly separable, few hidden units (even a single threshold unit) are

adequate, while the more complicated the problem is, the more hidden units or hidden layers are needed. Overfitting may occur in the presence of too many hidden units or hidden layers, whereas if too few hidden units are used, the classifier does not have enough information and free parameters to fit the training data well. The main question is whether to use more hidden units or more hidden layers. This forms the depth vs size problem, a difficult and unanswered problem in theoretical computer science. Most researchers relying on Kolmogorov's theory [219] decide using more hidden units in a three-layer network. Since we have seen that no constructive proof of the universal expressive power of a three-layer network exists, the question of improving classification performance by extending in depth is an open problem. Increasing a network in depth also increases the network complexity, and yields complicated network architectures. The main problem is how the hidden units at larger depths will be expressed in terms of previous depths. The difficulty of finding learning algorithms for training all layers along with the expressive power of three-layer networks gave favor to the decision of not using more hidden layers. Judd [201] and Abu-Mostafa [7] favor shallow circuits. Although there exist studies relating the minimum size to minimum constant depth [342], there is little known about the trade-off relation and, moreover, there is even less known about how design parameters like topology, weights and thresholds influence time and generalization performance [44].

2.4 No-Free-Lunch Theorems (NFLT)

The *No-Free-Lunch Theorems (NFLT)* [391] for pattern classification [392, 393] suggest that no classification method should be preferred over another classification method, not even with respect to random guessing. Which classifier performs better is problem-dependent and is determined by the type of the problem and by the prior knowledge or information acquired about the problem. Thus, if any problem is viewed as a "black-box", no algorithm seems to perform best for all types of problems. If it outperforms other methods for a class of problems, NFLT implies that it has worse performance on other classes of problems, so that the average performance is equal for all classifiers (even for random guessing).

2.4.1 The Original NFLT

The idea of the NFLT began with the search for a "conservation law in generalization" [323, 388] like the ones that hold in Physics, such as the conservation of energy, the conservation of momentum, or as the second law of

thermodynamics that states that in an isolated system the entropy can never decrease (or, equivalently, is always in its maximum state). NFLT gives an analogous interpretation in search, optimization and classification, where, if any search, optimization or classification problem is viewed as "black-box" (the isolated analogous), then no search algorithm, classification or learning method is favoured for all types of problems. There exist a class of problems where an algorithm performs better than another algorithm, however there exist other classes of problems where the same algorithm performs worse. The result is that *"for any algorithm, any elevated performance over one class of problems is offset by performance over another class"* [391] and therefore an algorithm should be tailored to a specific problem. The term *No-Free-Lunch* was introduced by Wolpert in [389, 390], but became more widely known by the *No-Free-Lunch Theorems* in optimization [391]. In his work, Wolpert presented the original framework for the no-free-lunch, in particular,

Theorem 2.3 The No-Free-Lunch Theorem *If the performance of some algorithm α_1's is superior to that of another algorithm α_2 for some set of optimization problems, then the reverse must be true for the set of all other optimization problems, so that for any pair of algorithms α_1, α_2 it holds:*

$$\sum_f P(d_m^y|f,m,\alpha_1) = \sum_f P(d_m^y|f,m,\alpha_2), \tag{2.34}$$

where f is the cost function of an optimization problem, and $P(d_m^y|f,m,\alpha)$ the performance of an algorithm α iterated m times on a cost function f of an optimization problem with cost vectors d_m^y for samples of size m.

A proof for the theorem can be found in Wolpert and Macready NFLT for optimization [391]. The interpretation of (2.34) is that the performance of $P(d_m^y|f,m,\alpha)$ is independent of any chosen algorithm α.

Wolbert and Macready [391] also extended NFLT to time-dependent optimization problems. Schumacher et al. [331] improved NFLT by proving it in a more general setting.

2.4.2 Supervised Learning and NFLT

Consequences of the NFLT were dramatic in Machine Learning and in optimization as it turned out that research of the past decades on general purpose "black-box" Machine Learning and optimization algorithms was meaningless. For supervised learning, Wolpert showed that if the prior distribution over the target functions is uniform and the off-training-set error, i.e. the error on

points not in the training set, is taken to be the performance criterion, then there is no difference between the learning algorihms [106, 393]:

Definition 2.4 (The expected off-training-set classification error.) *Given a fixed training set D, a hypothesis $h(\mathbf{x})$ to be learned, $P_A(h(\mathbf{x})|D)$ as the probability that the algorithm A will yield hypothesis $h(\mathbf{x})$ when trained on D, $F(\mathbf{x})$ the target function to be learned, $P(\mathbf{x})$ the input $\mathbf{x}$ probability, then the expected off-training-set classification error of an algorithm A is given by:*

$$\varepsilon_A(E|F,n) = \sum_{x \notin D} P(\mathbf{x})[1 - \delta(F(\mathbf{x}), h(\mathbf{x}))]P_A(h(\mathbf{x})|D), \qquad (2.35)$$

where the Kronecker δ function has value 1 if its two arguments match, and 0 otherwise; f is the cost function of an optimization problem and $P(d_m^y|f,m,\alpha)$ the performance of an algorithm α iterated m times on a cost function f of an optimization problem with cost d_m^y.

The no-free-lunch theorem in supervised learning states:

Theorem 2.5 (The no-free-lunch theorem in supervised learning.) *Let be given two learning algorithms $A_1(h|D)$ and $A_2(h|D)$. Then it holds:*

1. *For n training points, uniformly averaged over all F,*
   $\varepsilon_{A_1}(E|F,n) - \varepsilon_{A_2}(E|F,n) = 0$;

2. *For any fixed training set D, uniformly averaged over all F,*
   $\varepsilon_{A_1}(E|F,D) - \varepsilon_{A_2}(E|F,D) = 0$;

3. *For n training points, uniformly averaged over all priors $P(F)$,*
   $\varepsilon_{A_1}(E|n) - \varepsilon_{A_2}(E|n) = 0$;

4. *For any fixed training set D, uniformly averaged over all priors $P(F)$,*
   $\varepsilon_{A_1}(E|D) - \varepsilon_{A_2}(E|D) = 0$.

Part 1 and 2 state that no learning algorithm is superior for all target functions for fixed D and n, whereas Part 3 and 4 are related to nonuniform target function distributions instead of F.

2.4.3 Implications of the NFLT

Attempts to explain the NFLT in simpler ways can be found in [182]. The NFLT is found to be also difficult to be proven empirically as it is hard to enumerate $\sum_f P(d_m^y|f,m,\alpha)$. Due to its generality, NFLT initiated controversial

discussions, from Dembski's [97] interpretation that NFLT violates Darwinism and natural selection, to more Machine Learning focused approaches and optimization problem focused approaches.

Problems where NFLT is applicable can be found in [85, 190, 191, 254, 331], whereas problems where NFLT do not hold can be found in [86] for multiobjective problems, [103, 104] for optimization, [353] for problem description length, and [101] for Machine Learning. Woodward and Neil [394] suggest that NFLT does not hold for search algorithms applied to program induction due to the universal nature of the mapping between program space and functionality space, whereas they suggest a stronger version of NFLT for the class of problems where the goal is to minimise a quantity. NFLT have been extended to noise prediction [254], to multiobjective optimization problems [85], to supervised learning [392, 393], to cross-validation in supervised learning [147, 402], and to early stopping in supervised learning [71]. Michalski and Tecuci [265] multistrategy approach of combining the best features of two or more classifiers into a single classifier is disputed with respect to NFLT in [101]. According to [91, 384], there is no proof that NFLT hold for the NP class, as this would imply $NP = P$ (see Section 3.3.2). Based on this fact, many researchers escape NFLT by looking at specific types of problems, like NP-complete problems (see Section 3.3.3), which are closer to real world applications.

The debate may continue and the future will decide on how right or wrong NFLT are, however, the major contribution of NFLT is that it changed the way that researchers should work with their algorithms. NFLT motivated researchers to focus on finding ***when*** their algorithms work better rather than trying to beat the rest of algorithms. NFLT motivated also researchers to improve their algorithms by searching for and adjusting problem-dependent parameters. Under the light of the NFLT, optimization for improved classification should be studied under reasonable restrictions related with the specific domain of the classification problem.

Chapter 3

Combinatorial Optimization Algorithms

As seen in Section 2.2.8, training involves the minimization of a criterion function, which represents the error. Minimization problems and combinatorial optimization are closely related and therefore classification problems can also be viewed as part of combinatorial optimization problems. As mentioned in Section 2.2.10, our approach will combine the classical perceptron algorithm with simulated annealing, which is an effective combinatorial optimization algorithm. The combination is presented in details in Chapter 4, whereas in this Chapter our focus is on understanding the characteristics of combinatorial optimization algorithms, the type of problems that they are capable to solve, to describe local search and approximation as well as stochastic algorithms, which are the main ideas behind simulated annealing. Furthermore, we present the arsenal of alternative algorithms for solving combinatorial optimization problems.

3.1 Optimization Problems

Optimization problems can be divided according to the type of variables into problems with *continuous* variables and problem with *discrete* variables, where the latter is also called *combinatorial optimization* [281]. In general terms, the word *combinatorics*, from which the word *combinatorial* is originated, concerns with the study of arrangement and selection of discrete objects. *Combinatorial optimization* then, is concerned with the determination of an optimal arrangement or order [233]. Combinatorial optimization can also be viewed as the discipline of decision making in the case of discrete alternatives [3]. The algorithms try to answer a question of type "What is the shortest path?" or "What is the minimum (maximum) cost (benefit) from a selection of discrete

options?". The problem usually is a question whose answer is a function of several parameters. A solution (optimal or not) to a combinatorial optimization problem requires that a suitable algorithm, when applied to an instance of the problem, produces the desired solution [320]. According to [281], the task of solving a combinatorial optimization problem amounts to finding the best or optimal solution among a finite or countable infinite number of alternative solutions.

Combinatorial optimization has been a research field for decades, even for centuries according to [330], where a history of the field from the 18^{th} century till the 1960 is presented. In the last decade, our ability to solve large combinatorial optimization problems has been dramatically improved due to the advent of powerful computer systems. At the same time, the availability of such powerful computers along with reliable software, inexpensive hardware and adequate languages for modelling complex problems, much faster led to a much greater demand for optimization tools [180]. Consequently, the research interest with respect to reliable optimization methods for larger problems becomes increasingly demanding. Although combinatorial optimization problems dominate the field of optimization, there are also examples of problems where the unknown optimal solution might be a continuous quantity like a function or a shape of a body [248]. Annotated bibliography of the field can be found in Dell'Amico et al. [96]. The next sections are based on Papadimitriou/Steiglitz [281] and Aarts/Lenstra [3]. Early books on combinatorial optimization are by Lawler [233], Schrijver [329], and Nemhauser/Wolsey [273].

3.1.1 Definitions

Based on Aarts/Lenstra [3] we define:

Definition 3.1 *A combinatorial optimization problem is either a minimization problem or maximisation problem and is specified by a set of problem instances, also called the problem domain.*

Thus, for the optimization setting we have:

Definition 3.2 *An instance of a combinatorial optimization problem can be formalised as a pair* $(\mathbf{S_p}, \mathbf{f})$, *where the solution space* $\mathbf{S_p}$ *denotes the finite set of all possible solutions, and the cost (or objective) function* f *is a mapping defined as*

$$f : \mathbf{S_p} \rightarrow \mathbb{R}. \tag{3.1}$$

The instance $(\mathbf{S_p}, f)$ for most combinatorial optimization problems is not given explicitly, e.g. by listing all solutions and their costs. Usually, we have a data representation of an instance, and as we will see in Section 3.3 we need a *polynomial time* algorithm to verify whether a solution belongs to $\mathbf{S_p}$ and to compute the cost f for any solution in $\mathbf{S_p}$ [130].

Concerning minimization, which, unless stated otherwise, will be the case in our work, the problem is to find a solution $s_{opt} \in \mathbf{S_p}$ such that

$$f(s_{opt}) \leq f(s), \quad \text{for all } s \in \mathbf{S_p}. \tag{3.2}$$

The s_{opt} are called *globally optimal solutions.* Inputs and outputs are represented as strings over a finite alphabet. The number of bits taken to store a representation in a computer, as we will discuss in Setion 3.2.2, is taken as the *size of the problem* instance. The *solution set* is represented usually by a set of *decision variables* whose values have certain ranges or exact values. Thus, assigning values to the decision variables creates a solution. The decision variables are closely related to the representation model of the problem. If the problem belongs to *integer programming*, as we will discuss in Section 3.1.4, the decision variables are integers [329]. The space of potential solutions to a given combinatorial optimization problem instance is usually at least exponential in the size of that instance [181]. Solution representations are used to model neighbourhoods.

Definition 3.3 *A neighbourhood function is a mapping $\mathcal{N} : \mathbf{S_p} \rightarrow 2^{\mathbf{S_p}}$, which defines for each solution $s \in \mathbf{S_p}$ a set $\mathcal{N}(s) \subseteq \mathbf{S_p}$ of solutions that are close to s in some sense. The set $\mathcal{N}(s) \subseteq \mathbf{S_p}$ is the neighbourhood of solution s and each $i \in \mathcal{N}(s)$ is a neighbour of s.*

A neighbourhood structure that can be searched explicitly is called *exact.* The search for a solution is guided by the neighbourhood function as any combinatorial optimization algorithm iteratively tries to find better solutions by searching the neighbours. This is called *iterative improvement*, where the improvement leads to a solution of lower cost [3]. When such solution is found, then it replaces the current solution and search continues, otherwise the algorithm returns the current solution, which is called the *local minimum* solution.

Definition 3.4 *A solution s_l is locally minimal with respect to $\mathcal{N}$, if*

$$f(s_l) \leq f(s), \quad \textit{for all } i \in \mathcal{N}(s_l). \tag{3.3}$$

Neighbourhoods much depend on the nature of the problem, and local optimality depends on the neighbourhood function that is used. Finding efficient neighbourhood functions that lead to high-quality local minima is one of the challenges in optimization algorithms. There is no general rule for constructing efficient neighbourhoods, as there are many examples in the literature where even for the same problem there exist many different neighbourhood possibilities. The quality of the neighbourhood function is of major concern in combinatorial optimization algorithms, as the algorithms may find a poor local minimum and get stuck in low quality solutions. Therefore, much research is concerned with efficiently building neighbourhood structures.

The solutions in combinatorial optimization problems are evaluated by a cost function, which is also called the *objective function* of the problem. The goal is to find solutions with the minimum value of the objective function.

3.1.2 Classical Optimization Problems

According to [151], the root of combinatorial optimization lies in economics: the planning and management of operations and the efficient use of resources. The field of *operations research* has studied this type of problems in details. Combinatorial optimization problems nowadays can be encountered almost everywhere, from the field of economics, engineering, biology, operations research, and sciences, to specific tasks in stockmarkets, design of marketing and political campaigns, classification, positioning of satellites, VLSI circuiting, layout of mass transportation systems, scheduling, assignments of workers to jobs, coding and decoding design, efficient communication networks and so forth. The advent of powerful computers is credited for the explosion of the research interest in combinatorial optimization and the number of algorithmic solutions appeared in the last decades.

A large number of problems from a wide variety of application fields have been treated as combinatorial problems. Among all them the Travelling Salesman Problem (TSP) is the most famous [235]. The TSP owns its success to the combination of two features: simplicity of the problem definition and difficulty of solution [132]. In this problem, a salesman has to visit all the cities of the configuration space, starting from a home city and retaining the minimum length of his travel. The TSP problem belongs to a category of hard to solve problems where the computational effort grows non-polynomially with the size of the problem. Such problems belong to a category of NP-hard problems, where optimal solutions cannot be obtained in reasonable amounts of computational time [130, 131]. A problem A belongs to NP, if it can be solved in non-deterministic polynomial time, i.e. $A \in$ NP; A problem $B \in$

NP is NP-complete, if $\forall A \in$ NP: A can be represented as a B-problem and the representation $A \rightarrow B$ takes polynomial time deterministic. We now will briefly present selected examples from combinatorial optimization, trying to cover most properties of the related theory. The following paragraphs contain brief summaries of the selected classical optimization problems.

Travelling Salesman Problem (TSP)

The most famous combinatorial problem, as mentioned previously, is the Travelling Salesman Problem (TSP). An instance of the TSP is formed by a given integer $n > 0$ and the distance between every pair of the n cities, in the form of an nxn matrix $[d_{ij}]$, where $d_{ij} \in Z^+$. A *tour* is a closed path that visits every city exactly once. The problem is to find a tour with minimal total length. If $v(j)$ is a cyclic permutation v representing the city visited after city j, $j=1,...n$, then instance $(\mathbf{S_p}, f)$ is defined as

$$\mathbf{S_p} = \{\text{all cyclic permutations } v \text{ on } n \text{ nodes}\}, \tag{3.4}$$

and the cost function is a mapping of v to

$$\sum_{j=1}^{n} d_{jv(j)}. \tag{3.5}$$

A neighbourhood may be defined as

$$N_k(f) = \{p : p \in \mathbf{S_p} \text{ and } p \text{ can be obtained from } f \text{ as follows: remove } k \text{ edges from the tour; then replace them with } k \text{ edges}\}.$$

To find a *k-optimum* tour [240], which is a locally optimal solution with respect to the *k-change* neighbourhood, in an instance of the TSP, we define the function $I(t)$ for improvements, where $t \in \mathbf{S_p}$ and

$$I(t) := \begin{cases} \text{any } s \in N_k(t) \text{ such that } f(s) < f(t), & \text{if such an } s \text{ exists;} \\ \text{"No"}, & \text{otherwise.} \end{cases} \tag{3.6}$$

Meaning, that $I(t)$ searches $N_k(t)$ for a better tour s. If one solution is found, then it returns the improved tour; otherwise it returns that "no" such tour was found. Figure 3.1 shows an example of a tour f and another tour $g \in N_2(f)$ for an instance of nine cities, following a *2-change* strategy and a distance matrix determined by Euclidean distance between points in the plane.

A general iterative algorithm for finding a *k-optimum* tour is :

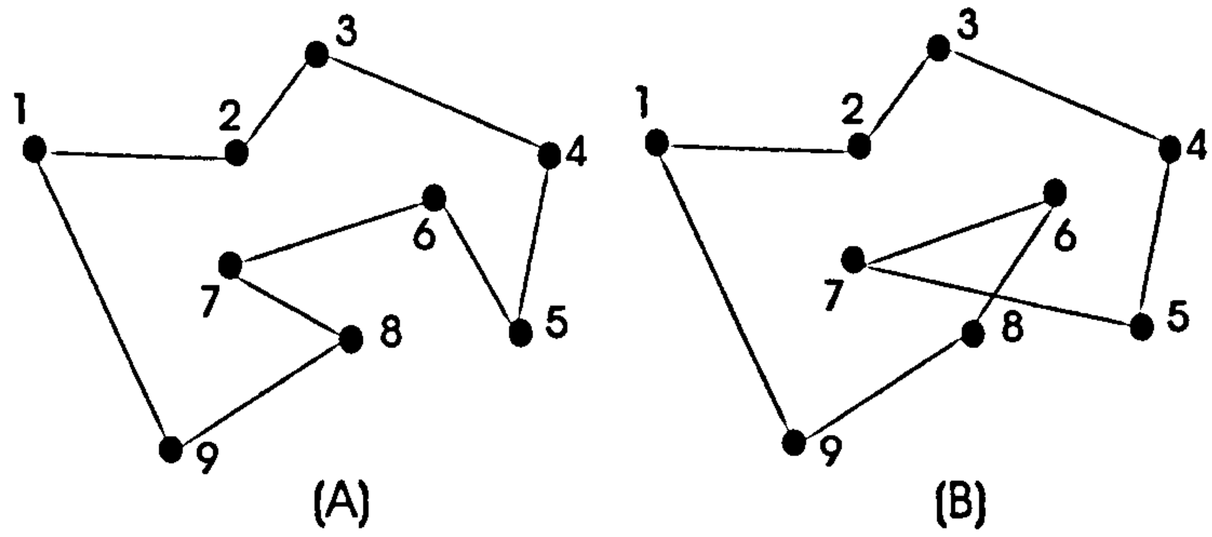


Figure 3.1: Travelling Salesman Tours.

```
procedure k-optimum
begin
  t:=initial tour
  while I(t)≠ "no"
   do t:=I(t);
  return t
end
```

Graph Optimization Problems

A number of combinatorial optimization problems come from graph theory and mainly from directed graphs, trees and networks. A *graph* G is a pair $G = (V, E)$, where V is a finite set of nodes (called *vertices*) and E has subsets of V of cardinality two (called *edges*). When the graph has directions assigned to its edges, then it is called *directed graph* and the edges are also called *arcs*. A *tree* is a special graph, which has a path between any two nodes, but without cycles. A *network* $N = (s, t, V, E, w)$ is a directed graph with a *source* $s \in V$ as starting point and a *terminal* $t \in E$ and weights $w(u, v) \in Z^+, (u, v) \in E$. Usually, the weights denote capacities or distances between the nodes. Classical combinatorial problems in graph optimization are: *the minimal spanning tree problem (MSTP)* [281], the *shortest-path problem (SPP)* [100, 118, 376], the *Max-Flow Problem (MFP)* [123], the *Min-Cost Flow Problem (MCFP)* [123, 233] and the *Graph Matching Problem (GMP)* [247].

String Matching Problems

The human DNA can be viewed as a very long string over a four letter alphabet { A, C, G, T }. String algorithms are algorithms used for matching a certain

string sequence on the DNA sequence or used to find the shortest string that is a good approximation to the original DNA string after deciphering the original string by removing overlaps of short string sequences. The latter problem is a NP-hard problem (see Section 3.3) formalized by Li [238] and referred to by Vaziravi [372] as the *shortest superstring problem*:

Definition 3.5 *Let a finite alphabet Σ and a set of n strings $S = \{s_1, ..., s_n\} \subseteq \Sigma^+$, the shortest superstring problem is to find a shortest string that contains each s_i as a substring.*

Linear programming is used by Blum et al. [54] for approximating the solution of the above problem.

Scheduling Problems

Scheduling is a class of combinatorial problems that is concerned with optimally executing a given set of tasks by using several processors, resources and constraints such us priorities, deadlines etc. [82]. The objective function is usually the total processing time between the start of the execution of the first task and the end of execution of the last task. As in other combinatorial problems, the problem is a minimization problem. Generally, this problem belongs to the category of hard problems. Many researchers have attacked the problem with feasible algorithms. Examples are Lenstra et al. [237], Hall [159], Vaessens et al. [366], and Steinhöfel et al. [351].

The Satisfiability Problem

Satisfiability is a problem in mathematical logic which decides whether a *Boolean formula* can output a true value, i.e. it is *satisfiable*. A *Boolean formula* is a Boolean expression, which is formed by the combination of *Boolean variables* x with the logical operations *AND* (denoted as multiplication $\cdot$), *OR* (denoted as addition $+$), and *NOT* (denoted as a Boolean variable x overlined $\overline{x}$). An example of Boolean formula is:

$$(x_1 + \overline{x}_2 + x_3 + \overline{x}_4) \cdot (x_1 + x_4) \cdot (x_2 + \overline{x}_3) \cdot (x_3 + x_4). \tag{3.7}$$

A Boolean formula consisting of many subformulas (called *clauses*) connected by *AND's* is called a *conjunctive normal form*. An instance of the conjunctive normal form satisfiability problem is:

Given j clauses $C_1, ...C_j$, is the conjunctive normal form $C_1 \cdot C_2 ... \cdot C_j$ satisfiable?

One solution to the problem is to try all possible assignments. However, this is not an efficient algorithm as it leads to 2^n assignments which, as we will see in Section 3.3 requires exponential time. Satisfiability is a hard combinatorial problem, and it played a crucial role in the formulation of complexity theory. It was the first problem to be shown to be NP-complete [83]. To determine the satisfiability, each of the clauses in a Boolean formula should return true (denoted as value 1). So the first clause in (3.7) should be $(x_1+\overline{x}_2+x_3+\overline{x}_4) \rightarrow 1$. Replacing value 1 with a value y and trying to maximise y turns the satisfiability problem into a maximisation problem. The formula is satisfiable, if $y \geq 1$ exists. In such a way, each clause produces a constraint for the satisfiability problem.

Some algorithms for the satisfiability problem can be found in Johnson [199], Yannakakis [397], Goemmans and Williamson [141], and Spencer [348].

3.1.3 Optimality vs Computation Time

As previously described, combinatorial optimization is concerned with problems where the set of feasible solutions is discrete. Many combinatorial optimization problems are hard to solve problems, forming a special class of problems called NP-hard problems (see Section 3.3.3)

Searching a solution for an NP-hard problem is guided by three approaches [3]:

1. Using enumerative methods that guarantee an optimal solution regardless of the computation time.

2. Using approximation methods that provide a near optimal solution in reasonable polynomial time.

3. Using some type of heuristic technique, usually tailored to the problem, without any a priori guarantee of optimal solution or computation time.

Therefore, according to the approach of solving NP-hard problems, two classes of algorithms can be defined [2, 320]: i) *exact algorithms* (also known as *enumerative* or just *optimization algorithms*, where the optimality is more important than the computation time needed to solve the problem. ii) *approximation algorithms* (also called *heuristic algorithms*), where a quick obtainable and sub-optimal solution is more important than finding the optimal solution. Examples of this class are *local search* and *stochastic* or *randomised algorithms.*

The preference of optimality at the risk of very large (probable impractical) computation time leads to working with optimization algorithms, while the

preference of algorithms that quickly reach to a good solution at the risk of sub-optimality leads to working with approximation algorithms. If the problem size is large, then enumerative methods, which are based on counting the solutions, are impractical and approximation methods seem to be the only way to find good solutions. The most significant optimization problems are in practice hard to solve by using enumerative techniques. For these problems we must resort to approximation algorithms, which usually search in a subspace of the total space and find a "good" solution in that space, rather than the best solution to the total space. The advantage is that the time requirement is small compared to enumerative methods. For example, the point-to-point shortest path problem admits $O(n^2)$ algorithms, where n is the number of nodes in the graph. In contrast, the *travelling salesman problem*, which asks for a shortest closed path that visits every node exactly once, is widely considered as unsolvable by polynomial algorithms and belongs to the class of NP-complete problems. These problems have naturally attracted the attention for many years by researchers.

We shortly present representatives of popular enumerative methods, and we describe approximation algorithms in Section 3.4, and stochastic algorithms in Section 3.4.4 which are in the focus of our work.

3.1.4 Selected Optimization Algorithms

Representatives of optimization algorithms can be found in linear programming, dynamic programming and branch-and-bound [186]. Whenever it is possible to exhaustively build up all solutions, we make the optimum choice from all of them. In this case, the algorithm that makes the best choice is called *greedy* algorithm, and the solution obtained a *greedy solution.* When the problem increases in size and the global optimum is feasible, other methods described below are more practical than greedy algorithms.

Linear Programming Algorithms

Linear programming(LP) studies problems where f in $(\mathbf{S_p}, f)$ is linear and the set $\mathbf{S_p}$ is specified by using linear equalities and inequalities. If $(\mathbf{S_p}, f)$ is non-linear, then we have the more general type of *non-linear programming problems.* Geometrically equalities and inequalities along with the objective function form linear constraints and define a convex polyhedron, where the optimum is always attained at a vertex of the polyhedron. The most well known algorithm for solving linear problems is the *simplex algorithm* [281]. Simplex algorithms work by starting from an initial solution at a vertex of the polyhedron and walking along edges to vertices of the polyhedron with

successively better values of the objective function. Although the algorithm can be guaranteed to find the global optimum, it is possible that it takes a number of steps that grows exponentially in the problem size, having a poor worst-case behaviour. Khachian [212] first proved the existance of polynomial time algorithms for LP, i.e. LP $\in$ P, whereas Karmarkar's algorithm [207] propose a projective method with proven polynomial time complexity, which is reasonably efficient compared to the simplex algorithm. However, Shrijver [329] points out that the quest for a simple polynomial algorithm for linear programming is still very much the major open problem in this field .

Linear programming, where the objective function has quadratic terms, is called *quadratic programming*, while linear programming with integer variables forms a special type of problems called *integer programming*. In contrast to linear programming, which can be solved efficiently in the worst case, integer programming with bounded variables are NP-hard problems. Integer programming and combinatorial optimization can be found in Nemhauser and Wolsey [273] and Shrijver [329]. Linear programming can also be tackled by approximation algorithms [372].

There are many books on linear programming [80, 206, 329]. Extensive discussion of linear programming and combinatorial optimization can be found in [11, 84, 246, 281].

Dynamic Programming Algorithms

In dynamic programming, the method is to iteratively break up the problem into subproblems until some simple case is reached that can be solved efficiently. In this case, optimal solutions for subproblems form the *optimal substructure*, based on which the optimal solution of the overall problem can be found. An introduction to *dynamic programming* can be found in [102]. Examples of using *dynamic programming* for classical combinatorial optimization problems can be found in [82, 167, 168].

Branch-and-bound Programming Algorithms

The idea of *branch-and-bound* is to compute upper and lower bounds on the optimum value so that the number of enumerative steps can be reduced significantly. The terms *branch* and *bounding* describe the splitting of the main problem into subproblems (branching) and the relaxation (bound) method for the calculated bounds used to construct a proof of optimality [82]. An early review of *branch and bound* programming can be found in [234]. Applications to the TSP and to scheduling problems can be found in [192] and [242].

3.2 Algorithmic Complexity

The complexity of an algorithm is a function of the size of the input and is defined as the *worst case* behaviour of the algorithm on any of these inputs.

3.2.1 Time Bounds

Most of the classical optimization problems as presented in Section 3.1.2 admit some algorithms that solve the problem. However, the time requirements is problematic as the size of the problem increases for these algorithms [281]. Time bounds may render useless an algorithm that in principle could correctly solve any instance of the problem. For example, in the TSP problem the number T of tours for n cities is

$$T = \frac{(n-1)!}{2} \tag{3.8}$$

This means that for finding the best tour in the 25 capitals of the members of the European Union, a computer has to exhaustively search the number of 3×10^{23} tours, performing at least the same number of steps. This may require years to complete, and so travellers for many years in the future have still to employ approximation methods to find a good tour around all European capitals.

For solving exhaustively the MSTP, there exist n^{n-2} spanning trees with n nodes [113], creating also an unsolvable search problem.

Improvement in running time can be achieved for most combinatorial optimization problems by using heuristics, randomisation algorithms, new data structures and methods for dealing with noisy data [11].

To deal with time bounds, the following definition is helpful:

Definition 3.6 *Let $f(n) \in Z^+$ and $g(n) \in R^+$. The rate of growth of complexity of an algorithm may be upper bounded by $O(g(n))$ if there exists a constant q such that for large enough n, $f(n) \leq q \cdot g(n)$. Then we write $f(n) = O(g(n))$.*

Using this notation, we may say that the rate of growth of the complexity of the Dijkstra algorithm [100] to solve SPP from Section 3.1.2 takes $O(n^2)$ time, while the Floyd-Warshall [118, 376] algorithm takes $O(n^3)$ time.

3.2.2 Size and Complexity

In combinatorial optimization, the input can be a graph, a family of finite sets, matrices or vectors. This input, after representing or *encoding* it in an

appropriate format, defines the input size of the problem. By *encoding* the input we mean to represent it as a sequence or *string* of symbols over some fixed alphabet, like bits, integers or ASCII characters. Then the *size* of the input is the length of the sequence or string.

The size of an integer for example, is the numbers of symbols required in order to represent it. The size to represent an integer n is defined as the smallest integer h such that $h \geq \log_B n$ with respect to basis B. We then say that the size required to represent an integer n is $\Theta(\log_B n)$, where $B \geq 2$ and Θ is defined as:

Definition 3.7 *Let $f(n) \in Z^+$ and $g(n) \in R^+$. The rate of growth of complexity of an algorithm may be bounded by $\Theta(g(n))$ if there exists constant q_1 and q_2 such that for large enough n, $q_1 \cdot g(n) \leq f(n) \leq q_2 \cdot g(n)$. Then we write $f(n) = \Theta(g(n))$.*

As

$$\log_B n = \frac{\log n}{\log B}, \tag{3.9}$$

and $\log B$ is a constant, we can finally say that the size required to represent an integer n is $\Theta(\log n)$.

The size of neighbourhoods for the TSP problem with *k-change* strategy has a perturbation of order k and therefore the size of neighbourhoods are of size $O(n^k)$. Thus, a polynomial search for exact neighbourhoods cannot exist.

3.3 Complexity Classes

In the framework of complexity theory, the aim is to find the optimum in (reasonable) polynomial time. Since in combinatorial optimization it is not possible to list all solutions and to choose the optimum, one has to exploit the special structure of the domain. It is universally accepted that polynomial time algorithms are practical algorithms. In many cases, when the objective function is too complicated, the constraints are too complex, or the problem size is too large, it might be impossible to find an optimum solution not even in exponential time [151]. The notion of *efficient* algorithms denotes algorithms that require a number of steps that grows as *polynomial* in the size of the input. Failures to develop such efficient algorithms are apparent for a number of combinatorial optimization problems, naming the TSP as the most famous one. These problems form the special class of *NP-complete* problems, which are hard computational problems. There exist a number of complexity classes according to the hardness of solving a combinatorial optimization problem. These classes are described briefly in the next sections.

3.3.1 Classes P and NP

Definition 3.8 *The class of combinatorial optimization problems that can be solved by a deterministic polynomial-time algorithm is denoted by P.*

Problems from P are candidates for having efficient algorithms, although many depends on the power k in $O(n^k)$, k =constant. Edmonts and Karp [108] showed that for the MCFP and MFP (see Section 3.1.2), one needs no more steps than a polynomial function of the number of bits of the problem description, i.e. they proved that there exist a *polynomial algorithm* for solving the problems. Papadimitriou and Steiglitz [281] show that linear problems like flow and matching problems can be solved in polynomial time. Spencer [348] investigated polynomial time algorithms for the satisfiability problem of restricted inputs.

Definition 3.9 *A problem R belongs to NP, if there exists a non-deterministic Turing machine (algorithm) that solves R on instances of size n in polynomial time $O(n^k)$.*

Krentel [220] uses the term *NP-optimization problems* to describe combinatorial optimization problems that consist of

- a set of valid instances;
- a set of feasible solutions for each instance, which is of length bounded by a polynomial in the size of input and there exist a polynomial time algorithm that decides whether a solution is in the set of feasible solutions;
- a polynomially time computable objective function;
- the property that the problem is a minimization or maximisation problem.

A decision problem can be associated with every NP-optimization problem by giving a bound on the optimal solution [372]. The answer to the decision problem is *yes*, if there is a feasible solution of cost $\leq u$, where $u \in \mathbb{R}$.

3.3.2 NP-complete Problems

Polynomial-time reductions have been used by researchers to compare algorithms. By the term *polynomial-time reduction* from problem R_1 to problem R_2 we define a polynomial-time algorithm that solves R_1 by making a polynomial number of calls of a subroutine that solves R_2. Polynomial time reductions are important because if problem R_1 polynomially reduces to R_2, and

there is a polynomial time algorithm for R_2, then there is a polynomial algorithm for R_1 [281]. Polynomial transformations leading to the *NP-complete* class as follows:

Definition 3.10 *An optimization problem R_1 is polynomially transformed to problem R_2, if given any string x, we can construct a string y within polynomial in the size of x time such that x is an instance of R_1, if and only if y is an instance of R_2.*

Definition 3.11 *An optimization problem $R \in NP$ is NP-complete, if all other problems in NP polynomially transform to R.*

We conclude immediately $P \subseteq NP$. The notion of *efficient* algorithms requires a number of steps that grows as a *polynomial* in the size of the input. NP-complete problems are characterized by the following properties [281]:

1. An NP-complete problem cannot be solved by any known polynomial algorithm.
2. If there is a polynomial algorithm for any NP-complete problem, then there are polynomial algorithms for all NP problems.

Although many researchers yield that there can be no polynomial algorithm for any NP-complete problem, nobody has been able to prove it. As stated before $P \subseteq NP$. However, if P=NP, then all the combinatorial optimization problems will become polynomial problems. If we also consider the large and very diverse collection of NP-complete problems, for which for many years of research no polynomial time algorithm could be found, it is not surprising that it is widely believed that $P \neq NP$.

The practical importance of the NP-complete class is that it is generally believed that these problems are computationally intractable, and if an algorithm correctly solves an NP-complete problem, then it will require an exponential amount of time. Practically, this will be inefficient even for small instances.

Proofs for the *NP-completeness* of various combinatorial optimization problems can be found in [83] for the satisfiability problem, in [365] for the scheduling problem, in [208] for the TSP and the three-dimensional matching problem. Knowing that a problem is *NP-complete* is useful as a starting point for to prove the *NP-completeness* of other problems. Karp [208] showed the importance of NP-completeness by presenting a diverse collection of computational problems; [130] contains hundreds of *NP-complete* problems. An impressive overview of combinatorial optimization problems along with a complexity analysis is presented in [31].

3.3.3 NP-hardness

In order to prove that a problem is *NP-complete*, we can use Cook's Theorem [83] and show that:

1. The problem R is in NP.
2. All other problems in NP polynomially transform to R.

For the second, we only need to show that a known *NP-complete* problem is polynomially transformed to R, and as polynomially transformability is transitive, this will be sufficient. The concept of *strong NP-hard* problems is defined by Vaziravi [372] as:

Definition 3.12 *A problem R is strongly NP-hard if every problem in NP can be polynomially reduced to R in such a way that numbers in the reduced instance are always written in unary.*

Most known NP-hard problems are in fact strongly NP-hard problems. Once a problem is known to be NP-complete, then the goal moves from finding an exact solution to the goal to solve it in polynomial time with approximate or good solutions. The *NP-complete* class with its intractability has led researchers to change the strategy and to find alternative methods such as *approximation algorithms, stochastic algorithms, local search* and many other *heuristics methods*, as discussed already in Section 3.1.3.

3.4 Approximation Algorithms

Since the early development of operations research, heuristics were used for solving optimization problems in an approximate way. The quality of an approximation method is measured by comparing the value of the objective function at the approximation solution with its value at the optimum solution. We define as *performance ratio* the supreme S_{app}/S_{opt} of the approximation over the optimum solution. Approximation with guaranteed *performance ratio* characterize situations in which the approximate solution is provably close to an optimal solution, for example within a factor bounded by a constant or by a slowly growing function of the input size [31].

Approximation algorithms produce solutions that are guaranteed to be a fixed percentage away from the optimum. The notion of approximation algorithms was first introduced by Garey et al. [128] and Johnson [199]. Approximation algorithms are a very active research area. Theoretical aspects, applications and recently published books related to approximation algorithms can be found in [31, 175, 372].

Approximation algorithms form a large class of algorithms that include a number of other type of algorithms. Representatives of approximation algorithms are: local search, stochastic (or randomised) algorithms, simulated annealing, genetic algorithms, tabu search. Some of these methods are based on metaphors from physics or biology. During the past decades, the number of heuristic methods for attacking combinatorial optimization problems has enormously increased. In [302], one can find a review of such methods. In the literature, approximation algorithms are also known as heuristic methods [320], but usually *heuristics* are methods without a formal guarantee of performance [281]. A collection of heuristic methods for the TSP can be found in [312].

The division between optimization and approximation algorithms is sometimes not very strict. Some algorithms can be used in both ways. Simulated annealing, for example, can be asymptotically viewed as an optimization algorithm, however, in practical implementation it behaves as an approximation algorithm [2].

Another approach in combinatorial optimization distinguishes [2] between *general* and *tailored* algorithms, where general algorithms are applicable to a wide variety of problems and may be also called *problem independent*, while tailored algorithms are problem-specific algorithms. High quality general approximation algorithms are desirable as they are fast, with high quality of results and applicable to a variety of problems. However, NFLT [391] show that there is no such algorithm that is problem independent and can approximate any combinatorial problem without having to adjust to a number of problem parameters.

3.4.1 Heuristic Search Methods

Problems that are considered too complex to be solved to optimality have to employ heuristic techniques to search for good solutions that approximate the optimal solution. Heuristics are now an established method to prove optimality within a specified tolerance as good bounds are obtained early in the algorithm, and they provide good solutions to problems for which current algorithms are incapable of proving optimality within reasonable time [180].

Constructing a feasible solution in a subset of the problem and then iteratively improving it by local moves and swaps was the first heuristic concept and is called *local search*. Local moves depend on the neighbourhood structure of the current solution, and improving moves are toward those neighbours that provide a better value of the objective function. The quality of the solution in local search is determined by the quality of the local optima found so far by

the algorithm. There are cases, where the local search algorithm is trapped into local minima that provide a bad quality solution to the overall problem. Nature inspired researchers to design heuristic methods that are based on analogies of the natural world, like properties of materials, natural selection, neural processing or learning properties that can be found in animals.

Based on these analogies from the natural world, the *simulated annealing* analogy is based on the property of annealing materials that describes the process when the disordered particles after some time are reaching a low energy ordered state. In combinatorial optimization terms, the concept is to slowly converge to a feasible solution by inserting some randomised moves that degrade the solution. The probability that such moves will be taken decreases as the algorithm progresses, simulating the cooling part of the annealing method. Similarly, *genetic* or *evolutionary* algorithms base its analogy to properties of natural mutation. In combinatorial optimization terms, every feasible solution is equivalent to a DNA string, and each such a string is assigned a value. Future generations of the population with good values are evolving and the mating of two individuals depends on their objective function values. Mating creates a new solution, which depends on the attributes of each parent. The process iteratively converges to good solutions. *Artificial neural networks* are an analogy based on models of the brain, as discussed already in Section 2.2.1. Their essential goal is to recognize patterns and to learn good responses to a given pattern.

Other methods to obtain good solutions to combinatorial problems described in the literature are *tabu search* and *Boltzmann machines*. The following paragraphs contain brief summaries of these search methods.

3.4.2 Local Search

If the problem size or lack of detailed problem information do prohibit the use of exact algorithms, *local search* is one of the alternative methods. The method provides a robust approach to obtain high-quality solutions to problems of realistic size in reasonable time. A local search algorithm starts with an initial solution and then searches its neighbours iteratively for improvements, i.e. for a better solution with improved cost. This is done by a *generation mechanism* that generates from the current solution the neighbours to be searched in the next step. We have already used in Section 3.1.2 a local search algorithm for finding a *k-optimum* tour in the TSP.

Local search is an active research area with theoretical and empirical knowledge, however, the area has not a complete theory unification [3]. Two issues related to local search are important [398]: a) the quality of solutions,

i.e. how good are the local optima for a chosen neighbourhood structure, and b) the complexity of the heuristic, i.e. how fast can the algorithm find a local or global optimum. Between the two issues there is a clear trade-off. The larger the neighbourhood structure, the better the quality of the local optima, but the worse is the complexity of the algorithm. Local search algorithms should balance between the two issues. The design of a good local search algorithm remains an experimental art, as only a few techniques and theoretical principles have been identified [3].

Local search has a number of limitations [324], e.g. local search may get stuck in poor quality local optima. Selecting a starting point and a neighbourhood structure is crucial for local search. Several researchers have investigated a multistart approach, however, with no major successes [4]. The dilemma of large neighbourhoods promising to provide better local optima against the smaller neighbourhoods that find faster the local optima leads to resolving this problem becoming very much an art to design efficient local search algorithms. Moreover, large neighbourhood structures are mostly independent of the starting solutions and their quality, while smaller neighbourhood structures very much depend on their starting solution and its quality [281]. Thus, biased or randomised starting points pose another problem. Another question in local search is how the neighbourhood is searched for local optima. One method is the *first improvement* method, where a change is accepted whenever it is found, and the method *best improvement* searches the entire neighbourhood structure for the solution with the lowest cost. Local search can generally be found to be faster for first improvement. Even the search order in the neighbourhood structure affects the efficiency. One can order the neighbourhood randomly, producing randomised local optima when first improvement is used, or the ordering can be indexed. Another question in first improvement local search is how a new neighbourhood search will restart. One method is to restart from the beginning of the ordering, and the other is to keep track of the point where the last search left off after gone around a circle of the solution; of why it is called *circular searching.*

A formalization on local search can be found in [321]. Early applications of local search for the TSP can be found in [90, 240, 304, 380] and Nicholson [275], which also covers scheduling problems. Other sophisticated attacks to the TSP with local search methods can be found in Fredman et al. [120], in Reinelt [303], and in Bentley [45]. Scheduling problems and local search algorithms based on simulated annealing, genetic algorithms and other approaches can be found in Vaessens et al. [366]. Johnson, Papadimitriou and Yannakakis [200] introduced a complexity theory for local search. Aarts and Lenstra [3] have a detailed review on local search methods in combinatorial optimization.

Local search owns its success to the fact that from local search many variations have been developed and new algorithms and methods have been generated and modelled over the past decades. Examples are the current popular *simulated annealing, genetic algorithms, tabu search* and some variations of *neural networks.* Maintaining the main feature of the classical local search algorithm, which is the iteration among neighbourhood solutions, the quality of the results can be improved if we introduce a more complex neighbourhood structure in order to have a larger solution space for exploration. Examples are the *large-step Markov chains* [257]. Also solvable classical problems can be viewed as local search problems with exact neighbourhood structure that can be searched efficiently [398]. Another improvement of the local search algorithm yields if we accept transitions that not only decrease the cost function, but also for a limited number of transitions increase the cost function. This feature of deterioration is embedded into a number of search algorithms as simulated annealing, tabu search, genetic algorithms and neural networks. The main advantage is initial solution independence and larger search in the configuration space leading to a better quality of results.

The increase of computer resources together with the use of sophisticated data structure of local search methods have led to the successful handling of many complex real world problems, establishing local search and its methods as a strong competitor of the field.

Simulated Annealing

The analogy of the physical process of annealing, in which a pure lattice structure of a solid is made by heating up the solid until it melts and then slowly cooling it down until its structure solidifies in a low-energy state, has inspired Kirkpatrick, Gelatt and Vecchi [213] and Ćerný [73] to introduce simulated annealing. Simulated annealing is a randomised neighbourhood search algorithm. Better cost neighbours, once identified, are always accepted. Also neighbours with worse costs are accepted under acceptance probability assumptions, which gradually decrease in the course of the algorithm's execution. The acceptance probability is controlled by a set of parameters that play the role of the cooling scheme, and the way of controlling the parameter is called *the cooling schedule.*

Simulated annealing has been widely used for attacking optimization problems with considerable success [2]. Simulated annealing overcomes the disadvantages of local search algorithms as it finds high quality solutions, which do not strongly depend on the initial solution.

Since the core of our work is related to simulated annealing, we will go

into more details about the method in Section 3.5.

Tabu search

Tabu search [139, 140] is based on keeping track of recent moves and making them forbidden for a given period of time. The algorithm is choosing among feasible moves that improve the objective function. If no improving moves are possible, the algorithm selects moves that degrades the solution. The next solution visited is always chosen to be a legal neighbour of the current solution with the best cost, even if that cost is worse than that of the current solution. The set of legal neighbours is restricted by a *tabu list*, which is dynamically updated during execution. It is designed in such a way that it prevents going back to recently visited solutions and defines solutions that are not accepted within the next few iterations. Tabu search is usually tailored to a given problem. This is a weakness of the algorithm, as there is little theoretical knowledge that guides the tailoring process, and one has to resort to the available practical experience [3]. Tabu search has been successfully applied to a number of problems [170].

Genetic Algorithms

Holland [179] introduced an approach called *genetic algorithm*, which uses concepts from population genetics and evolution theory to construct algorithms that try to optimise the *fitness of a population through recombination and mutation of their elements* [3]. The approach has been the basis for a number of variations like the *genetic local search* by Mühlenbein et al. [271].

Neural Networks

Over the years, many researchers have investigated the use of neural networks for solving combinatorial optimization problems. An overview of neural networks applied to combinatorial optimization is given in [245]. Many neural models have been proposed which differ in the network architecture and the computational model used. An overview of such models can be found in [2, 163, 166, 171]. Among the most famous neural networks are *Boltzmann machines* [173, 174] and *Hopfield networks* [183], which both can be defined in terms of local search; cf. Section 2.2.5.

Boltzmann Machines

Boltzmann machines are neural networks of simple computing elements, called *units*, that can have binary valued states representing either an *on* or an *off*

state. The units have real valued strength connections and a consensus function gives a quantitative measurement for the fitness of a global configuration determined by the states of all units. Boltzmann machines can be considered as a model for a massively parallel implementation of the simulated annealing algorithm [2].

3.4.3 Approximation Performance

The complexity for many optimization problems may require a computational time that grows non-polynomially with the size of the problem [2]. As a consequence optimal solutions cannot be obtained in reasonable amounts of computational time for many combinatorial problems. As mentioned at the beginning of Section 3.4, approximation algorithms are one way to tackle this problem.

The analysis of worst-case or average case error bounds is used for evaluating the performance of an approximation algorithm. The analysis may be theoretical or empirical. Empirical worst-case analysis can be the study of the robustness of an approach in practice. The performance of an approximation algorithm can be quantified by the *running time* and the *solution quality*. The running time is usually counted by the number of time units the algorithm needs to find a solution on a specific computer, or it can be determined by the number of iterations for finding a solution. The solution quality is measured by the ratio of its cost value to that of an optimal solution, or if that cannot be easily computed, then to some bound of the optimal value [3].

The theoretical concept of worst-case error bounds can be defined as follows [281, 372]:

Definition 3.13 *Let $\delta \geq 0$. An algorithm A is a δ-approximation algorithm, if on each instance I of the problem R, A produces a feasible solution $s(I)$ such that, if we denote by $\overline{s}(I)$ the optimal solution of instance I, it holds:*

$$\frac{|f(s(I)) - f(\overline{s}(I))|}{f(\overline{s}(I))} \leq \delta, \tag{3.10}$$

for all instances I.

Sometimes δ is a function of the input in order to describe the worst case behaviour of an approximation algorithm, leading, for example, to *n-approximate* algorithms, or $\ln n$*-approximate* algorithms for $O(n)$ and $O(\ln n)$ worst case behaviour, respectively. Equation 3.10 reflects the algorithmic behaviour on the most pathological instances. This may force us to further explore the combinatorial structure of the problem and to discover better algorithms for

exploiting this structure. The process of designing an approximation algorithm is a process of unravelling the combinatorial structure of the problem and then finding algorithmic techniques that exploit this structure. This is also the main idea of the *No-Free-Lunch Theory* [391].

Algorithms with high error bounds may be good algorithms for typical instances of the problem. It might be the case that we have algorithms with error bounds on typical instances of a small magnitude, e.g. 2%-5%, even though their worst case error bounds is much higher. An example of such a situation is the classical local search algorithm. For many years, the classical local search algorithm has been observed to perform well in practice, but badly in the worst case for many combinatorial optimization problems [361]. In any case, an approximation algorithm *"should be viewed as a core algorithmic idea that needs to be fine tuned to the types of instances arising in specific applications"* [372].

Approximation algorithms for *NP-complete* problems is a very active research area. A first review of related research can be found in [129]. Applications to combinatorial optimization problems can be found in [82, 159] (for scheduling problems), Goemans and Williamson [141] and Yannakakis [397] (for the satisfiability problem), and in [189, 199, 317, 318]. Recent books on approximation algorithms are [31, 175, 372]. These developments raise hopes that in the future we will have a comprehensive understanding of the approximability of NP-hard optimization problems.

There exists very little work on analysing the average performance of approximation algorithms, both in complexity and quality of approximations. Very few results exist for providing bounds on the complexity, and these are usually based on ad hoc methods [398]. The worst-case complexity of local search is also not known for many problems. For example, a very old local search algorithm like Lin's [240] for the TSP is still an open problem from the complexity point of view [2]. Yannakakis [398] proved that the local search algorithm for the stable configuration problem of neural networks takes exponential time in the worst-case. The algorithm by Goles et al. [145], where nodes flip simultaneously their state in iterations, leads to exponential time convergence. The same applies to the local search algorithm by Haken and Luby [158]. Thus, the problem of stable configurations in neural networks for solving combinatorial optimization tasks is a very difficult problem.

3.4.4 Stochastic Algorithms

In many real-world problems of combinatorial optimization, a smaller or larger extent of uncertainty about the outcome must be taken into consideration.

Uncertainty can be represented by using stochastic models, where the objective function gets dependent not only on the constraints but also on a random influence, and the aim is to optimise the objective function over the random influence. An early work exploring the use of randomisation in the design of algorithms was presented by Rabin [295]. We use the term *deterministic algorithms* to distinguish the rest of combinatorial optimization algorithms from the stochastic algorithms. Stochastic algorithms provide almost the same solutions as *deterministic* algorithms, if we have simple objective functions that can be explicitly defined or can be easily computed numerically to a desired degree of accuracy. This is not the case when dealing with NP-hard problems, where the objective function is complicated or cannot be defined explicitly. In this case, a random sampling or a simulated approach is more effective like in stochastic vehicle routing problems [48], the single machine total tardiness problem [105], and manpower planning under uncertainty [126].

The general form of stochastic combinatorial optimization problems is [152]:

$$\text{Minimise } F(x) = E(f(x,\omega)), \text{ where } x \in \mathbf{S_p}. \tag{3.11}$$

Therein, f is the objective function, ω denotes the influence of randomness (that is $\omega \in \Omega$, where (Ω, Σ, P) is the probability space for the stochastic model), and E denotes the mathematical expectation, which has not to be numerically computable as it can be estimated by random sampling. This means to draw N random *scenarios* $\omega_1, ..., \omega_N$ independently from each other, where an estimator of $F(x)$ is given by

$$E(f(x,\omega)) \approx \frac{1}{N} \sum_{\nu=1}^{N} f(x,\omega_\nu). \tag{3.12}$$

For the approximate solution of hard combinatorial problems several heuristics have been developed as described in Section 3.4.1. The combination of these heuristics with stochastic combinatorial optimization is used in a number of cases. For example, in [30] evolutionary strategies are studied for stochastic problems, whereas ant colony optimization is studied in [152], and simulated annealing in [153].

Local search with multiple random starts is another approach, which, however, is computationally expensive as after every iteration the starting point might be far away from the optimum and no information obtained from previous iteration, regarding the paths to optima, is used. Consequently, the process might return to the same point many times. In [95], a type of random local search, called *random bit climbing*, evaluates and accepts the first improved move from randomly selected neighbours.

Random search procedures have been used for optimization problems as early as in 1958 [65]. Although the simplicity of these techniques attracted researchers, the convergence of these procedures is usually extremely slow. In [258], the *adaptive random search algorithm* is introduced, where the step size of the random search procedure is optimised periodically throughout the search process. In [61], [291], a search method called *control random search* and in [298], a search algorithm called *probabilistic global search lausanne* sample points in the neighbourhood of a current solution by using probability density functions.

Stochastic algorithms can cover a large portion of the search space. Real world problems are more appropriate to stochastic processes than to deterministic processes. In [276] Ohno-Machado and Kuo stated that the hard problem of gene expression generation is a stochastic process and deterministic algorithms may not be appropriate for this domain. In [51, 210], stochastic algorithms are used for gene expression modelling.

3.5 Simulated Annealing (SA)

Simulated annealing is a local search method in combinatorial optimization that simulates the physical process of *annealing*. Simulated annealing is a high quality *general algorithm* with stochastic components, and asymptotically it can be viewed as an optimization algorithm, although in practice it behaves as an approximation algorithm. The essentials of the simulated annealing idea were proposed by Metropolis et al. in 1953 [263] and became a popular method by Kirkpatrick et al. [213] in 1983. Nowadays, simulated annealing is an active research area [319] and is competitive to any other combinatorial optimization method in obtaining good solutions in feasible time.

3.5.1 The Annealing Process

Annealing is the process of heating a solid until it melts followed by cooling it down until it crystallizes into a state with a perfect lattice. In physics, this process takes place in a heat bath.

The annealing process consists of two steps:

1. Increase the temperature of the heat bath to the maximum temperature until the solid melts.

2. Carefully decrease the temperature of the heat bath until the particles of the solid reach a ground state with a minimum energy.

When melted, the particles are arranged randomly. The issue of carefully decreasing the temperature is important, as otherwise the solid can be frozen into a metastable state rather than the ground state. Metastable states can also be reached in a process called quenching, in which the temperature of the heat bath is instantaneously lowered. If the cooling procedure is done carefully, then we avoid getting trapped into locally optimal lattice structures, and then the free energy of the solid is minimized.

The corresponding problem in simulated annealing is to find among a large number of potential solutions the solution with the minimum cost function.

3.5.2 Simulating the Annealing

The simulated annealing algorithm is based on the concept of Metropolis algorithm [263] that generates a sequence of states that are accepted if the next state has lower energy. Metropolis algorithm also generates states with higher energy that are probabilistically accepted. The acceptance criterion is called the *Metropolis criterion* and can be defined as follows:

Definition 3.14 *If i is the current state with energy E_i, then a subsequent state j with energy E_j is generated by applying a transformation based on a small distortion of the current state i. If $E_j \leq E_i$ then the state j is accepted as the current state of the Metropolis algorithm, otherwise, the state j is accepted with a certain probability given by*

$$\exp(\frac{E_i - E_j}{k_B T}), \tag{3.13}$$

where k_B is a physical constant known as the Boltzmann constant, and T is the temperature of the heat bath.

The convergence to equilibrium state can be reached if the temperature of the heat bath is lowered sufficiently slowly. The slow process of lowering the heat is essential for reaching thermal equilibrium. If the temperature is fixed, the state of solids is characterized by the *Boltzmann distribution*, which gives the probability of the solid to be in state i with energy E_i at temperature c:

$$P\{X = i\} = \frac{\exp(\frac{f(i)-f(j)}{c})}{\sum_j \exp(\frac{f(i)-f(j)}{c})}, \tag{3.14}$$

where X is a random variable denoting the current state of the solid at temperature c and $\sum_j \exp(\frac{f(i)-f(j)}{c})$ extends over all possible states. The analogy in simulated annealing is expressed by generating a large number of transitions at each temperature value.

Combining the annealing procedure and the Metropolis algorithm generates a sequence of solutions for a given combinatorial optimization problem. The next step is to define the equivalent to the energy of the state, the acceptance criterion, the neighbourhood structure together with a generation mechanism, and an equivalent to the temperature parameter.

The equivalent to the energy of the state is the cost function, which denotes the cost of a solution. The objective, as in local search, is to minimize the cost function. The acceptance criterion denotes whether a solution j with cost function $f(j)$ is accepted from a solution i with cost function $f(i)$. The acceptance criterion is based on the following acceptance probability:

$$P_t(acceptj) := \begin{cases} 1 & \text{if } f(j) \leq f(i), \\ \exp(\frac{f(i)-f(j)}{c}) & \text{if } f(j) > f(i). \end{cases} \tag{3.15}$$

The above acceptance criterion corresponds to the equivalent Metropolis criterion. The acceptance criterion can be implemented by comparing the value of $\exp(\frac{f(i)-f(j)}{c})$ with a random number generated from a uniform distribution of the interval $(0, 1]$.

The generation mechanism corresponds to the perturbation mechanism of the Metropolis algorithm. A *transition* is a combined action executed in the neighbourhood structure that results in the transformation of the current solution into a subsequent one. This action consists of two steps: i) application of the generation mechanism ii) application of the acceptance criterion. The parameter $c \in \Re^+$ plays the role of the temperature and is called the *control parameter.*

3.5.3 Markov Chains

The theory of Markov chains [115, 334] can be used for modelling mathematically the simulated annealing algorithm. *Markov chains* represent a sequence of trials, where the probability of the outcome of the trial depends only on the outcome of the previous trial. If $X(k)$ is a stochastic variable denoting the outcome of the k^{th} trial, then the transition probability for each pair i, j of outcome is defined as [3]

$$P_{ij}(k) = P\{X(k) = j | X(k-1) = i\}. \tag{3.16}$$

In simulated annealing, a trial is equivalent to a transition and the set of outcomes is equivalent to the finite set of solutions. The transition probability for simulated annealing is defined as [3]

Definition 3.15 *Given an instance* $(\mathbf{S_p}, \mathbf{f})$ *of a combinatorial optimization*

problem and a neighbourhood function N, the transition probability for simulated annealing algorithms is defined by:

$$\forall i,j \in \mathbf{S_p} : P_{ij}(k) = \begin{cases} G_{ij}(c_k)A_{ij}(c_k), & \text{if } i \neq j, \\ 1 - \sum_{l \in S, l \neq i} G_{ij}(c_k)A_{ij}(c_k), & \text{if } i = j, \end{cases} \tag{3.17}$$

where G_{ij} is the generation probability, and A_{ij} the acceptance probability.

The generation probability and the acceptance probability are defined as follows:

Definition 3.16 *The probability of generating a solution j from solution i is defined by*

$$\forall i,j \in \mathbf{S_p} : G_{ij}(c_k) = G_{ij} = \frac{1}{\lambda}\chi_{N(i)}(j), \tag{3.18}$$

where $\lambda = |N(i)|$ for all $i \in \mathbf{S_p}$, and $\chi_{N(i)}(j)$ is defined as

$$\chi_{N(i)}(j) = \begin{cases} 1, & \text{if } j \in N(i); \\ 0, & \text{if } j \notin N(i). \end{cases} \tag{3.19}$$

Definition 3.17 *The probability of accepting a solution j from solution i is defined by*

$$\forall i,j \in \mathbf{S_p} : A_{ij}(c_k) = \begin{cases} \exp(-\frac{(f(j)-f(i))}{c_k}), & \text{if } f(j) > f(i), \\ 1, & \text{if } f(j) \leq f(i). \end{cases} \tag{3.20}$$

Most combinatorial problems follow the above definitions when simulated annealing is applied. The generation probability from Definition 3.16 is independent of the control parameter c_k and uniformly distributed over the neighbourhoods $N(i)$. It follows the analogy to the Metropolis acceptance criterion in Definition 3.14.

The Markov chain is called finite, if it is defined by a finite set of outcomes. Simulated annealing is closely related to finite Markov chains. This is because in simulated annealing a trial corresponds to a transition and clearly the outcome depends only on the outcome of the previous trial. Moreover, the set of outcomes is given by the finite set of solutions. According to whether a transition probability in a Markov chain depends on the trial number k or not, we call this Markov chain *homogeneous* or *inhomogeneous Markov chain*, respectively. Häggström [155] gives a detailed theory on finite Markov chains.

3.5.4 Asymptotic Convergence

We will distinguish the conditions for asymptotic convergence according to the choice of homogeneous and inhomogeneous Markov chains. A number of

authors [2, 136, 156, 369] proved that simulated annealing finds with probability one an optimal solution, under certain conditions on the generation and acceptance mechanisms. The condition on simulated annealing formulated for homogeneous Markov chains are concentrated on the fact that it must be possible to construct a finite sequence of transitions with non-zero generation probabilities, leading from an arbitrary solution i to some optimal solution i_{opt}. This can be achieved by an infinite number of transitions to approximate Boltzmann distributions at a fixed temperature. In other words, we need to generate a sequence of infinitely long homogeneous Markov chains at descending values of the control parameter. The algorithm converges to a global minimum with probability 1, if for each value of the control parameter c_k the Markov chain is infinitely long:

$$\lim_{c \downarrow 0} \lim_{k \to \infty} P_c\{X(k) \in \hat{S}_p\} = 1, \tag{3.21}$$

where $\hat{S}_p$ is the set of optimal solutions. This implies the following process: First make an infinite number of trials, then decrease the temperature. However, this is impracticable and finite-time implementations can no longer guarantee to find optimal solution. Thus, implementations of the algorithm resort to an approximation of the asymptotic convergence. Several authors investigated possibilities to speed up the process for optimal annealing [79, 296, 346, 350].

The inhomogeneous model lowers the temperature at each trial. The convergence to optimum solutions is provided under some conditions [2, 3] of the convergence space, as the temperature is lowered at any step. Therefore, asymptotic convergence is guaranteed under these conditions for inhomogeneous Markov chain models. The sufficient conditions for the convergence are the following:

Theorem 3.18 *Given an instance* $(\mathbf{S_p}, \mathbf{f})$ *of a combinatorial optimization problem, and let* $P(k)$ *denote the transition matrix associated with equations 3.17, 3.18, 3.20, and let the following conditions be satisfied:*

1. $\forall i, j \in S_p, \exists p \geq 1, \exists l_0, l_p \in S_p,$

$$\text{with } l_0 = i, l_p = j, \text{ and } G_{l_k l_{k+1}} > 0, k = 0, 1, ...p-1. \tag{3.22}$$

2. *The control parameter* c_k *satisfies*

$$c_k \geq \frac{\Gamma}{\log(k + k_0)}, k = 0, 1, ... \tag{3.23}$$

for a sufficiently large $\Gamma > 0$ *and* $k_0 > 2$.

Then the Markov chain convergences in distribution to a vector q^* with components

$$q_i^* = \frac{1}{|\hat{S}_p|}\chi_(\hat{S}_p)(i), \forall i \in S_p, \tag{3.24}$$

or, in other words,

$$\lim_{k\to\infty} P_c\{X(k) \in \hat{S}_p\} = 1. \tag{3.25}$$

The proof of Theorem 3.18 can be found in [2, 3].

Those conditions are sufficient but not necessary. Necessary and sufficient conditions, were posted by Hajek [156]:

Theorem 3.19 *Let c_k be a sequence of values of the control parameter defined as*

$$c_k = \frac{\Gamma}{\log(k+2)}, k = 0, 1, ... \tag{3.26}$$

for some constant Γ. Then the asymptotic convergence of the simulated annealing algorithm, using the transition probabilities of (3.17), (3.18), (3.20), is guaranteed if and only if,

1. *Condition 1 of Theorem 3.18 holds true*
2. *i is reachable from j at height h, for arbitrary $i, j \in S_p$ and h*
3. *the constant Γ satisfies $\Gamma \geq D$, where D is the depth of the deepest local, nonglobal minimum.*

Proposed values for Γ and estimations of its value is the subject of numerous research efforts [25, 133, 134, 136], see [308] for an overview. Calculating the value of D can be a problem that may not be solvable in polynomial time for a number of combinatorial optimization problems [211]. Anily and Federguen [26] proved the asymptotic convergence of the inhomogeneous Markov chains for general conditions on the generation and acceptance probabilities, with respect to simulated annealing algorithms.

Implementations of simulated annealing algorithms based on inhomogeneous Markov chains are closer to real word problems. The reason is that, instead of an infinite number of transitions at fixed temperatures, we have a sequence of finite length Markov chains generated at descending values of the control parameter. This gives the advantage that, if the cooling is performed sufficiently slow, it approximates an infinite homogeneous Markov chain with guaranteed asymptotic convergence to optimal solutions, as condition 2 of Theorem 3.18 implies. This implementation is also referred to by Ingber [194] as *simulated quenching*. Therefore, the use of inhomogeneous Markov chains

comes at a cost for the guarantee of asymptotic convergence, however, its approximation guarantees optimal or near-optimal solutions for most problem instances [2].

3.5.5 SA Parameters

Using inhomogeneous Markov chains, a set of parameters called *cooling schedules* must be specified. The cooling schedule controls the convergence of the algorithm.

The focus of our work will be simulated annealing with inhomogeneous Markov chains. Therefore, for implementing simulated annealing algorithms, a set of three items need to be declared. These items are:

1. The *domain* of the problem, also known the *instance* or *configuration space* of the problem.

2. The *transition mechanism*, which generates and evaluates a new trial from an existing one.

3. Parameters that govern the convergence of the algorithm or else, the *cooling schedule parameters.*

Each of the above items needs to be analysed in more details as they consist of subsequent parameters that also need to be taken into account.

The Configuration Space

Implementing the representation of a problem, two elements have to be clearly stated by using concise and simple expressions:
a) the representation of the solution space, and
b) the expression of the cost function.
Usually, these expressions are functions that need to be minimised or maximised. Also, in this part of the simulated annealing process, we have to choose an arbitrary initial point to start the algorithm, i.e. we have to deal with the 'increase of the temperature' of the annealing process.

The Transition Mechanism

This part of simulated annealing is the most time consuming part of the algorithm. In this part, the mechanism has to be stated that generates and accepts new solutions from a current one. A well-defined structure of neighbours plays an important role for the quality of the transition mechanism and to the speed of the algorithm. The following three steps have to be executed:

1. A *generation mechanism* is applied to a current solution. The generation mechanism usually obtains new solutions from current ones by simple rearrangements that can be easily computed as permutations, inversions or swapping.

2. The difference in costs between the two solutions is calculated.

3. An acceptance criterion decides about the new solution. The acceptance criterion that is most frequently applied is the Metropolis criterion.

The Cooling Schedule Parameters

For a finite time implementation of simulated annealing, we have to specify parameters that perform a finite time approximation. This can be implemented by using inhomogeneous Markov chains that can be viewed as generated homogeneous Markov chains of finite length for a finite sequence of descending values of the control parameter. For the determination of the cooling schedule, we need to specify the following items for the *control parameter*:

1. An *initial value* of the control parameter. Usually, a large value is assigned to allow all transitions to be accepted. This corresponds to heating up the solid until all particles are randomly arranged.

2. A *decrement function* for decreasing the value of the control parameter.

3. A *stopping criterion* which specifies the final value of the control parameter.

Besides the specification of the control parameter finite sequence, we also need a finite number of transitions at each value of the control parameter in case of homogeneous Markov chains. The length of the Markov chain at fixed "temperatures" should be chosen in such a way that probabilities close to Boltzmann distribution are achieved at the end of each Markov chain.

If L_k is the length of the k^{th} Markov chain and c_k is the corresponding value of the control parameter, then the equilibrium quality achieved at the end of the entire computation depends on the changes and decreases of L_k and c_k. Large decrements of the control parameter c_k will require longer Markov chains in order to reach equilibrium. Short decrements of the control parameter c_k will allow shorter Markov chains, but require more c_k steps until it reaches the final value of the control parameter.

A cooling schedule proposed by Kirkpatrick, Gellat and Vecchi [213], which has been used in many applications of simulated annealing, is the following:

- *Initial value:*
  This should be a large value so that all possible transition can be accepted. $C_0 = 1$

- *Decrement of the control parameter:*
  Small changes in the control parameter can be obtained if the current control parameter is multiplied by a factor that is close to 1:

$$c_{k+1} = \alpha \cdot c_k,\ k = 1, 2, ... \tag{3.27}$$

  Typical values of α lie between 0.8 and 0.99.

- *Stop criterion:*
  The algorithm is terminated if the value of the cost function of the solution obtained in the last trial of a Markov chain remains unchanged for a number of consecutive transitions.

- *Length of Markov chains:*
  L_k is bounded by some constant L to avoid extremely long Markov chains for small values of c_k.

The quality of the simulated annealing algorithm depends on the speed of the convergence of the algorithm, which also depends on the choice of the parameters L_k and c_k, $k = 0, 1, ..$ There is a trade-off between small Markov chain lengths and large decrements of the control parameter c_k. Adequate cooling schedules have been proposed by [308].

3.5.6 SA Performance

Simulated annealing and local search algorithms are identical if the value of the control parameter is taken equal to zero. Therefore, simulated annealing can be viewed as a generalization of local search algorithms with proven better performance than local search algorithms. The main benefit of simulated annealing algorithms, if compared to local search algorithms, is that, besides accepting improvements in the cost function, it also accepts a limited number of deteriorations of the cost function. Initially more and larger deteriorations are accepted when the value of c is large. As the value of c decreases, only smaller deteriorations will be accepted. As the value of c approaches zero, no deteriorations at all will be accepted. Furthermore, there is no limitation on the size of an accepted deterioration, and large deteriorations are accepted but with small probability. Simulated annealing compared to local search is considerably less dependent on the topology of the cost function landscape. The potential of simulated annealing is that one has the benefits of the local search

algorithms, i.e. simplicity and general applicability, and at the same time it avoids traps in local minima due to limited acceptances of deteriorations of the cost function.

Simulated annealing can be viewed either as an

- optimization algorithm, if homogeneous Markov chains are used with infinite number of transitions performed, or as an
- approximation algorithm, if inhomogeneous Markov chains are used with finite number of transitions performed at each temperature.

The performance analysis of simulated annealing can be either a *theoretical* or an *empirical analysis*. In theoretical performance analysis, one wants to derive analytical expressions for the error and the running time for worst or average case behaviour, while in the empirical performance analysis one derives statistical averages on the error and time by running the algorithm a number of times. In empirical analysis we are usually interested in average cases.

There is a lack of theoretical average-case performance results in the literature. Such results, of course, are very important as a practitioner can estimate the expected performance of the algorithm. This is an open problem in simulated annealing [2]. On the contrary, there exist many studies of empirical performance behaviour, however, they lack the depth required to draw reliable conclusions. As Aarts and Korst [2] state *"results are often limited to one single run of the algorithm, instead of taking the average over a number of runs; the applied cooling schedules are often too simple and do not get the best out of the algorithm; results are often not compared to the results obtained with other (tailored) algorithms"*. We will try to eliminate these drawbacks in our approach.

In the case of simulated annealing, its applicability and flexibility are demonstrated by a large number of studies that used this algorithm in many different problem areas. An extended list of examples carried out at the eighties can be found in Aarts and Korst [2], showing that simulated annealing can handle a wide variety of problems. Simulated annealing outperforms almost all other algorithms with respect to efficiency, if large running times are allowed. For problems, where the running time is not an issue (for example industrial problems), simulated annealing can be the best choice. Its general applicability allows it to handle almost all known combinatorial optimization problems. The performance of the simulated annealing algorithm depends not so much on the nature of the algorithm, but on the implementation structures used for the representation of the neighbourhood structure. Sophisticated data structures that allow fast manipulations and massive parallel architectures will allow fast exploration and execution of the algorithm.

Usually, optimal simulated annealing requires an infinite number of transitions, which is equivalent with at least exponential time complexities for a close approximation of an optimal solution. This initiated investigations to speed up the convergence of optimal annealing. Sorkin [346] proved a polynomial time complexity of optimal annealing, if the neighbourhoods of a problem exhibit certain fractal properties, and the cost function is properly scaled between 0 and 1. Then the solution of cost no greater than ϵ from the optimal solution can be found in a time bounded by a polynomial in $1/\epsilon$. Aarts and Van Laarhoven [4] proposed a polynomial-time cooling schedule that leads to a polynomial-time execution of simulated annealing, but without any guarantee about the quality of the solution compared to the optimal cost. Albrecht et al. [20] proved for DNF approximations that the run-time sufficient to approach with probability $1 - \epsilon$ the minimum value of the objective function is $(n/\epsilon)^{O(\Gamma)}$ steps, where Γ depends on the maximum escape depth of local minima of the underlying energy landscape. Other approaches for improving the time complexity can be found in [79, 296].

As simulated annealing produces a high quality of results for sufficient running time, several approaches to speed up the algorithm have been proposed. For an overview of such methods, see Aarts and Lenstra [3]. Among them is parallel simulated annealing, where the execution of the algorithm is distributed to a number of parallel processors, which seems to be a promising approach [1, 3, 70]. Steinhöfel et al. [352] introduced the first, according to Aydin/Fogarty [32], application of parallel simulated annealing to job shop scheduling. For overviews on parallel simulated annealing, see [2]. Boltzmann machines [173], described in Section 3.5.7 and the LSA machine [19], described in details in Chapter 4 are such parallel simulated annealing methods that combine benefits from both neural networks and simulated annealing.

Other examples of combinations of simulated annealing with different local search algorithms for speeding up the search are the combination of simulated annealing with genetics algorithm [109, 241] and with neural networks [355].

3.5.7 Boltzmann Machines

The Boltzmann machine introduced by Hinton and Sejnowski [173] combines aspects from the field of neural computing and simulated annealing, resulting in a powerful computational model exploiting massive parallelism in a natural way. Boltzmann machines are unsupervised networks that change their network topology towards a minimum energy state by using simulated annealing. The model combines the power of neural nets and the advantages of simulated annealing for optimising the state configuration. Boltzmann ma-

chine units have bidirectional connections and binary values of states. Due to the incorporated simulated annealing algorithm, Boltzmann machines use a probabilistic transition mechanism.

Features of Boltzmann machines are:

- Boltzmann machines can be considered as a massively parallel implementation of the simulated annealing algorithm.
- Boltzmann machines can be used to solve combinatorial optimization problems and classification problems.
- The combination of neural computing, simulated annealing, and learning algorithms leads to the implementation of powerful machines, which employ the benefits of simplicity, generality and flexibility of simulated annealing, the power of massive parallelism of neural computing, and the maximisation of the confidence of reaching a globally optimum configuration.

Boltzmann machines are described in details in Aarts and Korst [2]. Boltzmann machines for solving combinatorial optimization are utilized by choosing appropriate connection patterns and strength values to map the Boltzmann machine structure onto the structure of the combinatorial optimization problem [2]. For each instance of the optimization problem, one can define a Boltzmann machine, where each configuration corresponds to a solution of the optimization problem. Maximizing a consensus function is then equivalent to trying to solve the combinatorial optimization problem. Boltzmann machine implementations are usually more efficient when tailored to a specific problem. Aarts and Korst [2] show that for the TSP problem it is hard to choose appropriate connection strengths, leading to poor performance, while for other problems Boltzmann machines performs well.

3.6 Combinatorial Optimization & Pattern Classification

Combinatorial optimization is applicable to the learning phase of the classification circle, where the goal is to minimize the classification error on the training set. The general classification model is related to estimating values of parameters from training data during the learning phase. One of the earliest applications of stochastic methods for pattern classification using a feedforward neural model can be found in [333].

Simple classification models of low dimensionality can use explicitly enumerative methods for finding optimal model parameters. For models that are

more complicated the methods from Section 2.2.8, like gradient descent, Newton descent, perceptron algorithm, backpropagation, and relaxation methods, can be used for minimizing the criterion function. For hard real world problems, where the underlying models are high dimensional and complicated, there are multiple minima, and therefore the previous methods become impractical. For instance, in [356] a combination of backpropagation with a heuristic method is proposed to escape from local minima that are caused by the backpropagation learning phase. Approximation methods and stochastic approaches can be reliable methods for sophisticated search of crucial parameters in learning, like weight vectors in a threshold circuit.

Genetic algorithms and Boltzmann machines are the two most preferable stochastic methods used for complex problems [106] so far. Boltzmann machines are used in pattern classification by clamping n nodes to the input vector $\mathbf{x}^n$, and clamping c nodes to give the output decision of the network. Therefore, the n and c nodes are the *visible nodes* of the network. The rest of the nodes are hidden nodes, and the connectivity of the nodes between them is determined by the simulated annealing algorithm, which tries to find the lowest-energy configuration, called the *state* of the Boltzmann machine. Applications of Boltzmann machines in speech recognition are investigated in [64, 290], handwritten recognition in [259], in fast learning in [148], and in computer-integrated manufacturing environments in [188].

Genetic algorithms in pattern classification are used as follows: Several classifiers (a *population*) are created, each varying somewhat from the other. An objective function for each classifier, such as the accuracy on a set of labelled examples, is calculated, called the *score* or, in biological terms, the *fitness*. According to the best scores, we retain the best classifiers in some portion of the total population, and therefore executing what is called in biological terms the *survival of the fittest*. The classifiers alter stochastically their state to produce next generations and iteratively we again retain the highest scores of the population until a desired criterion value has been reached. Several examples of genetic algorithms in pattern classification can be found in [33, 74, 142, 214, 256, 269, 279]. Genetic algorithms combined with support vector machines are used in [107] for a specific time series classification problem with lightning data, and recently in [193], for fine tuning the support vector parameters. Genetic algorithms and neural networks have been recently combined in [43].

Tabu search has very limited usage in pattern classification. A few applications where tabu search is used for training feedforward neural networks are presented in [38, 205, 338], and recently in [364].

In pattern classification the application of combinatorial optimization meth-

ods is also widely used during the feature selection phase for dimensionality reduction as in [122]. Combinatorial optimization methods can be also used when searching for optimal architectures of a classifier. In [395], the design of the optimal architecture for a neural network is formulated as a search problem in the architecture space, where two different approaches, one by simulated annealing and the other by tabu search, are investigated. The same problem is investigated in [399] by using genetic algorithms. A recent example for evolving decision trees with the combination of simulated annealing and genetic algorithms is presented in [119].

Combinatorial optimization as an effective minimisation method is the focus of our work, where a powerful optimization method like simulated annealing is combined with a classical learning method, namely the classical perceptron algorithm. This method and the corresponding results are presented in the next chapters.

Simulated annealing is an active research field that also attracts researchers from pattern classification. Recently, combinations of simulated annealing with support vector machines have been used for stereovision matching [278]. Albrecht and Wong [19] combined the perceptron algorithm with simulated annealing for a specific logarithmic cooling schedule, introducing the *logarithmic simulated annealing (LSA) machine.* The LSA machine is a new method to compute the threshold circuits by performing an epicurean-style supervised learning procedure, where several independent hypotheses are calculated from randomly chosen subsets of the total training samples. Our approach will be based on this classifier, which is described in details in Chapter 4.

There is a growing interest in combining classifiers and improving the consensus of results for a greater accuracy. According to Kuncheva [221], this interest in combining classifiers has grown astronomically in recent years. Our aim also is to investigate the performance of a new classifier, the LSA machine, that also combines classification methods. In the light of the NFLT, we will investigate the performance of the LSA machine with problem dependent parameters, trying to find guidelines for parameter tuning. The issue of guidelines unavoidable rises almost for all types of classifiers. While it is difficult to have mathematical proofs for these guidelines, they are based on plausible heuristics and have been found useful for many practical classification problems. We therefore contribute to the field of combinatorial optimization as well as to the field of pattern classification by providing guidelines and empirical results for successful applications of the combined methods. Moreover, we will investigate the performance of our classifier according to the network complexity. For this purpose, we will introduce a new recursive learning method for training hidden layers in constant depth circuits. Our findings make contributions

to a) the field of Machine Learning, as the proposed method is applicable in training feedforward neural networks, and to b) the field of circuit complexity by proposing an upper bound of hidden units sufficient to achieve a high classification rate. More details about the contributions of our work will be discussed in Section 7.5, after presenting the basic classifier of our approach in Chapter 4.

Chapter 4

The Basic Classifier of our Approach

The main idea of the LSA machine is to use a logarithmic cooling schedule to control the classification error on training samples caused by the perceptron algorithm. The search is guided by logarithmic simulated annealing (LSA), while the neighbourhood is defined by the perceptron algorithm.

The logarithmic cooling schedule applies an inhomogeneous Markov chain, whereas in most applications of simulated annealing homogeneous Markov chains are used as the underlying model. Homogeneous Markov chains are based on an infinite number of transitions at fixed temperatures, leading on one hand in theory to optimal solutions, however, on the other hand to unrealistic processes because of the infinite time involved in the algorithm. Therefore, the LSA machine, based on inhomogeneous Markov chains, has the ability to approximate optimal solutions. According to Hajeks theorem, approximations to optimal solution are guaranteed under certain conditions. However, to verify if Hajeks conditions are valid for a given configuration space, is often very difficult. Nevertheless, there are convincing results that this approach approximates optimal solutions even if it is not known if the configuration space meets Hajek's convergence conditions. Albrecht et al. [13] proved that the run-time sufficient to approach the minimum value of the objective function with probability $1-\epsilon$ is $(n/\epsilon)^{O(\Gamma)}$, where Γ depends on the maximum value of the escape depth of local minima of the underlying energy landscape. Placement problems were first analysed by this approach, however, the result is independent of the problem domain and can be applied to various optimization problems.

LSA machine as reported in [19] outperform the classical perceptron algorithm by up to 15% when the sample set is sufficiently large.

4.1 Characteristics of the LSA Machine

If the set of points $\vec{a}^j$, $j = 1, ..., m$, and $\vec{a}^j = (a_1^j, ..., a_n^j)$, can be separated by a linear function, the following convergence property can be proved for the perceptron algorithm [268]: Let $S = \{\vec{a}^j\}$ denote the "sample set" of input vectors classified as positive and negative samples. If $\vec{w}^*$ is a unit vector solution to the separation problem, i.e., $\vec{w}^* \cdot \vec{a} > 0$ for all $[\vec{a}, +] \in S$ and $\vec{w}^* \cdot \vec{a} < 0$ for all $[\vec{a}, -] \in S$, then the perceptron algorithm converges in at most $1/\sigma^2$ iterations, where $\sigma := \min_{[\vec{a},\eta]\in S} |\vec{w}^* \cdot \vec{a}|$, $\eta \in \{+, -\}$. The parameter σ has the interpretation of $\cos(\vec{w}^*, \vec{a})$ for the angle between $\vec{w}^*$ and $\vec{a}$. The value of σ can be exponentially small in terms of the dimension n.

In general, the simple perceptron algorithm performs well even if the sample set is not consistent with any weight vector $\vec{w}$ of linear threshold functions; see [127, 340]. If the sample set is linearly separable, Baum [40] has shown that under modest assumptions it is likely that the perceptron algorithm will find a highly accurate approximation of a solution vector $\vec{w}^*$ in polynomial time.

On the other hand, Höffgen and Simon [177] proved that finding a linear threshold function that minimises the number of misclassified vectors $\vec{a}^j$ is NP-hard for arbitrary sample sets (see also [176] for sigmoid functions). Variants of the perceptron algorithm on sample sets that are inconsistent with linear separation are presented in [53, 69]. For example, if the (average) inconsistency with linear separation is small relative to σ, then with high probability the perceptron algorithm will achieve a good classification of samples in polynomial time [69].

An extension of the perceptron algorithm by a simulated annealing-based search strategy is considered in our approach. The simulated annealing procedure employs a logarithmic cooling schedule $c(k) = \Gamma / \ln(k + 2)$, where the "temperature" decreases at each step.

The simulated annealing-based extension of the perceptron algorithm is activated when the number of misclassified examples increases for the new hypothesis compared to the previous one. In this case, a random decision is made according to the rules of simulated annealing. If the new hypothesis is rejected, a random choice is made among the misclassified examples for the calculation of the next hypothesis. To describe our extension of the perceptron algorithm in more details, we have to define the configuration space together with a neighbourhood relation.

4.1.1 Configuration Space and Neighbourhood Relation

The configuration space consists of all linear threshold functions with rational weights w_i represented by pairs of binary tuples each of length r: $w_i \in (\pm 1) \cdot \{0,1\}^r \times \{0,1\}^r$, $i = 1, ..., n$. We assume that the elements of the configuration space do have a fixed n^{th} coordinate, i.e. w_n represents the threshold. We denote the configuration space by

$$\mathcal{F} = \{ f(\vec{x}) \, : \, f(\vec{x}) = \sum_{i=1}^{n} w_i \cdot x_i, \; w_i \in (\pm 1) \cdot \{0,1\}^r \times \{0,1\}^r \}. \tag{4.1}$$

The neighbourhood relation depends on the given sample set S, where

$$S = \{ [\vec{a}, \eta] \, : \, \vec{a} = (a_1, .., a_n), \; a_n = 1, \; a_i = (p_i, q_i) \}. \tag{4.2}$$

where $p_i, q_i \in \{0,1\}^r$ and $\eta \in \{+, -\}$.

We define the objective function through the set of examples misclassified by the current configuration $f(\vec{x})$:

$$S\Delta f(\vec{a}) := \{ [\vec{a}, \eta] \, : \, \vec{a} \in S \text{ and } f(\vec{a}) < 0 \,\&\, \eta = + \text{ or } f(\vec{a}) \geq 0 \,\&\, \eta = - \}. \tag{4.3}$$

The objective function is given by

$$\mathcal{Z}(f(\vec{x})) := | S\Delta f(\vec{x}) | \, . \tag{4.4}$$

The set $\mathcal{N}_f$ of potential neighbours of $f(\vec{x})$ is derived from $S\Delta f(\vec{x})$ in accordance with the perceptron algorithm:

$$\mathcal{N}_f := \{ f' \mid w_i' := w_i - \frac{\sum_{i=1}^{n} w_i \cdot a_i}{\sqrt{\sum_{i=1}^{n} a_i^2}} \cdot a_i, \; \vec{a} \in S\Delta f \} \cup \{ f \}. \tag{4.5}$$

The probability of performing the transition between f and f' is defined by

$$\mathbf{Pr}\{ f \to f' \} = \begin{cases} G[f, f'] \cdot A[f, f'], & \text{if } f' \neq f, \\ 1 - \sum_{g \neq f} G[f, g] \cdot A[f, g], & \text{otherwise,} \end{cases} \tag{4.6}$$

where $G[f, f']$ denotes the generation probability and $A[f, f']$ is the probability of accepting f' once it has been generated by f.

4.1.2 Generation and Acceptance Probabilities

We use a non-uniform generation probability which is based upon $S\Delta f$ from (4.3): For $f' \in \mathcal{N}_f$ associated with $\vec{a} \in S\Delta f$ in (4.5) we set

$$U(\vec{a}) := \begin{cases} |f(\vec{a})|, & \text{if } f(\vec{a}) < 0 \text{ and } \eta(\vec{a}) = +, \\ f(\vec{a}), & \text{if } f(\vec{a}) \geq 0 \text{ and } \eta(\vec{a}) = -. \end{cases} \tag{4.7}$$

The generation probability is then defined by

$$G[f, f'] := \frac{U(\vec{a})}{\sum_{\vec{a} \in S\Delta f} U(\vec{a})}. \tag{4.8}$$

Thus, preference is given to the neighbours that maximise the deviation. The acceptance probabilities $A[f, f']$, $f' \in \mathcal{N}_f$, are derived from the underlying analogy to thermodynamic systems:

$$A[f, f'] := \begin{cases} 1, & \text{if } \mathcal{Z}(f') - \mathcal{Z}(f) \leq 0, \\ e^{-(\mathcal{Z}(f') - \mathcal{Z}(f))/c}, & \text{otherwise,} \end{cases} \tag{4.9}$$

where c is a control parameter having the interpretation of a *temperature* in annealing procedures. The actual decision, whether or not f' should be accepted for $\mathcal{Z}(f') > \mathcal{Z}(f)$, is performed in the following way: f' is accepted, if

$$e^{-(\mathcal{Z}(f') - \mathcal{Z}(f))/c} \geq \rho, \tag{4.10}$$

where $\rho \in (0, 1]$ is a uniformly distributed random number. The value ρ is generated in each trial in case of $\mathcal{Z}(f') > \mathcal{Z}(f)$.

4.1.3 Inhomogeneous Markov Chains

Let $\mathbf{a}_f(k)$ denote the probability of being in configuration $f \in \mathcal{F}$ after k steps have been performed according to (4.6),...,(4.10). The probability $\mathbf{a}_f(k)$ is given by

$$\mathbf{a}_f(k) := \sum_h \mathbf{a}_h(k-1) \cdot \mathbf{Pr}\{h \to f\}, \tag{4.11}$$

where $\mathbf{Pr}\{h \to f\}$ is from (4.6). The recursive application of (4.11) defines a Markov chain of probabilities $\mathbf{a}_f(k)$, where $f \in \mathcal{F}$ and $k = 1, 2, \ldots$. If the parameter $c = c(k)$ in (4.9) is a constant c, the chain is said to be a *homogeneous* Markov chain; otherwise, if $c(k)$ is lowered at any step, the sequence of probability vectors $\vec{\mathbf{a}}(k)$ is an *inhomogeneous* Markov chain, cf. Section 3.5.

We consider a special type of inhomogeneous Markov chains only. The motivation for this choice is based upon the convergence properties of the two

types of Markov chains: Convergence propositions about homogeneous Markov chains rely on an infinite number of transitions at fixed "temperatures" c. The probability distribution approached in the limit is the Boltzmann distribution $e^{-z(f)/c}/F$, where F is a normalisation value. If $c \to 0$, the Boltzmann distribution tends to the distribution over optimum configurations, cf. Section 3.5.4. In practice, however, it is infeasible to perform an infinite number of transitions at fixed temperatures. The convergence analysis of inhomogeneous Markov chains avoids the intermediate step, and in our approach the "temperature" $c(k)$ changes in accordance with

$$c(k) = \frac{\Gamma}{\ln(k+2)},\ k = 0, 1, \ldots. \tag{4.12}$$

The choice of $c(k)$ is motivated by Hajek's theorem [156] on logarithmic cooling schedules presented in Section 3.5.4. We denote by $\mathcal{F}_{\min}$ the set of optimum solutions (minimizing the classification error). Basically, Hajek's theorem states

Theorem 4.1 *Under some natural assumptions about the configuration space $\mathcal{F}$ and the neighbourhood relation $\mathcal{N}_f$, the asymptotic convergence*

$$\sum_{f \in \mathcal{F}_{\min}} a_f(k) \underset{k \to \infty}{\longrightarrow} 1 \tag{4.13}$$

of the stochastic algorithm defined by (4.6),...,(4.10) is guaranteed if and only if Γ from (4.12) is lower bounded by the maximum value of the minimum escape height from local minima.

Thus, the speed of convergence associated with the logarithmic cooling schedule (4.12) is mainly defined by the value of Γ. Since we deal with sample data and a relatively complicated neighbourhood relation (4.5), we cannot verify the applicability of Theorem 4.1. Nevertheless, previous research [14, 19] encourages us to employ (4.12) in stochastic local search procedures.

4.2 The Epicurean Learning Method

This approach goes along the lines of a learning method which has been called Epicurean learning by Cleary et al. in [81] (with reference to [325]), motivated by Epicurus' paradigm that all hypotheses fitting the known data should be retained [135, 239]. The learning method in the LSA machine is equivalent to training each threshold unit, which in our work is also called *perceptron*, with a set of examples. Given a dataset D of a classification problem, let T be our training set, drawn from the available data D. We assume that T consists of positive and negative examples, i.e. $T = T^+ \cup T^-$. Then for a class

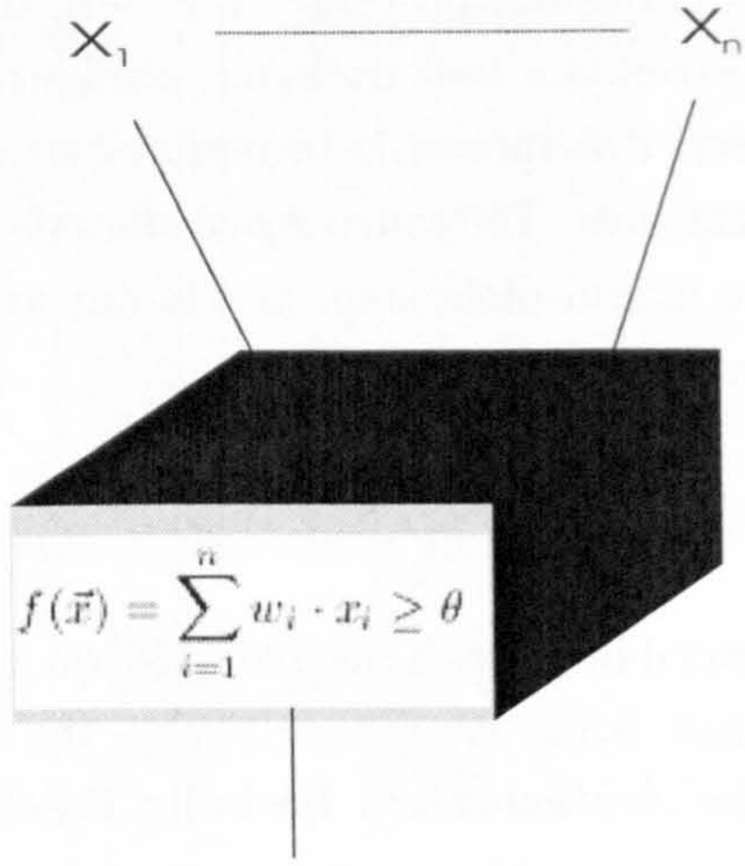


Figure 4.1: Depth-one LSA Machine Architecture.

i $(i = 1, .., m)$, let $C_i^+ = \{x \in D \mid c(x) = i\}$ be the set of positive examples in D, and $C_i^- = D - C_i^+$ be the set of negative examples of class i in D.

Each threshold unit $j (j = 1, .., t)$ is trained by a randomly selected training set $T_{i,j}$, which is defined as:

$$\begin{aligned} T_{i,j} &= T_{i,j}^+ \cup T_{i,j}^-, \text{ where} \\ T_{i,j}^+ &\subseteq C_i^+, |T_{i,j}^+| = \alpha |C_i^+|, \alpha \in (0,1), \text{ and} \\ T_{i,j}^- &\subseteq C_i^-, |T_{i,j}^-| = \beta |C_i^-|, \beta \in (0,1). \end{aligned}$$

The size of the training set and the proportion of positive and negative examples in this set are significant parameters of the learning procedure. Since we usually consider circuits with $t > 1$ threshold units, α and β determine the "multiplicity" of a single example, i.e. how often an example appears on average in the randomly selected training sets.

4.3 Topology

The core of the LSA machine is based on depth-one threshold circuits, as shown in Figure 4.1. Following notations from the circuit complexity theory we will not count the input layer as a depth-one layer. Therefore, depth-two circuits are constructed as shown in Figure 4.2. We recall that our depth-one threshold circuit is considered as a depth-two neural network based on the neural model of Section 2.2.2.

The first layer of our circuit is trained over t randomly selected sample sets

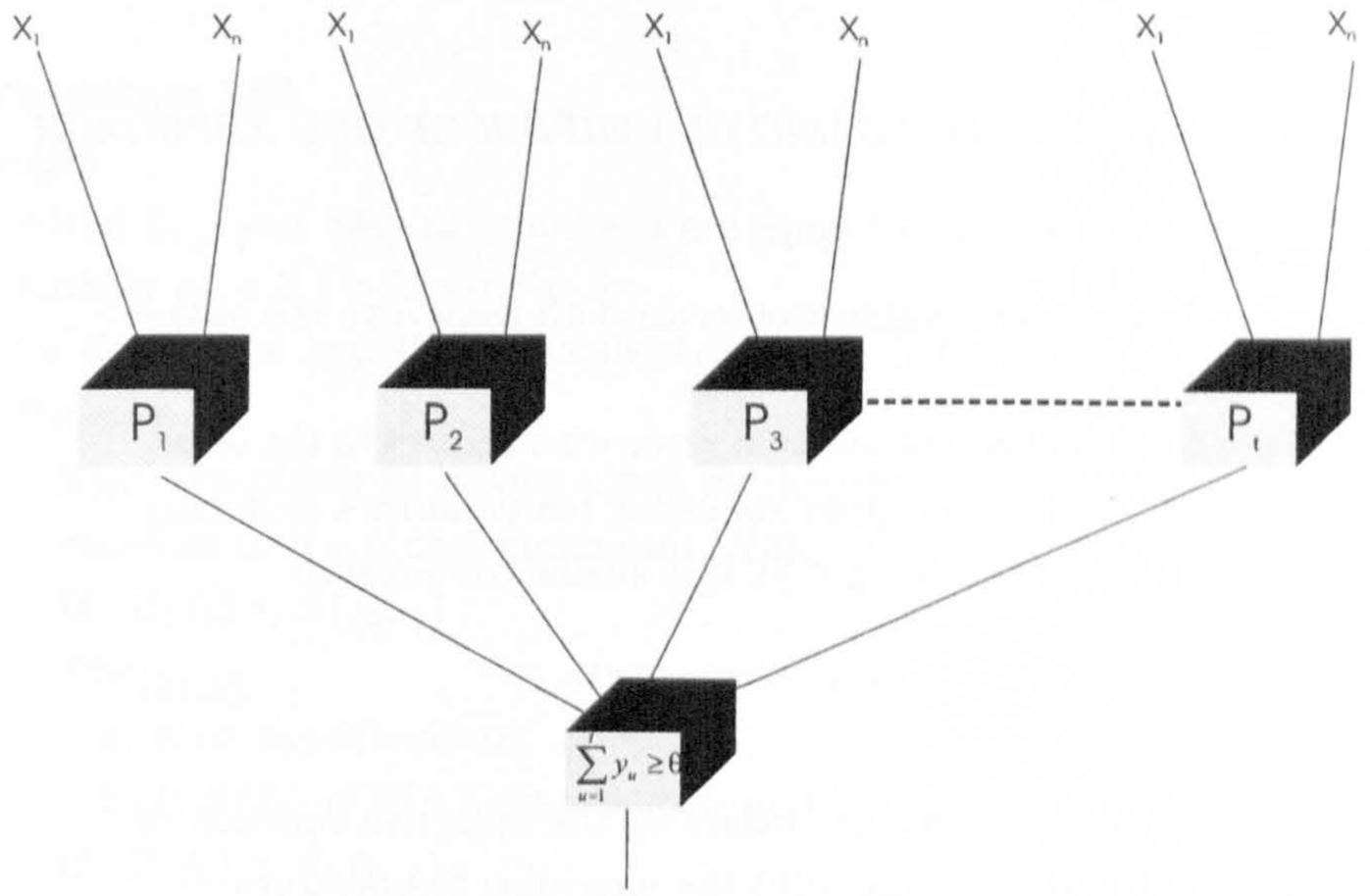


Figure 4.2: Depth-two LSA Machine Architecture.

of the training set T. Independent hypotheses $f_1(\vec{x}), ..., f_t(\vec{x})$ of the type

$$f(\vec{x}) = \sum_{i=1}^{n} w_i \cdot x_i \geq \theta \tag{4.14}$$

are calculated for each threshold unit.

The output gate at the second level calculates $|\{j : f_j(\vec{x}) = 1|/t$ where t is the number of threshold units at layer one and $|\{j : f_j(\vec{x}) = 1|$ denotes the total number of threshold units of layer one that decide for a positive example. Therefore, outputs at first layer are collected and the class decision is finally denoted by a voting function on the output of the second layer.

4.4 Key Features

LSA machines can be characterized by the following three key features:

Key Feature 1:

The first layer of the circuit is computed by a combination of the perceptron algorithm and logarithmic simulated annealing with a heuristic of choosing the elements that are far away from being correctly classified.

To these elements higher probabilities are assigned for being our next hypothesis. A new hypothesis is accepted if one of the follow-

ing is valid:
A) it produces lower classification error with respect to the objective function, or
B) it produces higher classification error with respect to the objective function, and for the given annealing temperature a uniformly randomly selected number $\rho \in [0,1]$ is smaller or equal to

$$e^{-(o(w_k)-o(w_{k-1}))/c(k)}, \tag{4.15}$$

where $o(w_k), o(w_{k-1})$ are the values of the objective function of hypotheses k and $k-1$, and $c(k)$ the annealing temperature.

Key Feature 2:

The second key feature of the LSA machine is the logarithmic cooling scheme that is based on Hajek's theorem. Applying this feature to the LSA machine, we manage to use inhomogeneous Markov chains of finite length to restrict the classification error.

The annealing temperature is defined by Hajek's theorem [156] as given in Theorem 3.19 and Theorem 4.1 .

Key Feature 3:

The third key feature of the LSA machine is the training procedure that uses an Epicurean-style learning. Each threshold function is calculated from a random selection of positive S^+ and negative S^- samples out of the entire training set:

$$|S^+| = \alpha \cdot |T^+|; |S^-| = \beta \cdot |T^-|; \tag{4.16}$$

Each substructure of the classification circuit of the LSA machine is trained by a set of randomly choosen training samples out of the entire training set and thus performing an Epicurean-style learning. Issues arising from these settings are dealing, e.g., with the number of examples used for training and testing, the choice of α and β, and the choice of Γ.

4.5 The LSA Algorithm

For a given sample set associated with a single threshold unit, the *LSA algorithm*, as described in [19], has the following pseudocode:

Procedure LSA
begin
initial f_{k-1} and best f_b hypothesis are equal to zero for $k \geq 1$
initially $w_i = 0, i = 1, ...n, \theta = 0$;
for the current hypothesis calculate deviation $U(\vec{a})$; see Eq. 4.7;
Repeat
Next hypothesis f_k is determined by random choice among the elements of $N_{f_{k-1}}$ that maximizes $U(\vec{a})$;
If $\mathcal{Z}(f_k) \leq \mathcal{Z}(f_{k-1})$;
Then
a) Next hypothesis=f_k ;
b) If $\mathcal{Z}(f_k) \leq \mathcal{Z}(f_b)$, then update best hypothesis $f_b = f_k$;
If $\mathcal{Z}(f_k) > \mathcal{Z}(f_{k-1})$;
Then
a) Draw $\rho \in [0,1]$, a uniformly distributed random number;
b) **If** $e^{-(\mathcal{Z}(f_k) - \mathcal{Z}(f_{k-1}))/c(k)} \geq \rho$
Then Next hypothesis=f_k ;
Else Next hypothesis=f_{k-1};
$k := k + 1$;
Until $\mathcal{Z}(f_k) = 0$, or k=predefined K of maximum steps;
return hypothesis $f_k, \vec{w}(k)$
end

The algorithm implements an inhomogeneous Markov chain as the temperature is lowered at each step. LSA algorithm performs actually, a modified type of backpropagation for training threshold circuits. Modificd types of backpropargation have been also tried in a number of studies. For example, Tom [360] uses a hybrid type of elementary gates that are linear combination of analog (sigmoid) and discrete (threshold) functions depending on the values of a heuristic parameter, i.e. for a specific range of values a gate can be purely analog (sigmoid function), while for other values it becomes purely binary. Bartlett and Downs [36] describe and mathematically justify a method of representing weights of threshold functions in depth two circuits as random variables with smooth distribution functions, which allows to train such circuits by methods similar to backpropagation. The approach was extended by Corwin et al. [87] and Goodman and Zeng [146], where in their study, the learning scheme is based on the standard back-propagation, but with "pseudo-gradient" descent, which uses the gradient of a sigmoid function during training in order to determine output values of gates, instead of

binary truth values produced by threshold functions, i.e. the outputs of gates are real positive numbers, either a or b, with $a, b \in (0, 1)$. In [87], a learning scheme is proposed that uses parameterised sigmoid functions in order to alter the shape of the sigmoid during training. Further methods and improvements of learning schemes are presented in [23, 255, 288, 289]. For example, Plagianakos et al. [288, 289] introduced an evolutionary learning scheme that utilizes differential evolution strategies, including a parallelised version of the approach. Anastasiadis et al. [23] devised a new globally convergent modification of the Resilient Propagation-Rprop algorithm, where the convergence analysis ensures that the adaptive local learning rates of the Rprop's schedule generate a descent search direction at each iteration.

4.6 Parameters

When implementing the LSA machine, a number of parameters must be specified. Trying to categorize these parameters, we might divide them into parameters of the network structure, parameters of the simulated annealing method, and parameters of the learning method.

4.6.1 Network Structure

The number of computational threshold units used in a depth-two LSA implementation determines the size of the network. At this stage, the depth is not considered as a free parameter. Once the depth-two LSA machine is fine-tuned, we will move towards increasing depth to depth-four, and we will investigate the depth vs size problem. As the complexity increases, several new issues evolve related to the network topology complexity and to the learning algorithm complexity for levels at larger depths. What topology should be used in depth-four? How will units at higher levels be trained? To answer these questions at least partly, a new learning method will be introduced in Chapter 6 for training units at higher levels.

4.6.2 Simulated Annealing

.

Parameters that must be specified in every simulated annealing application are the crucial parameter Γ of the cooling schedule and the length K of the inhomogeneous Markov chain, i.e. after K steps the process stops with the best so far weight configurations, or it stops earlier if the threshold unit reaches a zero classification error in the training set.

4.6.3 Learning Procedure

Here, the focus is on the selection of the α and β parameters of the third key feature. This choice is identical to choosing the size of positive and negative examples in the randomly selected subsets. The values of α and β determine how often a single example is used in randomly selected training subsets. Another important parameter is the division of the original dataset into training and testing examples. When implementing a larger depth LSA machine, the complexity of parameter settings of the learning procedure increases.

4.6.4 Research Issues

We will try to answer a number of open questions related to the LSA machine. Questions that we will investigate are:

- What is the impact of the number of threshold units, i.e. the impact of the size of the neural network on the classification error?
- What is the impact of the depth of the neural network on the classification error?
- Is there any general rule for an appropriate choice of the value of the Γ parameter in relation to the size of the training samples?
- How many times ("multiplicity") should a single training sample be used for training the network in order to avoid overfitting?

4.7 Defining Experimental Parameters

The parameters of the method described in Section 4.6 are parameters of the i) learning process, ii) simulated annealing process, and iii) network structure. We define here our free parameters, which are $\{|S_L^j|, K, \Gamma, P\}$, where: i) $|S_L^j|$ is related to the learning process of a single (the j-th) threshold unit, ii) K and Γ are related to the simulated annealing process, and iii) P is related to the network structure and to the total number of gates determining the final discrimination function.

1. The Learning Process Parameter $|S_L^j|$: For the learning process, we divide the total number of available data $|S|$, as in many Machine Learning methods [297] into 2/3 of the data to be in the training set S_L, and the rest 1/3 of the data to be in the test set S_T for estimating the classification accuracy. The same partition of data is used for the SJGS datasets by Rampone in [297]. In order to train the unit j, i.e. to calculate a

function of the type $f_j = \sum w_i \cdot x_i \geq \vartheta$, where $i = 1,..,n$ is the number of input gates in threshold unit j, and $j = 1,..,P$, we use a sample set S_L^j, which is randomly sampled from S_L, associated with threshold unit j. The number of examples in $|S_L^j|$ determines the sample complexity problem, which is related to PAC learnability and VC-dimension, as described in Section 2.3.1. We want $|S_L^j|$ to be as large as possible in relation to the generalisation error, which should be at the lowest possible level. For the size complexity problem related to $|S_L^j|$, we use integer steps $x_0 \geq 2$ for calculating the number of random examples $|S_L^j| = |S_L|/x_0$. Therefore investigation of parameter $|S_L^j|$ is identical to find a value x_0 that obtains best classification result.

2. The Simulated Annealing Parameter K: Parameter K determines the number of steps when searching for better solutions in the simulated annealing process. While this number of steps determines also the learning effort, parameter K, although a parameter of the simulated annealing process, is also a parameter of the learning process of our classifier. Parameter K should be sufficiently large for searching the solution space in order to achieve a small learning error. Experimental results from [12] suggest $K \leq (m/\delta)^{\Gamma}$ for the number of transitions in the local search procedure defined in Section 4, where $1 - \delta$ is a confidence parameter, i.e. after K steps the probability to be in an optimum solution is at least $1 - \delta$, and m is the size of neighbourhoods. However, this might result to extremely high values for K, even for small values of Γ.

3. The Simulated Annealing Parameter Γ: The parameter Γ is difficult to estimate a priori. Rough estimations can be derived from the size $|S_L^j|$ of the sample set, e.g. $\Gamma = |S_L^j|/3$, which, however, may result in extremely large values for K as described previously. On the other hand, one can try to estimate Γ by preliminary experiments on $|S_L^j|$, i.e. we estimate the depth D of the deepest local minimum as described in Hajek's Theorem 3.19: If the current best value $\mathcal{Z}_b$ of the objective function is recorded, one can monitor the maximum increase D of the objective function before the next improvement of $\mathcal{Z}_b$ occurs. The maximum value of D, which we will call the *maximum escape depth*, observed over a sufficently long time period can be used as an upper bound for Γ.

4. The Network Parameter P: Parameter P determines the size of the depth-two circuit and defines the circuit complexity problem. It is also related to the final discrimination function $F = \sum_{j=1}^{P} f_j$. Since parameter P is the number of computational units used in the network, one

can relate P to hardware resources in a parallel implementation of the circuit. Therefore, bounding the number of perceptrons is desirable and important in terms of time and available computational resources.

4.8 Learning and Convergence Parameters

During our experiments, we monitor a number of other parameters for the purpose of studying the behaviour of parameters in relation to the average classification error e_T. This value is calculated from m experiments with a fixed set of parameters $\{|S_L^j|, K, \Gamma, P\}$. Our auxiliary parameters are related to *learning* and *convergence* properties of the learning process.

Related to learning properties, we monitor for each threshold unit the learning error, and we are interested in the average learning error e_L, which comes as the average taken for all threshold units and all m experiments. Minimizing the learning error e_L is the decisive task in the sample complexity problem, according to PAC learnability. Other parameters related to learning properties that we monitor are the number U_0 of threshold units that are learned with zero error and the number U_1 of threshold units that are trained with error $\leq \epsilon$, where in our case $\epsilon = 1\%$.

Related to convergence properties, we monitor for each threshold unit the last step k where an accepted new hypothesis has lowered the training error. The value K_{max} is the maximum of these k with respect to all threshold units and for $m = 10$ experiments. Then, K_{max} can be compared to K giving an estimation of a training confidence δ for adequately searching the neighbourhood structure of the simulated annealing method for minimum values. Let $\Delta K_{max} = (K - K_{max})/K$ be the difference of K_{max} from K. ΔK_{max} approaches zero if $K_{max} \approx K$. The higher ΔK_{max}, the higher the training confidence δ for adequately searching the neighbourhood structure. Therefore, K_{max} can also be considered as an estimation of the quality of the predefined length K of inhomogeneous Markov chain, where, if $K_{max} \not\approx K$, we may, in generally, consider that adequate search is achieved. In the same way, K_{aver} is the average of all k steps and all m experiments, giving an estimation of the actual transition steps for most of the threshold units, denoted by $\Delta K_{aver} = (K - K_{aver})/K$.

Finally, an important parameter of the convergence is the overall training time. We have seen in Section 3.5 that simulated annealing converges to optimality if time is allowed to approach infinity. However, polynomial time algorithms with good approximations of optimality are desirable and therefore we monitor the speed of our classifier for the evaluation process. The time ($T(sec)$) is measured on a 3.2 GHz CPU unit with 384 MB RAM.

4.9 Methodology

Investigating problem dependent parameters requires a large number of experiments. Finding the optimum parameters by enumerating the parameter set for various values of the four parameters $\{|S_L^j|, K, \Gamma, P\}$ is practically intractable due to the large number of experiments. For example, the number of experiments E needed for just one free parameter, if the rest of the parameters are fixed, assuming a partition of the parameter space into just 10 values and m independent random trials (so that the generalisation error will be the average of the trials) with x datasets results in $E = 10 \cdot m \cdot x$, where for our experiments we have $m = 10$, $x = 5$ and therefore $E = 500$ experiments for just one free parameter.

We try to reduce the number of free parameters in order to focus on specific parameters that may effect the generalisation performance of our method. Thus, in our experiments the number of examples $|S_L|, |S_T|$ used for training and testing our classifier have size $S_L = 2/3 \cdot S$ and $S_T = 1/3 \cdot S$, respectively, where $S = S_L \cup S_T$. The division of the available data into two disjoint sets of the above size is common practice in Machine Learning [269, 297]. Other methods for defining the learning and testing sets, like the 10-fold cross validation methodology, have not been considered as this would significantly increase the number of required experiments.

Furthermore, in order to reduce the free parameters, we maintain within the sample sets used for training each unit the original distribution of positive and negative examples of the dataset. Therefore, in our case we have $\alpha = \beta$ for the parameters α, β defined in Section 4.4. In this way we can view the associated set of random samples for training each unit as a smaller classification problem, which has the same proportion of positive and negative examples as the larger classification problem.

Since the optimum parameter setting cannot be reached, we will try to demonstrate the impact of each one of the four free parameters by focusing on experiments for each parameter for fixed values for the rest of parameters. Prior knowledge of the learning algorithm's behaviour on the parameters has been acquired by a number of experiments carried out before. Examples of such experiments can be found in [14, 229, 230, 231]. Thus, to fix the rest of parameters not arbitrary values are assigned, but values reasonably close to accepted values for a good approximation of the best classification accuracy. Therefore, the setting of the parameters demonstrated here will be a fine-tuning method for each parameter and each dataset.

The $x = 5$ datasets for setting up the problem dependent parameters are described in Section 5.1. The selection of the specific datasets was based on the

following criteria. All datasets are well known datasets from the UCI Repository, used by many researchers as benchmark datasets to test their methods. Therefore there exist a large number of experiments for these datasets allowing us to have a comparison of our method with the best results from the literature. Also we tried to have datasets that consist of different types of data like nominal data as in IE, EI, and "Neither"" datasets, integer data as in WBCD and data with real values as in Pima Indians dataset. Moreover, while WBCD and Pima Indians are binary classification problems having two classes, IE, EI, and "Neither"" datasets are part of a multiclassification dataset, the Splice-junction Gene Sequence Dataset, treated as three binary classification problems as described in Section 2.2.7. Thus, the selected datasets are related to both binary and multiclasification problems.

Nominal data in a preprocessing step are encoded to binary values as described when presenting each dataset. For each of the $m = 10$ experiments for each step of each parameter, different sets of S_L and S_T are used, where again the proportion of positive and negative examples in both S_L and S_T sets follows the proportion of the whole dataset. For each experiment, S_T is constructed prior to S_L by randomly sampling examples from the original dataset. Each unit is trained by examples randomly selected from S_L. The output values of each of the $m = 10$ experiments are averaged and this average determines the reported classification accuracy e_t of each step of the parameter. While investigating each parameter we try to inter-relate best classification accuracy with monitored properties of the problem. Tables and plots can be helpful and are provided where appropriate.

Initially we optimise $|S_L^j|$ by fine-tuning x_0 parameter, for setting up the size of samples for training each unit. We use $P = 100$ perceptrons, $K = 25,000$ and $\Gamma = D_{cp}$, where cp corresponds to each of the classification problems. Then x_0 is fixed in values that obtain for each dataset the best classification rate. Afterwards, length of inhomogeneous Markov chain K is fine-tuned. We also run experiments to compare the importance of parameters x_0 and K (see Section 5.4). Parameter Γ is investigated afterwards for the fixed values of x_0 and K for each dataset. Finally we run experiments varying the number of perceptrons P to investigate the impact of the size of the network to the classification accuracy. The results for parameter P for depth-two are used in next chapter to investigate the circuit complexity problem.

Investigating the circuit complexity problem we need to work on larger depths and we introduce a new learning method that is based on further partitioning the training set S_L to two disjoint sets of approximately the same size, one for training depth-two as previously, where after all depth-two units have been trained all their weights have the fixed calculated values from the

learning procedure, and the other for generating new samples for training next depth after applying it to the trained depth-two units. In this way training on next depth is based on generated data from the outputs of previous depth and the aim in of our method is to learn the units that perform well and increase their significancy while at the same time decrease the significancy of those units that provide less accurate results. A detailed analysis of the new learning approach is presented in Chapter 6.

For investigating the circuit complexity problem we will compare the results for depth-two circuits with results of depth-four circuits of the same size. If the problem is NP-hard then we want to see whether a small increase in depth achieves better classification accuracy than depth-two, i.e to decide on the depth vs size problem. If the problem is non-NP-hard then it is expected that a small increase in depth reduces significantly the size of the network, i.e the size changes from an exponential size in depth-two to a polynomial size in depth-four. If this occurs in any of our dataset then this dataset is non-NP-hard. We will also compare our results with classical theory from threshold circuit theory to estimate the size of the network for achieving high classification accuracy.

The experiments on depth-four rely on the results of the previous fine tuned parameters and we use the values obtained for best classification rates in each dataset and for the three parameters $\{x_0, K, \Gamma\}$, which we call this set of values as set-1 . We also use another set of values for parameters $\{x_0, K, \Gamma\}$, namely set-2, of non-optimized values, where x_0 is estimated from experiments with $P = 50$ perceptrons in depth-two circuits, where inhomogeneous Markov chain was set to $K = 20,000$ and Γ was set to one third of the size of the sample set $|S_L^j|$.

After investigating the circuit complexity problem we will provide rules for setting up values related to the problem of the network size, the size of samples used for training each unit, the length of inhomogeneous Markov chains and the constant Γ of the cooling schedule. These values are based on experimental results on five datasets. To evaluate our findings derived from the initial five dataset we will use ten additional datasets, which are described in Section 6.8, and we will apply our paradigms for parameter settings. Again the additional datasets come from the UCI Repository and there exist in the literature many reported results. Thus, we can compare our experimental results to those reported in the literature. We focus on the applicability of the estimated network size and the applicability of the sample size complexity rule. Our sample size complexity rule requires a number of experiments where we monitor three properties (monotonicity, U_0 and K_{max}. Therefore we run three set of experiments on new datasets one for each criterion of the sample

size rule to evaluate the importance of each of the three monitored properties (see Table 6.8). Finally, to evaluate the applicability of the estimated network size result, we will run further experiments on the ten datasets to compare the results of the estimated network size N with results from two different networks, one with smaller size (half of the estimated network size, $N/2$) and one with larger size (double of the estimated network size, $2N$); cf. Table 6.9.

Chapter 5

Parameter Settings

In this chapter, we will investigate the detailed performance of our classifier by setting the general parameters and trying to relate them to problem dependent parameters.

The *No Free Lunch Theorems* established a new way of studying algorithms; cf. Section 2.4.3. Before NFLT, when a new algorithm was designed it was tested against as many problems as possible and compared to as many existing algorithms to evaluate it. As we have seen in Section 2.4, there exists no universal algorithm that solves all type of problems most efficiently. Moreover, no algorithm under no assumptions related to the problem performs better than any other, even against "random guess" as the chosen heuristic method. In this chapter, the parameters of the adopted learning algorithm will be explored. The setting of the parameters, has to be adjusted to properties of the dataset. We have introduced two types of such properties, one type related to learning properties of the sample set, and the other type related to convergence properties of the sample set, as described in Section 4.7.

The main parameters that will be considered in this chapter for their impact on the classification accuracy are:

a) the number of examples $|S_L^j|$ (randomly chosen from S_L) used in the sample set for training a threshold unit j, where $j = 1, .., P$;

b) the training effort in terms of the length K of inhomogeneous Markov chains;

c) the parameter Γ of the cooling schedule of simulated annealing;

d) the number of threshold units P in the network.

In this chapter we will focus only on depth-two networks. Networks of larger depth for the impact of depth in the LSA machine are investigated in Chapter 6.

5.1 The Datasets

Our datasets are from the well known *UCI Machine Learning Repository* [262], used by many researchers as a benchmark collection of datasets for testing their classification algorithms. All datasets that we use have missing values in some of the positions. The problem of handling missing data has been of particular interest to many researchers [243, 322]. Suggested methods to deal with missing data range from naive methods of just discarding or just ignoring the samples that have missing values, to methods of replacing the data with plausible values or by using more sophisticated methods to calculate values from other data in the set.

5.1.1 Splice-junction Gene Sequences Datasets (SJGSD)

Splice junctions are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as EI sites), and recognizing intron/exon boundaries (IE sites). Additionally, a third class is introduced which is referred to as "Neither". Given a position in the middle of a sequence window, 60 DNA sequence elements are used to decide if this is an IE, EI, or "Neither" class. The database consists of 3190 vectors representing 60 attributes with nominal representation. The class distribution is: 25% for IE (767 instances); 25% for EI (768 instances); and 50% for "Neither" (1655 instances). In order to discriminate between the three classes, we introduce three databases, each related to a single class as positive examples:

1. **The "IE dataset"**: consists of 767 positive examples (IE class) and 2,423 negative examples (union of EI class and "Neither" class);

2. **The "EI dataset"**: consists of 768 positive examples and 2,422 negative examples;

3. **The "Neither dataset"**: consists of 1,655 positive examples and 1,535 negative examples.

In our approach the partition of sample data into training samples and test samples follows [297] with about 33% as test data. Our encoding of data is done as described in the StatLog project (http://www.ncc.up.pt/liacc/ML/

statlog), where the symbolic variables representing the nucleotides (A,G,T,C) were replaced by 3 binary indicator variables:

$$A \rightarrow 100;\ C \rightarrow 010;\ G \rightarrow 001;\ T \rightarrow 000; \tag{5.1}$$

Therefore, the 60 attributes produce 180 binary attributes. We replaced missing values with the plausible value of 111. The dataset documentation indicates that much better performance is generally observed if attributes closest to the junction are used. Rambone [297] suggests a restricted window to 20 nucleotides, which in our case this means using binary attributes from the 61 position to the 120 position of the 180 length binary string.

Another encoding of data is described in [297], where the symbolic variables representing the nucleotides (A,G,T,C) were replaced by 4 binary indicator variables.

5.1.2 Pima Indians Diabetes Dataset (Pima)

The Pima Indians dataset contains data collected by the National Institute of Diabetes and Digestive and Kidney Diseases, USA. The set consists of 768 cases, where 268 have the diagnosis diabetes (35% of the data) and therefore are positive examples, and 500 (65% of the data) have the diagnosis non-diabetes and are negative examples. The data have eight attributes: number of times pregnant, plasma glucose concentration, diastolic blood pressure, triceps skin fold thickness, serum insulin, body mass index, diabetes pedigree function, and age. Although in the text file accompanying the data set in the UCI Repository, it is claimed that there are no missing data in the set, a closer look reveals that in six out of eight columns there are zeros clearly denoting unknown values, and therefore they are missing values. Many researchers have used this dataset probably without knowing that it contains missing data, and therefore we consider studies about Pima Indians referring to 768 cases, without referring to methods of handling missing data, as studies treating the missing values by ignoring them. The same method will also be used here, so in our experiments we will use the database as it is given in the UCI repository. The accuracy of various classifiers for the data set varies from 72% to 82%, where the highest values were obtained when all missing data positions were removed from the dataset [383].

5.1.3 Wisconsin Breast Cancer Dataset (WBCD)

The WBCD database is the result of efforts made at the University of Wisconsin Hospital for accurately diagnosing breast masses based solely on a Fine

Dataset = Splice-Junction IE						
P=100	K=25,000	$\|S\|$=3,186	$\|S_L\|$=2,124	$\|S_T\|$=1,062	$\Gamma = 0.75 \cdot \|S_L^j\|$	
x_0	$\|S_L^j\|$	e_L%	U_0%	U_1%	e_T%	±%
2	1,062	0.6%	4.7%	83.1%	4.05%	0.15%
3	708	0%	75.1%	100%	4.05%	0.14%
4	531	0%	98.5%	100%	4.07%	0.18%
5	424	0%	100%	100%	3.87%	0.21%
6	354	0%	100%	100%	3.95%	0.22%
7	303	0%	100%	100%	4.04%	0.31%
8	265	0%	100%	100%	4.05%	0.19%
10	212	0%	100%	100%	4.15%	0.16%

Table 5.1: Sample Size and Learning Properties for IE.

Dataset = Splice-Junction IE						
P=100	K=25,000	$\|S\|$=3,186	$\|S_L\|$=2,124	$\|S_T\|$=1,062	$\Gamma = 0.75 \cdot \|S_L^j\|$	
x_0	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
2	24,879	0.5%	16,533	33.9%	4.05%	5416
3	24,002	4.0%	6,400	74.4%	4.05%	1344
4	23,568	5.7%	2,647	89.4%	4.07%	551
5	11,300	54.8%	590	97.6%	3.87%	65
6	2,577	89.7%	353	98.6%	3.95%	39
7	1,413	94.3%	254	99.0%	4.04%	31
8	1,084	95.7%	201	99.2%	4.05%	15
10	583	97.7%	144	99.4%	4.15%	9

Table 5.2: Sample Size and Convergence Properties for IE.

Needle Aspiration (FNA) test. WBCD is a binary classification problem where each vector represents 9 features. The output indicates either a benign case (positive example) or a malignant case (negative example). The data set consists of 699 samples with integer values, where 16 samples have missing values which in our experiments have been discarded in a pre-processing step. The remaining 683 data are divided into 444 benign and 239 malignant cases.

5.2 Sample Size Complexity and Training Quality

For the size complexity problem related to $|S_L^j|$, we use integer steps $x_0 \geq 2$ for calculating the number of random examples $|S_L^j| = |S_L|/x_0$.

SJGS Datasets

One might expect, as the three datasets IE, EI, "Neither" are different

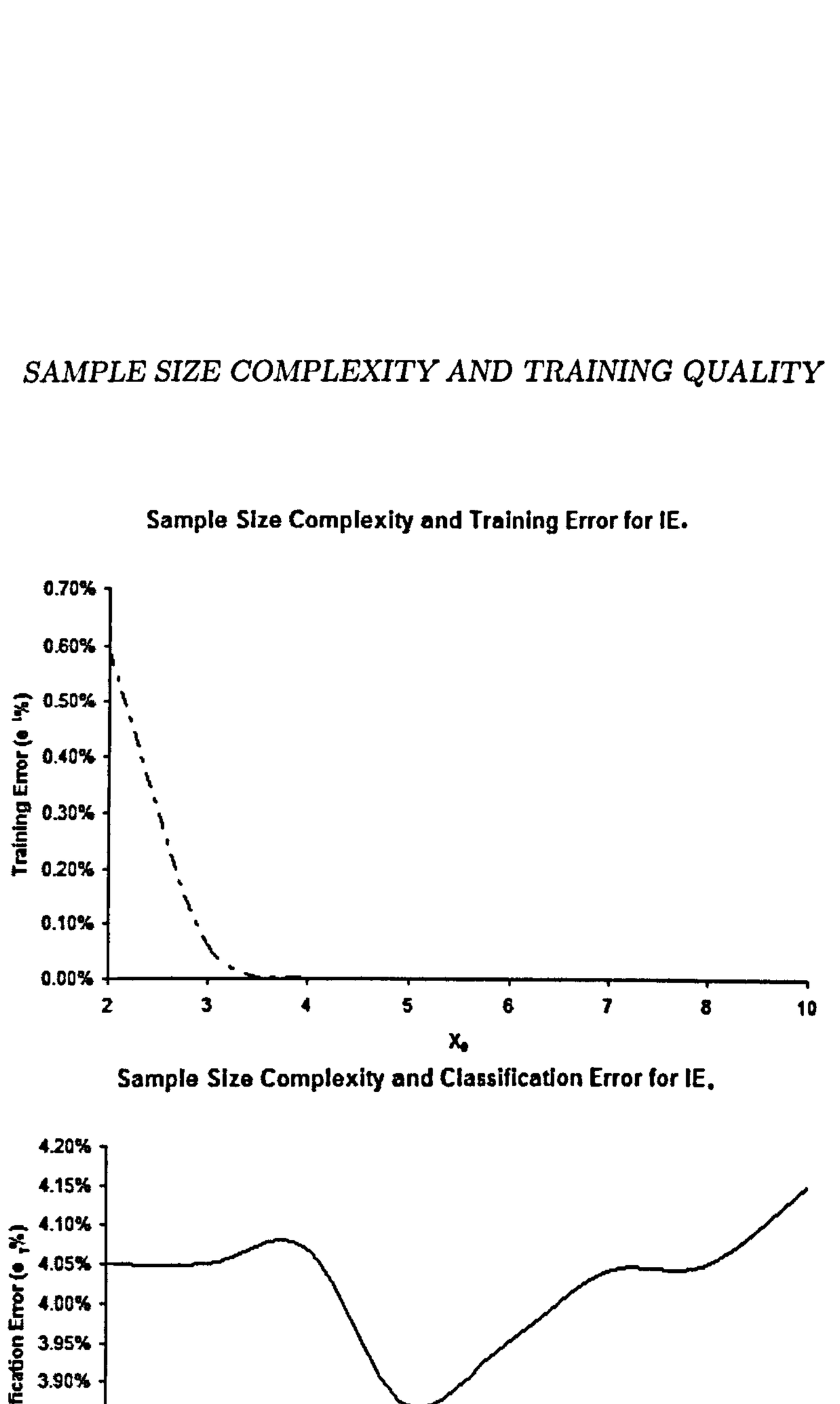


Figure 5.1: Sample Size Complexity Results for IE.

Dataset = Splice-Junction EI

P=100 K=25,000 $|S|$=3,186 $|S_L|$=2,124 $|S_T|$=1,062 $\Gamma = 0.75 \cdot |S_L^j|$

x_0	$\lvert S_L^j \rvert$	e_L%	U_0%	U_1%	e_T%	±%
2	1,062	0.1%	44.0%	100%	2.75%	0.14%
3	708	0%	98.1%	100%	2.66%	0.15%
4	531	0%	100%	100%	3.02%	0.09%
5	424	0%	100%	100%	3.25%	0.18%
6	354	0%	100%	100%	3.30%	0.18%
7	303	0%	100%	100%	3.45%	0.34%
8	265	0%	100%	100%	3.70%	0.33%
10	212	0%	100%	100%	4.07%	0.32%

Table 5.3: Sample Size and Learning Properties for EI.

Dataset = Splice-Junction EI

P=100 K=25,000 $|S|$=3,186 $|S_L|$=2,124 $|S_T|$=1,062 $\Gamma = 0.75 \cdot |S_L^j|$

x_0	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
2	24,988	0.0%	6,542	73.8%	2.75%	3,529
3	20,135	19.5%	1,204	95.2%	2.66%	389
4	10,279	58.9%	655	97.4%	3.02%	139
5	2,525	89.9%	369	98.5%	3.25%	36
6	1,928	92.3%	264	98.9%	3.30%	32
7	1,006	96.0%	208	99.2%	3.45%	24
8	677	97.3%	175	99.3%	3.70%	14
10	482	98.1%	133	99.5%	4.07%	9

Table 5.4: Sample Size and Convergence Properties for EI.

views onto the same splice-junction dataset, that they might have the same training properties and therefore might have the same parameter setting for the best classification rate. However, results show that this is not the case. Each dataset is considered as a separate dataset and best classification rates are obtained for different values of the parameter set. Therefore, different x_0 values are suggested for each dataset. Moreover, preliminary experiments calculating the value of the maximum escape depth for P=100 suggest values of $\Gamma_{IE} = 0.75 \cdot |S_L^j|$, $\Gamma_{EI} = 0.75 \cdot |S_L^j|$, $\Gamma_{Neither} = 0.50 \cdot |S_L^j|$.

IE

Table 5.1 and Table 5.2 display experimental results for the IE dataset and values of $x_0 = \{2, 3, 4, 5, 6, 7, 8, 10\}$. The dataset is trained with relatively good learning error, even for large sample sets and most of the threshold units are capable of learning the examples with zero or 1% error.

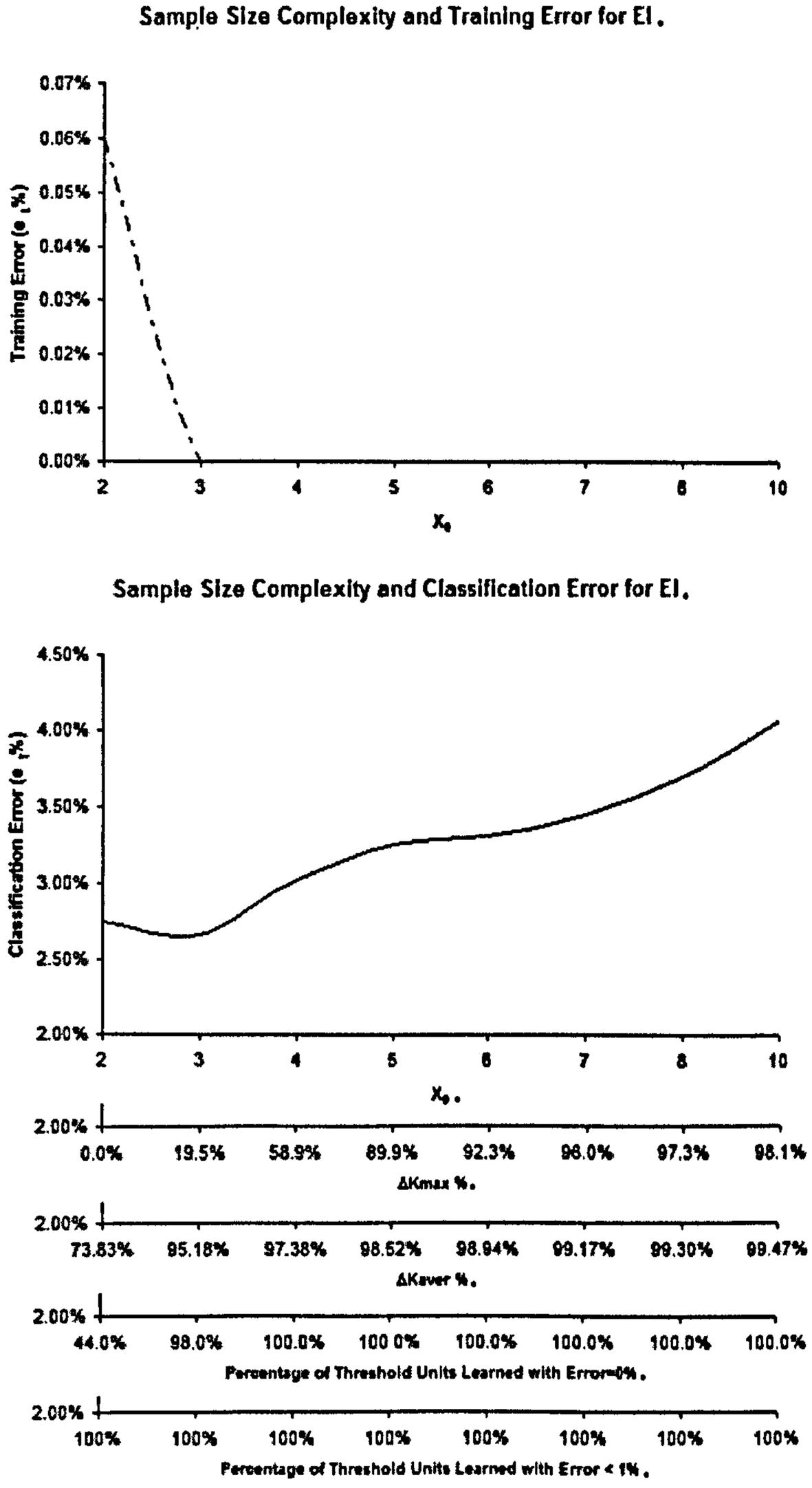


Figure 5.2: Sample Size Complexity Results for EI.

Dataset = Splice-Junction "Neither"

P=100 K=25,000 $|S|$=3,186 $|S_L|$=2,124 $|S_T|$=1,062 $\Gamma = 0.50 \cdot |S_L^j|$

x_0	$\|S_L^j\|$	e_L%	U_0%	U_1%	e_T%	±%
2	1,062	3.7%	0%	0%	8.08%	0.32%
3	708	1.5%	1.7%	38.2%	6.61%	0.18%
4	531	0.5%	40.0%	87.1%	6.27%	0.21%
5	424	0.2%	86.2%	98.4%	6.14%	0.26%
6	354	0%	98.8%	99.8%	6.06%	0.19%
7	303	0%	100%	100%	6.32%	0.25%
8	265	0%	100%	100%	6.59%	0.16%
10	212	0%	100%	100%	7.04%	0.34%

Table 5.5: Sample Size and Learning Properties for "Neither".

Dataset = Splice-Junction "Neither"

P=100 K=25,000 $|S|$=3,186 $|S_L|$=2,124 $|S_T|$=1,062 $\Gamma = 0.50 \cdot |S_L^j|$

x_0	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
2	24,992	0.0%	19,296	22.8%	8.08%	5,744
3	24,965	0.1%	16,528	33.9%	6.61%	3,553
4	24,899	0.4%	11,224	55.1%	6.27%	1,126
5	24,894	0.4%	4,486	82.1%	6.14%	424
6	22,098	11.6%	1,644	93.4%	6.06%	169
7	16,113	35.5%	846	96.6%	6.32%	117
8	5,796	76.8%	572	97.7%	6.59%	35
10	1,823	92.7%	369	98.5%	7.04%	22

Table 5.6: Sample Size and Convergence Properties for "Neither".

The best classification accuracy e_T is obtained for $x_0 = 5$, where at this value U_0^5 stabilizes to 100% from the previous step $U_0^4 = 98.5\%$, K_{max} considerably lowers its value in relation to the previous step $x_0 = 4$, and this value ($K_{max}^5 = 11300$) is considerably lower in relation to K. U_1 approaches values close to 100% . Remarkable is that training time for best classification accuracy is rapidly ($T = 65$ seconds) and is considerably lower than at previous steps of x_0.

Results from Table 5.1 and Table 5.2 are plotted in Figure 5.1. The diagrams show e_L and e_T errors in relation to x_0. Moreover, for the second diagram we draw various secondary x axes for demonstrating the behaviour of the classification error over ΔK_{max}, ΔK_{aver}, U_0, and U_1. These axes also depend on x_0 values. They are used for visualising results and identifying best classification accuracy patterns by observing points of local minima.

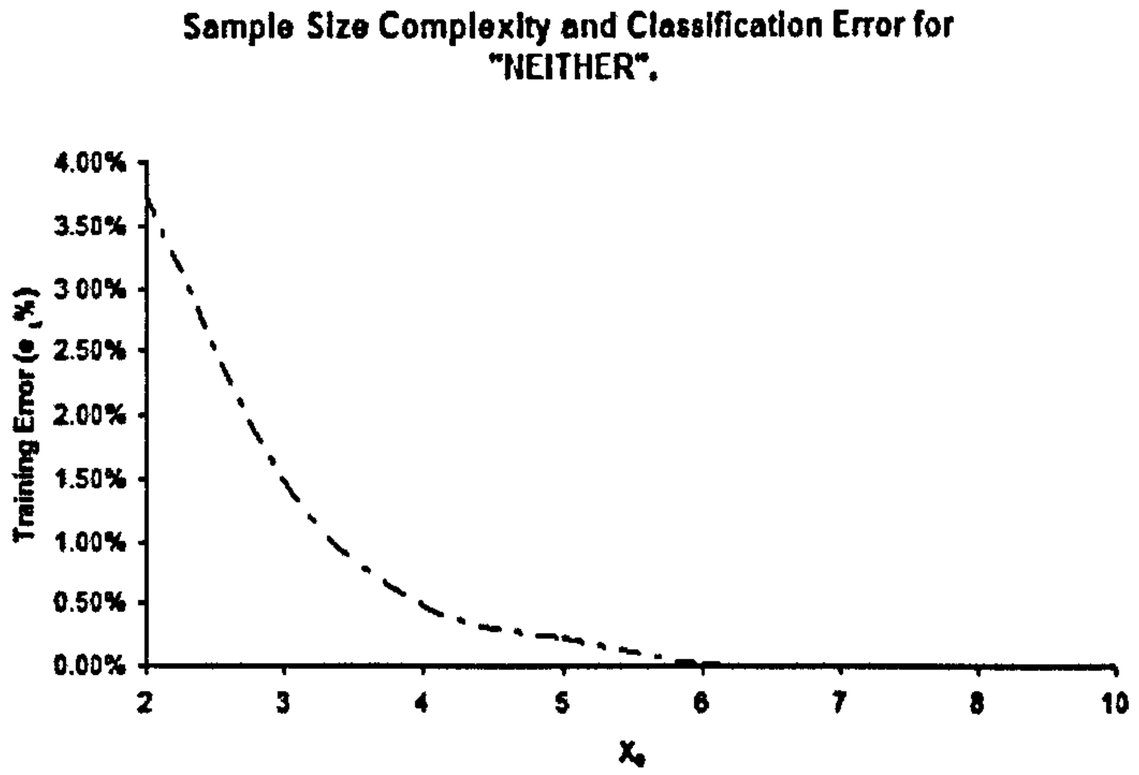


Figure 5.3: Sample Size Complexity Results for "Neither".

Dataset = Pima Indians

P=100 K=25,000 $|S|$=768 $|S_L|$=512 $|S_T|$=256 $\Gamma = 0.66 \cdot |S_L^1|$

x_0	$\|S_L^j\|$	e_L%	U_0%	U_1%	e_T%	±%
2	256	25.8%	0%	0%	27.30%	0.65%
4	128	21.1%	0%	0%	28.00%	0.64%
6	85	17.5%	0%	0%	28.16%	0.83%
8	64	14.6%	0%	0%	27.73%	1.03%
10	51	10.9%	2.0%	2.0%	26.74%	0.67%
15	34	6.53%	17.1%	17.1%	25.39%	1.22%
20	26	4.5%	46.0%	46.0%	25.66%	0.61%
25	20	3.1%	63.2%	63.2%	25.12%	1.03%
30	17	2.8%	70.1%	70.1%	26.45%	1.33%

Table 5.7: Sample Size and Learning Properties for Pima Indians.

EI

Recognizing exon/intron boundaries in the EI dataset (Table 5.3, Table 5.4, and Figure 5.2) seems to be easier than recognizing intron/extron boundaries in the IE dataset, as our classification method has the best classification accuracy for much larger in size sample sets: $x_0^{EI} = 3$ corresponding to 708 examples, whereas $x_0^{IE} = 5$ corresponding to 424 examples. This, however, leads to more training time ($T = 389$ seconds) for the best classification accuracy. The best classification rate is obtained when U_0 approaches values close to 100%, K_{max} is significantly lower than K, and ΔK_{aver} approaches 100%. The former two are common observations with those in IE dataset.

"Neither""

We observe that for "Neither" (Table 5.5, Table 5.6, and Figure 5.3) the classification error is higher compared to the two previous datasets. "Neither" contains as positive examples the classes that are not EI and not IE classes.

One might say that the classification error is expected to be close to the worst case classification error of the two datasets, however, the classification error is closer to $e_T^{IE} + e_T^{EI}$. We observe a similar pattern for "Neither" in Rampone's results [297].

Probably, class "Neither" contains the effort of learning two classes at the same time: class "not(IE)" and class "not(EI)" requiring a considerable effort compared to learning only one class. The distribution of positive and negative examples in class "Neither" is balanced by 50% positive and 50% negative

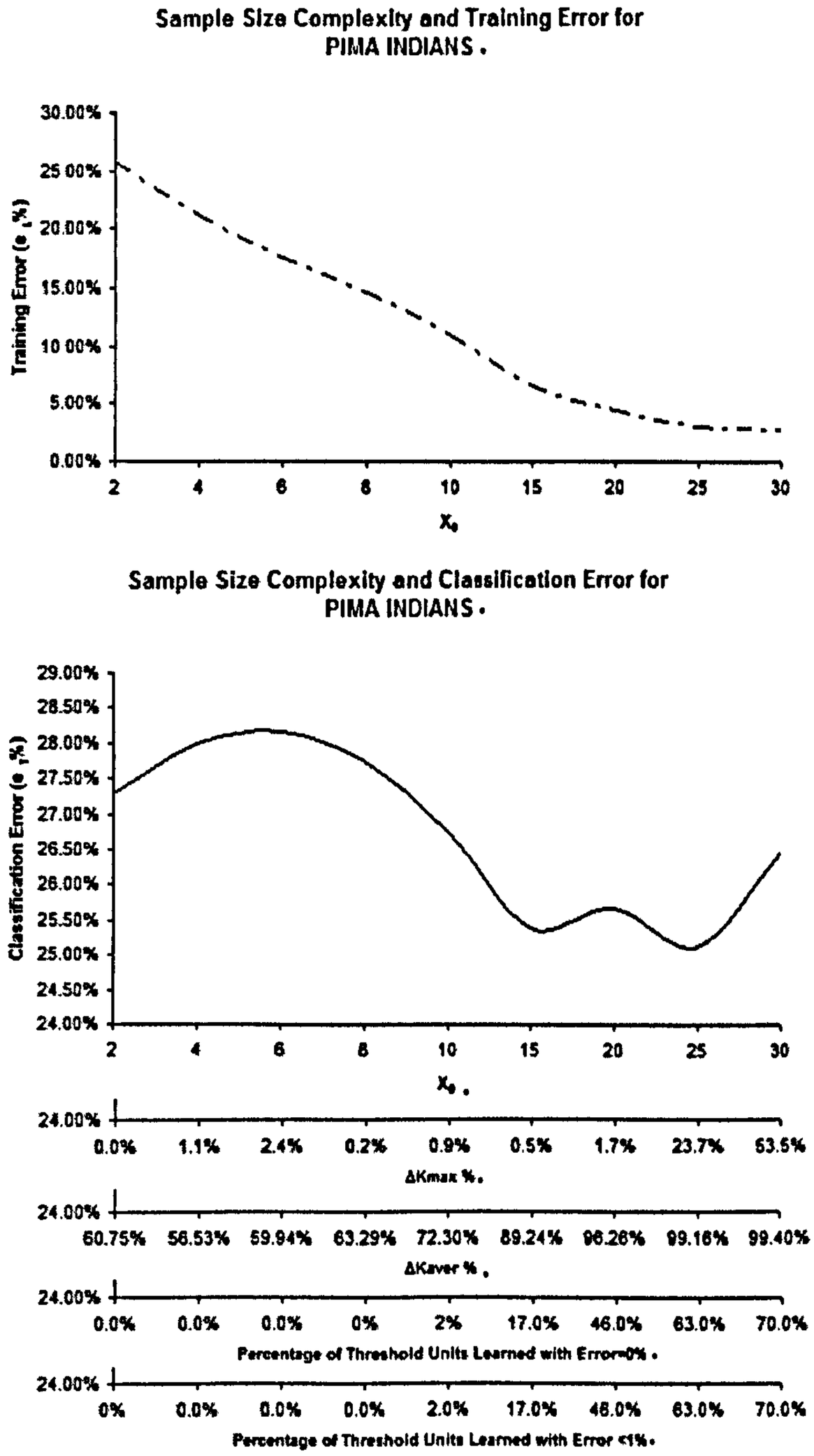


Figure 5.4: Sample Size Complexity Results for Pima Indians.

Dataset = Pima Indians

P=100 K=25,000 $|S|$=768 $|S_L|$=512 $|S_T|$=256 $\Gamma = 0.66 \cdot |S_L^j|$

x_0	K_{max}	$\Delta(K_{max})\%$	K_{aver}	$\Delta(K_{aver})\%$	$e_T\%$	T(sec)
2	24,990	0.0%	9,812	60.8%	27.30%	829
4	24,728	1.1%	10,868	56.5%	28.00%	436
6	24,392	2.4%	10,015	59.9%	28.16%	301
8	24,944	0.2%	9,178	63.3%	27.73%	278
10	24,765	0.9%	6,925	72.3%	26.74%	251
15	24,869	0.5%	2,689	89.2%	25.39%	127
20	24,586	1.7%	934	96.3%	25.66%	76
25	19,065	23.7%	210	99.2%	25.12%	58
30	11,623	53.5%	150	99.4%	26.45%	31

Table 5.8: Sample Size and Convergence Properties on Pima Indians.

examples, whereas datasets IE and EI have a distribution of 25% positive and 75% negative examples. Class "Neither", as described above, differs also in the variable of the estimated maximum escape depth yielding to a lower value of Γ than in the other two datasets.

The best classification rate is obtained for $x_0 = 6$, where $e_L = 0\%$, and U_0, U_1 approach 100%. In comparison to the previous datasets, K_{max} is significantly smaller than K, and therefore K_{aver} is also significantly smaller compared to K, so that $\Delta K_{aver} > 90\%$.

Pima Indians Dataset

The Pima Indians dataset has low reported classification accuracies, as mentioned already in Section 5.1.2. Results in Table 5.7, Table 5.8 and Figure 5.4 reveal that the learning hardness is reflected in the sample size, where much larger values like $x_0 = 25$ are needed for best classification accuracy.

High rates of learning accuracy related to small values of e_L cannot be achieved even for very small sample sizes. We tried lowering the examples in the sample set reaching discrimination between examples only if 4 examples are used. For the Pima Indians dataset preliminary experiments calculating the maximum escape depth for $P = 100$ suggest value of $\Gamma_{pima} = 0.66 \cdot |S_L^j|$.

The importance of the results is that we identify the same pattern behaviour related to K_{max}, where again the best classification accuracy is achieved on x_0^i when K_{max}^i is significantly smaller than K_{max}^{i-1}, which is usually very close to K.

Dataset = WBCD

P=100 K=25,000 $|S|$=683 $|S_L|$=455 $|S_T|$=228 $\Gamma = 0.75 \cdot |S_L^j|$

x_0	$\lvert S_L^j \rvert$	e_L%	U_0%	U_1%	e_T%	±%
2	228	3.2%	1.9%	7.5%	1.05%	0.42%
3	152	2.1%	20.9%	34.5%	0.71%	0.33%
4	114	1.2%	48.8%	66.0%	0.58%	0.22%
5	91	0.8%	56.0%	66.0%	0.55%	0.20%
6	76	0.5%	80.1%	80.1%	0.83%	0.34%
7	65	0.4%	85.0%	85.0%	1.02%	0.22%
8	57	0.2%	91.6%	91.6%	0.99%	0.20%
10	46	0.1%	96.5%	96.5%	1.15%	0.58%
15	30	0%	100%	100%	2.68%	0.64%

Table 5.9: Sample Size and Learning Properties for WBCD.

Dataset = WBCD

P=100 K=25,000 $|S|$=683 $|S_L|$=455 $|S_T|$=228 $\Gamma = 0.75 \cdot |S_L^j|$

x_0	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
2	24,708	1.2%	5,402	78,4%	1.05%	502
3	24,929	0.3%	6,664	73,3%	0.71%	306
4	24,875	0.5%	5,738	77,0%	0.58%	133
5	24,911	0.4%	4,609	81,6%	0.55%	118
6	24,478	2.1%	3,212	87,1%	0.83%	82
7	24,919	0.3%	2,753	89,0%	1.02%	36
8	24,674	1.3%	1,986	92,1%	0.99%	20
10	23,989	4.0%	1,359	94,6%	1.15%	9
15	23,782	4.9%	557	97,8%	2.68%	5

Table 5.10: Sample Size and Convergence Propertics for WBCD.

WBCD

For the WBCD preliminary experiments calculating the maximum escape depth for $P = 100$ suggest value of $\Gamma_{wbcd} = 0.75 \cdot |S_L^j|$. For WBCD we see from Table 5.9, Table 5.10 and Figure 5.5 that we cannot achieve a very small K_{max}, i.e. K_{max} is always close to K, even for small sets. Therefore, we are not able to relate K_{max} to classification accuracy for this dataset. Best classification rate is achieved for $x_0 = 5$. Minimizing e_L, U_0, and U_1 does not affect classification rate. The results presented here outperform all other methods in the literature, to the best of our knowledge.

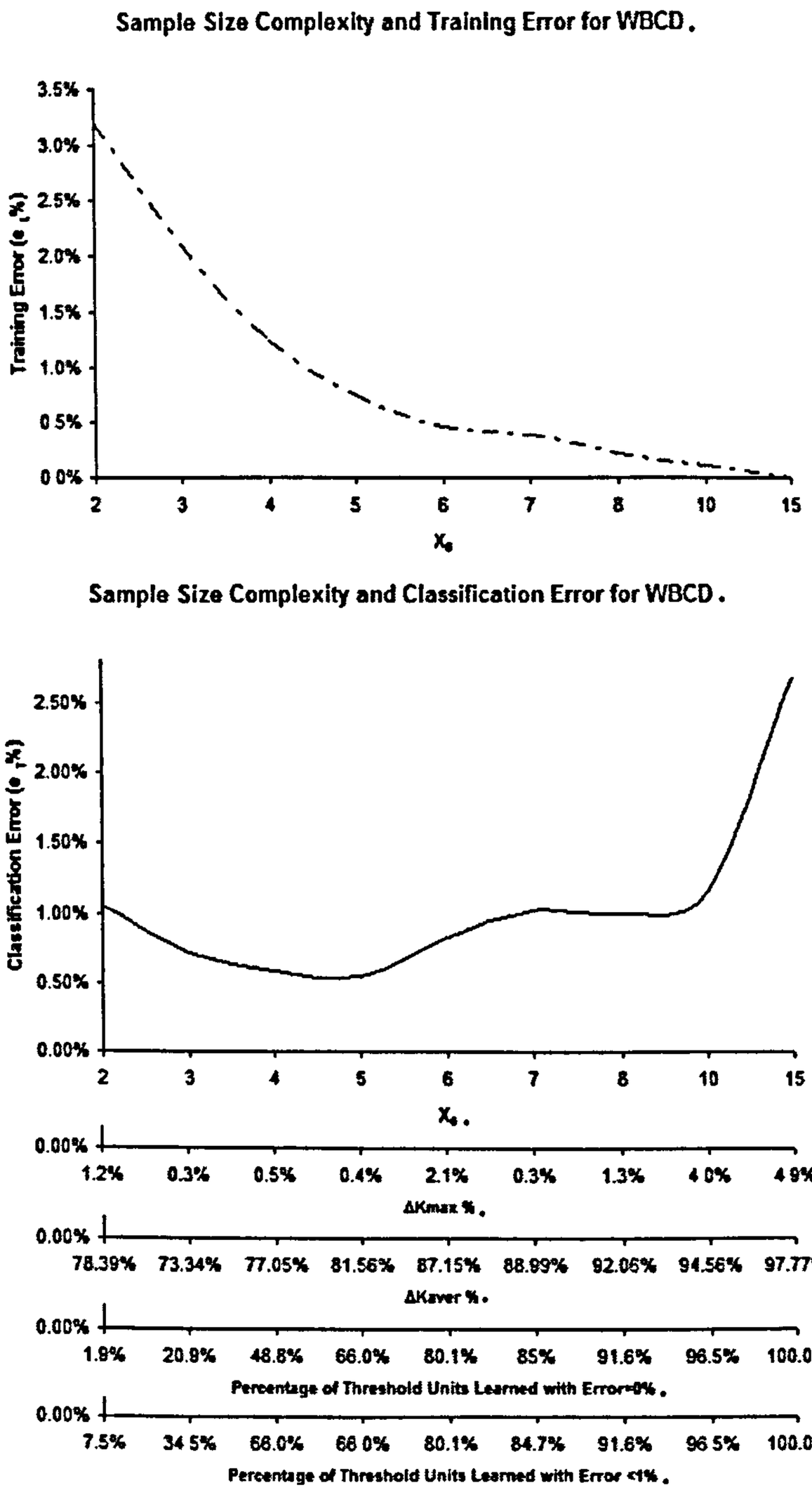


Figure 5.5: Sample Size Complexity Results for WBCD.

Dataset = Splice-Junction IE					
P=100	$x_0 = 5$	$\|S\|$=3,186	$\|S_L\|$=2,124	$\|S_T\|$=1,062	$\Gamma = 0.75 \cdot \|S_L^i\|$
K	e_L%	U_0%	U_1%	e_T%	±%
500	0.4%	63.7%	87.5%	3.90%	0.18%
1,500	0%	94.6%	99.8%	4.07%	0.15%
5,000	0%	100%	100%	3.95%	0.24%
15,000	0%	100%	100%	3.87%	0.21%
25,000	0%	100%	100%	3.87%	0.21%
50,000	0%	100%	100%	3.87%	0.21%
150,000	0%	100%	100%	3.87%	0.21%

Table 5.11: IMC Length and Learning Properties for IE.

Dataset = Splice-Junction IE						
P=100	$x_0 = 5$	$\|S\|$=3,186	$\|S_L\|$=2,124	$\|S_T\|$=1,062	$\Gamma = 0.75 \cdot \|S_L^i\|$	
K	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(scc)
500	500	0%	375	25%	3.90%	38
1,500	1,495	0%	532	65%	4.07%	51
5,000	4,932	1%	564	89%	3.95%	54
15,000	11,189	25%	551	96%	3.87%	65
25,000	11,300	55%	590	98%	3.87%	65
50,000	11,353	77%	592	99%	3.87%	67
150,000	11,296	92%	549	99%	3.87%	119

Table 5.12: IMC Length and Convergence Properties for IE.

5.3 Length of Inhomogeneous Markov Chains

Experimenting with the sample complexity problem, we observed that by relating value x_0 to a problem dependent parameter of the convergence space such as K_{max} does provide good classification rates, where $\Delta K_{max}^i \geq \delta_{x0}$ such that $K_{max}^i \not\approx K$ and $K_{max}^{i-1} \approx K$. We experimentally obtained values for best classification rates at: $x_0(IE) = 5$, $x_0(EI) = 3$, $x_0(neither) = 6$, $x_0(pima) = 25$, and $x_0(wbcd) = 5$.

In this section we investigate the learning effort in terms of the length of inhomogeneous Markov chains (IMCs). Do large chains allow better classification rates, or do they lead to overfitting? Does early stopping improve classification rate? These are questions related to the length of IMCs that we will experimentally investigate.

Table 5.11 till Table 5.20 and Figure 5.6 till Figure 5.10 present the results for specific values of $K = \{500, 1500, 5000, 15000, 25000, 50000, 150000\}$ transitions. We also run experiments for much larger K=1,500,000 transitions for

Dataset = Splice-Junction EI

P=100 $x_0 = 3$ $|S|$=3,186 $|S_L|$=2,124 $|S_T|$=1,062 $\Gamma = 0.75 \cdot |S_L^j|$

K	e_L%	U_0%	U_1%	e_T%	±%
500	0.5%	12.0%	86.0%	2.91%	0.14%
1,500	0.1%	64.8%	100%	2.81%	0.20%
5,000	0%	88.6%	100%	2.74%	0.15%
15,000	0%	94.3%	100%	2.69%	0.10%
25,000	0%	98.0%	100%	2.66%	0.15%
50,000	0%	98.3%	100%	2.68%	0.14%
150,000	0%	98.8%	100%	2.67%	0.12%
1,500,000	0%	98.8%	100%	2.72%	0.12%

Table 5.13: IMC Length and Learning Properties for EI.

Dataset = Splice-Junction EI

P=100 $x_0 = 3$ $|S|$=3,186 $|S_L|$=2,124 $|S_T|$=1,062 $\Gamma = 0.75 \cdot |S_L^j|$

K	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
500	500	0%	434	13%	2.91%	69
1,500	1,489	1%	849	43%	2.81%	143
5,000	4,824	4%	1,332	73%	2.74%	248
15,000	14,332	4%	1,934	87%	2.69%	289
25,000	23,135	7%	2,239	91%	2.66%	389
50,000	47,526	5%	2,542	95%	2.68%	421
150,000	126,899	15%	3,126	98%	2.67%	474
1,500,000	707,703	53%	5,405	99%	2.72%	5,984

Table 5.14: IMC Length and Convergence Properties for EI.

EI, trying to minimise U_0, and for WBCD, trying to obtain a larger value for ΔK_{max}. The latter is more related to the question of what length of IMCs is large enough for WBCD so that transitions are finally bounded by values $K_{max} << K$.

IE

Dataset IE (Table 5.11, Table 5.12 and Figure 5.6) stabilizes its classification rate after K=15,000 transitions. The main reason for this stability is that K_{max} has a stable value of its maximum value of an acceptance transition, which is around K=11,300. Thus, any further increase in the IMC length has no effect on the classification rate.

Dataset = Splice-Junction "Neither"

P=100	$x_0 = 6$	$\|S\|$=3,186	$\|S_L\|$=2,124	$\|S_T\|$=1,062	$\Gamma = 0.50 \cdot \|S_L^1\|$
K	e_L%	U_0%	U_1%	e_T%	±%
500	4.6%	15.6%	28.4%	6.73%	0.17%
1,500	0.3%	70.2%	88.8%	6.31%	0.27%
5,000	0%	94.5%	98.8%	6.21%	0.27%
15,000	0%	98.5%	99.8%	6.23%	0.16%
25,000	0%	98.8%	99.8%	6.06%	0.19%
50,000	0%	98.9%	99.8%	6.17%	0.18%
150,000	0%	98.8%	99.8%	6.29%	0.29%

Table 5.15: IMC Length and Learning Properties for "Neither".

Dataset = Splice-Junction "Neither"

P=100	$x_0 = 6$	$\|S\|$=3,186	$\|S_L\|$=2,124	$\|S_T\|$=1,062	$\Gamma = 0.50 \cdot \|S_L^1\|$	
K	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
500	500	0%	412	18%	6.73%	39
1,500	1,496	0%	958	36%	6.31%	82
5,000	4,984	0%	1388	72%	6.21%	109
15,000	14,325	5%	1754	88%	6.23%	118
25,000	22,098	12%	1644	93%	6.06%	169
50,000	49,525	1%	2249	95%	6.17%	217
150,000	143,633	4%	2669	98%	6.29%	254

Table 5.16: IMC Length and Convergence Properties for "Neither".

EI

Dataset EI (Table 5.13, Table 5.14, and Figure 5.7) gives no stable value for K_{max}, however, the longer the chain the larger the distance between K_{max} and K. One can say that classification rate stabilizes close to the value of best classification rate after K=25,000 transitions, where also U_0 stabilizes. We tried to obtain classification rate results for $U_0 = 100\%$ by extending the chain to 1,500,000 transitions, which turned out to be inefficient, as we obtained exactly the same value of U_0 as for 150,000 transitions. Moreover, a slight increase in the classification rate seems to make it not reasonable to further extend the length of IMC.

"Neither"

Dataset "Neither" (Table 5.15, Table 5.16, and Figure 5.8) shows the best

Dataset = Pima Indians

P=100 $x_0 = 25$ $|S| = 768$ $|S_L| = 512$ $|S_T| = 256$ $\Gamma = 0.66 \cdot |S_L^j|$

K	e_L%	U_0%	U_1%	e_T%	±%
500	3.7%	60.3%	60.3%	26.09%	1.22%
1,500	3.2%	62.6%	62.6%	25.55%	1.25%
5,000	3.2%	59.2%	59.2%	25.55%	1.18%
15,000	3.2%	61.3%	61.3%	25.66%	1.14%
25,000	3.1%	63.0%	63.0%	25.12%	1.03%
50,000	3.2%	60.0%	60.0%	24.96%	1.02%
150,000	3.6%	56.8%	56.8%	26.26%	1.29%

Table 5.17: IMC Length and Learning Properties for Pima Indians.

Dataset = Pima Indians

P=100 $x_0 = 25$ $|S| = 768$ $|S_L| = 512$ $|S_T| = 256$ $\Gamma = 0.66 \cdot |S_L^j|$

K	K_{max}	$\Delta(K_{max})$%	K_{aver}	$\Delta(K_{aver})$%	e_T%	T(sec)
500	500	0%	118	76%	26.09%	5
1,500	1,472	2%	167	89%	25.55%	6
5,000	4,338	13%	220	96%	25.55%	12
15,000	10,525	30%	229	98%	25.66%	31
25,000	19,065	24%	210	99%	25.12%	58
50,000	44,532	11%	406	99%	24.96%	88
150,000	55,314	63%	335	100%	26.26%	342

Table 5.18: IMC Length and Convergence Properties for Pima Indians.

Dataset = WBCD

P=100 $x_0 = 5$ $|S| = 683$ $|S_L| = 455$ $|S_T| = 228$ $\Gamma = 0.75 \cdot |S_L^j|$

K	e_L%	U_0%	U_1%	e_T%	±%
500	11.0%	5.5%	5.5%	1.62%	0.36%
1,500	3.2%	25.0%	25.0%	1.05%	0.31%
5,000	1.2%	56.8%	56.8%	0.53%	0.18%
15,000	1.0%	61.8%	61.8%	0.44%	0.15%
25,000	0.8%	66.0%	66.0%	0.55%	0.20%
50,000	0.7%	67.3%	67.3%	0.66%	0.31%
150,000	0.7%	67.8%	67.8%	0.57%	0.21%
1,500,000	0.6%	72.2%	72.2%	0.57%	0.21%

Table 5.19: IMC Length and Learning Properties for WBCD.

Dataset = WBCD

P=100	$x_0 = 5$	$\|S\| = 683$	$\|S_L\| = 455$	$\|S_T\| = 228$	$\Gamma = 0.75 \cdot \|S_L^1\|$	
K	K_{max}	$\Delta(K_{max})\%$	K_{aver}	$\Delta(K_{aver})\%$	$e_T\%$	T(scc)
500	500	0%	365	27%	1.62%	8
1,500	1,497	0%	1096	27%	1.05%	16
5,000	4,993	0%	2313	54%	0.53%	31
15,000	14,945	0%	3720	75%	0.44%	71
25,000	24,911	0%	4609	82%	0.55%	118
50,000	49,167	2%	6294	87%	0.66%	202
150,000	139,814	7%	7653	95%	0.57%	431
1,500,000	1,394,062	7%	30352	98%	0.57%	9,477

Table 5.20: IMC Length and Convergence Properties for WBCD.

classification rate at value K=25,000, and a certain type of overfitting occurs for larger values of K. At this value K_{max} has difference $\Delta K_{max} = 0.12$ from K, however, for larger chains ΔK_{max} becomes smaller again and classification error simultaneously slightly increases.

Pima Indians

Dataset Pima Indians (Table 5.17, Table 5.18, and Figure 5.9) shows low classification errors for values $K = \{25000, 50000\}$. Overfitting occurs for larger chains.

WBCD

WBCD dataset (Table 5.19, Table 5.20, and Figure 5.10) shows low classification errors from early Markov chain lengths, such as $K = 5,000$ transitions, without overfitting for larger chains.

While ΔK_{max} can be used for the sample size complexity in finding the appropriate value of x_0, it cannot be used for the length of IMCs. Neither does the rest of monitored parameters allow us to make a connection to the length with respect to properties of these values. The length depends very much on the nature of the problem, however, a value of 25,000 transitions for our datasets seems to be an acceptable value that is quite close to good and best classification accuracies.

Dataset	(x_0, K)	e_L%	U_0%	U_1%	e_T%	±%	T(sec)
IE	(3, 25,000)	0.1%	75%	100%	4.05%	0.14%	1,344
IE	(3, 150,000)	0%	83%	100%	4.00%	0.22%	4,098
IE	(5, 25,000)	0%	100%	100%	3.87%	0.21%	65
"Neither"	(3, 25,000)	1.50%	2%	38%	6.61%	0.18%	3,553
"Neither"	(3, 150,000)	1.17%	5%	51%	6.43%	0.24%	21,362
"Neither"	(6, 25,000)	0%	99%	100%	6.06%	0.19%	169
Pima	(3, 25,000)	21.1%	0%	0%	28.00%	0.64%	436
Pima	(3, 150,000)	20.2%	0%	0%	27.82%	0.58%	2,555
Pima	(25, 25,000)	3.2%	60%	60%	24.96%	1.02%	88
WBCD	(3, 25,000)	2.1%	21%	35%	0.71%	0.33%	306
WBCD	(3, 150,000)	2.0%	22%	40%	0.88%	0.14%	1,491
WBCD	(5, 25,000)	0.8%	68%	68%	0.55%	0.20%	118

Table 5.21: Sample Size Matters More than IMC Length.

5.4 Sample Size vs IMC Length

In the previous section, we have seen the effect of IMCs on the best values of x_0 obtained in Section 5.2. The question rises whether we obtain an improved classification rate, if we apply large Markov chains to large sample sets as those obtained for $x_0 = 3$. Table 5.21 demonstrates this scenario for four datasets, as for EI we already obtained $x_0 = 3$ from the size complexity problem. For each dataset, the first row shows the initial hypothesis, i.e. results for large sets and K=25,000 transitions are displayed. In the second row, we considerably increased the chain length to allow sufficient steps for decreasing the learning error e_L. In the third row, we display the results from optimal values of x_0. We have slight improvements for the learning error e_L, which is the effect of simulated annealing convergence. However, the improvement of classification errors for IE, "Neither", and Pima dataset is considerably smaller than the improvement obtained by changing x_0. Thus, we have a considerably better classification rate from improving the size of examples rather than from allowing more time for searching in the configuration space.

5.5 The Cooling Schedule

The parameter Γ defines the cooling schedule of the simulated annealing process. Γ depends on the size of the sample sets $|S_L^j|$ and a rule of thumb used in previous experiments is to estimate Γ by preliminary experiments on

Dataset = IE	$\Gamma = \lvert S_L^j \rvert \cdot G\%$		$D = \lvert S_L^j \rvert \cdot M\%$			
P=100		K=25,000			$x_0 = 5$	
$\lvert S_L \rvert$=2,124		$\lvert S_L^j \rvert = 424$			$\lvert S_T \rvert$=1,062	
$G\%$	$M\%$	$e_L\%$	$U_0\%$	K_{aver}	$e_T\%$	T(scc)
0	71	0%	100%	2,620	4.04%	238
10	72	0%	100%	1,754	3.82%	153
25	73	0%	100%	1,102	4.03%	95
33	74	0%	100%	927	3.72%	71
50	73	0%	100%	692	3.82%	77
66	74	0%	100%	653	3.84%	68
75	74	0%	100%	590	3.87%	65
100	73	0%	100%	999	3.87%	85

Table 5.22: Cooling Schedule Γ for IE.

Dataset = EI	$\Gamma = \lvert S_L^j \rvert \cdot G\%$		$D = \lvert S_L^j \rvert \cdot M\%$			
P=100		K=25,000			$x_0 = 3$	
$\lvert S_L \rvert$=2,124		$\lvert S_L^j \rvert = 708$			$\lvert S_T \rvert$=1,062	
$G\%$	$M\%$	$e_L\%$	$U_0\%$	K_{aver}	$e_T\%$	T(scc)
0	67	0%	91%	6,282	2.82%	1,146
10	70	0%	95%	3,942	2.77%	721
25	73	0%	97%	2,507	2.76%	484
33	74	0%	97%	2,115	2.92%	359
50	72	0%	97%	2,002	2.77%	317
66	74	0%	97%	2,054	2.77%	341
75	73	0%	98%	2,239	2.66%	376
100	70	0%	98%	1,930	2.69%	311

Table 5.23: Cooling Schedule Γ for EI.

$|S_L^j|$ by estimating the depth D of the deepest local minimum, as described in Section 4.7, item 3.

We want to analyze the behaviour of this parameter for our classifier. We use K=25,000, $P = 100$, and the corresponding value of x_0 determined for each dataset in Section 5.2.

As Γ depends on the size of $|S_L^j|$, we run experiments for $\Gamma = |S_L^j| \cdot G\%$ for values of $G = \{0, 10, 25, 33, 50, 66, 75, 100\}$. We investigate Γ by monitoring the same settings for other parameters. We are interested in the relation of Γ to the classification rate, if there is a relation to any of the monitored parameters, and also to check the applicability of Hajek's theory with respect to the maximum depth D of local minima. For this reason we also monitor D, which is represented as a percentage $M\%$ of the size of sample set: $D = |S_L^j| \cdot M\%$. Results are presented in Table 5.22 till Table 5.26.

Dataset = "Neither"	$\Gamma = \lvert S_L^j \rvert \cdot G\%$	$D = \lvert S_L^j \rvert \cdot M\%$
P=100	K=25,000	$x_0 = 6$
$\lvert S_L \rvert$=2,124	$\lvert S_L^j \rvert = 354$	$\lvert S_T \rvert$=1,062

$G\%$	$M\%$	$e_L\%$	$U_0\%$	K_{aver}	$e_T\%$	T(sec)
0	46	1%	94%	5,482	6.68%	360
10	47	1%	95%	4,630	6.45%	356
25	50	0%	98%	2,514	6.27%	192
33	49	0%	98%	2,145	6.21%	196
50	50	0%	99%	1,644	6.06%	169
66	50	0%	98%	1,574	6.10%	156
75	50	0%	99%	1,517	6.21%	152
100	49	0%	99%	1,523	6.13%	156

Table 5.24: Cooling Schedule Γ for "Neither".

Dataset = Pima Indians	$\Gamma = \lvert S_L^j \rvert \cdot G\%$	$D = \lvert S_L^j \rvert \cdot M\%$
P=100	K=25,000	$x_0 = 25$
$\lvert S_L \rvert = 512$	$\lvert S_L^j \rvert = 20$	$\lvert S_T \rvert = 256$

$G\%$	$M\%$	$e_L\%$	$U_0\%$	K_{aver}	$e_T\%$	T(sec)
0	65	3%	63%	315	25.20%	84
10	65	3%	61%	367	25.10%	83
25	65	4%	60%	373	25.76%	89
33	65	4%	58%	371	25.10%	88
50	65	3%	60%	290	25.31%	83
66	65	3%	63%	233	25.10%	88
75	65	3%	58%	238	25.90%	86
100	65	3%	59%	309	26.20%	84

Table 5.25: Cooling Schedule Γ for Pima Indians.

Dataset = WBCD	$\Gamma = \lvert S_L^j \rvert \cdot G\%$	$D = \lvert S_L^j \rvert \cdot M\%$
P=100	K=25,000	$x_0 = 5$
$\lvert S_L \rvert = 455$	$\lvert S_L^j \rvert = 91$	$\lvert S_T \rvert = 228$

$G\%$	$M\%$	$e_L\%$	$U_0\%$	K_{aver}	$e_T\%$	T(sec)
0	64	1%	62%	6,489	0.55%	127
10	65	1%	63%	5,952	0.57%	125
25	64	1%	64%	5,312	0.79%	128
33	65	1%	65%	5,194	0.53%	122
50	63	1%	66%	5,065	0.48%	122
66	64	1%	64%	4,574	0.53%	117
75	70	1%	66%	4,609	0.55%	118
100	63	1%	64%	4,622	0.48%	117

Table 5.26: Cooling Schedule Γ for WBCD.

Our experimental results from Table 5.22 through Table 5.26 show that K_{aver} is related to the value of Γ, i.e if $\Gamma \approx D$, then fewer transitions K_{aver} are needed. We observe the best classification rate if $\Gamma = D$ for EI, "Neither", and Pima indians, and good approximations for IE and WBCD. This confirms our procedure of estimating D, as described in Section 4.7, item 3.

Similar non-monotonic behaviour of parameter Γ has been reported in the literature. Strenski and Kirkpatric [354], when analyzing a small instance of a graph partitioning problem, observed that optimal schedules may be non-monotonic, which is a surprising result, since the convergence proofs suggest a monotone decrease of the control parameter value. Moreover, Hajek and Sasaki [157] found for a small artificial intelligence problem that the control parameter values of finite-time optimal annealing schedules are all either 0 or ∞. Nonmonotonicity of optimal sequences for the control parameter is also reported by Boese and Kahng [57]. Aarts and Lenstra [3] state that the findings about nonmonotonicity of optimal sequences for the control parameter of finite-time optimal schedules are only proved for small instances, and is not clear if there is any impact on larger instances.

5.6 The Size of Threshold Circuits

What size of threshold circuit gives us the best classification rates? The parameter P denotes the size of depth-two circuits. The importance of this problem has been described in Section 2.3.3, and Section 4.7. We run experiments for P threshold units, and the results are presented in Table 5.27. The parameter settings are K=25,000 for all datasets, $x_0 = 5$ and $\Gamma = 0.75 \cdot |S_L^j|$ for IE and WBCD; $x_0 = 3$ and $\Gamma = 0.75 \cdot |S_L^j|$ for EI; $x_0 = 6$ and $\Gamma = 0.5 \cdot |S_L^j|$ for "Neither"; and $x_0 = 25$ and $\Gamma = 0.66 \cdot |S_L^j|$ for Pima Indians. We performed experiments for $P = \{10, 30, 48, 66, 80, 84, 100, 130, 154, 180, 252, 500\}$. The values of P have been chosen in such a way that we can compare depth-two circuits to depth-four circuits in the next chapter.

Table 5.27 displays the classification errors for all five datasets and all numbers P. The classification errors for all datasets are also shown in Figure 5.11. We observe an improvement and then stabilisation of error rates with increased circuit size. This is especially significant for the splice-junction datasets, where the size of sample sets is much larger, whereas in Pima Indians and WBCD datasets the stabilisation occurs for smaller values of P already.

As P varies for each dataset, our observations on the other monitored parameters can be summarized as follows:

1. for IE dataset: As P changes we observed for next parameters the stable

P	IE	EI	Neither	Pima	Wbcd
10	4.6%	3.1%	7.3%	28.2%	1.7%
30	4.1%	2.8%	6.4%	25.8%	1.1%
48	4.0%	2.8%	6.3%	25.6%	0.7%
66	4.0%	2.8%	6.2%	25.2%	0.6%
80	4.0%	2.8%	6.1%	25.1%	0.5%
84	4.0%	2.8%	6.1%	25.1%	0.5%
100	3.9%	2.7%	6.1%	25.1%	0.6%
130	3.9%	2.7%	6.0%	25.1%	0.6%
154	3.9%	2.7%	6.0%	25.4%	0.5%
180	3.8%	2.6%	5.9%	25.3%	0.5%
252	3.8%	2.6%	5.9%	25.3%	0.4%
500	3.8%	2.7%	5.9%	25.2%	0.4%

Table 5.27: Classification Error Rates on Test Sets.

values $e_L = 0\%$, $U_0 = 100\%$, $U_1 = 100\%$; K_{aver} ranges from 570 to 640 transitions, and $K_{max} = 13,602$. This confirms that K=25,000 transitions are adequate for training.

2. for EI dataset: As P changes we observed $e_L = 0\%$, and $U_1 = 100\%$, but U_0 reaches at most 98.2%; K_{aver} ranges from 1200 to 2400 transitions, and $K_{max} = 21,469$.

3. for "Neither" dataset: As P changes we observed $e_L = 0\%$, but U_0 ranges between 98.0% and 98.7%; K_{aver} ranges from 1200 to 2400 transitions, and $K_{max} = 24,609$.

4. for Pima Indians dataset: As a hard to learn dataset, even for a small size of sample sets, we obtained a learning error e_L in the range of 2.6%-3.1%, with a corresponding U_0 in the range of 62%-68%; due to the small sample sets, K_{aver} ranges from 200 to 300 transitions, however, there exist hard sample sets with $K_{max} = 22,219$.

5. for WBCD: As P changes we observed U_0 in the range of 55%-66%, with a few misclassified examples in each sample set, i.e. e_L in the range of 0.6%-0.8%; it takes a number of transitions to achieve this, as K_{aver} ranges between 4500 and 6500 transitions; $K_{max} = 24994$ remains close to K, a similar behaviour as we already observed in Section 5.3.

The diagrams of Figure 5.11 and values from Table 5.27 can be used for cut-off values of P for best classification rates. A detailed analysis of the circuit size for best classification rates will be presented in the next chapter.

5.7 ROC Analysis

Receiver operating characteristics (ROC) of a classifier is another measure of the performance of a classifier, also used for inspecting the performance of

the classifier for a specific parameter. ROC analysis was utilized in signal detection theory earlier [149]. A number of researchers [62, 106, 292] argue that using classification accuracy in order to compare classifiers is not an adequate method. This applies mainly when the number of negative cases is much greater than the number of positive cases. If we assume that for some data obtaining positive samples is very difficult, and we have a distribution of 95% negative cases and just 5% of positive cases. If the classifier correctly classifies all negative examples, then it achieves 95% classification accuracy, although it has no ability to classify the desired positive examples. Usually, a curve of false positive rate FP versus true positive rate TP is plotted, while a parameter is varied. The curve always goes through two points (0,0) and (1,1), where in the first point the classifier always gets all negative cases right, but it gets all positive cases wrong, while in the second point it gets all positive cases right but all negative cases wrong. The ideal point is (0,1) where the classifier always gets both the negative and positive cases right. Uniform random guessing can be displayed as a diagonal line from (0,0) to (1,1). Any classifier that wants to perform better than random guessing should be above that line, and for the comparison of classifiers we compare points on the ROC curve, where the part of ROC curve that is above all other curves denotes a better classifier. ROC points can also be used for parameter setting where the closer to (0,1) the better the parameter setting.

True positive TP and false positive FP rates are calculated by

$$TP = \frac{Pos^+}{Pos^- + Pos^+}, \tag{5.2}$$

and

$$FP = \frac{Neg^+}{Neg^- + Neg^+}, \tag{5.3}$$

where Neg^+ is the number of misclassified negative examples, Pos^- the number of misclassified positive examples, Pos^+ and Neg^- the numbers of correctly classified positive and negative examples, respectively.

There has recently been a growing interest in using *receiver operating characteristics (ROC)* curves in Machine Learning [59]. The analysis of rule learning methods by using ROC curves is presented in [125]. The area under the ROC curve is used as an evaluation criterion, instead of the commonly used classification accuracy. The interpretation of this area is discussed by Bradley [62], who noticed that the area under the curve can be seen as the probability of ranking a true positive example higher than a false positive. The area usually cannot be calculated easily. A method that can be used for calculating the area is the trapezoid method [62], which, however, underestimates the

area if only a few points are available. Other methods for approximating and interpreting the area are described in [260].

We use ROC analysis applied to x_0 as one of the crucial parameters in our work and we want to cross-check our selection of the size of sample sets. While ROC curves provide a method of finding the best classification point, the method has very limited use in our case, as it only defines the value of the parameter that maximizes the performance, without providing us with information about the relation of the parameter to other parameters. Therefore, monitoring parameters is unavoidable for extracting guidelines and relating a parameter to other parameters.

ROC curves are presented in Figure 5.12. Since most points are concentrated in a very small area of the graph, we also present along with each ROC graph a focus with respect to area under discussion. High performance is credited to WBCD, as it reaches a point very close to (0,1); the same applies to IE, EI and the "Neither" dataset. The Pima Indians ROC points reflect the low performance that we have seen in previous Sections.

x_0		2	3	4	5	6	7	8	10
IE	FP	0.018	0.024	0.024	0.024	0.025	0.027	0.025	0.029
IE	TP	0.888	0.906	0.908	0.914	0.915	0.916	0.918	0.918
IE	e_T	4.05%	4.05%	4.07%	3.87%	3.95%	4.04%	4.05%	4.15%
EI	FP	0.019	0.020	0.025	0.028	0.029	0.030	0.030	0.033
EI	TP	0.944	0.954	0.954	0.955	0.954	0.951	0.941	0.936
EI	e_T	2.75%	2.66%	3.02%	3.25%	3.30%	3.45%	3.70%	4.07%
"Neither"	FP	0.124	0.090	0.058	0.059	0.065	0.059	0.060	0.061
"Neither"	TP	0.960	0.956	0.937	0.937	0.943	0.933	0.929	0.921
"Neither"	e_T	8.08%	6.61%	6.27%	6.14%	6.06%	6.32%	6.59%	7.04%
WBCD	FP	0.018	0.019	0.017	0.014	0.013	0.013	0.014	0.013
WBCD	TP	0.993	0.999	0.999	0.999	0.994	0.991	0.993	0.989
WBCD	e_T	1.05%	0.71%	0.58%	0.55%	0.83%	1.02%	0.99%	1.15%

x_0		2	4	6	8	10	15	20	25
Pima	FP	0.078	0.090	0.098	0.102	0.095	0.093	0.096	0.084
Pima	TP	0.364	0.369	0.379	0.400	0.423	0.449	0.448	0.440
Pima	e_T	24.30%	28.00%	28.16%	27.73%	26.74%	25.39%	25.66%	25.12%

Table 5.28: ROC Points and Classification Errors for Sample Size Variations.

Table 5.28 presents the (FP,TP) points for the concentrated points of the ROC graph, and we compare them to the corresponding value x_0 of best classification that we found in Section 5.2. The comparison shows that indeed the best classification rates correspond to the closest to (0,1) points. The (FP,TP) data also reveal the high accuracy $TP = 0.999$ (almost 100% correct) of our WBCD method in predicting positive examples, which in breast cancer diagnosis is more important than predicting negative examples. At the same time, it reveals that the low classification accuracy in Pima Indians depends

	ΔK_{max}%			
x_0	IE	EI	Neither	Pima
x_0^{best-1}	5.7 %	0%	0.4%	1.7%
x_0^{best}	54.8 %	19.5%	11.6%	23.7%

Table 5.29: Training Confidence for Best Classification Rates.

very much on predicting positive examples, as it performs much better in predicting negative examples. This performance analysis of a classifier on positive and negative example is the main benefit of ROC analysis against the traditional method of just comparing classification rates.

5.8 Parameter Setting Paradigms

Optimization of parameter settings is a hard task where, actually, only approximated values that provide good classification rates can be found. Finding optimised parameters induces a combinatorial optimization problem by itself, which usually needs a large number of experiments for good approximation. For the tables presented in this chapter we performed approximately 2,000 experiments for finding guidelines for the four free parameters $\{x_0, K, \Gamma, P\}$. Based on these experiments, we tried to find paradigms for a priori setting values of these problem dependent parameters, according to the NFLT. We summarize in this section our findings:

1. Parameter x_0: We found that the value of this parameter is very important for the quality of classification accuracy. We recall from Section 4.7 that this parameter determines the size $|S_L^j|$ of training examples used for training threshold unit j, i.e. determines the sample complexity problem. Based on our experiments we identify three patterns for x_0.

   (a) Parameter x_0 is almost monotonic against classification error e_T. Except for Pima Indians, e_T decreases its value in almost every step of x_0, till a minimum is reached. Afterwards, any increase for x_0, is followed by increased values for e_T.

   (b) Parameter x_0 and best classification error e_T are related to the maximum transition step (K_{max}), where an accepted hypothesis of the simulated annealing method has lowered the misclassification error in the training data, in the following way: for IE,EI,"Neither" and Pima Indians we observe that the best classification error is achieved at step x_0^{best}, where $K_{max}^{best} \not\approx K$, whereas for the previous

step x_0^{best-1} we obtained $K_{max}^{best-1} \approx K$. We estimate a value δ_{x0}, where we may consider that $K_{max} \approx K$. Table 5.29 shows the percentage of difference (ΔK_{max}) of K_{max} from K for the four datasets, for values of x_0^{best} of the best classification error and for values x_0^{best-1} from the previous step. We estimate from Table 5.29 that ΔK_{max} can be lower bounded by 11.6% of the "Neither" dataset for considering $K_{max} \not\approx K$, and ΔK_{max} can be upper bounded by 5.7% of the "IE" dataset for considering that $K_{max} \approx K$ Therefore, we achieve for the four datasets best classification rates, if $K_{max}^{i} < K - \delta_{x0} \cdot K$, where $\delta_{x0} \geq 0.1$, and for $x_0^{i-1} < x_0^{i}$ we obtain $K_{max}^{i-1} \approx K$.

(c) Parameter x_0 and best classification error e_T are related to the number of units that achieve zero classification error in the training data (U_0) in the following way: for IE, $U_0 = 100\%$, i.e. all units are able to learn the training data with zero error; for EI and "Neither" $U_0 \geq 98\%$, whereas U_0 achieves 100% in the next step of x_0; for Pima Indians we cannot achieve the same high values, however, U_0 for best classification error has considerable higher value than in previous steps of x_0; WBCD is the only dataset that increasing values of U_0 leads to worse classification error e_T, having the best value for e_T for $U_0 = 56\%$. This pattern is related to fitting and overfitting properties, where we try to maximise the number of units that achieve zero classification error without decreasing the generalization quality. The trade between learning accuracy and generalisation accuracy is common in Machine Learning as described in Section 2.1.2. The process seems to suggest the maximum value of U_0 for IE, the closer to maximum values for EI, "Neither", a considerable high value (compared to previous steps of x_0) for Pima Indians, whereas overfitting is unavoidable for WBCD after some certain value for U_0.

2. Parameter K: The predefined length K of the inhomogeneous Markov chain determines the number of transitions in the neighbourhood structure when searching for adopting better hypothesis in the simulated annealing process. One may relate this parameter to learning effort by relating long inhomogeneous Markov chain to better fitting the training data. For IE, we obtained K_{max}=11,300 transitions therefore according to the above estimations for parameter x_0 values of K >12,500 are acceptable values for adequately searching the neighbourhood structure, which is demonstrated in our experiments by obtaining stable best classi-

fication rate for $K \geq$15,000. Dataset EI, obtains high classification rates $e_T < 2.70\%$ for $K \geq$15,000, with slight variations close to this value for larger inhomogeneous Markov chains. Dataset "Neither" obtains best classification rate for K =25,000. Pima Indians obtains best classification rate for K =50,000, whereas very close to the best classification error is obtained for K =25,000. WBCD obtains the best classification error for K =15,000. The error $e_T = 0.44\%$ is the best reported in the literature for WBCD. The initial value K =25,000 that we used in our experiments seems to be adequate inhomogeneous Markov chain length for searching the neighbourhood structure. Larger inhomogeneous Markov chains lead to overfitting for "Neither" and Pima Indians. We run experiments with extremely long Markov chains (K=1,500,00) for EI and WBCD while trying to minimize U_0 for EI, and while trying to achieve $K_{max} << K$ for WBCD. No improvement in the classification error has been obtained for both datasets for such long inhomogeneous Markov chains. Investigating early stopping for K no improvement in e_T was obtained with respect to lower values in any of the parameters U_0, U_1, ΔK_{max}, ΔK_{aver}. Comparing in Table 5.21 the results on classification error from variations of x_0 and K we found that variations in x_0 have larger impact to e_T than variations of K. We may conclude that an initial value of K=25,000 is adequate for searching the whole neighbourhood structure and leads to high classification rates in the five datasets and therefore we may suggest for new datasets that we may start experiments for K =25,000, unless first experiments for $x_0 = 2$ result to $Kmax << 25,000$, where if we want to speed up the process, we may adopt a new K, such that $K \approx K_{max}$.

3. Parameter Γ: For our experiments we used as an upper bound for Γ the *maximum escape depth* pattern by running preliminary experiments for each dataset. Maximum escape depth lead to values of $\Gamma(IE) = 0.75 \cdot |S_L^j(IE)|$, $\Gamma(EI) = 0.75 \cdot |S_L^j(EI)|$, $\Gamma(\text{"}Neither\text{"}) = 0.5 \cdot |S_L^j(\text{"}Neither\text{"})|$, $\Gamma(Pima) = 0.66 \cdot |S_L^j(Pima)|$, and $\Gamma(WBCD) = 0.75 \cdot |S_L^j(WBCD)|$. We studied for each dataset the variations of e_T with respect to Γ and we found that the relation is nonmonotonic, a pattern that is also reported in the literature as described in Section 5.5. Maximum escape depth results to high classification rates and since it is difficult to identify other patterns for a priori setting parameter Γ, we may use maximum escape depth for estimating Γ.

4. Parameter P: The next chapter analyzes the effect of parameter P (the number of perceptrons) with respect to the size and the depth of the cir-

cuit. Experiments in this chapter show that no pattern can be identified from the monitored parameters as listed in Section 5.6.

Based on the above findings we suggest the following parameter setting paradigms:

1. For depth-two circuits of size P

2. Run m experiments for maximum sample size with $x_0 = 2$, for P, for maximum $\Gamma = |S_L^j|$, and an adequate number of transitions K (K=25,000) for determining Γ and K values of the simulated annealing process, for our next experiments; monitoring the maximum escape depth D_{esc} allows us to set $\Gamma = D_{esc}$ for the rest of experiments; if K_{max} <<25,000, we set for the rest of our experiments $K = max\{K_{max}^i\}$, where $i = 1, ..., m$;

3. To estimate the size of sample sets, we proceed as follows: For settings P, K, $\Gamma = D_{esc}$, we start with values of $x_0 = 2, 3, ...$ and perform trial experiments, monitoring K_{max} and U_0; since in our previous experiments (except for Pima Indians) we observed that the classification error decreases almost monotonically, before an increase occurs, we suggest to monitor the monotonicity of the classification rate until an increase occurs for a number of subsequent steps;

4. If one of the following conditions state true for a number of n experiments on x_0, then the current value of x_0 is selected:
   a) An increase in classification error occurred for $x > x_0$ or
   b) $K_{max}^i < K - \delta_{x0} \cdot K$, where $\delta_{x0} \geq 0.1$, and for $x_0^{i-1} < x_0^i$ we observed $K_{max}^{i-1} \approx K$ or
   c) $U_0 = 100\%$

These paradigms will be extended in the next chapter, after analysing parameter P.

5.9 Comparison of Results

The quality of results obtained in our work can be compared to the highest results reported in the literature. More specific:

Regarding the IE, EI and "Neither" dataset, we obtained best error rates 3.7%, 2.6% and 5.9%, respectively; or else, correct classification rates 96.3%, 97.4% and 94.1%, respectively. Previous results on 25% of samples used as

test examples [362], therefore considerable smaller test sets than in our experiments, report an error rate on IE, EI and "Neither" classes of 7.55%, 5.74% and 3.99% (best results obtained by different methods), respectively. From these methods we mention the back propagation method achieving 10.75%, 5.74% and 5.29%, respectively, nearest neighbour method achieving 9.09%, 11.65% and 31.11%, respectively, and ID3 algorithm achieving 13.99%, 10.58% and 8.84%, respectively. Anastasiadis et al. [23] achieved 100% combined correct classification on test data representing 25% of the sample set. In [336], a combined correct classification rate of 92.3% is reported. Rampone reports classification errors 4.3%, 3.1% and 4.9% for IE, EI and "Neither" for the same number of test samples as in our work, where our results outperform on IE and EI, but not on "Neither". In [60], approximately the same number of test samples is also used for the evaluation of artificial nonmonotonic neural networks (ANNNs) on the SJGSD set. Our classification results outperform the results obtained on EI and IE by all eight methods presented in Fig. 16 of [60], (for "Neither", it is difficult to judge from Fig. 16, and for ANNNs it suggests zero errors, which is not mentioned explicitly in the text). In [244] combined classification errors of 2.9% and 8.1% are reported by using SVM and C4.5 respectively, however, these errors are obtained for very small test sets by using the *10-fold cross validation methodology* [270], i.e. dividing the dataset into 10 disjoint sets where nine sets are used for training and one set for testing, therefore, providing testsets with just 10% of the data.

Regarding the Pima Indians dataset the accuracy of various classifiers varies from 72% to 82%, where the highest values were obtained when all missing data positions were removed from the dataset [383]. Since we used the dataset without treating the missing data positions it is difficult to draw conclusions on the accuracy of our classifier compared to the best in the literature.

Regarding the WBCD dataset the 0.44% classification error, or else the 99.56% correct classification rate obtained in our work is the highest reported in the literature. Many researchers [6, 23, 77, 216, 285, 337, 386] (to name just a few) have tackled this database with reported results ranging from 90% to about 99%. Anastasiadis et al. [23] using the same classifier as for the SJGSD, achieved on WBCD 97.5% correct classification for 25% of the dataset as test data. The closest to our classification error is the 99.48% reported in [121] is obtained by combining a special type of perceptron (the Adatron) and SVM, where, however, the test set consists of just 10% of the data following the 10-fold cross validation methodology. Following the same methodology, Opper and Winther in [277] report 3.07% classification error by using SVM, 2.93% by using naive mean field theory, 3.4% by using multilayer neural networks,

4.0% by using linear discriminant, 4.1% by using RBF neural networks, and 5.8% by using CART decision trees. Cristianini and Campbell in [89] by using SVM and a distribution of 550 data for training and 133 for testing, report 95.56% classification accuracy. Therefore, our classification results on WBCD are (to the best of our knowledge) the highest reported in literature, obtained on test set consisting of 33% of the dataset, thus, on much larger test sets than in [23, 89, 121, 277].

Summarizing about the quality of our results, we note that the classification error results obtained by the LSA machine **outperform all other methods** in the literature for WBCD and are in **the highest rank of reported results** in the literature for the IE and EI dataset. Although simulated annealing convergence properties require infinite time, our experiments with the LSA machine show that the best classification rates can be very fast, once the size of sample sets is estimated. Therefore the LSA machine is fast and can obtain results that **outperform or are at least competitive** to best classification results in the literature.

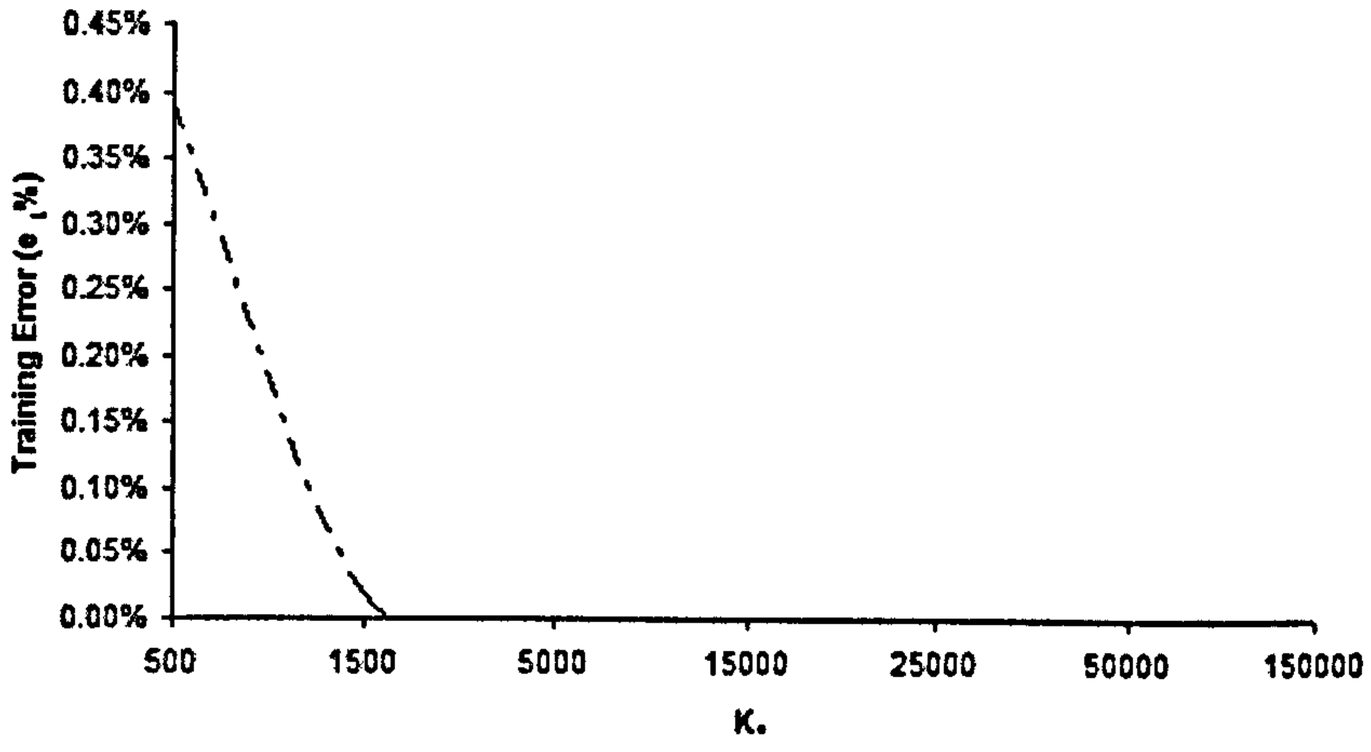


Figure 5.6: IMC Length for IE.

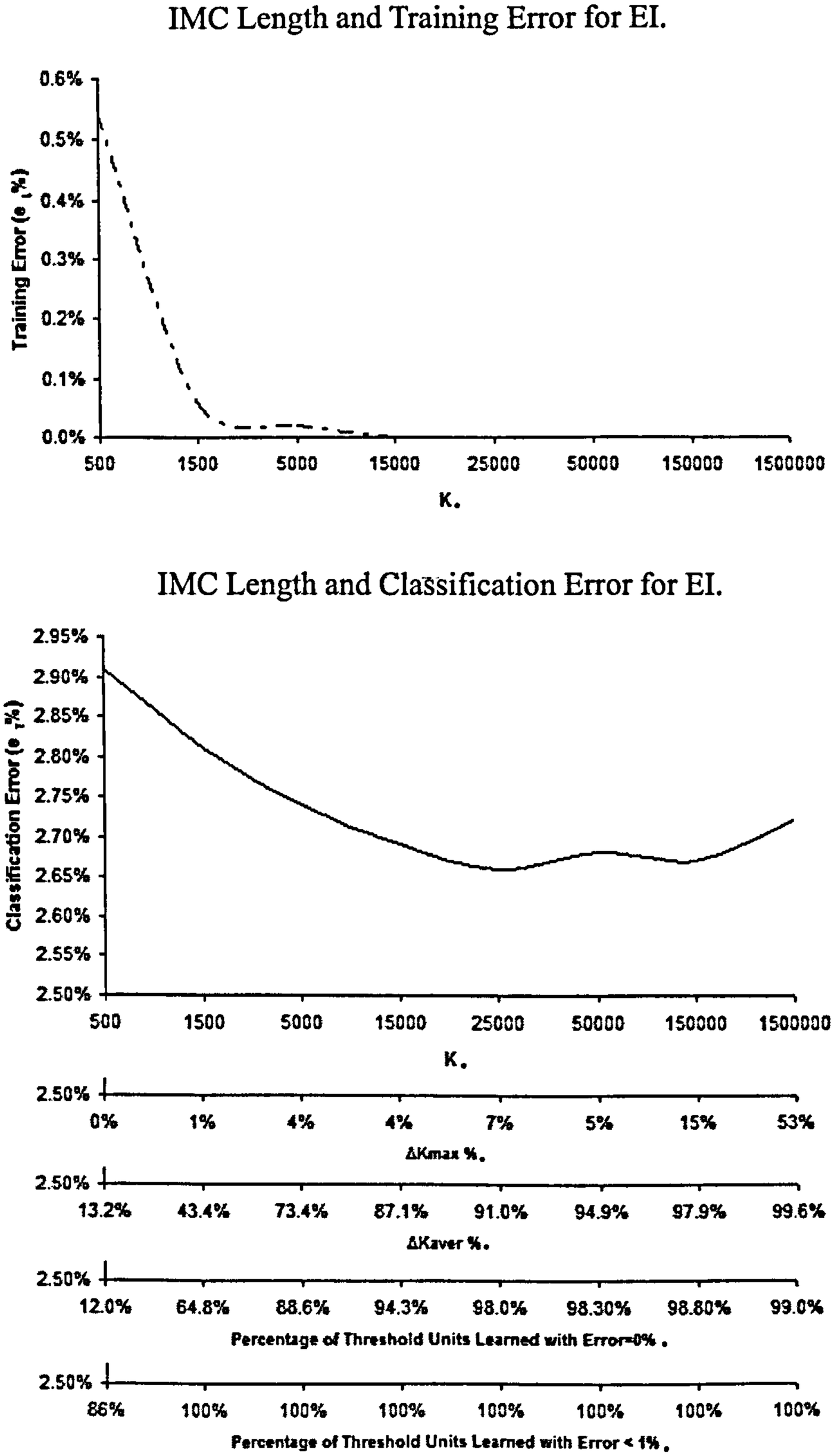


Figure 5.7: IMC Length for EI.

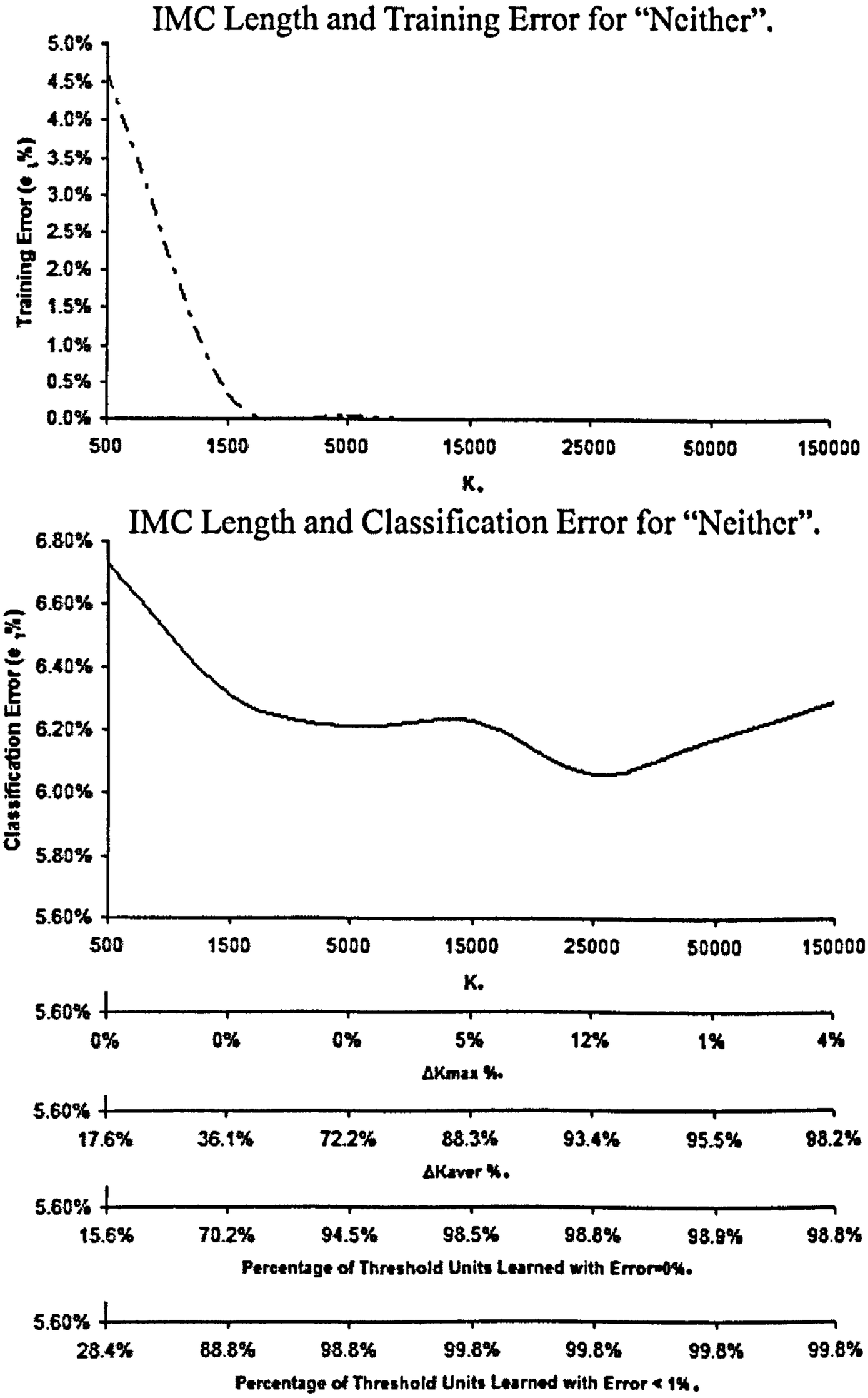


Figure 5.8: IMC Length for "Neither".

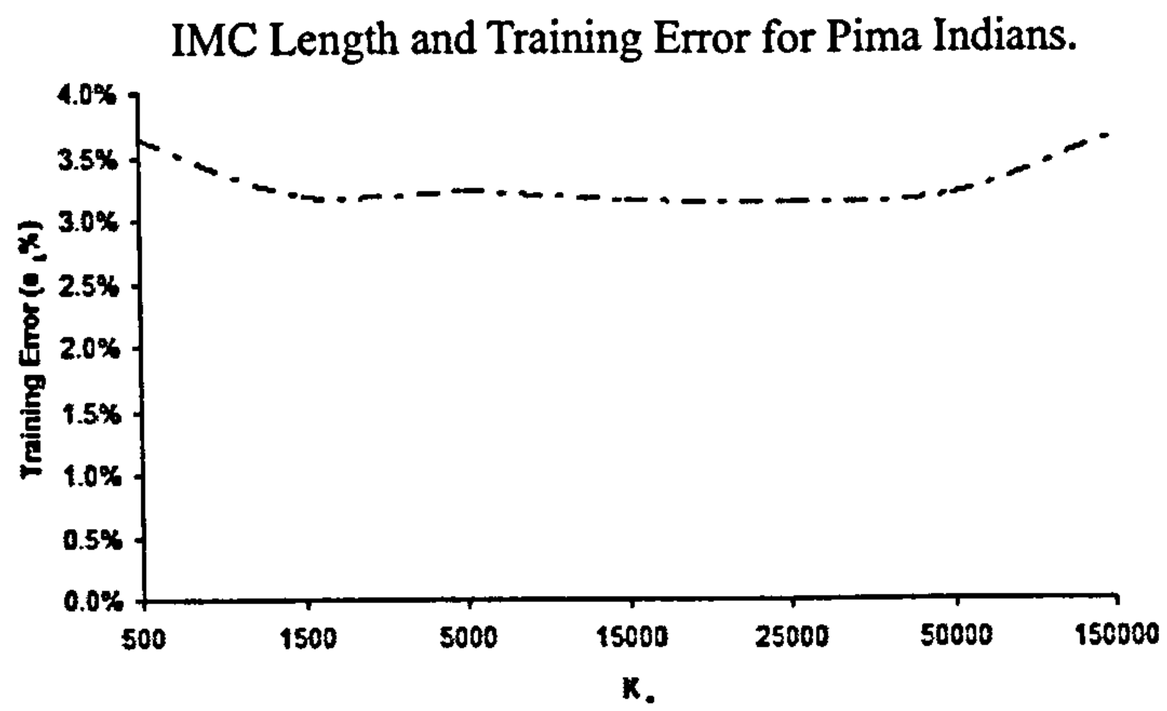
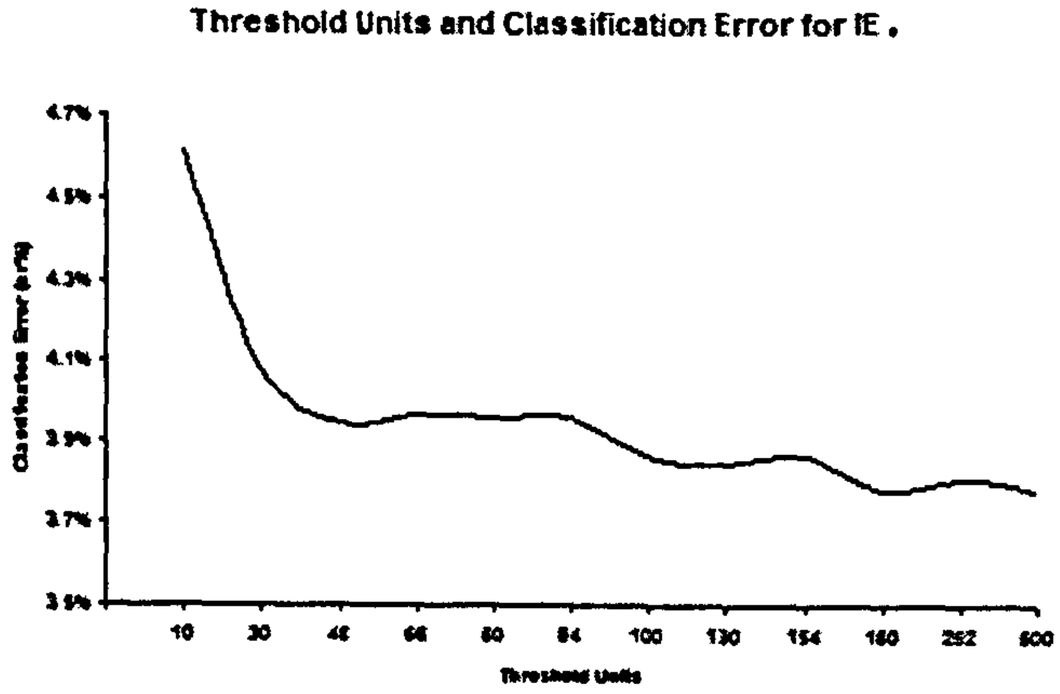
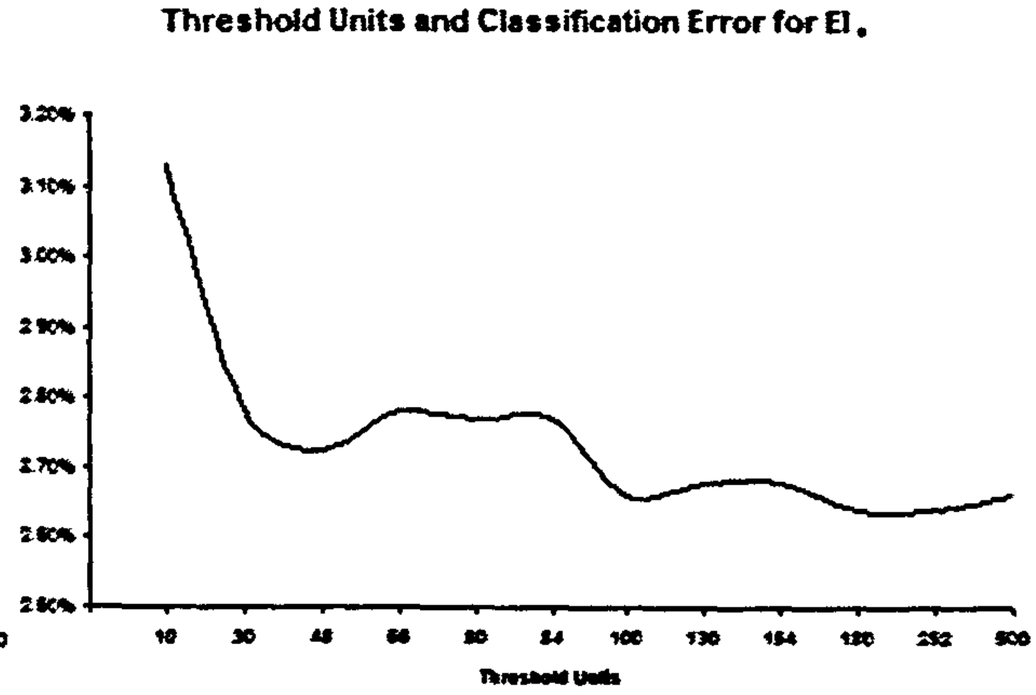
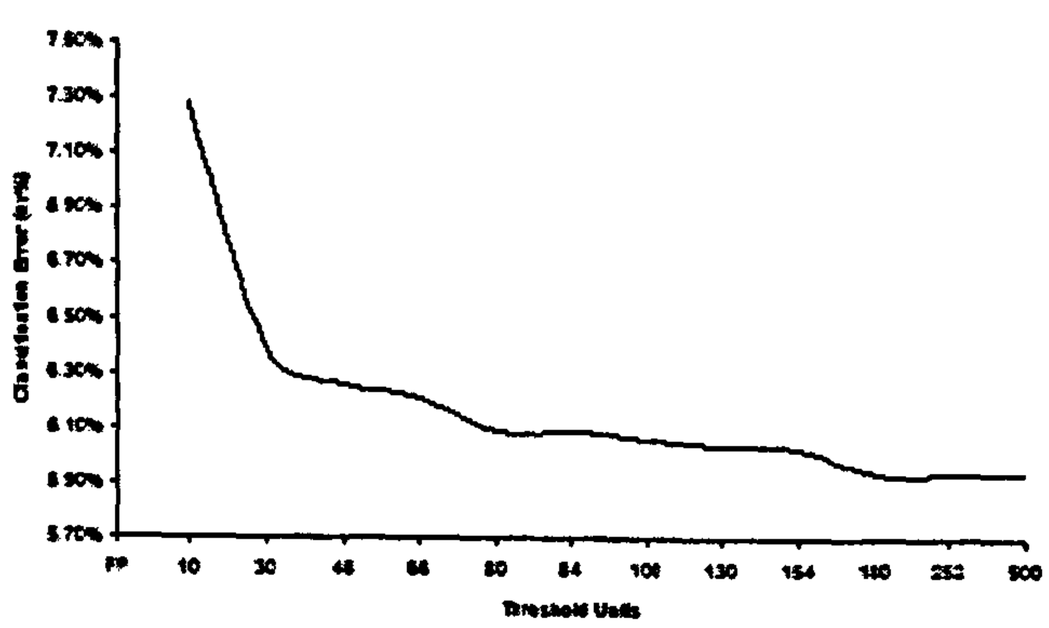


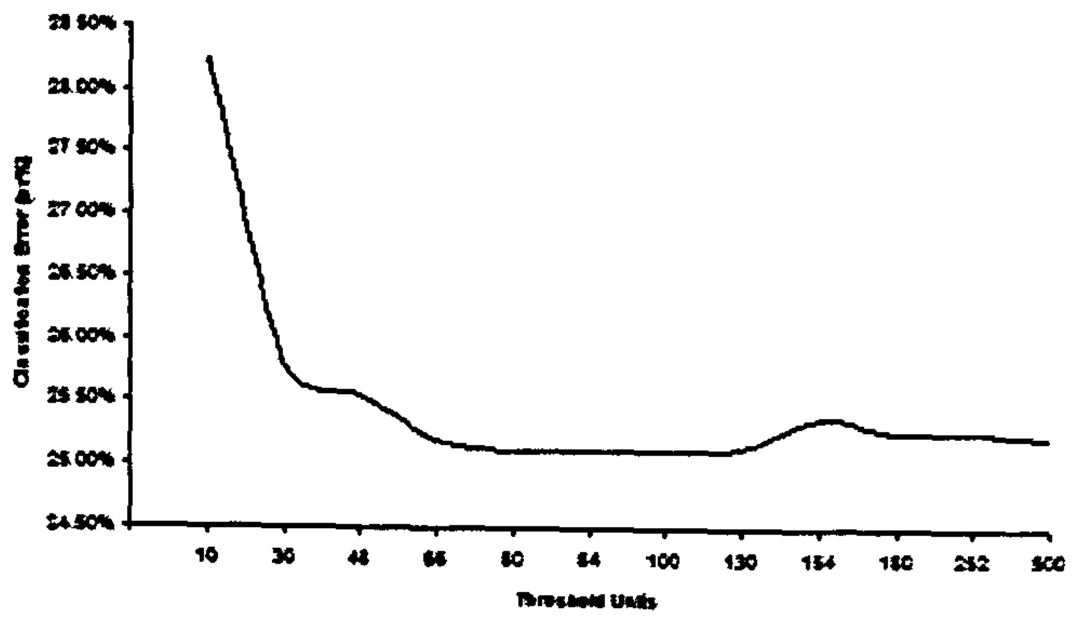
Figure 5.9: IMC Length for Pima Indians.

Figure 5.10: IMC Length for WBCD.









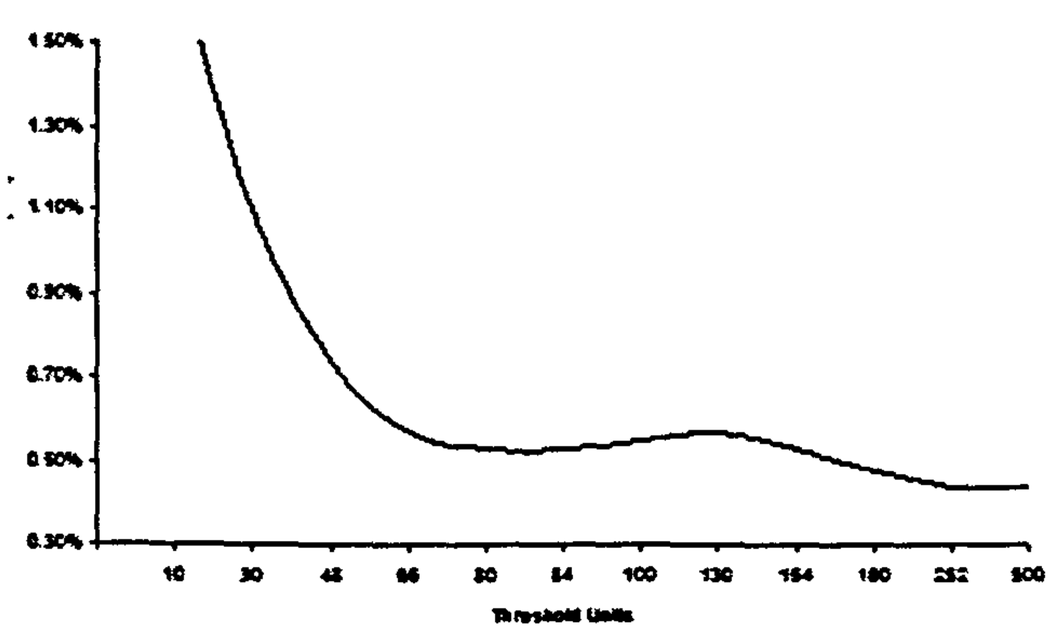
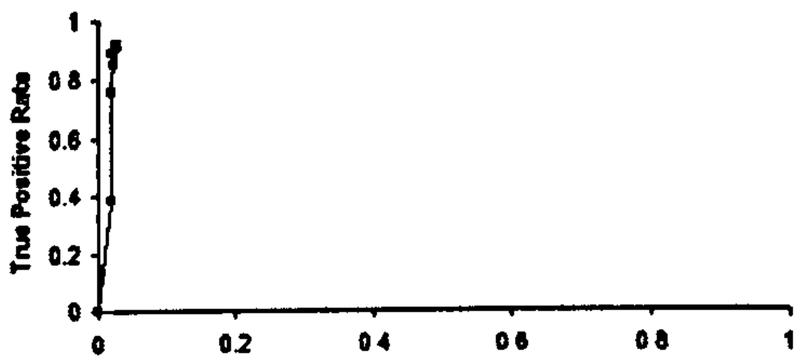
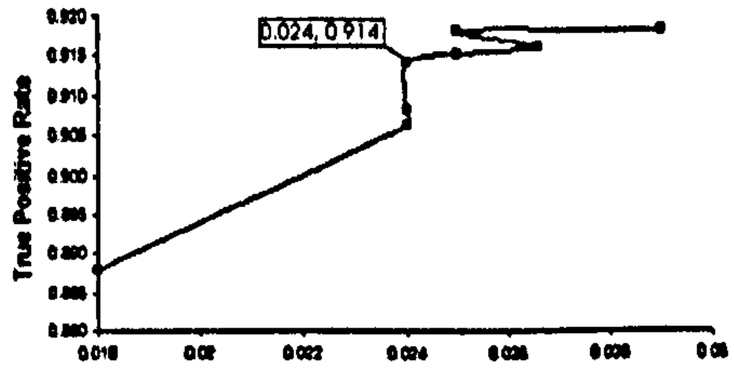


Figure 5.11: Size of Threshold Units and Classification Error.

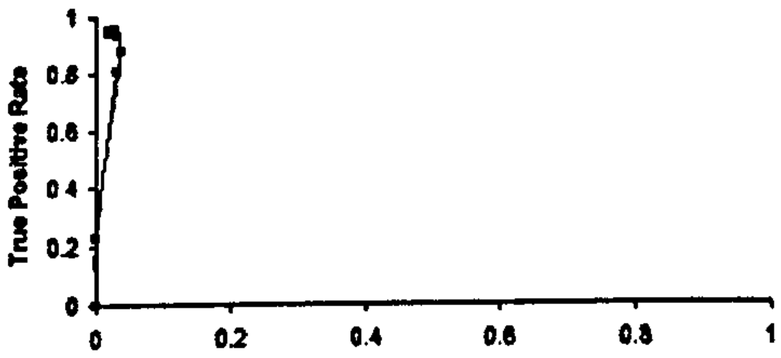
ROC Graph for Parameter x_0 on IE.



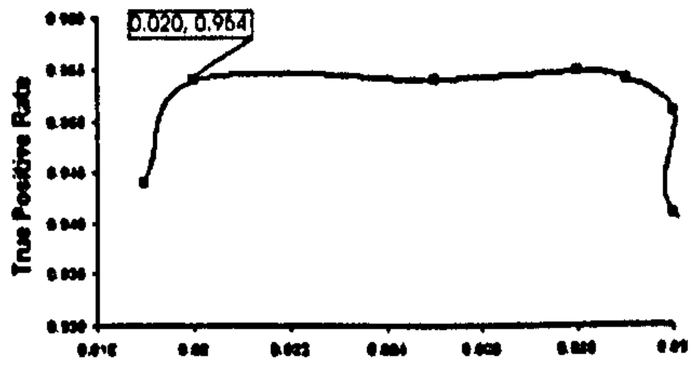
ROC Graph for Parameter x_0 on IE.



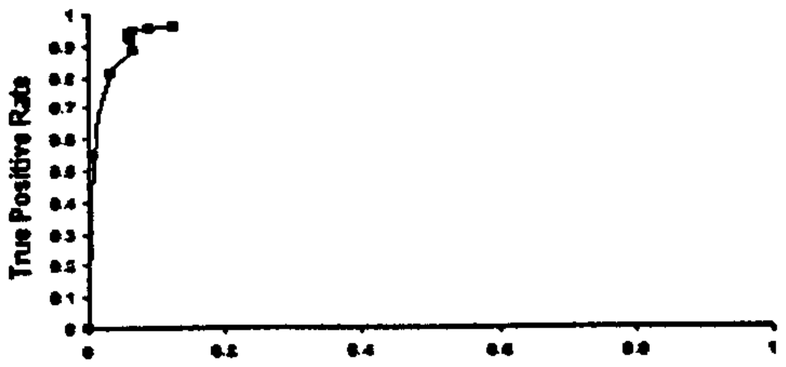
ROC Graph for Parameter x_0 on EI.



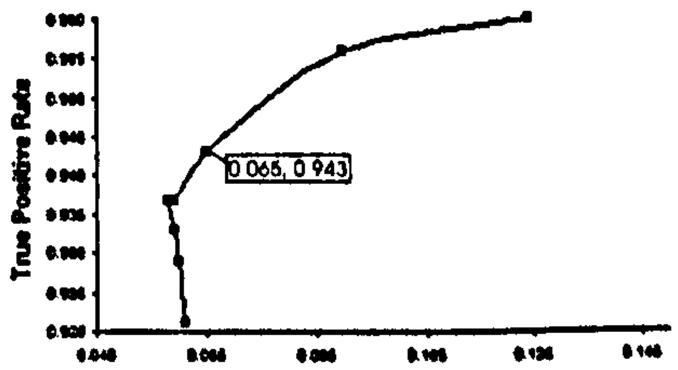
ROC Graph for Parameter x_0 on EI.



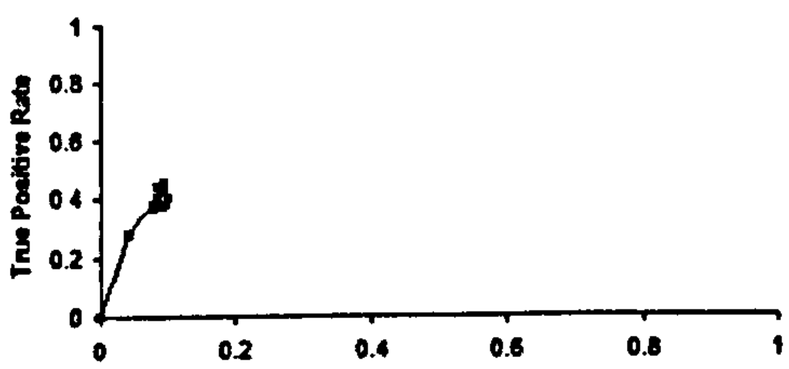
ROC Graph for Parameter x_0 on "Neither".



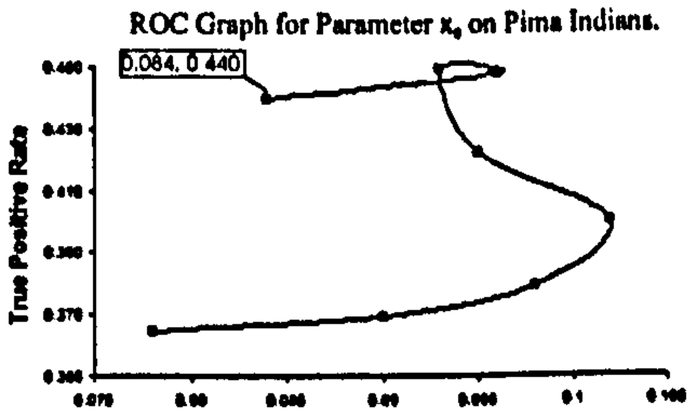
ROC Graph for Parameter x_0 on "Neither".



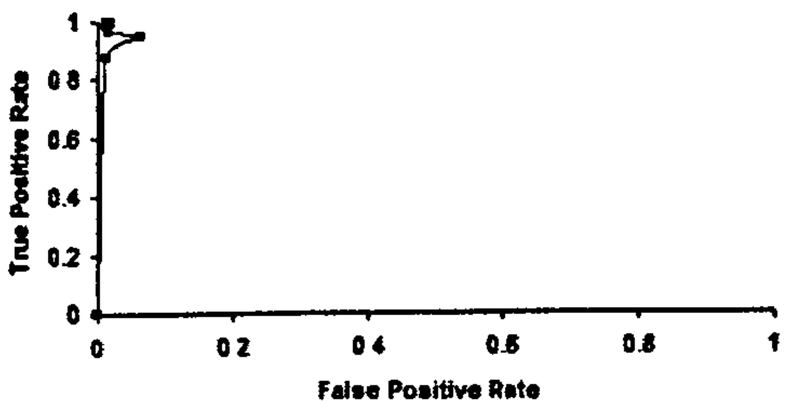
ROC Graph for Parameter x_0 on Pima Indians.



ROC Graph for Parameter x_0 on Pima Indians.



ROC Graph for Parameter x_0 on WBCD.



ROC Graph for Parameter x_0 on WBCD.

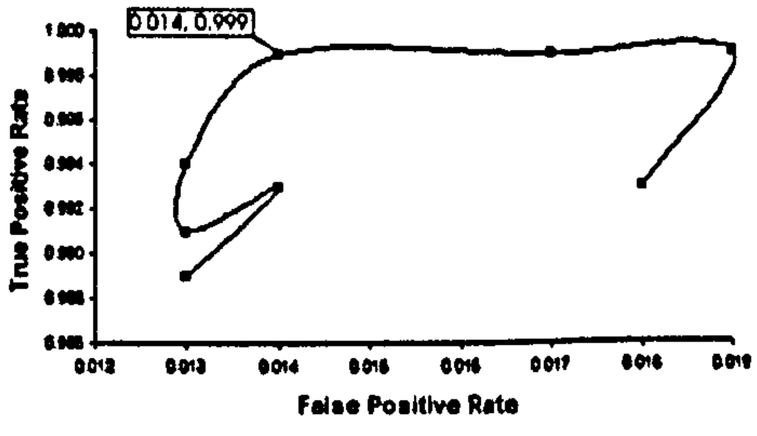


Figure 5.12: ROC Graphs in Relation to Sample Size.

Chapter 6

Circuit Complexity of Classifications

In this chapter, we investigate the circuit complexity of classification problems in a Machine Learning setting, i.e. we attempt to find some rule that allows us to calculate *a priori* the number of threshold units that is sufficient to achieve a small error rate after training a circuit on sample data S_L. We recall that the particular threshold units are computed by a combination of the classical perceptron algorithm with a specific type of stochastic local search; cf. Section 4.1. The circuit complexity is analysed for depth-two and depth-four threshold circuits, where we introduce a novel approach to compute depth-four circuits. We use again the same classification problems *cp* from the UCI Machine Learning repository [262], as studied in the previous Chapter. We perform two types of experiments, one by using the same set of values $\{x_0^{cp}, K_{cp}, \Gamma_{cp}\}$ as studied in the previous Chapter, and one by using a set of values obtained by preliminary experiments on $\{P_{cp} = 50, K_{cp} = 20000\}$. It is important to note that we aim at problems that are not linearly separable, even for relatively small subsets of the sample set, which are used to train single threshold gates of the circuit (otherwise, the "pure" perceptron algorithm would be more appropriate).

6.1 Circuit Depth vs Circuit Size

The effect of circuit depth on circuit size is one of the hardest problems in theoretical computer science. The problem emerged already in the discussion about perceptrons [268] in the context of the circuit complexity of $XOR(x_1, ..., x_n)$. To identify sequences of Boolean functions $\{f(x_1, ..., x_n)\}_{n=n_0}^{\infty}$ with "superpolynomial" (or exponential) gate number in constant depth circuits of unbounded fan-in gates is a very difficult problem and only slow progress has been made over the past decades. For example, Razborov and Widgerson

[300] designed a sequence of functions that requires at least the superpolynomial number $n^{\Omega(logn)}$ of threshold gates in any circuit of depth 3. Furthermore, it has been shown [22, 72] that computing the permanent of an $n \times n$ matrix requires a superpolynomial number of gates in any threshold circuit of constant depth; for more information about the computational power of small depth circuits, see [373]. Some guidelines for the network size are proposed by Duda et al. [106], where for a multilayer classifier with backpropagation the number of hidden units is proposed to be chosen in such a way that the number of total weights in the classifier is roughly $S_L/10$.

We specifically investigate the circuit complexity inherent to a given problem in terms of the number of threshold gates. Let $\mathcal{C}_d^{cp}[e = R]$ denote circuits that achieve an error rate of R on test samples of problem *cp*. We compare the "typical" gate complexity, i.e. the number of perceptrons N_{per}, in circuits $\mathcal{C}_2^{cp}[e = R]$ and $\mathcal{C}_4^{cp}[e = R]$ for R close to the lowest error rates we obtain on problem *cp*. Since it seems to be very difficult to find sequences of complex, explicitly defined functions even for small values of d, one can ask whether the inherent complexity of datasets from real-world problems is actually relatively low, i.e. if the datasets allow circuit representations of low complexity for a sufficiently large circuit depth $d > 2$ and unbounded fan-in gates. We note that here we are dealing with particular functions of fixed variable numbers, not with sequences of functions, and our $\mathcal{C}_d$ circuits cover only a small range of all possible depth d circuits of unbounded fan-in linear threshold gates.

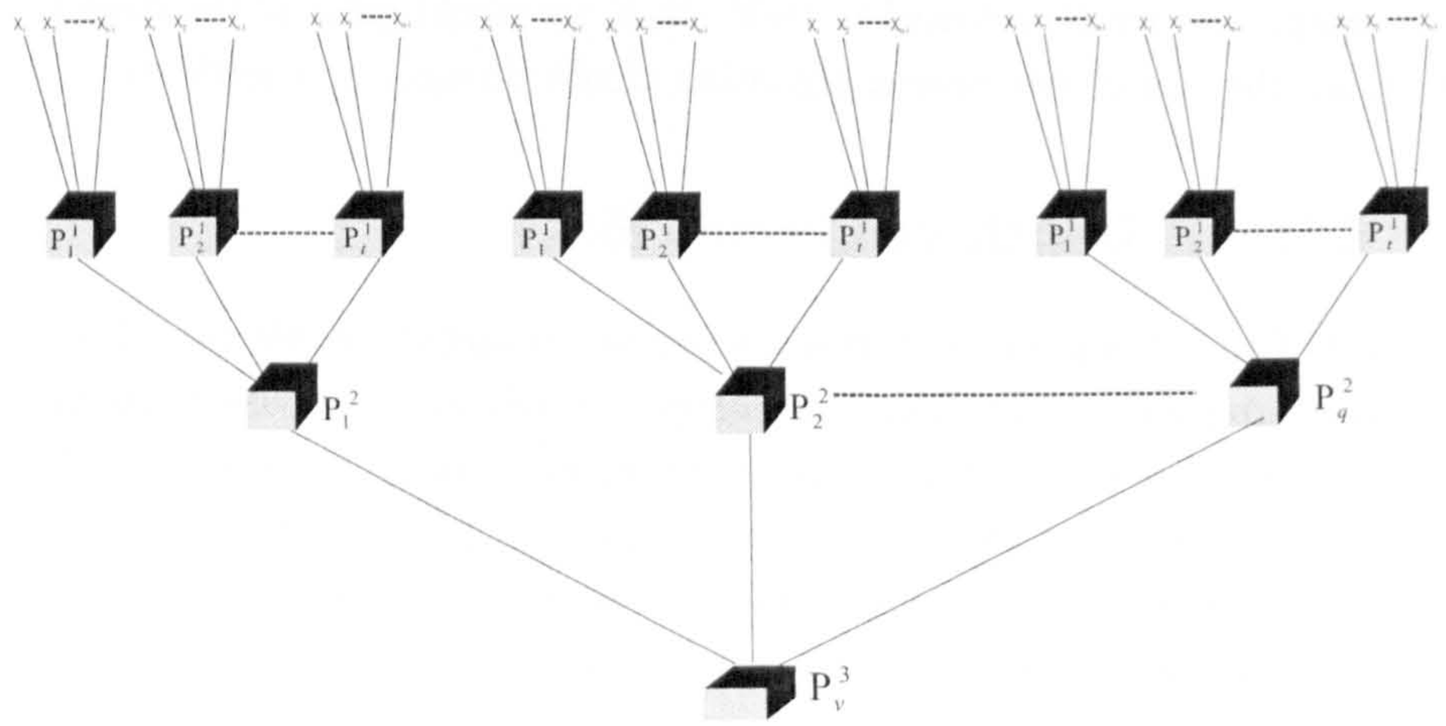


Figure 6.1: Pseudo Depth-three Circuit.

6.2 The Generation of Classification Circuits

There exist many possible $\mathcal{C}_d$ circuits of unbounded fan-in linear threshold gates of depth d with different network connectivities. We performed experiments with network connectivities as in Figure 6.1, where the q units of the third layer (depth-two) combine the results of the t units at layer two (depth-one). The q output gates are then combined by a voting function at depth-three, producing a depth-three network. Our experimental results turn to statistically averaging at depth-three the overall output of depth-one, and therefore such network architectures would collapse to depth-two networks of $q \cdot t$ units. Thus, extensions in depth cannot be done without actually training units at larger depths $d \geq 2$. Therefore, we will call such a network a *pseudo depth-three* network.

We will introduce a novel method for expanding circuits to larger depths. The method for extending the depth in can also be used in any feedforward neural network. The approach is applicable to circuits of any even depth $d = 2 \cdot j$; however, the circuit size increases exponentially in d. For circuits of depth $d \geq 2$ we proceed as follows: A given sample set $\mathcal{S}$ is split into two disjoint subsets $\mathcal{S}_{\mathrm{L}}$ and $\mathcal{S}_{\mathrm{T}}$, where $\mathcal{S}_{\mathrm{L}}$ is used to calculate the approximating threshold circuit, and the rest of the samples $\mathcal{S}_{\mathrm{T}}$ is left for testing purposes. The set $\mathcal{S}_{\mathrm{L}}$ is then further partitioned into $d/2$ subsets $\mathcal{S}_{\mathrm{L}}^j$, $1 \leq j \leq d/2$, and we apply the following recursive procedure: $\mathcal{S}_{\mathrm{L}}^1$ is used to calculate N_1 threshold circuits $\mathcal{C}_1$ of depth 2. The outputs of a single subcircuit $\mathcal{C}_1$ are combined by a voting function with a pre-determined threshold. Then, if the N_{j-1} circuits $\mathcal{C}_{j-1}$ of depth $2 \cdot (j-1)$ have been calculated already, we generate from $\mathcal{S}_{\mathrm{L}}^j$ randomly N_j primary training sets $T_{[j,i]}^{\mathrm{pr}}$, $1 \leq i \leq N_j$, and each $T_{[j,i]}^{\mathrm{pr}}$ applied to circuits of type $\mathcal{C}_{j-1}$ generates a secondary training set $T_{[j,i]}^{\mathrm{snd}}$. The sets $T_{[j,i]}^{\mathrm{snd}}$ are then used together with the classification information from $\mathcal{S}_{\mathrm{L}}^j$ to calculate the perceptron gates at depth $2 \cdot j - 1$, which are then again combined by a voting function at depth $2 \cdot j$.

Note that in Chapter 5 we used the notation $\mathcal{S}_{\mathrm{L}}^j$ when splitting the training set into P subsets (randomly sampled from S_L) for training P threshold units, where $1 \leq j \leq P$. Since here we split the training set into equal subsets for training different depths, the notation $\mathcal{S}_{\mathrm{L}}^j$ is now related to training sets used for training depth $2 \cdot (j-1)$.

After training the preceding levels of a circuit all preceding input gates have fixed their weights. Subsequently, when applying to the fixed subcircuits the next primary training sets $\mathcal{S}_{\mathrm{L}}^j$, $j > 1$, we generate new secondary training sets $T_{[j,i]}^{\mathrm{snd}}$, based on the outputs of the preceding level, for training the next level. In this way, we have a real increase in depth, as information is passed

from one level to the next one. In other words, generating samples based on results of the previous depth is like training perceptrons to increase the output significance of those subcircuits in the previous depth that perform well, and to reduce the significance of those subcircuits that perform worse.

The idea behind training next depths with generated samples from previous depths is similar to the idea of the *credit assignment problem*, where likewise we try to find in case of an error which of the possible contributors caused it, and its significance is decreased, while we increase the significance of those that contribute to the correct answer. Since the sample sets used for training depth-d are output gates of depth-$(d-1)$, the adjustment of depth-d weights to fit the sample sets is equivalent to learning and giving credits to the significant contributors from depth-$(d-1)$.

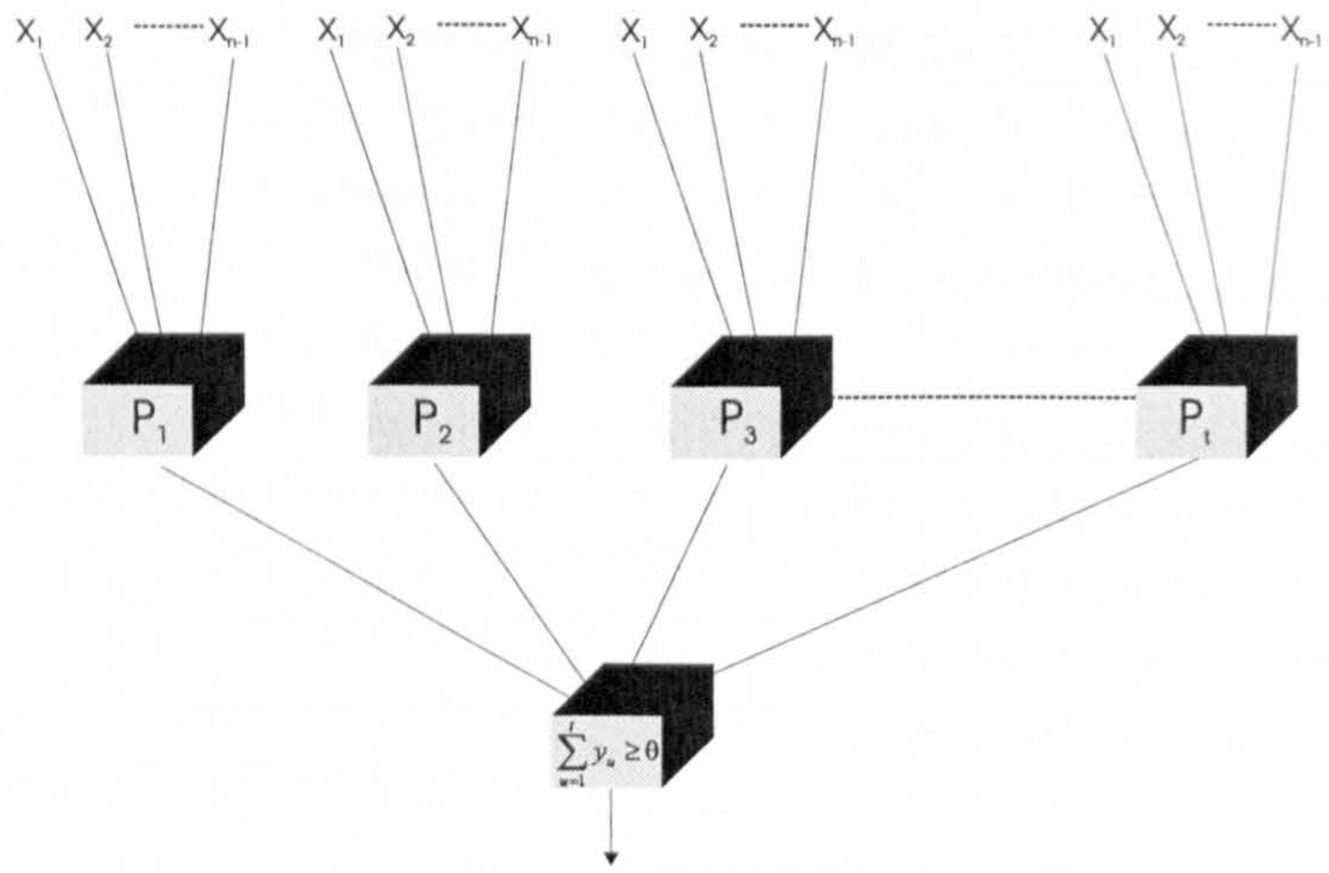


Figure 6.2: Depth 2 Circuit $\mathcal{C}_2$ with $t+1$ Threshold Gates.

6.3 The Structure of Classification Circuits

We use a homogeneous structure of circuits consisting of at most three types of gates: perceptrons calculated from $\mathcal{S}$ (see Chapter 5), fixed type voting functions, and fixed type counting functions. The structure of classification circuits $\mathcal{C}_d$ is described by induction:

1. $d = 2$: $\mathcal{C}_2$ consists of t perceptrons, each of them connected to the n input variables. The binary outputs of the t threshold gates (perceptrons) are the inputs to a simple voting function $\sum_{u=1}^{t} y_u \geq \vartheta$; see Figure 6.2. The threshold $\vartheta \geq t/2$ is a parameter of our approach.

2. $d = 2 \cdot j > 2$: $\mathcal{C}_d$ is built from $q \cdot t$ modified copies of circuits of type

$\mathcal{C}_{d-2}$. The modification relates to the output gate of circuits of type $\mathcal{C}_{d-2}$: The voting function is substituted by a counting function $\sum_{u=1}^{t} y_u$, i.e. the outputs are now integers (rather than binary values only), and the circuits are denoted by $\mathcal{C}'_{d-2}$. We take q copies of type $\mathcal{C}'_{d-2}$ to constitute the inputs to a single threshold function (perceptron). The t binary outputs of the threshold gates (each depending on q variable inputs) are again the inputs to the voting function $\sum_{u=1}^{t} y_u \geq \vartheta$; see Figure 6.3.

We note that perceptrons are "located" at depths $d' = 2 \cdot j - 1$ only, $j \geq 1$. By simple calculations we obtain the following formula for the number $N(d, d')$, $d \geq d' + 1$, of perceptrons at depth $d' = 2 \cdot j - 1$ in a circuit C_d:

$$N(d, d') = (t \cdot q)^{\frac{d}{2}-j} \cdot t, \quad \text{for } d' = 2 \cdot j - 1. \tag{6.1}$$

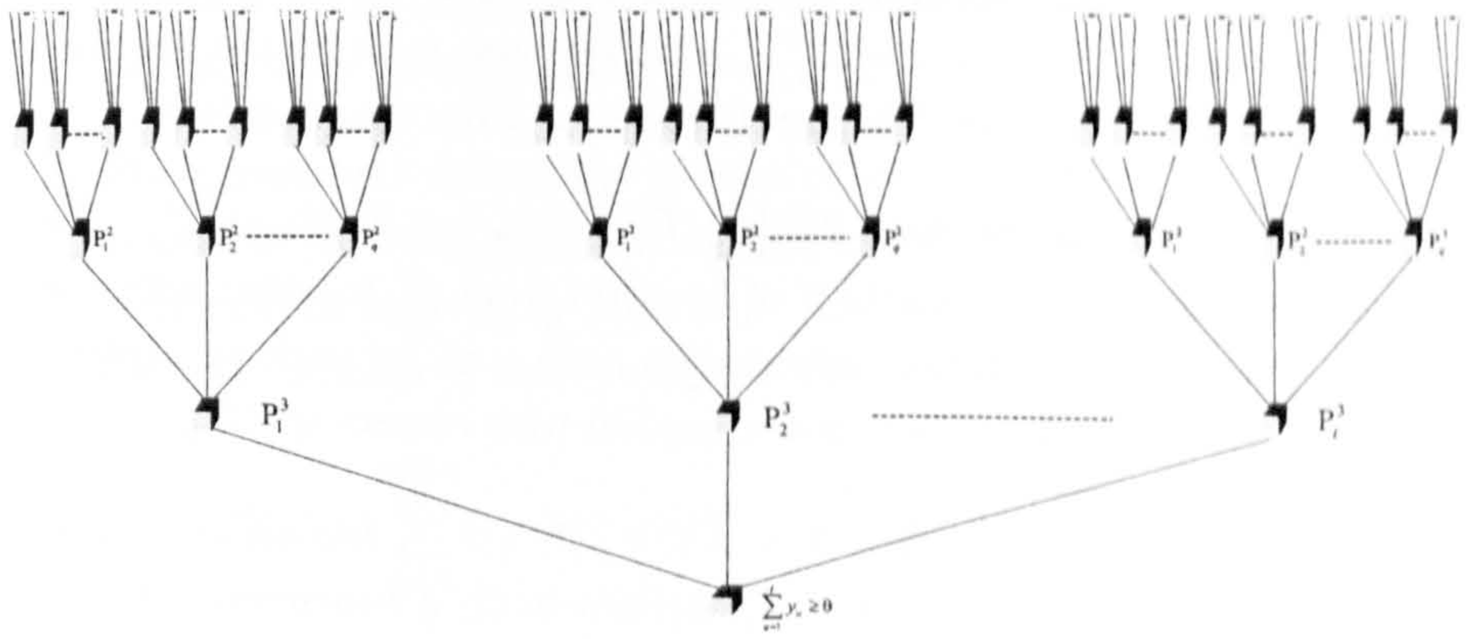


Figure 6.3: Regular Structure of Depth 4 Circuits C_4.

6.4 The Computation of Classification Circuits

The sample set $\mathcal{S}$ is split into two disjoint subsets $\mathcal{S}_{\mathrm{L}}$ and $\mathcal{S}_{\mathrm{T}}$, where $\mathcal{S}_{\mathrm{L}}$ with $| \mathcal{S}_{\mathrm{L}} | \approx \frac{2}{3} \cdot | \mathcal{S} |$ is used to calculate the approximating threshold circuit (as suggested in many papers on Machine Learning; cf. [297], p. 677, and the literature therein). The rest of the samples $\mathcal{S}_{\mathrm{T}}$ is used for testing purposes. We assume again that both sets are balanced in the number of positive and negative examples.

In our approach, we try to provide for each perceptron in $\mathcal{C}_d$ its individual training set of approximately the same size p. Therefore, the set $\mathcal{S}_{\mathrm{L}}$ is further

partitioned into $d/2$ subsets $\mathcal{S}_{\mathrm{L}}^{j}$, $1 \leq j \leq d/2$, of approximately equal size $2 \cdot |\mathcal{S}_{\mathrm{L}}| / d$.

From each $\mathcal{S}_{\mathrm{L}}^{j}$ we calculate $N(d, 2 \cdot j - 1)$ *primary training sets* $T_{[j,i]}^{\mathrm{pr}}$, $1 \leq i \leq N(d, 2 \cdot j - 1)$, by randomly selecting p elements from $\mathcal{S}_{\mathrm{L}}^{j}$. The parameter p has to be chosen either by preliminary experiments as suggested in Chapter 5, or in such a way that

$$p \cdot N(d, 2 \cdot j - 1) \geq \alpha \cdot |\mathcal{S}_{\mathrm{L}}^{j}|, \tag{6.2}$$

where $\alpha \geq 1$ is the multiplicity of samples in primary training sets: On average, each sample $\vec{a}$ appears in α different primary training sets. This ensures that a particular $\vec{a}$ is used in different combinations of samples.

We now describe how the actual training sets $T_{[j,i]}^{\mathrm{snd}}$, called *secondary training sets*, are determined from the $T_{[j,i]}^{\mathrm{pr}}$. The $T_{[j,i]}^{\mathrm{snd}}$ are used by the learning procedure of our classifier (see Chapter 4) as sets $\mathcal{S}$ from equation (4.2).

1. $j = 1$: In this case we have $T_{[1,i]}^{\mathrm{snd}} = T_{[1,i]}^{\mathrm{pr}}$, $1 \leq i \leq t \cdot (t \cdot q)^{\frac{d}{2}-1}$. The training samples are elements $\vec{a}$ of length n from $\mathcal{S}_{\mathrm{L}}^{j}$ together with the corresponding classification information $\eta(\vec{a})$, and for each perceptron P_i^1 the procedure from Section 4 is applied with respect to $T_{[1,i]}^{\mathrm{pr}}$.

2. $j > 1$: Each perceptron P_i^j, $1 \leq i \leq t \cdot (t \cdot q)^{\frac{d}{2}-j}$, has an associated primary training set $T_{[j,i]}^{\mathrm{pr}}$, and P_i^j defines in C_d q subcircuits of type $C'_{2\cdot(j-1)}$ with a counting gate as "output". The primary training set $T_{[j,i]}^{\mathrm{pr}}$ is applied to the input nodes of the q subcircuits of type $C'_{2\cdot(j-1)}$ and therefore produces p tuples of the type $(m_1 m_2 ... m_q)$ with an associated classification $\eta(m_1 m_2 ... m_q)$, which is known from $T_{[j,i]}^{\mathrm{pr}}$. The perceptron P_i^j is trained on $T_{[j,i]}^{\mathrm{snd}} := \{[(m_1 m_2 ... m_q), \eta]\}$ by the learning procedure of our classifier (see Chapter 4).

Since the q subcircuits of type $C'_{2\cdot(j-1)}$ are calculated from different training sets at each preceding level, we can expect $m_u \neq m_v$ in $\vec{m} = (m_1 m_2 ... m_q)$, although the components of $\vec{m}$ are calculated from the same sample set.

6.4.1 Depth Related Parameters

Increasing the depth increases the number of investigated parameters. Apart from the set of parameters investigated in Chapter 5 (x_0, K, Γ), which now applies to two different depths, i.e. $\Gamma^{d=1}$ and $\Gamma^{d=3}$, additional parameters are now: p, q, t, $t/2 \leq \vartheta \leq t$, and $d \geq 2$, where d is an even number.

1. The values of q and ϑ are either depending on other parameters or can

be chosen relatively independently of the particular sample set, e.g. $q = 3, ..., 7$, $\vartheta = \lceil t/2 \rceil + 1$.

2. The circuit depth is important for the classification rate, see Section 6.5, but due to the exponentional increase of gates, $d \leq 6$, seems to be a natural limit.

3. The parameter p defines the sample size complexity and should be as large as possible. The parameters q and t impose a condition on p with respect to the maximum number of secondary training samples $[(m_1 m_2 ... m_q), \eta]$: For m_i we have $-t \leq m_i \leq t, 1 \leq i \leq q$ and therefore we need to ensure $p \leq (2 \cdot t + 1)^q$, which seems to be valid a priori for the usual range of q and t. In our approach, p is determined by experiments in Chapter 5 for one set of experiments, and by some preliminary experiments for the second set of experiments as follows: on a depth-two circuit with $t = 50$ we evaluate the average training error on randomly chosen $\tilde{p} := |\mathcal{S}_L|/x$ samples for $x \geq 2$. We take a small x_0 such that the training error stabilizes for $x > x_0$.

4. The parameter t determines the size of the circuit for fixed d and q. The number of counting gates (voting gate as output gate) at depth $d'' = 2 \cdot j$ in circuits C_d is given by $N(d, 2 \cdot j - 1)/t = (q \cdot t)^{d/2-j}$; see (6.1). Thus, the total number of gates in C_d, i.e. the size $S(C_d)$, is given by

$$S(C_d) = \sum_{h=0}^{d/2-1} (t+1) \cdot (q \cdot t)^h \quad = \quad (t+1) \cdot \frac{(q \cdot t)^{\frac{d}{2}} - 1}{q \cdot t - 1}. \qquad (6.3)$$

The problem we are specifically interested in is whether circuits C_d^t with a larger $d > \tilde{d}$ but smaller $t < \tilde{t}$ achieve approximately the same classification rate as $C_{\tilde{d}}^{\tilde{t}}$ for $S(C_d^t) << S(C_{\tilde{d}}^{\tilde{t}})$. Details of experimental results are discussed in Section 6.5.2.

6.5 Computational Experiments

The comparison of $C_2^{cp}[e = R]$ and $C_4^{cp}[e = R]$ provides some empirical evidence that the best classification rate is obtained on approximately the same circuit size, i.e. the same number of linear threshold functions. Moreover, counting the gates for the asymptotically optimal design of linear threshold circuits for the most complex n-ary Boolean functions [250, 252] results in the same range of circuit size as in $C_2^{cp}[e \approx R_{\min}]$ and $C_4^{cp}[e \approx R_{\min}]$ for all our problems cp considered in our study (the actual input size n_L^{cp} is taken from the number

of bits necessary to enumerate all training samples of *cp*). Based on this observation, we propose a formula to estimate the number of perceptrons that have to be trained in order to achieve a high classification accuracy. Of course, for problems that are linearly separable, like the mushroom dataset from the UCI Machine Learning repository, the proposed formula overestimates the number of threshold gates significantly.

6.5.1 Estimations of Circuit Size

Classification problems *cp* can be encoded as Boolean functions f_{cp} on n input variables where the sample sets usually provide only a tiny fraction of input-output pairs of f_{cp}. For the splice-junction data we have "DNA windows" of length 60 with the usual DNA information from $\{A, C, G, T\}$. Therefore, in strictly binary notation we have $n = 120$. We use the term "strictly binary", as we have used and described a binary representation in Chapter 5 for the splice-junction data that is based on 3 bits for each attribute. For the WBCD we have 9 attributes, each taking a value from 1 to 10, i.e. the binary encoding leads us to $n = 36$. The Pima Indians dataset has 8 attributes with integer as well as real values. We can binary encode a range of an attribute by dividing it into equal subranges, which qualitatively define "high", "medium", "low" and any "intermediate" values. Thus, even if we partition a range into four subranges we need $n = 16$ bits for binary encoding.

In all cases, the sample data provide only a tiny fraction of the theoretically possible number of function values. A priori, we can argue that not all combinations of binary inputs are feasible (or even a small fraction only has indeed a valid interpretation). We take this into account by simply encoding (enumerating) the sample data instead of using the variable number n as the input to the core of the classification circuit: We apply a methodology which is well-known from the synthesis of partially defined Boolean functions [249] in order to obtain some rough estimations of the size of circuits representing the sample data. As before, we denote by $s_L^{cp} := |\mathcal{S}_L|$ the size of the training data set, i.e. we have $s_L^{\text{splice}} = 2127$, $s_L^{\text{wbcd}} = 455$ and $s_L^{\text{pima}} = 512$. By n_{cp} we denote the number of binary variables calculated from the number of input attributes and the range of values each attribute can take, i.e. $n_{\text{splice}} = 120$, $n_{\text{wbcd}} = 36$ and $n_{\text{pima}} = 16$. The s_L^{cp} training data can be enumerated by using $n_L^{cp} := \lceil \log s_L^{cp} \rceil$ binary variables, i.e. we have $n_L^{\text{splice}} = 12$, $n_L^{\text{wbcd}} = 9$ and $n_L^{\text{pima}} = 9$.

For a given problem *cp*, we introduce the classification circuit C_{cp}: The circuit is built from threshold functions of unbounded fan-in (as basic gates; cf. [250, 251]) and has minimum size with respect to the gate number $S(C_{cp})$. The number of input nodes is n_{cp}. We now try to approximate C_{cp} by a

composition of two circuits

$$\widehat{C_{cp}} = C[n_{cp} \rightarrow n_L^{cp}] + C[n_L^{cp}], \tag{6.4}$$

where $C[n_{cp} \rightarrow n_L^{cp}]$ is a multi-output circuit that calculates the encoding of elements from S_L. The encoding then becomes the binary input to $C[n_L^{cp}]$. Based on the results about local encoding [249] we assume

$$S(C[n_{cp} \rightarrow n_L^{cp}]) < S(C[n_L^{cp}]) \text{ and therefore } S(\widehat{C_{cp}}) < 2 \cdot S(C[n_L^{cp}]). \tag{6.5}$$

Actually, $S(C[n_{cp} \rightarrow n_L^{cp}])$ depends strongly on the distribution of sample data within the whole domain of feasible samples of *cp* and equation (6.5) is valid for sufficiently large n^{cp} and complex *cp* only. Nevertheless, we will focus on $S(C[n_L^{cp}])$ alone which seems to be justified by our experimental observation that indeed the chosen problems are associated with complex functions f_{cp}.

To estimate $S(C[n_L^{cp}])$, we use the asymptotically optimal design of Boolean functions by linear threshold functions as presented in [250]:

$$S(f_n) \leq 2 \cdot \sqrt{\frac{2^n}{n}} \cdot \left(1 + \underline{o}(\sqrt{2^n/n})\right), \tag{6.6}$$

where $f_n = f(x_1, ..., x_n)$ is an arbitrary Boolean function. Moreover, almost all Boolean functions f_n asymptotically require $2 \cdot \sqrt{2^n/n}$ linear threshold gates for their representation [250] (i.e. almost all functions are as complex as the most complex functions).

6.5.2 Classification Rate vs. Depth

We computed classification results for C_2 and C_4 circuits for a number of different pairs (t, q). According to equation (6.1), the number of perceptrons that have to be trained in C_4 is given by $N_{\text{per}} = t \cdot ((t \cdot q) + 1)$. The value N_{per} was taken as the number of perceptrons in C_2. We also have $|S_T| \approx |S|/3$, $|S_L^1| = |S_L^2| = |S_L|/2$ (for $d = 4$).

We perform two set of experiments. The first set called *set-1* is obtained from parameters from the previous Chapter for each dataset. We recall for set-1: $x_0^{IE} = 5$, $x_0^{EI} = 3$, $x_0^{Neither} = 6$, $x_0^{wbcd} = 5$, $x_0^{pima} = 25$, $K = 25,000$, for all datasets and $\Gamma_{IE} = 3p/4$, $\Gamma_{EI} = 3p/4$, $\Gamma_{Neither} = p/2$, $\Gamma_{wbcd} = 3p/4$, and $\Gamma_{pima} = 2p/3$.

Set-2 is based on preliminary experiments on depth-two circuits for $N_{\text{per}} = 50$. The experiments lead us to the following settings: $p_{\text{splice}} := |S_L|/4$ for all three classes, $p_{\text{wbcd}} := |S_L|/6$, and $p_{\text{pima}} := |S_L|/18$. The rest of parameters on set-2 has been selected to be the same for all datasets: $K = 20,000$ and

	Circuits of depth two.					Circuits of depth four.			
N_{per}	Splice-Junction			WBCD	(t,q)	Splice-Junction			WBCD
	IE	EI	Neither			IE	EI	Neither	
30	4.1%	2.8%	6.4%	1.1%	(3,3)	4.1%	3.3 %	7.2%	1.2%
48	4.0%	2.8%	6.3%	0.7%	(3,5)	4.2%	3.3 %	7.4%	1.4%
66	4.0%	2.8%	6.2%	0.6%	(3,7)	4.2%	3.5 %	7.0%	1.2%
80	4.0%	2.8%	6.1%	0.5%	(5,3)	4.1%	3.0 %	7.0%	1.1%
84	4.0%	2.8%	6.1%	0.5%	(3,9)	4.4%	3.1 %	7.1%	1.9%
130	3.9%	2.7%	6.0%	0.6%	(5,5)	4.1%	2.9 %	7.0%	1.1%
154	3.9%	2.7%	6.0%	0.5%	(7,3)	3.9%	3.1 %	6.8%	1.1%
180	3.8%	2.6%	5.9%	0.5%	(5,7)	4.1%	3.2 %	6.8%	1.3%
252	3.8%	2.6%	5.9%	0.4%	(7,5)	4.0%	2.9 %	6.9%	0.9%
252	3.8%	2.6%	5.9%	0.4%	(9,3)	4.0%	2.9 %	6.6%	0.8%

Table 6.1: Error Rates on Test Sets $\mathcal{S}_T^i$, $i = 1, ..., 4$ for Set-1 Experiments.

	Circuits of depth two.					Circuits of depth four.			
N_{per}	depth-two			WBCD	(t,q)	Splice-Junction			WBCD
	IE	EI	Neither			IE	EI	Neither	
30	4.1%	3.4%	6.3%	1.1%	(3,3)	4.3%	3.6 %	7.0%	1.7%
48	4.0%	3.3%	6.0%	0.9%	(3,5)	4.2%	3.5 %	6.7%	2.0%
66	4.0%	3.3%	6.0%	0.9%	(3,7)	4.1%	3.8 %	7.2%	1.3%
80	4.0%	3.3%	6.0%	1.2%	(5,3)	4.1%	3.4 %	7.0%	1.0%
84	4.0%	3.3%	6.0%	1.2%	(3,9)	4.5%	3.8 %	7.3%	1.4%
130	3.9%	3.3%	6.0%	1.1%	(5,5)	4.0%	3.3 %	6.6%	0.9%
154	3.9%	3.3%	6.0%	1.1%	(7,3)	3.9%	3.4 %	6.3%	0.7%
180	3.8%	3.2%	5.9%	1.1%	(5,7)	4.0%	3.3 %	6.5%	1.1%
252	3.8%	3.1%	5.9%	1.0%	(7,5)	3.7%	3.1 %	6.5%	0.4%
252	3.8%	3.1%	5.9%	1.0%	(9,3)	3.7%	3.3 %	6.2%	0.8%

Table 6.2: Error Rates on Test Sets $\mathcal{S}_T^i$, $i = 1, ..., 4$ for Set-2 Experiments.

$\Gamma = p/3$.

In both sets of experiments and for all datasets we used a training sample size at depth-three of $|S_L^2|/2$, to secure that the multiplicity for training at depth-three is $\alpha > 1$. The parameter Γ for training at depth-three has the same value as at depth one for each dataset and for each set of experiments, i.e. for training depth-three gates in $cp = WBCD$ we used $\Gamma = 3p/4$ in set-1 and $\Gamma = p/3$ in set-2.

As stated in Chapter 5, the optimization of parameter setting involves an extensive number of experiments. Thus, we tried for experiments on circuit complexity to have, in general, two different sets of experimental parameters, both being different but at the same time close to good parameter settings obtained in the previous chapter. Table 6.1 till Table 6.3 present the experimental results. We did not observe significant CPU time differences for the average CPU time for both depth-two and depth-four circuits.

Since for both types of circuits and the best values of error rates on test samples the maximum error rate on training data is equal or close to zero for all classes defined by SJGSD and below 1% for WBCD, the total error

Circuits of depth two.			Circuits of depth four.		
N_{per}	Set-1	Set-2	(t,q)	Set-1	Set-2
30	25.8%	26.3%	(3,3)	27.4%	28.7%
48	25.6%	25.8%	(3,5)	27.6%	28.1%
66	25.2%	25.8%	(3,7)	27.8%	26.6%
80	25.1%	25.4%	(5,3)	25.6%	26.7%
84	25.1%	25.3%	(3,9)	28.6%	26.7%
130	25.1%	25.1%	(5,5)	25.9%	26.5%
154	25.4%	25.1%	(7,3)	24.8%	25.6%
180	25.4%	25.4%	(5,7)	26.2%	27.8%
252	25.3%	25.1%	(7,5)	25.2%	25.9%
252	25.3%	25.1%	(9,3)	25.2%	25.1%

Table 6.3: Error Rates on Pima Indians Test Sets $\mathcal{S}_T^i$.

rate does not change significantly if test samples as well as training data are taken into account on circuits calculated from training data only. Moreover, in the context of the present Chapter, we focus on the circuit complexity for best classifications within some small deviation of the error rate, as explained below.

For each experimental set, let R_d^{cp} denote the maximum of the three best (not necessarily different) classification rates for problem cp and circuits C_d. We set for $d = 2, 4$:

$$\mathcal{S}_{\min}^{cp}(d) := \min_{R=R_d^{cp}} \mathcal{S}(C_d^{cp}[e = R]), \tag{6.7}$$

where $e = R$ means an error rate of R by the given circuit. We now allow a margin of deviation from R_d^{cp} by Δ. As an estimation of the circuit size that ensures a high classification rate we take

$$\mathcal{S}_d^{cp} := \max\{\mathcal{S}(C_d^{cp}[e = R]) : (R_d^{cp} < R \leq R_d^{cp} + \Delta) \& (\mathcal{S}(C_d^{cp}[e = R]) \leq \mathcal{S}_{\min}^{cp}(d))\}. \tag{6.8}$$

If the max-operation is over an empty set, we take $\mathcal{S}_d^{cp} := \mathcal{S}_{\min}^{cp}(d)$. Taking the max-operation in equation (6.8) gives some confidence that the error rates have already stabilized. The calculation of the corresponding values for the two types of circuits and the five classification problems is summarised in Table 6.4.

From Table 6.1 and Table 6.2 we observe that the classification rate for depth-two circuits provides the same best values for IE, "Neither" and Pima Indians for settings from the two different parameter sets, while EI and WBCD seem to benefit from Chapter 5 parameter settings.

Comparing depth-two and depth-four results presented in Table 6.1 and Table 6.2, we conclude that experiments with parameter set-1, which, in general, are better fine-tuned than parameter set-2, leads to better classification

Circuit size estimates	Splice-Junction IE	EI	Neither	WBCD	Pima
Set-1 Experiments					
$\mathcal{S}_2^{cp}$	154	154	154	66	66
$\mathcal{S}_4^{cp}$	130	84	130	66	130
Set-2 Experiments					
$\mathcal{S}_2^{cp}$	154	154	154	180	84
$\mathcal{S}_4^{cp}$	130	84	130	130	130

Table 6.4: Circuit Size Estimates for $\Delta := 0.2\%$.

Circuit size estimates	Splice-Junction IE	EI	Neither	WBCD	Pima
$\mathcal{S}_{cp}$	154	154	154	180	130
$\mathcal{S}_{\text{pred}}^{cp}$	147	147	147	60	60

Table 6.5: Circuit Size Estimates Compared to $\mathcal{S}_{\text{pred}}^{cp}$.

rates in depth-two circuits. For set-2 parameters we see that for $cp =$ WBCD the differences in the classification rate are very small, with slightly better results for $d = 4$; for $cp =$ "Neither" we have slightly better results for $d = 2$, whereas for the rest of datasets and set-2 parameters the classification rate in both depth-two and depth-four circuits is approximately the same. From the above it is obvious that the reason for set-1 having better classification rates for $d = 2$ much depends on the fact of fine-tuning the parameters for depth-one (see Chapter 5), without at the same time having fine-tuned parameters at depth-three.

We obtain approximately the same circuit size estimations according to equation (6.8) for both $d = 2$ and $d = 4$ (except for the EI-class), and for both sets of experiments (see Table 6.4).

We set $\mathcal{S}_{cp} := \max\{\mathcal{S}_2^{cp}, \mathcal{S}_4^{cp}\}$ (see Table 6.5) and compare the values to $\mathcal{S}_{\text{pred}}^{cp} := x \cdot 2 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2}$; cf. (6.6), where $x = 4$ is chosen on the following grounds: The RHS of equation (6.5) doubles the complexity $\mathcal{S}(\widehat{\mathcal{C}_{cp}})$, and we assume that the third factor on the RHS of equation (6.6), which summarizes the complexity of auxiliary sub-circuits, at most doubles the product of the first two factors for relatively small values of $(2^{n_L^{cp}}/n_L^{cp})^{1/2}$. We recall that $n_L^{\text{splice}} = 12$, $n_L^{\text{wbcd}} = 9$ and $n_L^{\text{wbcd}} = 9$,; see Section 6.5.1. Thus we conjecture

$$\mathcal{S}_{\text{pred}}^{cp} \lessapprox 8 \cdot \sqrt{2^{n_L^{cp}}/n_L^{cp}}. \tag{6.9}$$

We note that the simple rules from equations (6.7) and (6.8) together with (6.9) generate the same $\mathcal{S}_{cp}$ for $cp \in \{\text{IE, EI, Neither}\}$. Furthermore, if cp = WBCD for set-1 experiments, we obtain for both depths $\mathcal{S}_{cp} = 66$, whereas for set-2 experiments and $N_{\text{per}} = 48$ or $N_{\text{per}} = 66$, the error rate in Table 6.2 is only 0.9%, i.e. $\mathcal{S}_{\text{pred}}^{wbcd} = 60$ from Table 6.4 seems to be a good estimation from both sets of experiments. We have a different picture for Pima Indians and $\mathcal{S}_4^{pima}$, however, $\mathcal{S}_2^{pima}$ approximates $\mathcal{S}_{\text{pred}}^{pima}$.

In the context of Machine Learning, the RHS of equation (6.6) provides an estimation of the size of elements of the hypotheses space (circuits of threshold functions) that represent particular objects (concepts). The method to prove a lower bound for equation (6.6) was utilised in [251, 252] (cf. also [29]) to obtain a lower bound for a sufficient number of samples such that the error rate on test samples is below ε with probability at least $(1 - \delta)$. The lower bound (sufficient number) is expressed in terms of the VC-dimensions of neural nets, which basically equals the number of neurons (threshold gates); see also [253]. The lower bound is of the type

$$\max\{4 \cdot n \cdot (n+k)^2/\varepsilon \cdot \log(13/\varepsilon)\,;\, 4/\varepsilon \cdot \log(2/\delta)\}, \tag{6.10}$$

where k is the number of gates in neural nets (which represent the hypotheses). We note that $k = \mathcal{S}_{\text{pred}}^{cp}$ (or even $k = 2 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2}$) and ε in the range of values from Table 6.1, and Table 6.2 would result in a number of examples much larger than the number of samples available from our datasets, which is relatively independent of δ. Thus, the bound (6.10) for a sufficient number of samples seems to be too large, at least for the cp from Chapter 5.

Our computational study on the datasets from the UCI Machine Learning Repository provides empirical evidence that $8 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2})$ is an appropriate upper bound for the size of threshold circuits in order to achieve a high generalisation capability of circuits. The value of n_L^{cp} is taken as the number of bits necessary to encode all training data by binary strings, and is independent from the number of features of a problem. The upper bound has been analysed for relatively small n_L^{cp} only. For increasing n_L^{cp} one can expect that the constant becomes significantly smaller (but still larger than 2). For the problems we investigated we observed that the number of gates in circuits with low error rates is approximately the same for both depth-two and depth-four circuits, in the absence of fine-tuning of parameters (set-2 of experimenents). The comparison of the circuit size to the asymptotic bound for the most complicated Boolean functions suggests that the chosen problems are indeed of a complex nature.

	Circuits of depth two.					Circuits of depth four.			
N_{per}	Splice-Junction			WBCD	(t,q)	Splice-Junction			WBCD
	IE	EI	Neither			IE	EI	Neither	
30	4.5%	3.4%	7.8%	2.2%	(3,3)	4.1%	3.3 %	7.2%	1.2%
48	4.4%	3.3%	7.5%	1.8%	(3,5)	4.2%	3.3 %	7.4%	1.4%
66	4.2%	3.3%	7.2%	1.6%	(3,7)	4.2%	3.5 %	7.0%	1.2%
80	4.3%	3.3%	7.6%	1.8%	(5,3)	4.1%	3.0 %	7.0%	1.1%
84	4.2%	3.3%	7.1%	1.5%	(3,9)	4.4%	3.1 %	7.1%	1.9%
130	4.1%	3.2%	7.2%	1.6%	(5,5)	4.1%	2.9 %	7.0%	1.1%
154	4.3%	3.2%	7.4%	1.6%	(7,3)	3.9%	3.1 %	6.8%	1.1%
180	4.2%	3.1%	7.0%	1.5%	(5,7)	4.1%	3.2 %	6.8%	1.3%
252	4.2%	3.2%	7.1%	1.3%	(7,5)	4.0%	2.9 %	6.9%	0.9%
252	4.2%	3.2%	7.2%	1.5%	(9,3)	4.0%	2.9 %	6.6%	0.8%

Table 6.6: Error Rates if Monitoring Depth-two in Depth-four Circuits on Test Sets $\mathcal{S}_T^i$, $i = 1, ..., 4$, for Set-1 Experiments.

6.6 Monitoring Depth-two Classification Error in Depth-four Circuits

The experiments from Section 6.5.2 refer to a comparison of two different classification circuits, the $\mathcal{C}_2$ and $\mathcal{C}_4$ circuits. It is interesting to compare one classification circuit of $\mathcal{C}_4$ with respect to the classification error at depth-four against the possible classification errors at the preceding level of the same circuit. For monitoring the possible classification error at depth-two we use a voting function at depth-two. Thus, output gates from depth-one are connected to two types of gates: i) connected to counting functions at depth-two so that the computation of depth-four threshold function is performed as described in this Chapter, and ii) to a voting function at depth-two in order to monitor the possible classification error at depth-two during the test phase. Table 6.6 presents this comparison for the four datasets and set-1 experiments.

We observe that the error rate improves from depth-two to depth-four in all datasets, except for some values related to pairs (3,7) and (3,9), showing that the idea of generating new samples for training next depth generated circuits is justified. Therefore, adjusting the depth-three weights in order to fit the sample sets produced by outputs from the previous level improves the classificaton rate of the previous level by learning and giving credits to significant contributors, without any overfitting problems. The comparison of depth-two circuits trained by $2S_L/3$ of available data S against depth-four circuits, where each even depth is trained by equally splitted samples S_L^j leads to almost the same classification rates as in Table 6.2. The finding that there exist internal improvements of classification rates from one level to the next one

encourages us to further investigate depth-four circuits. It seems that the size of $|S_L^j|$ is very important for $j = 1$. Moreover, no parameter optimization has yet been applied to depth-three. These observations encourage us to include into our future work more research on depth-four circuits.

6.7 Tackling Classification Problems by the LSA Machine

Based on our findings from Chapter 4, Chapter 5, and Chapter 6 we suggest the following *a priori* parameter settings for tackling classification problems *cp* by depth-two threshold circuits.

1. For the training sample size $S_L = 2|S|/3$ calculate n_L^{cp};
2. Calculate the size of circuits N by applying $N = 8 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2}$; cf. Equation (6.9);
3. Run m experiments for maximum sample size with $x_0 = 2$, for N, for maximum $\Gamma = |S_L^j|$, and an adequate number of transitions K (K=25,000 seems to be adequate) for determining Γ and K values of the simulated annealing process, for our next experiments; monitoring the maximum escape depth D_{esc} allows us to set $\Gamma = D_{esc}$ for the rest of experiments; if $K_{max} << 25000$, we set for the rest of our experiments $K = max\{K_{max}^i\}$, where $i = 1, ..., m$;
4. To estimate the size of sample sets, we proceed as follows: For settings N, K, $\Gamma = D_{esc}$, we start with values of $x_0 = 2, 3, ...$ and perform trial experiments, monitoring K_{max} and U_0; since in our previous experiments (except for Pima Indians) we observed that the classification error decreases almost monotonically, before an increase occurs, we suggest to monitor the monotonicity of the classification rate until an increase occurs for a number of subsequent steps;
5. If one of the following conditions state true for a number of n experiments on x_0, then the current value of x_0 is selected:
   a) An increase in classification error occurred for $x > x_0$ or
   b) $K_{max}^i < K - \delta_{x0} \cdot K$, where $\delta_{x0} \geq 0.1$, and for $x_0^{i-1} < x_0^i$ we observed $K_{max}^{i-1} \approx K$ or
   c) $U_0 = 100\%$

If the dataset is linearly seperable, then these steps will probably find $x_0 = 2$, $e_T = 0$, however, the method unavoidably overestimates the circuit

size. If we reach a large value of x_0, then probably one cannot find sizes of random sample sets that are linearly seperable. The above steps are in compliance with the NFLT, as they take into account a number of problem-specific parameters.

The benefits of the above-mentioned steps are the fast convergence to good solutions of classification problems. Either condition b) or condition c) in step 5, secure good approximations to near optimum solutions with fast convergence, as in both cases the simulated annealing process stops for most of the threshold units the training process before reaching the termination value K.

Recently, Lane and Gobet [225] emphasized the need for developing rules for reproducibility and comprehensibility of computational models. They propose that a computational model should be released as three components: a) A well-documented implementation; b) A set of tests illustrating each of the key processes within the model; and c) A set of canonical results, for reproducing the model's predictions in important experiments. Covering aspects of reproducibility in our work, we illustrated key processes within our classifier by using sets of tests for each parameter (as it has been presented in Chapter 5), while our reported results have the detailed parameter settings that allow to reproduce all the experiments. Moreover, with respect to reproducibility, we try to evaluate the above steps by using additional datasets. Thus, taking into account the scope of the present work, the description of our methodology in Chapter 4 till Chapter 6 meets the criteria b) and c), and the algorithmic aspects of a) are also described in detail.

6.8 Additional Datasets

All our datasets are from the UCI Repository and will be used for evaluating our rules for tackling new datasets described in previous section.

6.8.1 Hayes-Roth Datasets (Hayes)

Barbara and Frederick Hayes-Roth [162] created this dataset for recognition and classification of exemplars. The datasets has three classes (named hayes-1, hayes-2 and hayes-3 in the rest of the text), which consist of five numerical attributes: name (deleted for our experiments), hobby (a number between 1 and 3), age (a number between 1 and 4), education level (a number between 1 and 4), marital status (a number between 1 and 4). We binary encoded the four attributes as:

3 bit for hobby → 1=001, 2=010, 3=100
4 bit for age → 1=0001, 2=0010, 3=0100, 4=1000
4 bit for education level → 1=0001, 2=0010, 3=0100, 4=1000
4 bit for marital status → 1=0001, 2=0010, 3=0100, 4=1000

The dataset in UCI Repository consists of a training file of 132 samples and a test file of 28 samples. We merged the two file and used 2/3 of data for training and 1/3 for testing, therefore we reduced the number of training samples and increased the number of test samples, preserving however in our testset the 28 original test samples.

6.8.2 Iris Plant Datasets (Iris)

Fisher's [117] iris plant dataset is the oldest and perhaps the most frequently used dataset in Machine Learning with innumerable publication of results in pattern classification literature. The set consists of 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two; the latter are not linearly separable from each other. There are four attributes containing measurements in cm of sepal length, sepal width, petal length, and petal width, and the task is to identify the three classes: Iris Setosa (linearly seperable), Iris Versicolour, and Iris Virginica. We denote the three classification problems, as iris-1, iris-2 and iris-3. We binary represent the numerical values in the following way:

We divide the space from the minimum to the maximum value for each attribute in 10 ranges, creating for each of the four attributes a binary string of 10 bits length, where only one bit has value 1 (for the corresponding matching of the real value) and the rests values of 0. The string 1000000000, for instance, is close to the minimum value of the measurement space for attribute x_n and is in the range $\{min, min+(max-min)/10\}$, whereas 0000000001 is in the range $\{min+9(max-min)/10, max\}$. Therefore instead of four input variables we create a binary dataset of 40 attributes.

6.8.3 Mushrooms Dataset (Mushrooms)

The mushroom dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, and definitely poisonous, therefore is a binary class dataset. The dataset has 22 nominal values, and despite the large number of samples 8,124, and despite that as stated in UCI Repository there is no simple rule for determining the edibility of a mushroom,

the data are linearly separable. We also by using this dataset found that our classifier achieves zero classification error on $|S|/3$ test samples.

6.8.4 The Monk's Problems (Monk)

The Monk's problems were the basis of a first international comparison of learning algorithms in [359]. The Monk-1 and Monk-2 datasets, used in this research, have a target concept to be identified. Monk-1 consists of 556 samples, monk-2 consists of 601 samples, and both datasets, if we omit the "id" attribute, which is unique for each sample, have 6 attributes. Two attributes (called a3, a6) receive one of the two values from $\{1,2\}$, three attributes (called a1, a2, a4) receive values from $\{1,2,3\}$, and one attribute (a5) receives values from $\{1,2,3,4\}$. We transform the values into binary strings by using the same transformation as in Hayes-Roth dataset, thus our datasets consists of 17 binary attributes. Problem Monk-1 targets at learning concept: (a1 = a2) or (a5 = 1), and Monk-2 at learning concept: EXACTLY TWO of $\{ a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1, a6 = 1 \}$. While many learning algorithms reach zero classification error, these problems show the limitations of our method.

6.8.5 US Congressional Voting Records Database (Votes)

The US Congressional Voting Records (Votes) data set includes votes for each of the U.S. House of Representatives Congressmen on 16 key issues (attributes) identified by the Congressional Quarterly Almanaco (CGA) in 1984. The dataset consists of 435 samples of two classes (267 democrats, 168 republicans). The 16 key isues congressman's votes for which Boolean data (as yes, no votes) exist in voting records are: { handicapped infants, water project cost sharing, adoption of the budget resolution, physician fee freeze, El Salvador aid, religious groups in schools, anti-satellite test ban, aid to nicaraguan contras, mx-missile, immigration, synfuels corporation cutback, education spending, superfund right to sue, crime, duty-free exports, and export administration act to South Africa }.

6.8.6 Waveform Datasets (Wave)

Waveform is a three class dataset (waveform 1, waveform 2, waveform 3) where each class is a "wave" generated from a combination of 2 of 3 "base" waves. The dataset contains 5,000 samples, with 21 attributes with continuous values from 0 to 6. We used in our research these datasets with their original numerical values.

Domain	$\|S\|$	$\|S_L\|$	n_L^{cp}	N	G	K
Hayes-Roth 1	160	106	7	35	60%	8,000
Hayes-Roth 2	160	106	7	35	60%	11,000
Hayes-Roth 3	160	106	7	35	LS	LS
Iris 1	450	300	9	60	LS	LS
Iris 2	450	300	9	60	66%	25,000
Iris 3	450	300	9	60	66%	25,000
Monks 1	556	370	9	60	50%	25,000
Monks 2	601	401	9	60	50%	25,000
Mushrooms	8,124	5,416	13	200	LS	LS
Votes	435	290	9	60	66%	25,000
Waveform 1	5,000	3,333	12	147	66%	25,000
Waveform 2	5,000	3,333	12	147	66%	25,000
Waveform 3	5,000	3,333	12	147	66%	25,000

Table 6.7: Datasets and Preliminary Results for Settings N, Γ, and K as in 1,..,3.

6.9 Evaluating Parameter Settings on New Datasets

We will apply the rules of Section 6.7 to the above described collection of additional datasets.

Table 6.7 shows the estimated circuit size and the results from preliminary experiments for $x_0 = 2$, $\Gamma = |S_L^j|$, and K=25,000 with respect to depth-two circuits and several additional datasets. Here, we proceeded as suggested by our paradigms 1,..,3. The aim is to estimate G for $D_{esc} \approx \Gamma = G \cdot |S_L^j|$ and to estimate the predefined K determined by the comparison of K_{max} <<25,000. For the latter we found that only on the "Hayes-Roth" datasets, K_{max} was considerably shorter than K=25,000, and therefore, in Table 6.7, $K^{Hayes} \neq$ 25,000. 'LS' indicates a zero error classification for the dataset, and we consider the set as lineary seperable. We found zero classification error also for depth-four circuits of equal size of the corresponding depth-two circuits. Therefore such datasets will be excluded from further investigation.

Having set the three parameters, the last but important parameter that needs to be set is the sample size used for training a single threshold unit, which is determined by x_0 for the case of depth-two circuits. Table 6.8 presents results for x_0, and the classification rate e_T obtained. The values were obtained if applying each of the three criteria of step 5. We display in boldface the x_0 with respect to the criterion with the best classification rate for each dataset.

Table 6.8 reveals that monitoring the change in monotonicity of the classification rate e_T against x_0 (criterion 5a) allows us to have a high classification performance. Only the dataset "votes" gives a better classification rate by using criterion 5b, however, for this dataset a better value has been obtained for another value of x_0, whereas for no other dataset any other better value

	Criterion 5a		Criterion 5b		Criterion 5c	
Domain	x_0	e_T	x_0	e_T	x_0	e_T
Hayes-Roth 1	**2**	11.1% ±1.0%	4	13.3% ±1.5%	-	-
Hayes-Roth 2	**3**	11.5% ±2.0%	**3**	11.5% ±2.0%	-	-
Iris 2	**2**	0.7% ±0.1%	4	2.0 % ±0.3%	-	-
Iris 3	**2**	0.7% ±0.1%	3	1.0 % ±0.4%	-	-
Monks 1	**4**	22.7% ±1.3%	15	56.8 % ±4.8%	-	-
Monks 2	**4**	34.2% ±2.6%	15	52.5 % ±6.7%	-	-
Votes	**2**	3.9% ±0.4%	4	3.2 % ±0.4%	-	-
Waveform 1	**4**	14.7% ±0.1%	60	14.9% ±0.3%	-	-
Waveform 2	**4**	11.7% ±0.2%	70	13.9% ±0.3%	70	13.9% ±0.3%
Waveform 3	**5**	11.4% ±0.2%	60	12.6% ±0.3%	70	13.2% ±0.2%

Table 6.8: Classification Errors and x_0 for each Criterion of Step 5 of Paradigms.

Domain	e_T^2 for N	e_T^2 for $N/2$	e_T^2 for $2 \cdot N$	(t,q)	e_T^4 for N
Hayes-Roth 1	11.1% ±1.0%	11.9% ±2.1%	11.1 ±1.0%	(3,5)	15.6% ±1.7%
Hayes-Roth 2	11.5% ±2.0%	11.9% ±2.6%	10.4 ±1.7%	(3,5)	13.0% ±2.1%
Iris 2	0.7% ±0.1%	0.7% ±0.1%	0.7 ±0.1%	(3,7)	0.7% ±0.1%
Iris 3	0.7% ±0.1%	0.7% ±0.1%	0.7 ±0.1%	(3,7)	0.9% ±0.4%
Monks 1	22.7% ±1.3%	23.1% ±2.4%	23.1 ±1.1%	(3,7)	36.3% ±8.7%
Monks 2	34.2% ±2.6%	34.7% ±3.8%	37.3 ±4.7%	(3,7)	38.3% ±7.1%
Votes	3.2% ±0.4%	3.4% ±0.1%	3.3 ±0.1%	(3,7)	4.4% ±1.4%
Waveform 1	14.7% ±0.1%	14.7% ±0.1%	14.8 ±0.1%	(7,3)	14.4% ±0.3%
Waveform 2	11.7% ±0.2%	11.7% ±0.2%	11.8 ±0.1%	(7,3)	11.2% ±0.2%
Waveform 3	11.4% ±0.2%	11.6% ±0.2%	11.4 ±0.1%	(7,3)	11.5% ±0.1%

Table 6.9: Classification Errors for N, $N/2$, $2 \cdot N$, (Depth-two Circuits) and for N and Depth-four Circuits.

has been obtained other than those in Table 6.8. Trying to achieve zero classification error (criterion 5c), is indeed hard as only for the two "waveform" datasets this has been indeed acieved. '-' indicates datasets, where we could not achieve $U_0 = 100\%$ for training sample sizes $|S_L^j| \geq 10$, whereas training sample sizes $|S_L^j| < 10$, lead to bad generalization performance. We conclude that observing the monotonicity (criterion 5a) seems to be an appropriate criterion for determining training sample sizes.

We compare N according to Equation (6.9) to half of circuit sizes and double circuit sizes with respect to parameters from Table 6.7 and Table 6.8. Moreover, in Table 6.9 we compare our best classification rate obtained by depth-two circuits with depth-four circuits of equal size, where $x_0^4 = 2$, $\Gamma_1^{cp} = \Gamma_3^{cp}$ and $K_1^{cp} = K_3^{cp}$ for all datasets.

The comparison shows that our upper bound $8 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2})$ for estimating the circuit size does indeed imply the highest classification rates. Except for the 'Hayes-Roth 2' dataset, where a 10.4 classification rate is obtained for double circuit size, the highest classification rate is obtained for N according

to (6.9). Comparing depth-two and depth-four circuits, we see that for the largests of our datasets depth-four circuits obtain better classification rates.

We should observe that most of our classification rates **are competitive or outperform** reported classification results from the literature. More specific:

Regarding "Hayes-Roth" datasets, most classification error rates in the literature vary in the range of 20%-25%, as recently in [52, 110, 111, 112, 116, 137, 197, 203, 396], few studies achieve classification error rates results in the range of 10%-20% as recently in [10, 49, 306]. Berzal et al. [49], for example, achieved 15.6% by introducing ART, a hybrid classification model, that builds decision lists based on ideas from the association rule mining context, Ahmad and Dey [10] used feature selection and C4.5 for achieving 14.4% error rate, and finally Ricks and Ventura [306] by training a quantum neural network achieved error rate 11.03%. It seems to be rare to find in literature results of classification errors $< 10\%$. Chen et al. [75] achieved 7.1% error rate, which is the lowest to our best of knowledge. They introduced an inductive learning algorithm for using it in Chinese text-to-speech systems. We obtained in our research a 7.4 % average classification error on the three datasets, which is very close to Chen's et al. results, where we note that our testset is larger than the proposed in UCI Repository.

Regarding Iris Plant Datasets, we identified from classification error results found in the literature that most results are in the range 7%-3% as in [10, 34, 52, 99, 110, 111, 112, 116, 137, 198, 203, 224, 309, 363, 396] to name a few recent published results. Recently high classification results have been reported in [267, 284, 286, 306, 316, 368, 375]. Van Gestel et al. [368] using linear function kernel in least squares support vector machine classificrs obtain 1.4% classification error, Paul and Kumar [284] with a fuzzy neural system called SuPFuNIS achieved zero classification error, whereas the rest reported here studies have worse classification errors. In our research we obtained an average 0.5% classification error for the three classes (0.7%, 0.7% and 0% for each of the three classes), which is close to the highest reported value in [284].

Monks is the only dataset where we have worse results than those reported in the literature, where for many classifiers zero classification error is reported. However, this dataset is a conceptual dataset, where the data represent rules that have to be learned.

Regarding the U.S. Congressional Voting Records Database, many studies report a classification error of 4%-5% as recently in [111, 112, 309, 396]. Some better results close to 3% are recently reported by Ahmad and Dey [10] using feature selection and C4.5, whereas Tsoumakas et al. [363] achieve 2.54% error rate by combining heterogeneous classifiers. Eklund [110] used

multi-class decision trees with multivariate tests at internal decision nodes to get classification error 2.8%. Our experiments have resulted close to the best classification errors by obtaining 3.2% error rate.

Finally, for the Waveform datasets results in UCI Repository report 14% classification error by the optimal Bayes classifier, 22% by the nearest neighbour algorithm by using 300 samples only for training, and 28% by the CART decision tree. Recent results on the dataset report 14.41% in Banfield et al. work [34]with a decision tree classifier with a random forest approach in the selection of attributes to be used for building the tree, and 16.3% in Jing's et al. work [198] by using parameter optimization in a special type of Bayesian classifier. Our approach obtains best classification error of 14.4%, 11.2%, and 11.4% for each of the three datasets, a performance which is close to optimal Bayes performance.

Chapter 7

Conclusions

Minimising the error of missclassified samples for a network is an NP-hard problem for non-linearly separable problems and only approximations of the optimal weight vector can be found. We investigated a combination of a combinatorial optimization algorithm with the classical perceptron algorithm, called the LSA machine introduced by Albrecht and Wong [19], for exploring the applicability of search strategies on hard pattern classification prolems with respect to classification accuracy, to learning and convergence properties, and to circuit complexity. In compliance with the NFLT, we presented a detailed analysis of parameters settings, where we monitored and tried to inter-relate the best results with learning and convergence properties. Investigating the circuit complexity of our approach we introduced a new learning method for training larger depths and we suggest an upper bound for the size of the network for approximating best classification accuracy. Results on circuit complexity and parameter settings allow us to suggest guidelines and a selection of steps in order to tackle new classification problems by threshold circuits that are trained by the LSA machine method. Our experimental results from parameter settings and circuit complexity are very competitive to those existing in the literature for almost all datasets that we used.

7.1 Results on Problem-dependent Parameters

According to the No-Free-Lunch-Theorems, the performance of learning algorithms is problem-dependent, which has been demonstrated by our experiments. Therefore, since we identified that the classification accuracy of the LSA machine is affected by problem dependent parameters, fine tuning of them was unavoidable. We have defined four problem-dependent parameters, the network size, the sample size for training a unit, the length of inhomogeneous Markov chain and the constant Γ of the simulated annealing cooling schedule.

Also we defined parameters related to learning and convergence properties of our method that are monitored during execution time. The more interesting from them used after our experiments for setting parameters in new datasets are: a) the number of units U_0 that have learned with zero error their associated sample set, b) the maximum step K_{max} from all units of the m experiments for a value of the investigated parameter, where the classification error has been minimised. Unfortunately, optimization of the parameter set induces itself a combinatorial optimization problem due to the large number of experiments that are required to exhaustively search for the best values of the parameter set. Thus, we investigated the impact of each parameter on the classification accuracy for fixed values of the other parameters. Relating classifier parameters to problem-dependent attributes allows us to *a priori* estimate values for these parameters and employ methods that reduces significantly the number of required experiments to achieve high classification rates.

Our experiments showed that the the network size and the size of samples for training each unit in the network are very important problem dependent parameters for approximating high classification accuracy. Initially we focused on the problem of the sample complexity, i.e. defining the learning capacity of each unit in the network, where we investigated for the required number of samples for training each unit that approximates high classification accuracy in unseen data, which is related with the VC-dimension theory for neural networks. We investigated sizes of sample sets for training each unit that are randomly sampled from the available training set. We performed experiments for sizes of sample sets for training each unit that derive from the integer devision of $x_0 = 2, 3, 4, ...$ with the size of the available training set. We have seen that the classification accuracy of our method depends on the proper selection of x_0, which we find is more important than parameters K and Γ of the simulated annealing algorithm. Unfortunately we cannot estimate *a priori* the value of x_0 from parameters of the problem. Nevertheless, we suggest a limited number of experiments to determine x_0 in new problems by monitoring the monotonicity of the classification accuracy with x_0, and/or learning and convergence properties like U_0 and like the variance of K_{max} with the selected length of inhomogeneous Markov chain. We also suggest methods to find proper values for the length of inhomogeneous Markov chain K and for the parameter Γ of the logarithmic cooling scedule of the simulated annealing algorithm.

We should note that VC-dimension theory provides lower bounds (see equation (6.10)) for the sufficient number of samples such that the error rate on test samples is below ε with probability at least $(1 - \delta)$. Our experimental results for parameter setting shows that the bound from equation (6.10) for a

sufficient number of samples seems to be too large, at least for the classification problems from Chapter 5, or in other words we obtained, following parameter settings, error rates ε having datasets with considerably much smaller number of available training samples than the sufficient number of samples that equation (6.10) suggests.

7.2 Results on Circuit Complexity

We particularly were interested and focused on the circuit complexity issue, after analyzing the main parameter settings. To identify sequences of Boolean functions $\{f(x_1,...,x_n)\}_{n=n_0}^{\infty}$ with a "superpolynomial" (or exponential) gate number in constant depth circuits of unbounded fan-in gates is a very difficult problem, and only slow progress has been made over the past decades. Researchers relying on the possible relevance of Kolmogorov's theorem with respect to the expressive power of depth-two circuits prefer working with swallow depth circuits. The question of improving classification performance on NP-hard problems by extending the depth is an open problem. For non-NP-hard problems if high classification accuracy requires exponential size in depth-2, it is expected that the size changes to polynomial size for larger depths, i.e. we can achieve high classification rate by using smaller network size of larger depth. There exists very limited research on larger depths. The main problem is how the hidden units at larger depth can be expressed in terms of previous levels in such a way that we really have a depth increase. The difficulty of finding learning algorithms for training all layers along with the expressive power of depth-two (which, generally, are considered to be good classifiers) gave favour to shallow layers. Due to the above reasons, there is no theoretical or constructive proof that decides the size vs depth problem.

We introduced a novel and feasible approach for training units at larger depths, which is a step forward in neural network design as the topology and the recursive training method can be applied regardless of the learning method used for threshold gates.

Our research on circuits of depth-2 and depth-4 on datasets from the UCI repository was combined with classical results from threshold circuit theory, and we suggest $8 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2}$ as an appropriate upper bound for the size of threshold circuits in order to achieve a high generalisation capability of circuits on complex problems that are not linearly separable. The value of n_L^{cp} is taken as the number of bits necessary to encode all training data $|S_L^{cp}|$ by binary strings for the domain cp, i.e. $n_L^{cp} = log|S_L^{cp}|$. The upper bound has been analysed for relatively small n_L^{cp} only. For increasing n_L^{cp} one can expect that the constant becomes significantly smaller. It is important to note that

the suggested upper bound is related to the expressive capacity of Boolean functions to represent a domain *cp* with $|S_L^{cp}|$ training data. Therefore, in our approach no training method is associated with the expressive capacity of Boolean functions, and hence it can be applied to any circuit, irrespective of the training methods used.

We compared and tried to provide an answer to the hard depth vs size problem. Since we split the training samples for depth-four circuits, we have, of course, better results for depth-two for datasets with small number of samples, because depth-two uses all available training samples at once. For larger datasets and for larger circuit sizes N we obtain almost the same results for depth-four and depth-two circuits. Finally, our experimental results on depth-2 and depth-4 showed also that our classification problems are indeed NP-hard, since the size of the network that produces high classification accuracy does not decrease with increased depth.

7.3 Tackling New Datasets

The guidelines that eventually limit the number of experiments when tackling a new classification problem can be described as follows: Initially, we determine the circuit size from the number of available training samples by using the upper bound $8 \cdot (2^{n_L^{cp}}/n_L^{cp})^{1/2}$ as described above. Then, auxiliary experiments for depth-two circuits with this circuit size estimation for the largest available training samples ($x_0 = 2$), for the largest value $\Gamma_0 = |S_L^j|$, and for an adequate number of transitions (K_0=25,000), allow us to find appropriate estimations for the simulated annealing parameters Γ and K. We select Γ as an estimation of the largest escape depth from local minima. We monitor from all our preliminary examples the largest step K_{max}, where the misclassification error was lowered from all threshold units of all experiments m. If $K_{max} << K$, we select $K = K_{max}$ for our next experiments, otherwise we proceed with K=25,000. Now, the only parameter left for estimation is the number of training examples in each sample set that trains a particular threshold unit. We showed that one can considerably reduce the number of experiments by observing for $x_0 \geq i$, $i = 2, 3, ..$ the monotonicity of the classification rate, and by monitoring the value of K_{max} and the number of threshold units U_0 with zero learning error on training sets. Following a condition where we select x_0 for sample size when one of the following first occurs: either when the monotonicity turns from decreased values to an increased value, or when $K_{max} < K - \delta_{x0} \cdot K$ for $\delta_{x0} \geq 0.1$ and in previous step of x_0 we observed $K_{max} \approx K$, or when all threshold units learn the training examples with zero error ($U_0 = 100\%$), we approximate best classification rates.

We applied the above rules to additional datasets finding that the estimated circuit size gives indeed the best classification results. Also our experimental results following the above rules on the additional datasets are again very competitive or even outperform those in the literature.

7.4 Classification Accuracy and Convergence

Comparing our experimental results after adjusting the problem-dependent parameters we found that our classification accuracy on hard pattern classification problems is very competitive, where almost on all datasets our results are at least as good as the highest reported in the literature or even outperform them. We note that our approach is an appropriate method for complex, real-world problems, whereas for simple problems or for conceptual problems, where the task is to learn rules, our approach shows its limitations.

Simulated annealing is considered an optimization method producing high quality results. The randomised nature of simulated annealing ensures asymptotic convergence to optimal solutions, however, it typically requires an exponential time and it is usually an efficient method to approximate optimal solutions in reasonable time. Combined with the perceptron algorithm, we have seen that the result is a successful search strategy to approximate optimal weight configurations, in order to solve pattern classification problems. Adjusting problem dependent parameters, i.e. those related to the sample size complexity and the network complexity, we found that approximation of optimal solutions is achieved for values that require considerably less computational time, than for a search time determined by increasing the length of inhomogeneous Markov chains. Therefore, we found that the LSA machine has fast convergence to high quality classification rates that are competitive to the best reported in the literature. Moreover, even when the adjustment of parameters implies increased time requirements, our results show still good competitive results by selecting parameters that lead to fast classification. Thus, for applications where accuracy is vital, like medical decisions, our classifier can be used for high quality decisions, whereas for applications where time is vital, like online applications, fast financial decisions, or stockmarket applications, high quality results or at least good competitive results are obtained in limited time. Thus, we found that our classifier combines accuracy and speed, making it an appropriate choice for complex real-world problems.

7.5 Contributions of this Work

Theory of computational complexity and performance analysis of approximation algorithms are currently emerging areas. While the heuristic methods described in Section 3.4.1 seem to do remarkably well on certain problems, very little has been established about the performance of these algorithms. As stated in [281] *"explaining and predicting the performance of these heuristic algorithms is one of the most important challenges facing the fields of optimization and algorithms."* According to [3], it is too early to achieve a complete unification and demystification of the area, whereas according to the NFLT, any approximation algorithm should be investigated by employing the specific nature of the problem. This work is towards the complexity and performance analysis of a specific stochastic local search algorithm, which is combined with the perceptron algorithm for pattern classification, namely the LSA machine. One of the tasks is to analyse the performance of the LSA machine and to give guidelines in the light of the NFLT for fine-tuning parameters in accordance to the nature of the problem. The issue of guidelines rises for almost all classifiers. While it is difficult to have mathematical proofs for these guidelines, they are based on plausible heuristics and have been found useful for many practical classification problems. Thus, a contribution (Contribution 1) will be to define rules for setting parameters that best fit the LSA machine with respect to a specific problem.

The depth vs size problem, as described in Section 2.3, is one of the hardest problems in theoretical computing, with very little process over the past decades. The debate about the relevance of Kolmogorov's theorem [219] to neural networks has been used to justify the focus on universal depth-two classifiers. Judd's work [201] gives also some preference to shallow depths, mainly due to simplicity in topology. Moving to larger depths, a number of questions arise of how to construct higher layers, what data should be used, what learning algorithm should be used, and what will be the size of layers, which all increases the overall complexity of the depth vs size problem. This work contributes in this problem (Contribution 2) by extending in depth the LSA machine and investigating the performance achieved by two types of LSA machines, a depth-two LSA machine and a depth-four LSA machine. To do so, a new learning approach is introduced for recursively creating and learning the training sets for further depths (Contribution 3). This novel approach gives future directions for how to treat higher levels in neural networks.

One of the major findings of our research is that the size of the network can be bounded by the input size of the problem and an approximate upper bound of $8 \cdot \sqrt{2^n/n}$ threshold gates as being sufficient for a small error rate, where

$n := \log|S_L|$ and S_L is the training set (Contribution 4). To propose an upper bound for the size of threshold circuits is important for the following reasons: Firstly, it allows us to *a piori* set one of the difficult problem dependent parameters, and hence we can focus on the rest of problem dependent parameters in a complex classification problem. Secondly, it contributes to a hard, unsolved problem in computer science, namely the size vs. depth problem for circuits. Thirdly, it is important for parallel processing theory in terms of resources required to approximate best classification rates. Fourthly, it shows that the number of nodes in neural networks depends on the available number of training examples, and hence it suggests that a constant size neural network is not appropriate for most of the classification problems. Consequently, in terms of the NFLT, we could say that there is *No-Free-Lunch* for circuit size, unless strict assumptions are made about the classification problem.

Results related to our work have been published already. Investigations about parameter settings and applications of the LSA machine are presented in [14, 229]. First trials to investigate the depth increase of the LSA machine are presented in [15, 16, 17, 18]. The new learning algorithm and the depth vs size problem are presented in [232]. Experimental results of our new learning algorithm are presented in [230, 231]. A first response to our approach can be found in [215, 216], where the authors are using our reported classification error from [14] for the WBCD dataset as one of the best results presented in the literature in order to compare their results to our work. The authors obtain the same classification error of 1.2 % as in our work in [14], but this result was obtained by our first experiments with the LSA machine and was improved since then to 0.4 % by using parameter optimization; cf. Chapter 5.

7.6 Future Work

We have demonstrated the successful application of a popular algorithm, the simulated annealing algorithm, in combination with a traditional classification algorithm, the classical perceptron method. Our approach is guided by the NFLT, which implies to relate our parameters to problem dependent learning and convergence properties. This methodology can be further used for investigating combinations of alternative combinatorial optimization algorithms, either with the perceptron algorithm, or with other pattern classification methods. Such combinatorial optimization methods for classification problems have been described in Section 3.6. One further way could be, for example, to use evolutionary algorithms in combination with the classical perceptron algorithm. The need for combined classifiers is a primary topic in [221].

We are concerned with two open research directions related to this area:

1. Further research on the hard and unsolved circuit depth vs circuit size problem;
2. Investigations on dimensionality reductions.

Research on Depth vs Size

The importance and difficulty of the depth vs size problem along with our findings have been described in previous sections. New complexity issues arise from the generation of new samples for training larger depths. Do large or small numbers of bit-positions created by the circuit topology and considered to be depth-three features have any impact on the classification rate? Further directions are to be investigate on uneven size of sample sets for training different layers, with more examples used for training depth-one and further issues of parameter settings for depth-four.

Research on Dimensionality Reduction

Further potential lies in the analysis of pre-processing steps. One popular way to pre-process data is feature selection in order to reduce the dimensionality of training data. A disadvantage of neural networks is the lack of information in the solution function about the dependence on variables of the problem, so that one knows the importance of each feature. Feature selection is a common method of preprocessing for identifying the least important features in a classification problem. Dimensionality is part of the complexity of a problem, and therefore reducing the dimensionality is one way to reduce the problem complexity. We have seen that, usually, the available training examples represent only a tiny fraction of the theoretically possible number of the domain of a function. A priori, we can argue that not all combinations of binary inputs are feasible (or even a small fraction only has indeed a valid interpretation). Taking this into account, we simply encoded (enumerated) the number of available training data instead of using the variable number of n features in the estimation of an upper bound of the circuit size. Therefore, our proposed bound is rather independent of the dimensionality of the problem. Since feature selection in a given (training) sample set might reduce the sample size itself, one might obtain a smaller n_L^{cp} and consequently a sharper bound in Equation (6.9).

Bibliography

[1] E.H.L. Aarts, F.M.J. De Bont, J.H.A. Habers, P.J.M. Van Laarhoven. Parallel Implementation of the Statistical Cooling Algorithm, *Integration*, 4(1986):209-238.

[2] E.H.L. Aarts, J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley & Sons, New York, 1997.

[3] E.H.L. Aarts, J.K. Lenstra (eds). *Local Search in Combinatorial Optimization*, Wiley & Sons, Chichester, 1998.

[4] E.H.L. Aarts, P.J.M. Van Laarhoven. Simulated Annealing: An Introduction *Statistica Neerlandica*, 43(1985):31-52.

[5] S. Abe. *Pattern Classification: Neuro-fuzzy Methods and their Comparison*, Springer Verlag, London, 2001.

[6] J. Abonyi, J.A. Roubos. Structure Identification of Fuzzy Classifiers, *5th online World Conference on Soft Computing in Industrial Applications (WSCS)*, 2000.

[7] Y.S. Abu-Mostafa. Lower Bound for Connectivity in Local-learning Neural Networks, *Journal of Complexity*, 4(1988):246-255.

[8] S. Agmon. The Relaxation Method for Linear Inequalities, *Canadian Journal of Mathematics*, 6(1954):382-392.

[9] S. Ahmad, V. Tresp. Some Solutions to the Missing Feature Problem in Vision, in S. Hanson, J.D. Cowan, C.L. Giles (eds) *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Mateo CA, 5(1993):393-400.

[10] A. Ahmad, L. Dey. A Feature Selection Technique for Classificatory Analysis, *Pattern Recognition Letters*, 26(1)(2005):43-56.

[11] R.K. Ahuja, T.L. Magnanti, J.B. Orlin. *Network Flows*, Prentice-Hall, New Jersey, 1993.

[12] A. Albrecht, On the Complexity to Approach Optimum Solutions by Inhomogeneous Markov Chains, in *Proc. Genetic and Evolutionary Computation Conference (GECCO'04)*, Springer-Verlag, LNCS series, 3102(2004):642-653.

[13] A. Albrecht, S.K. Cheung, K.S. Leung, and C.K. Wong. On the Convergence of Inhomogeneous Markov Chains Approximating Equilibrium Placements of Flexible Objects. *Computational Optimization and Applications*, 19(2001):179-208.

[14] A. Albrecht, G. Lappas, S.A. Vinterbo, C.K. Wong, L. Ohno-Machado. Two Applications of the LSA Machine, in *Proc. 9^{th} International Conference on Neural Information Processing (ICONIP'02)*, Singapore, 2002, pp. 184-189.

[15] A. Albrecht, G. Lappas. On the Evaluation of Dividing Samples for Training an Extended Depth LSA Machine, in *Proc. of the 5^{th} International Conference on Neural Networks and Applications (NNA'04)*, Udine, Italy, 2004.

[16] A. Albrecht, G. Lappas. On the Evaluation of Dividing Samples for Training an Extended Depth LSA Machine, *Transactions on Systems*, 3(3)(2004):1251-1257.

[17] A. Albrecht, G. Lappas. Classification Improvement by an Extended Depth LSA Machine, in *Proc. of the 4^{th} International Conference on Soft Computing, Optimization, Simulation and Manufacturing Systems (SOSM '04)*, Miami, 2004.

[18] A. Albrecht, G. Lappas. Classification Improvement by an Extended Depth LSA Machine, *Transactions on Systems*, 3(3)(2004):1120-1125.

[19] A. Albrecht and C.K. Wong. Combining the Perceptron Algorithm with Logarithmic Simulated Annealing, *Neural Processing Letters*, 14(1)(2001):75-83.

[20] A. Albrecht, C.K. Wong. DNF Approximation by Local Search, *Computational Optimization and Applications*, 27(1)(2004):53-82.

[21] E. Allender. A Note on the Power of Threshold Circuits, *IEEE Symposium on the Foundation of Computer Science*, (1989), pp 30.

[22] E. Allender, The Permanent Requires Large Uniform Threshold Circuits, in *Proc. 2nd Annual International Computing and Combinatorics Conference*, Springer-Verlag, LNCS series, 1090(1996):127-135.

[23] A.D. Anastasiadis, G.D. Magoulas and M.N. Vrahatis. New Globally Convergent Training Scheme Based on the Resilient Propagation Algorithm, *Neurocomputing*, 64(2005):253-270.

[24] J.A. Anderson. *An Introduction to Neural Networks*, MIT Press, Cambridge MA, 1996.

[25] S. Anily, A. Federgruen. Ergodicity in Parametric Nonstationary Markov Chains: an Application to Simulated Annealing Methods. *Operations Research*, 35(1987):867-874.

[26] S. Anily, A. Federgruen. Simulated Annealing Methods with General Acceptance Probabilities. *Journal of Applied Probability*, 24(1987):657-667.

[27] M. Anthony. *Discrete Mathematics of Neural Networks*, SIAM, Philadelphia PA, 2001.

[28] M. Anthony, P.L. Bartlett. *Neural Network Learning*, Cambridge University Press, UK, 1999.

[29] M. Anthony, *Boolean Functions and Artificial Neural Networks*, CDAM Research Report LSE-CDAM-2003-01, Department of Mathematics and Centre for Discrete and Applicable Mathematics, The London School of Economics and Political Science, London, UK, 2003.

[30] D.V. Arnold. Evolution Strategies in Noisy Environments - a Survey of Existing Work, in: L. Kallel, B. Nauts, A. Rogers (eds) *Theoretical Aspects of Evolutionary Computing*, Springer, Berlin, (2001):239-250.

[31] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer-Verlag, Berlin Heidelberg, 2003.

[32] M.E. Aydin, T.C. Fogarty. A Distributed Evolutionary Simulated Annealing Algorithm for Combinatorial Optimization Problems, *Journal of Heustics*, 10(2004):269-292.

[33] S. Bandyopadhyay, C.A. Murthy, S.K. Pal. Pattern Classification with Genetic Algorithms, *Pattern Recognition Letters*, 16(6)(1995):801-809.

[34] R.E. Banfield, L.O. Hall, K.W. Bowyer, D. Bhadoria, W.P. Kegelmeyer, S. Eschrich. A Comparison of Ensemble Creation Techniques, in *Proceedings Multiple Classifier Systems: 5th International Workshop, (MCS2004)*, Lecture Notes in Computer Science, 3077(2004):223-232.

[35] P.L. Bartlett. The Sample Complexity of Pattern Classification with Neural Networks: the Size of the Weights is More Important Than the Size of the Network, *IEEE Transactions on Information Theory*, 44(2)(1998):525-536.

[36] P.L. Bartlett and T. Downs. Using Random Weights to Train Multilayer Networks of Hard-limiting Units, *IEEE Trans. Neural Networks*, 3(1992):202-210.

[37] P.L. Bartlett, V. Maiorov, R. Meir. Almost Linear VC-dimension Bounds for Piecewise Polynomial Networks, *Neural Computation*, 10(1998):2159-2173.

[38] R. Battiti, G. Tecchiolli. Training Neural Nets with the Reactive Tabu Search, *IEEE Transactions on Neural Networks*, 6(1995):1185-1200.

[39] E.B. Baum. On the Capabilities of Multilayer Perceptrons, *Journal of Complexity*, 4(1988):193-215.

[40] E.B. Baum. The Perceptron Algorithm is Fast for Nonmalicious Distributions, *Neural Computation*, 2(1990):248-260.

[41] E.B. Baum, D. Haussler. What Size Net Gives Valid Generalization?, *Neural Computation*, 1(1)(1989):151-160.

[42] L.E. Baum, T. Petrie. Statistical Inference for Probabiblistic Functions of Finite State Markov Chains, *Annals of Mathematical Statistics*, 37(1966):1554-1563.

[43] R.G. Beiko, R.L. Charlebois. GANN: Genetic Algorithm Neural Networks for the Detection of Conserved Combination of Features in DNA, *BMC Bioinformatics*, 6(36)(2005), available online on www.biomedcentral.com/1471-2105/6/36.

[44] V. Beiu. Digital Integrated Circuit Implementations, in: E. Fiesler, R. Beale,(eds) *Handbook on Neural Computation*, Oxford University Press, 1997.

[45] J.L. Bentley. Fast Algorithms for Geometric Traveling Salesmen Problems, *ORSA Journal on Computing*, 4(1992):387-411.

[46] J.O. Berger. *Statistical Decision Theory and Bayesian Analysis*, Springer Verlag, New York, 1985.

[47] J.M. Bernardo, A.F.M. Smith. *Bayesian Theory*, Wiley, New York, 1996.

[48] D. Bertsimas, D. Simchi-Levi. A New Generation of Vehicle Routing Research: Robust Algorithms, Addressing Uncertainty *Operations Research*, 44(1996):286-304.

[49] F. Berzal, J.C. Cubero, D. Sanchez, J.M. Serrano. ART: A Hybrid Classification Model, *Machine Learning*, 54(1)(2004):67-92.

[50] C.M. Bishop. *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 2003.

[51] W.J. Blake, M. Ka, C.R. Cantor, J.J. Collins. Noise in Eukaryotic Gene Expression, *Nature* 422(6932)(2003):633-637.

[52] R. Blanco-Vega, J. Hernandez-Orallo, M.J. Ramirez-Quintana. Analysing the Trade-Off Between Comprehensibility and Accuracy in Mimetic Models, *Lecture Notes in Computer Science*, 3245(2004):338-346.

[53] A. Blum, A. Frieze, R. Kannan and S. Vempala. A Polynomial-time Algorithm for Learning Noisy Linear Threshold Functions, *Algorithmica*, 22(1998):35-52.

[54] A. Blum, T. Jiang, M. Li, J. Tromp, M. Yannakakis. Linear Approximation of Shortest Superstring, *Journal of the ACM*, 41(1994):630-647.

[55] A. Blum, R.L. Rivest. Training a 3-Node Neural Network is NP-Complete, *Neural Networks*, 5(1992):117-127.

[56] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension, *Journal of the ACM*, 36(4)(1989):929-965.

[57] K.D. Boese, A.B. Kahng. Best-so-far vs. Where-you-are: Implications for Optimal Finite-time Annealing *Systems and Controls Letters*, 22(1994):71-78.

[58] B.E. Boser, I. Guyon, V. Vapnik. A Training Algorithm for Optimal Margin Classifiers, in: D. Haussler (ed), *Proceedings of the 4th Workshop on Computational Learning Theory*, ACM Press, San Mateo CA, 1992, pp 144-152.

[59] H. Boström. Maximizing the Area Under the ROC Curve Using Incremental Reduced Error Pruning, in, *Proceedings of the ICML 2005 workshop on ROC Analysis in Machine Learning*, 2005, pp 144-152.

[60] B. Boutsinas, M.N. Vrahatis. Artificial Nonmonotonic Neural Networks, *Artificial Intelligence*, 132(2001):1-38.

[61] P. Brachetti, M.F. Ciccoli, G. Pillo, S. Lucidi. A New Version of Price's Algorithm for Global Optimization *Journal of Global Optimization*, 10(1997):165-184.

[62] A.P. Bradley. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms, *Pattern Recognition*, 30 (1997):1145-1159.

[63] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone. *Classification and Regression Trees*, Chapmann & Hall, New York, 1993.

[64] J.S. Bridle, R.K. Moore. Boltzmann Machines for Speech Pattern Processing, *Proccedings of the Institute of Acoustics*, 6(4)(1984):315-322.

[65] S.H. Brooks. Discussion of Random Methods for Locating Surface Maxima, *Operations Research*, 6(1958):244-251.

[66] B.G. Buchanan, E.A. Feigenbaum, J. Lederberg. On Generality and Problem Solving: A Case Study Using the DENDRAL Program. in: B. Meltzer, D. Michie (eds), *Machine Intelligence 6*, Edinburgh University Press, Edinburgh Scotland, 1971, pp 165-190.

[67] W.L. Buntine. A Guide to the Literature on Learning Probabilistic Networks From Data, *IEEE Transactions on Knowledge and Data Engineering*, 8(2)(1996):195-210.

[68] T. Bylander. Learning Linear Threshold Functions in the Presence of Classification Noise, *Proceedings 7th Annual ACM Workshop on Computational Learning Theory*, 1994, pp 340-347.

[69] T. Bylander. Learning Linear Threshold Approximations Using Perceptrons, *Neural Computation*, 7(1995):370-379.

[70] A. Casoto, F. Romeo, A.L. Sangiovanni-Vincentelli. A Parallel Simulated Annealing Algorithm for the Placement of Macro-cells, *IEEE Transactions on Computer-Aided Design*, 6(1987):838-847.

[71] Z. Cataltepe, Y.S. Abu-Mostafa, M. Magdon-Ismail. No Free Lunch for Early Stopping, *Neural Computation*, 11(1999):995-1009.

[72] H. Caussinus, P. McKenzie, D. Thérien, H. Vollmer. Nondeterministic NC^1 computation, *J. Comput. System Sci*, 57(1998):200-212.

[73] V. Ćerný. Thermodynamical Approach to the Traveling Salesman Problem: an Efficient Simulation Algorithm, *Journal of Optimization Theory and Applications*, 45(1985):41-51.

[74] E.I. Chang, R.P. Lippmann. Using Genetic Algorithms to Improve Pattern Classification Performance, *Proceedings of the 1990 Conference on Advances in Neural Information Processing Systems*, Denver CO, 1990, pp 797-803.

[75] W. Chen, F. Lin, J. Li, B. Zhang. A New Prosodic Phrasing Model for Chinese TTS Systems, in *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium (NLPRS 2001)*, Tokyo, Japan, 2001, pp 169-175.

[76] V. Cherkassky, F. Mulier. *Learning from Data: Concepts, Theory and Methods*, Wiley, New York, 1998.

[77] D. Chiang, W. Chen, Y. Wang, L. Hwang. Rules Generation from the Decision Tree, *Journal of Information Science and Engineering*, 17(2001):325-339.

[78] N. Christianini, J. Swave-Taylor, N. Cristianini. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, Cambridge UK, 2000.

[79] M. Christoph, K.H. Hoffmann. Scaling Behaviour of Optimal Simulated Annealing Schedules. *Journal of Physics A*, 26(1993):3267-3277.

[80] V. Chvátal. *Linear Programing*, F.H. Freeman and Co, New York, 1983.

[81] J.G. Cleary, L.E. Trigg, G. Holmes, M.A. Hall. Experiences with a Weighted Decision Tree Learner, In: M. Bramer, A. Preece, F. Coenen, (eds.), *Research and Development in Intelligent Systems XVII*, BCS Series, Springer-Verlag, 2000, pp 35-47.

[82] E.G.Jr. Coffman (ed). *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, 1976.

[83] S.A. Cook. The Complexity of Theorem Proving Procedures, *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, ACM, 1971, pp 151-158.

[84] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver. *Combinatorial Optimization*, John Wiley & Sons, New York, 1998.

[85] D. Corne, J. Knowles. No Free Lunch and Free Leftovers Theorems for Multiobjective Optimization Problems, *in Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)* Springer LNCS, pp 327-341.

[86] D. Corne, J. Knowles. Some Multiobjective Optimizers are Better than Others, *in Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, IEEE Press, 4(2003):2506-2512.

[87] E.M. Corwin, A.M. Logar and W.J.B. Oldham. An Iterative Method for Training Multilayer Networks with Threshold Functions, *IEEE Trans. Neural Networks*, 5(1994):507–508.

[88] T.M. Cover, P.E. Hart. Nearest Neighbour Pattern Classification, *IEEE Transactions on Information Theory*, 13(1)(1967):21-27.

[89] N. Cristianini, C. Campbell, J. Shawe-Taylor. A Multiplicative Updating algorithm for Training Support Vector Machine, *in Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, 1999.

[90] G.A. Croes. A Method for Solving Travelling-Salesman Problems, *Operations Research*, 6(1958):791-812.

[91] J. Culberson. On the Futility of Blind Search, *Evolutionary Computation*, 6(2)(1999):109-127.

[92] Y.L. Cun, J.S. Denker, S.A. Solla. Optimal Brain Damage, in D.S. Touretzky(ed) *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Mateo CA, 2(1990):598-605.

[93] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function, *Mathematical Control Signal Systems*, 2(1989):303-314.

[94] B.V. Dasarathy (ed). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society, Washington, 1991.

[95] L. Davis. Bit-climbing Representational Bias and Test Suite Design, in L. Booker, R. Belew (eds) *Proceedings of the* 4^{th} *international conference on Genetic Algorithms*, Morgan Kauffman (1991).

[96] M. Dell' Amico, F. Maffioli, S. Martello (eds). *Annoted Bibliographies in Combinatorial Optimization*, Wiley, Chichester, 1997.

[97] W.A. Demski. *No Free Lunch: Why Specified Complexity Cannot be Purchased without Intelligence*, Rowman & Littlefield Publishers, 2001.

[98] P.A. Devijver, J. Kittler (eds). *Pattern Recognition Theory and Applications*, Springer Verlag, Berlin, 1987.

[99] T.G. Diettrich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2)(2000):139-157.

[100] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerishe Mathematik*, 1(1959):269-271.

[101] P. Domingos. How to Get a Free Lunch: A Simple Cost Model for Machine Learning, *in Proceedings AAAI-98/ICML-98 Workshop on the Methodology of Applying Machine Learning*, Madison WI, 1998, pp 1-7.

[102] S.E. Dreyfus, A.M. Law. *The Art and Theory of Dynamic Programming*, Academic Press Inc., New York, 1977.

[103] S. Droste, T. Jansen, I. Wegener. Perhaps Not a Free Lunch But At Least a Free Apetizer, *in Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando FL, 1(1999):833-839.

[104] S. Droste, T. Jansen, I. Wegener. Optimization with Randomised Search Heuristics - the (A)NFL Theorem, Realistic Scenarios and Difficult Functions, *Theoretical Computer Science*, 287(2003):131-144.

[105] J. Du, J.Y.T. Leung. Minimizing Total Tardiness on One Machine is NP-hard, *Mathematics on Operations Research*, 15(1990):483-495.

[106] R.O. Duda, P.E. Hart, D.G. Stork. *Pattern Classification*, Wiley-Interscience, New York, 2001.

[107] D. Eads, D. Hill, S. Davis, S. Perkins, J. Ma, R. Porter, J. Theiler. Genetic Algorithms and Support Vector Machines for Time Series Classification, *in Proceedings of the 5th Conference on the Applications and Science of Neural Networks, Fuzzy Systems and Evolutionary Computation*, 2002, pp 74-85.

[108] J. Edmonds, R.M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems, *Journal of the ACM*, 19(2)(1972):248-264.

[109] A.E. Eiben, E.H.L. Aarts, K.M. Van Hee. Global Convergence of Genetic Algorithms: A Markov Chain Analysis, in H.P. Schwefel, R. Männer (eds) *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, Springer, Berlin, 496(1991):4-12.

[110] P.W. Eklund. A Performance Survey of Public Domain Supervised Machine Learning Algorithms, *KVO Technical Report*, University of Queensland, 2002.

[111] V. Estruch, C. Ferri, J. Hernandez, M.J. Ramirez. Beam Search Extraction and Forgetting strategies on Shared Ensembles, in *Proc. of 4th International Workshop on Multiple Classifiers Systems (MCS03)*, LNCS, Springer Verlag, 2709(2003):206-216.

[112] V. Estruch, C. Ferri, J. Hernandez, M.J. Ramirez. Mimetic Classifiers, in *Proc. of Conference on Machine Learning and Data Mining (MLDM03)*, LNAI Springer Verlag, 2734(2003):156-171.

[113] S. Even. *Graph Algorithms*, Computer Science Press, Maryland, 1979.

[114] J.A. Feldman, D.H. Ballard. Connectionist Models and their Properties, *Cognitive Science*, 6(1982):205-254.

[115] W. Feller. *An Introduction to Probability Theory and its Applications*, Wiley, New York, 1950.

[116] C. Ferri, P. Flach, J. Hernandez-Orallo, Delegating classifiers, in *Proceedings of the twenty-first International Conference on Machine learning, ACM International Conference Proceeding Series*, Banff, Canada, 69(2004):37-44.

[117] R.A. Fisher. The Use of Multiple Measurements in Taxonomic Problems, *Annals of Eugenics*, 7(1936):179-188.

[118] R.W. Floyd. Algorithm 97: Shortest Path, *Communication ACM*, 5(6)(1962):345.

[119] G. Folino, C. Pizzuti, G. Spezzano. Genetic Programming and Simulated Annealing: A Hybrid Method to Evolve Decision Trees, *Lecture Notes in Computer Science*, 1802(2004):294-303.

[120] M.L. Fredman, D.S. Johnson, L.A. McGeoch, G. Ostheimer. Data Structures for Traveling Salesmen, *Journal of Algorithms*, 18(1995):432-479.

[121] T.T. Friess, N. Cristianini, C. Campbell. The Kernel Adatron Algorithm: A Fast and Simple Learning Procedure for Support Vector Machine, *in Proceedings of the* 15^th *International Conference on Machine Learning*, Morgan Kaufmann, 1998.

[122] H. Frolich, O. Chapelle, B. Schölkopf. Feature Selection for Support Vector Machines by Means of Genetic Algorithms, *in Proceedings of the International IEEE Conference on Tools with AI*, 2003, pp 142-148.

[123] L.R.Jr. Ford, D.R. Fulkerson. *Flows in Networks*, Princeton University Press, N.J., 1962.

[124] K. Fukunaga. *Introduction to Statistical Pattern Recognition*, Academic Press, New York, 1990.

[125] J. Fürnkranz, P. Flach. Roc'n'rule Learning - Towards a Better Understanding of Covering Algorithms. *Machine Learning*, 58(2005):39-77.

[126] A. Futschik, C. Pflug. Confidence Sets for Discrete Stochastic Optimization, *Annals of Operations Research*, 56(1995):95-108.

[127] S.I. Gallant. Perceptron-based Learning Algorithms, *IEEE Trans. on Neural Networks*, 1(1990):179-191.

[128] M.R. Garey, R.L. Graham, J.D. Ullman. An Analysis of Some Packing Algorithms, *Combinatorial Algorithms (Courant Computer Science Symposium)*, 9(1972):39-47.

[129] M.R. Garey, D.S. Johnson. Approximation Algorithms for Combinatorial Problems: An Annotated Bibliography, in J.F. Traub (ed) *Algorithms and Complexity: New Directions and Recent Results*, Academic Press Inc., New York, 1976.

[130] M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, 1979.

[131] M.R. Garey, D.S. Johnson. Strong NP-completeness Results: Motivation, Examples, and Implications, *Journal of the ACM*, 25(1978):499-508.

[132] R.S. Garfinkel. Motivation and Modelling, in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds) *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985, pp 17-36.

[133] S.B. Gelfand, S.K. Mitter. Analysis of Simulated Annealing for Optimization. *Proceedings of the 24th IEEE Conference on Decision and Control*, New York (1985), pp 779-786.

[134] S. Geman, D. Geman. Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1984):721-741.

[135] C.-F. Geyer. *Epikur*, Junius-Verlag, Hamburg, 2000.

[136] B. Gidas. Nonstationary Markov Chains and Convergence of the Annealing Algorithm, *Journal of Statistical Physics*, 39(1985):73-131.

[137] R. Giraldez, J.S. Aguilar-Ruiz, J.C. Riquelme. Knowledge-based Fast Evaluation for Evolutionary Learning, *IEEE Transactions on Systems, Man & Cybernetics Part C. Special Issue on Knowledge Extraction and Incorporation in Evolutionary Computation*, 35(2)(2005):254-261.

[138] F. Girosi, T. Poggio. Representation Properties of Networks: Kolmogorov's Theorem is Irrelevant, *Neural Computation*, 1(4)(1989):465-469.

[139] F. Glover. Tabu Search: part I, *ORSA Journal on Computing*, 1(1989):190-206.

[140] F. Glover. Tabu Search: part II, *ORSA Journal on Computing*, 2(1990):4-32.

[141] M.X. Goemans, D.P. Williamson. New $\frac{3}{4}$-approximation Algorithms for the Maximum Satisfiability Problem, *SIAM Journal of Discrete Mathematics*, 7(1994):656-666.

[142] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading MA, 1989.

[143] P.W. Goldberg, M.R. Jerrum. Bounding the Vapnik-Chervonenkis Dimension of Concept Classes Parametrised by Real Numbers, *Machine Learning*, 18(2/3)(1995):131-148.

[144] R.M. Golden. *Mathematical Methods for Neural Network Analysis and Design*, MIT Press, Cambridge MA, 1996.

[145] E. Goles Chacc, J. Olivos. The Convergence of Symmetric Threshold Automata, *Information and Control*, 51(1981):98-104.

[146] R. Goodman and Z. Zeng. A Learning Algorithm for Multilayer Perceptrons with Hard-limiting Threshold Units, in *Proc. IEEE Neural Networks for Signal Processing*, 1994, pp 219-228.

[147] C. Goutte. Note on Free Lunches and Cross-Validation, *Neural Computation*, 9(6)(1997):1245-1249.

[148] M. Grana, A.D. Anjou, F.X. Albizuri, M. Hernandez, F.J. Torrealdea, A. de la Hera, A.I. Gonzalez. Experiments of Fast Learning with High Order Boltzmann Machines, *Applied Intelligence*, 7(4)(1997):1065-1076.

[149] D.M. Green, J.A. Swets. *Signal Detection Theory and Psychophysics*, Wiley, New York, 1974.

[150] H.J. Greenberg, W.E. Hart, G. Lancia. Opportunities for Combinatorial Optimization in Computational Biology *INFORMS J. Comput.*, 16(2004):211-231.

[151] M. Grotschel, L. Lovasz. Combinatorial Optimization: A Survey, *Handbook of Combinatorics*, North-Holland, Netherlands, 2005.

[152] W.J. Gutjahr. A Converging ACO Algorithm for Stochastic Combinatorial Optimization, in A. Albrecht, K. Steinhöfel (eds) *Stochastic Algorithms: Foundations and Applications*, LNCS, 2827(2003):10-25.

[153] W.J. Gutjahr, C. Pflug. Simulated Annealing for Noisy Cost Functions, *Journal of Global Optimization*, 8(1996):1-13.

[154] R.F. Hadley, M.B. Hayward. Strong Semantic Systematicity from Hebbian Connectionist Learning, *Minds and Machines*, 7(1997):1-37.

[155] O. Häggström. *Finite Markov Chains and Algorithmic Applications*, Cambridge University Press, UK, 2002.

[156] B. Hajek. Cooling Schedules for Optimal Annealing, *Mathematics of Operations Research*, 13(1988):311-329.

[157] B. Hajek, G. Sasaki. Simulated Annealing: to Cool it or not. *Systems Control Letters*, 12(1989):527-542.

[158] A. Haken, M. Luby. Steepest Descent can Take Exponential Time for Symmetric Connection Networks, *Complex Systems*, 2(1988):191-196.

[159] L.A. Hall. Approximation Algorithms for Scheduling, in D. Hochbaum (ed), *Approximation Algorithms for NP-hard problems*, PWS Publishing, Boston, MA, 1997.

[160] M.H. Hassoun. *Foundamentals of Artificial Neural Networks*, MIT Press, Cambridge MA, 1995.

[161] D. Haussler, M. Kearns, R. Schapire. Bounds on the Sample Complexity of Bayesian Learning Usng Information Theory and the VC Dimension, *Machine Learning*, 14(1994):83-113.

[162] B. Hayes-Roth, F. Hayes-Roth, Concept Learning and the Recognition and Classification of Exemplars. *Journal of Verbal Learning and Verbal Behavior*, 16(1977):321-338.

[163] S. Haykin. *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.

[164] D.O. Hebb. *The Organization of Behaviour*, Wiley, New York, 1949.

[165] R. Hecht-Nielsen. Theory of the Backpropagation Neural Network, *in Proceedings of the International Joint Conference on Neural Networks (IJCNN)*,IEEE, New York, 1(1989):593-605.

[166] R. Hecht-Nielsen. *Neurocomputing*, Addison Wesley, New York, 1990.

[167] M. Held, R.M. Karp. A Dynamic Programming Approach to Sequencing Problems, *Journal SIAM*, 10(1)(1962):196-210.

[168] M. Held, R.M. Karp. The Travelling Salesman Problem and Minimum Spanning Trees, *Operations Research*, 18(1970):1138-1162.

[169] R.J. Henery. Classification, in D. Michie, D.J. Spiegelhalter, C.C. Taylor (eds), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, New York, 1994.

[170] A. Hertz, D. De Warra. The Tabu Search Metaheuristic: How we Used it, *Annals of Mathematics and Artificial Intelligence*, 1(1991):111-121.

[171] J. Hertz, A. Krogh, R.G. Palmer. *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City, CA, 1991.

[172] G.E. Hinton, J.A. Anderson (eds). *Parallel Models of Associative Memory*, Erlbaum, Hillsdale NJ, 1981.

[173] G.E. Hinton, T.J. Sejnowski. Optimal Perceptual Inference, *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Washington DC, 1983, pp 448-453.

[174] G.E. Hinton, T.J. Sejnowski. Learning and Relearning in Boltzmann Machines, in D.E. Rumelhart, J.L. McClelland, The PDP Research Group (eds), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1*, MIT press, Cambridge, MA, 1986

[175] D. Hochbaum (ed). *Approximation Algorithms for NP-hard Problems*, PWS Publishing, Boston, MA, 1997.

[176] K.-U. Höffgen. Computational Limitations on Training Sigmoid Neural Networks, in *Proc. 2nd Int. Conf. on Artificial Neural Networks*, (1992) pp. 109-112.

[177] K.-U. Höffgen, H.-U. Simon. Robust Trainability of Single Neurons, *Proc.* 5^{th} *Annual ACM Workshop on Computational Learning Theory*, 1992, pp 428-439.

[178] S.B. Holden, P.J.W. Rayner. Generalisation and PAC Learning: Some New Results for the Class of Generalised Single-Layer Networks, *IEEE Transactions on Neural Networks*, 6(2)(1995):368-379.

[179] J.H. Holland. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Michigan, 1975.

[180] K. Hoffman. Combinatorial Optimization: Current Successes abd Directions for the Future, *Journal of Computational and Applied Mathematics*, 124(1)(2000):341-360.

[181] H.H. Hoos, T. Stutzle. *Stochastic Local Search: Foundations and Applications*, Elsevier, San Francisco, 2005.

[182] Y.C. Ho, D.L. Pepyne. Simple Explanation of the No-Free-Lunch Theorem and Its Implications, *Journal of Optimization Theory and Applications*, 115(3)(2002):549-570.

[183] J.J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *in Proceedings of the National Academy of Sciences of the United States of America*, 79(1982):2554-2558.

[184] J.J. Hopfield, D.W. Tank. 'Neural' Computation of Decisions in Optimization Problems, *Biological Cybernetics*, 52(1985):141-152.

[185] K. Hornik, M. Stinchcombe, H.L.Jr. White. Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, 2(5)(1989):359-366.

[186] E. Horowitz, S. Sahni. *Foundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1984.

[187] D.R. Hush, J.M. Salas, B.G. Horne. Error Surfaces for Multi-layer Perceptrons, *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, IEEE, New York, 1(1991):759-764.

[188] H.B. Hwarng, N.F. Hubele. Boltzmann Machines that Learn to Recognize Patterns on Control Charts, *Statistics and Computing*, 2(4)(1992):191-202.

[189] O.H. Ibarra, C.E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems, *Journal of the ACM*, 22(1975):463-468.

[190] C. Igel, M. Toussaint. On Classes of Functions for which No Free Lunch Results Hold, *Information Processing Letters*, 86(2003):317-321.

[191] C. Igel, M. Toussaint. A No-Free-Lunch Theorem for Non-Uniform Distributions of Target Functions, *Journal of Mathematical Modelling and Algorithms*, 3(4)(2004):313-322.

[192] E. Ignall, L. Schrage. Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems, *Operations Research*, 13(3)(1965):400-412.

[193] F. Imbault, K. Lebart. A Stochastic Optimization Approach for Parameter Tuning of Support Vectors Machines, *17th International Conference on Pattern Recognition (ICPR 04)*, 4(2004):597-600.

[194] L. Ingber. Simulated Annealing: Practice Versus Theory, *Mathematical and Computer Modelling*, 18(1993):280-289.

[195] P. Jackson. *Introduction to Expert Systems*, Addison Wesley, Reading MA, 1990.

[196] A.K. Jain, J. Mao, K.M. Mohiuddin. Artificial Neural Networks: A Tutorial, *Computer*, 29(3)(1996):31-44.

[197] N. Japkowicz, S. Stephen. The Class Imbalance Problem: A Systematic Study, *Intelligent Data Analysis*, 6(5)(2002):429-450.

[198] Y. Jing, V. Pavlovic, J.M. Rehg. Efficient Discriminative Learning of Bayesian Network Classifiers via Boosted Augmented Naive Bayes, in *Proc. 22nd International Conf. on Machine Learning (ICML)*, Bonn, Germany, 2005.

[199] D.S. Johnson. Approximation Algorithms for Combinatorial Problems, *Journal of Computer and System Sciences*, 9(1974):256-278.

[200] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis. How Easy is Local Search?, *Journal of Computer and System Sciences*, 37(1988):79-100.

[201] J.S. Judd. On the Complexity of Loading Shallow 1-dimension Networks, *Journal of Complexity*, 4(1988):177-192.

[202] J.S. Judd. *Neural Network Design and the Complexity of Learning*, MIT Press, Cambridge MA, 1990.

[203] A. Kalton, P. Langley, K. Wagsta, J. Yoo. Generalized Clustering, Supervised Learning, and Data Assignment, in *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, ACM Press, San Francisco, CA, 2001, pp 299-304.

[204] A. Kandel. *Fuzzy Techniques in Pattern Classification*, Wiley, New York, 1982.

[205] D. Karaboga, A. Kalinli. Training Recurrent Neural Networks for Dynamic System Identification Using Parralel Tabu Search Algorithm, *12th IEEE International Symposium on Intelligent Control*, Istanbul, 1997, pp 113-118.

[206] H. Karloff. *Linear Programing*, Birkhäuser, Boston, 1991.

[207] N. Karmarkar. A New Polynomial-time Algorithm for Linear Programming, *Combinatorica*, 4(1984):373-396.

[208] R.M. Karp. Reducibility among Combinatorial Problems, in R.E. Miller, J.W. Thatcher (eds) *Complexity of Computer Computations*, Plenum Press, New York (1972), pp 85-103.

[209] M. Karpinski, A.J. Macintyre. Polynomial Bounds for VC Dimension of Sigmoidal and General Pfaffian Neural Networks, *Journal of Computer and System Sciences*, 54(1)(1997):169-176.

[210] J. Kastner, J. Solomon, S. Fraser. Modeling a Hox Gene Network in Silico Using a Stochastic Simulation Algorithm, *Dev Biol*, 246(1)(2002):122-131.

[211] W. Kern. On the Depth of Combinatorial Optimization Problems. *Discrete Appied Mathematics*, 43(1993):115-129.

[212] L.G. Khachian. A Polynomial Algorithm for Linear Programming, *Doklady Akad. Nauk USSR*, 244(5)(1979):1093-1096.

[213] S. Kirkpatrick, C.D. Gelatt, Jr.M.P. Vecchi. Optimization by Simulated Annealing, *Science*, 220(1983):671-680.

[214] J.K. Kishore, L.M. Patnaik, V. Mani, V.K. Agrawal. Application of Genetic Programming for Multiclassification Pattern Classification, *IEEE Transactions on Evolutionary Computation*, 4(3)(2000):242-258.

[215] T. Kiyan, T. Yildirim. Breast Cancer Diagnosis Using Statistical Neural Networks, in *Proceedings of the International XII Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN '03)*, Hanakkale, Turkey, E8(2003):754-759.

[216] T. Kiyan, T. Yildirim. Breast Cancer Diagnosis Using Statistical Neural Networks, *Journal of Electrical and Electronics Engineering*, 4(2)(2004):1149-1153.

[217] T. Kohonen. *Self-Organization and Associative Memory*, Springer Verlag, Berlin, 1988.

[218] P. Koiran, E.D. Sontag. Neural Networks with Quadratic VC Dimension, *Journal of Computer and System Sciences*, 54(1)(1997):190-198.

[219] A.N. Kolmogorov. On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of one Variable and Addition, *Doklady Akademiia Nauk*, 114(5)(1957):953-956.

[220] M.W. Krentel. The Complexity of Optimization Problems, *Journal of Computer and System Sciences*, 36(1988):490-509.

[221] L. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*, Wiley-IEEE, New York, 2004.

[222] V. Kúrková. Kolmogorov's Theorem is Relevant, *Neural Computation*, 3(4)(1991):617-622.

[223] V. Kúrková. Kolmogorov's Theorem and Multilayer Neural Networks, *Neural Computation*, 5(3)(1992):501-506.

[224] N. Landwehr, M. Hall, E. Frank. Logistic Model Trees,*Machine Learning*, 59(1-2)(2005):161-205.

[225] P.C.R. Lane, F. Gobet. Developing Reproducible and Comprehensible Computational Models, *Artificial Intelligence*, 144(2003):251-263.

[226] P.C.R. Lane, J.B. Henderson. Incremental Syntactic Parsing of Natural Language Corpora with Simple Synchrony Networks, *IEEE Transactions on Knowledge and Data Engineering*, 13(2)(2001):219-231.

[227] P.C.R. Lane, J.B. Henderson. Towards Effective Parsing with Neural Networks: Inherent Generalisations and Bounded Resource Effects, *Applied Intelligence*, 19(2003):83-99.

[228] A. Lapedes, R. Farber. How Neural Nets Work, in D.Z. Anderson (ed) *Advances in Neural Information Processing Systems*, American Institute of Physics, New York, 1988, pp 442-456.

[229] G. Lappas, V. Ambrosiadou. Binary and Multicategory Classification Accuracy of the LSA Machine, in *Proceedings of the International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2003)*, Kastoria, Greece, 2003, pp 340-345.

[230] G. Lappas, K. Steinhöfel, A. Albrecht. Classification of Splice-Junction Gene Sequences by a Special Type of Threshold Circuits, *Texts in Algorithmics: Algorithms in Bioinformatics*, 6(2006):27-43.

[231] G. Lappas. Generating Classification Circuits for Approximating Biological Decision Problems, in *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics (ICNAAM 2005), Computational Biology: Algorithm and Models Minisymposium*, Rhodes, 2005, Wiley-VCH, pp 800-808.

[232] G. Lappas, R.J. Frank, A. Albrecht. A Computational Study on Circuit Size vs. Circuit Depth, *International Journal on Artificial Intelligence Tools*, 15(2)(2006):143-162.

[233] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.

[234] E.L. Lawler, D.E. Wood. Branch and Bound methods: A Survey, *Operations Research*, 14(1966):699-719.

[235] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985.

[236] P.M. Lee. *Bayesian Statistics: An Introduction*, Edward Arnold, London, 1989.

[237] J.K. Lenstra, D.B. Shmoys, *É.* Tardos. Approximation Algorithms for Scheduling Unrelated Parallel Machines, *Mathematical Programming*, 46(1990):259-271.

[238] M. Li. Towards a DNA Sequencing Theory, *Proceedings of the 31st IEEE Annual Symposium on Foundations of Computer Science*, 1990, pp 125-134.

[239] M. Li, P.M.B. Vitanyi. Inductive Reasoning and Kolmogorov Complexity, *Journal of Computer System and Sciences*, 44(2)(1992):343-384.

[240] S. Lin. Computer Solutions to the Travelling Salesman Problem, *BSTJ*, 44(10)(1965):2245-2269.

[241] F.T. Lin, C.Y. Kao, C.C. Hsu. Applying the Genetic Approach to Simulated Annealing in Solving Some NP-hard Problems, *IEEE transactions on Systems, Man and Cybernetics*, 23(1994):1752-1767.

[242] J.D.C. Little, K.G. Murty, D.W. Sweeny, C. Karel. An Algorithm for the Travelling-Salesman Problem, *Operations Research*, 11(1963):972-989.

[243] R.J.A. Little, D.B. Rubin. *Statistical Analysis with Missing Data*, John Wiley and Sons, New York, 1987.

[244] A.C Lorena, A.C.P.F.de Carvalho. Evaluation of Noise Reduction Techniques in the Splice Junction Recognition Problem, *Genetic Molecular Biology*, 27(4)(2004):665-672.

[245] C.K. Looi. Neural Network Methods in Combinatorial Optimization, *Computers and Operations Research*, 19(1992):191-208.

[246] L. Lovász. *Combinatorial Problems and Exercises*, North-Holland, Amsterdam, 1993.

[247] L. Lovász, M.D. Plummer. *Matching Theory*, North-Holland, Amsterdam, 1986.

[248] D. Luenberger. *Optimization by Vector Space Methods*, J. Wiley & Sons, New York, 1997.

[249] O.B. Lupanov. On a Method to Design Control Systems - The Local Encoding Approach (in Russian), *Problemy Kibernetiki*, 14(1965):31-110.

[250] O.B. Lupanov. On the Design of Circuits by Threshold Elements, *Problemy Kibernetiki*, 26(1973):109-140.

[251] W. Maas. Bounds for the Computational Power and Learning Complexity of Analog Neural Nets, in *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, 1993, pp 335-344.

[252] W. Maass. On the complexity of learning on neural nets, in *Proc. Computational Learning Theory: EuroColt'93*, Oxford University Press, 1994, pp 1-17.

[253] W. Maass, R.A. Legenstein, N. Bertschinger. Methods for estimating the computational power and generalization capability of neural microcircuits, in *Proc. Advances in Neural Information Processing Systems*, 2005.

[254] M. Magdon-Ismail. No Free Lunch for Noise Prediction, *Neural Computation*, 12(3)(2000):547-564.

[255] G.D. Magoulas, M.N. Vrahatis, T.N. Grapsa and G.S. Androulakis, A Training Method for Discrete Multilayer Neural Networks, in S.W. Ellacot, J.C. Mason and I.J. Anderson (eds.), *Mathematics of Neural Networks: Models, Algorithms & Applications*, Kluwer Academic Publishers, Operations Research Computer Science Interfaces series, 1997.

[256] U. Mahlab, J. Shamir, J.H. Caulfield. Genetic Algorithm for Optical Pattern Recognition, *Optics Letters*, 16(1991):648-650.

[257] O. Martin, S.W. Otto, E.W. Felten. Large-step Markov Chains for the TSP Incorporating Local Search Heuristics, *Operations Research Letters*, 11(1992):219-224.

[258] S.F. Mashri, G.A. Bekey. A Global Optimization Algorithm Using Adaptive Random Search, *Applied Mathematics and Computation*, Elsevier North Holland, 7(1980):353-375.

[259] G. Mayraz, G.E. Hinton. Recognizing Handwritten Digits Using Hierarchical Products of Experts, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2)(2002):189-198.

[260] D. McClish. Analyzing a Portion of the ROC Curve, *Medical Decision Making*, 9(1989):190-195.

[261] W.S. McCulloch, W.H. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics*, 5(1943):115-137.

[262] C.J. Mertz, P.M. Murphy. *UCI Repository of Machine Learning Databases*, http://www.ics.uci.edu/ mlearn/MLRepository.html, (1996).

[263] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller. Equation of State Calculations by Fast Computing Machines, *Journal of Chemical Physics*, 21(1953):1087-1092.

[264] R.S. Michalsky. Pattern Recognition as Rule-guided Inductive Inference, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4)(1980):349-361.

[265] R.S. Michalsky, G. Tecuci (eds). *Machine Learning: A Multistrategy Approach*, Morgan Kaufmann, San Mateo CA, 1994.

[266] D. Michie, D.J. Spiegelhalter, C.C. Taylor (eds). *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, New York, 1994.

[267] B. Minaei-Bidgoli, A. Topchy, W.F. Punch. A Comparison of Resampling Methods for Clustering Ensembles, *International Conference on Machine Learning; Models, Technologies and Applications, (MLMTA 2004)*, Las Vegas, NE, 2004.

[268] M. Minsky, S. Papert. *Perceptrons*, MIT Press, Cambridge, MA, 1969.

[269] M. Mitchell. *An Introduction to Genetic Algorithms*, MIT Press, Cambridge MA, 1996.

[270] T.M. Mitchell. *Machine Learning*, McGraw-Hill, New York, 1997.

[271] H. Mühlenbein, M. Gorges-Schleuter, O. Krämer. Evolution Algorithms in Combinatorial Optimization *Parallel Computing*, 7(1988):65-85.

[272] E.I. Neciporuk. The Synthesis of Networks from Threshold Elements, *Problemy Kibernetiki II*, 1964, pp 49-62.

[273] G.L. Nemhauser, L. Wolsey. Integer and Combinatorial Optimization, Wiley & Sons, New York, 1988.

[274] H. Ney. On the Probabilistic Interpretation of Neural Network Classifiers and Discriminative Training Criteria, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2)(1995):107-119.

[275] T.A.J. Nicholson. A Sequential Method for Discrete Optimization Problems and its Application to the Assignement, Traveling Salesman and Three Scheduling Problems, *Journal of the Institute of Mathematics and its Applications*, 13(1965):362-375.

[276] L. Ohno-Machado, W.K. Kuo. Stochastic Algorithms for Gene Expression Analysis, in A. Albrecht, K. Steinhofel (eds) *Stochastic Algorithms: Foundations and Applications*, LNCS, 2827(2003):39-49.

[277] M. Opper, O. Winther. Gaussian Processes and SVM: Mean Field and Leave-One-Out, in A.J. Smola, P.L. Bartlett, B. Schölkopf, D. Schurmans (eds), *Advances in Large Margin Classifiers*, MIT Press, Cambridge MA, 2000.

[278] G. Pajares, J.M. de l Cruz. On Combining Support Vector Machines and Simulated Annealing in Stereovision Matching, *IEEE Transitions on Sestem Man Cybernetics Part B: Cybernetics*, 34(4)(2004):1646-1657.

[279] S.K. Pal, P.P. Wank (eds). *Genetic Algorithms for Pattern Recognition*, CRC Press, Boca Raton FL, 1996.

[280] C.H. Papadimitriou. *Computational Complexity*, Addison-Wesley, Reading, Mass., 1994.

[281] C.H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, New York, 1998.

[282] E. Parzen. On Estimation of a Probability Density Function and Mode, *Annals of Mathematical Statistics*, 33(3)(1962):1065-1076.

[283] E.A. Patrick, F.P. Fischer. A Generalized k-nearest Neighbour Rule, *Information and Control*, 16(2)(1970):128-152.

[284] S. Paul, S. Kumar. Subsethood-product Fuzzy Neural Inference System (SuPFuNIS), *IEEE Transactions on Neural Networks*, 13(3)(2002):578-599.

[285] C.A. Pena-Reyes, M. Sipper. Fuzzy CoCo: A Cooperative Coevolutionary Approach to Fuzzy Modeling, *IEEE Trans. on Fuzzy Systems*, 9(2001):727-737.

[286] M. Pertselakis, N. Tsapatsoulis, S. Kollias, S. Stafylopatis. An Adaptive Resource Allocating Neural Fuzzy Inference System, in *Proceedings of IEEE Inteligent Systems Application to Power Systems (ISAP'03)*, Limnos, Greece, 2003.

[287] C. Peterson, B. Söderberg. Artificial Neural Networks, in E.H.L. Aarts, J.K. Lenstra (eds), *Local Search in Combinatorial Optimization*, Wiley & Sons, Chichester, 1998.

[288] V.P. Plagianakos, G.D. Magoulas, N.K. Nousis, M.N. Vrahatis. Training Multilayer Networks with Discrete Activation Functions, in *Proc. INNS–IEEE International Joint Conference on Neural Networks (IJCNN2001)*, Washington DC, 2001, pp 2805-2810.

[289] V.P. Plagianakos, M.N. Vrahatis. Parallel Evolutionary Trained Algorithms for "Hardware-friendly" Neural Networks, *Natural Computing*, 1(2002):307-322.

[290] R.W. Prager, T.D. Harrison, F. Fallside. Boltzmann Machines for Speech Recognition, *Computer Speech and Language*, 1(1)(1986):3-27.

[291] W.L. Price. A Control Random Search Procedure for Global Optimization, in L.C.W. Dixon, G.P. Szego(eds) *Towards Global Optimization 2*, North Holland, Amsterdam, 1978.

[292] F. Provost, T. Fawcett, R. Kohavi. The Case Against Accuracy Estimation for Comparing Induction Algorithms. in *Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, San Francisco CA, 1998, pp 43-48.

[293] J.R. Quinlan. Learning Efficient Classification Procedures and their Application to Chess Endgames. In R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds) *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, San Francisco CA, 1983, pp 463-482.

[294] J.R. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Francisco CA, 1993.

[295] M.O. Rabin. Probabilistic Algorithms, in J.F. Traub (ed) *Algorithms and Complexity: New Directions and Recent Results*, Academic Press Inc., New York, 1976.

[296] S. Rajasekaran, J.H. Reif. Nested Annealing: a Provable Improvement to Simulated Annealing, *Theoretical Computer Science*, 99(1992):157-176.

[297] S. Rampone, Recognition of Splice Junctions on DNA Sequences by BRAIN Learning algorithm, *Bioinformatics*, 14(1998):676-684.

[298] B. Raphael, I.F.C. Smith. Global Search through Sampling Using a PDF, in A. Albrecht, K. Steinhofel (eds) *Stochastic Algorithms: Foundations and Applications*, LNCS, 2827(2003):71-82.

[299] N. Rashevsky. *Mathematical Biophysics*, University of Chicago Press, Chicago, 1938.

[300] A. Razborov and A. Wigderson. $n^{\Omega(logn)}$ Lower Bounds on the Size of Depth 3 Threshold Circuits with AND Gates at the Bottom, *Inf. Proc. Letters*, 45(1993):303-307.

[301] R.D. Reed, R.J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, MIT Press, Cambridge MA, 1999.

[302] C.R. Reeves (ed). *Modern Heuristic Techniques for Combinatorial Problems*, J. Wiley, New York, 1993.

[303] G. Reinelt. Fast Heuristics for Large Geometric Traveling Salesmen Problems, *ORSA Journal on Computing*, 4(1992):206-217.

[304] S. Reiter, G. Sherman. Discrete Optimizing, *Journal of SIAM*, 13(3)(1965):864-889.

[305] M.D. Richard, R.P. Lippmann. Neural Network Classifiers Estimate Bayessian *a-posteriori* Probabilities, *Neural Computation*, 3(4)(1991):461-483.

[306] B. Ricks, D. Ventura. Training a Quantum Neural Network, in S. Thrun, L.K. Saul, B. Schölkopf (eds), *Advances in Neural Information Processing Systems 16 (Neural Information Processing Systems, (NIPS 2003)*, MIT Press, Vancouver , Canada, 2004.

[307] B.D. Ripley. *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambribge UK, 1996.

[308] F. Romeo, A.L. Sangiovanni-Vincentelli. A Theoretical Framework for Simulated Annealing. *Algorithmica*, 6(1991):302-345.

[309] T. Roos, H. Wettig, P. Grunwald, P. Myllymaki, H. Tirri. On Discriminative Bayesian Network Classifiers and Logistic Regression, *Machine Learning* 59(3)(2005):267-296.

[310] F. Rosenblatt. The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain, *Psychological Review*, 65(1958):386-408.

[311] F. Rosenblatt. *Principles of Neurodynamics*, Spartan Books, Washington, 1962.

[312] S.D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis. An Analysis of Several Heuristics for the Travelling Salesman Problem, *Journal of SIAM*, 6(1977):563-581.

[313] D.B. Rubin, R.J.A. Little. *Statistical Analysis with Missing Data*, Wiley, New York, 1987.

[314] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley, B.W. Suter. The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function, *IEEE Transactions on Neural Networks*, 1(4)(1990):296-298.

[315] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group (eds). *Parallel Distributed Processing*, MIT Press, Cambribge MA, 1986.

[316] D. Ruta, B. Gabrys. Application of the Evolutionary Algorithms for Classifier Selection in Multiple Classifier Systems with Majority Voting, in J. Kittler, F. Roli (eds), *Proceedings of the Second International Workshop on Multiple Classifier Systems*, Lecture Notes in Computer Science, Springer-Verlag, London, 2096(2001):399-408.

[317] S. Sahni. General Techniques for Combinatorial Approximation, *Operations Research*, 25(1977):920-936.

[318] S. Sahni, T. Gonzalez. *P*-complete Approximation Problems, *Journal of the ACM*, 23(1976):555-565.

[319] P. Salamon, P. Sibani, R. Frost. *Facts, Conjectures and Improvements for Simulated Annealing*, SIAM, Philadelphia, 2002.

[320] S.M. Sait, H. Youssef. *Iterative Computer Algorithms with Application in Engineering: Solving Combinatorial Optimization Problems*, IEEE Computer Society Press, Los Alamitos, CA, 1999.

[321] S.L. Savage. Some Theoretical Implications of Local Search, *Mathematical Programing*, 10(1976):354-366.

[322] J. Schafer, M/K. Olsen. Multiple Imputation for Multi-variate Missing-Data Problems: A Data Analyst's Pespective, *Multivariate Behavioral Research*, 33(1998):545-571.

[323] C. Schaffer. A Conservation Law for Generalization Performance, in W.W. Cohen, H. Hirsh (eds) *Proceeding of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann, San Francisco CA, 1994, pp 259-265.

[324] A.A. Schäffer, M. Yannakakis. Simple Local Search Problems that are Hard to Solve, *SIAM Journal on Computing*, 20(1991):56-87.

[325] R.E. Schapire, Y. Freund, P. Bartlett, W.S. Lee. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods, *The Annals of Statistics*, 26(5)(1998):1651-1686.

[326] L.M. Schmitt. Theory of Genetic Algorithms, *Theoret. Comput. Sci.*, 259(2001):1-61.

[327] B. Schölkopf, C.J.C. Burges, V. Vapnik. Extracting Support Data for a Given Task, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, AAAI Press, CA, 1995, pp 252-257.

[328] B. Schölkopf, C.J.C. Burges, A.J. Smola (eds). *Advances in Kernel Methods: Support Vector Learning*, MIT Press, Cambridge MA, 1999.

[329] A. Schrijver. *Theory of Linear and Integer Programming*, Wiley-Interscience, New York, 1986.

[330] A. Schrijver. On the History of Combinatorial Optimization (till 1960)in: K. Aardal, G.L. Nemhauser, R. Weismantel, (eds) *Handbooks in Operations Research and Management Science 12: Discrete Optimization*, North-Holland, Netherlands, 2005.

[331] C. Schumacher, M.D. Vose, L.D. Whitley. The No Free Lunch and Problem Description Length, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufmann, San Francisco CA, 2001, pp 565-570.

[332] J. Schürmann. *Pattern Classification: A Unified View of Statistical and Neural Approaches*, H. Wiley & Sons, New York, 1996.

[333] O.G. Selfridge. Pandemonium: A Paradigm for Learning,*Proceedings of Mechanisation of Thought Processes* London, 1958, 513-526.

[334] E. Seneta. *Non-negative Matrices and Markov Chains*, Springer-Verlag, New York, 1981.

[335] I.K. Sethi, A.K. Jain (eds). *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*, North-Holland, Amsterdam, 1991.

[336] R. Setiono. Extracting M-of-N rules from Trained Neural Networks, *IEEE Trans. on Neural Networks*, 11(2000):512-519.

[337] R. Setiono. Generating Concise and Accurate Classification Rules for Breast Cancer Diagnosis, *Artif. Intell. Medicine*, 18(2000):205-217.

[338] R.S. Sexton, B. Alidaee, R.E. Dorsey, J.D. Johnson. Global Optimization for Artificial Neural Networks: A Tabu Search Application, *European Journal of Operational Research*, 106(2-3)(1998):570-584.

[339] J. Shave-Taylor, N. Cristianini. *Kernel Methods for Pattern Analysis*, Cambridge University Press, Cambridge UK, 2004.

[340] J. Shavlik, R.J. Mooney and G. Towell. Symbolic and Neural Learning Programs: An experimental comparison, *Machine Learning*, 6(1991):111-143.

[341] E.H. Shortliffe. *Computer-based Medical Consultations: MYCIN*, Elsevier, New York, 1976.

[342] K.-Y. Siu, V.P. Roychowdhury. On Optimal Threshold Circuits for Multiplication and Related Problems, *SIAM Journal on Discrete Mathematics*, 7(1994):284-292.

[343] F.W. Smith. Design of Multicategory Pattern Classifiers with Two-category Classifier Design Procedures, *IEEE Transactions on Computers*, 18(6)(1969):548-551.

[344] A.J. Smola, P.L. Bartlett, B. Schölkopf, D. Schuurmans (eds). *Advances in Large Margin Classifiers*, MIT Press, Cambridge MA, 2000.

[345] E.D. Sontag. VC-dimension of Neural Networks, in C.M. Bishop (ed) *Neural Networks and Machine Learning*, Springer Verlag, Berlin, 1998, pp 69-95.

[346] G.B. Sorkin. Efficient Simulated Annealing on Fractal Energy Landscapes, *Algorithmica*, 6(1991):367-418.

[347] D.F. Specht. Probabilistic Neural Networks, *Neural Networks*, 3(1)(1990):109-118.

[348] J. Spencer. *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.

[349] A. Stahlberger, M. Riedmiller. Fast Network Pruning and Feature Extraction Using the Unit-OBS Algorithm, in M.C. Mozer, M.I. Jordan, T. Petsche (eds) *Advances in Neural Information Processing Systems*, MIT Press, Cambridge MA, 9(1997):655-661.

[350] J. Stander, B.W. Silverman. Temperature Schedules for Simulated Annealing, *Statistics and Computing*, 4(1994):21-32.

[351] K. Steinhöfel, A. Albrecht, C.K. Wong. Two Simulated Annealing-based Heuristics for the Job Shop Scheduling Problem, *European Journal of Operations Research*, 118(3)(1999):524-548.

[352] K. Steinhöfel, A. Albrecht, C.K. Wong. Fast Parallel Heuristics for the Job Shop Scheduling Problem, *Computers & Operations Research*, 29(2)(2002):151-169.

[353] M.J. Streeter. Two Broad Classes of Functions for Which a No Free Lunch Result Does Not Hold, *Lecture Notes in Computer Science*, Springer-Verlag, 2724(2003):1418-1430.

[354] P.N. Strenski, S. Kirkpatrick. Analysis of Finite Length Annealing Schedules, *Algorithmica*, 6(1991):346-366.

[355] Z. Szewczyk, P. Hajela. Neural Network Approximations in a Simulated Annealing Based Optimal Structural Design, *Structural and Multidisciplinary Optimization*, 5(3)(1993):159-165.

[356] Z. Tang, X. Wang, H. Tamura, M. Ishii. An Algorithm of Supervised Learning for Multilayer Neural Networks, *Neural Computation*, 15(2003):1125-1142.

[357] G. Temple. The General Theory of Relaxations Applied to Linear Systems, *Proc. Roy. Soc. London*, 169(1939):476-500.

[358] S. Theodoridis, K. Koutroumbas. *Pattern Recognition*, Academic Press, New York, 2003.

[359] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, J. Zhang. The MONK's Problems - A Performance Comparison of Different Learning Algorithms, *Technical Report CS-CMU-91-197*, Carnegie Mellon University, 1991.

[360] D.J. Tom. Training Binary Node Feed Forward Neural Networks by Back-propagation of Error, *Electronics Letters*, 26(1990):1745-1746.

[361] C.A. Tovey. Local Improvement on Discrete Structures, in E.H.L. Aarts, J.K. Lenstra (eds), *Local Search in Combinatorial Optimization*, Wiley & Sons, Chichester, 1998.

[362] G. Towell, J. Shavlik. Knowledge-based artificial neural networks, *Artificial Intelligence*, 70(1994):119-165.

[363] G. Tsoumakas, L. Angelis, I. Vlahavas. Selective Fusion of Heterogeneous Classifiers, *Intelligent Data Analysis*, 9(6)(2005), IOS Press.

[364] C. Tung, C. Chou. Pattern Classification Using Tabu Search to Identify the Spatial Distribution of Groundwater Pumbing, *Hydrogeology Journal*, 12(5)(2004):488-496.

[365] J.D. Ullman. NP-complete Scheduling Problems, *Journal of Computer and System Sciences*, 10(1975):384-393.

[366] R.J.M. Vaessens, E.H.L. Aarts, J.K. Lenstra. Job Shop Scheduling by Local Search, *INFORMS Journal on Computing* , 8(1996):302-317.

[367] L.G. Valiant. A Theory of the Learnable, *Communications of the ACM*, 27(11)(1984):1134-1142.

[368] T. Van Gestel, J.A. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, J. Vandewalle. Benchmarking Least Squares Support Vector Machine Classifiers, *Machine Learning*, 54(1)(2004):5-32.

[369] P.J.M. Van Laarhoven. *Theoretical and Computational Aspects of Simulated Annealing*, PhD Thesis, Erasmus University Rotterdam, 1988.

[370] V. Vapnik, A.Y. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Theory of Probability and Its Applications*, 16(2)(1971):264-280.

[371] V. Vapnik. *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

[372] V.V. Vazirani. *Approximation Algorithms*, Springer-Verlag, Berlin Heidelberg, 2003.

[373] H. Vollmer. Some Aspects of the Computational Power of Boolean Circuits of Small Depth, Habilitationsschrift, University Würzburg, 1999.

[374] A. Wald. *Statistical Decision Functions*, Wiley, New York, 1950.

[375] M. Wallace, N. Tsapatsoulis, S.D. Kollias. Intelligent Initialization of Resource Allocating RBF networks, Neural Networks, 18(2)(2005):117-122.

[376] S. Warshall. A Theorem on Boolean Matrices, *Journal of the ACM*, 9(1)(1962):11-12.

[377] A. Webb. *Statistical Pattern Recognition*, John Wiley & Sons, Chichester UK, 2004.

[378] A. Weigend, B.A. Huberman, D. Rumelhart. Predicting the Future: a Connectionist Approach, *International Journal of Neural Systems*, 3(1990):193-210.

[379] A. Weigend, D. Rumelhart, B.A. Huberman. Generalization by Weight Elimination with Application to Forecasting, in R.P. Lippmann, J.E. Moody, D.S. Touretzky (eds) *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Mateo CA, 3(1991):875-882.

[380] P. Weiner, S.L. Savage, A. Bagghi. Neighborhood Search Algorithms for Guaranteeing Optimal Travelling Salesman Tours Must be Inefficient, *Journal of Computer and System Sciences*, 12(1976):25-35.

[381] R.S. Wenocur, R.M. Dudley. Some Special Vapnik-chervonenkis Classes, *Discrete Mathematics*, 33(1981):313-318.

[382] P.J. Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, Wiley, New York, 1994.

[383] L.K. Westin. Missing Data and the Preprocessing Perceptron, UMINF-04.02, Department of Computing Science, Umeå University, 2004.

[384] D. Whitley, J.P. Watson, A. Howe, L. Barbulescu. Testing, Evaluation and Performance of Optimization and Learning Systems, *Technical Report: The GENITOR Research Group in Genetic Algorithms and Evolutionary Computation*, Colorado State University, Colorado, 2002.

[385] P.H. Winston. *Artificial Intelligence*, Addison Wesley, Reading MA, 1984.

[386] W.H. Wolberg, O.L. Mangasarian. Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology, *Proc. of the National Academy of Sciences, USA*, 87(1990):9193-9196.

[387] M.T. Wolfinger, W.A. Svrcek-Seiler, Ch. Flamm, I.L. Hofacker, P.F. Stadler. Exact Folding Dynamics of RNA Secondary Structures, *J. Physics A-Math. Gen.*, 37(2004):4731-4741.

[388] D.H. Wolpert (ed). *The Mathematics of Generalization*, Addison-Wesley, Reading MA, 1995.

[389] D.H. Wolpert. The Lack of a Prior Distinctions Between Learning Algorithms, *Neural Computation*, 8(1996):1341-1390.

[390] D.H. Wolpert. On Bias Plus Variance, *Neural Computation*, 9(1996):1248-1271.

[391] D.H. Wolpert and W.G. Macready. No Free Lunch Theorems for Optimization, *IEEE Trans. on Evolutionary Computation*, 1(1997):67-82.

[392] D.H. Wolpert. Any Two Learning Algorithms are (Almost) Exactly Identical, http://ic.arc.nasa.gov/publications/pubFull.php, 2000.

[393] D.H. Wolpert. The Supervised Learning No-free-lunch Theorems, *in Proc. 6th Online World Conference on Soft Computing in Industrial Applications*, 2001.

[394] J.R. Woodward, J.R. Neil. No Free Lunch, Program Induction and Combinatorial Problems, *6th European Conference on Genetic Programming (EuroGP 2003)*, Essex UK, 2003.

[395] A. Yamazaki, T.B. Ludermir. Neural Networks Training with Globl Optimization Techniques, *International Journal of Neural Systems*, 13(2)(2003):77-86.

[396] Y. Yang, X. Wu. Parameter Tuning for Induction-Algorithm-Oriented Feature Elimination, *IEEE Intelligent Systems*, 19(2)(2004):40-49.

[397] M. Yannakakis. On the Approximation of Maximum Satisfiability, *Journal of Algorithms*, 3(1994):475-502.

[398] M. Yannakakis. Computational Complexity, in E.H.L. Aarts, J.K. Lenstra (eds), *Local Search in Combinatorial Optimization*, Wiley & Sons, Chichester, 1998.

[399] X. Yao. Evolving Artificial Neural Networks, *Proceedings of the IEEE*, 87(9)(1999):1423-1447.

[400] T.Y. Young, T.W. Calvert. *Classification, Estimation and Pattern Recognition*, Elsevier, New York, 1974.

[401] L. Zadeh. Fuzzy Sets, *Information and Control*, 8(3)(1965):338-353.

[402] H. Zhu, R. Rohwer. No Free Lunch for Cross-validation, *Neural Computation*, 8(7)(1996):1421-1426.