SENSITIVITY ANALYSIS OF MULTILAYER NEURAL NETWORKS

ANDRIES PETRUS ENGELBRECHT

Dissertation submitted in partial fulfilment of the requirements

for the degree of

Doctor of Philosophy

at

The University of Stellenbosch

Promotor: Prof I Cloete

December 1999

Declaration

I, the undersigned, hereby declare that the work contained in this dissertation is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

ANDRIES PETRUS ENGELBRECHT

December 1999

To my parents,
without whose loving support
this would not have happened.

Abstract

The application of artificial neural networks to solve classification and function approximation problems is no longer an art. Using a neural network does not simply imply the presentation of a data set to the network and relying on the so-called "black-box" to produce - hopefully accurate - results. Rigorous mathematical analysis now provides a much better understanding of what is going on inside the "black-box". The knowledge gained from these mathematical studies allows the development of specialized tools to increase performance, robustness and efficiency.

This thesis proposes that sensitivity analysis of the neural network output function be used to learn more about the inner working of multilayer feedforward neural networks. New sensitivity analysis techniques are developed to probe the knowledge embedded in the weights of networks, and to use this knowledge within specialized sensitivity analysis algorithms to improve generalization performance, to reduce learning and model complexity, and to improve convergence performance.

A general mathematical model is developed which uses first order derivatives of the neural network output function with respect to the network parameters to quantify the effect small perturbations to these network parameters have on the output of the network. This sensitivity analysis model is then used to develop techniques to locate and visualize decision boundaries, and to determine which boundaries are implemented by which hidden units. The decision boundary detection algorithm is then used to develop an active learning algorithm for classification problems which trains only on patterns close to decision boundaries. Patterns that convey little information about the position of boundaries are therefore not used for training. An incremental learning algorithm for function approximation problems is also developed to

incrementally grow the training set from a candidate set by adding to the training set those patterns that convey the most information about the function to be approximated. The sensitivity of the network output to small perturbations of the input pattern is used as measure of pattern informativeness. Sensitivity analysis is also used to develop a network pruning algorithm to remove irrelevant network parameters. The significance of a parameter is quantified as the influence small perturbations on that parameter have on the network output. Variance analysis is employed as pruning heuristic to decide if a parameter should be removed or not.

Elaborate experimental evidence is provided to illustrate how each one of the developed sensitivity analysis techniques addresses the objectives of improved performance, robustness and efficiency. These results show that the different models successfully utilize the neural network learner's current knowledge to obtain optimal architectures and to make optimal use of the available training data.

Opsomming

Die toepassing van kunsmatige neurale netwerke om klassifikasie- en funksiebenaderingsprobleme op te los, is nie meer 'n kuns nie. Die gebruik van 'n neurale netwerk impliseer nie meer bloot die toepassing van 'n data stel op die netwerk, en die verwagting dat die "swart boks" - hoopvol akkurate - resultate lewer nie. Omvattende wiskundige analises verskaf nou 'n baie beter begrip van wat binne die "swart boks" aangaan. Die kennis wat van hierdie wiskundige analises gewin is, laat die ontwikkeling van gespesialiseerde hulpmiddels toe om prestasie, robuustheid en effektiwiteit te verbeter.

Hierdie tesis stel voor dat sensitiwiteitsanalise van die neurale netwerk afvoer funksie aangewend word om meer oor die inner werking van multi-vlak vorentoe-voer neurale netwerke te leer. Nuwe sensitiwiteitsanalise tegnieke word ontwikkel om die kennis vervat in die gewigte van netwerke te ondersoek, en om hierdie kennis aan te wend binne gespesialiseerde sensitiwiteitsanalise algoritmes om sodoende veralgemeningseienskappe te verbeter, om die kompleksiteit van leer en model kompleksiteit te verminder, en om konvergensie eienskappe te verbeter.

'n Algemene wiskundige model is ontwikkel wat gebruik maak van die eerste orde afgeleides van die neurale netwerk afvoer funksie met betrekking tot netwerk parameters om die effek van klein versteurings aan hierdie netwerk parameters op die afvoer van die netwerk te kwantifiseer. Hierdie sensitiwiteitsanalise model word dan gebruik om tegnieke te ontwikkel om besluitnemingsgrense op te spoor en te visualiseer, en om te bepaal watter besluitnemingsgrense word deur watter versteekte eenhede geïmplementeer. Die algoritme om besluitnemingsgrense op te spoor word dan aangewend om 'n aktiewe-leer algoritme vir klassifikasie probleme te ontwikkel, wat leer deur gebruik te maak van slegs daardie patrone wat naby

besluitnemingsgrense lê. Gevolglik word patrone wat min inligting bevat in verband met die ligging van besluitnemingsgrense nie vir leer aangewend nie. 'n Inkrementele leer algoritme is ook ontwikkel vir funksiebenaderingsprobleme waarin die leerversameling inkrementeel vanuit 'n kandidaat leerversameling gegroei word deur daardie patrone by te voeg wat die meeste inligting vervat oor die funksie wat benader word. Die sensitiwiteit van die netwerk afvoer tot versteurings in die toevoer patroon word gebruik as 'n maatstaf van die informatiwiteit van daardie patroon. Sensitiwiteitsanalise is ook gebruik om 'n algoritme te ontwikkel wat irrelevante parameters van die netwerk snoei. Die belangrikheid van 'n parameter word gekwantifiseer as die invloed wat klein versteurings in daardie parameter het op die afvoer van die netwerk. Variansie analise word gebruik as heuristiek om te besluit of 'n parameter gesnoei kan word al dan nie.

Omvattende eksperimentele bewyse word verskaf om te illustreer hoe elkeen van die sensitiwiteitsanalise tegnieke wat in hierdie tesis ontwikkel is, die doelwitte van verbeterde prestasie, robuustheid en effektiwiteit adresseer. Hierdie resultate toon aan dat die onderskeie modelle suksesvol gebruik maak van die neurale netwerk se huidige kennis om optimale argitekture op te stel, en om optimaal van die beskikbare leerdata gebruik te maak.

Acknowledgements

The following people were instrumental to the success of this work:

- My parents, for giving me the much needed support and guidance through my educational life to bring me to this stage of academic maturity.
- My supervisor, Prof Ian Cloete, for his support, advice and encouragement.
- Lizelle Fletcher, Department of Statistics, University of South Africa, for her input on the statistical aspects of this dissertation.
- All the people who spiritually supported me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"Keep it simple:
as simple as possible,
but no simpler"
- A Einstein.

The past decade has shown artificial neural networks (NN) to be very powerful modeling and analysis tools. Much research effort has been expended to better understand why NNs are so successful - to probe the so-called "black box". We now know that using a NN to solve a problem does not just involve pushing data into the network and expecting good results at the other end. There are more to using NNs...

This thesis shows how sensitivity analysis of the neural network output can be utilized to analyze the knowledge embedded in the network weights in order to develop efficient sensitivity analysis algorithms to improve learning performance. Having the support of rigorous mathematical analysis, the neural network sensitivity analysis model developed in this thesis addresses the following important performance aspects of neural networks:

- **Generalization performance:** The accuracy achieved on a set of data points not seen during training is one of the most important measures of performance, referred to as generalization. Although the objective of learning algorithms is to minimize the training error, the objective of learning should also be to minimize the generalization

error. If a NN has too many free parameters (i.e. weights), and is overtrained, the network may overfit the training set, causing memorization of the training data. Such memorization of training data leads to bad generalization.

- **Complexity:** The computational complexity of a learning algorithm is influenced by the optimization method used and the architecture of the network. The number of learning calculations is directly dependent on the number of weights in the network.
- **Convergence:** The convergence characteristics of a learning algorithm refer to the ability to find an accurate approximation to the function that maps model inputs to outputs. That is, the ability to converge to "good" local minima.
- **Comprehensibility:** Comprehensibility refers to a general understanding of the operation of the NN and the data being modeled. Here we specifically refer to the ability to extract hidden features, or knowledge, about the data; or some explanation facility to present the numerically encoded knowledge in a symbolic way.

Much research concentrated on developing techniques to address these aspects individually, i.e. the optimal setting of initial weights, optimal learning rates and momentum, finding optimal NN architectures, sophisticated optimization techniques, adaptive activation functions, noise injection and ensemble networks. Optimal weight settings and optimal learning parameter settings target the improvement of generalization and training time. The construction of optimal architectures improves generalization performance and may reduce complexity depending on the complexity of the architecture selection algorithm. Optimization techniques improve convergence and generalization, but often at the expense of increased complexity. Adaptive activation functions improve training time and generalization, while ensemble networks improve generalization at the expense of increased complexity. Comprehensibility is facilitated through algorithms that extract symbolic rules from trained networks, and algorithms that visualize classification boundaries.

In this thesis the performance aspects discussed above are addressed through the development of decision boundary visualization, active learning and network pruning algorithms which make use of sensitivity information of the NN output to small perturbations in network parameters. The next section outlines the objectives of this thesis and shows how the

sensitivity analysis models developed in this thesis address these performance issues.

1.1 Thesis Objectives

The main objective of this thesis is to develop a sensitivity analysis model for multilayer feedforward neural networks, and to design different sensitivity analysis tools based on this model. The thesis has as sub objectives that the developed tools should improve generalization, reduce complexity, improve convergence and facilitate comprehensibility of the network and the problem domain. As vehicle to achieve these goals, the thesis concentrates on one neural network optimization technique, i.e. gradient descent, and considers only stochastic learning where weight adjustments are made after each pattern presentation.

The sensitivity analysis model is based on the analysis of the influence that small perturbations of NN parameters (which can be input units, hidden unit activations and weights) have on the output of the NN. First order derivatives of the NN output function with respect to network parameters are used to quantify this influence perturbations have on the output. Based on these derivatives, the following sensitivity analysis tools are developed:

- An algorithm to **visualize the position of decision boundaries**, which helps to better understand the functioning of the NN and the problem domain.
- A **selective learning algorithm** for classification problems, which uses the decision boundary algorithm to dynamically select training patterns near decision boundaries. The selective learning algorithm addresses generalization, complexity and convergence.
- An **incremental learning algorithm** for function approximation problems, which addresses generalization, complexity and convergence.
- A **pruning algorithm** to optimize NN architectures in order to reduce complexity and to improve generalization.

It is illustrated in the chapters how these tools achieve the sub objectives.

1.2 Contribution

The study of NN output sensitivity analysis led to the development of a few new algorithms, each supported by a mathematical model. The specific contributions of this thesis are:

- A comparison of the characteristic principles and complexity of the two approaches to NN sensitivity analysis, i.e. (1) with regard to the objective function and (2) with regard to the NN output function. This comparison led to the conclusion that these approaches are conceptually the same, while NN output sensitivity analysis is less complex and does not rely on simplifying assumptions.

- An algorithm to visualize decision boundaries, that can be used to locate the boundaries formed in input space and to determine which boundary is implemented by which hidden unit. The algorithm can also be used to detect irrelevant input and hidden units. NN rule extraction algorithms can use this algorithm to extract thresholds for continuous valued parameters in rule clauses.

- A selective learning algorithm for classification problems that prunes patterns from a candidate training set. The selective learning algorithm selects for training only those patterns that lie closest to decision boundaries. For this purpose the decision boundary algorithm is used to find all patterns near boundaries.

- An incremental learning algorithm for function approximation problems which dynamically grows the training set by adding to it only those patterns that have the highest influence on the NN output.

- A pruning algorithm which can be used to prune input units, hidden units and weights. A new statistical pruning heuristic is developed based on variance analysis.

- Mathematical derivations of the sensitivity analysis equations for feedforward, functional link and product unit neural networks.

In addition to the contributions listed above, appendix C contains an algorithm which dynamically adapts the sigmoid activation functions in the hidden and output layers. This algorithm, referred to as γ-learning, has as objective to improve generalization performance,

and as such fits into one of the objectives of this thesis to produce techniques to improve generalization. To this extend, the appendix analyzes the effects that the scaling of output values has on training time and show how generalization is improved by learning the shape of the sigmoid activation function.

1.3 Outline

The thesis is organized as follows. The next section presents an overview of learning using multilayer NNs, with the objective to introduce the assumptions used throughout the thesis.

Chapter 2 presents an overview of sensitivity analysis. It is shown how sensitivity analysis originates from perturbation theory. Different uses of sensitivity analysis in NNs are discussed in section 2.2, and a comparison between the two types of NN sensitivity analysis, i.e. with respect to the objective function and with respect to the NN output function, is presented in section 2.4.3. Section 2.4.3 shows that these two approaches to sensitivity analysis are conceptually the same, but that NN output function sensitivity analysis is less complex. Section 2.5 illustrates experimentally that the output sensitivity analysis equations accurately approximate the true derivatives of the function that maps inputs to outputs. The sensitivity analysis applications covered in this thesis are introduced in section 2.6. Section 2.7 shows mathematically how sensitivity analysis can be applied to different NN types, including feedforward, functional link and product unit NNs.

Algorithms to visualize decision boundaries are developed in chapter 3. Definitions of decision boundaries are given in section 3.2, supported by a mathematical explanation. Section 3.3 illustrates, using artificial and real-world problems, how the decision boundary algorithms can be used to locate and visualize boundaries.

Chapter 4 develops two new active learning algorithms based on sensitivity analysis. The sensitivity analysis selective learning algorithm is developed in section 4.3, while section 4.4 presents the sensitivity analysis incremental learning algorithm. For both algorithms a mathematical model is presented, the computational complexity is analyzed and results of their

application are reported in comparison with standard fixed set learning. Before the presentation of these algorithms, an overview of active learning is presented in section 4.1.1, and a general mathematical model for active learning is developed in section 4.2. A definition for pattern informativeness - the fundamental principal of active learning algorithms developed in this thesis - is given in section 4.2.3.

Neural network architecture selection is considered in chapter 5. An overview of architecture selection methods is presented in section 5.1, and a summary of pruning algorithms is given in section 5.1.1. A general pruning algorithm is presented in section 5.2. A pruning algorithm that uses output sensitivity analysis is developed in section 5.2.3. A definition of parameter significance is given in section 5.3.2, and a new statistical test is designed in section 5.3.3. Pruning heuristics based on parameter significance and the statistical test are presented in section 5.3.4. Sensitivity analysis pruning algorithms are developed in section 5.3.5. Results of the application of the sensitivity analysis pruning algorithm are presented in section 5.4.

Conclusions and topics for future research are discussed in chapter 6.

Appendix A summarizes the symbols used throughout the thesis. Definitions of key terms used and defined in the thesis are listed in appendix B. An automatic scaling learning algorithm is presented in appendix C. The effects of scaling of output values are discussed, the new gamma learning algorithm is developed and results are presented.

Appendix D contains the complete learning equations for feedforward NNs trained using gradient descent, online learning. The derivations of the NN output sensitivity analysis equations for feedforward NNs and product unit NNs, as well as for objective function sensitivity analysis, are presented in appendix E.

A list of publications based on the research presented in this thesis is given in appendix F.

1.4 Neural Network Learning

Various multilayer NN types have been developed, for example feedforward NNs, recurrent neural networks (RNN), functional link neural networks (FLNN), and product unit neural networks (PUNN). This thesis concentrates on **feedforward NNs** where **gradient descent**

is used to compute the error between the target and network output values. These calculated errors are then backpropagated through all the layers of the network to adjust the network weights [Rumelhart *et al* 1986, Zurada 1992b]. There are no feedback connections to previous layers. In a RNN, feedback connections are used as a mechanism to model the temporal characteristics of the problem being learned. Architecture specific RNNs have been developed that simply duplicate layers to store the state of these layers at previous time steps. For this purpose the hidden layer and/or the output layer can be duplicated for any number of time steps [Ludik 1995a]. In a functional link NN the input layer is expanded to a layer of functional units, where a functional unit is a higher-order combination of linear input units [Ghosh *et al* 1992, Hussain *et al* 1997, Zurada 1992b]. In a product unit NN, product units are used in the hidden layers to compute the netto input value to hidden units, instead of summation units [Durbin *et al* 1989, Ghosh *et al* 1992, Janson *et al* 1993, Leerink *et al* 1995].

While sensitivity analysis can be applied to all these NN types, as illustrated mathematically in section 2.7, this thesis applies the developed sensitivity analysis techniques to feedforward NNs only, since the main objective of this thesis is to introduce new sensitivity analysis techniques (for a specific NN type) and to illustrate the application of these techniques (to that NN type) to improve learning performance. Further research, beyond this thesis, will investigate the application of the sensitivity analysis techniques developed in this thesis to other NN types. Thus, for the purpose of this thesis a three layer feedforward NN architecture with one input layer, one hidden layer and one output layer is assumed. With reference to figure A.1, a bias unit is added to the input and hidden layers. Generalization to more than one hidden layer is straightforward as illustrated in appendix E. Although sensitivity analysis can be applied to any differentiable activation function, this thesis assumes sigmoid activation functions which are differentiable, monotonic increasing functions. In the light of these assumptions, what follows next is an explanation of learning and generalization.

Consider a finite set of input-target pairs $D = \{d^{(p)} = (\vec{z}^{(p)}, \vec{t}^{(p)}) | p = 1, \cdots, P\}$ sampled from a stationary density $\Omega(D)$, with $z_i^{(p)}, t_k^{(p)} \in \mathbb{R}$ for $i = 1, \cdots, I$ and $k = 1, \cdots, K$; $z_i^{(p)}$ is the value of input unit z_i and $t_k^{(p)}$ is the target value of output unit o_k for pattern p. According to the signal-plus-noise model

$$\vec{t}^{(p)} = \mu(\vec{z}^{(p)}) + \vec{\zeta}^{(p)} \tag{1.1}$$

where $\mu(\vec{z})$ is the unknown function, the input values $z_i^{(p)}$ are sampled with probability density $\omega(\vec{z})$, and the $\vec{\zeta_k}^{(p)}$ are independent, identically distributed noise sampled with density $\phi(\vec{\zeta})$, having zero mean.

The objective of learning is then to approximate the unknown function $\mu(\vec{z})$ using the information contained in the finite data set D. For NN learning this is achieved by dividing the set D randomly into a training set D_T and a test set D_G. The approximation to $\mu(\vec{z})$ is found from the training set D_T, and the generalization accuracy is estimated from the test set D_G.

Since prior knowledge about $\Omega(D)$ is usually not known, a nonparametric regression approach is used by the NN learner to search through its hypothesis space $\mathcal{H}$ for a function $\mathcal{F}_{NN}(D_T; W)$ which gives a good estimation of the unknown function $\mu(\vec{z})$, where $\mathcal{F}_{NN}(D_T; W) \in \mathcal{H}$. For multilayer NNs, the hypothesis space consists of all functions realizable from the given network architecture as described by the weight vector W. The function $\mathcal{F}_{NN} : \mathbb{R}^I \longrightarrow \mathbb{R}^K$ is found which minimizes the empirical error

$$\mathcal{E}_T(D_T; W) = \frac{1}{P_T} \sum_{p=1}^{P_T} (\mathcal{F}_{NN}(\vec{z}^{(p)}, W) - \vec{t}^{(p)})^2 \tag{1.2}$$

where P_T is the total number of training patterns. The hope is that a small empirical (training) error will also give a small true error, or generalization error, defined as

$$\mathcal{E}_G(\Omega; W) = \int (\mathcal{F}_{NN}(\vec{z}, W) - \vec{t})^2 d\Omega(\vec{z}, \vec{t}) \tag{1.3}$$

For the purpose of NN learning, the empirical error in equation (1.2) is referred to as the objective function to be optimized by the optimization method (e.g. gradient descent, scaled conjugate gradient, simulated annealing, etc.). For the purposes of this thesis gradient descent is used to optimize weights.

In the spirit of this explanation of learning, the sensitivity analysis techniques developed in this thesis have as objective to decrease the generalization error $\mathcal{E}_G$ through manipulation of the way in which training patterns in D_T are presented for learning, and through architecture manipulation.

The notation and symbols used in this thesis are summarized in appendix A, and introduced throughout the thesis when needed.

Chapter 2

Sensitivity Analysis

$$\mathcal{P}(\theta + \Delta\theta) - \mathcal{P}(\theta) \approx \mathcal{P}'(\theta)\Delta\theta$$

This chapter presents a short overview of NN sensitivity analysis techniques. The chapter includes a comparison of objective function and NN output sensitivity analysis, and an illustration of the applicability of NN output sensitivity analysis to different NN types.

2.1 Introduction

Sensitivity analysis has its birth from perturbation analysis, which is a study of the behavior of a function in a small region about a point, or more than one point. Perturbation analysis allows the study of the characteristics of a function (or performance measure) under small perturbations of the function's parameters [Ho 1987, Ho 1988, Holtzman 1992, Zurada *et al* 1997]. In perturbation analysis we are interested in evaluating the disturbance in the function's response to small perturbations in its parameters. Assuming that the performance function is differentiable, the relationship between the perturbed response of this function and parameter perturbations is expressed by a Taylor expansion of that function. For example, for a one dimensional performance function $\mathcal{P}$,

$$\mathcal{P}(\theta + \Delta\theta) = \mathcal{P}(\theta) + \frac{\Delta\theta}{1!}\mathcal{P}'(\theta) + \frac{\Delta\theta^2}{2!}\mathcal{P}''(\theta) + \cdots \tag{2.1}$$

where θ is a parameter of the function, and $\Delta\theta$ is a small perturbation of θ. The Taylor expansion shows that the derivatives of the function with respect to the perturbed parameter encapsulate the characteristics of that function under the $\Delta\theta$ perturbations.

Sensitivity analysis is a technique to calculate these derivatives, and to use the derivatives to draw conclusions about the characteristics of the function.

In terms of neural networks (NN), the performance measure can be expressed as either the objective (error) function to be optimized (as in equation (1.2)), or the NN output function $\mathcal{F}_{NN}(D_T; W)$. The parameters of both these NN performance measures are the weights, input unit activations and hidden unit activations. Sensitivity analysis of a NN therefore refers to the study of the behavior of the objective function or the NN output function with respect to small perturbations in the weights, input and/or hidden unit activations, as derived from the corresponding function derivatives. Sensitivity analysis of any of the before mentioned performance functions, i.e. the objective function or the NN output function, requires the performance function to be differentiable, which is the case if differentiable activation functions are used in the hidden and output layers.

The main objective of this chapter is to give a short overview of NN sensitivity analysis techniques. In particular, sensitivity analysis of the objective function $\mathcal{E}_T$ with respect to NN parameters and sensitivity analysis of the NN output function $\mathcal{F}_{NN}$ with respect to parameters are discussed and compared. An overview of the uses of sensitivity analysis in NNs is presented in section 2.2. Section 2.3 presents, in general, a theoretical discussion of the origin of sensitivity analysis. Neural network sensitivity analysis techniques are discussed in section 2.4, with a comparison of the two main approaches to NN sensitivity analysis for network pruning. The proposed uses of NN sensitivity analysis are introduced in section 2.6, while section 2.5 presents experimental results to illustrate the accuracy of the NN output sensitivity analysis approach in approximating the derivative of the underlying function. The chapter concludes with an illustration that the output sensitivity analysis approach can be applied easily to NNs of different types.

2.2 Sensitivity Analysis in Neural Networks

Parameter sensitivity information have been used for different purposes in neural networks:

- **Optimization:** The calculation of the gradient of a function forms an important part of optimization. One of the first uses of sensitivity analysis is therefore in optimization problems [Blanning 1974, Cao 1985, Holtzman 1992, Kibsgaard 1992]. In NNs, derivatives of the objective function with respect to the weights are computed to locate minima by driving these derivatives to zero (refer to equations (D.14) and (D.21)) [Rumelhart *et al* 1986]. Second order derivatives have also been used to develop more sophisticated optimization techniques to improve convergence and accuracy [Battiti 1992, Becker *et al* 1988, Møller 1993]. Koda uses stochastic sensitivity analysis to compute the gradient for time-dependent networks such as recurrent neural networks (RNN) [Koda 1995, Koda 1997].

- **Robustness and stability:** Neural network robustness and stability analysis is the study of the conditions under which the outcome of the NN changes. This study is important for hardware implementation of NNs to ensure stable networks that are not adversely affected by weight, external input and activation function perturbations [Alippi *et al* 1995, Oh *et al* 1995, Stevenson *et al* 1990, Wang *et al* 1994]. Instead of using derivatives to compute the gradient of the objective function with respect to the weights, Jabri and Flower use differences to approximate the gradient, thereby significantly reducing hardware complexity [Jabri *et al* 1991].

- **Generalization:** Fu and Chen state that good generalization must imply insensitivity to small perturbations in inputs [Fu *et al* 1993]. They derive equations to compute the sensitivity of the NN output vector to changes in input values, and show under what conditions global NN sensitivity can be reduced. For example, using small slopes for the sigmoid activation function, using as small as possible weights, reducing the number of units, and ensuring activation levels close to 0 or 1 (for appropriate activation functions such as the sigmoid function) will reduce network sensitivity. Choi and Choi derive a NN sensitivity norm which expresses the sensitivity of the NN output with respect to input perturbations [Choi *et al* 1992]. This NN sensitivity norm is then used to select

from sets of optimal weights the weight set with lowest NN sensitivity, which results in the best generalization.

- **Measure of nonlinearity**: Lamers and Kok use the variance of the sensitivity of the NN output to input parameter perturbations as a measure of the nonlinearity of the data set [Lamers *et al* 1998]. This measure of nonlinearity is then used to show that the higher the variance of noise injected to output values, the more the problem is linearized.

- **Causal inference:** Sensitivity analysis has been used to assess the significance of model inputs. Engelbrecht, Cloete and Zurada use exact derivative calculations to compute the significance of each input parameter [Engelbrecht *et al* 1995b]. Inputs with high significance values have a high influence on the NN output. Goh derived a similar method using differences to approximate the gradient of the NN output function with respect to inputs [Goh 1993]. Real-world applications of causal inferencing using sensitivity analysis include the work of Modai *et al* where sensitivity analysis is used to find those psychiatric parameters that have the highest influence on the short-term outcome of psychiatric disorders [Modai *et al* 1995], and Guo and Uhrig who use sensitivity analysis to find those parameters that have the highest influence in the loss in electricity production in a nuclear power plant [Guo *et al* 1992]. In a Bayesian context, Laskey derives equations to compute parameter significances for Bayesian neural networks [Laskey 1995].

- **Selective learning**: Hunt and Deller use weight perturbation analysis to determine the influence each pattern has on weight changes during training [Hunt *et al* 1995]. Only patterns that exhibit a high influence on weight changes are used for training. Chapter 4 presents new active learning models based on sensitivity analysis (also introduced in [Engelbrecht *et al* 1998a]). These models use a measure of pattern informativeness to dynamically select patterns during training.

- **Decision boundary visualization:** Goh uses an approximation to the derivative of the NN output function with respect to inputs, using differences, to graphically visualize decision boundaries [Goh 1993]. Chapter 3 shows how exact derivative calculations can be used to locate and visualize decision boundaries (also introduced and

applied in [Engelbrecht *et al* 1998a, Engelbrecht *et al* 1998b, Engelbrecht *et al* 1999a, Engelbrecht 1999c, Viktor *et al* 1998a]). Viktor uses the decision boundary algorithm developed in chapter 3 to improve the accuracy of rules extracted from trained NNs in a cooperative learning environment [Viktor 1998b], while Engelbrecht and Viktor show the same results for individual learners [Engelbrecht *et al* 1999a].

- **Pruning:** Sensitivity analysis has been applied extensively to NN pruning. Approaches range from pruning weights, inputs and/or hidden units using approximations to compute derivatives or using exact derivative calculations. One technique is to compute the sensitivity of the objective function with respect to NN parameters [Burrascano 1993, Cibas *et al* 1996, Gorodkin *et al* 1993b, Hassibi *et al* 1993, Hassibi *et al* 1994, Karnin 1990, Le Cun 1990, Moody *et al* 1995, Mozer *et al* 1989, Pedersen *et al* 1996, Schittenkopf *et al* 1997]. Another method of sensitivity analysis pruning is to compute the sensitivity of the NN output function to parameter perturbations [Czernichow 1996, Cloete *et al* 1994c, Dorizzi *et al* 1996, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Engelbrecht *et al* 1999e, Fletcher *et al* 1998, Takenaga *et al* 1991, Viktor *et al* 1995, Zurada *et al* 1994, Zurada *et al* 1997]. Section 2.4 elaborates on these pruning methods, while chapter 5 presents a pruning algorithm based on output sensitivity analysis.

- **Learning derivatives:** Basson and Engelbrecht develops a new learning algorithm for feedforwards NNs that also learns the first-order derivatives of the NN output with respect to each input unit while learning the underlying function [Basson *et al* 1999]. The NN consists of two parts, one representing the learned function, and the other representing the derivatives of the learned function. Concepts from sensitivity theory are used to create a training set for the training of the derivative part of the NN using gradient descent.

2.3 Theoretical Development

This section presents a formal definition of perturbation analysis, and shows how sensitivity analysis follows from a Taylor expansion of the performance measure around the parameter

of interest:

Definition 2.1 Perturbation Analysis: *Let $\mathcal{P}$ be a performance measure of a system, and θ a parameter of this system. Without loss of generality, assume θ is scalar. Then, from a Taylor expansion of $\mathcal{P}$ around θ, the change in performance due to perturbation $\Delta\theta$ of θ is expressed as [Ho 1987, Ho 1988]*

$$\mathcal{P}(\theta + \Delta\theta) = \mathcal{P}(\theta) + \frac{\Delta\theta}{1!}\mathcal{P}'(\theta) + \frac{\Delta\theta^2}{2!}\mathcal{P}''(\theta) + \frac{\Delta\theta^3}{3!}\mathcal{P}'''(\theta) + \cdots \tag{2.2}$$

Therefore, perturbation analysis is the study of the performance of the system $\mathcal{P}$ with respect to a small perturbation $\Delta\theta$ of parameter θ. The sum $\frac{\Delta\theta}{1!}\mathcal{P}'(\theta) + \frac{\Delta\theta^2}{2!}\mathcal{P}''(\theta) + \cdots$ is the change in performance $\mathcal{P}(\theta)$ due to the perturbation $\Delta\theta$. Ideally, $\frac{\Delta\theta}{1!}\mathcal{P}'(\theta) + \frac{\Delta\theta^2}{2!}\mathcal{P}''(\theta) + \cdots \to 0$ when $\Delta\theta \to 0$.

Equation (2.2) shows that the derivatives play a very important role in determining the influence of parameter perturbations on the output of the performance function. This chapter investigates how the derivatives can be used to quantify the response of the system to parameter perturbations, and how these derivatives can be calculated. This study of how the derivatives influence the performance function is referred to as **sensitivity analysis**. Algorithms that use derivatives to analyze models are referred to as **sensitivity analysis techniques**.

Sensitivity analysis techniques differ mainly in the performance measure used, the order of the derivatives that are considered, whether the analysis is in continuous time or for discrete time intervals, and the way in which the derivatives are calculated.

Due to computational considerations, sensitivity analysis is based on approximations of equation (2.2), usually first-order or second-order approximations. For example, a first-order approximation yields

$$\mathcal{P}(\theta + \Delta\theta) \approx \mathcal{P}(\theta) + \mathcal{P}'(\theta)\Delta\theta \tag{2.3}$$

Sensitivity analysis that uses a second-order approximation is more accurate, but also more complex and time consuming than a first-order approximation due to the calculation of

the Hessian matrix [Bishop 1992, Buntine *et al* 1994, Gorodkin *et al* 1993b, Le Cun 1990, Hassibi *et al* 1994].

Usually, sensitivity analysis is done at discrete time intervals for that time interval only. Sensitivity analysis can also be performed for continuous time models, referred to as stochastic sensitivity analysis [Cao 1985, Koda 1995, Koda 1997]. Koda derives stochastic sensitivity analysis formulae for RNNs [Koda 1995, Koda 1997]. He derives a sensitivity density function (to be used to estimate the gradient for learning purposes) viewed as the sensitivity of the state of the NN at an unit at time T with respect to a perturbation in the input at another unit at time $t < T$.

Different methods can be used to calculate the gradient information needed for sensitivity analysis, i.e. through simulations, analytical approximations or exact analytical calculations (considering only first order derivatives):

- **Simulation calculations:** A brute-force way of calculating first order sensitivity information is to perform different simulations, each with only the parameter value θ being different. The performance function $\mathcal{P}$ is evaluated for each simulation. The average performance over all simulations gives an indication of the model's performance with respect to changes in parameter θ. This brute-force approach is mathematically expressed as [Ho 1987, Ho 1988]

$$\lim_{N\to\infty,\Delta\theta\to 0}[\frac{1}{N}\sum_{n=1}^{N}\mathcal{P}(\theta+\Delta\theta)-\frac{1}{N}\sum_{n=1}^{N}\mathcal{P}(\theta)]/\Delta\theta \tag{2.4}$$

  where N is the total number of simulations. This simulation approach to compute sensitivity information is time consuming and only approximate.

- **Analytical approximations:** From equation (2.3), the first order derivatives of the performance measure can be approximated using differences [Goh 1993, Jabri *et al* 1991, Kibsgaard 1992, Mozer *et al* 1989]

$$\frac{\partial\mathcal{P}}{\partial\theta}=\mathcal{P}'(\theta)\approx\frac{\mathcal{P}(\theta+\Delta\theta)-\mathcal{P}(\theta)}{\Delta\theta} \tag{2.5}$$

  which holds only when $\Delta\theta \to 0$. A fixed value is selected for the perturbation $\Delta\theta$, and the derivative $\mathcal{P}'(\theta^{(p)})$ is evaluated for each observation/pattern p. The method of

derivative approximation by means of differences is computationally less expensive than calculation through simulation, but still an approximation of the true derivatives.

Iwatsuki, Kawamata and Higuchi [Iwatsuki *et al* 1989], as well as Choi and Choi [Choi *et al* 1992], approximate sensitivity information by means of statistical calculations. They define statistical sensitivity analysis as the performance measure variance normalized by the variance of the parameter variations.

- **Exact analytical calculations:** Exact derivative calculations of first-order and second-order Taylor expansions have been used extensively to calculate sensitivity information [Cibas *et al* 1996, Czernichow 1996, Fu *et al* 1993, Dorizzi *et al* 1996, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Engelbrecht *et al* 1998a, Fletcher *et al* 1998, Schittenkopf *et al* 1997, Burrascano 1993, Guo *et al* 1992, Hashem 1992, Hassibi *et al* 1994, Laskey 1995, Le Cun 1990, Schittenkopf *et al* 1997, Takenaga *et al* 1991, Zurada *et al* 1994, Zurada *et al* 1997]. While exact calculations (as in appendix E) are more expensive, they are more accurate than analytical approximations. Second-order sensitivity calculations are even more expensive due to the calculation of the Hessian matrix. Usually, approximations to the Hessian matrix are used instead [Bishop 1992, Gorodkin *et al* 1993b, Le Cun 1990, Hassibi *et al* 1994].

A totally different approach to sensitivity analysis is developed by Holtzman [Holtzman 1992]. Instead of using the approximation in equation (2.3), the parameter θ is considered as a random variable, and $< \mathcal{P}(\theta) >$ and $var(\mathcal{P}(\theta))$ are estimated as a function of the mean and variance of θ; $< \bullet >$ denotes the expectation of $\bullet$. A small variance $var(\mathcal{P}(\theta))$ reveals an insignificant parameter θ.

2.4 Neural Network Sensitivity Analysis

Neural networks can approximate most linear and non-linear input-output mappings through a combination of weights. Funahashi [Funahashi 1989] and Hornik, Stinchcombe and White [Hornik 1989, Hornik *et al* 1990] have proven that any continuous mapping can be approximately realized by a multilayer NN with monotonically increasing differentiable activation functions. Furthermore, Hornik *et al* [Hornik *et al* 1990] and Gallant and White

[Gallant *et al* 1992] show that, when a NN converges towards the underlying (target) function, all the NN derivatives also converge towards the derivatives of the underlying function. This property of NNs allows efficient use of the NN derivatives to compute sensitivity information. (Also refer to section 2.5 for experiments which illustrate the accuracy of analytical derivative calculations.)

Section 2.3 presented a short overview of the different ways in which sensitivity information can be calculated. This section considers only the two main approaches to NN sensitivity analysis, i.e. with respect to the objective function and with respect to the NN output function, using exact analytical calculations. A general formulation of the two approaches is given, and their assumptions and complexity are discussed. The functionality of the two approaches are then compared, with reference to pruning.

2.4.1 Objective Function Sensitivity Analysis

One of the most widely used methods of NN sensitivity analysis is that of the objective function with respect to NN parameters. Usually, the sum squared error (SSE) as defined in equation (1.2) is used as objective funtion. If $\mathcal{E}$ denotes the objective function, $\vec{\theta} = (\theta_1, \cdots, \theta_I)$ the parameter vector of the NN, θ_i a single parameter, and $\Delta\theta_i$ a small perturbation of that parameter, then from (2.2)

$$\mathcal{E}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) = \mathcal{E}(\vec{\theta}) + \mathcal{E}'(\vec{\theta})\Delta\theta_i + \frac{1}{2}\mathcal{E}''(\vec{\theta})\Delta\theta_i^2 + \cdots \tag{2.6}$$

is the Taylor expansion of $\mathcal{E}$ around θ_i. From equation (2.6), the change in error due to perturbation $\Delta\theta_i$ is

$$\mathcal{E}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) - \mathcal{E}(\vec{\theta}) = \mathcal{E}'(\vec{\theta})\Delta\theta_i + \frac{1}{2}\mathcal{E}''(\vec{\theta})\Delta\theta_i^2 + \cdots \tag{2.7}$$

The first order term $\mathcal{E}'(\vec{\theta})$ is used in gradient descent optimization to drive the NN to a local minimum [Rumelhart *et al* 1986]. In this case θ_i represents a weight of the NN. The second order term has also been used in optimization to improve convergence [Battiti 1992, Becker *et al* 1988]. Objective function sensitivity analysis has been used widely in pruning of NN parameters. Optimal Brain Damage (OBD) [Gorodkin *et al* 1993b, Le Cun 1990, Pedersen *et al* 1996] and Optimal Brain Surgeon (OBS) [Hassibi *et al* 1993,

Hassibi *et al* 1994, Pedersen *et al* 1996] prune weights with low "saliency," while Optimal Cell Damage (OCD) [Cibas *et al* 1994a, Cibas *et al* 1994b, Cibas *et al* 1996] prunes irrelevant input and hidden units. OBD, OBS and OCD use second order derivatives to approximate saliencies.

Objective function sensitivity analysis in OBD, OBS and OCD are based upon assumptions to reduce the complexity of calculating equation (2.7) [Cibas *et al* 1994b, Cibas *et al* 1996, Gorodkin *et al* 1993b, Hassibi *et al* 1993, Hassibi *et al* 1994, Le Cun 1990]. These assumptions are discussed below:

- **Extremal approximation assumption:** It is assumed that pruning is applied only after convergence is reached. At the local minimum, the derivative of the objective function is approximately zero, which means that the first term, $\mathcal{E}'(\vec{\theta})\Delta\theta_i$, can be removed from equation (2.7). The extremal approximation assumption also relies on the assumption that noise follows a Gaussian, zero mean distribution. The extremal approximation assumption is not valid if many outliers occur in the training set.

- **Quadratic approximation assumption:** The objective function is assumed to be well approximated by a second-order expansion around its minimum point. This is not always the case, especially for flat surfaces with a bath tub-like shape. Gorodkin *et al* show for some experiments that the second order approximation does not give an accurate description of the cost function [Gorodkin *et al* 1993b]. This assumption also applies only for objective functions that are linear or quadratic.

- **Diagonal approximation assumption:** OBD and OCD assume that the off-diagonal terms of the Hessian matrix are zero. This assumption is only valid if it can be assumed that the principal curvature of the error surface is captured in the diagonal terms. However, there may be regions in the error surface of a problem where small changes in some weights result in very large changes in the error gradient. That is, in regions where the principle curvature is not parallel to the weight axes. Becker and Le Cun illustrate this to be true for experiments investigated by them [Becker *et al* 1988]. They show through an eigenvalue decomposition of the Hessian matrix that off-diagonal terms also have high eigenvalues, indicating regions in the error surface that are sensitive

to weight perturbations. Hassibi and Stork also found the diagonal assumption to be incorrect, leading to the pruning of the wrong weights [Hassibi *et al* 1993]. The diagonal assumption loses some information about the characteristics of the objective function and error surface. For this reason, OBS does not assume off-diagonal terms to be zero, but uses the full Hessian matrix which is extremely expensive to compute - especially for large networks.

- **Levenberg-Marquardt assumption:** It is assumed that the errors between the target and output values, $t_k^{(p)} - o_k^{(p)}$ are approximately zero. All $t_k^{(p)} - o_k^{(p)}$ terms are therefore removed from the sensitivity equations (refer to section E.2 for the equations with and without the error terms). Assuming a Gaussian noise with zero mean for the input space, the correlation between errors and the second-order derivatives of the objective function vanishes for large training sets. However, if many outliers occur in the training set, the Levenberg-Marquardt assumption may lead to inaccuracies.

Using these assumptions, OBD and OCD define the saliency measure S_{θ_i} of parameter θ_i from equation (2.7) as

$$S_{\theta_i} = \mathcal{E}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) - \mathcal{E}(\vec{\theta}) \approx \frac{1}{2} H \theta_i^2 \tag{2.8}$$

and OBS as

$$S_{\theta_i} = \mathcal{E}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) - \mathcal{E}(\vec{\theta}) \approx \frac{1}{2} \frac{\theta_i^2}{[H^{-1}]_{ii}} \tag{2.9}$$

where $H = \frac{\partial^2 \mathcal{E}}{\partial \vec{\theta}^2}$ is the Hessian matrix containing all the second order derivatives, and $[H^{-1}]_{ii}$ denotes the i^{th} diagonal element of the inverted Hessian.

The equations for calculating $\frac{\partial^2 \mathcal{E}}{\partial \theta_i^2}$ (refer to section E.2 where the complete equations are derived) depend on the objective function (usually the SSE function) and the activation functions (usually sigmoid functions). For NNs that use a different objective function, the OBD, OBS and OCD sensitivity analysis models change substantially. Changes in activation functions cause only minor changes in the sensitivity equations.

2.4.2 Output Sensitivity Analysis

Sensitivity analysis of the NN output, from now on referred to as NN output sensitivity analysis, is based on the first-order approximation in equation (2.3), and has been used for several NN applications [Basson *et al* 1999, Choi *et al* 1992, Cloete *et al* 1994c, Czernichow 1996, Dorizzi *et al* 1996, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Engelbrecht *et al* 1998a, Engelbrecht *et al* 1998b, Goh 1993, Guo *et al* 1992, Engelbrecht *et al* 1999a, Engelbrecht 1999c, Engelbrecht *et al* 1999d, Engelbrecht *et al* 1999e, Fletcher *et al* 1998, Fu *et al* 1993, Laskey 1995, Modai *et al* 1995, Takenaga *et al* 1991, Viktor *et al* 1995, Viktor *et al* 1998a, Zurada *et al* 1994, Zurada *et al* 1997]. Without loss of generality assume one output unit. If $\mathcal{F}_{NN}$ denotes the output function of the NN, $\vec{\theta} = (\theta_1, \cdots, \theta_I)$ the parameter vector, θ_i a single parameter and $\Delta\theta_i$ a small perturbation of θ_i, then

$$\mathcal{F}_{NN}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) - \mathcal{F}_{NN}(\vec{\theta}) = \mathcal{F}'_{NN}(\vec{\theta})_{\theta_i}\Delta\theta_i \tag{2.10}$$

The change in output due to the perturbation is then entirely described by the derivative

$$\mathcal{F}'_{NN}(\theta)_{\theta_i} = \lim_{\Delta\theta_i \to 0} \frac{\mathcal{F}_{NN}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) - \mathcal{F}_{NN}(\vec{\theta})}{\Delta\theta_i} \tag{2.11}$$

Output sensitivity analysis therefore consists of simply calculating $\frac{\partial \mathcal{F}_{NN}}{\partial \theta_i}$ for all parameters θ_i (refer to section E.1.1 for the complete equations for feedforward NNs).

Output sensitivity analysis has been used to study the generalization characteristics of NNs [Choi *et al* 1992, Fu *et al* 1993], for causal inferencing to determine the significance of input parameters [Engelbrecht *et al* 1995b, Goh 1993, Guo *et al* 1992, Laskey 1995, Modai *et al* 1995], to quantify the degree of non-linearity in the data [Lamers *et al* 1998], to detect and visualize decision boundaries [Engelbrecht *et al* 1998a, Engelbrecht *et al* 1998b, Engelbrecht *et al* 1999a, Engelbrecht 1999c, Goh 1993, Viktor *et al* 1998a], to prune oversized NN architectures [Cloete *et al* 1994c, Czernichow 1996, Dorizzi *et al* 1996, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Engelbrecht *et al* 1999e, Fletcher *et al* 1998, Takenaga *et al* 1991, Viktor *et al* 1995, Zurada *et al* 1994, Zurada *et al* 1997], for active learning [Engelbrecht *et al* 1998a, Engelbrecht *et al* 1999d], and for automatically learning first-order derivatives [Basson *et al* 1999]. For pruning purposes, sensitivity analysis is used to compute the significance of weights, input and hidden units [Cloete *et al* 1994c,

Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Engelbrecht *et al* 1999e, Viktor *et al* 1995, Zurada *et al* 1994, Zurada *et al* 1997].

Output sensitivity analysis as presented in this thesis is not based on any assumptions to reduce model complexity. The only assumptions are that (1) the activation functions are at least once differentiable, and (2) the NN should be well trained to accurately approximate the true derivatives. This is necessary for pruning to correctly remove irrelevant parameters. This assumption is not as strict as the extremal approximation assumption of OBD, OBS and OCD, since the validity of the output sensitivity analysis model does not depend on the network being in a local minimum.

The output sensitivity equations are **independent of the objective function**, since the NN output is taken as performance function, and not the objective function as in OBD, OBS and OCD. Whatever error function is used, the equations to compute the derivatives of the NN output function with regard to network parameters, as given in appendix E.1.1, remain the same. They do, however, depend on the type of activation function used, due to the need to calculate $\frac{\partial o_k}{\partial \theta}$, where θ can be an input unit (refer to equation (E.6)), a hidden unit (refer to equation (E.7)), or a weight (refer to equations (E.8) and (E.9)), and the need to calculate $\frac{\partial y_j}{\partial \theta}$, where θ can be an input unit (refer to equation (E.11)), or a weight (refer to equation (E.9)).

2.4.3 Comparison of Neural Network Sensitivity Analysis Models

The objective of this section is to compare the two approaches to NN sensitivity analysis as discussed in sections 2.4.1 and 2.4.2. After a general comparison of the complexity and characteristics of the two approaches, the section concludes with a mathematical comparison between OBD and output sensitivity analysis for NN pruning.

The main difference between objective and NN output sensitivity analysis is the performance function used. Objective function sensitivity analysis uses the error function to be optimized as measure of the change in error caused by small parameter perturbations as expressed in equation (2.7). Output sensitivity analysis, on the other hand, uses the actual NN output

function as performance measure to quantify the change in NN output due to small perturbations, as expressed in equation (2.10). Conceptually, the two approaches mean the same: The error of a single pattern is computed as the difference $(t_k^{(p)} - o_k^{(p)})$ between the target value $t_k^{(p)}$ and the actual output value $o_k^{(p)}$ of the k^{th} output unit for pattern p. A change in error, due to some perturbation, is determined by a change in the output value caused by that perturbation. This relationship is further illustrated in table 2.1 and equations (2.13) and (2.14), considering the assumptions as for OBD.

Parameter	**Error Sensitivity**	**Output Sensitivity**
z_i	$\frac{\partial^2 \mathcal{E}_k}{\partial z_i^2} \approx (f'_{o_k})^2 [\sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji}]^2$	$\frac{\partial o_k}{\partial z_i} = f_{o_k} \sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji}$
	from (E.39)	from (E.6)
y_j	$\frac{\partial^2 \mathcal{E}_k}{\partial y_j^2} \approx (f'_{o_k})^2 w_{kj}^2$	$\frac{\partial o_k}{\partial y_j} = f'_{o_k} w_{kj}$
	from (E.45)	from (E.7)
w_{kj}	$\frac{\partial^2 \mathcal{E}_k}{\partial w_{kj}^2} \approx (f'_{o_k})^2 y_j^2$	$\frac{\partial o_k}{\partial w_{kj}} = f'_{o_k} y_j$
	from (E.51)	from (E.8)
v_{ji}	$\frac{\partial^2 \mathcal{E}_k}{\partial v_{ji}^2} \approx (f'_{y_j})^2 (f'_{o_k})^2 w_{kj}^2 z_i^2$	$\frac{\partial o_k}{\partial v_{ji}} = f'_{y_j} f'_{o_k} w_{kj} z_i$
	from (E.59)	from (E.9)

Table 2.1: Comparison of Objective Function and Output Sensitivity Analysis

While objective function and NN output sensitivity analysis mean conceptually the same thing, it is more complex to compute objective function sensitivity information. Since the goal of learning is to minimize the objective function, the first order derivative of the objective function, $\mathcal{E}$, is approximately zero at convergence. Thus requiring second order derivatives to be computed. Since this needs the calculation of the Hessian matrix, objective function sensitivity analysis is computationally expensive. In contrast, with NN output sensitivity analysis, first order information is sufficient to quantify the influence of parameter perturbations, since we can assume that [Zurada *et al* 1997]

$$\lim_{\Delta\theta_i \to 0} (\frac{1}{2} \Delta \mathcal{F}''_{NN}(\vec{\theta}) \Delta\theta_i^2 + \cdots) = 0 \tag{2.12}$$

where $\mathcal{F}_{NN}$ is the NN output function. It is much less expensive to calculate the Jacobian matrix than the Hessian matrix.

Output sensitivity analysis is also more general than objective function sensitivity analysis in

that the latter depends on the error function (objective function) to be optimized. Usually, the sum squared error function is used, but for any other error function, the sensitivity equations as summarized in table 2.1 need to be redefined. The output sensitivity analysis equations, on the other hand, remain the same whatever objective function is used.

Since both sensitivity analysis approaches have been applied to pruning of NN architectures, this application is used to find a mathematical relationship between the two methods. For this purpose OBD is used, considering all its assumptions as listed in section 2.4.1. To derive this relationship, assume a NN with one output o_k. Although the comparison below is for one pattern only, it can quite easily be generalized to include the entire training set through application of a suitable norm. Table 2.1 summarizes the sensitivity equations for the two approaches as obtained from appendix E.

From table 2.1, irrespective of which NN parameter is considered, the following general relationship applies (assuming least squares as objective function):

$$\frac{\partial^2 \mathcal{E}_k}{\partial \theta^2} \approx (\frac{\partial o_k}{\partial \theta})^2 \tag{2.13}$$

This supports the statement that objective function sensitivity analysis and NN output sensitivity analysis are conceptually the same (under the assumptions listed in section 2.4.1). This means that the same parameter significance ordering will occur for the two methods. In the case of pruning, the same parameters will therefore be pruned (also refer to section 5.2.3 where this relationship is explored again in the context of pruning). In general, for more than one output the following relationship holds:

$$\frac{\partial^2 \mathcal{E}}{\partial \theta^2} = \sum_{k=1}^{K} \frac{\partial^2 \mathcal{E}_k}{\partial \theta^2} \approx \sum_{k=1}^{K} (\frac{\partial o_k}{\partial \theta})^2 \tag{2.14}$$

illustrating that the change in model error due to parameter perturbations is simply an additive function of the changes in NN output due to these perturbations. Therefore, instead of using a more complex objective function sensitivity approach, output sensitivity analysis can be used to the same effect.

The next section empirically investigates how well the true derivatives of the learned function are approximated.

2.5 First-Order Output Sensitivity Analysis Results

It has been proven theoretically that feedforward NNs with monotonically increasing differentiable activation functions can approximate any continuous mapping [Funahashi 1989, Hornik 1989, Hornik *et al* 1990]. The derivatives of the realized function then also converge to the true derivatives [Gallant *et al* 1992, Hornik *et al* 1990]. This section empirically investigates how well the true derivatives are approximated by the analytical sensitivity equations given in appendix E. For this purpose, two one-dimensional functions are approximated using a three layer feedforward network with sigmoidal activation functions trained by gradient descent.

Firstly, the experimental procedure is described. For each function, two experiments are performed. The first uses the same training and test set pairs for 50 simulations, where each simulation starts with different initial conditions. The use of the same training and test sets allow graphs to be plotted to illustrate the approximation accuracy. The second experiment performs 50 simulations, each on different training and test set pairs, and different initial conditions.

The correlation coefficient, defined as [Steyn *et al* 1995]

$$\begin{aligned} r &= \frac{\sum_{i=1}^{n}(x_i-\overline{x})\sum_{i=1}^{n}(y_i-\overline{y})}{\sigma_x\sigma_y} \\ &= \frac{\sum_{i=1}^{n}x_iy_i-\frac{1}{n}\sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i}{\sqrt{\sum_{i=1}^{n}x_i^2-\frac{1}{n}(\sum_{i=1}^{n}x_i)^2}\sqrt{\sum_{i=1}^{n}y_i^2-\frac{1}{n}(\sum_{i=1}^{n}y_i)^2}} \end{aligned} \tag{2.15}$$

where x_i and y_i are observations, $\overline{x}$ and $\overline{y}$ are respectively the averages over all observations x_i and y_i, and σ_x and σ_y are the standard deviations of the x_i and y_i observations respectively, can be used to quantify the linear relationship between variables x and y. In this section, the correlation coefficient is used to quantify the linear relationship between the approximated (learned) function and the true function, as well as between the approximated derivative and the true derivative. A correlation value close to 1 indicates a good approximation to the true function. For example, the correlation coefficient

$$r = \frac{\sum_{p=1}^{P}o_k^{(p)}t_k^{(p)}-\frac{1}{P}\sum_{p=1}^{P}o_k^{(p)}\sum_{p=1}^{P}t_k^{(p)}}{\sqrt{\sum_{p=1}^{P}o_k^{(p)^2}-\frac{1}{P}(\sum_{p=1}^{P}o_k^{(p)})^2}\sqrt{\sum_{p=1}^{P}t_k^{(p)^2}-\frac{1}{P}(\sum_{p=1}^{P}t_k^{(p)})^2}} \tag{2.16}$$

is calculated as measure of how well the NN approximates the true function.

All results reported in this section are averages over the 50 simulations.

Function 1

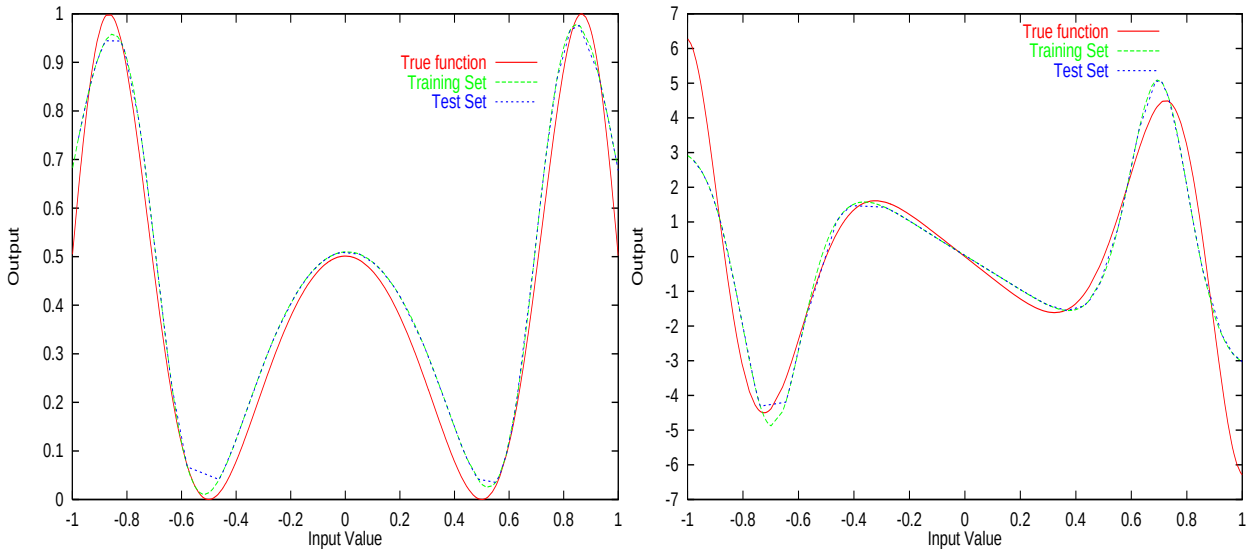
The first function approximated is

$$o = f(z) = \sin(2\pi(1 - z^2)) \tag{2.17}$$

where z was sampled from an uniform distribution, i.e. $z \sim U(-1, 1)$, and the output o was scaled to the range $[0, 1]$. No noise term was added to the function. The true derivative of this function is calculated as

$$f^{'}(z) = -4\pi z \cos(2\pi(1 - z^2)) \tag{2.18}$$

For training purposes, the learning rate was fixed at 0.05, while the momentum was 0.9. Training sets consisted of 240 patterns, while the test sets consisted of 60 patterns. Training stopped when a mean squared error (MSE) of 0.05 was reached on the training sets, which is the point at which overfitting was observed.



(a) Approximation to $f(z) = \sin(2\pi(1 - z^2))$ (b) Approximation to $f^{'}(z) = -4\pi z \cos(2\pi(1 - z^2))$

Figure 2.1: Output sensitivity for $f(z) = \sin(2\pi(1 - z^2))$

Figure 2.1(a) plots the true function and its approximation for the training and test sets. The approximation to the true derivatives is illustrated by figure 2.1(b). The average correlation between the true and approximated functions over the 50 simulations is given in table 2.2. These results illustrate that sensitivity analysis of the output with respect to the inputs do represent good approximations to the true derivatives, having high correlation values and very similar correlations between the training and test sets. Note that these results depend on how well the network is trained. Better approximations to the true derivatives can be obtained if the network is trained to a higher accuracy, provided that the network does not overfit the training set.

	Correlation Coefficient for Function Approximation	
Experiment	**Training Set**	**Test Set**
Same sets	0.993506 ± 0.000352	0.992226 ± 0.000438
Different sets	0.965214 ± 0.020753	0.961285 ± 0.023324

	Correlation Coefficient for Derivative Approximation	
Experiment	**Training Set**	**Test Set**
Same sets	0.938029 ± 0.003602	0.941139 ± 0.003329
Different sets	0.931128 ± 0.010456	0.928263 ± 0.01374

Table 2.2: Correlation coefficients for $f(z) = \sin(2\pi(1 - z^2))$, and its derivative

Function 2

The second function was used with noise added, where $\zeta \sim N(0, 1)$,

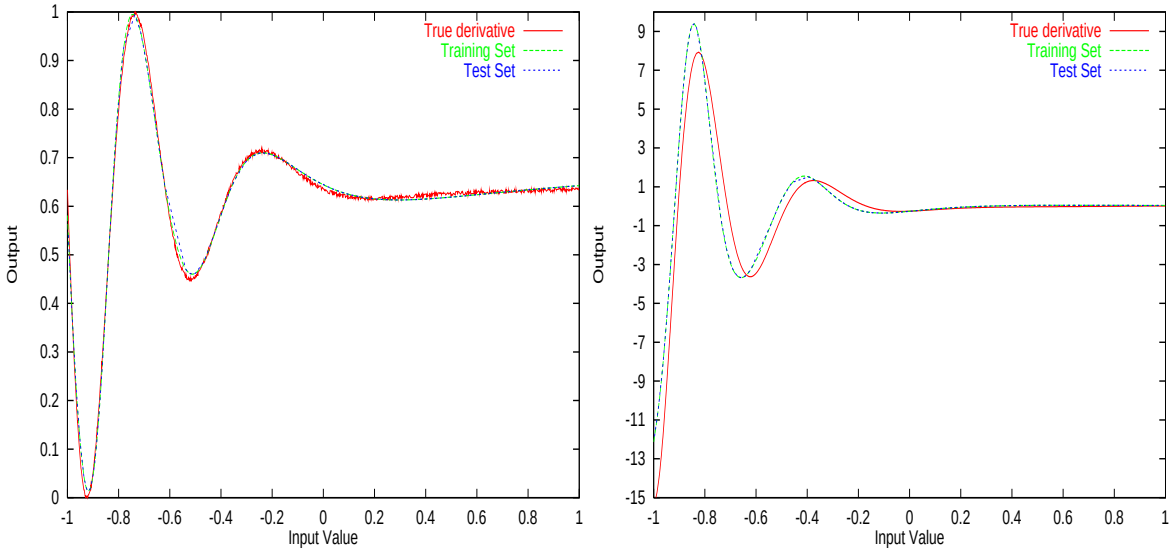
$$o = f(z) = e^{-3z} \sin(2\pi z(z - 1)) + \zeta \tag{2.19}$$

where $z \sim U(-1, 1)$, and the output was scaled to the range $[0, 1]$. The true derivative of this function is calculated as

$$f'(z) = e^{-3z}[(4\pi z - 2\pi)\cos(2\pi z(z - 1)) - 3\sin(2\pi z(z - 1))] \tag{2.20}$$

A fixed learning rate of 0.01 and a momentum of 0.9 were used. Training sets consisted of 720 patterns and test sets contained 180 patterns. Learning stopped when a MSE of 0.01 was reached on the training sets, which is the point where overfitting was observed.

Figures 2.2(a) and 2.2(b) respectively illustrate the approximation to the true function and its derivatives. Table 2.3 summarizes the average correlation values. Again, these results show high correlations between the true and approximated functions. Also, the correlation values obtained for the training and test sets are very similar, illustrating no severe discrepancies between the NN's generalization of the derivative and the true derivative of the function. These correlation values show an acceptable approximation to the true derivative using the output sensitivity analysis equations, thus indicating that output sensitivity analysis can be used effectively in applications that need accurate approximations to the true derivatives. Of course, better approximations to the true derivatives will be obtained for a smaller MSE on the training set, provided that no overfitting occurs.



(a) Approximation to $f(z) = e^{-3z}\sin(2\pi z(1-z))$

(b) Approximation to $f'(z) = e^{-3z}\{(4\pi z - 2\pi)\cos(2\pi z(z-1)) - 3\sin(2\pi z(z-1))\}$

Figure 2.2: Output sensitivity for $f(z) = e^{-3z}\sin(2\pi z(1-z))$

	Correlation Coefficient for Function Approximation	
Experiment	**Training Set**	**Test Set**
Same sets	0.997326 ± 0.001751	0.997950 ± 0.001114
Different sets	0.998592 ± 0.000072	0.998463 ± 0.0000131

	Correlation Coefficient for Derivative Approximation	
Experiment	**Training Set**	**Test Set**
Same sets	0.931647 ± 0.001554	0.928797 ± 0.001298
Different sets	0.998592 ± 0.001272	0.998463 ± 0.004036

Table 2.3: Correlation coefficients for $f(z) = e^{-3z}\sin(2\pi z(1-z))$, and its derivative

2.6 Proposed Uses of Neural Network Output Sensitivity Analysis

The main focus of this thesis is on the uses of output sensitivity analysis information of feedforward multilayer neural networks. The uses presented and discussed in the following chapters have as objectives to improve generalization and convergence performance, and to decrease model and time complexity.

The first application of sensitivity analysis is to visualize the position of decision boundaries in input space, and to determine which hidden unit implements which boundary. Chapter 3 presents definitions of decision boundaries and shows how sensitivity analysis can be used to extract meaning from these boundaries. The boundary detection model presented in the next chapter is used in the selective learning algorithm developed in chapter 4.

Chapter 4 presents two new active learning algorithms that use output sensitivity information to dynamically select training patterns from a candidate training set. Both a selective learning algorithm for classification problems and an incremental learning algorithm for function approximation are presented. Sensitivity information is used as a measure of pattern informativeness, facilitating the selection of the most informative patterns.

Chapter 5 presents a pruning algorithm based on NN output sensitivity analysis. Parameter significance values, computed as a function of output sensitivity measures, are used to identify

and prune irrelevant parameters. The pruning model is applied to the pruning of the input and hidden layers.

2.7 Sensitivity Analysis of Different Neural Network Types

Sensitivity analysis can be applied to different multilayer NN types, as long as these NN types use differentiable activation functions. This section shows mathematically that output sensitivity analysis can be applied to feedforward neural networks (FFNN), functional link neural networks (FLNN) and product unit neural networks (PUNN). Koda developed a sensitivity analysis model for continuous-time recurrent neural networks (RNN), which will not be repeated here [Koda 1995, Koda 1997]. The interested reader is referred to Koda's work for illustrations of the applicability of sensitivity analysis to RNNs. Also, Czernichow [Czernichow 1996] and Dorizzi *et al* [Dorizzi *et al* 1996] showed that output sensitivity analysis can be applied to Radial Base Function NNs.

The section starts with FFNNs, and then shows how the sensitivity equations change for the other network types.

2.7.1 Feedforward Neural Networks

This section presents a general formulation of sensitivity analysis for feedforward neural networks. The reader is referred to appendix E.1.1 where the complete derivations for the sensitivity of the NN output to network parameters are given for this network type, using the NN architecture depicted by figure A.1.

Assume a three layer architecture with an input layer, one hidden layer and an output layer, and a bias unit in the input and hidden layers. For a FFNN, we have in general that the sensitivity $S_{oz}^{(p)}$ of the output layer to input perturbations for a specific pattern p is:

$$S_{oz} = F_o^{(p)'} W F_y^{(p)'} V \tag{2.21}$$

where W and V are respectively the weight matrices between the hidden and output layers,

and the input and hidden layers. The derivative matrices are defined as

$$\begin{aligned} F_o^{(p)'} &\doteq \text{diag}(f_{o_1}^{(p)'}, f_{o_2}^{(p)'}, \cdots, f_{o_K}^{(p)'}) \\ F_y^{(p)'} &\doteq \text{diag}(f_{y_1}^{(p)'}, f_{y_2}^{(p)'}, \cdots, f_{y_J}^{(P)'}) \end{aligned}$$

For FFNNs with sigmoid activation functions the derivatives are defined as

$$\begin{aligned} f_{o_k}^{(p)'} &= o_k^{(p)}(1 - o_k^{(p)}) \\ f_{y_j}^{(p)'} &= y_j^{(p)}(1 - y_j^{(p)}) \end{aligned}$$

where $o_k^{(p)}$ and $y_j^{(p)}$ are respectively the activations of output unit o_k and hidden unit y_j for pattern p.

2.7.2 Functional Link Neural Networks

This section illustrates that sensitivity analysis can also be applied to functional link neural networks (FLNN). In FLNNs the input layer is expanded into a layer of functional, higher-order units [Ghosh *et al* 1992, Hussain *et al* 1997, Zurada 1992b]. The input layer, with dimension I, is therefore expanded to functional units $h_1, h_2, \cdots, h_L$, where L is the total number of functional units, and each functional unit h_l is a function of the input parameter vector $(z_1, \cdots, z_I)$, i.e. $h_l(z_1, \cdots, z_I)$. The weight matrix U between the input layer and the layer of functional units is defined as

$$U_{li} = \begin{cases} 1 & \text{if functional unit } h_l \text{ is dependent of } z_i \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

For FLNNs, all the sensitivity equations with reference to input parameter perturbations need to be updated as follows:

$$S_{oz}^{(p)} = F_o^{(p)'} W F_y^{(p)'} V F_{h,i}^{(p)'} \tag{2.23}$$

where V is the weight matrix between the functional link layer and the hidden layer, and

$$F_{h,i}^{(p)'} \doteq \text{diag}(f_{h_1,i}^{(p)'}, f_{h_2,i}^{(p)'}, \cdots, f_{h_L,i}^{(p)'}) \tag{2.24}$$

with

$$f_{h_l,i}^{(p)'} = \frac{\partial h_l}{\partial z_i^{(p)}} \tag{2.25}$$

The sensitivity equations with reference to other network parameters remain the same as for FFNNs.

2.7.3 Product Unit Neural Networks

Lastly, the applicability of sensitivity analysis to product unit neural networks (PUNN) is illustrated. Assume a NN with product units in the hidden layer, with linear activations. That is, the normal summation units are replaced with product units such that the net input to hidden unit y_j is given as [Durbin *et al* 1989, Ghosh *et al* 1992, Janson *et al* 1993, Leerink *et al* 1995]

$$net_{y_j}^{(p)} = \prod_{i=1}^{I} z_i^{(p)^{v_{ji}}} \tag{2.26}$$

Since the hidden units are linearly activated, the sensitivity of output unit o_k to input unit z_i for pattern p is expressed as

$$S_{oz,ki}^{(p)} = f_{o_k}^{(p)'} \sum_{j=1}^{J} w_{kj} \frac{v_{ji}}{|z_i^{(p)}|} e^{\rho_j} \cos(\pi \phi_j) \tag{2.27}$$

where

$$\begin{aligned} \rho_j &= \sum_{i=1}^{I} v_{ji} \ln |z_i^{(p)}| \\ \phi_j &= \sum_{i=1}^{I} v_{ji} \mathcal{I}_i \end{aligned}$$

with

$$\mathcal{I}_i = \begin{cases} 0 & \text{if } z_i^{(p)} > 0 \\ 1 & \text{if } z_i^{(p)} < 0 \end{cases}$$

It is assumed that $z_i^{(p)} \neq 0$.

The reader is referred to appendix E.1.2 for the complete PUNN sensitivity analysis derivations.

The purpose of this section was to show mathematically that output sensitivity analysis can be applied to different NN types. For the rest of this thesis a three layer FFNN with sigmoid

activation functions is assumed. Future work beyond this thesis will include an empirical investigation to test the sensitivity analysis tools, developed in later chapters, on the other NN types.

Chapter 3

Sensitivity Analysis Decision Boundary Visualization

"The sublime and the ridiculous
are often so nearly related that
it is difficult to class them separately"
-Tom Paine

This chapter illustrates how sensitivity analysis of the NN output with respect to input and hidden units can be used to visually inspect the position of decision boundaries, and to determine which decision boundary is implemented by which hidden units.

3.1 Introduction

The first application of NN sensitivity analysis is presented in this chapter, i.e. the detection and visualization of decision boundaries in *classification* problems. The main objective of NN learning in classification problems is to construct optimal decision boundaries in the input space to discriminate among the different classes. *A decision boundary is a region in input space of maximum ambiguity in classification* - in other words, a region of uncertainty in the classification. It is the task of the hidden units to form these boundaries that separate the

different classes. Provided that the network contains an optimal number of hidden units, each hidden unit implements an unique discriminating boundary. The optimal number of hidden units can be obtained from pruning an oversized network, or by growing an undersized network as discussed in chapter 5.

Decision boundaries reveal many interesting characteristics of the modeled data set, which helps to better understand the problem being modeled:

- **Relevant input parameters.** If no boundary can be defined over an input parameter, that parameter does not contribute to the classification and can therefore be removed from the model. This is exactly the effect of the sensitivity analysis pruning algorithm presented in chapter 5, where an average over training pattern sensitivities is used as pruning criterion. Lee and Langrebe explicitly use decision boundaries to perform input layer pruning. They define a decision boundary feature matrix from which relevant features (i.e. relevant input parameters) are extracted [Lee *et al* 1992]. Irrelevant features are removed from the network.

- **Rule extraction.** Decision boundaries aid in the extraction of conditional if-then rules from continuous valued input parameters of the form *IF input_parameter relational_operator boundary THEN...* Such rules help to better understand under which input conditions classification changes, since they accurately describe boundary conditions [Engelbrecht *et al* 1999a, Viktor *et al* 1998a, Viktor 1998b].

- **Informative patterns.** Decision boundaries describe regions of input space where classification is uncertain [Engelbrecht *et al* 1998a]. Patterns in these regions are considered as being most informative, since they convey the most information to refine boundaries. These informative patterns are used for dynamic pattern selection during training in chapter 4.

- **Hidden unit analysis.** Visualization of decision boundaries helps to understand the functioning of hidden units. It helps to determine which hidden unit implements which boundary, allowing the identification of hidden units that learn the same boundary, or

hidden units that implement no boundary at all [Engelbrecht 1999c]. Rule extraction algorithms find the set of hidden units that cause an output unit to produce a positive classification [Craven *et al* 1993, Towell *et al* 1993, Viktor *et al* 1995, Viktor 1998b]. Then, the set of input units that causes each hidden unit to be activated is determined. Knowledge of which hidden unit implements which decision boundary makes it easier to determine thresholds over the possible values of input parameters that cause a change in classification [Engelbrecht *et al* 1999a, Viktor *et al* 1998a, Viktor 1998b]. These thresholds are the positions of the decision boundaries. For example, in a rule such as *if* $A < 0.5$ *then* B, the value 0.5 is a threshold for parameter A, and a hidden unit will approximate the boundary at $A = 0.5$.

Several approaches have been developed to locate decision boundaries. The objective of this chapter is to illustrate how sensitivity analysis can be used to *visualize* the position of boundaries over each input parameter, and to see which boundary is implemented by which hidden unit. The objective is not to present a new algorithm to define the equations of the boundaries. The selective learning algorithm presented in the next chapter uses the decision boundary algorithm to dynamically select patterns close to boundaries.

This chapter is outlined as follows. A short overview of decision boundary detection algorithms is presented in section 3.1.1. The sensitivity analysis decision boundary detection algorithm is formulated in section 3.2. The algorithm is then illustrated on artificial and real-world problems in section 3.3, illustrating how a better understanding of the underlying data can be obtained.

3.1.1 Related Work

Usually, the detection of decision boundaries involves computationally expensive algorithms. The simplest way to find boundaries is to extensively sample input space, which is a time consuming process - especially when many boundaries exist over input space [Lee *et al* 1992, Pratt *et al* 1994]. The search space can, however, be reduced if prior knowledge of the problem is available. For example, Cohn, Atlas and Ladner use distribution information from the environment to find decision boundaries [Cohn *et al* 1994b]. However, in many real-world

applications prior knowledge is not available.

Baum implements an iterative search to locate boundaries [Baum 1991]. This approach is efficient for less complex problems with a few boundaries, but scales to be expensive for problems with many boundaries.

Hwang, Choi, Oh and Marks invert the NN output function, and use this inverted function together with selective sampling to locate boundaries [Hwang *et al* 1990, Hwang *et al* 1991]. The inversion algorithm receives a sampled output, and through iterative gradient descent computes the desired input vector which forms the boundary. Crucial to the efficacy of this algorithm is the output sampling density. If too small, the algorithm may fail to locate all the boundaries.

Pratt and Christensen develop a method that combines hidden unit activation space sampling with solving a system of linear equations to obtain decision boundaries [Pratt *et al* 1994]. The characteristics of the sigmoid function is utilized to reduce sampling space: using sigmoid activation functions, the activation value of hidden units is within the finite range $[0, 1]$. Activation values in this range are sampled for each hidden unit, with the objective to find the set of activation values that causes an output unit to produce the value 0.5. The next step is to find the points in input space that produce these sample activations. This involves solving a set of linear equations $Az = y$, where A is a $J \times I$ matrix, z is a $1 \times I$ vector and y is a $1 \times J$ vector; I is the input dimension, and J is the dimension of hidden space.

Goh introduces a sensitivity analysis approach to visualize the position of decision boundaries closely related to the approach presented in this chapter [Goh 1993]. While this chapter uses exact analytical calculations to calculate sensitivity information, Goh approximates input parameter sensitivity using differences. In addition, this chapter also shows how this approach can be used to identify the boundary implemented by each hidden unit.

3.2 Sensitivity Analysis Decision Boundary Detection

This section identifies two types of decision boundaries that can be formed for **a single input parameter**, and presents formal definitions for these boundary types. A theoretical

explanation of these boundaries are then given, and algorithms for the visualization of the boundaries are developed. Since it is very difficult to visualize n-dimensional boundaries where $n > 3$, this study concentrates on definitions and visualizations from two dimensions.

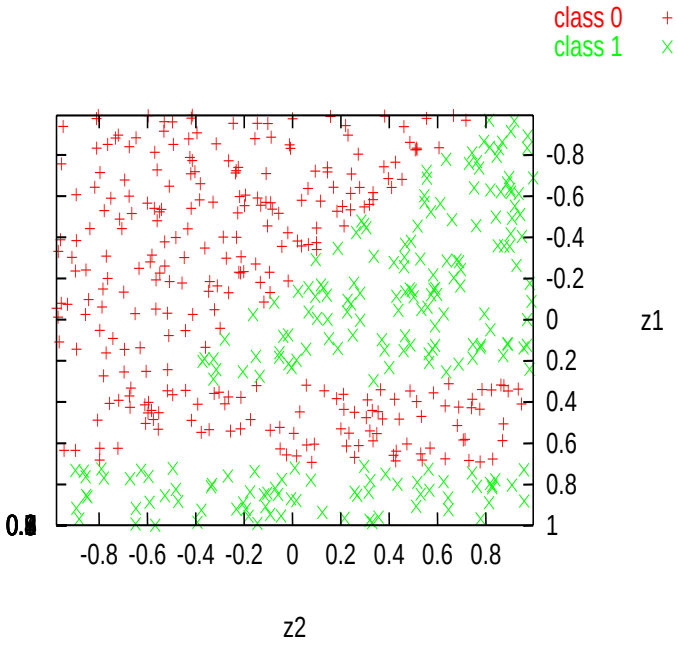


Figure 3.1: Artificial rule classification problem defined in equation (3.5)

Figure 3.1 illustrates, for an artificial problem defined in equation (3.5) on page 44, the two types of boundaries, referred to in this thesis as *axis-parallel boundaries* and *non-axis-parallel boundaries*. An axis-parallel boundary for an input z_i is a boundary that can be described by the equation $z_i = c$, where c is a value of z_i. A non-axis-parallel boundary spans over a range of values of z_i, for example $[c_1, c_2]$, and is governed by an equation which is a function of at least one other input parameter. Examples of axis-parallel boundaries in figure 3.1 are $z_1 = 0.3$ and $z_1 = 0.7$, while a non-axis-parallel boundary spans over $z_1 \in [-1, 0.3]$ and $z_2 \in [-0.4, 0.8]$.

Consider the following definitions of these decision boundaries, based on the assumptions given:

Assumption 3.1: A NN implements a differentiable mapping $\mathcal{F}_{NN}(D_T; W) : \mathbb{R}^I \rightarrow \mathbb{R}^K$, where D_T is the training set, W represents the weights, I is the dimension of input space, and K is the dimension of output space.

Assumption 3.2: The activation functions f_{o_k} in the output layer are monotonically increasing, bounded, and produces a binary discrete output $o_k^{(p)} = f_{o_k}(net_{o_k}^{(p)})$ for each output unit o_k, representing a specific class. ($net_{o_k}^{(p)} = \sum_{j=1}^{J+1} w_{kj} y_j^{(p)}$, where y_{J+1} is the hidden layer bias unit (see equation (D.4)))

If a continuous activation function such as the sigmoid function is used, the class to which a pattern belongs is determined from a rule such as

$$o_k^{(p)} = \begin{cases} 1 & \text{if } f_{o_k}(net_{o_k}^{(p)}) \geq 0.7 \\ 0 & \text{if } f_{o_k}(net_{o_k}^{(p)}) \leq 0.3 \end{cases}$$

which ensures a binary output. To ensure that output values are produced close to 1 or 0, a sigmoid function with large steepness λ is usually used, for example $\lambda \geq 5$ in

$$f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-\lambda net_{o_k}}}$$

Definition 3.1 Axis-parallel Decision Boundary: *Under assumptions 1 and 2, if there exists an input parameter value $z_i^{(p)}$ and a small perturbation Δz_i of $z_i^{(p)}$ such that, for any output unit o_k, $f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) \neq f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{p)})$, then a decision boundary is located in the range $[z_i^{(p)}, z_i^{(p)} + \Delta z_i]$ of input parameter z_i, where p denotes a single pattern.*

Definition 3.2 Non-axis-parallel Decision Boundary: *Under assumptions 1 and 2, if there exist two input parameter values $z_{i_1}^{(p)}$ and $z_{i_2}^{(p)}$ with $z_{i_1}^{(p)} < z_{i_2}^{(p)}$ such that for $z_i^{(p)} \in [z_{i_1}^{(p)}, z_{i_2}^{(p)}]$, a small perturbation Δz_i of $z_i^{(p)}$, and any output unit o_k, $f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) \neq f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{(p)})$, then a decision boundary spans over the range of values $[z_{i_1}^{(p)}, z_{i_2}^{(p)}]$ of input parameter z_i, where p denotes a single pattern.*

From definition 3.1, a decision boundary is located at the point in input space where a small perturbation to a value of an input unit causes the value of an output unit to change from

one class to another. Similarly, definition 3.2 defines a range of input parameter values over which a decision boundary is formed.

These definitions are theoretically justified from a first order Taylor expansion of f_{o_k} around $z_i^{(p)}$. For any differentiable function f_{o_k}, the characteristics of that function under small perturbations Δz_i of its input parameter z_i are expressed by the Taylor series

$$f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{(p)}) = f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) + \Delta z_i \frac{\partial f_{o_k}}{\partial z_i^{(p)}} + \frac{1}{2}\Delta z_i^2 \frac{\partial^2 f_{o_k}}{\partial z_i^{(p)^2}} + \cdots \tag{3.1}$$

Under the assumption of small Δz_i,

$$\lim_{\Delta z_i \to 0} (\frac{1}{2}\Delta z_i^2 \frac{\partial^2 f_{o_k}}{\partial z_i^{(p)^2}} + \cdots) = 0 \tag{3.2}$$

and equation (3.1) can be reduced to

$$f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{(p)}) \approx f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) + \Delta z_i \frac{\partial f_{o_k}}{\partial z_i^{(p)}} \tag{3.3}$$

Returning to definitions 3.1 and 3.2, under assumption 3.2, the second term in equation (3.3) determines whether the value of an output unit changes. That is, the higher the value of $\frac{\partial f_{o_k}}{\partial z_i^{(p)}}$, the greater the chance that $f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) \neq f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{(p)})$. Therefore, patterns with high $\frac{\partial f_{o_k}}{\partial z_i^{(p)}}$ values lie closest to decision boundaries [Engelbrecht *et al* 1998a, Engelbrecht *et al* 1998b, Engelbrecht *et al* 1999a, Engelbrecht 1999c, Viktor *et al* 1998a].

This point is further illustrated in figure 3.2, which plots, for example, the sigmoid activation function $f(z)$ and its derivative $\frac{\partial f}{\partial z}$. The peak of the derivative at $z = 0$ coincides with the inflection point of $f(z)$. For classification problems, the inflection point is used as threshold to decide between the two discrete output values.

Since $o_k^{(p)} = f_{o_k}(net_{o_k}^{(p)})$, $\frac{\partial o_k}{\partial z_i^{(p)}} = \frac{\partial o_k}{\partial f_{o_k}} \frac{\partial f_{o_k}}{\partial z_i^{(p)}}$, which is simply the sensitivity of output unit o_k to perturbations in the input value $z_i^{(p)}$. If sigmoid activation functions are used, equation (E.6) is used to calculate $\frac{\partial o_k}{\partial z_i^{(p)}}$ for each pattern p.

The first-order sensitivity analysis above assigns a "measure of closeness to boundaries" for each pattern [Engelbrecht *et al* 1998a, Engelbrecht *et al* 1999a, Engelbrecht 1999c,

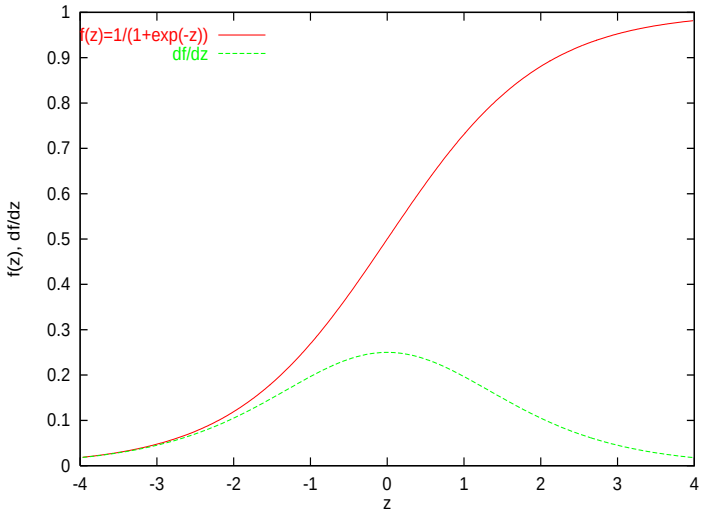


Figure 3.2: Sigmoid activation function, and derivative

Viktor *et al* 1998a]. Chapter 4 refers to this measure of closeness as *pattern informativeness* (refer to definition 4.2). Highly informative patterns convey the most information about decision boundaries. These patterns lie closest to the boundaries, and therefore have the highest $\frac{\partial o_k}{\partial z_i^{(p)}}$ values.

The position of boundaries can be visualized drawing scatter plots of $\frac{\partial o_k}{\partial z_i^{(p)}}$ versus $z_i^{(p)}$ for each output o_k and input z_i, for each pattern p. Peaks in these graphs give an indication of the position of axis-parallel boundaries, while a set of approximately equally high sensitivity values corresponds to a non-axis-parallel decision boundary (refer to section 3.3 for some illustrations). These graphs can be used to investigate regions of input space for which classification is uncertain, and to determine under what conditions the result of the classification changes.

In addition to finding and visualizing the position of decision boundaries, another objective is to determine which boundary is implemented by which hidden unit. For this purpose a similar first order sensitivity analysis can be used to visualize over which part of the input space a hidden unit is active. In this case, using a Taylor series expansion of the hidden unit activation y_j around input parameter values $z_i^{(p)}$ (similar to that in equation (3.3)), graphs of $\frac{\partial y_j}{\partial z_i^{(p)}}$ versus $z_i^{(p)}$ indicate the range of input values over which hidden unit y_j is active, hence giving an indication of the boundary implemented by hidden unit y_j. Here the sensitivity of hidden unit y_j with regard to input parameter perturbations is calculated using equation

(E.11) - assuming sigmoidal activation functions and a three layer architecture. Visualization of these sensitivity analysis results may reveal hidden units implementing the same boundary, thus duplicating function, and hidden units that learn no boundaries at all (for which $\frac{\partial y_j}{\partial z_i^{(p)}}$ will be approximately zero).

Find below two complete algorithms to visualize boundaries defined in the input space and to determine which hidden unit implements which boundary. First, the algorithm to visualize boundaries formed in the input space:

1. for each pattern $p = 1, \cdots, P$
   (a) for each input parameter z_i, $i = 1, \cdots, I$ and output unit o_k, $k = 1, \cdots, K$ compute
   $$\frac{\partial o_k}{\partial z_i^{(p)}}$$
   with one forward pass through the network, using equation (E.6) for sigmoid activation functions
   (b) plot $\frac{\partial o_k}{\partial z_i^{(p)}}$ versus $z_i^{(p)}$ for each input-output pair (z_i, o_k) on separate graphs

Next, an algorithm to visually inspect which decision boundary is implemented by which hidden unit, i.e. to find the point(s) in input space for which the hidden unit is most active:

1. for each pattern $p = 1, \cdots, P$
   (a) for each input parameter z_i, $i = 1, \cdots, I$ and hidden unit y_j, $j = 1, \cdots, J$ compute
   $$\frac{\partial y_j}{\partial z_i^{(p)}}$$
   with one forward pass through the network, using equation (E.10) for sigmoid activation functions
   (b) plot $\frac{\partial y_j}{\partial z_i^{(p)}}$ versus $z_i^{(p)}$ for each input-hidden pair (z_i, y_j) on separate graphs

The next section applies these visualization algorithms to artificial and real-world problems.

3.3 Experimental results

The purpose of this section is to illustrate, on artificial problems, how sensitivity analysis can be used to (1) visualize decision boundaries formed in the input space, and (2) to visually inspect which boundary has been implemented by which hidden unit. The section then reports results of the algorithms' application to one real-world problem. This approach to decision boundary visualization has been applied successfully by Engelbrecht and Viktor [Engelbrecht *et al* 1999a], Viktor, Engelbrecht and Cloete [Viktor *et al* 1998a] and Viktor [Viktor 1998b].

Simple artificial experiments have been chosen to illustrate specific aspects, and to compare the NN visualized boundaries with theoretically known boundaries. The results of only one simulation of each experiment are reported. For each experiment gradient descent training was used with sigmoid activation functions in the hidden and output layers. A learning rate of 0.1 and momentum of 0.9 were used for all experiments.

The first artificial problem is to discriminate between two classes using two dimensional input. One class is inside a circle of radius 0.5 centered at the origin, and the other class outside the circle, but bounded by a unit square. The classification rule is expressed as

$$\text{class} = \begin{cases} 0 & \text{if } \sqrt{z_1^2 + z_2^2} \leq 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

where z_1 and z_2 are the input parameters. A training set of 250 patterns and a test set of 150 patterns were randomly generated, sampling the inputs from an uniform distribution, i.e. $z_1, z_2 \sim U(-1, 1)$. The theoretical boundaries for this problem is illustrated in figure 3.3, using the patterns as contained in the training set. A 2-3-1 architecture was used to learn this classification rule, that is, two inputs, three hidden units and one output unit. Training was stopped after 500 epochs.

From equation (3.4), the theoretical boundaries for this problem are $z_1 = 0.5, z_1 = -0.5, z_2 = 0.5$ and $z_2 = -0.5$. This problem is used to illustrate

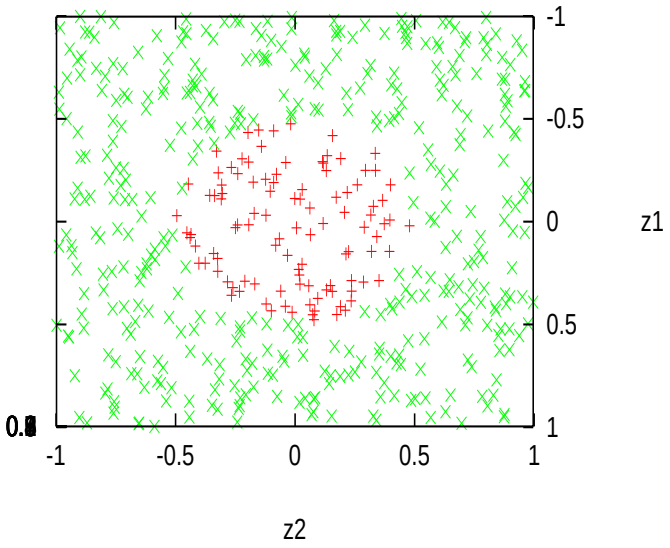
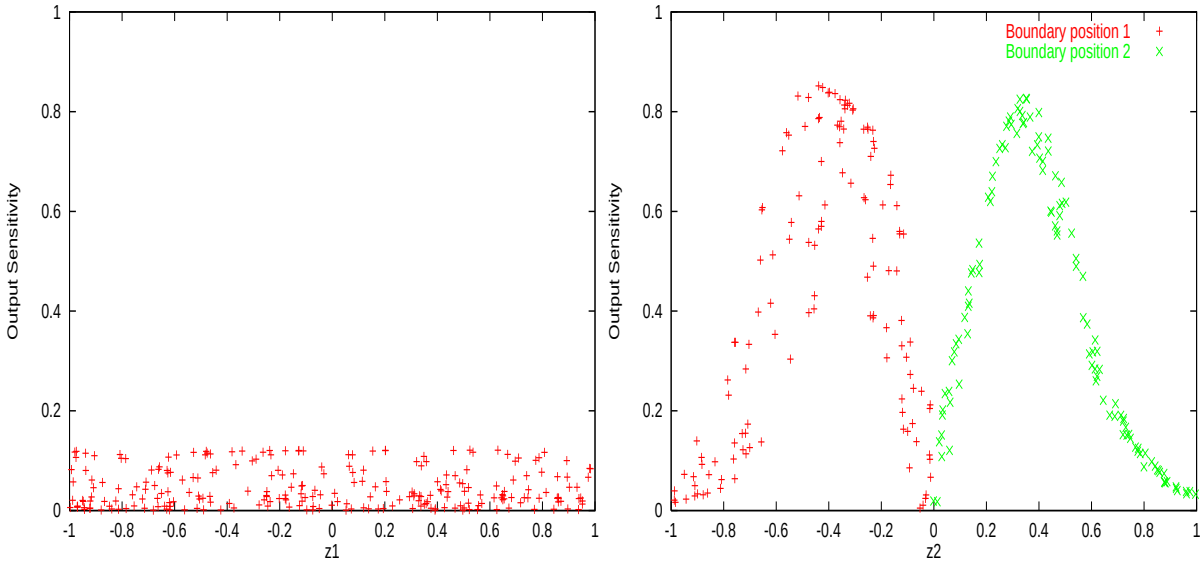


Figure 3.3: Circle classification problem defined in (3.4)



(a) Input z_1

(b) Input z_2

Figure 3.4: Boundary positions for circle classification problem after 50 epochs

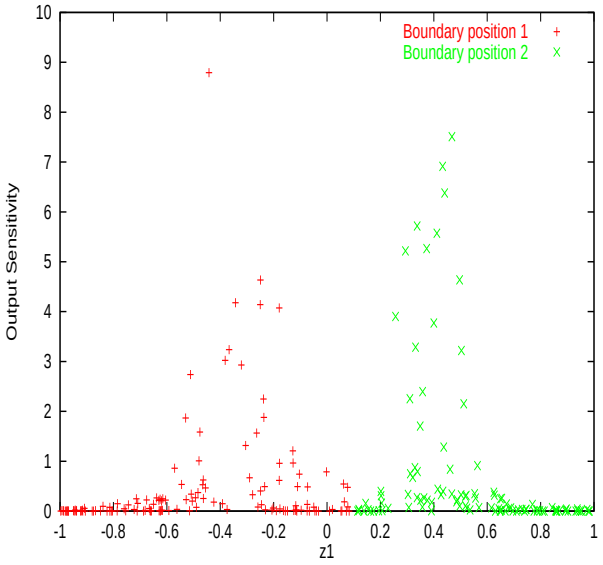


Figure 3.5: Boundary positions for circle classification problem after 500 epochs

1. that the visualized boundary positions are close to the theoretical boundaries;

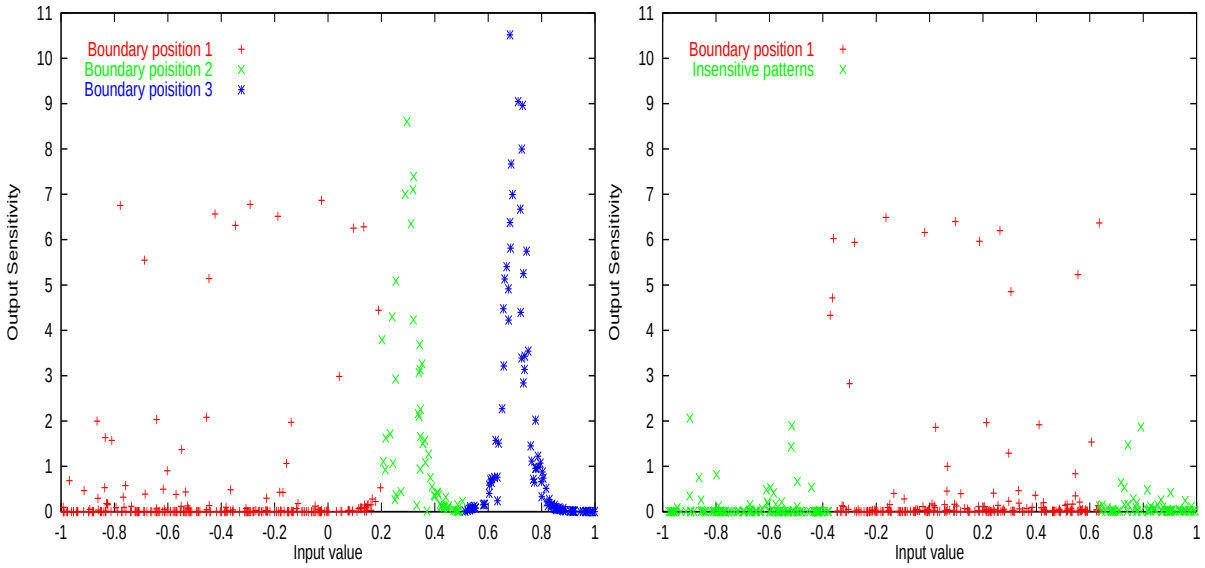
2. the visualization of axis-parallel decision boundaries.

Figures 3.4 and 3.5 visualize the position of the boundaries for this problem. These figures plot the values of $\frac{\partial o_k}{\partial z_i^{(p)}}$ for each pattern p. A different color is used for different boundaries to have a clear distinction between boundaries. Figure 3.4 illustrates the boundaries for inputs z_1 and z_2 after 50 epochs. At this point no boundaries have formed for z_1, while two clear axis-parallel boundaries formed for z_2. Note the peaks for input z_2 at approximately $z_2 = -0.4$ and $z_2 = 0.4$. Figure 3.5 illustrates two axis-parallel boundaries for z_1 after 500 epochs at approximately $z_1 = -0.45$ and $z_2 = 0.48$ (similar results were obtained for z_2). Figure 4.2(b) on page 89 displays a scatter plot that visualizes the learned decision boundaries after 500 epochs.

The second artificial problem implements the following classification rule:

$$\text{class} = \begin{cases} 1 & \text{if } (z_1 \geq 0.7) \text{ or } ((z_1 \leq 0.3) \text{ and } (z_2 \geq -0.2 - z_1)) \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

A training set of 400 patterns and a test set of 100 patterns were randomly created, with $z_1, z_2 \sim U(-1, 1)$. The theoretical boundaries for this problem are illustrated on the scatter plot in figure 3.1, on page 37. A 2-4-1 network was trained for 200 epochs.



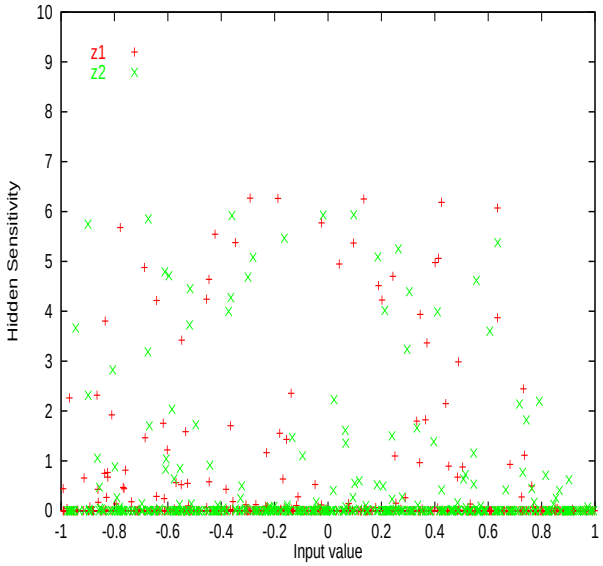
(a) Input z_1 (b) Input z_2

Figure 3.6: Artificial rule classification boundary positions after 200 epochs

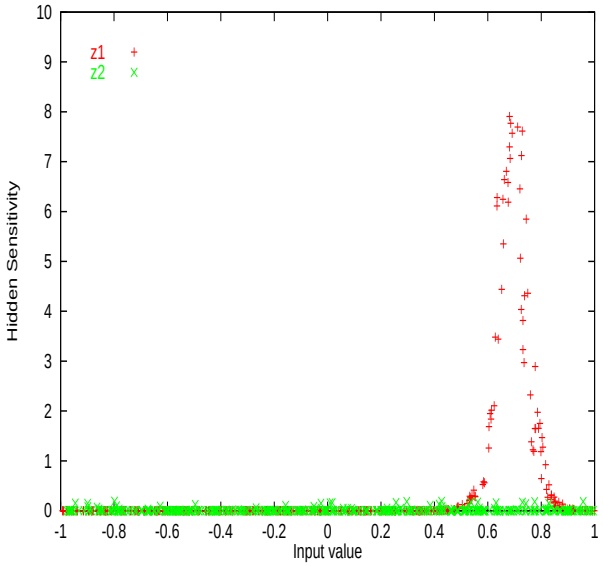
Classification rule (3.5) clearly defines axis-parallel decision boundaries at $z_1 = 0.7$ and $z_1 = 0.3$, and a non-axis-parallel boundary over $z_1 \in [-1, 0.3]$ and $z_2 \in [-0.4, 0.8]$. This problem is used to illustrate

1. that the visualized boundary positions accurately approximate these theoretical boundaries;
2. the visualization of axis-parallel and non-axis-parallel decision boundaries;
3. which boundaries are implemented by which hidden units;
4. hidden units that do not implement any boundaries (3 hidden units are sufficient for this problem).

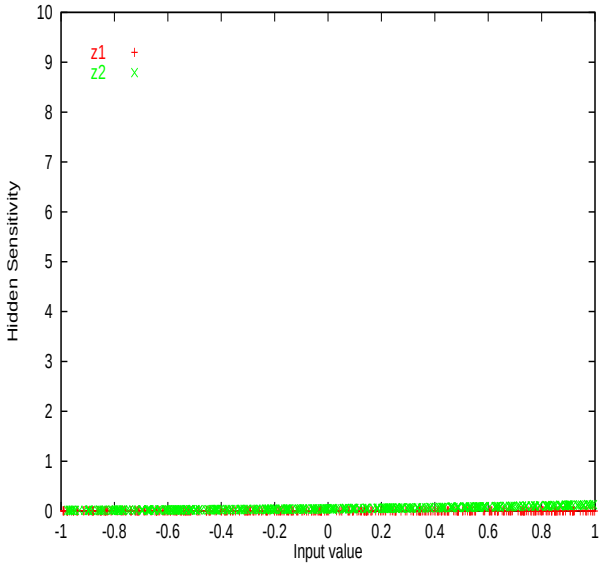
Figure 3.6 visualizes the boundary positions for inputs z_1 and z_2. Distinctive peaks formed at approximately $z_1 = 0.7$ and $z_1 = 0.3$, respectively for the two axis-parallel boundaries. Figure 3.6 also shows a range of high sensitivity values, with output sensitivity approximately 7, for $z_1 \in [-0.8, 0.3]$, and a high sensitivity range for $z_2 \in [-0.35, 0.65]$ - closely representing the non-axis-parallel boundary.



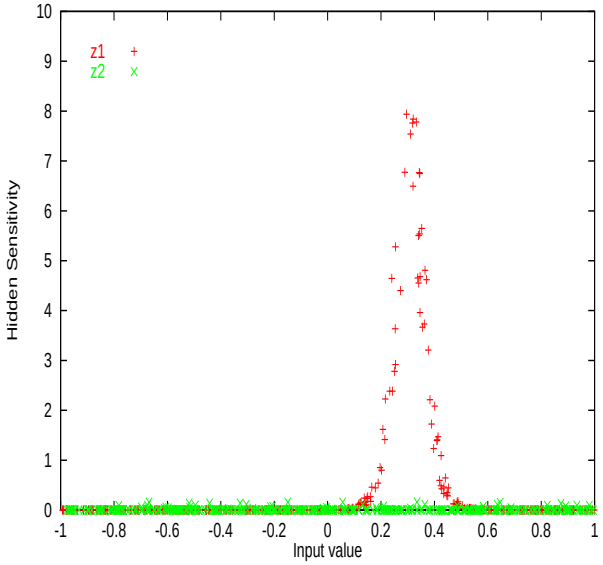
(a) Boundary implemented by hidden unit y_1



(b) Boundary implemented by hidden unit y_2



(c) Boundary implemented by hidden unit y_3



(d) Boundary implemented by hidden unit y_4

Figure 3.7: Artificial rule classification boundary implemented by each hidden unit

Figure 3.7 visualizes for each hidden unit its implemented decision boundary. From these figures hidden unit y_1 implements the non-axis-parallel boundary, with hidden unit sensitivity of approximately 6 (on the y-axis), which spans $z_1 \in [-0.8, 0.3]$ and $z_2 \in [-0.35, 0.65]$. Hidden y_2 implements the axis-parallel boundary at $z_1 = 0.7$, while y_4 implements the boundary at $z_1 = 0.3$. Hidden unit y_3 has sensitivity values of approximately zero for all patterns, indicating that y_3 implements no decision boundary and can therefore be pruned. The pruning algorithm in chapter 5 utilizes these pattern sensitivity information to select hidden units for pruning.

The last artificial problem is borrowed from [Belue *et al* 1995], which is a four-class problem using two dimensional input of which one is insignificant. A total of 600 patterns were drawn from four independent bivariate normal distributions. Classes were distributed according to

$$N_2\left(\mu = \begin{pmatrix} m_i \\ 0 \end{pmatrix}, \sum = \begin{bmatrix} 0.50 & 0.05 \\ 0.05 & 0.50 \end{bmatrix}\right) \tag{3.6}$$

for $i = 1, \cdots, 4$, where μ is the mean vector and $\sum$ is the covariance matrix; $m_1 = -3, m_2 = 0, m_3 = 3$ and $m_4 = 6$. For training purposes, 480 training patterns and 120 test patterns were used. Training was done with a 2-3-4 network, trained for 200 epochs. All inputs were scaled to the range $[-1, 1]$.

Figure 3.8 represents a scatter plot of the decision boundaries obtained from the training set. This figure illustrates three axis-parallel boundaries at $z_1 \approx 0.5, z_1 \approx 0.0$ and $z_1 \approx -0.4$. This experiment is specifically chosen to illustrate

1. the applicability of the visualization algorithm to multi-output problems;
2. that no boundary is located for the insignificant input parameter z_2;
3. which boundaries are implemented by which hidden units.

Figure 3.9 shows that no boundaries were formed for input z_2, indicating that this input can be removed. These graphs also illustrate axis-parallel boundaries at $z_1 = -0.4$ for output o_1, $z_1 = -0.4$ and $z_1 = 0$ for output o_2, $z_1 = 0$ and $z_1 = 0.5$ for output o_3, and $z_1 = 0.5$ for output o_4. Figure 3.10 illustrates that hidden unit y_1 implements the boundary at $z_1 = -0.4$, y_2 implements the boundary at $z_1 = 0$ and y_3 implements the boundary at $z_1 = 0.5$.

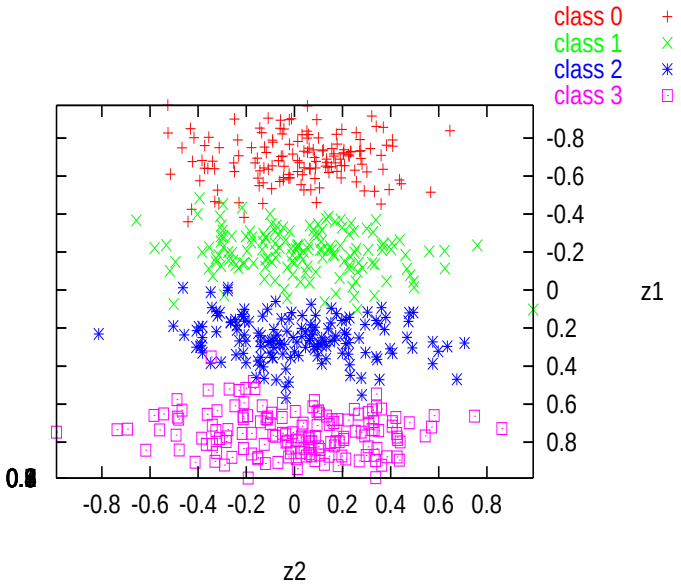
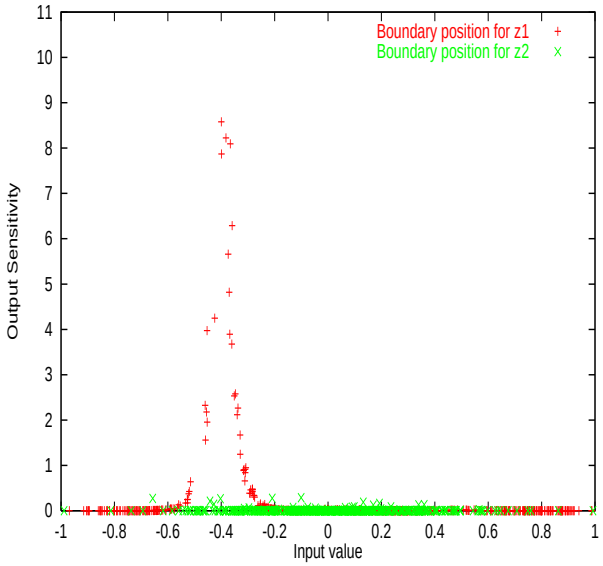


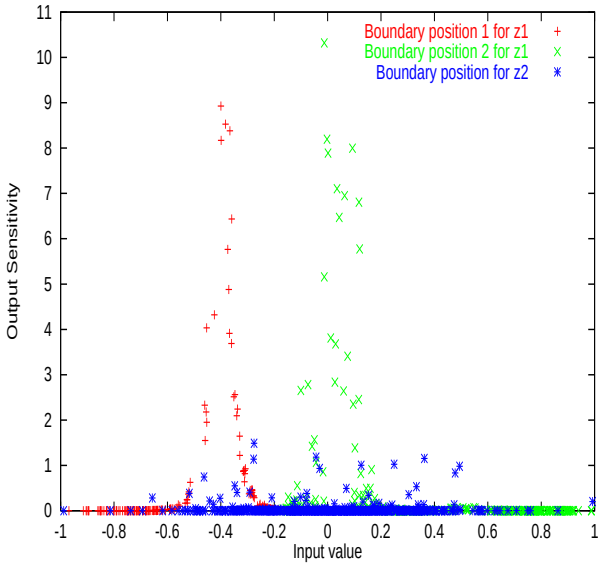
Figure 3.8: Four-class artificial classification problem defined in equation (3.6)

Next, results are shown of the application of the decision boundary visualization algorithm to a real-world problem. The iris classification problem (obtained from the UCI repository [UCI]) is used for this purpose. For the iris problem, a 4-2-3 architecture was used with a pruned architecture consisting of only 2 hidden units (refer to chapter 5 where the optimal architecture is determined), with a training set of 100 patterns and a test set of 50 patterns. Training was stopped when a 100% correct classification on the test set were obtained, using a learning rate of 0.1 and momentum of 0.9. Figure 3.11 visualizes the boundaries obtained for the output unit corresponding to class *iris versicolor* for the two input parameters *petal length* and *petal width*. No boundaries formed for the other two input units.

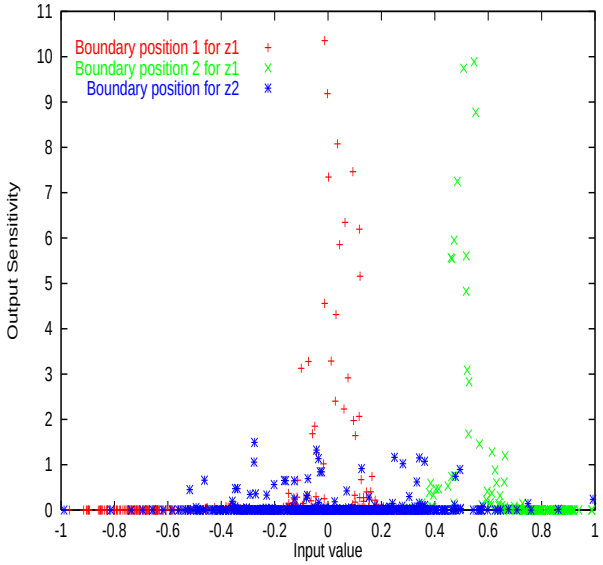
The boundaries illustrated in figure 3.11 were used as thresholds in a NN rule extraction algorithm, which resulted in a 2% increase in the accuracy of the extracted rules over the test set, with a 4% increase in the accuracy of the least accurate rule [Engelbrecht *et al* 1999a, Viktor *et al* 1998a, Viktor 1998b].



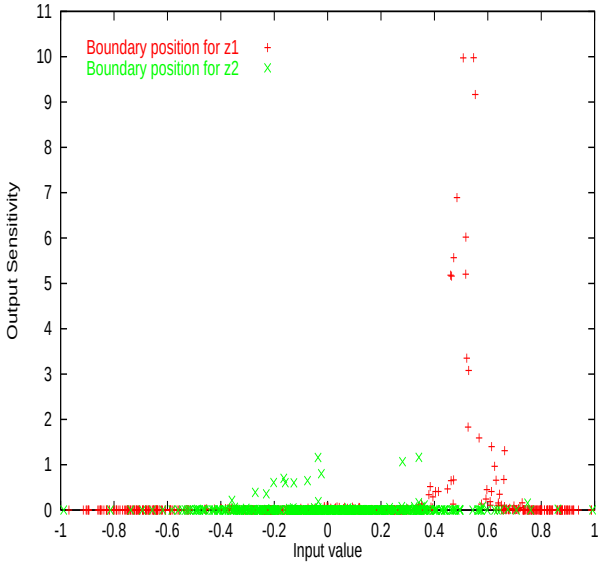
(a) Boundary position for output o_1



(b) Boundary positions for output o_2

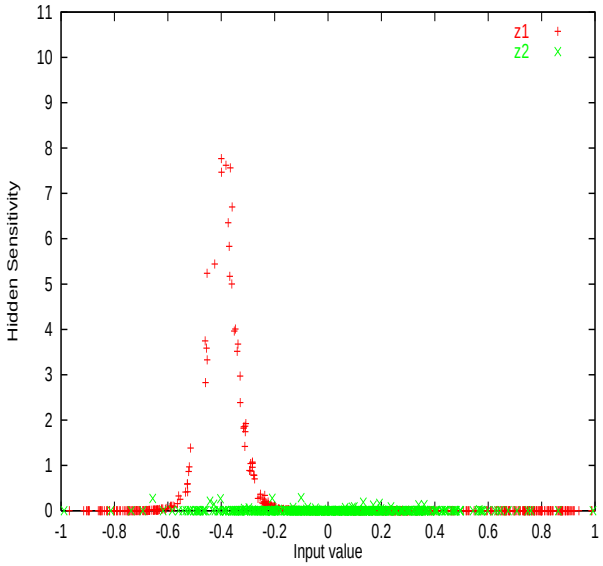


(c) Boundary positions for output o_3

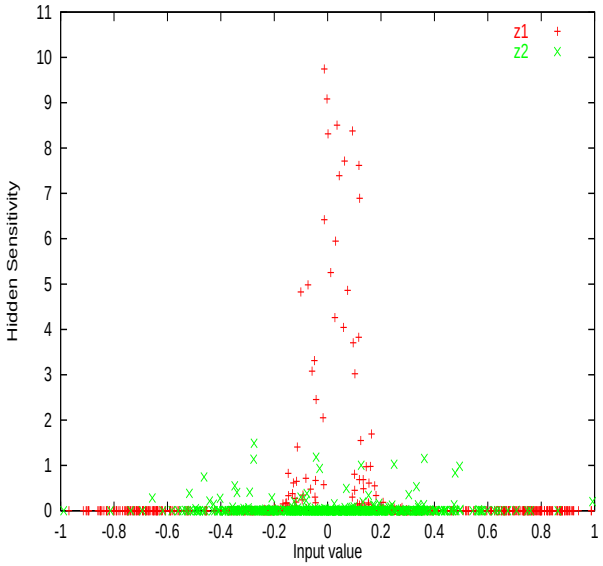


(d) Boundary position for output o_4

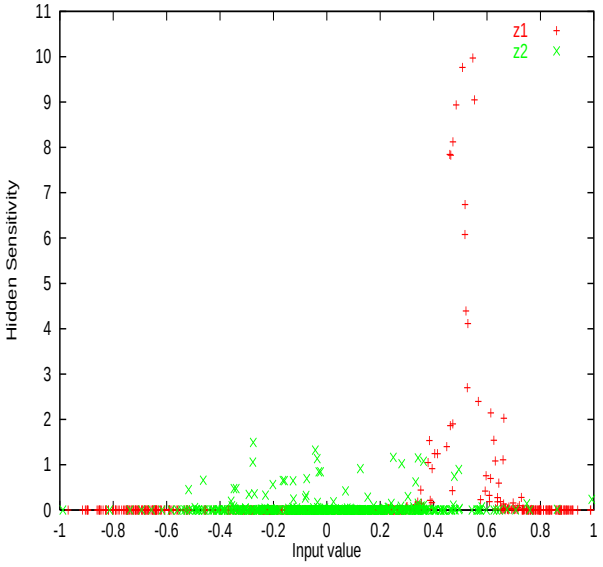
Figure 3.9: Four-class artificial problem boundary positions for z_1 and z_2



(a) Boundary implemented by hidden unit y_1

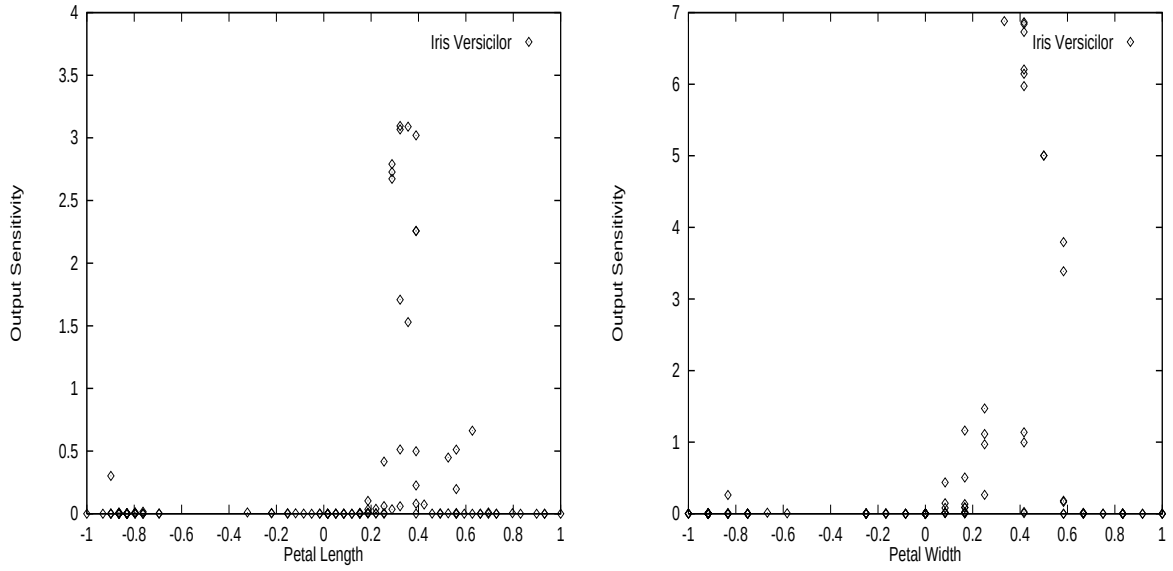


(b) Boundary implemented by hidden unit y_2



(c) Boundary implemented by hidden unit y_3

Figure 3.10: Four-class artificial problem boundary implemented by each hidden unit



(a) Boundary position over *petal length*

(b) Boundary position over *petal width*

Figure 3.11: Boundaries formed for *Iris Versicolor*

3.4 Conclusions

This chapter presented a simple idea to locate and visualize decision boundaries formed in the input space. Derived from a first order expansion of the Taylor series of an output unit function around values of an input parameter, sensitivity information was used to locate two types of decision boundaries that can be formed over an input parameter, referred to as axis-parallel and non-axis-parallel decision boundaries (viewed in one dimension only). Formal definitions for these decision boundary types were given, and their visualization illustrated on artificial problems. These experiments showed that the presented sensitivity analysis algorithm can effectively be used to

1. accurately visualize the position of decision boundaries
2. determine which hidden unit implements which boundary
3. visually identify irrelevant input parameters
4. visually identify hidden units that implement no boundaries at all.

Results on a real-world problem further illustrated the applicability of the decision boundary algorithm. The algorithm succeeded in extracting accurate boundaries which improved the accuracy of rules extracted by a NN rule extraction algorithm [Engelbrecht *et al* 1999a, Viktor *et al* 1998a]. For a more elaborate analysis of the efficiency of the decision boundary algorithm presented in this chapter, the reader is referred to [Viktor 1998b].

The objective of the chapter was simply to present the idea that sensitivity analysis can be used to locate patterns close to boundaries. This information will be used in the following chapters to further develop NN sensitivity analysis tools.

Chapter 4

Active Learning using Sensitivity Analysis

"What can be done with fewer is done in vain with more."
William of Ockham (1285-1349)

This chapter presents two new active learning algorithms which use output sensitivity information to dynamically select training patterns from a candidate set during training. A selective learning algorithm is developed for classification problems and an incremental learning algorithm for function approximation problems.

4.1 Introduction

Ockham's razor states that unnecessarily complex models should not be preferred to simpler ones - a very intuitive principle [MacKay 1992a, Thodberg 1991]. A neural network (NN) model is described by the network weights. Model selection in NNs consists of finding a set of weights that best performs the learning task. In this sense, this thesis views the data, and not just the architecture as part of the NN model, since the data is instrumental in finding the "best" weights. Model selection is then viewed as the process of designing an optimal NN architecture as well as the implementation of techniques to make optimal use of the available

training data. Following from the principle of Ockham's razor, is a preference then for both simple NN architectures and optimized training data. Usually, model selection techniques address only the question of which architecture best fits the task. This chapter explores the question of which data best describes the task. Sensitivity analysis data selection techniques are therefore investigated to make optimal use of the available training data. Chapter 5, on the other hand, considers architecture selection using sensitivity analysis.

Training data consists of input-target vector pairs, which is only a finite sample from the distribution describing the input space. The objective of NN training is to find a good approximation to the function that relates input vectors to corresponding target vectors, not only for the training space, but for the entire input space. That is, the trained NN should generalize well. This objective is achieved by iteratively adjusting weights using some optimization algorithm.

Gradient descent is one of the most popular optimization techniques used for NN training, resulting in the widely used backpropagation (BP) neural network [Battiti 1992, Rumelhart *et al* 1986]. Although backpropagation neural networks (BPNN) have been used successfully in many applications, they may suffer from problems inherent to gradient descent optimization. That is, convergence of standard BPNNs [Rumelhart *et al* 1986] tends to be slow, and they usually yield local optimum solutions. Consequently, generalization performance is reduced. These weaknesses are even more evident for recurrent neural networks [Cloete *et al* 1994a, Ludik 1995a], and is worsened even further when the training set is very large. In cases where non-monotonic activation functions are used, more local minima are introduced, and the chance of getting stuck in a local minimum is increased [Gaynier *et al* 1995].

Much research has been done to improve the generalization performance and training time of multi layer NNs. Research mostly concentrated on the optimal setting of initial weights [Denoeux *et al* 1993, Wessels *et al* 1992], optimal learning rates and momentum [Darken *et al* 1992, Magoulas *et al* 1997, Orr *et al* 1993, Salomon *et al* 1996, Vogl *et al* 1988, Weir 1990, Yu *et al* 1997, Jacobs 1989], finding optimal NN architectures using pruning techniques [Cibas *et al* 1996, Engelbrecht *et al* 1996, Engelbrecht *et al* 1999e, Fletcher *et al* 1998, Hassibi *et al* 1994, Karnin 1990, Le Cun 1990, Schittenkopf *et al* 1997, Zurada *et al* 1997, Reed 1993] (also see chapter 5) and construction techniques [Fritzke 1995, Hirose *et al* 1991,

Huang 1994, Kwok *et al* 1995, Lee 1991, Liang *et al* 1994, Lim *et al* 1994], sophisticated optimization techniques [Battiti 1992, Becker *et al* 1988, Cohen *et al* 1997, Møller 1993, Rosen *et al* 1997, Tang *et al* 1994], adaptive activation functions [Engelbrecht *et al* 1995a, Fletcher *et al* 1994, Maillard *et al* 1994, Weigl *et al* 1994, Zurada 1992a] (also refer to appendix C where a new automatic scaling algorithm is presented which dynamically changes the shape of the sigmoid activation function) and ensemble learning [Baxt 1992, Tumer *et al* 1996, Hashem *et al* 1994, Jacobs *et al* 1997, Kehagias *et al* 1997, Sollich *et al* 1996]. This chapter explores an alternative approach to improve generalization and training time, i.e. *active learning using sensitivity analysis.*

Standard error back-propagating NNs are passive learners. These networks passively receive information about the problem domain, randomly sampled to form a fixed size training set. Random sampling is believed to reproduce the density of the true distribution. However, more gain can be achieved if the learner is allowed to use current attained knowledge about the problem to guide the acquisition of training examples. As passive learner, a NN has no such control over what examples are presented for learning. The NN has to rely on the teacher (considering supervised learning) to present informative examples.

The generalization abilities and convergence time of NNs are much influenced by the training set size and distribution: Literature has shown that to generalize well, the training set must contain enough information to learn the task. Here lies one of the problems in model selection: the selection of concise training sets. Without prior knowledge about the learning task, it is very difficult to obtain a representative training set. Theoretical analyses provide a way to compute worst-case bounds on the number of training examples needed to ensure a specified level of generalization. A widely used theorem concerns the Vapnik-Chervonenkis (VC) dimension [Abu-Mostafa 1989, Abu-Mostafa 1993, Baum *et al* 1989, Cohn *et al* 1991, Hole 1996, Opper 1994]. This theorem states that the generalization error $\mathcal{E}_G$ of a learner with VC-dimension d_{VC} trained on P_T random examples will, with high confidence, be no worse than a bound of order d_{VC}/P_T. For NN learners, the total number of weights in a one hidden layer network is used as an estimate of the VC-dimension. This means that the appropriate number of examples to ensure an $\mathcal{E}_G$ generalization is approximately the number of weights divided by $\mathcal{E}_G$.

The VC-dimension provides overly pessimistic bounds on the number of training examples, often leading to an overestimation of the required training set size [Cohn *et al* 1991, Gu *et al* 1997, Opper 1994, Röbel 1994c, Zhang 1994]. Experimental results have shown that acceptable generalization performances can be obtained with training set sizes much less than that specified by the VC-dimension [Cohn *et al* 1991, Röbel 1994c]. Cohn and Tesauro show that for experiments conducted, the generalization error decreases exponentially with the number of examples, rather than the $1/P_T$ result of the VC bound [Cohn *et al* 1991]. Experimental results by Lange and Männer show that more training examples do not necessarily improve generalization [Lange *et al* 1994]. In their paper, Lange and Männer introduce the notion of a critical training set size. Through experimentation they found that examples beyond this critical size do not improve generalization, illustrating that an excess patterns have no real gain. The critical training set size is problem dependent.

While enough information is crucial to effective learning, too large training set sizes may be of disadvantage to generalization performance and training time [Lange *et al* 1996, Zhang 1994]. Redundant training examples may be from uninteresting parts of input space, and do not serve to refine learned weights - it only introduces unnecessary computations, thus increasing training time. Furthermore, redundant examples might not be equally distributed, thereby biasing the learner.

The ideal then, is to implement structures to make optimal use of available training data. That is, to select for training only informative examples, or to present examples in a way to maximize the decrease in training and generalization error. To this extent, active learning algorithms have been developed.

Cohn, Atlas and Ladner define *active learning* (also referred to in the literature as example selection, sequential learning, query-based learning) *as any form of learning in which the learning algorithm has some control over what part of the input space it receives information from* [Cohn *et al* 1994b]. An active learning strategy allows the learner to dynamically select training examples, during training, from a candidate training set as received from the teacher (supervisor). The learner capitalizes on current attained knowledge to select examples from the candidate training set that are most likely to solve the problem, or that will lead to a maximum decrease in error. Rather than passively accepting training examples from the

teacher, the network is allowed to use its current knowledge about the problem to have some deterministic control over which training examples to accept, and to guide the search for informative patterns. By adding this functionality to a NN, the network changes from a passive learner to an active learner.

With careful dynamic selection of training examples, shorter training times and better generalization may be obtained. Provided that the added complexity of the example selection method does not exceed the reduction in training computations (due to a reduction in the number of training patterns), training time will be reduced [Hunt *et al* 1995, Sung *et al* 1996, Zhang 1994]. Generalization can potentially be improved, provided that selected examples contain enough information to learn the task. Cohn [Cohn 1994a] and Cohn, Atlas and Ladner [Cohn *et al* 1994b] show through average case analyses that the expected generalization performance of active learning is significantly better than passive learning. Seung, Opper and Sompolinsky [Seung *et al* 1992], Sung and Niyogi [Sung *et al* 1996] and Zhang [Zhang 1994] report similar improvements. Results presented by Seung, Opper and Sompolinsky indicate that generalization error decreases more rapidly for active learning than for passive learning [Seung *et al* 1992].

This thesis identifies two main approaches to active learning, i.e. *incremental learning* and *selective learning*. Incremental learning starts training on an initial subset of a candidate training set. During training, at specified selection intervals (e.g. after a specified number of epochs, or when the error on the current training subset no longer decreases), further subsets are selected from the candidate examples using some criteria or heuristics, and added to the training set. The training set consists of the union of all previously selected subsets, while examples in selected subsets are removed from the candidate set. Thus, as training progresses, the size of the candidate set decreases while the size of the actual training set grows. Note that this thesis uses the term *incremental learning* to denote data selection, and should not be confused with the NN architecture selection growing approach. The term *NN growing* is used in this thesis to denote the process of finding an optimal architecture starting with too few hidden units and adding units during training.

In contrast to incremental learning, selective learning selects at each selection interval a new training subset from the original candidate set. Selected patterns are not removed from the

candidate set. At each selection interval, all candidate patterns have a chance to be selected. The subset is selected and used for training until some convergence criteria on the subset is met (e.g. a specified error limit on the subset is reached, the error decrease per iteration is too small, the maximum number of epochs allowed on the subset is exceeded). A new training subset is then selected for the next training period. This process repeats until the NN is trained to satisfaction.

The main difference between these two approaches to active learning is that no examples are discarded by incremental learning. In the limit, all examples in the candidate set will be used for training. With selective learning, training starts on all candidate examples, and uninformative examples are discarded as training progresses.

The rest of this chapter is organized as follows. Section 4.1.1 summarizes research related to active learning. Section 4.2 presents a general mathematical formulation of active learning, and learning aspects such as subset selection criteria, subset termination criteria and subset sizes are discussed. The concept of *pattern sensitivity* is introduced and pattern sensitivity norms are defined. The sensitivity of a patterns is then used as a measure of the *informativeness* of that pattern.

The *Sensitivity Analysis Selective Learning Algorithm* (SASLA) is presented in section 4.3. SASLA is developed specifically for selective learning of classification problems. A mathematical model is presented and the idea of pattern selection around decision boundaries is discussed. An algorithm is given, and complexity and convergence issues are addressed. Section 4.3.5 presents results and conclusions on the application of SASLA to artificial and real-world classification problems.

The *Sensitivity Analysis Incremental Learning Algorithm* (SAILA) is presented in section 4.4. SAILA is specifically developed for application to function approximation and time series problems. A mathematical model and algorithm are presented, and complexity and convergence issues are discussed. Results obtained from the application of SAILA to function approximation and time series problems are presented and discussed in section 4.4.5. The objective is to show that the sensitivity analysis active learning algorithms either improve on passive learning, or perform at least as good as passive learning. For this purpose, the

proposed active learning algorithms are compared to standard BP learning, using gradient descent to illustrate how the new learning algorithms efficiently address the problems of gradient descent. The term *fixed set learning* (FSL) is used to denote standard passive BP learning.

Section 4.5 draws conclusions on the efficiency and applicability of the presented active learning algorithms.

For notational convenience, a three layer NN architecture (input layer, one hidden layer and output layer) is assumed. The sensitivity analysis models presented in this thesis can, however, be easily extended to NN architectures which contain more than one hidden layer (see appendix E.1.1). Although the results presented in this chapter concentrate on gradient descent optimization, the sum squared error objective function and sigmoid activation functions, the sensitivity analysis models assume no specific optimization method, objective function or activation functions. The only assumptions are that the activation functions are at least once differentiable, and monotonic increasing. Note that optimal architectures are not necessarily assumed. The objective is to compare the performance of the different learning algorithms, irrespective of the optimality of that architecture.

How does this chapter fit into the objectives of the thesis? The chapter follows the main theme of exploring applications of sensitivity analysis to multilayer feedforward NNs. In this case, the application is data selection through active learning as part of the model selection process. The data selection algorithms presented also address the sub objectives of studying generalization performance, training time and convergence under the proposed sensitivity analysis applications.

4.1.1 Related Work

Several methods have been developed that manipulates the presentation of training patterns. These methods can be divided into two groups: **training set manipulation techniques** and **active learning techniques**. Training set manipulation techniques performs a pre-processing step in the training data to assign a specific order in which patterns will be presented for learning. This order is maintained during training, and does not change dynamically. Such training set manipulation techniques have as objectives to decrease training

time and to improve generalization performance through the preprocessing of the training set. Active learning algorithms dynamically changes the training sets during training.

Four different training set manipulation techniques are reviewed below, and it is shown, with reference to active learning definition 4.1 below, why these training set manipulation techniques are not considered as active learning algorithms. A survey of selective learning and incremental learning algorithms is then presented.

Definition 4.1 Active Learning: *Define active learning as any form of learning in which the learning algorithm has some deterministic control during training over what part of the input space it receives information from [Cohn et al 1994b].*

Training set manipulation techniques

Ohnishi, Okamoto and Sugie suggested a method called Selective Presentation where the original training set is divided into two training sets. One set contains typical patterns, and the other set contains confusing patterns [Ohnishi *et al* 1990]. With "typical pattern" the authors mean a pattern far from decision boundaries, while "confusing pattern" refers to a pattern close to a boundary. The two training sets are created once before training. Generation of these training sets assumes prior knowledge about the problem, i.e. where in input space decision boundaries are. In many practical applications such prior knowledge is not available, thus limiting the applicability of this approach. The Selective Presentation strategy alternately presents the learner with typical and then confusing patterns. The learner therefore has no control over the patterns presented for training, and the two sets remain fixed during training. Selective Presentation do not adhere to the definition of active learning, and is not viewed as an active learning algorithm.

Kohara developed Selective Presentation Learning for forecasting applications [Kohara 1995]. Before training starts, the algorithm generates two training sets. The one set contains all patterns representing large next-day changes, while patterns representing small next-day changes are contained in the second set. Large-change patterns are then simply presented more often than small-change patterns (similar to Selective Presentation). Again, the learner plays no role in the pattern selection process, and each training set remains fixed during training.

Selective Presentation Learning does not adhere to the definition of active learning.

Slade and Gedeon [Slade *et al* 1993] and Gedeon, Wong and Harris [Gedeon *et al* 1995] proposed Bimodal Distribution Removal, where the objective is to remove outliers from training sets during training. Frequency distributions of pattern errors are analyzed during training to identify and remove outliers. Although the NN uses current attained knowledge to prune outliers from the training set, this thesis does not consider Bimodal Distribution Removal as an active learning algorithm. It is rather a training set filtering algorithm. The NN still trains on all non-outlier training patterns whether they are informative or not. If the original training set contains no outliers, the method simply reduces to FSL with the added complexity of analyzing an error frequency distribution at each epoch.

Cloete and Ludik have done extensive research on *training strategies*. Firstly, they proposed Increased Complexity Training where a NN first learns easy problems, and then the complexity of the problem to be learned is gradually increased [Cloete *et al* 1993, Ludik *et al* 1993, Ludik 1995a, Ludik *et al* 1995b]. The original training set is split into subsets of increasing complexity before training commences. A drawback of this method is that the complexity measure of training data is problem dependent, thus making the strategy unsuitable for some tasks. Secondly, Cloete and Ludik developed *incremental training strategies*, i.e. Incremental Subset Training [Cloete *et al* 1994a, Ludik 1995a, Ludik *et al* 1995b] and Incremental Increased Complexity Training [Ludik *et al* 1994, Ludik 1995a, Ludik *et al* 1995b]. In Incremental Subset Training, training starts on a random initial subset. During training, random subsets from the original training set are added to the actual training subset. Incremental Increased Complexity Training is a variation of Increased Complexity Training, where the complexity ranked order is maintained, but training is not done on each complete complexity subset. Instead, each complexity subset is further divided into smaller random subsets. Training starts on an initial subset of a complexity subset, and is incrementally increased during training. Finally, Delta Training Strategies were proposed [Cloete *et al* 1994b, Ludik 1995a, Ludik *et al* 1995b]. With Delta Subset Training examples are ordered according to inter-example distance, e.g. Hamming or Euclidean distance. Different strategies of example presentations were investigated: smallest difference examples first, largest difference examples first, and alternating difference.

The training strategies proposed by Cloete and Ludik do not adhere to the definition of active learning. The learner has no control over the training subsets created, since subsets are created before training starts.

Selective learning

Not much research has been done in selective learning. Hunt and Deller developed Selective Updating, where training starts on an initial candidate training set [Hunt *et al* 1995]. Patterns that exhibit a high influence on weights, i.e. patterns that cause the largest changes in weight values, are selected from the candidate set and added to the training set. Patterns that have a high influence on weights are selected at each epoch by calculating the effect that patterns have on weight estimates. These calculations are based on matrix perturbation theory, where an input pattern is viewed as a perturbation of previous patterns. If the perturbation is expected to cause large changes to weights, the corresponding pattern is included in the training set. The learning algorithm does use current knowledge to select the next training subset, and training subsets may differ from epoch to epoch. Selective Updating has the drawback of assuming uncorrelated input units, which is often not the case for practical applications.

Another approach to selective learning is simply to discard those patterns that have been classified correctly [Barnard 1991, Hampshire *et al* 1990]. The effect of such an approach is that the training set will include those patterns that lie close to decision boundaries. If the candidate set contains outlier patterns, these patterns will, however, also be selected. This error selection approach therefore requires a robust estimator (objective function) to be used in the case of outliers.

Incremental learning

Research on incremental learning is more abundant than for selective learning. Most current incremental learning techniques have their roots in information theory, adapting Fedorov's optimal experiment design for NN learning [Cohn 1994a, Fukumizu 1996, MacKay 1992a, MacKay 1992b, Plutowski *et al* 1993, Sung *et al* 1996]. The different information theoretic incremental learning algorithms are very similar, and differ only in whether they consider

only bias, only variance, or both bias and variance terms in their selection criteria.

Cohn developed neural network Optimal Experiment Design (OED), where the objective is to select at each iteration a new pattern from a candidate set which minimizes the expectation of the mean squared error (MSE) [Cohn 1994a]. This is achieved by minimizing output variance as estimated from the Fisher information matrix [Cohn 1994a, Cohn *et al* 1996]. The model assumes an unbiased estimator and considers only the minimization of variance. OED is computationally very expensive because it requires the calculation of the inverse of the information matrix.

MacKay proposed similar Information-Based Objective Functions for active learning, where the objective is to maximize the expected information gain by maximizing the change in Shannon entropy when new patterns are added to the actual training set, or by maximizing cross-entropy gain [MacKay 1992a, MacKay 1992b]. Similar to OED, the maximization of information gain is achieved by selecting patterns that minimize the expected MSE. Information-Based Objective Functions also ignore bias, by minimizing only variance. The required inversion of the Hessian matrix makes this approach computationally expensive.

Plutowski and White proposed to select patterns that minimize the Integrated Squared Bias (ISB) [Plutowski *et al* 1993]. At each iteration, a new pattern is selected from a candidate set that maximizes the change, ΔISB, in the ISB. In effect, the patterns with error gradient most highly correlated with the error gradient of the entire set of patterns is selected. A noise-free environment is assumed and variance is ignored. Drawbacks of this method are the need to calculate the inverse of a Hessian matrix, and the assumption that the target function is known.

Sung and Niyogi proposed an information theoretic approach to active learning that considers both bias and variance [Sung *et al* 1996]. The learning goal is to minimize the expected misfit between the target function and the approximated function. The patterns that minimizes the expected squared difference between the target and approximated function are selected to be included in the actual training set. In effect, the net amount of information gained with each new pattern is then maximized. No assumption is made about the target function. This technique is computationally expensive, since it requires computations over two expectations,

i.e. the a-posteriori distribution over function space, and the a-posteriori distribution over the space of targets one would expect given a candidate sample location.

A drawback of the incremental learning algorithms summarized above is that they rely on the inversion of an information matrix. Fukumizu showed that, in relation to pattern selection to minimize the expected MSE, the Fisher information matrix may be singular [Fukumizu 1996]. If the information matrix is singular, the inverse of that matrix may not exist. Fukumizu continues to show that the information matrix is singular if and only if the corresponding NN contains redundant units. Thus, the information matrix can be made non-singular by removing redundant hidden units. Fukumizu developed an algorithm that incorporates an architecture reduction algorithm with a pattern selection algorithm. This algorithm is complex due to the inversion of the information matrix at each selection interval, but ensures a non-singular information matrix.

Approximations to the information theoretical incremental learning algorithms can be used. Zhang illustrates that information gain is maximized when a pattern is selected whose addition leads to the greatest decrease in MSE [Zhang 1994]. Zhang developed Selective Incremental Learning where training starts on an initial subset which is increased during training by adding additional subsets, where each subset contains those patterns with largest errors. Selective Incremental Learning has a very low computational overhead, but is negatively influenced by outlier patterns since these patterns have large errors.

Dynamic Pattern Selection, developed by Röbel [Röbel 1994a, Röbel 1994b, Röbel 1994c], is very similar to Zhang's Selective Incremental Learning. Röbel defines a generalization factor on the current training subset, expressed as $\mathcal{E}_G/\mathcal{E}_T$ where $\mathcal{E}_G$ and $\mathcal{E}_T$ are the MSE of the test set and the training set respectively. As soon as the generalization factor exceeds a certain threshold, patterns with highest errors are selected from the candidate set and added to the actual training set. Testing against the generalization factor prevents overfitting of the training subset. A low overhead is involved.

Very different from the methods described previously, are incremental learning algorithms for classification problems, where decision boundaries are utilized to guide the search for optimal training subsets. Cohn, Atlas and Ladner developed Selective Sampling, where patterns are

sampled only within a *region of uncertainty* [Cohn *et al* 1994b]. Cohn *et al* proposed an SG-network (most specific / most general network) as an approach to compute the region of uncertainty. Two separate networks are trained: one to learn a "most specific" concept s consistent with the given training data, and the other to learn a "most general" concept, g. The region of uncertainty is then all patterns p such that $s(p) \neq g(p)$. In other words, the region of uncertainty encapsulates all those patterns for which s and g present a different classification. A new training pattern is selected from this region of uncertainty and added to the training set. After training on the new training set, the region of uncertainty is recalculated, and another pattern is sampled according to some distribution defined over the uncertainty region - a very expensive approach. To reduce complexity, the algorithm is changed to select patterns in batches, rather than individually. An initial pattern subset is drawn, the network is trained on this subset, and a new region of uncertainty is calculated. Then, a new distribution is defined over the region of uncertainty that is zero outside this region. A next subset is drawn according to the new distribution and added to the training set. The process repeats until convergence is reached.

Query-Based Learning, developed by Hwang, Choi, Oh and Marks [Hwang *et al* 1991] differs from Selective Sampling in that Query-Based Learning generates new training data in the region of uncertainty. The objective is to increase the steepness of the boundary between two distinct classes by narrowing the regions of ambiguity. This is accomplished by inverting the NN output function to compute decision boundaries. New data in the vicinity of boundaries are then generated and added to the training set.

Seung, Opper and Sompolinsky proposed Query by Committee [Seung *et al* 1992]. The optimal training set is built by selecting one pattern at a time from a candidate set based on the principle of maximal disagreement among a committee of learners. Patterns classified correctly by half of the committee, but incorrectly by the other half, are included in the actual training set. Query by Committee is time consuming due to the simultaneous training of several networks, but will be most effective for ensemble networks.

The incremental learning algorithms reviewed in this section all make use of the NN learner's current knowledge about the learning task to select those patterns that are most informative. These algorithms start with an initial training set, which is increased during training by

adding a single informative pattern, or a subset of informative patterns.

In the next section a general formulation of active learning is presented.

4.2 Mathematical Model for Active Learning

One way of finding the optimal training set is to construct a power set of the candidate training set, and to train a NN on each element of this power set (each element is a subset of the candidate set). Although this approach will ultimately select the best training set, it is obviously impractical due to the enormous search space involved. A much more effective approach is active learning, where the NN uses its current knowledge to reduce the search space for an optimal training set, or training subset sequence. (Active learning does not just find an optimal training set, but also optimizes the sequence in which subsets is presented for training.)

This section presents a mathematical formulation for active learning using set theory. Further sections use the definitions and notations given in section 4.2.1. A general active learning algorithm is presented in section 4.2.2. This section also has as objective to introduce the concept of pattern informativeness which forms the basis of the development of the sensitivity analysis selective and incremental learning algorithms.

4.2.1 Mathematical Formulation

This section first introduces notations that will be used throughout the rest of this chapter and following chapters. Let $\vec{z}^{(p)} = (z_1^{(p)}, z_2^{(p)}, \cdots, z_I^{(p)})$ denote a specific input vector corresponding to pattern p with input dimension I, and each $z_i^{(p)} \in \mathbb{R}$. Vectors $\vec{t}^{(p)} = (t_1^{(p)}, t_2^{(p)}, \cdots, t_K^{(p)})$ and $\vec{o}^{(p)} = (o_1^{(p)}, o_2^{(p)}, \cdots, o_K^{(p)})$ denote the corresponding target vector and actual NN output vector respectively. The dimension of both $\vec{t}^{(p)}$ and $\vec{o}^{(p)}$ is K, and each $t_k^{(p)}, o_k^{(p)} \in \mathbb{R}$. Denote a single pattern p by the tuple $(\vec{z}^{(p)}, \vec{t}^{(p)})$.

The data set $D = \{(\vec{z}^{(p)}, \vec{t}^{(p)}) | p = 1, \cdots, P_D\}$ is the original, complete data set with $P_D = |D|$ patterns; $|D|$ is the cardinality of set D. The original data set is divided into a candidate training set D_C, a test set D_G and a validation set D_V, where $P_C = |D_C|$ is the total number

of patterns in the candidate set, $P_G = |D_G|$ is the number of test patterns, and $P_V = |D_V|$ is the total number of validation patterns. Furthermore, the sets are mutually disjoint, i.e. $D_C \cap D_G = \emptyset$ and $D_C \cap D_V = \emptyset$, and $D_C \cup D_G \cup D_V = D$.

The validation set is used during training to guard against overfitting. Training stops as soon as the accuracy as measured from the validation set deteriorates. The test set is used to measure the generalization performance of the NN.

Active learning algorithms parse through the candidate training set D_C to construct an actual training set D_T from training subsets D_{S_s} for $s = 1, 2, \cdots, S$, where $D_T \subseteq D_C$. Let $D_{S_1}, D_{S_2}, \cdots, D_{S_S}$ denote training subsets, where for each $s = 1, 2, \cdots, S$, $D_{S_s} = \{(\vec{z}^{(p)}, \vec{t}^{(p)}) | p = 1, \cdots, P_{S_s})\}$. Then, $D_{S_s} \subseteq D_C$, and each subset has $P_{S_s} = |D_{S_s}|$ patterns. D_{S_s} is the training subset selected at subset selection interval τ_s. If ξ_1 is the epoch number corresponding to subset selection interval τ_1, then $\xi_1 < \xi_2 < \cdots < \xi_S$, where S is the total number of subset selection intervals. An epoch is defined as one pass through the current actual training subset, D_T. D_{S_0} is the initial training subset.

If $\mathcal{F}_{NN}(D_T; W)$ represents the function learned by the NN, then D_T is simply the candidate training set, D_C, in the case of normal FSL. When SASLA is used, D_T is the current training subset, D_{S_s}, selected at selection interval τ_s. When SAILA is used, D_T is the union of all subsets, i.e. $D_T \leftarrow \bigcup_{s=1}^{s'} D_{S_s}$, where $\tau_{s'} \leq \tau_S$ is the current subset selection interval, and τ_S is the final selection interval. (The operator $\leftarrow$ denotes assignment.)

For the purposes of active learning, define the following active learning operators:

1) $\mathcal{A}^-(D_C, \mathcal{F}_{NN}(D_T; W)) = D_S$, where $D_S \subseteq D_C$. The operator $\mathcal{A}^-$ receives as input the candidate set D_C, performs some calculations on each pattern $p \in D_C$, and produces the subset D_S with the characteristics $D_S \subseteq D_C$, that is $|D_S| \leq |D_C|$. The objective of this operator is therefore to produce a subset D_S from D_C which is smaller than, or equal to, D_C. Then, let $D_T \leftarrow D_S$, where D_T is the actual training set.

2) $\mathcal{A}^+(D_C, D_T, \mathcal{F}_{NN}(D_T; W)) = D_S$, where D_C, D_T and D_S are sets such that $D_T \subseteq D_C$, $D_S \subseteq D_C$. The operator $\mathcal{A}^+$ performs calculations on each pattern $p \in D_C$ to determine if that element should be added to the current training set. Selected patterns are added to subset D_S. Thus, $D_S = \{p | p \in D_C,$ and p satisfies the selection criteria$\}$. Then, $D_T \leftarrow D_T \cup D_S$

(the new subset is added to the current training subset), and $D_C \leftarrow D_C - D_S$.

Active learning operator $\mathcal{A}^-$ corresponds with selective learning where the training set is "pruned," while $\mathcal{A}^+$ corresponds with incremental learning where the actual training subset "grows". Inclusion of the NN function $\mathcal{F}_{NN}$ as a parameter of each operator indicates the dependence on the NN's current knowledge.

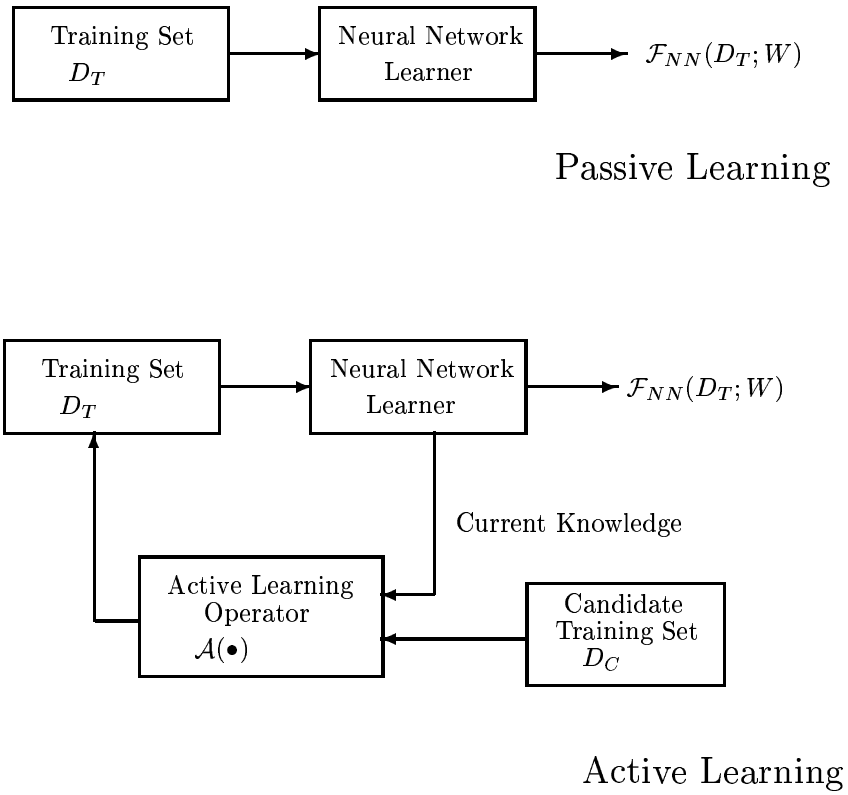


Figure 4.1: Passive vs Active Learning

4.2.2 General Active Learning Algorithm

Dynamic pattern selection and NN training are done interactively. In active learning, pattern selection is inextricably part of the learning algorithm. This is illustrated in figure 4.1. The top part of figure 4.1 illustrates passive learning, with no interaction to change the training set. The NN learner simply receives the fixed training set $D_T \leftarrow D_C$, trains on it, and produces the NN output function $\mathcal{F}_{NN}(D_T; W)$. The interaction between the learning algorithm and

the active learning operator to dynamically change the training set is illustrated in the bottom part of the figure. The operator $\mathcal{A}$ receives the learner's current knowledge, $\mathcal{F}_{NN}(D_T; W)$, and the candidate training set D_C to find a new informative training set.

In general, active learning is summarized by the following algorithm:

1. Initialize the NN architecture. Construct an initial training subset D_{S_0} from the candidate set D_C. Let $D_T \leftarrow D_{S_0}$.

2. Repeat

   (a) Repeat

   Train the NN on training subset D_T

   until local convergence on D_T is reached to produce the function $\mathcal{F}_{NN}(D_T; W)$.

   (b) Apply the active learning operator to generate a new subset D_{S_s} at subset selection interval τ_s, using either

   $$D_{S_s} \leftarrow \mathcal{A}^{-}(D_C, \mathcal{F}_{NN}(D_T; W)), \quad D_T \leftarrow D_{S_s}$$

   for selective learning, or

   $$\begin{aligned} D_{S_s} &\leftarrow \mathcal{A}^{+}(D_C, D_T, \mathcal{F}_{NN}(D_T; W)) \\ D_T &\leftarrow D_T \cup D_{S_s}, \quad D_C \leftarrow D_C - D_{S_s} \end{aligned}$$

   for incremental learning

until global convergence is reached.

The algorithm above refers to two convergence terms, i.e. *local convergence* and *global convergence*. Local convergence refers to convergence on the training subset D_T, as triggered by subset termination criteria. Local convergence is reached when the NN has learned the current training subset to satisfaction, or when no more gain can be achieved from the current subset. Global convergence refers to the criteria used to determine if the NN has learned the complete task to satisfaction. It is possible that local convergence also means global convergence.

Note that no reference is made to the optimization method, or to the NN architecture, illustrating the general active learning algorithm's independence of the training method and architecture.

Active learning design issues need to be addressed when developing an active learning algorithm, i.e. initial subset selection and size, subset selection criteria, subset termination criteria, and subset sizes. These issues are described briefly below, and will be revisited when the two sensitivity analysis active learning algorithms are presented.

Initial subset selection and size

The purpose of the initial subset is to initiate training. As soon as training starts, the NN starts to build its knowledge of the problem. Only then can pattern selection be applied effectively to refine the NN's knowledge.

There are basically two approaches to select an initial training subset, depending on the active learning algorithm. For selective learning, the initial subset is the candidate training set which will be pruned subsequently through application of the $\mathcal{A}^-$ operator. For incremental learning, a subset is selected from the candidate set such that $D_{S_0} \subset D_C$. The subset is selected through application of a heuristic or function of the candidate set. One way is just to select P_{S_0} random patterns from D_C according to some distribution law (i.e. uniform, normal). Prior knowledge about the candidate set, if available, can also be used to construct a distribution function according to which initial patterns will be selected. Alternatively, patterns can be selected according to some function defined over input space, making use of the initial network state as captured by the initial weights. For example, SAILA uses a function based on the sensitivity of the NN output to small pattern perturbations, using the

NN's initial weights. Although the NN has no self-attained knowledge at this point, the initial weights do provide information about the patterns that mostly influence the NN output in this initial state. Training will therefore start on a set of patterns that has the largest influence on the NN output, and consequently in weight changes (refer to section 4.2.3 where this is explained).

The initial subset size depends on the active learning algorithm. Selective learning, for example SASLA (presented in section 4.3) and Selective Updating [Hunt *et al* 1995], starts on all candidate training patterns. Incremental learning can start on a single pattern, or a small subset of patterns. The size of the initial subset is not that crucial. However, the following should be kept in mind. If the initial training subset is too large, not much gain will be achieved with incremental learning, since the learner will be offered a limited chance to use its knowledge in the search for the most informative patterns. An incremental learning algorithm should rather use a small subset of the candidate set. Provided that measures are implemented to prevent overfitting on subsets, the initial subset size will be increased with new patterns when no further gain can be achieved from that set. A smaller subset may mean more subset selection intervals, but, as indicated by research results, does not adversely affect training time [Cloete *et al* 1994b, Hunt *et al* 1995, Ludik *et al* 1993, Sung *et al* 1996, Zhang 1994] (also see section 4.4.5 for results that illustrate this fact).

Subset termination criteria

Learning each training subset to the same accuracy as required for the complete set of training data may cause overfitting of the subsets. The network may converge to an unacceptable local optimum solution, instead of generalizing to a global optimum solution. The efficiency of active learning algorithms relies on subset termination criteria to signal the selection of a new subset. The term *subset selection interval* refers to the "time" (epoch) at which a termination criterion is triggered and a next subset is selected.

The first criterion that springs to mind is to select a new subset at each epoch, which is done in OED [Cohn 1994a]. This can be a time consuming process, especially if the subset selection criteria is as complex as in OED (requiring the inversion of the Fisher information

matrix). Instead of selecting a new subset at each epoch, subsets can be selected at each ξ epochs. However, subset selection at specified epoch intervals has potential drawbacks. If the epoch selection interval is too large and no overfitting measures are implemented, overfitting may occur on the current subset, resulting in bad generalization. Less severely, the training interval may be too small to learn the current subset to satisfaction, therefore limiting the gain achieved from exploiting the NN's current knowledge of the problem.

A subset termination criterion should rather include some reference to the error on the training subset. Plutowski and White suggest to repeat training on the current subset until no more gain can be achieved (the training error no longer decreases) [Plutowski *et al* 1993]. Alternatively, they suggest to define a tolerance for each subset. As soon as the error on the subset is lower than the specified subset tolerance, a new subset is selected. In their training strategies, Ludik and Cloete propose to linearly decrease the initial network error until the required error on all patterns is reached [Ludik 1995a, Ludik *et al* 1995b]. A required error, E_{S_s}, is defined for each subset D_{S_s} as a function of an error decrement, $E_{decrement}$. Let $E_{S_0} = \mathcal{E}(D_{S_0}; W_0)$ be the initial error of the network before training. The error function $\mathcal{E}(D; W)$ can be any measure of the accuracy of the NN fit, e.g. MSE. Then, the error decrement is calculated as

$$E_{decrement} = \frac{E_{S_0} - E_C}{P_C / P_{S_s}}$$

where E_C is the required error on the entire candidate set, P_C and P_{S_s} are the number of patterns in the candidate set and the subset respectively. The required error for subset s is then computed as

$$E_{S_s} = E_{S_0} - ((s-1) \times E_{decrement})$$

Training on the current subset stops as soon as the error is less, or equal to, E_{S_s}.

Zhang proposes a termination criterion based on the fact that learning capacity is proportional to the total number of weights in the NN [Zhang 1994]. Training on a subset continues until the error is less than the required error per connection (weight). Let τ be the allowable error tolerance per connection. Then, training on the current training subset D_T repeats until

$$E_T = \mathcal{E}(D_T; W) \leq \frac{J(I+K)}{\tau}$$

where I, J and K are respectively the number of input, hidden and output units. Zhang

found a value of $\tau \in [100, 200]$ to be reasonable.

The termination criteria proposed by Ludik and Cloete, and Zhang, do not directly refer to generalization. Overfitting may therefore still occur. Röbel proposed a criterion that validates generalization performance of the current training set, thereby preventing overfitting on subsets [Röbel 1994a, Röbel 1994b, Röbel 1994c]. Röbel defines the generalization factor $\rho = \frac{E_V}{E_T}$, where E_V and E_T are the MSE on the validation set D_V and current training subset D_T respectively. The generalization factor indicates the error made in training on D_T only, instead of training on the entire input space. By requiring that $\rho \leq 1.0$, overfitting is prevented. Using Röbel's criterion, a new training subset is constructed when $\rho(\xi) > \varphi_\rho(\xi)$, where $\varphi_\rho(\xi) = \min\{\varphi_\rho(\xi-1), \overline{\rho}+\sigma_\rho, 1.0\}$; ξ is the current epoch, $\overline{\rho}$ is the average generalization factor over a fixed number of preceding epochs, and σ_ρ is the standard deviation.

Keep in mind that ρ does not give an indication of the accuracy of learning, but only the ratio between the training and validation error. For function approximation problems (as is the case with Röbel's work) where the MSE is used as measure of accuracy, a generalization factor $\rho < 1$ means that the validation error is smaller than the training error - which is desirable. As ρ becomes large (greater than 1), the difference between the training error and validation error increases, which indicates an increase in validation error with a decrease in training error - an indication of overfitting. For classification problems where the percentage correctly classified patterns is used as measure of accuracy, ρ should be larger than 1.

In addition to subset termination criteria, the normal global convergence criteria on the entire set of training data are also employed. If at any time any one of the global termination criteria is triggered, training stops.

While this section reviewed published termination criteria, later sections elaborate on the subset termination criteria used for this study.

Subset selection criteria

The subset selection criterion is embodied in the active learning operator as defined in section 4.2.1. The selection criterion is the rule, heuristic or function according to which patterns

are selected from the candidate set, and forms the base of the active learning model. For example, for Selective Updating the selection criterion is to select those patterns that is expected to cause the largest changes in weights. For the minimization of the integrated squared bias, the selection criterion is to select patterns that maximizes the decrement in ISB. The selection criterion for SASLA and SAILA, which are discussed in sections 4.3 and 4.4 respectively, is to select those patterns that cause the output to be most sensitive to small perturbations in the input vector.

Subset size

The subset size generally depends on the active learning operator. The size P_{S_s} of a subset may be a fixed number of patterns for each selection interval, or the number of patterns may vary between selection intervals. For example, in OED only one new pattern is selected, Selective Incremental Learning selects subsets of a specified fixed size, for SASLA subset sizes are a function of the average pattern informativeness, and for SAILA only one pattern is selected.

Subset sizes do have an influence on the efficacy of the active learning algorithm. For incremental learning, too large subset sizes may cause too few selection intervals to optimally utilize the NN's knowledge. Smaller subsets give more opportunity to select informative patterns, but increase the number of selection intervals. For computationally expensive selection criteria, for example methods based on information theory, too many selection intervals may adversely affect convergence times.

4.2.3 Pattern Informativeness

This section introduces the basic idea of sensitivity analysis active learning algorithms. SASLA and SAILA are based on, and built upon, the concept of *pattern informativeness*. A pattern that has a negligible effect on the NN outputs is said to be uninformative for learning purposes, while informative patterns have a strong influence on the NN outputs.

Definition 4.2 Pattern Informativeness: *Define the informativeness of a pattern as the sensitivity of the NN output vector to small perturbations in the input vector. Let* $\Phi^{(p)}$ *denote the informativeness of pattern p. Then,*

$$\Phi^{(p)} \doteq ||\vec{S}_o^{(p)}|| \tag{4.1}$$

where $\vec{S}_o^{(p)}$ *is the output sensitivity vector for pattern p (defined in (4.3)), and* $||\bullet||$ *is any suitable norm.*

This study suggests the maximum-norm,

$$\Phi_\infty^{(p)} = ||\vec{S}_o^{(p)}||_\infty = \max_{k=1,\cdots,K}\{|S_{o,k}^{(p)}|\} \tag{4.2}$$

where $S_{o,k}^{(p)}$ refers to the sensitivity of a single output unit o_k to changes in the input vector $\vec{z}$. In equations (4.1) and (4.2), the output sensitivity vector is defined as

$$\vec{S}_o^{(p)} = ||S_{oz}^{(p)}|| \tag{4.3}$$

where $S_{oz}^{(p)}$ is the output-input layer sensitivity matrix. Each element $S_{oz,ki}^{(p)}$ of the sensitivity matrix is computed using equation (E.6) in appendix E for sigmoid activation functions. Suitable norms for calculating the output sensitivity vector are the sum-norm, or the Euclidean-norm. For each element k of $\vec{S}_o^{(p)}$,

Sum-norm:

$$S_{o,k}^{(p)} = ||S_{oz}^{(p)}||_1 = \sum_{i=1}^{I} |S_{oz,ki}^{(p)}| \tag{4.4}$$

Euclidean-norm:

$$S_{o,k}^{(p)} = ||S_{oz}^{(p)}||_2 = \sqrt{\sum_{i=1}^{I} (S_{oz,ki}^{(p)})^2} \tag{4.5}$$

Using definition 4.2 and equation (4.2), a pattern is considered informative if any one, or more, of the output units is sensitive to small perturbations in the input vector. The larger the value of $\Phi_\infty^{(p)}$, the more informative is pattern p. To illustrate this idea, consider the weight update equations in appendix D. From equations (D.14) and (D.21),

$$\Delta w_{kj}, \Delta v_{ji} \propto (t_k^{(p)} - o_k^{(p)}) \tag{4.6}$$

Each new pattern can be viewed as a perturbation of a previously presented pattern. Let $\Phi_{\infty}^{(p)} = |S_{o,k}^{(p)}|$. Then, if $\Phi_{\infty}^{(p)}$ is large, the output of unit o_k is significantly influenced, and the value of $(t_k^{(p)} - o_k^{(p)})$ changes significantly from the previous presentation. On the other hand, if $\Phi_{\infty}^{(p)}$ is small, no significant change in the output value of unit o_k will occur from the previously presented pattern. That is, the value of $(t_k^{(p)} - o_k^{(p)})$ does not change much, making pattern p an insignificant contributor to the determination of the gradient direction - and therefore being uninformative to the learning process.

The sensitivity analysis active learning algorithms presented in this chapter use pattern informativeness as subset selection criterion. The active learning operators therefore incorporate pattern informativeness to select training subsets. The NN knowledge used, is the sensitivity measures calculated from the learned weights.

4.3 Sensitivity Analysis Selective Learning

Selective learning is an active learning strategy that effectively "prunes" the original training set during training. The NN uses its current learned knowledge to select at each selection interval a subset of informative patterns from the candidate training set. Training commences on the training subset until subset termination criteria are triggered, upon which a new training subset is selected.

Selective learning algorithms certainly make sense for application to classification problems only. In classification problems, a pattern can be identified as being classified correctly or not. If the learner is already sure of the classification of a pattern, there is no need to re-learn that pattern. However, during the learning process, the learner may become uncertain about a previously correct classification, in which case the corresponding pattern should be brought back into the training subset. A selective learning algorithm should therefore have a good understanding of what information must be used for training, and what information can be overlooked. It certainly makes sense that patterns which are most likely to help the NN solve the problem must be preserved during training. For classification problems, the objective of NN training is to find optimum decision boundaries in input space that give good generalization. The patterns that are more likely to contribute to this objective are patterns

in a region close to a boundary - referred to as the *region of uncertainty* (see page 65). In fact, studies have shown that training on patterns near boundaries generalizes better than networks trained on the same number of randomly chosen examples [Ahmad *et al* 1989, Baum 1991, Cohn *et al* 1994b, Hwang *et al* 1991, Ohnishi *et al* 1990, Zhang 1994].

This section presents the Sensitivity Analysis Selective Learning Algorithm (SASLA), which uses sensitivity analysis to select patterns in the region of a decision boundary. First order derivatives of the output units with respect to input units are used to determine how close a pattern lies to a decision boundary, using the model presented in chapter 3. The most informative patterns lie closest to the boundaries. SASLA was developed with classification problems in mind. However, it can also be applicable to function approximation problems. Future research will investigate SASLA application to function approximation and time series prediction.

To the author's knowledge, this is the only study that utilizes NN output sensitivity analysis for selective learning purposes. Closely related to this approach, is Selective Updating developed by Hunt and Deller [Hunt *et al* 1995], which uses principles from matrix perturbation theory to assess the effect patterns have on weight estimates (also refer to page 62). Also, selection around decision boundaries can be achieved by selecting only those patterns that are not yet correctly classified.

Section 3 illustrates how sensitivity analysis can be used to determine if a pattern lies close to a decision boundary using the decision boundary visualization ideas presented in chapter 3. Section 4.3.2 presents a short description of the mathematical model underlying SASLA, utilizing the characteristics of decision boundaries. A pseudocode algorithm of SASLA is given in section 4.3.3, and its complexity is discussed in section 4.3.4. Section 4.3.5 presents a detailed discussion of the experiments conducted, and compares SASLA with FSL. This section also compares SASLA with the selection of unclassified patterns.

4.3.1 Decision Boundaries

The objective of a NN classifier is to construct optimal decision boundaries over input space. Active learning algorithms which sample from a region around decision boundaries have been

shown to refine boundaries, resulting in improved generalization performance. Paramount to the success of decision boundary active learning algorithms, is the method used to detect boundaries - if too complex, the model will be impractical.

Chapter 3 reviewed some approaches to decision boundary detection and presented a sensitivity analysis approach to find boundaries. The sensitivity analysis selective learning algorithm uses these first order derivatives of the NN outputs with regard to the inputs to determine if a pattern lies close to a boundary. SASLA does not detect decision boundaries *per se*, but uses sensitivity analysis to assign a "measure of closeness to boundaries" for each pattern [Engelbrecht *et al* 1998a, Engelbrecht *et al* 1999a, Viktor *et al* 1998a]. Patterns close to decision boundaries are defined to be informative. Section 3.2 explained that the value of $\frac{\partial o_k}{\partial z_i^{(p)}}$ is used to find the position of decision boundaries. Patterns with high sensitivity, i.e. high $\frac{\partial o_k}{\partial z_i^{(p)}}$, lie closest to decision boundaries. These patterns contain the most information for learning purposes.

Other learning algorithms have been developed that use different approaches to select patterns near decision boundaries. Ohnishi, Okamoto and Sugie use known characteristics of the problem to select patterns around the decision boundaries once before training [Ohnishi *et al* 1990]. Selective Sampling, developed by Cohn, Atlas and Ladner, uses distribution information from the environment to calculate a region of uncertainty around boundaries, and select patterns from these regions only [Cohn *et al* 1994b]. Baum finds boundaries using an iterative search and selects patterns in the region of these boundaries [Baum 1991]. In Query-Based Learning, a NN inversion algorithm is used together with selective sampling to generate new patterns around the decision boundaries to further refine these boundaries [Hwang *et al* 1991].

4.3.2 Mathematical Model

Using the notations and definitions introduced in section 4.2.1, this section defines the SASLA operator, $\mathcal{A}^{-}_{SASLA}$, and shows how the operator is used to generate a new training subset. Define the SASLA operator as

$$\mathcal{A}^{-}_{SASLA}(D_C, \mathcal{F}_{NN}(D_T; W)) = \{p \in D_C | \Phi_{\infty}^{(p)} > \Psi(\vec{\Phi}_{\infty})\} \tag{4.7}$$

where $\Phi_\infty^{(p)}$ is defined in equation (4.2), and $\vec{\Phi}_\infty$ is the vector

$$\vec{\Phi}_\infty = (\Phi_\infty^{(1)}, \ldots, \Phi_\infty^{(p)}, \ldots, \Phi_\infty^{(P_C)}) \tag{4.8}$$

where P_C is the total number of patterns in the candidate set.

The function Ψ implements the rule used to select patterns. For the SASLA implementation,

$$\Psi(\vec{\Phi}_\infty) = (1-\beta)\overline{\Phi}_\infty \tag{4.9}$$

where β is the subset selection constant (discussed below), and $\overline{\Phi}_\infty$ is the average pattern informativeness,

$$\overline{\Phi}_\infty = \frac{\sum_{p=1}^{P_C} \Phi_\infty^{(p)}}{P_C} \tag{4.10}$$

Patterns with informativeness a factor larger than the average informativeness over all patterns are therefore selected.

The subset selection constant β is crucial to the efficacy of the algorithm. This selection constant, which lies in the range $[0, 1]$, is used to control the region around decision boundaries within which patterns will be considered as informative. The larger the value of β, the more patterns will be selected. If β is too small, only a few patterns will be selected which may not include enough information to form boundaries, with a consequent reduction in generalization performance. Low values form β will however mean less computational costs. A conservative choice of β close to 1 improves the chances of selecting patterns representing enough information about the target concept, ensuring most of the candidate patterns to be included in the initial training subset. A conservative value for β does, however, not mean a small reduction in training set size. As training progresses, more and more patterns become uninformative, resulting in larger reductions in training set size. Section 4.3.5 shows that for such a conservative choice of β, the training set is substantially reduced early in training. The effects of different values for β are also investigated. If $\beta = 1$, SASLA simply generalizes to normal FSL.

At each subset selection interval τ_s (corresponding to epoch ξ_s), let the new subset be $D_{S_s} \leftarrow \mathcal{A}^-_{SASLA}(D_C, \mathcal{F}_{NN}(D_T; W))$. That is, select from the original candidate set D_C the subset $D_{S_s} \subseteq D_C$. Then, let D_T be the subset D_{S_s} for the next training interval. Note that, after subset selection, the NN does not train on the difference $D_C - D_{S_s}$. The actual training set for the current training interval is thus reduced by $|D_C - D_{S_s}|$ patterns.

4.3.3 Selective Learning Algorithm

The sensitivity analysis selective learning algorithm is outlined below. Note that the algorithm makes no reference to a specific optimization method or objective function. Although SASLA depends on the choice of activation functions (requiring differentiable functions) and NN architecture, the formulation is general to illustrate the algorithm's applicability to any activation function and architecture. It is only step 2.b.i, where the sensitivity matrix for a pattern is calculated, that changes for different activation functions and architectures.

1. Initialize weights and learning parameters. Initialize the pattern selection constant, $\beta = 0.9$ for a conservative choice. Construct the initial training subset, $D_{S_0} \leftarrow D_C$. Let $D_T \leftarrow D_{S_0}$.

2. Repeat

   (a) Repeat

   Train the NN on training subset D_T

   until a termination criterion on D_T is triggered.

   (b) Compute the new training subset D_{S_s} for the next subset selection interval τ_s:

   i. For each $p \in D_C$, compute the sensitivity matrix $S_{oz,ki}^{(p)}$ using equation (E.6) for sigmoid activation functions.

   ii. Compute the output sensitivity vector $\vec{S}_o^{(p)}$ for each $p \in D_C$ from equation (4.3).

   iii. Compute the informativeness $\Phi^{(p)}$ of each pattern $p \in D_C$ using equation (4.2).

   iv. Compute the average pattern informativeness from equation (4.10).

   v. Apply operator $\mathcal{A}_{SASLA}^{-}$ in equation (4.7) to find the subset D_{S_s} of most informative patterns. Then, let $D_T \leftarrow D_{S_s}$.

until global convergence is reached.

Design issues specific to SASLA are discussed next:

- **Initial subset selection and size**: SASLA starts training on the entire candidate training set. The initial training subset is therefore just D_C, consisting of $P_C = |D_C|$ patterns.
- **Subset termination criteria**: The original implementation of SASLA selects a new training subset at each epoch, allowing for many opportunities to make optimal use of the NN's knowledge in the search for the most informative patterns. Section 4.3.5 shows that, even for subset selection at each epoch, the total number of calculations is reduced compared to that of FSL. To further reduce computations, subsets can be selected at longer time intervals, provided that mechanisms are implemented to prevent overfitting.
- **Subset selection criteria**: The SASLA active learning operator, $\mathcal{A}^{-}_{SASLA}$, in equation (4.7) embodies the subset selection criterion.
- **Subset size**: The size of each subset may differ from one selection interval to the next. Subset sizes depend on the subset selection constant, β, and pattern informativeness as obtained from the knowledge embedded in the current NN weights.

The usual global termination criteria are used, i.e. training stops when the maximum number of epochs is exceeded, or when the MSE is lower than the given threshold, or when the percentage correctly classified patterns is higher than the given threshold. The complexity of SASLA is investigated in the next section.

4.3.4 Model Complexity

An important requirement for any new learning algorithm is that its complexity should not be unacceptably higher than existing algorithms. If possible, the complexity should be reduced. The complexity of the proposed selective learning algorithm is explored in this section and compared to that of FSL. For the purposes of this exposition, complexity is expressed as the number of calculations and comparisons made during one training sweep through the training set (one epoch). Calculations include additions, subtractions, multiplications and divisions.

Since the selective learning operator $\mathcal{A}^{-}_{SASLA}$ is applied to each pattern in the candidate set D_C, a computational cost is assigned to each pattern presentation. Let $\mathcal{C}^{(p)}_{FSL}$ and $\mathcal{C}^{(p)}_{SASLA}$ respectively denote the cost per pattern presentation for FSL and SASLA. Then,

$$\mathcal{C}^{(p)}_{FSL} = \mathcal{C}^{(p)}_{W} + \mathcal{C}^{(p)}_{V} \tag{4.11}$$

and

$$\mathcal{C}^{(p)}_{SASLA} = \begin{cases} \mathcal{C}^{(p)}_{\mathcal{A}^{-}_{SASLA}} + \mathcal{C}^{(p)}_{W} + \mathcal{C}^{(p)}_{V} & \text{if pattern } p \text{ is selected} \\ \mathcal{C}^{(p)}_{\mathcal{A}^{-}_{SASLA}} & \text{if pattern } p \text{ is not selected} \end{cases} \tag{4.12}$$

where $\mathcal{C}^{(p)}_{W}$ is the cost of updating weights between the hidden and output layer for one pattern, and $\mathcal{C}^{(p)}_{V}$ is the cost of updating weights between the input and hidden layer. The term $\mathcal{C}^{(p)}_{\mathcal{A}^{-}_{SASLA}}$ represents the cost to calculate the informativeness of pattern p and to make the selection decision. Equation (4.11) illustrates a fixed computational cost for each pattern presentation for FSL, while the computational cost per pattern presentation for SASLA depends on whether the pattern is selected for inclusion in the training subset.

The total number of pattern presentations for FSL after ξ epochs is

$$\mathcal{T}_{FSL}(D_C) = \sum_{\varepsilon=1}^{\xi} P_C = \xi P_C \tag{4.13}$$

For SASLA, the total number of pattern presentations is

$$\begin{aligned} \mathcal{T}_{SASLA}(D_C) &= \mathcal{T}_{SASLA}(D_{S_0}) + \mathcal{T}_{SASLA}(D_{S_1}) + \cdots + \mathcal{T}_{SASLA}(D_{S_S}) \\ &= \sum_{s=1}^{S} \sum_{\varepsilon=\xi_{s-1}}^{\xi_s} P_{S_s} \\ &= \sum_{s=1}^{S} (\xi_s - \xi_{s-1}) P_{S_s} \end{aligned} \tag{4.14}$$

From equations (4.11) to (4.14), the total cost for all SASLA pattern presentations is

$$\mathcal{C}_{SASLA} = \mathcal{C}_{FSL} + \mathcal{C}_{\mathcal{A}^{-}_{SASLA}} - (\mathcal{T}_{FSL}(D_C) - \mathcal{T}_{SASLA}(D_C))(\mathcal{C}^{(p)}_{W} + \mathcal{C}^{(p)}_{V}) \tag{4.15}$$

where the term $\mathcal{T}_{FSL}(D_C) - \mathcal{T}_{SASLA}(D_C)$ is the difference in the total number of pattern presentations between FSL and SASLA after ξ epochs, and $\mathcal{C}_{SASLA}$ and $\mathcal{C}_{FSL}$ respectively represent the total training cost of SASLA and FSL; $\mathcal{C}_{\mathcal{A}^{-}_{SASLA}}$ is the total cost of the selective

learning operator. From equation (4.15), a cost saving by SASLA is achieved when $\mathcal{C}_{SASLA} - \mathcal{C}_{FSL} < 0$. If $\mathcal{C}$ denotes the cost saving, then

$$\mathcal{C} = \mathcal{C}_{\mathcal{A}^-_{SASLA}} - (\mathcal{T}_{FSL}(D_C) - \mathcal{T}_{SASLA}(D_C))(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) < 0 \tag{4.16}$$

If equation (4.16) becomes true, SASLA is more cost effective than FSL.

For one pattern presentation, $\mathcal{C}_W^{(p)} = 2K(J+1)(7+I+J)$ calculations, and $\mathcal{C}_V^{(p)} = (2+5K)(I+1)(J+1)$ calculations, where I, J and K are respectively the number of input, hidden and output nodes (refer to (D.14) and (D.21) for the weight update equations).

Referring to the sensitivity analysis selective learning algorithm in section 4.3.3, the total cost of applying the operator is

$$\mathcal{C}_{\mathcal{A}^-_{SASLA}} = S \times (\mathcal{C}_{S_{oz}} + \mathcal{C}_{\vec{S_o}} + \mathcal{C}_{\vec{\Phi}} + \mathcal{C}_{\overline{\Phi}} + \mathcal{C}_{\Psi}) \tag{4.17}$$

where S is the total number of subset selection intervals $\tau_1, \cdots, \tau_s, \cdots, \tau_S$. In equation (4.17), $\mathcal{C}_{S_{oz}}$ is the total cost of calculating the sensitivity matrix for all candidate patterns. That is, $\mathcal{C}_{S_{oz}} = P_C \mathcal{C}_{S_{oz}^{(p)}}$. Referring to equation (E.6), the cost $\mathcal{C}_{S_{oz}^{(p)}}$ of calculating the sensitivity matrix for a single pattern is simply $3IJK$ calculations, since the derivative $f_{o_k}^{(p)'}$ is already calculated when $\delta_{o_k}^{(p)}$ is computed (refer to equation (D.13)), and the derivative $f_{y_j}^{(p)'}$ is already available after calculating $\delta_{y_j}^{(p)}$ (refer to equation (D.20)). Therefore,

$$\mathcal{C}_{S_{oz}} = P_C \times C_{S_{oz}^{(p)}} = P_C \times (3IJK) \tag{4.18}$$

The total number of calculations per selection interval to compute the output sensitivity vector for all candidate patterns is (from equation (4.3))

$$\mathcal{C}_{\vec{S_o}} = P_C \times (IK) \tag{4.19}$$

To compute the informativeness of all candidate patterns, the total number of comparisons is (from equation(4.2))

$$\mathcal{C}_{\vec{\Phi}} = P_C \times (K) \tag{4.20}$$

To compute the average pattern informativeness, the cost is (from equation (4.10))

$$\mathcal{C}_{\overline{\Phi}} = P_C + 1 \tag{4.21}$$

Finally, the cost of applying the selection operator is (from equations (4.7) and (4.9))

$$\mathcal{C}_{\Psi} = P_C + 2 \tag{4.22}$$

Substitution of equations (4.18) to (4.22) into equation (4.17) yields

$$\mathcal{C}_{\mathcal{A}^-_{SASLA}} = S(P_C(3IJK + IK + K + 2) + 3) \tag{4.23}$$

Which gives a cost saving of

$$\begin{aligned} \mathcal{C} &= S(P_C(3IJK + IK + K + 2) + 3) \\ &- (\xi P_C - \mathcal{T}_{SASLA}(D_C))(2K(J+1)(7+I+J) + (2+5K)(I+1)(J+1)) \end{aligned} \tag{4.24}$$

An analysis of equation (4.24) is presented next to establish under which conditions the cost saving is negative or positive.

- In the worst case, all candidate patterns lie close to decision boundaries such that SASLA prunes no patterns during training. In this case $\mathcal{C} = \mathcal{C}_{\mathcal{A}^-_{SASLA}}$. Hence, SASLA is computationally more complex than FSL when all patterns lie close to boundaries.
- Since $3IJK + IK + K + 2$ is much less than $2K(J+1)(7+I+J) + (2+5K)(I+1)(J+1)$, with a difference of $21JK + 4IJK + 2KJ^2 + 18K + 16IK + 2IJ + 2I + 2J$, SASLA will be computationally less expensive than FSL when the difference $\xi P_C - \mathcal{T}_{SASLA}(D_C)$ is large. As illustrated in section 4.3.5, this is the case for all the experiments investigated.
- The complexity of SASLA can further be reduced by using fewer subset selection intervals, instead of selecting a new training subset at each epoch.
- Even for small candidate training set sizes, SASLA can be less complex than FSL since the cost for weight updates is much larger than the cost of applying the selective learning operator.

The computational complexity of SASLA and FSL is evaluated for each experiment in section 4.3.5.

4.3.5 Experimental results

The first goal of this chapter, i.e. the presentation of SASLA, has been dealt with in the previous sections. This section addresses the other goals, i.e. an evaluation of the performance of SASLA. For this purpose, SASLA is compared with FSL to establish whether SASLA's performance is better than, or at least comparable to, that of FSL.

Firstly, an artificial problem is used for illustrative convenience. This problem is used to easily visualize the dynamic selection of training patterns around decision boundaries. The section then continues with a rigorous exploration of SASLA's application to several real-world problems of varying complexity, using a conservative subset selection constant of $\beta = 0.9$. The effects of varying β are illustrated on two of these real-world problems. The performance of SASLA, with $\beta = 0.9$, is then compared to a selective learning approach that selects only those patterns not yet correctly classified (this thesis refer to this selection approach as *error selection* (ES)). A summary of the problems investigated is presented below. The experimental procedure is described and the measures used to assess the performance of the model are listed. An analysis of the experimental results is then given.

Experiments

Real-world problems of varying complexity have been selected to test the performance of the proposed sensitivity analysis selective learning algorithm. Problems differ in candidate training set size, input and output dimensions, complexity and the characteristics of data values (i.e., continuous, discrete, binary, whether missing values occur, etc). Table 4.1 presents a summary of these problems, which includes characteristics of the data sets. For problems where attributes have missing values, these missing values were replaced by the average value for that attribute during a data pre-processing phase. The table also lists the oversized NN architecture for each problem (the optimized architectures are obtained in chapter 5). Table 4.2 lists for each problem the learning parameters used, including the learning rate, momentum, training and test set sizes, and the number of epochs the networks were trained.

Problem	Attribute Types	NN Architecture	Missing Values	Class Distribution	Source
circle	continuous	2-3-1	none	class1 - 75% class2 - 25%	artificial
breast cancer	discrete	10-15-1	yes	benign - 65.5% malignant - 34.5%	[Prechelt 1994]
diabetes	continuous	8-40-1	none	class1 - 65.1% class2 - 34.9%	[Prechelt 1994]
iris	continuous	4-10-3	none	setosa - 33.3% versicolor - 33.3% virginica - 33.3%	[UCI]
wine	continuous	13-10-3	none	class1 - 33.1% class2 - 39.9% class3 - 27.0%	[UCI]
glass	continuous	9-30-6	none	class1 - 32.7% class2 - 35.5% class3 - 7.9% class4 - 6.1% class5 - 4.2% class6 - 13.6%	[Prechelt 1994]
hepatitis	binary continuous	19-20-1	in most attributes	class1 - 79.87% class2 - 20.13%	[UCI]
thyroid	binary continuous	21-20-3	none	class1 - 2.31% class2 - 5.11% class3 - 92.58%	[Prechelt 1994]

Table 4.1: Characteristics of data sets used to test SASLA

Experimental procedure

A short overview of the experimental procedure is given in this section. All the assumptions about NN model parameters are listed. Each problem investigated is referred to as an *experiment*. One NN training and testing session of an experiment is referred to as a *simulation*. For each experiment, 50 simulations were executed for each of the learning models. For the purposes of this thesis, 50 FSL simulations, 50 SASLA simulations and 50 ES simulations were carried out. This choice of the number of simulations allows the normality assumption and faithful comparison of the means [Mitchell 1997].

The overfitting characteristics of each learning model was investigated by using an oversized NN architecture.

Refer to a *simulation pair* as one simulation from each of the learning models with the same

Problem	Train Set/ Test Set	Number of Epochs	Learning Rate	Momentum
circle	250/150	500	0.1	0.9
breast cancer	480/120	1000	0.1	0.9
diabetes	560/140	5000	0.1	0.9
iris	120/30	1000	0.1	0.9
wine	142/36	500	0.1	0.9
glass	172/42	2000	0.1	0.6
hepatitis	123/31	200	0.1	0.9
thyroid	4000/3000	300	0.05	0.5

Table 4.2: Learning parameters for SASLA vs FSL experiments

simulation number. NNs corresponding to the simulations of a simulation pair had the same architecture, learning parameters (learning rate and momentum), initial random weights, and training and test sets.

For each experiment, the original data set was scaled such that all input values are in the range $[-1, 1]$, and all outputs are in the range $[0.1, 0.9]$. Fifty training and test set pairs were then randomly generated from the original data set. Each simulation used a different training and test set pair. Training continued for a fixed number of epochs (refer to table 4.2), and was not stopped when a specified error limit was reached. This facilitates the study of the overfitting effects of the different learning models using Röbel's generalization factor defined in section 4.1.1.

Both FSL and SASLA used on-line learning where weights were updated after each pattern was presented. Patterns were selected randomly from the training set during training. The learning rate and momentum for each experiment are listed in table 4.2. For the comparison with FSL, SASLA used a conservative subset selection constant of $\beta = 0.9$, and a new training subset was selected after each epoch. Since this section deals with classification problems only, the number of correctly classified patterns was used as measure of training accuracy and generalization. For the purposes of this exposition assume that a pattern is correctly classified if for each output unit o_k and a pattern p, $((o_k^{(p)} \geq 0.7 \text{ and } t_k^{(p)} = 0.9) \text{ or } (o_k^{(p)} \leq 0.3 \text{ and } t_k^{(p)} = 0.1))$.

Results reported are averages over the 50 simulations, with 95% confidence intervals as obtained from the t-distribution.

Performance Measures

A study of the performance of a learning model includes an investigation into the model's generalization, training time (computational complexity) and convergence characteristics. Generalization is expressed as the percentage of test patterns correctly classified by the NN. Part of the investigation into generalization performance is a study of overfitting. The generalization factor as defined by Röbel [Röbel 1994a, Röbel 1994b, Röbel 1994c] (also refer to section 4.1.1) is used to assess the overfitting characteristics of a model, but with reference to the percentage correctly classified patterns instead of MSE. In this case, the smaller the generalization factor, the more does overfitting affect that model.

Training time is an important consideration when studying the performance of a model. For this study training time is expressed as a function of the number of pattern presentations. A cost, expressed as the number of calculations and comparisons, is associated with each pattern. The difference between the learning time of different learning models is determined as the difference in the number of pattern presentations and the total cost saving after each epoch.

The convergence characteristics of a learning model are expressed as the percentage of simulations that did not reach a specified generalization level.

Results

This section presents results of the application of sensitivity analysis selective learning to an artificial and several real-world classification problems. The experimental procedure and learning parameter settings are as discussed in previous sections.

Firstly, an artificial problem is used to illustrate the working of this selective learning algorithm. The objective is to show how sensitivity analysis can be used to select patterns around decision boundaries. The artificial problem, as introduced in chapter 3, equation

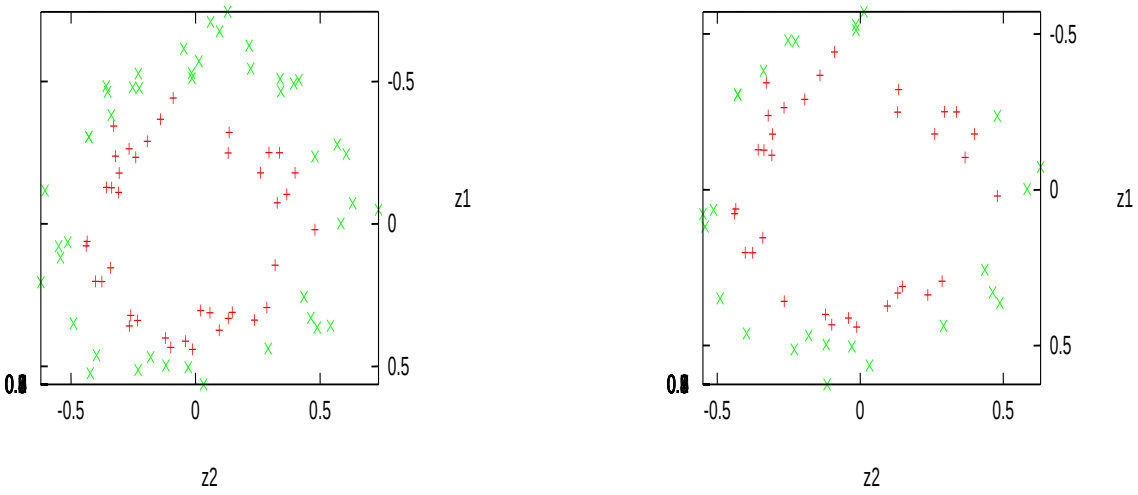
(3.4) (page 42), is to discriminate between two classes, where the one class is inside a circle of radius 0.5 centered at the origin, and the other class outside the circle, but bounded by an unit square (refer to equation (3.4) and figure 3.3).

The visualization of the boundaries for this problem was illustrated in chapter 3, section 3.3. Figure 3.5 (page 42) show that only a small percentage of the training patterns in the candidate training set lie close to the boundaries, suggesting that we may benefit from discarding patterns far away from boundaries.

For the same simulation, figures 4.2(a) and 4.2(b) illustrate the patterns used for training as selected by SASLA for epoch 200 and epoch 500 respectively. Comparison of these figures with figure 3.3 (page 42), which represents the candidate training set, reveals a substantial reduction in the number of training patterns. These figures also show the distribution of the selected patterns around the -0.5 and 0.5 boundaries. SASLA retains those patterns important to form the decision boundaries. From figure 4.2 we see that those patterns which are most distant from the boundary are discarded from epoch 200 to epoch 500.

Next, results are presented to illustrate the performance of SASLA on this artificial problem, and to compare the performance with that of FSL. Results reported are averages over all simulations. Figure 4.3(a) represents the learning profiles for both FSL and SASLA. Training error and generalization performance correctly classified patterns, are plotted against the number of pattern presentations (representing 500 epochs of training). Early in training, the training error and generalization of SASLA exceeded that of FSL. The highest average training error reached by SASLA is 94.4% with a corresponding 92.4% average generalization, compared to a 92.1% average training error and 89.7% average generalization for FSL.

SASLA converged much faster than FSL in terms of the number of pattern presentations, as illustrated in figure 4.3(a). SASLA's faster convergence is also illustrated in figure 4.3(b), which shows for different generalization levels how many pattern presentations were needed by the learning models to reach these generalization levels (considering only those simulations that did converge to these levels). SASLA performed similarly to FSL for up to a generalization of approximately 77%. For generalization higher that 77%, FSL took substantially longer than SASLA to reach the same generalization level. For example, FSL took on average 35 768



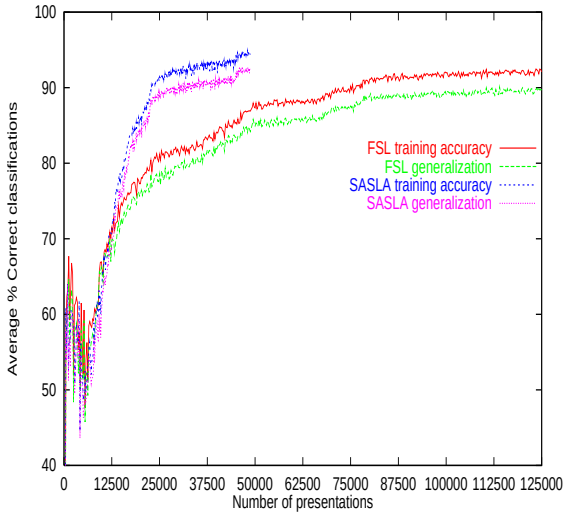
(a) At epoch 200

(b) At epoch 500

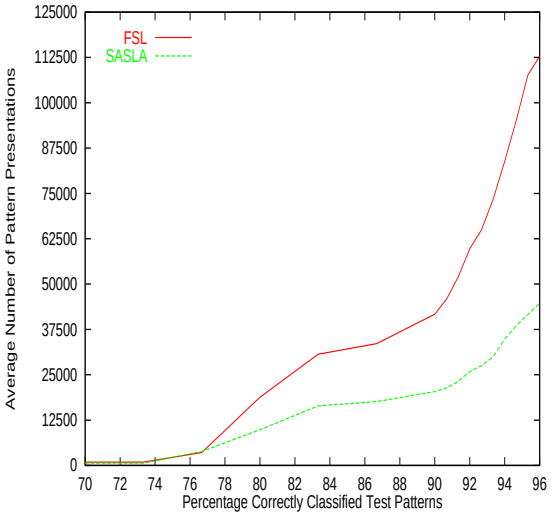
Figure 4.2: Selected training patterns at different epochs for the circle classification problem

presentations to reach a 90% generalization, whereas SASLA took 20 378 presentations. To reach a high generalization level of 96%, FSL needed 84 556 presentations compared to the 27 439 presentations of SASLA. The faster convergence by SASLA was achieved by the rapid reduction in the number of training patterns. Figure 4.3(c) shows an exponential decrease in the number of training patterns. Even after 100 epochs, the training set was reduced on average by 55%; 68 out of the original 250 patterns were used for training during epoch 500 - a substantial reduction.

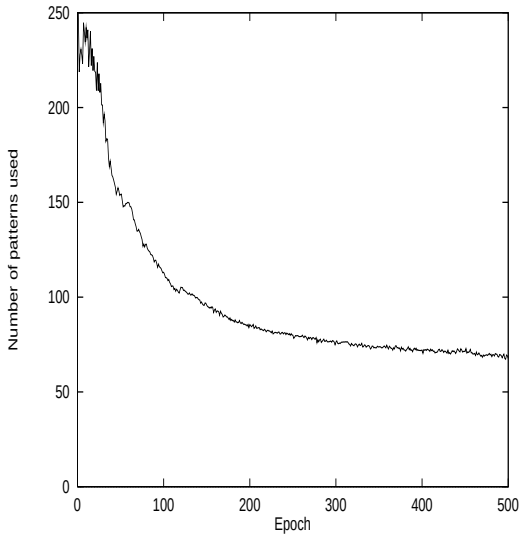
The question now arises whether the reduction in the number of training patterns, in addition to the added computational cost, saves any costs compared to FSL. At the final epoch, SASLA achieved an average cost saving of $10\,854\,830 \pm 459\,497$ computations. A saving in computational cost was observed very early in training, at epoch 40. This early cost saving was achieved through an exponential reduction in the training set size, as illustrated in figure 4.3(c).



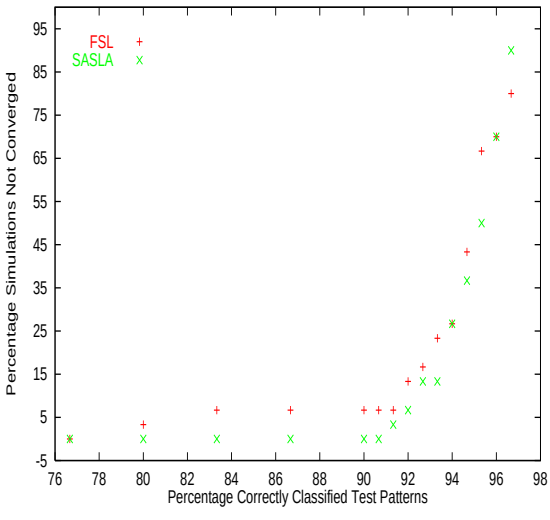
(a) Average percentage correctly classified patterns vs presentations



(b) Number of presentations for different generalization levels



(c) Average number of selected patterns



(d) Percentage simulations that did not converge to different generalization levels

Figure 4.3: Results summary for the circle classification problem

Next, consider figure 4.3(d) which illustrates the percentage of simulations that did not reach specific generalization levels. SASLA showed better convergence results than FSL. SASLA started having non-convergent simulations from a 91.3% generalization level, whereas FSL started having non-convergent simulations from an 80% generalization level. Up to a generalization of 96%, SASLA had more converged simulations than FSL. It is only for a high generalization of 96.67% that SASLA had more non-convergent simulations.

In the next part of this section, results of the application of SASLA to the real-world problems listed in table 4.1 are presented and discussed. Results are presented under three headings: **Real-world Problems** which compares SASLA, with a selection constant $\beta = 0.9$, with FSL, **Selection Constant's Effects** which investigates the performance of SASLA under different values of the selection constant, β, using the *glass* and *wine* problems, and **Error Selection** which compares SASLA with ES, using the *glass, wine* and *circle* problems.

Real-World Problems

This section compares the performances of SASLA (with selection constant $\beta = 0.9$) to that of FSL. The generalization performance, overfitting effects, complexity and convergence of the two learning algorithms are compared.

Tables 4.3 and 4.4 present extensive summaries of the results obtained for all problems considered. Table 4.3 lists for each problem the following statistics as obtained at the final training epoch (as given in table 4.1):

- The average training error, $\overline{\mathcal{E}}_T$, and the average generalization, $\overline{\mathcal{E}}_G$, over the 50 simulations, as the percentage correctly classified patterns. Also included are the 95% confidence intervals computed from the t-distribution.
- The best average generalization, $\overline{\mathcal{E}}_G^{best}$, and the number of pattern presentations needed to reach this best generalization.
- The average generalization factor, $\overline{\rho}$, over the 50 simulations, together with 95% confidence intervals. The generalization factor is computed per simulation as $\rho = \mathcal{E}_G/\mathcal{E}_T$, where $\mathcal{E}_G$ is the error on the test set (generalization), and $\mathcal{E}_T$ is the error on the training

Problem	$\overline{\mathcal{E}}_T$	$\overline{\mathcal{E}}_G$	$\overline{\mathcal{E}}_G^{best}$	Presentation	$\overline{\rho}$
breast cancer	(-0.048 ± 0.004)	(0.016 ± 0.009)			(0.064 ± 0.009)
SASLA	$95.17 \pm 0.42\%$	$93.65 \pm 0.71\%$	96.2%	1530	0.984 ± 0.006
FSL	$99.98 \pm 0.02\%$	$92.03 \pm 0.76\%$	96.2%	2880	0.921 ± 0.007
diabetes	(-0.070 ± 0.014)	(0.002 ± 0.017)			0.058 ± 0.020
SASLA	$82.98 \pm 0.84\%$	$60.87 \pm 1.42\%$	66.0%	20480	0.735 ± 0.019
FSL	$89.97 \pm 8.52\%$	$60.47 \pm 1.31\%$	64.3%	192660	0.676 ± 0.020
iris	(-0.033 ± 0.008)	(-0.009 ± 0.017)			(0.027 ± 0.014)
SASLA	$95.22 \pm 0.56\%$	$93.93 \pm 1.44\%$	94.4%	20470	0.989 ± 0.017
FSL	$98.52 \pm 0.63\%$	$94.8 \pm 0.98\%$	95.2%	7320	0.963 ± 0.010
wine	(-1.283 ± 0.262)	(-0.645 ± 0.521)			(0.005 ± 0.006)
SASLA	$98.33 \pm 0.33\%$	$98.89 \pm 0.64\%$	100.0%	21317	1.006 ± 0.007
FSL	$99.46 \pm 0.27\%$	$99.53 \pm 0.47\%$	100.0%	28189	1.001 ± 0.006
glass	(-2.733 ± 1.353)	(-3.714 ± 1.455)			(-0.0144 ± 0.026)
SASLA	$69.06 \pm 1.07\%$	$70.90 \pm 1.83\%$	88.1%	281485	1.029 ± 0.030
FSL	$71.79 \pm 1.23\%$	$74.62 \pm 1.79\%$	90.4%	337120	1.043 ± 0.033
hepatitis	(-0.103 ± 0.007)	(0.017 ± 0.020)			(0.107 ± 0.022)
SASLA	$89.70 \pm 0.74\%$	$77.56 \pm 2.21\%$	77.9%	7580	0.865 ± 0.026
FSL	$99.98 \pm 0.03\%$	$75.85 \pm 2.08\%$	76.7%	14023	0.759 ± 0.021
thyroid	(-0.018 ± 0.017)	(-0.018 ± 0.017)			(-0.0002 ± 0.0018)
SASLA	$93.39 \pm 1.73\%$	$92.82 \pm 1.75\%$	94.2%	70900	0.994 ± 0.005
FSL	$95.22 \pm 0.29\%$	$94.65 \pm 0.30\%$	94.5%	1104000	0.994 ± 0.002

Table 4.3: Comparison of SASLA ($\beta = 0.9$) and FSL error performance measures

set. Since the error is expressed as the percentage correctly classified patterns, the ideal performance is when $\mathcal{E}_G \geq \mathcal{E}_T$; thus, when $\rho = \mathcal{E}_G/\mathcal{E}_T \geq 1$. Usually, $\mathcal{E}_G < \mathcal{E}_T$, but when $\mathcal{E}_G$ is much less than $\mathcal{E}_T$ it indicates severe overfitting of the training set. The objective is therefore to obtain a generalization $\mathcal{E}_G$ as close to $\mathcal{E}_T$ as possible and therefore ρ close to 1. A higher generalization factor ρ thus indicates less overfitting of the training set.

In addition to the averages and confidence intervals, table 4.3 also presents estimates of the difference in performance between SASLA and FSL for the training error, generalization and generalization factor. For example, considering the training errors $\mathcal{E}_{T_i}^{SASLA}$ and $\mathcal{E}_{T_i}^{FSL}$, for each simulation i, the estimate

$$\overline{\sigma} = \frac{1}{50}\sum_{i=1}^{50}(\mathcal{E}_{T_i}^{SASLA} - \mathcal{E}_{T_i}^{FSL}) \tag{4.25}$$

Problem	$\overline{P}_T$	Total presented patterns	Cost Saving ($\times 10^6$)
breast cancer			
SASLA	63.66 ± 8.20	$75\,250.70 \pm 8\,667.86$	$-690.871331 \pm 19.554722$
FSL	480	480 000	
diabetes			
SASLA	304.00 ± 11.20	$1\,501\,858.66 \pm 41\,254.48$	$-6\,488.901950 \pm 292.618026$
FSL	560	2 800 000	
iris			
SASLA	22.50 ± 1.52	$26\,310.36 \pm 1\,575.59$	-172.210654 ± 3.656954
FSL	120	120 000	
wine			
SASLA	78.45 ± 2.88	$45\,117.77 \pm 1\,068.71$	-32.810974 ± 4.913917
FSL	142	710 000	
glass			
SASLA	131.42 ± 1.67	$287\,906.74 \pm 1\,847.39$	176.860996 ± 49.938683
FSL	172	344 000	
hepatitis			
SASLA	46.64 ± 3.5	$10\,746.30 \pm 692.67$	-38.909426 ± 3.374711
FSL	123	24 600	
thyroid			
SASLA	$2\,706.80 \pm 209.34$	$848\,795.13 \pm 19\,710.10$	$-264.849156 \pm 274.009798$
FSL	4 000	1 200 000	

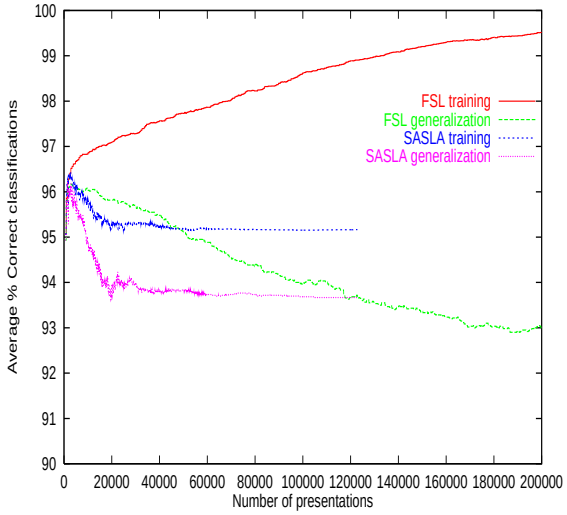
Table 4.4: Comparison of SASLA ($\beta = 0.90$) and FSL computational complexity

is computed to obtain a 95% confidence interval, using the t-distribution. These estimates can be used to determine if there is a significant difference in performance between SASLA and FSL. The performance difference estimates are the first entries for each problem (the values between parentheses).

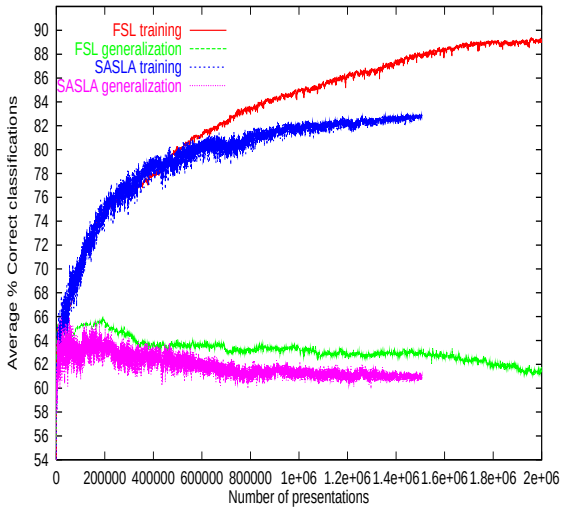
The training error and generalization performance of SASLA and FSL are also compared in figure 4.4. While table 4.3 presents results at the last epoch, figure 4.4 illustrates how the training error and generalization evolve during training.

Table 4.4 summarizes the following information for each learning algorithm and problem at the final epoch:

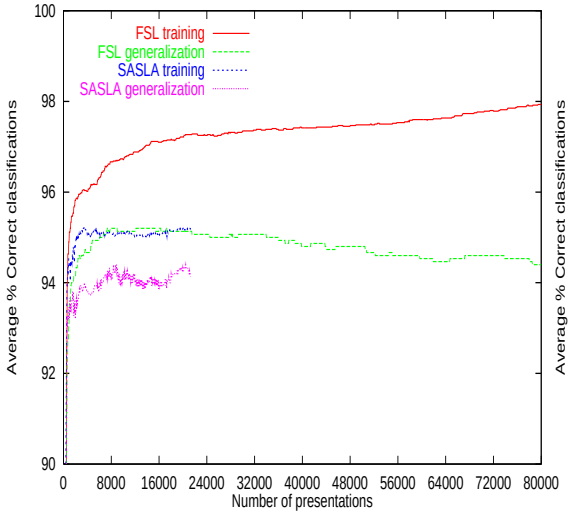
- The average number of patterns, $\overline{P}_T$, selected from the candidate set and used for training, together with 95% confidence intervals for SASLA.



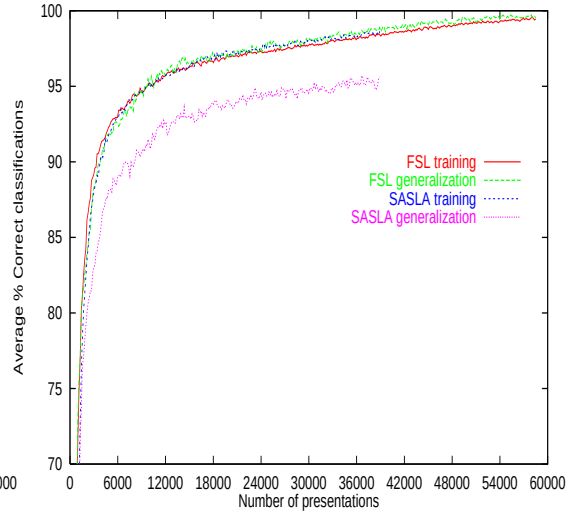
(a) Breast Cancer



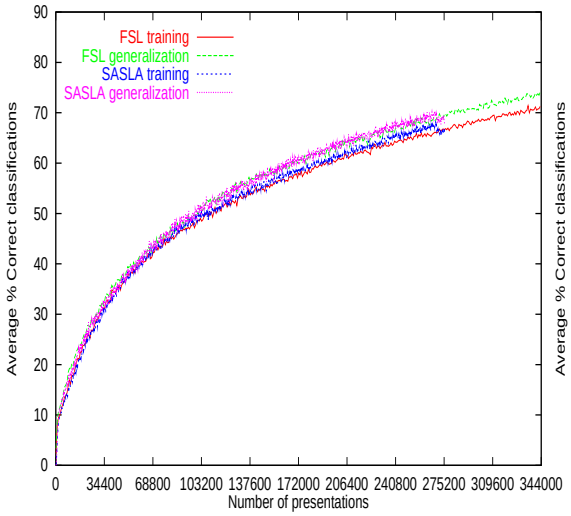
(b) Diabetes



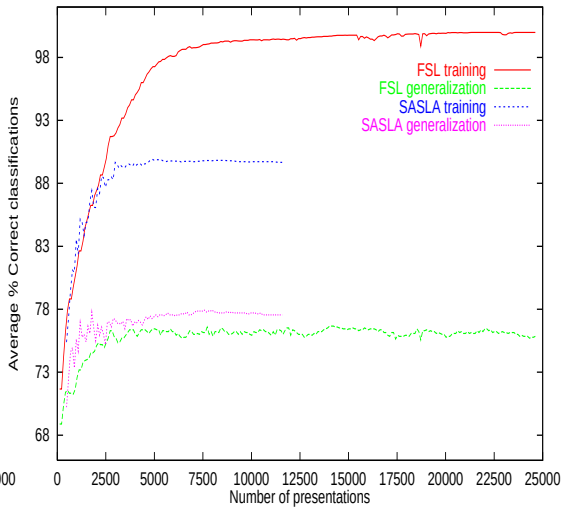
(c) Iris



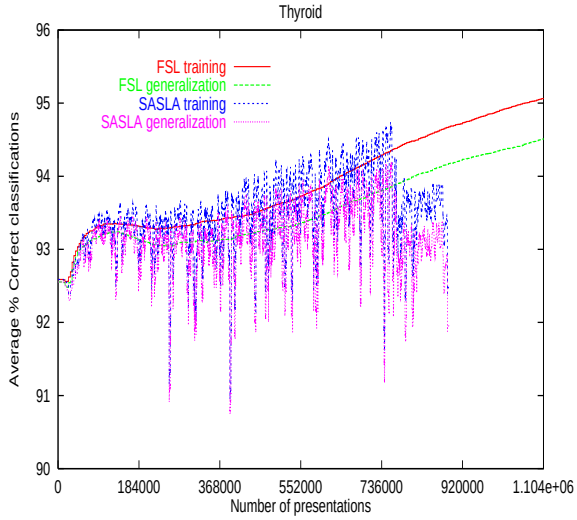
(d) Wine



(e) Glass



(f) Hepatitis



(g) Thyroid

Figure 4.4: Average percentage correct classified patterns vs presentations for real-world problems

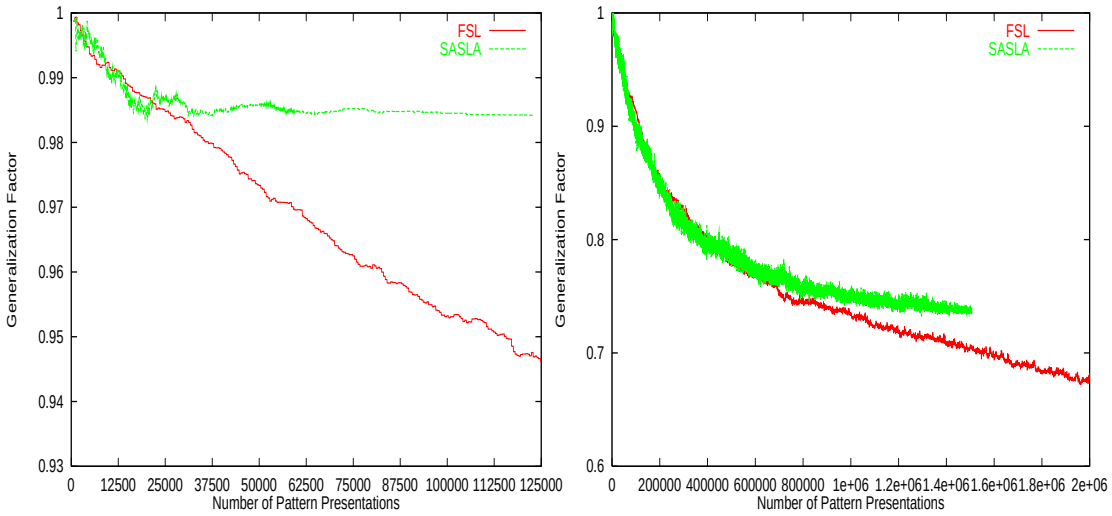
- The average number of pattern presentations, together with 95% confidence intervals for SASLA.
- The average saving in computational cost achieved by using SASLA, with 95% confidence intervals.

With reference to generalization performance, it is not possible to label any of the training methods as being superior, since they produced approximately the same average generalization and best generalization performances. Considering the training error, FSL achieved higher accuracies than SASLA for all problems. Note that even though FSL achieved higher accuracies on the training set, SASLA had better generalization performances for the *breast cancer, diabetes* and *hepatitis* problems, while having approximately the same generalization performance for the *iris, wine* and *thyroid* problems. It is only for the *glass* problem that FSL obtained better generalization than SASLA.

The consequence of FSL having better training accuracies than SASLA, but approximately the same generalization performance, is that FSL tends to overfit the training data. Table 4.3 shows that FSL overfitted more at the final epoch than SASLA for the *breast cancer, diabetes, iris* and *hepatitis* problems. Figure 4.5 illustrates the overfitting effects of the two algorithms for these problems, where the generalization factor is plotted as a function of the number of pattern presentations. Figure 4.5 shows that FSL increasingly overfits as training progresses, while the overfitting characteristics of SASLA stabilize. The two algorithms had the same generalization factors for the other problems.

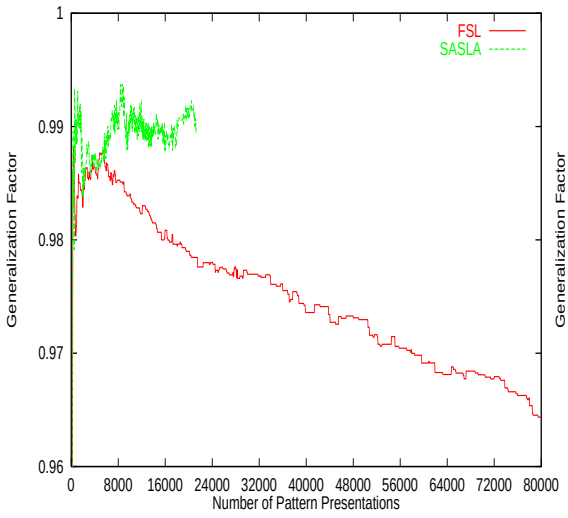
The larger overfitting effects for FSL can be explained by the fact that FSL trains on the entire training set, which, for real-world problems, usually contains outlier and noisy patterns. Dynamic pattern selection around decision boundaries reduces the effects of outliers and patterns with a very large noise component, since these patterns lie the furthest away from decision boundaries compared to "normal" patterns, and will have a smaller chance to be selected. Since overfitting occurs when the network starts to memorize individual patterns (when there are too many free parameters, which was the case for these experiments), this explains why SASLA exhibits less overfitting than FSL.

While SASLA and FSL showed approximately the same generalization performance, SASLA

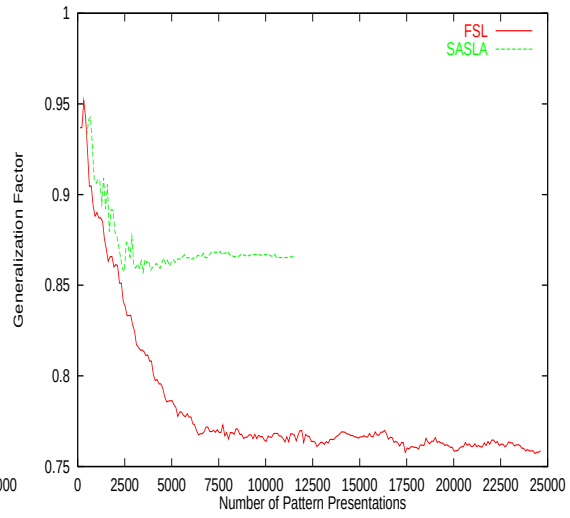


(a) Breast Cancer

(b) Diabetes



(c) Iris



(d) Hepatitis

Figure 4.5: Average generalization factor vs pattern presentations for real-world problems

Problem	Epoch
breast cancer	4
diabetes	15
iris	5
wine	8
glass	-
hepatitis	15
thyroid	218

Table 4.5: Last epoch at which FSL is less expensive than SASLA

needed much less pattern presentations than FSL to reach the same generalization performance levels (refer to figure 4.4). This conclusion is supported by the fourth and fifth columns of table 4.3. These columns show that SASLA needed substantially less pattern presentations to reach its best generalization performance than FSL (except for the *iris* problem).

Table 4.3 and figure 4.4 show that SASLA required less pattern presentations to reach good generalization levels compared to FSL. But does this mean that SASLA is computationally less complex than FSL, taking in consideration the added complexity of applying the pattern selection operator? Using the number of calculations per epoch as measure of computational complexity, equation (4.24) expresses the saving in the number of calculations by using SASLA. The last column in table 4.4 lists the saving in computational cost (at the final epoch) by using SASLA. Note that substantial savings in the number of calculations were achieved for all problems, except the *glass* problem. Even for the conservative subset selection constant, $\beta = 0.9$, SASLA was computationally more feasible than FSL very early in training, as indicated in table 4.5 which lists the last epoch at which FSL was, on average, computationally less expensive than SASLA. Although SASLA showed to be computationally more complex than FSL for the *glass* problem, SASLA still used less training presentations than FSL for 2000 training epochs. (The next section shows that SASLA saved on computational costs for lower β values but at decreased accuracy.) The larger cost for the *glass* problem is attributed to the slow and small decrease in the training set size, such that the computational cost of applying the selection operator is larger than the saving due to the decreased training set size.

For the other problems, these very large computational cost savings were made possible by

the large reduction in training set size through application of the selective learning operator. For these problems, the training set size reduced exponentially. The number of training patterns was rapidly reduced early in training, after which the training set size asymptotically converged to the set of patterns that defines the decision boundaries. The size of the thyroid candidate training set was, however, not that rapidly reduced. Table 4.6 shows for selected epochs the percentage reduction of the original training set.

	Percentage reduction after epoch			
Problem	**25**	**100**	**200**	**300**
breast cancer	$70.85 \pm 2.73\%$	$81.48 \pm 2.43\%$	$83.85 \pm 2.27\%$	$85.26 \pm 1.91\%$
diabetes	$20.57 \pm 3.47\%$	$30.69 \pm 2.63\%$	$38.05 \pm 1.83\%$	$41.08 \pm 1.99\%$
iris	$64.17 \pm 1.40\%$	$74.32 \pm 1.26\%$	$77.90 \pm 1.23\%$	$79.22 \pm 1.28\%$
wine	$19.21 \pm 1.57\%$	$32.44 \pm 1.62\%$	$37.47 \pm 1.72\%$	$40.87 \pm 1.94\%$
glass	$0.69 \pm 0.23\%$	$4.40 \pm 0.43\%$	$7.49 \pm 0.57\%$	$9.76 \pm 0.67\%$
hepatitis	$52.23 \pm 3.97\%$	$59.46 \pm 3.12\%$	$62.00 \pm 2.84\%$	-
thyroid	$17.45 \pm 3.43\%$	$30.21 \pm 4.84\%$	$35.33 \pm 6.35\%$	$32.33 \pm 5.23\%$

	Percentage reduction after epoch			
Problem	**500**	**1000**	**2000**	**5000**
breast cancer	$86.02 \pm 1.78\%$	$86.74 \pm 1.71\%$	-	-
diabetes	$45.03 \pm 1.86\%$	$47.70 \pm 1.85\%$	$48.07 \pm 2.21\%$	$45.71 \pm 2.0\%$
iris	$80.20 \pm 1.40\%$	$81.25 \pm 1.27\%$	-	-
wine	$45.07 \pm 2.11\%$	-	-	-
glass	$11.76 \pm 0.72\%$	$17.68 \pm 1.04\%$	$23.63 \pm 1.01\%$	-
hepatitis	-	-	-	-
thyroid	-	-	-	-

Table 4.6: Training set reduction by SASLA ($\beta = 0.9$)

Finally, figure 4.6 compares the convergence performance of SASLA with that of FSL. This figure plots the percentage of the 50 simulations that did not converge to specific generalization levels. SASLA performed exceptionally well for the *breast cancer* and *diabetes* problems, having much more converged simulations than FSL, especially for the higher generalization levels. For the *iris* problem SASLA performed in general better than FSL, having more converged simulations for most of the generalization levels. It is not possible to identify a superior model for the *hepatitis* and *thyroid* problems. However, notice that SASLA tended to have more converged simulations for the higher generalization levels than FSL. Also, where FSL had more converged simulations, it was by a small percentage. FSL showed better

convergence results for the *wine* and *glass* problems.

Selection Constant's Effects

Section 4.3.2, page 79, explained the importance of the subset selection constant β. While the results presented in the previous section are for $\beta = 0.9$, this section illustrates the effect of varying β values on performance. For this purpose the *glass* and *wine* problems are used, and 50 simulations executed for $\beta = 0.1, 0.3, 0.5$ and 0.7. The results, which include averages over the 50 simulations and associated 95% confidence intervals, are summarized in tables 4.7 and 4.8 for the *glass* problem (at epoch 2000), and tables 4.9 and 4.10 for the *wine* problem (at epoch 500).

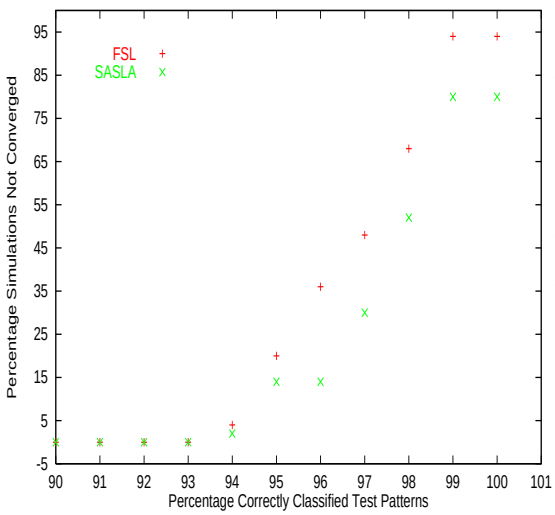
β	$\overline{\mathcal{E}}_T$	$\overline{\mathcal{E}}_G$	$\overline{\mathcal{E}}_G^{best}$	**Presentation**	$\overline{\rho}$
0.9	$69.06 \pm 1.07\%$	$70.90 \pm 1.83\%$	88.10%	281485	1.029 ± 0.030
0.7	$66.43 \pm 1.37\%$	$68.10 \pm 1.82\%$	85.71%	248760	1.031 ± 0.036
0.5	$62.73 \pm 1.76\%$	$64.38 \pm 1.70\%$	83.33%	200510	1.032 ± 0.036
0.3	$59.95 \pm 1.41\%$	$60.90 \pm 2.11\%$	80.95%	177310	1.023 ± 0.045
0.1	$57.26 \pm 1.19\%$	$57.10 \pm 1.99\%$	78.57%	121385	0.999 ± 0.035

Table 4.7: Comparison of error performance measures for different β values for the *glass* problem

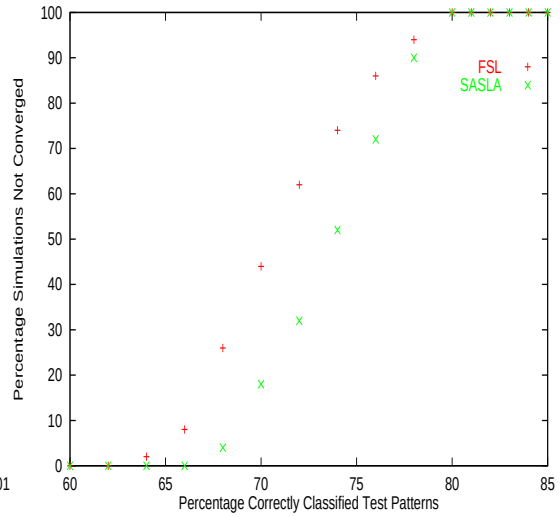
β	$\overline{P}_T$	**Total presented patterns**	**Cost Saving** ($\times 10^6$)
0.9	131.42 ± 1.67	$287\,906.74 \pm 1\,847.39$	176.860996 ± 49.938683
0.7	111.60 ± 1.75	$252\,744.90 \pm 2\,124.33$	$-773.633863 \pm 57.424777$
0.5	94.48 ± 1.98	$214\,170.60 \pm 2\,405.99$	$-1\,816.374341 \pm 65.038596$
0.3	81.30 ± 1.94	$180\,981.50 \pm 2\,468.39$	$-2\,713.542092 \pm 66.725490$
0.1	69.20 ± 3.03	$152\,859.90 \pm 2\,698.49$	$-3\,473.725183 \pm 72.945519$

Table 4.8: Comparison of computational complexity for different β values for the *glass* problem

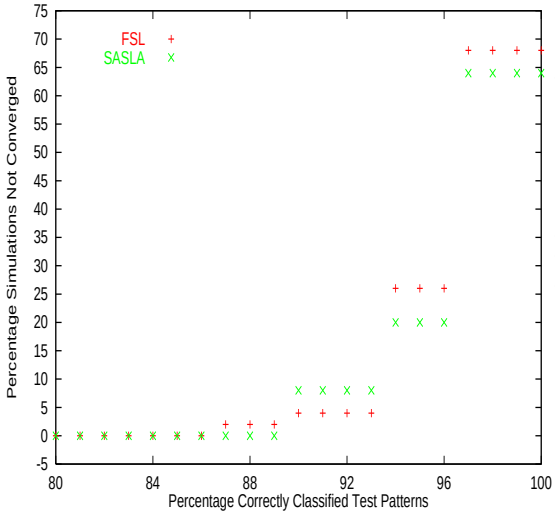
Tables 4.7 and 4.9 show that the lower the value of β, the worse the accuracy of SASLA: The average training error and generalization over the 50 simulations deteriorated for both problems, while the best generalization also decreased for the *glass* problem. Also note that higher β values achieved generalization accuracies larger than the training accuracy. As



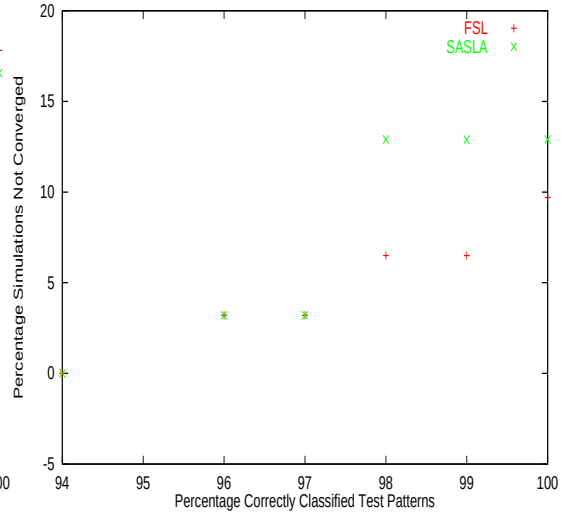
(a) Breast Cancer



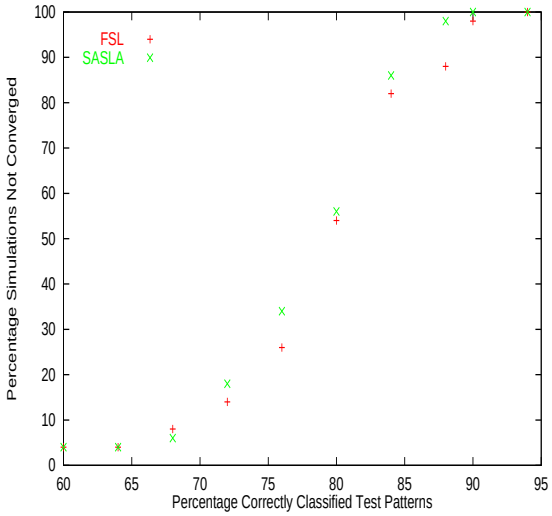
(b) Diabetes



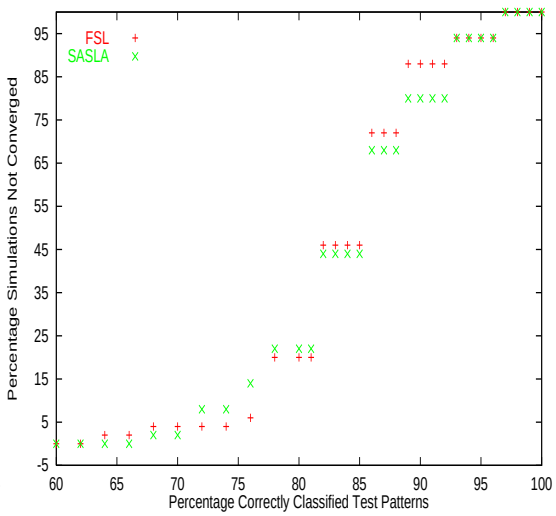
(c) Iris



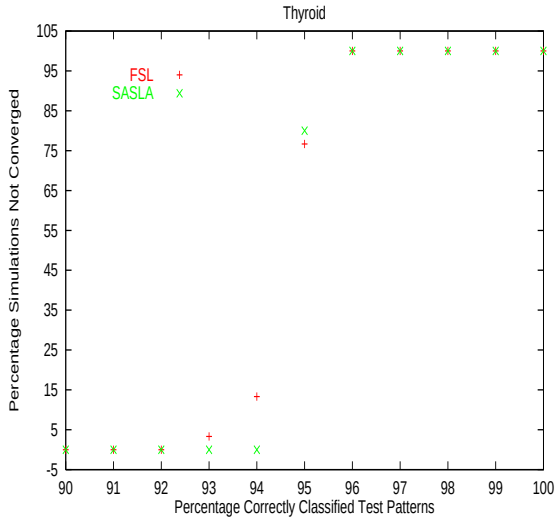
(d) Wine



(e) Glass



(f) Hepatitis



(g) Thyroid

Figure 4.6: Percentage simulations that did not converge to generalization levels for real-world problems

β	$\overline{\mathcal{E}}_T$	$\overline{\mathcal{E}}_G$	$\overline{\mathcal{E}}_G^{best}$	**Presentation**	$\overline{\rho}$
0.9	$98.33 \pm 0.33\%$	$98.89 \pm 0.64\%$	100.0%	21317	1.006 ± 0.007
0.7	$97.56 \pm 0.36\%$	$97.97 \pm 0.86\%$	100.0%	9823	1.004 ± 0.011
0.5	$97.01 \pm 0.46\%$	$97.42 \pm 0.91\%$	100.0%	9956	1.004 ± 0.012
0.3	$96.95 \pm 0.41\%$	$97.14 \pm 0.94\%$	100.0%	8958	1.002 ± 0.012
0.1	$96.64 \pm 0.45\%$	$96.22 \pm 0.95\%$	100.0%	9597	0.996 ± 0.013

Table 4.9: Comparison of error performance measures for different β values for the *wine* problem

β	$\overline{P}_T$	**Total presented patterns**	**Cost Saving** ($\times 10^6$)
0.9	78.45 ± 2.88	$45\,117.77 \pm 1\,068.71$	-32.810974 ± 4.913917
0.7	60.42 ± 1.94	$34\,624.29 \pm 749.06$	-81.060013 ± 3.444165
0.5	51.39 ± 1.35	$28\,384.00 \pm 1\,138.55$	-109.752868 ± 5.235048
0.3	45.48 ± 1.35	$22\,697.76 \pm 1\,311.81$	-125.124575 ± 2.123308
0.1	40.97 ± 1.15	$22\,112.00 \pm 821.32$	-138.591524 ± 3.776447

Table 4.10: Comparison of computational complexity for different β values for the *wine* problem

the value of β becomes smaller, the difference between generalization and training accuracy becomes smaller. Lower β values do, however, save on computations, due to the fact that less patterns are selected for the candidate set. However, the savings in computational cost using low β values do not justify the loss in accuracy for the *glass* problem, while the *iris* problem's decrease in accuracy is acceptable. The larger savings in computational cost for smaller β values are due to the larger decreases in training set sizes compared to larger β values. Tables 4.11 and 4.12 illustrate for the *glass* and *wine* problems how lower β values decreased the training set size for selected epochs.

Tables 4.13 and 4.14 show that the smaller the value of β the more simulations do not converge to generalization levels specified in the table, indicating that the lower β values do not select enough information to achieve the higher generalization accuracies.

What can be concluded from the above experiments is that the best value for β is problem dependent. In selecting a value for β the trade-off between computational cost savings and

	Reduction at epoch				
β	**50**	**100**	**500**	**1000**	**2000**
0.9	$1.14 \pm 0.30\%$	$2.97 \pm 0.43\%$	$7.76 \pm 0.51\%$	$12.86 \pm 1.06\%$	$18.36 \pm 1.15\%$
0.7	$7.03 \pm 0.62\%$	$9.80 \pm 0.84\%$	$21.66 \pm 1.14\%$	$28.35 \pm 1.27\%$	$50.55 \pm 1.02\%$
0.5	$16.69 \pm 1.46\%$	$21.84 \pm 1.06\%$	$37.24 \pm 1.31\%$	$39.15 \pm 1.61\%$	$45.07 \pm 1.15\%$
0.3	$27.66 \pm 1.47\%$	$33.29 \pm 0.84\%$	$47.84 \pm 0.95\%$	$51.21 \pm 1.49\%$	$52.73 \pm 1.13\%$
0.1	$37.30 \pm 2.06\%$	$41.13 \pm 1.43\%$	$54.95 \pm 1.66\%$	$59.91 \pm 1.47\%$	$59.77 \pm 1.76\%$

Table 4.11: Training set reduction for different β values for the *glass* problem

	Reduction at epoch			
β	**25**	**50**	**100**	**500**
0.9	$19.21 \pm 1.57\%$	$27.37 \pm 1.42\%$	$32.44 \pm 1.62\%$	$45.07 \pm 2.11\%$
0.7	$37.89 \pm 1.54\%$	$43.98 \pm 1.48\%$	$48.64 \pm 1.19\%$	$57.97 \pm 1.35\%$
0.5	$48.62 \pm 1.04\%$	$54.02 \pm 1.20\%$	$57.95 \pm 0.97\%$	$63.81 \pm 0.95\%$
0.3	$56.47 \pm 0.71\%$	$61.22 \pm 0.79\%$	$64.63 \pm 0.75\%$	$68.36 \pm 0.81\%$
0.1	$60.97 \pm 0.65\%$	$65.58 \pm 0.67\%$	$68.04 \pm 0.73\%$	$71.15 \pm 0.81\%$

Table 4.12: Training set reduction for different β values for the *wine* problem

	Generalization Levels						
β	**60%**	**65%**	**70%**	**75%**	**80%**	**85%**	**90%**
0.9	0%	0%	10%	29%	65%	90%	98%
0.7	2%	6%	18%	46%	80%	96%	100%
0.5	2%	20%	38%	70%	94%	100%	100%
0.3	8%	28%	76%	88%	98%	100%	100%
0.1	16%	52%	84%	94%	100%	100%	100%

Table 4.13: Comparison of convergence results for different β values for the *glass* problem

	Generalization Levels			
β	**94%**	**96%**	**98%**	**100%**
0.9	0%	4%	12%	12%
0.7	0%	6%	38%	38%
0.5	0%	10%	36%	36%
0.3	4%	12%	42%	42%
0.1	4%	12%	38%	38%

Table 4.14: Comparison of convergence results for different β values for the *wine* problem

decreased performance needs to be considered. It is suggested that future research includes an investigation into adaptive β values, where the value of β increases when performance deteriorates, and decreases when performance improves.

Error Selection

The reader may now ask why don't we simply select those patterns from the candidate set that are not yet correctly classified. While this approach will be computationally less expensive than SASLA, due to the simple selection operator, the error selection approach steps into trouble when the candidate set contains outlier patterns (assuming a non-robust estimator such as sum squared error is used [Hoaglin *et al* 1983, Huber 1981]). Since outlier patterns will be classified incorrectly, an error selection approach will select these outlier patterns for training. For sensitivity analysis patterns selection, on the other hand, the chance that outliers will be included in the training set is reduced, since the selection of patterns depends on how far they lie from the decision boundaries - outliers lie further away from decision boundaries than "normal" patterns. SASLA might initially select these outlier patterns, but will discard them as decision boundaries get more refined during training.

The objective of this section is to compare results obtained using an error selection scheme with that of SASLA. For this purpose the *glass* and *wine* problems, as well as the *circle* problem with outliers added to the candidate set (5% of the candidate set), were used to train neural networks based on the following pattern selection schema: if there exists an output unit such that $(t_k^{(p)} = 0.9)\&\&(o_k^{(p)} < 0.7)$ or $(t_k^{(p)} = 0.1)\&\&(o_k^{(p)} > 0.3)$, then select pattern p for training. This selection criterion coincides with the thesis assumption as to when a pattern should be accepted as being correctly classified (see page 87).

Table 4.15 summarizes the error performance results for the ES approach in comparison with SASLA for $\beta = 0.9$, while table 4.16 and table 4.17 respectively summarizes the complexity and convergence results. For all problems SASLA showed to have significantly better generalization performance than ES. Table 4.15 shows that SASLA has been influenced by the occurrence of outlier for the *circle* problem, previously having a 92.5% average generalization on clean training data (refer to page 89). The results show that ES is even more affected by

the outliers, having an average generalization of 73.4%, compared to 88.3% on clean data. From table 4.16 it is evident that ES is computationally less intensive than SASLA, using substantially less training patterns than SASLA. This very large reduction in training set size by ES compared to SASLA is a cause of the worse ES generalization performance, since the reduced training set may not contain sufficient patterns to refine the boundaries.

SASLA showed to have better convergence characteristics than ES. Table 4.17 illustrates that ES had substantially more simulations that did not converge to the different generalization levels.

Problem	$\overline{\mathcal{E}}_T$	$\overline{\mathcal{E}}_G$	$\overline{\mathcal{E}}_G^{best}$	**Presentation**	$\overline{\rho}$
glass	(9.779 ± 1.118)	(10.143 ± 1.750)			(0.0005 ± 0.029)
SASLA	$69.06 \pm 1.07\%$	$70.90 \pm 1.83\%$	88.10%	281485	1.029 ± 0.030
ES	$59.28 \pm 1.15\%$	$60.76 \pm 2.27\%$	78.60%	192125	1.029 ± 0.043
wine	(0.060 ± 0.022)	(0.079 ± 0.029)			(-0.0144 ± 0.026)
SASLA	$98.33 \pm 0.33\%$	$98.89 \pm 0.64\%$	100.0%	21317	1.006 ± 0.007
ES	$92.29 \pm 0.02\%$	$90.97 \pm 0.03\%$	100.0%	5704	0.985 ± 0.020
circle	(0.039 ± 0.055)	(0.039 ± 0.052)			(0.0003 ± 0.027)
SASLA	$75.51 \pm 4.40\%$	$79.28 \pm 3.68\%$	94.08%	73264	1.059 ± 0.034
ES	$71.61 \pm 3.52\%$	$73.39 \pm 3.17\%$	86.51%	25992	1.059 ± 0.028

Table 4.15: Comparison of SASLA ($\beta = 0.9$) and ES error performance measures

Problem	$\overline{P}_T$	**Total presented patterns**	**Cost Saving** ($\times 10^6$)
glass			
SASLA	131.42 ± 1.67	$287\,906.74 \pm 1\,847.39$	176.860996 ± 49.938683
ES	67.92 ± 1.88	$183\,540.80 \pm 2\,037.27$	$-4\,333.405094 \pm 55.071407$
wine			
SASLA	78.00 ± 2.88	$45\,117.77 \pm 1\,068.71$	-32.810974 ± 4.913917
ES	11.16 ± 3.46	$7\,766.29 \pm 486.73$	-290.322597 ± 2.237983
circle			
SASLA	153.93 ± 14.19	$100\,459.60 \pm 4\,430.68$	-1.540722 ± 0.797522
ES	79.63 ± 12.81	$31\,324.80 \pm 1\,031.40$	-13.985036 ± 0.185652

Table 4.16: Comparison of SASLA ($\beta = 0.9$) and ES computational complexity

Problem	Generalization levels						
glass	**60%**	**65%**	**70%**	**75%**	**80%**	**85%**	**90%**
SASLA	0%	0%	10%	30%	64%	90%	98%
ES	10%	18%	62%	90%	100%	100%	100%
wine	**90%**	**95%**	**96%**	**97%**	**98%**	**99%**	**100%**
SASLA	0%	0%	4%	4%	12%	12%	12%
ES	10%	48%	48%	78%	78%	78%	78%
circle	**75%**	**80%**	**85%**	**90%**	**95%**		
SASLA	0%	6%	40%	86%	100%		
ES	0%	14%	84%	100%	100%		

Table 4.17: Comparison of SASLA ($\beta = 0.9$) and ES convergence results

In conclusion, although ES is computationally more efficient than SASLA, the better convergence and generalization results of SASLA make it the preferred selective learning algorithm.

4.3.6 Conclusive Remarks

Section 4.3 presented a new selective learning algorithm that uses pattern sensitivity information to dynamically select those training patterns that contain the most information about the position of decision boundaries. Although, from the problems investigated in this section, it is not possible to label either this selective learning algorithm, SASLA, or FSL as having a superior generalization performance, SASLA generally showed to be more robust to overfitting. Furthermore, SASLA required less pattern presentations to reach approximately the same generalization levels. A major advantage of SASLA is the exponential reduction in training set size, and consequently a very large saving in computational costs. SASLA also showed a more favorable convergence rate than FSL for most of the problems, especially for higher generalization levels.

It was shown that the best value for the selection constant β is in fact problem dependent. Smaller β values do reduce computational effort substantially, but at the cost of degraded generalization performance.

As stated in section 4.1.1, not much research has been done in selective learning. The available literature does not contain sufficient information to reproduce experiments for comparative

purposes. However, a comparison between SASLA and ES has been done, which showed, for the problems investigated, SASLA to have significantly better generalization and convergence results, while ES did save on computations. The *circle* problem, with outliers added, illustrated ES to be affected more by the occurrence of outliers than SASLA.

It is proposed that future research beyond this thesis includes an investigation into the effects of dynamic β values, and larger subset selection intervals. Such a study may even further reduce the complexity of SASLA and increase its generalization performance.

4.4 Sensitivity Analysis Incremental Learning

Based on Shannon's sampling theorem, Malinowski and Zurada showed that there is an optimal number of training points for which best generalization is obtained, given a NN architecture and approximation accuracy (assuming continuous target functions on which a fast Fourier transform can be performed) [Malinowski *et al* 1993]. It is obvious beneficial to have such an optimal set of training points to approximate a function. Training time can be reduced by training on only the absolutely necessary data, and generalization improved, since the NN learner is not bombarded with confusing data points. However, calculation of such an optimal training set assumes knowledge of the function to be approximated and the computationally expensive calculation of the discrete Fourier transform of this function. In reality, we usually do not have prior knowledge about the function, but only a set of data points that captures the relationship between input parameters and function output.

Function approximation then involves training the NN on all the available training data. Section 4.1 discussed why we rather seek to implement mechanisms to trim the training set in order to train only on those patterns that convey the most information about the function to be approximated. One paradigm of such pattern selection algorithms is the set of incremental learning algorithms.

Incremental learning is an active learning strategy where the learner dynamically selects, during training, the most informative patterns from a candidate training set. Incremental learning algorithms prune the candidate set, and grows the actual training subset by adding

to the current training subset those patterns selected and removed from the candidate set. At each subset selection interval, the learner uses its current knowledge about the function to select patterns from the candidate set. After pattern selection, training continues on the increased training subset.

In general, the idea is to select the patterns that have the highest influence on the learning objective. The selected patterns should maximally decrease the discrepancy between the approximated function and the true function. Several methods have been developed to dynamically select the most influential patterns, differing in the measure of pattern informativeness. Measures of pattern informativeness include pattern error, expected MSE, and integrated squared bias (refer to section 4.1.1 which surveys these methods).

This section presents the Sensitivity Analysis Incremental Learning Algorithm (SAILA), which uses pattern sensitivity information as measure of informativeness. First order derivatives of the output units with respect to the input units are used to quantify the influence a pattern has on the output value of the function approximated by the network. Patterns with the highest influence on outputs cause the largest changes in weights to achieve maximum gain in bringing the approximation closer to the true function. In the development of this algorithm we consider only function approximation problems, including time series. However, SAILA can be applied to classification problems. Future research will include an empirical investigation into the applicability of SAILA to classification problems.

This is the first attempt to apply sensitivity analysis to incremental learning.

Section 4.4.1 discusses how sensitivity information can be used to select training patterns. A mathematical model of SAILA is presented in section 4.4.2, while a pseudocode algorithm of SAILA is given in section 4.4.3. This section also discusses algorithm design issues, including subset termination criteria, subset size, etc. The complexity of the algorithm is analyzed in section 4.4.4, while section 4.4.5 presents experimental results.

4.4.1 Pattern Selection Rationale

The objective of incremental learning is not to bombard the learner with all the information in the training set at once, but to incrementally present information about the problem - a

soft approach to learning. The learner is given the opportunity to first master the information it currently has before receiving more from the teacher (supervisor). In this way the learner is in control, and guides the learning process to request only information that will maximally increase its performance.

The selection of new information, or training patterns, must be done in a way to achieve maximum gain in terms of learning performance and speed of learning. SAILA has the viewpoint that the derivatives of the learned function embody the information to achieve this objective. The learner therefore uses the derivatives of the function as reflected by its current knowledge to guide the search for the most informative patterns. For this purpose SAILA uses the definition of *pattern informativeness* (definition 4.2, section 4.2.3) to assign an informative measure to patterns in the candidate training set. The learner then selects as new information those patterns with the highest informative measure.

But, what does this really mean? Chapter 2 illustrated in section 2.5 that a FFNN accurately approximates the first order derivatives of the learned function, using the analytical equations to calculate $\frac{\partial o_k}{\partial z_i^{(p)}}$ (defined in equation (E.6)). By recognizing that the peaks of the derivative of a continuous function represent the areas of the highest tempo of change in that function, a maximum change in the NN's weights can be achieved if training starts on patterns that lie in the region of derivative peaks. This is illustrated next from the sensitivity equations in appendix E and the learning equations in appendix D, assuming sigmoid activation functions.

From equation (E.6), for a pattern p,

$$\frac{\partial o_k}{\partial z_i^{(p)}} = f_{o_k}^{(p)'} \sum_{j=1}^{J} w_{kj} f_{y_j}^{(p)'} v_{ji} \tag{4.26}$$

The factors $f_{o_k}^{(p)'}$ and $f_{y_j}^{(p)'}$ occur in the equations to calculate weight changes during training. From equations (D.14) and (D.13),

$$\Delta w_{kj} = \eta (t_k^{(p)} - o_k^{(p)}) f_{o_k}^{(p)'} y_j \tag{4.27}$$

and from (D.21) and (D.20),

$$\Delta v_{ji} = \eta f_{y_j}^{(p)'} \sum_{k=1}^{K} (t_k^{(p)} - o_k^{(p)}) f_{o_k}^{(p)'} w_{kj} z_i \tag{4.28}$$

Thus, a change in $\frac{\partial o_k}{\partial z_i^{(p)}}$, due to a perturbation Δz_i of $z_i^{(p)}$, implies a change $\Delta f_{y_j}^{(p)'}$ in $f_{y_j}^{(p)'}$ and a change $\Delta f_{o_k}^{(p)'}$ in $f_{o_k}^{(p)'}$. This is because the output values $y_j^{(p)}$ of the hidden units and the output values $o_k^{(p)}$ of the output units change, since

$$\begin{aligned} f_{y_j}^{(p)'} &= y_j^{(p)}(1 - y_j^{(p)}) \\ f_{o_k}^{(p)'} &= o_k^{(p)}(1 - o_k^{(p)}) \end{aligned}$$

Because $\Delta w_{kj} \propto f_{o_k}^{(p)'}, \Delta v_{ji} \propto f_{o_k}^{(p)'}$ and $\Delta v_{ji} \propto f_{y_j}^{(p)'}$, a maximum change in weights is achieved for patterns p which have the highest absolute derivatives $|\frac{\partial o_k}{\partial z_i^{(p)}}|$, i.e. the most informative patterns.

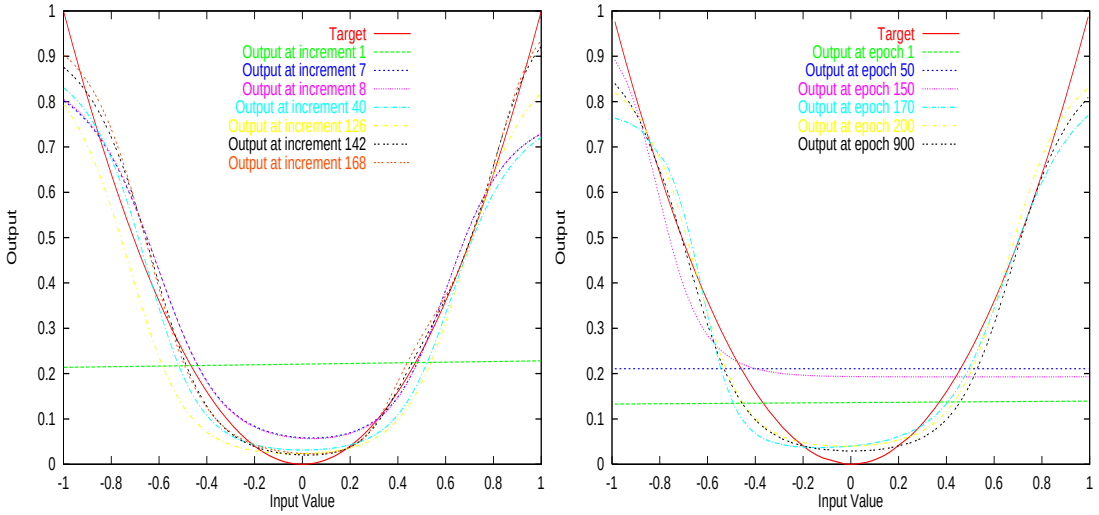
The effect of incrementally selecting and learning the most informative patterns is that the position of the hidden units are correctly fixed over input space early in training. Since the objective of learning in function approximation is to fit the hidden units over input space, much efficiency is gained if this process is speeded up, leaving only the fine tuning of the fit after the hidden units have been oriented correctly.

The selection of informative patterns is next illustrated with two simple one-dimensional functions. Performance results are postponed until section 4.4.5.

Function 1

The first illustrative example is to approximate the function $f(z) = z^2$, where $z \sim U(-1, 1)$. The candidate training set for SAILA consisted of 120 patterns, while FSL had a fixed training set of 120 patterns. A 1-3-1 NN architecture was used for both FSL and SAILA, with the same initial conditions. Figure 4.7 plots the NN output for different learning intervals for both SAILA (see figure 4.7(a)) and FSL (see figure 4.7(b)). For SAILA the NN output is plotted for different training increments, where an increment is the subset selection interval when a new pattern is added to the training subset. For FSL the output is plotted for selected epochs.

Figure 4.7(a) shows that very early in training, at increment 7 (epoch 75) which corresponds in total to only 399 pattern presentations, SAILA succeeded in approximating the shape of the target function. FSL only succeeded to approximate the shape of the true function after 170



(a) SAILA (b) FSL

Figure 4.7: SAILA and FSL output for $f(z) = z^2$

epochs, or 20 400 pattern presentations. The figures also show a much better approximation to the true function by SAILA compared to FSL. Even after FSL was trained for 1000 epochs (120 000 pattern presentations), its approximation is not as good as that of SAILA, which was stopped after increment 168 (98 586 pattern presentations, epoch 1000).

Figure 4.8 plots for selected increments the pattern informativeness values for all the patterns in the complete training set. This figure also indicates the pattern selected at each increment. Note how these patterns correspond to the peaks of the pattern informativeness graphs. For this problem a hidden unit is fitted over each side of the parabola. The figure illustrates how SAILA first selects patterns in the region where the output of each hidden unit is 0.5, then moving towards the asymptotic ends of the sigmoid activation functions.

Function 2

The next function was chosen to illustrate what happens when the peaks of the true function varies substantially (and consequently also the peaks of the derivative). The function $f(z) = \sin(2\pi z)e^{-z}$, as depicted by the solid line in figure 4.9, where $z \sim U(-1, 1)$, was trained using a 1-10-1 NN architecture. The output values were scaled to the range $[0, 1]$. The fixed training

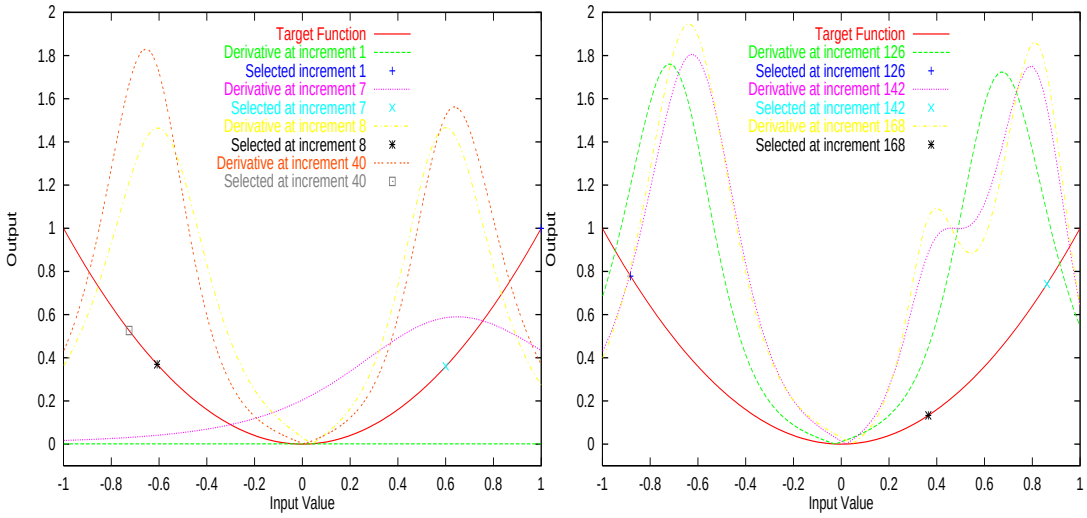


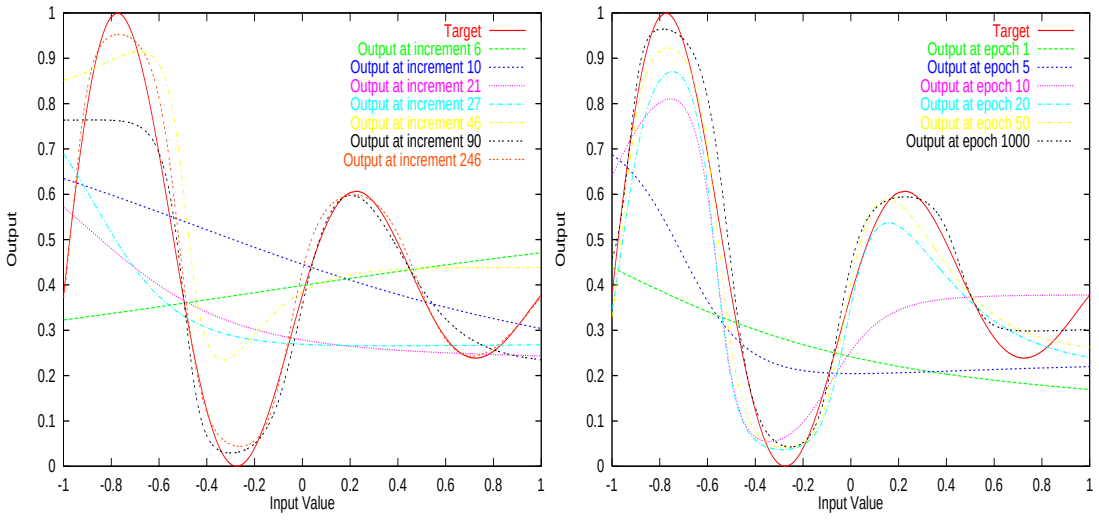
Figure 4.8: SAILA pattern informativeness for $f(z) = z^2$

set for FSL and the candidate training set for SAILA consisted of 600 patterns.

The evolution of the approximations by each learning algorithm is illustrated in figure 4.9 for selected increments and epochs. Figure 4.9(a) illustrates that SAILA starts learning the function from left to right, reason being that the derivatives are much larger on the left side. After increment 246 (epoch 996), which corresponds to 116 601 pattern presentations, a very good approximation to the entire function was obtained. In contrast, figure 4.9(b) shows that FSL had trouble approximating the function. FSL failed even after 1000 epochs, corresponding to 600 000 pattern presentations, to approximate the far right part of the function. SAILA was successful in this region of the function because the learner explicitly received information in this region when the corresponding patterns became most informative, allowing the network to accurately fit the hidden units in this difficult region.

Figure 4.10 plots the pattern informativeness values for each pattern at selected increments. This figure also shows how SAILA learned the function by first approximating the left side of the function with the larger derivatives, then gradually moving towards the right side when all the high sensitivity patterns of the left side have been removed from the candidate set. Note in the figure how the selected patterns correspond to the peaks of the pattern informativeness graphs.

The reader may now ask why don't we simply select those patterns from the candidate set that



(a) SAILA (b) FSL

Figure 4.9: SAILA and FSL output for $f(z) = \sin(2\pi z)e^{-z}$

have the highest error as done by Zhang [Zhang 1994] and Röbel [Röbel 1994a, Röbel 1994b, Röbel 1994c]. Surely, patterns that have the largest error do lead to large changes in weights due to the $(t_k^{(p)} - o_k^{(p)})$ term in the weight update equations (see equations (D.13) and (D.20)). However, this approach steps into trouble when the candidate set contains outlier patterns, for which a large error will be obtained. Error pattern selection algorithms will select these outlier patterns early in training. For sensitivity analysis pattern selection, the chance that outliers will be included in the training set is reduced, since the error term plays no explicit role in the sensitivity calculations. The inclusion of outliers in the training set depends on how far they lie from the peaks of the derivative of the approximated function. Recall that training in SAILA starts with those patterns close to derivative peaks, moving out - in the limit - to patterns with low sensitivity. Therefore, if outliers are located far from derivative peaks, their selection is postponed until all patterns with larger sensitivity have been selected. As illustrated in section 4.4.5, it is possible that training will stop without including low sensitivity patterns in the training set, thus excluding these outliers from the training set. It is only when outliers lie in the region of high derivative peaks that SAILA will include them in the training set.

Important to the success of incremental learning algorithms are subset termination criteria

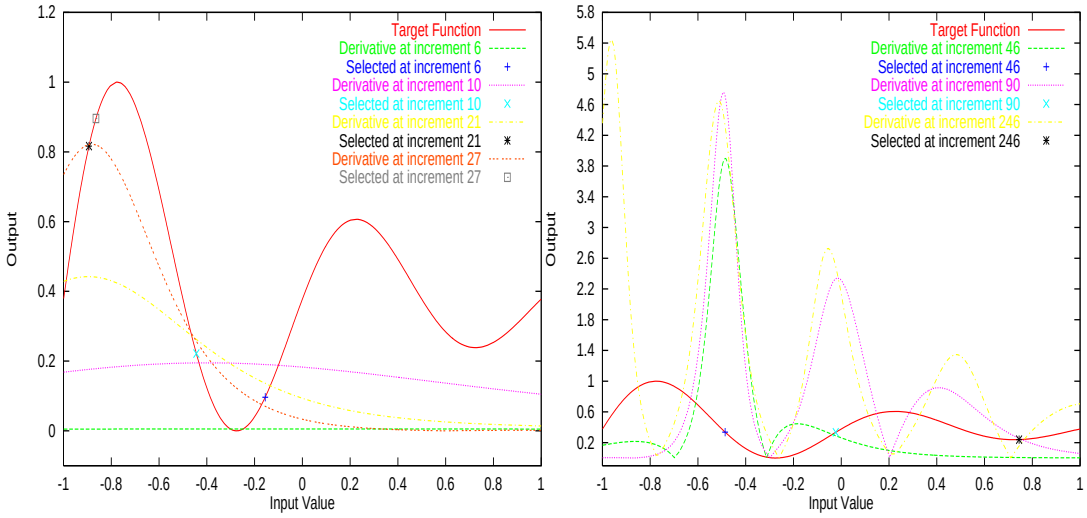


Figure 4.10: SAILA pattern informativeness for $f(z) = \sin(2\pi z)e^{-z}$

and subset sizes. These SAILA design issues are discussed in section 4.4.3.

4.4.2 Mathematical Model

Using the notations and definitions introduced in section 4.2.1, this section defines the SAILA operator, $\mathcal{A}^{+}_{SAILA}$, and shows how this operator is used to select new patterns from the candidate set D_C to be added to the training subset D_T. Define the SAILA operator as

$$\mathcal{A}^{+}_{SAILA}(D_C, D_T, \mathcal{F}_{NN}(D_T; W)) = \{p \in D_C | \Phi^{(p)}_{\infty} = \max_{q=1,\cdots,P_C} \{\Phi^{(q)}_{\infty}\}; \forall q \in D_C, \quad \text{not yet selected}\} \tag{4.29}$$

where $\Phi^{(p)}_{\infty}$ is defined in equation (4.2).

At subset selection interval τ_s, let $D_{S_s} \leftarrow \mathcal{A}^{+}_{SAILA}(D_C, D_T, \mathcal{F}_{NN}(D_T; W))$. That is, select from the candidate set D_C the subset $D_{S_s} \subseteq D_C$, with the number of patterns $P_{S_s} = |D_{S_s}|$ a fixed size. Then, let $D_T \leftarrow D_T \cup D_{S_s}$ and $D_C \leftarrow D_C - D_{S_s}$. Therefore, after selection interval τ_s, $D_T \leftarrow \cup_{s'=1}^{s} D_{S_{s'}}$, and $D_C \leftarrow D_C - \cup_{s'=1}^{s} D_{S_{s'}}$. The effect of the SAILA operator is therefore to gradually prune the candidate set and to grow the training set, by adding to the training set all patterns selected and removed from the candidate set.

Training for the next interval is done using the increased training subset D_T. The NN therefore trains only on $P_T = |D_T|$ patterns which is a subset of the original set of training patterns.

4.4.3 Incremental Learning Algorithm

A pseudocode algorithm for incremental learning using sensitivity analysis is given below. A general formulation is given to show SAILA's applicability to any differentiable activation function.

1. Initialize weights and learning parameters. Initialize the subset size, P_{S_s}, i.e. the number of patterns selected from the candidate set. Construct the initial training subset $D_{S_0} \subset D_C$. Let $D_T \leftarrow D_{S_0}$.

2. Repeat

   (a) Repeat

   Train the NN on training subset D_T

   until a termination criterion on D_T is triggered (discussed below).

   (b) Compute the new training subset D_{S_s} for the next subset selection interval τ_s:

   i. For each $p \in D_C$, compute the sensitivity matrix $S^{(p)}_{oz,ki}$ using equation (E.6) for sigmoid activation functions.

   ii. Compute the output sensitivity vector $\vec{S}^{(p)}_o$ for each $p \in D_C$ from equation (4.3).

   iii. Compute the informativeness $\Phi^{(p)}$ of each pattern $p \in D_C$ using equation (4.2).

   iv. Apply operator $\mathcal{A}^{+}_{SAILA}$ in (4.29) to find the subset D_{S_s} of the P_{S_s} most informative patterns. Then, let $D_T \leftarrow D_T \cup D_{S_s}$ and $D_C \leftarrow D_C - D_{S_s}$.

   until global convergence is reached.

The algorithm design issues specific to SAILA are discussed next:

- **Initial subset selection and size:** The SAILA implementation discussed in this thesis starts training on one pattern, selected from the candidate set D_C using the operator

defined in equation (4.29). This choice was made to investigate the efficiency of SAILA under a conservative initial subset size.

- **Subset selection criteria:** The SAILA operator $\mathcal{A}^{+}_{SAILA}$, defined in equation (4.29) embodies the subset selection criterion, which chooses the most informative pattern based on first order sensitivity information.

- **Subset size:** SAILA implements a very conservative subset size: at each subset selection interval, only one new pattern is selected from D_C and added to D_T.

- **Subset termination criteria:** The objective of SAILA is to continue training on the current training set D_T to achieve maximum gain from the new patterns before requesting new information. However, the learner should not be allowed to memorize the training subset, and should not spend too much time on D_T without achieving sufficient gain. For this purpose several termination criteria are incorporated and tested, after each sweep through D_T (after each training epoch), to test whether a new pattern needs to be added:

  1. The total number of training sweeps through the current training subset is limited to make sure that the NN does not indefinitely train on the set without achieving much.

  2. If the error on D_T or the validation set D_V decreases sufficiently (i.e. sufficient gain is achieved from learning on D_T), a new subset is selected. For the current implementation a new subset is selected when the error is decreased by 80% since training started on the current training subset. This may sound a bit strict, but other criteria will stop training on the subset if the learner cannot achieve this goal.

  3. If the average decrease in error per epoch for the current training interval is too small for the training set D_T or the validation set D_V, a new subset is selected. This measure will prevent the learner from training on D_T with achieving too little gain. Also, by considering the error on the validation set, overfitting is avoided. The current SAILA implementation scales the threshold for the average error per epoch linearly in the order of magnitude of the training error. The algorithm starts

with a threshold of 0.001, dividing it by 10 as the order of magnitude of the error decreases. If the average decrease in error for D_T or D_V is less than this threshold, a new subset is selected.

4. If the error $\mathcal{E}_V$ on the validation set increases too much, a new subset is selected. Currently, D_T is increased when $\mathcal{E}_V > \overline{\mathcal{E}}_V + \sigma_{\mathcal{E}_V}$, where $\overline{\mathcal{E}}_V$ is the average validation error over the previous epochs, and $\sigma_{\mathcal{E}_V}$ is the standard deviation in validation error. The goal of this termination test is to prevent the learner from memorizing the current training subset.

Learning by SAILA therefore amounts to the optimization of the empirical error function

$$\mathcal{E}(D_C; W) = \sum_{s=0}^{S} \mathcal{E}(\cup_{s'=0}^{s} D_{S_{s'}}; W) \tag{4.30}$$

by first optimizing $\mathcal{E}(D_{S_0}; W)$, then $\mathcal{E}(D_{S_0} \cup D_{S_1}; W)$, then $\mathcal{E}(D_{S_0} \cup D_{S_1} \cup D_{S_2}; W)$, etc.

4.4.4 Model Complexity

This section analyzes the complexity of SAILA, and compares it to that of FSL. For this purpose complexity is expressed as the number of calculations and comparisons made during training. Calculations include additions, subtractions, multiplications and divisions. It is also assumed that FSL implements a mechanism to avoid overfitting, similar to that implemented by SAILA. That is, after each epoch the validation error $\mathcal{E}_V$ on the validation set D_V is calculated to detect overfitting when $\mathcal{E}_V > \overline{\mathcal{E}}_V + \sigma_{\mathcal{E}_V}$. The cost of evaluation of the errors on D_T, D_C and D_V by SAILA is then the same as that by FSL, and can thus be ignored in the analysis of computational complexity.

Firstly, the computational cost of FSL is summarized from section 4.3.4. The total cost of learning after ξ epochs is, from equations (4.11) and (4.13),

$$\begin{aligned} \mathcal{C}_{FSL} &= \mathcal{T}_{FSL}(D_C)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) \\ &= \xi P_C (\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) \end{aligned} \tag{4.31}$$

To compute the computational cost of SAILA, assume that training starts on P_{S_0} patterns, that a fixed number of P patterns is selected at each selection interval, that there are in

total S selection intervals, and each selection interval consists of $\xi_s - \xi_{s-1}$ epochs. The total number of pattern presentations for SAILA after S selection intervals is

$$\mathcal{T}_{SAILA}(D_T) = \sum_{s=1}^{S}(\xi_s - \xi_{s-1})(P_{s_0} + (s-1)P) \tag{4.32}$$

where ξ_s is the epoch number corresponding to subset selection interval τ_s, and $\xi_0 = 0$. Equation (4.32) gives the total number of SAILA pattern presentations. The total number of patterns in D_T after S subset selection intervals is $|D_T| = \min\{SP, P_C\}$, while $|D_C| = \max\{P_C - SP, 0\}$.

The total cost of incremental training using sensitivity analysis is

$$\mathcal{C}_{SAILA} = \mathcal{C}_{\mathcal{A}^+_{SAILA}} + \mathcal{T}_{SAILA}(D_T)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) \tag{4.33}$$

which is the cost of applying the selection operator S times to the shrinking candidate set, plus the cost of training on the selected patterns in D_T. The cost of applying the SAILA operator is expressed as

$$\begin{aligned}
\mathcal{C}_{\mathcal{A}^+_{SAILA}} &= \mathcal{C}_{\mathcal{A}^+_{SAILA}}(D_{C_0}) + \mathcal{C}_{\mathcal{A}^+_{SAILA}}(D_{C_1}) + \cdots + \mathcal{C}_{\mathcal{A}^+_{SAILA}}(D_{C_S}) \\
&= \sum_{s=1}^{S} \mathcal{C}_{\mathcal{A}^+_{SAILA}}(D_{C_s}) \\
&= \sum_{s=1}^{S}\sum_{p=1}^{P_{C_s}}(\mathcal{C}_{S_{oz}}^{(p)} + \mathcal{C}_{\vec{S}_o}^{(p)} + \mathcal{C}_{\Phi}^{(p)}) + P\sum_{s=1}^{S} P_{C_s}
\end{aligned} \tag{4.34}$$

where D_{C_s} is the candidate set at subset selection interval τ_s, and P_{C_s} is the number of patterns in the candidate set at selection interval τ_s. The total cost to select P patterns from the current candidate set for all selection intervals is $P\sum_{s=1}^{S} P_{C_s}$. The cost $\mathcal{C}_{S_{oz}}^{(p)}$ of calculating the sensitivity matrix, the cost $\mathcal{C}_{\vec{S}_o}^{(p)}$ to compute the output sensitivity vector, and the cost $\mathcal{C}_{\Phi}^{(p)}$ to compute the informativeness of pattern p are respectively derived from equations (4.18), (4.19) and (4.20):

$$\begin{aligned}
\mathcal{C}_{S_{oz}}^{(p)} &= 3IJK \\
\mathcal{C}_{\vec{S}_o}^{(p)} &= IK \\
\mathcal{C}_{\Phi}^{(p)} &= K
\end{aligned}$$

Therefore,

$$\mathcal{C}_{\mathcal{A}^+_{SAILA}} = \sum_{s=1}^{S} P_{C_s} \times (3IJK + IK + K) + P\sum_{s=1}^{S} P_{C_s} \tag{4.35}$$

A cost saving using SAILA is obtained when

$$\begin{aligned} \mathcal{C} &= \mathcal{C}_{SAILA} - \mathcal{C}_{FSL} \\ &= \mathcal{C}_{\mathcal{A}^+_{SAILA}} + (\mathcal{T}_{SAILA}(D_T) - \mathcal{T}_{FSL}(D_C))(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) < 0 \end{aligned} \tag{4.36}$$

The computational complexity of SAILA depends on the subset size P and the number of subset selection intervals S. This dependency on P and S is not straightforward, but is based on a few trade-offs. If P is small, the incremental decrease in the size of the candidate training set is small, and the cost $\mathcal{C}_{\mathcal{A}^+_{SAILA}}$, as in equation (4.35), is large compared to when larger subset sizes are used. Also, smaller P may cause more subset selection intervals. On the other hand, larger P causes less subset selection intervals, but may cause longer training times on the larger subsets, thereby increasing costs. Equation (4.35) further shows that a large number of subset selection intervals also increases complexity due to an increased number of sensitivity evaluations over the current candidate training set D_{C_s}. However, coupled with a large P, the size of the candidate training set decreases more rapidly, with fewer patterns for which sensitivity calculations need to be performed.

So, when will SAILA be computationally less expensive than FSL? From equation (4.36), SAILA needs less computations when

$$\mathcal{C}_{\mathcal{A}^+_{SAILA}} + \mathcal{T}_{SAILA}(D_T)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) < \mathcal{T}_{FSL}(D_C)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) \tag{4.37}$$

From section 4.3.4, page 84, $(3IJK + IK + K) < (\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)})$, and since the number of subset selection intervals is less than the number of epochs, i.e. $S \leq \xi$, and $P_{C_s} \leq P_C$, we have that $\sum_{s=1}^{S} P_{C_s} \leq \xi P_C$. Then, for the current SAILA implementation where $P = 1$, $\mathcal{C}_{\mathcal{A}^+_{SAILA}} < \mathcal{T}_{FSL}(D_C)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)})$. Equation (4.37) will then be true when $\mathcal{T}_{FSL}(D_C) >> \mathcal{T}_{SAILA}(D_T)$, since $\mathcal{T}_{FSL}(D_C)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)}) >> \mathcal{T}_{SAILA}(D_T)(\mathcal{C}_W^{(p)} + \mathcal{C}_V^{(p)})$. The experimental results reported in the next section show this to be true.

4.4.5 Experimental Results

The objective of this section is to study the performance of SAILA through comparison of its generalization, overfitting, convergence and complexity with that of FSL. Two one-dimensional, one two-dimensional and two time series problems were used as benchmarks. The

Problem	Equation	Train/Test/ Validation Sets	Learning Rate	Momen- tum	Maximum Epochs	Archi- tecture
F1	(4.38)	120/40/40	0.1	0.9	2000	1-3-1
F2	(4.39)	600/200/200	0.1	0.9	2000	1-10-1
F3	(4.40)	600/200/200	0.1	0.9	2000	2-5-1
TS1	(4.41)	600/200/200	0.05	0.9	4000	2-5-1
TS2	(4.42)	180/60/60	0.05	0.9	1000	10-10-1

Table 4.18: Summary of the functions used to test SAILA

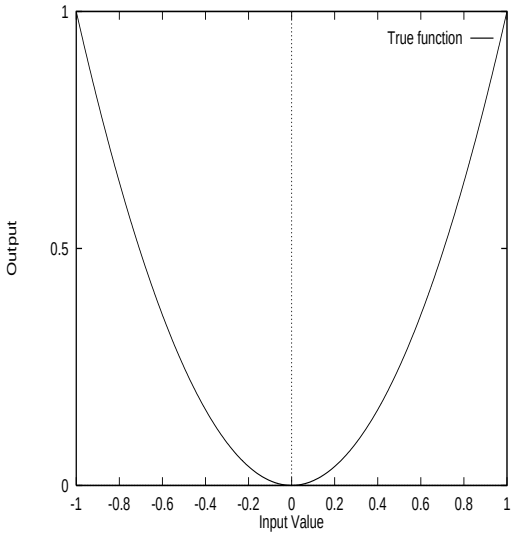
characteristics of these problems are described below, as well as the experimental procedure. An overview of the performance measures are given, after which the results of the experiments are presented.

While this section compares SAILA with normal fixed set learning, the author acknowledges the importance of detailed comparisons of SAILA with other incremental learning algorithms. Such a comparative study is currently being done by a master's student under the author's supervision. The interested reader is referred to [Adejuma 1999].

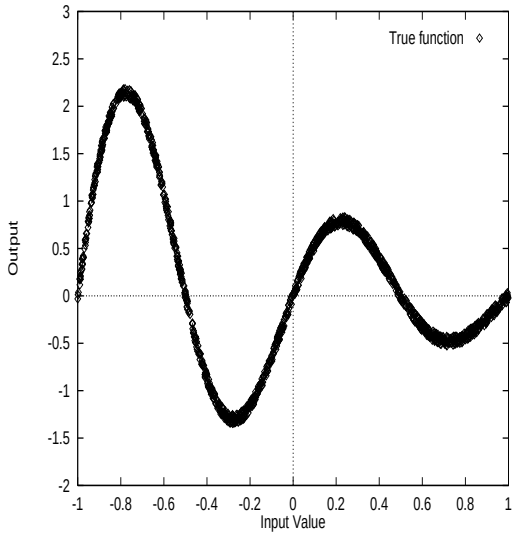
Experiments

Function approximation problems of varying complexity are selected to test the performance of the proposed incremental learning algorithm. The problems differ in input dimensions, the number of hidden units needed to solve the problem, and the smoothness of the function. A summary of these problems is presented in table 4.18. Figures 4.11 and 4.12 visualize each of the functions and time series. The learning parameters for each problem are also listed in table 4.18.

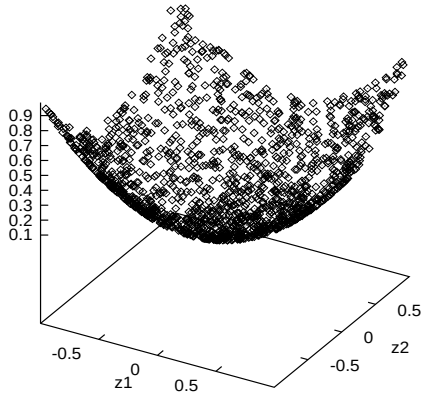
The architectures given in table 4.18 are not optimal for these problems. Oversized architectures were used to allow an investigation into the overfitting effects of the two learning algorithms. Figures 4.11 and 4.12 illustrate each problem, and serve as an indication of the complexity of the problems. A symbolic label is given for each problem for easy reference. Each function and time series is described next.



(a) Function F1

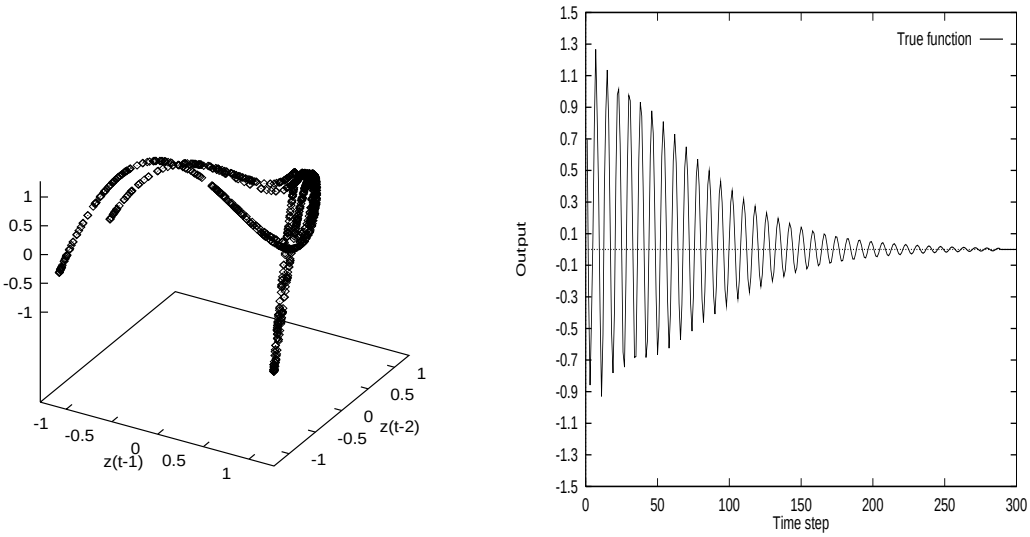


(b) Function F2



(c) Function F3

Figure 4.11: Functions used to test SAILA



(a) Time Series TS1

(b) Time Series TS2

Figure 4.12: Time series used to test SAILA

Function F1 is defined as

$$F1: \quad f(z) = z^2 \tag{4.38}$$

where $z \sim U(-1, 1)$. All the output values are in the range $[0, 1]$, and no noise was added. This function is illustrated in figure 4.11(a).

Function F2, as illustrated in figure 4.11(b), is defined as

$$F2: \quad f(z) = \sin(2\pi z)e^{-z} + \zeta \tag{4.39}$$

where $z \in U(-1, 1)$, and $\zeta \sim N(0, 0.1)$. Output values were scaled to the range $[0, 1]$.

Function F3, which is illustrated in figure 4.11(c), is defined as

$$F3: \quad f(z_1, z_2) = \frac{1}{2}(z_1^2 + z_2^2) \tag{4.40}$$

where $z_1, z_2 \sim U(-1, 1)$, and all outputs are in the range $[0, 1]$.

The henon-map, represented by figure 4.12(a), was selected as one of the time series problems to test SAILA. This times series, labelled TS1, is defined as

$$\begin{aligned} TS1: o_t &= z_t \\ z_t &= 1 + 0.3z_{t-2} - 1.4z_{t-1}^2 \end{aligned} \tag{4.41}$$

where $z_1, z_2 \sim U(-1, 1)$. The output values o_t were scaled such that $o_t \in [0, 1]$.

The last problem was a difficult time series with irrelevant parameters taken from [Cibas *et al* 1994a, Cibas *et al* 1994b, Cibas *et al* 1996], and is defined as

$$\begin{aligned} TS2 : o_t &= z_t \\ z_t &= 0.3z_{t-6} - 0.6z_{t-4} + 0.5z_{t-1} + 0.3z_{t-6}^2 - 0.2z_{t-4}^2 + \zeta_t \end{aligned} \tag{4.42}$$

where $z_t \sim U(-1, 1)$ for $t = 1, \cdots, 10$, and $\zeta_t \sim N(0, 0.05)$ is zero-mean noise sampled from a normal distribution. All output values were scaled to the range $[0, 1]$. This more difficult time series problem is illustrated in figure 4.12(b).

Experimental Procedure

The experimental procedure followed in this section is very similar to that of section 4.3.5. For each function, or experiment, 50 simulations were executed for both SAILA and FSL. The results reported are averages over these simulations. NNs that correspond to the simulations of a simulation pair had the same architecture, learning parameters, initial weights, and candidate training, test and validation sets. The candidate training set D_C, test set D_G and validation set D_V were randomly created such that $D_C \cap D_G = \emptyset$ and $D_C \cap D_V = \emptyset$. Table 4.18 summarizes the sizes of these data sets.

Performance Measures

The performance of SAILA is investigated through analysis of its generalization, training time, complexity and convergence. Generalization is expressed as the mean squared error (MSE) on the test sets. Röbel's generalization factor is used to illustrate the overfitting characteristics of the learning models (refer to section 4.2.2, page 73). Since the MSE is used as measure of the accuracy of the function approximation, a generalization factor $\rho > 1$ indicates that the generalization error $\mathcal{E}_G$ is larger than the training error $\mathcal{E}_T$. Therefore, if ρ increases during training, it serves as an indication that the learner is overfitting the training set (only when $\rho > 1$). When ρ decreases, the overfitting effects are reduced. If, at any learning interval, the generalization factor of one learning algorithm is larger than the generalization factor of

another, it does not mean that the former algorithm overfits more than the latter. Overfitting can only be concluded if ρ increases substantially above 1.

Training time is expressed as the number of pattern presentations, while the total number of calculations and comparisons as expressed in equation (4.36) serves as measure of computational complexity. The convergence characteristics of the two learning algorithms are evaluated as the percentage of the simulations that did not converge to specific generalization levels.

Results

This section presents results of the application of the sensitivity analysis incremental learning algorithm to the functions summarized in table 4.18 and figures 4.11 and 4.12. (An explanation of the working of SAILA was presented in section 4.4.1.) The results are summarized under four headings: **generalization performance, overfitting effects, complexity** and **convergence**.

Generalization Performance

Table 4.19 presents a summary of the training and generalization performances of SAILA and FSL for all functions and time series problems. The results listed are averages over the 50 simulations, together with 95% confidence intervals, as obtained at the final epoch (as given in table 4.18). Values in parentheses are estimates of the difference in performance between SASLA and FSL, calculated similarly to that of SASLA in equation (4.25). Generalization and training errors are expressed as the MSE over the respective data sets.

Table 4.19 show that SAILA performed significantly better than FSL for functions F1 and F3, and time series TS1. Although FSL performed better than SAILA for function F2, it is by a very small margin. For time series TS2, FSL had significantly lower training error than SAILA, but the two algorithms obtained very similar generalization performances (FSL severely overfitted the training set as discussed below).

Figures 4.13 and 4.18, which plot accuracy as a function of the number of pattern presentations

Problem	$\overline{\mathcal{E}}_T$	$\overline{\mathcal{E}}_G$	$\overline{\mathcal{E}}_G^{best}$	Presentation	$\overline{\rho}$
F1	(-0.123 ± 0.040)	(-0.119 ± 0.039)			(0.063 ± 0.046)
SAILA	0.037 ± 0.003	0.039 ± 0.003	0.039	179 260	1.069 ± 0.053
FSL	0.160 ± 0.040	0.158 ± 0.040	0.158	42120	1.007 ± 0.043
F2	(0.008 ± 0.005)	(0.009 ± 0.005)			0.035 ± 0.024
SAILA	0.035 ± 0.004	0.036 ± 0.004	0.036	100 000	1.032 ± 0.021
FSL	0.027 ± 0.003	0.027 ± 0.003	0.030	385 800	0.997 ± 0.018
F3	(-0.044 ± 0.023)	(-0.045 ± 0.024)			(-0.015 ± 0.029)
SAILA	0.011 ± 0.003	0.013 ± 0.003	0.006	182 250	1.004 ± 0.025
FSL	0.055 ± 0.023	0.056 ± 0.024	0.057	560 400	1.019 ± 0.024
TS1	(-0.080 ± 0.034)	(-0.081 ± 0.034)			(-0.004 ± 0.016)
SAILA	0.003 ± 0.0005	0.003 ± 0.0005	0.004	212 540	1.013 ± 0.017
FSL	0.083 ± 0.034	0.084 ± 0.034	0.085	363 000	1.016 ± 0.012
TS2	(0.019 ± 0.003)	(0.006 ± 0.005)			(-2.3034 ± 0.795)
SAILA	0.029 ± 0.003	0.033 ± 0.005	0.026	49 820	1.252 ± 0.208
FSL	0.009 ± 0.001	0.027 ± 0.004	0.027	180 000	3.555 ± 0.797

Table 4.19: Comparison of SAILA and FSL error performance measures

respectively for the functions and time series problems, illustrate that the generalization performance of SAILA was consistently better than that of FSL for all problems except for function F2 and time series TS2. Figure 4.13(b) does show that SAILA and FSL had approximately the same accuracies per pattern presentation for function F2. SAILA initially had a better generalization than FSL for time series TS2, but at the final epoch FSL achieved a slightly better generalization (refer to 4.18(b)).

Overfitting Effects

The effectiveness and importance of the validation error subset termination criteria to control overfitting are very evident from figures 4.14 and 4.19. These figures show how the generalization factor ρ for the two learning algorithms evolved during training. A prominent fluctuating overfitting characteristic is observed for SAILA. As soon as overfitting increased too much, a new training subset was selected. The selection of training subsets then continued until the overfitting effects were reduced. For all the problems ρ for SAILA was close to 1 which indicates no severe overfitting. FSL exhibited the same overfitting characteristics for all functions

except time series TS2 (figure 4.19(b)), for which FSL substantially overfitted since training started (also refer to table 4.19 which show that FSL had a very high average generalization factor of 3.555 at the final epoch). After 60 000 presentations, FSL started to overfit function F2 (figure 4.14(b)), having a continuous increase in ρ. For function F3 (figure 4.14(c)) a generalization factor less than 1 was obtained for SAILA throughout training, whereas FSL achieved this only after 34 500 pattern presentations.

Complexity

Using equation (4.36) to calculate the cost saving by using SAILA instead of FSL, table 4.20 illustrates that SAILA was computationally much less demanding than FSL. For all the experiments SAILA required much less pattern presentations, which resulted in the large cost savings - even for the conservative subset size of one pattern. For all problems, except function F1, the savings in computational cost by using SAILA increased linearly as a function of the epochs. Figure 4.16 shows that the computational cost for function F1 did not follow this trend. For function F1, FSL became computationally more efficient than SAILA after 1 114 epochs (corresponding to 133 680 FSL pattern presentations). At this point most of the patterns from the candidate set have been selected (refer to figure 4.17(a)). However, FSL did not really gain anything, since SAILA achieved a much lower error than FSL, even after 4 060 SAILA pattern presentations (corresponding to 263 epochs, using only 33 of the candidate patterns for training). At this point SAILA was less expensive to use.

The cost effectivity of SAILA is further illustrated in the fifth column of table 4.20, which lists the number of presentations required by the different algorithms to reach their best generalization performances. For all problems, except function F1, SAILA used substantially less presentations to reach its best generalization. Note that although FSL used less presentations, its best generalization is higher than that of SAILA.

The lower computational cost by SAILA was made possible by the fact that SAILA used only a small percentage of the candidate set to achieve its good generalization results. Figure 4.17 shows how the size of the actual training set growed during training for each of the problems. Table 4.21 shows for selected epochs the percentage of the original candidate set that was used

Problem	$\overline{P}_T$	Total presented patterns	Cost Saving ($\times 10^6$)
F1			
SAILA	120	185 627.34 ± 5 823.73	−8.013119 ± 0.838617
FSL	120	240 000	
F2			
SAILA	97.34 ± 13.14	156 720.27 ± 10 302.30	−584.538752 ± 9.664226
FSL	600	1 200 000	
F3			
SAILA	107.02 ± 8.50	105 220.36 ± 8 053.00	−319.836765 ± 2.404214
FSL	600	1 200 000	
TS1			
SAILA	252.92 ± 23.87	436 258.54 ± 36 259.02	−573.114675 ± 10.935313
FSL	600	1 200 000	
TS2			
SAILA	85.48 ± 9.93	31 428.68 ± 3 830.49	−209.530168 ± 5.986019
FSL	180	180 000	

Table 4.20: Comparison of SAILA and FSL computational complexity

by SAILA for training. This table also shows in comparison with FSL how many patterns were in the training subset at the final epoch (as given in table 4.18), and the point of best average generalization performance. Except for function F1, SAILA used substantially less patterns than the size of the candidate set to reach its best generalization errors.

	Growth after epoch					Number of Patterns in		
Problem	50	200	600	1000	2000	D_C	in D_T at final epoch	at best generalization
F1	5.9%	18.8%	72.0%	92.8%	100%	120	120	100
F2	1.1%	2.9%	9.8%	12.6%	16.2%	600	97	97
F3	0.5%	0.9%	5.2%	9.1%	16.8%	600	107	98
TS1	0.4%	0.5%	3.0%	7.0%	16.9%	600	253	235
TS2	1.0%	1.4%	20.5%	47.5%	-	180	85	50

Table 4.21: Training subset growth by SAILA

Convergence

The convergence performance of SAILA is compared to that of FSL in figures 4.15 and 4.20. These figures plot the percentage of simulations that did not reach specific generalization errors. SAILA had exceptionally better convergence results than FSL for function F1 (figure 4.15(a)) and time series TS1 (figure 4.20(a)). For function F1, FSL failed to converge even for high generalization levels of 0.1 (42% of the simulations did not converge at this level). SAILA, on the other hand, had a 100% convergence up to a 0.054 generalization level. For time series TS1, FSL also had non-convergent simulations from a high generalization level of 0.1 (32% of the simulations did not converge), whereas SAILA had a 100% convergence up to a level of 0.01. For function F3 (figure 4.15(c)) SAILA had much better convergence results than FSL up to a generalization level of 0.005, after which FSL performed better. For this problem FSL also started to have non-convergent simulations from a generalization error of 0.1, for which 20% of the simulations did not converge. SAILA had non-convergent simulations only from a level of 0.028. The convergence results for time series TS2 (figure 4.20(b)) were very similar for the two learning algorithms. For function F2 (figure 4.15(b)) FSL showed better convergence than SAILA.

In general, SAILA had very good convergence properties compared to FSL.

Conclusive Remarks

Section 4.4 presented a new incremental learning algorithm that uses pattern informativeness to select from a candidate set the patterns that convey the most information about the function being approximated. The problems investigated in this section show that SAILA presented exceptionally well generalization results compared to FSL, and exhibited good convergence results compared to that of FSL. The results presented show a definite saving in computational cost using SAILA.

The results presented were for a conservative subset size of only one pattern. It is proposed that future research beyond this thesis includes an investigation into larger subset sizes.

Results published in the literature for the incremental learning algorithms overviewed in

Algorithm	**Epoch**	$\overline{\mathcal{E}}_T$	$\overline{\mathcal{E}}_G$	$\overline{P}_T$
SAILA	2000	0.003 ± 0.0002	0.003 ± 0.0002	100 ± 9
Röbel	2000	0.004 ± 0.001	0.003 ± 0.001	69 ± 10

Table 4.22: Comparison of SAILA with Röbel on henon-map

section 4.1.1 are not appropriate for comparison with SAILA, since incomplete information about the experimental procedures are given in these publications. This prompted the author of this thesis to commission a masters student to perform an in-depth comparison of the performances of incremental learning algorithms under clean and noisy data [Adejuma 1999]. However, a single set of results from [Röbel 1994b], concerning the henon-map (TS1), can be compared with that of SAILA. Table 4.22 concludes this section to show that very similar performance results were obtained by the two incremental learning algorithms. The results show Röbel to have a larger variance in performance than SAILA, whose results are more concentrated around the average performance.

4.5 Conclusion

This chapter presented two new active learning algorithms: a selective learning algorithm for classification problems (SASLA), and an incremental learning algorithm for function approximation problems (SAILA). These active learning algorithms use a measure of pattern informativeness to dynamically select, during training, those patterns from a candidate set that convey the most information of the problem being learned. First order derivatives of the NN output function with respect to input parameters is used to quantify the informativeness of patterns. For classification problems the most informative patterns lie closest to the decision boundaries, and encapsulate the optimal position of these boundaries. For function approximation problems the most informative patterns describe the areas of the approximated function where the tempo in change of the function output is very high. These patterns cause the largest changes in weights.

The selective learning algorithm starts training on the entire candidate set and prunes uninformative patterns from the training set, while incremental learning starts on a subset of the

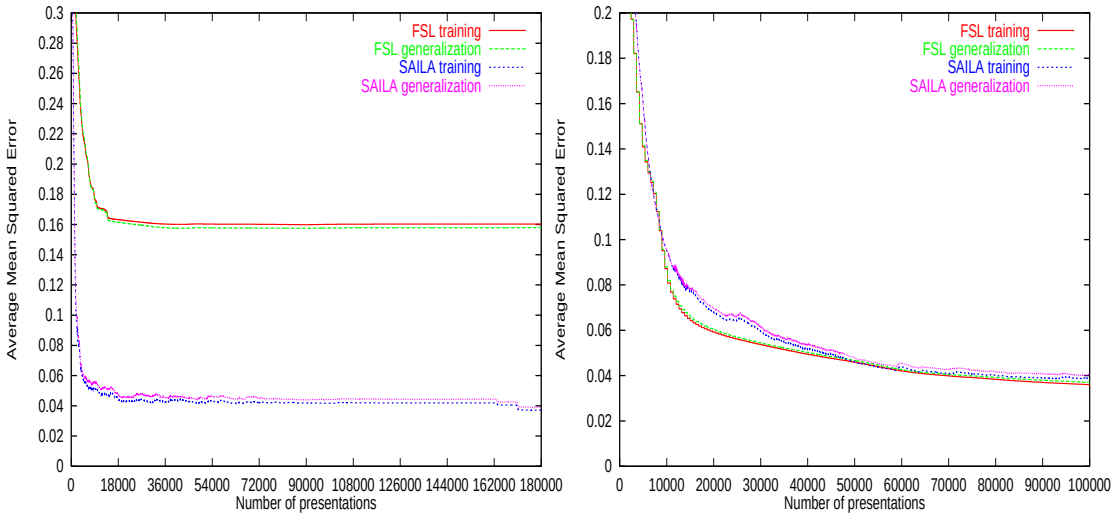
candidate set and incrementally adds the most informative patterns.

The results presented showed both approaches to be very effective, resulting in good generalization performance compared to FSL. The new algorithms exhibited good convergence and overfitting properties. Both active learning approaches resulted in a substantial reduction in computational cost compared to FSL.

The implementations discussed in this chapter make use of very conservative design choices, and were applied to feedforward NNs only. Research beyond this thesis will include

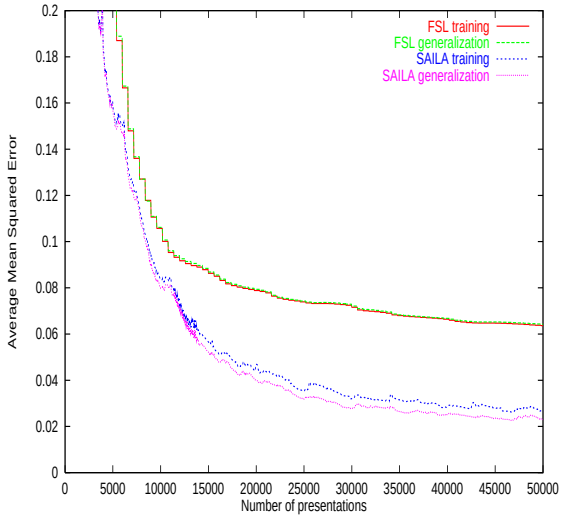
- an investigation of the performance of the algorithms for less restrictive values of the subset sizes;
- the implementation of dynamic β values for SASLA;
- an investigation into the applicability of SAILA to classification problems;
- a study of the application of the proposed active learning algorithms to different NN types, including recurrent NNs, functional link NNs and product unit NNs; and
- a study of the performance of the algorithms under different levels of noise and the occurrence of outlier patterns in the training set.

A comparative study of different incremental learning algorithms (including SAILA) is currently being done by Adejuma [Adejuma 1999].



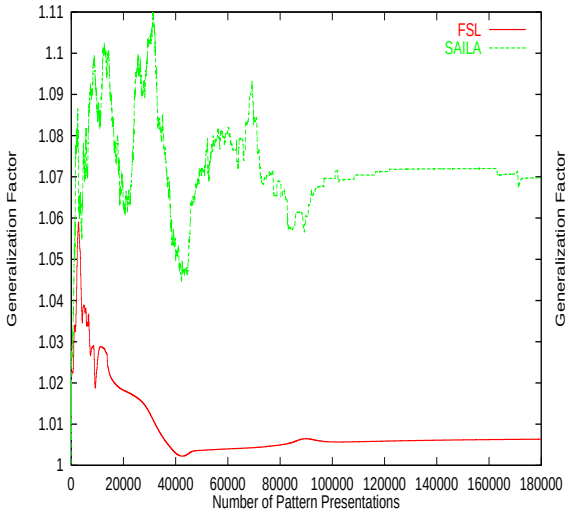
(a) Function F1

(b) Function F2

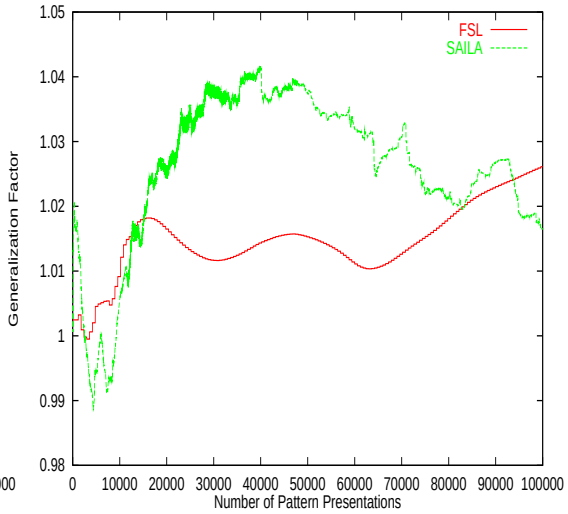


(c) Function F3

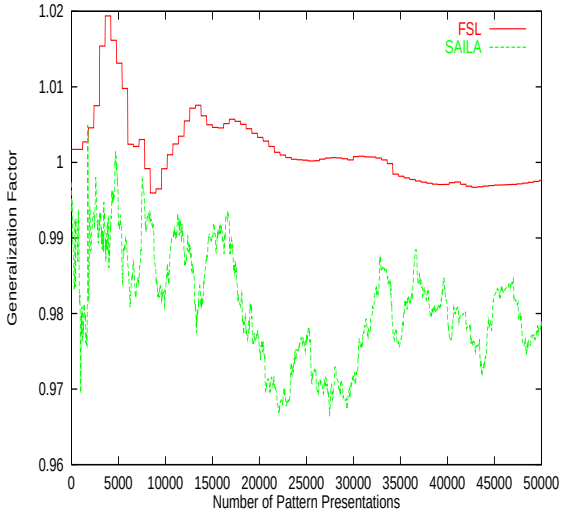
Figure 4.13: Average MSE vs presentations for function approximation problems



(a) Function F1

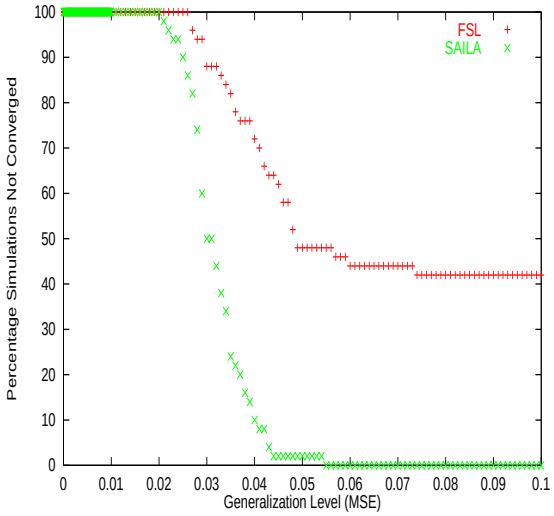


(b) Function F2

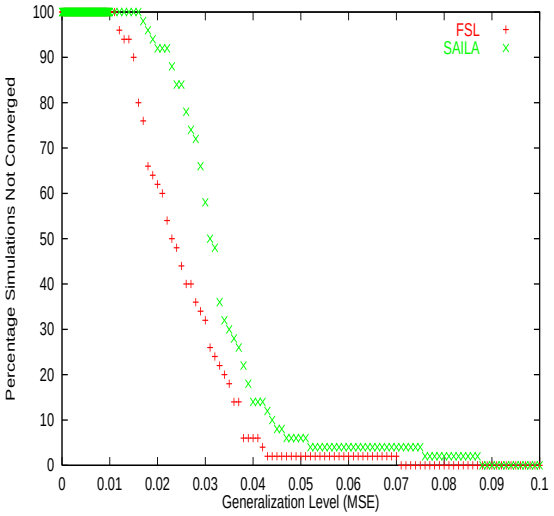


(c) Function F3

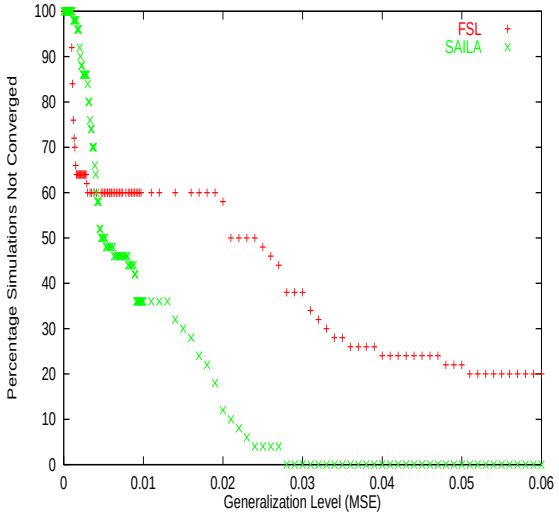
Figure 4.14: Average generalization factor vs generalization levels for function approximation problems



(a) Function F1



(b) Function F2



(c) Function F3

Figure 4.15: Percentage simulations that did not converge to generalization levels for function approximation problems

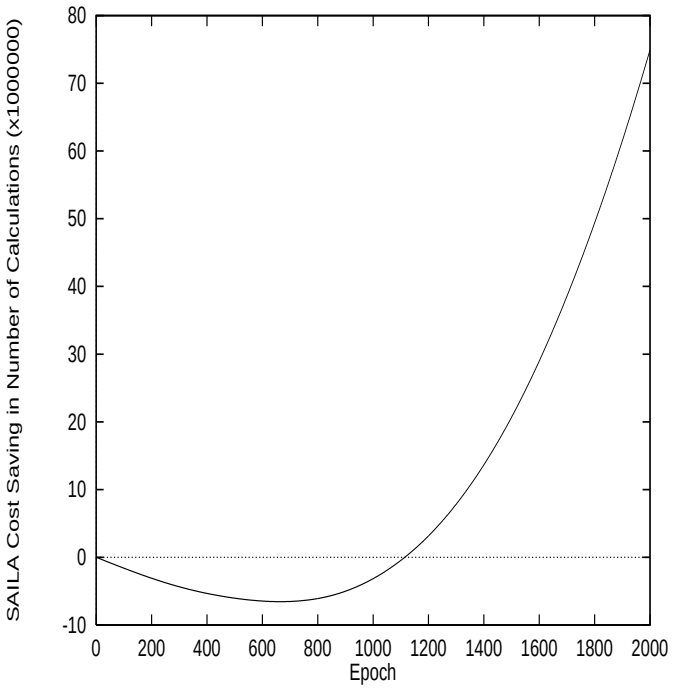
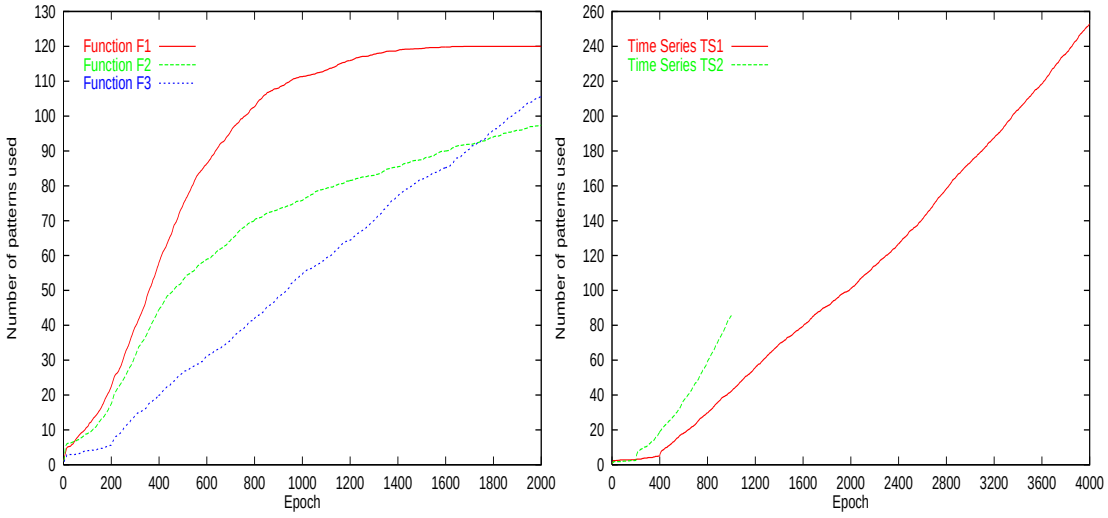


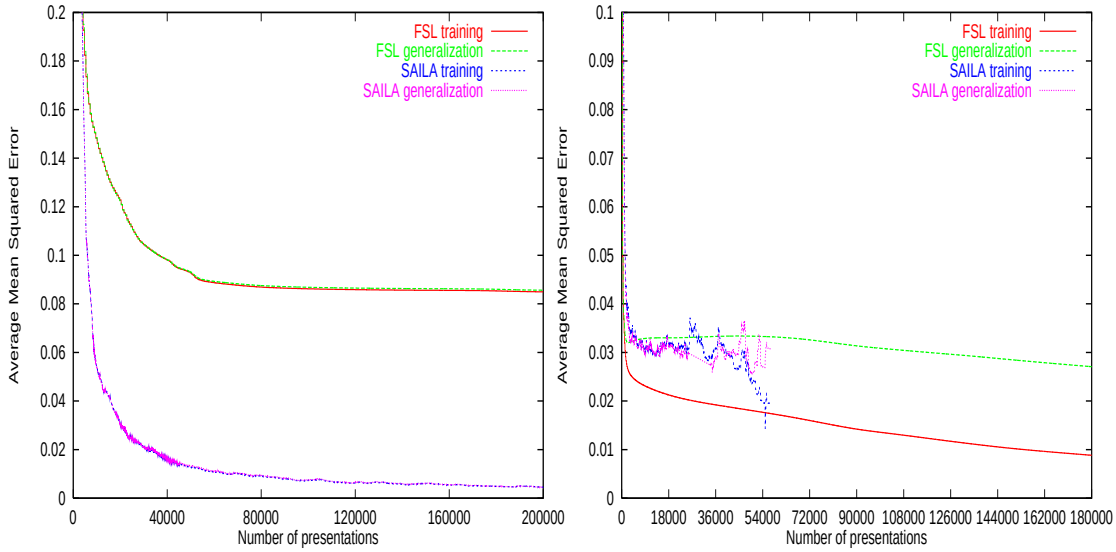
Figure 4.16: SAILA cost saving for function F1



(a) For function approximation problems

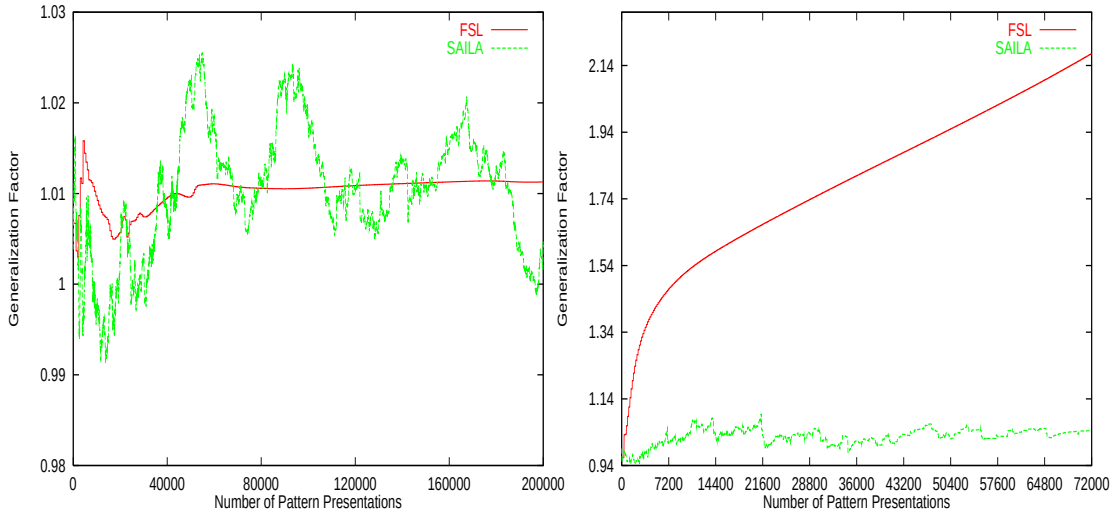
(b) Function F2

Figure 4.17: Average number of patterns used per epoch by SAILA



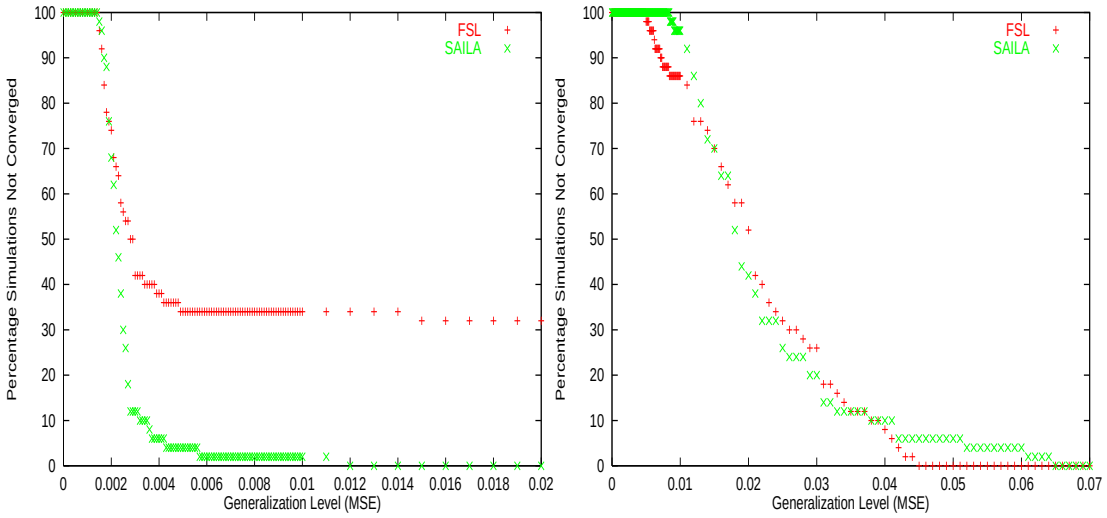
(a) Time Series TS1 (b) Time Series TS2

Figure 4.18: Average MSE vs presentations for time series problems



(a) Time Series TS1 (b) Time Series TS2

Figure 4.19: Average generalization factor vs generalization levels for time series problems



(a) Time Series TS1

(b) Time Series TS2

Figure 4.20: Percentage simulations that did not converge to generalization levels for time series problems

Chapter 5

Architecture Selection using Sensitivity Analysis

"Plurality is not to be assumed without necessity"
-William of Ockham (1285-1349)

This chapter presents an algorithm to prune feedforward NN architectures using sensitivity analysis. Sensitivity information is used to quantify the significance of input and hidden units, and a statistical test based on variance analysis is used to decide which units to prune.

5.1 Introduction

Generalization is a very important aspect of neural network learning. Since it is a measure of how well the network interpolates to points not used during training, the ultimate objective of NN learning is to produce a learner with low generalization error. That is, to minimize the true risk function

$$\mathcal{E}_G(\Omega; W) = \int (\mathcal{F}_{NN}(\vec{z}, W) - \vec{t})^2 d\Omega(\vec{z}, \vec{t}) \tag{5.1}$$

where, from chapter 1, $\Omega(\vec{z}, \vec{t})$ is the stationary density according to which patterns are sampled, W describes the network weights, and $\vec{z}$ and $\vec{t}$ are respectively the input and target vectors. The function $\mathcal{F}_{NN}$ is an approximation of the true underlying function. Since Ω is

generally not known, $\mathcal{F}_{NN}$ is found through minimization of the empirical error function

$$\mathcal{E}_T(D_T; W) = \frac{1}{P_T} \sum_{p=1}^{P_T} (\mathcal{F}_{NN}(\vec{z}^{(p)}, W) - \vec{t}^{(p)})^2 \tag{5.2}$$

over a finite data set $D_T \sim \Omega$. When $P_T \to \infty$, then $\mathcal{E}_T \to \mathcal{E}_G$. The aim of NN learning is therefore to learn the examples presented in the training set well, while still providing good generalization to examples not included in the training set. It is however possible that a NN exhibits a very low training error, but bad generalization due to overfitting (memorization) of the training patterns.

This trade-off between training error and generalization has prompted many research in the generalization performance of NNs. Average generalization performance has been studied theoretically to better understand the behavior of NNs trained on a finite data set. Research shows a dependence of generalization error on the training set, the network architecture and weight values. Chapter 4, and the references therein, illustrated the influence of the training set on generalization performance. Schwartz, Samalam, Solla and Denker show the importance of training set size for good generalization in the context of ensemble networks [Schwartz *et al* 1990]. Other research uses the VC-dimension [Abu-Mostafa 1989, Abu-Mostafa 1993, Cohn *et al* 1991, Opper 1994] to derive bounds on the generalization error as a function of network and training set size. Best known are the bounds derived by Baum and Hausler [Baum *et al* 1989] and Hausler, Kearns, Opper and Schapire [Haussler *et al* 1992]. While these bounds are derived for, and therefore limited to, discrete input values, Hole derives generalization bounds for real valued inputs [Hole 1996].

Bounds on generalization have also been developed by studying the relationship between training error and generalization error. Based on Akaike's Final Prediction Error (FPE) and Information Criterion (AIC) [Akaike 1974], Moody derives the Generalized Prediction Error (GPE) which gives a bound on the generalization error as a function of the training error, training set size, the number of effective parameters, and the effective noise variance [Moody 1992, Moody 1994a]. Murata, Yoshizawa and Amari derive a similar Network Information Criterion [Murata *et al* 1991, Murata *et al* 1994a, Murata *et al* 1994b]. Using a different approach, i.e. Vapnik's Bernoulli theorem, Depenau and Møller derive a bound as a function of training error, the VC-dimension and training set size [Depenau *et al* 1994].

These research results give, sometimes overly pessimistic, bounds that help to clarify the behavior of generalization and its relationship with architecture, training set size and training error. Another important issue in the study of generalization is that of overfitting. Overfitting means that the NN learns too much detail, effectively memorizing training patterns. This normally happens when the network complexity does not match the size of the training set, i.e. the number of adjustable weights (free parameters) is larger than the number of patterns. If this is the case, the weights learn individual patterns, and even capture noise. This overfitting phenomenon is the consequence of training on a finite data set, minimizing the empirical error function given in equation (5.2), which differs from the true risk function given in equation (5.1).

Amari *et al* developed a statistical theory of overtraining in the asymptotic case of large training set sizes [Amari *et al* 1995, Amari *et al* 1996]. They analytically determine the ratio in which patterns should be divided into training and test sets to obtain optimal generalization performance and to avoid overfitting. Overfitting effects under large, medium and small training set sizes have been investigated analytically by Amari *et al* [Amari *et al* 1995] and Müller *et al* [Müller *et al* 1995].

Remedies of overfitting have been studied theoretically and empirically:

- **Training set size**: The best way to avoid overfitting is to use a large training set. From the central limit statement [Mitchell 1997], the empirical error function converges to the true risk function as the number of training patterns tends to infinity. Amari *et al* show that if the training set contains at least 30 times as many patterns as there are weights, the network is unlikely to suffer from overfitting [Amari *et al* 1995]. For noise free data, less patterns may be sufficient. While large training set sizes do prevent overfitting, chapter 4 showed that redundancy in large training sets may also lead to overfitting. Therefore, provided that large training sets do not contain redundancy, effects of overfitting can be reduced.

- **Estimations of generalization ability**: During training, the generalization error is constantly monitored to stop training as soon as performance deteriorates. Two methods of generalization estimation have been used, i.e. early stopping and theoretical

generalization bounds:

- **Early stopping (stopped training)** [Amari *et al* 1995, Hassoun 1995, Reed 1993, Sarle 1995]: It is generally observed from simulations that generalization initially decreases early in training, reaches a minimum, and then starts to increase, while the training error continues to decrease. The ideal is to stop training when the generalization error starts to increase, i.e. when overfitting steps in. To find this point of overfitting, patterns not used in the test set are divided into a training set and a validation set. During training, the validation set - which is not used for weight adjustments - is used to estimate the generalization performance. Training is stopped at the point where the validation error is minimized.

  Early stopping is fast and easy to implement. Sarle shows empirically that early stopping is efficient [Sarle 1995]. However, the technique is impractical for small data sets, since the sizes of the training and validation sets will be insufficient. Early stopping is also plagued by the question of how many patterns should be allocated to the validation set. Sarle presents answers to this question for single input problems [Sarle 1995], while Amari *et al* present a theoretical analysis to determine the optimal sizes of the training and validation sets [Amari *et al* 1995].

- **Generalization bounds**: Theoretical generalization bounds, as overviewed earlier in this chapter, can be used to estimate the generalization error during training [Baum *et al* 1989, Depenau *et al* 1994, Hole 1996, Moody 1992, Murata *et al* 1994a, Murata *et al* 1994b]. When the generalization error exceeds the estimates, training should be stopped. This approach may not always be effective, since these bounds are usually overly pessimistic.

- **Noise**: Artificial noise is added to inputs during training [Holmström *et al* 1992]. Provided that the noise is sampled from a distribution having small variance and zero mean, it can be assumed that the resulting changes in network output have insignificant consequences. The addition of noise to existing patterns effectively generates new training patterns, thereby reducing the chances of overfitting.

- **Architecture Selection**: Referring to one of Ockham's statements, if several networks fit the training set equally well, then the simplest network (i.e. the network with the

smallest number of weights) will on average give the best generalization performance [Thodberg 1991]. This hypothesis has been investigated and confirmed by Sietsma and Dow [Sietsma *et al* 1991]. A network with too many free parameters may actually memorize training patterns and may also accurately fit the noise embedded in the training data, leading to bad generalization. Overfitting can thus be prevented by reducing the size of the network through elimination of individual weights or units. The objective is therefore to balance the complexity of the network with goodness of fit of the true function. This process is referred to as *architecture selection*. Several approaches have been developed to select the optimal architecture, i.e. regularization, network construction (growing) and pruning. These approaches will be overviewed in more detail below.

This thesis considers architecture selection as one of the remedies for overfitting. Learning is not just perceived as finding the optimal weight values, but also to find the optimal architecture. However, it is not always obvious what architecture is the best. Finding the ultimate best architecture requires a search over all possible architectures. For large networks an exhaustive search is prohibitive, since the search space consists of 2^w architectures, where w is the total number of weights [Moody *et al* 1995]. Instead, heuristics are used to reduce the search space. A simple method is to train a few networks of different architecture and to choose the one which results in the lowest generalization error as estimated from the GPE [Moody 1992, Moody 1994a] or the Network Information Criterion (NIC) [Murata *et al* 1991, Murata *et al* 1994a, Murata *et al* 1994b]. This approach is still expensive and requires many architectures to be investigated to reduce the possibility that the optimal model is not found. The NN architecture can alternatively be optimized by trial and error. An architecture is selected, and its performance is evaluated. If the performance is unacceptable, a different architecture is selected. This process continues until an architecture is found which produces an acceptable generalization error.

Other approaches to architecture selection are divided into three categories:

- **Regularization**: Neural network regularization involves the addition of a penalty term to the objective function to be minimized. In this case the objective function changes

to

$$\mathcal{E} = \mathcal{E}_T + \lambda \mathcal{E}_C \tag{5.3}$$

where $\mathcal{E}_T$ is the usual measure of data misfit, and $\mathcal{E}_C$ is a penalty term, penalizing network complexity (network size). The constant λ controls the influence of the penalty term. With the changed objective function, the NN now tries to find a locally optimal trade-off between data-misfit and network complexity. Neural network regularization has been studied rigorously by Girosi, Jones and Poggio [Girosi *et al* 1995], and Williams [Williams 1995].

Several penalty terms have been developed to reduce network size automatically during training. Weight decay, where $\mathcal{E}_C = \frac{1}{2}\sum w_i^2$, has as objective to drive small weights to zero [Bös 1996, Hanson *et al* 1989, Kamimura *et al* 1994a, Krogh *et al* 1992]. It is a simple method to implement, but suffers from penalizing large weights at the same rate as small weights. To solve this problem, Hanson and Pratt propose the hyperbolic and exponential penalty functions which penalize small weights more than large weights [Hanson *et al* 1989]. Nowlan and Hinton developed a more complicated soft weight sharing, where the distribution of weight values is modeled as a mixture of multiple Gaussian distributions [Nowlan *et al* 1992]. A narrow Gaussian is responsible for small weights, while a broad Gaussian is responsible for large weights. Using this scheme, there is less pressure on large weights to be reduced.

Weigend, Rumelhart and Huberman propose weight elimination where the penalty function $\mathcal{E}_C = \sum \frac{w_i^2/w_0^2}{1+w_i^2/w_0^2}$, effectively counts the number of weights [Weigend *et al* 1991]. Minimization of this objective function will then minimize the number of weights. The constant w_0 is very important to the success of this approach. If w_0 is too small, the network ends up with a few large weights, while a large value results in many small weights. The optimal value for w_0 can be determined through cross-validation, which is not cost effective.

Chauvin introduces a penalty term which measures the "energy spent" by the hidden units, where the energy is expressed as a function of the squared activation of the hidden units [Chauvin 1989, Chauvin 1990]. The objective is then to minimize the energy spent by hidden units, and in so doing, to eliminate unnecessary units.

Kamimura and Nakanishi show that, in an information theoretical context, weight decay actually minimizes entropy [Kamimura *et al* 1994a]. Entropy can also be minimized directly by including an entropy penalty term in the objective function [Kamimura 1993, Kamimura *et al* 1994b]. Minimization of entropy means that the information about input patterns is minimized, thus improving generalization. For this approach entropy is defined with respect to hidden unit activity. Schittenkopf, Deco and Brauer also propose an entropy penalty term and show how it reduces complexity and avoids overfitting [Schittenkopf *et al* 1997].

Yasui develops penalty terms to make minimal and joint use of hidden units by multiple outputs [Yasui 1997]. Two penalty terms are added to the objective function to control the evolution of hidden-to-output weights. One penalty causes weights leading into an output unit to prevent one another from growing, while the other causes weights leaving a hidden unit to support one another to grow.

While regularization models are generally easy to implement, the value of the constant λ in equation (5.3) may present problems. If λ is too small, the penalty term will have no effect. If λ is too large, all weights might be driven to zero. Regularization therefore requires a delicate balance between the normal error term and the penalty term. Another disadvantage of penalty terms is that they tend to create additional local minima [Hanson *et al* 1989], increasing the possibility of converging to a bad local minimum. Penalty terms also increase training time due to the added calculations at each weight update. In a bid to reduce this complexity, Finnoff, Hergert and Zimmermann show that the performance of penalty terms is greatly enhanced if they are introduced only after overfitting is observed [Finnoff *et al* 1993].

- **Network construction (growing)**: Network construction algorithms start training with a small network and incrementally add hidden units during training when the network is trapped in a local minimum [Fritzke 1995, Hirose *et al* 1991, Huang 1994, Hüning 1993, Jutten *et al* 1995b, Kwok *et al* 1995, Moody *et al* 1996, Wynne-Jones 1992]. A small network forms an approximate model of a subset of the training set. Each new hidden unit is trained to reduce the current network error - yielding a better approximation. Crucial to the success of construction algorithms is

effective criteria to trigger when to add a new unit, when to stop the growing process, where and how to connect the new unit to the existing architecture, and how to avoid restarting training. If these issues are treated on an ad hoc basis, overfitting may occur and training time may be increased.

- **Network pruning**: Neural network pruning algorithms start with an oversized network and remove unnecessary network parameters, either during training or after convergence to a local minimum. Network parameters that are considered for removal are individual weights, hidden units and input units. The decision to prune a network parameter is based on some measure of parameter relevance or significance. A relevance is computed for each parameter and a pruning heuristic is used to decide when a parameter is considered as being irrelevant or not. A large initial architecture allows the network to converge reasonably quickly, with less sensitivity to local minima and the initial network size. Larger networks have more functional flexibility, and is guaranteed to learn the input-output mapping with the desired degree of accuracy. Due to the larger functional flexibility, pruning weights and units from a larger network may give rise to a better fit of the underlying function, hence better generalization [Moody 1994a].

  A more elaborate discussion on pruning and references to relevant research are given in section 5.1.1.

The objective of all architecture selection algorithms is to find the smallest architecture that accurately fits the underlying function. In addition to improving generalization performance and avoiding overfitting (as discussed earlier), smaller networks have the following advantages. Once an optimized architecture has been found, the cost of forward calculations is significantly reduced, since the cost of computation grows almost linearly with the number of weights. From the generalization bounds overviewed earlier, the number of training patterns required to achieve a certain generalization performance is a function of the network architecture. Smaller networks therefore require less training patterns. Also, the knowledge embedded in smaller networks is more easily described by a set of simpler rules. Viktor, Engelbrecht and Cloete show that the number of rules extracted from smaller networks is less for pruned networks than that extracted from larger networks [Viktor *et al* 1995]. They also show that rules extracted from smaller networks contain only relevant clauses, and that the combinatorics of the rule

extraction algorithm is significantly reduced. Furthermore, for smaller networks the function of each hidden unit is more easily visualized. The complexity of decision boundary detection algorithms is also reduced.

With reference to the bias/variance decomposition of the MSE function [Geman *et al* 1992], smaller network architectures reduce the variance component of the MSE. NNs are generally plagued by high variance due to the limited training set sizes. This variance is reduced by introducing bias through minimization of the network architecture. Smaller networks are biased because the hypothesis space is reduced, thus limiting the available functions that can fit the data. The effects of architecture selection on the bias/variance trade-off have been studied by Gedeon, Wong and Harris [Gedeon *et al* 1995].

This thesis concentrates on pruning as method of architecture selection, specifically pruning based on sensitivity analysis.

The rest of this chapter is organized as follows. An overview of pruning approaches is presented in section 5.1.1. Section 5.2 presents a general pruning algorithm and discusses important pruning design issues, such as when to start pruning, and how to decide what to prune. Section 5.3 develops a new sensitivity analysis pruning algorithm that uses output sensitivity information to remove irrelevant parameters. This section also presents a new statistical hypothesis test, based on variance analysis of sensitivity information, to determine which parameters to prune. Results of the pruning algorithm are presented in section 5.4.

For the purposes of this chapter, a three layer NN architecture (input layer, one hidden layer, output layer) is assumed. As illustrated in appendix E, section E.1.1, the algorithm can easily be extended to include more hidden layers. No specific objective function and optimization method are assumed, since output sensitivity analysis pruning is generally applicable to any objective function and optimization method. It is assumed that activation functions are at least once differentiable. For this chapter, sigmoidal activation functions are used. Extension to other differentiable activation functions is straightforward.

This chapter follows the main theme of the thesis by exploring the use of sensitivity analysis for NN pruning. The pruning algorithm also addresses the sub objectives of improving generalization performance and decreasing complexity.

5.1.1 Related Work

This section presents a brief overview of existing pruning techniques. The objective is to present a flavor of the vast number of different approaches to determine the relevance of parameters. For more detailed discussions, the reader is referred to the given references.

The importance of optimized NN architectures have been discussed in previous sections. Given an arbitrary problem, it is not at all obvious what architecture is optimal. Initially, experts relied on their intuition to guess the correct architecture. Needless to say that such intuitive guesses were frequently wrong, requiring other architectures to be tried until an acceptable one was found. This trial and error process is not effective, and may be very time consuming. Also, guessing the size of a network might produce an architecture that fits the data, but it may contain irrelevant units and weights. Instead, a brute-force pruning approach can be followed, where each weight is set to zero and the change in error evaluated. If the resulting error increase is too high, the weight's value is restored, otherwise the weight is removed. While this approach will produce a minimal network size, it is prohibitively expensive for very large initial networks.

Several research efforts have targeted this architecture selection problem. The first results in the quest to find a solution to the architecture optimization problem were the derivation of theoretical bounds on the number of hidden units to solve a particular problem [Baum 1988, Cosnard *et al* 1992, Hayashi 1993, Kamruzzaman *et al* 1992, Ludik 1995a, Ludik *et al* 1996, Sakurai *et al* 1992a, Sakurai 1992b, Sartori *et al* 1991]. However, these results are based on unrealistic assumptions about the network and the problem to be solved. Also, they usually apply to classification problems only. While these bounds do improve our understanding of the relationship between architecture and training set characteristics, they do not predict the correct number of hidden units for a general class of problems.

Recent research concentrated on the development of more efficient pruning techniques to solve the architecture selection problem. Several different approaches to pruning have been developed. This chapter groups these approaches in the following general classes: intuitive methods, evolutionary methods, information matrix methods, hypothesis testing methods and sensitivity analysis methods.

- **Intuitive pruning techniques**: Simple intuitive methods based on weight values and unit activation values have been proposed by Hagiwara [Hagiwara 1993]. The *goodness factor* G_i^l of unit i in layer l, $G_i^l = \sum_p \sum_j (w_{ji}^l o_i^l)^2$, where the first sum is over all patterns, and o_i^l is the output of the unit, assumes that an important unit is one which excites frequently and has large weights to other units. The *consuming energy*, $E_i^l = \sum_p \sum_j w_{ji}^l o_j^{l+1} o_j^l$, additionally assumes that unit i excites the units in the next layer. Both methods suffer from the flaw that when an unit's output is more frequently 0 than 1, that unit might be elected as being unimportant, while this is not necessarily the case. Magnitude based pruning assumes that small weights are irrelevant [Hagiwara 1993, Lim *et al* 1994]. However, small weights may be of importance, especially compared to very large weights which cause saturation in hidden and output units. Also, large weights (in terms of their absolute value) may cancel each other out.

- **Evolutionary pruning techniques**: The use of genetic algorithms (GA) to prune NNs provides a biological plausible approach to pruning [Kuscu *et al* 1994, Reed 1993, Whitley *et al* 1990, White *et al* 1993]. Using GA terminology, the population consists of several pruned versions of the original network, each needed to be trained. Differently pruned networks are created by application of mutation, reproduction and cross-over operators. These pruned networks "compete" for survival, being awarded for using fewer parameters and for improving generalization. GA NN pruning is thus a time consuming process.

- **Information matrix pruning techniques**: Several researchers have utilized approximations to the Fisher information matrix to determine the optimal number of hidden units and weights. Based on the assumption that outputs are linearly activated, and that least squares estimators satisfy asymptotic normality, Cottrell *et al* computes the relevance of a weight as a function of the information matrix, approximated by [Cottrell *et al* 1994]

$$\mathcal{I} = \frac{1}{P} \sum_{p=1}^{P} \frac{\partial \mathcal{F}_{NN}}{\partial w} (\frac{\partial \mathcal{F}_{NN}}{\partial w})^T \tag{5.4}$$

  Weights with a low relevance are removed.

  Hayashi [Hayashi 1993], Tamura, Tateishi, Matumoto and Akita [Tamura *et al* 1993],

Xue, Hu and Tompkins [Xue *et al* 1990] and Fletcher, Katkovnik, Steffens and Engelbrecht [Fletcher *et al* 1998] use Singular Value Decomposition (SVD) to analyze the hidden unit activation covariance matrix to determine the optimal number of hidden units. Based on the assumption that outputs are linearly activated, the *rank* of the covariance matrix is the optimal number of hidden units (also see [Fujita 1992]). SVD of this information matrix results in an eigenvalue and eigenvector decomposition where low eigenvalues correspond to irrelevant hidden units. The rank is the number of non-zero eigenvalues. Fletcher, Katkovnik, Steffens and Engelbrecht use the SVD of the conditional Fisher information matrix, as given in equation (5.4), together with likelihood-ratio tests to determine irrelevant hidden units[Fletcher *et al* 1998]. In this case the conditional Fisher information matrix is restricted to weights between the hidden and output layer only, whereas previous techniques are based on all network weights. Each iteration of the pruning algorithm identifies exactly which hidden units to prune.

Principal Component Analysis (PCA) pruning techniques have been developed that use the SVD of the Fisher information matrix to find the principle components (relevant parameters) [Levin *et al* 1994, Kamimura 1993, Schittenkopf *et al* 1997, Takahashi 1993]. These principle components are linear transformations of the original parameters, computed from the eigenvectors obtained from a SVD of the information matrix. The result of PCA is the orthogonal vectors on which variance in the data is maximally projected. Non-principle components/parameters (parameters that do not account for data variance) are pruned. Pruning using PCA is thus achieved through projection of the original w-dimensional space onto a $w^{'}$-dimensional linear subspace ($w^{'} < w$) spanned by the eigenvectors of the data's correlation or covariance matrix corresponding to the largest eigenvalues.

- **Hypothesis testing techniques**: Formal statistical hypothesis tests can be used to test the statistical significance of a subset of weights, or a subset of hidden units. Steppe, Bauer and Rogers [Steppe *et al* 1996] and Fletcher, Katkovnik, Steffens and Engelbrecht [Fletcher *et al* 1998] use the *likelihood-ratio test statistic* to test the null hypothesis that a subset of weights is zero. Weights associated with a hidden unit are tested to see if they are statistically different from zero. If these weights are not statistically different from zero, the corresponding hidden unit is pruned.

Belue and Bauer propose a method that injects a noisy input parameter into the NN model, and then use statistical tests to decide if the significances of the original NN parameters are higher than that of the injected noisy parameter [Belue *et al* 1995]. Parameters with lower significances than the noisy parameter are pruned.

Similarly, Prechelt [Prechelt 1995] and Finnoff *et al* [Finnoff *et al* 1993] test the assumption that a weight becomes zero during the training process. This approach is based on the observation that the distribution of weight values is roughly normal. Weights located in the left tail of this distribution are removed.

- **Sensitivity analysis pruning techniques**: The two main approaches to NN sensitivity analysis have been introduced in chapter 2. Both sensitivity analysis with regard to the objective function and sensitivity analysis with regard to the NN output function resulted in the development of a number of pruning techniques. Possibly the most popular of these are OBD [Gorodkin *et al* 1993b, Le Cun 1990, Leung *et al* 1996, Reed 1993] and its variants, OBS [Hassibi *et al* 1993, Hassibi *et al* 1994] and Optimal Cell Damage (OCD) [Cibas *et al* 1994a, Cibas *et al* 1994b, Cibas *et al* 1996]. A parameter saliency measure is computed for each parameter, indicating the influence small perturbations to the parameter have on the approximation error. Parameters with a low saliency are removed. These methods are based on the assumptions discussed in section 2.4.1, and are time consuming due to the calculation of the Hessian matrix. Buntine and Weigend [Buntine *et al* 1994] and Bishop [Bishop 1992] derived methods to simplify the calculation of the Hessian matrix in a bid to reduce the complexity of these pruning techniques. In OBD, OBS and OCD, sensitivity analysis is performed with regard to the training error. Pedersen, Hanson and Larsen [Pedersen *et al* 1996] and Burrascano [Burrascano 1993] develop pruning techniques based on sensitivity analysis with regard to the generalization error. Other objective function sensitivity analysis pruning techniques have been developed by Mozer and Smolensky [Mozer *et al* 1989], Karnin [Karnin 1990], and Moody and Utans [Moody *et al* 1995].

  NN output sensitivity analysis pruning techniques have been developed that are less complex than objective function sensitivity analysis, and that do not rely on

simplifying assumptions. Zurada, Malinowski and Cloete introduced output sensitivity analysis pruning of input units [Zurada *et al* 1994], further investigated by Cloete and Engelbrecht [Cloete *et al* 1994c] and Engelbrecht, Cloete and Zurada [Engelbrecht *et al* 1995b]. Engelbrecht and Cloete extended this approach to also prune irrelevant hidden units [Engelbrecht *et al* 1996, Engelbrecht *et al* 1999e]. This work, and extensions thereof to include statistical pruning heuristics, is presented in this chapter.

A similar approach to NN output sensitivity analysis was followed by Dorizzi *et al* [Dorizzi *et al* 1996] and Czernichow [Czernichow 1996] to prune parameters of a Radial Basis Function (RBF) NN. Takenaga *et al* uses output sensitivity analysis to reduce the feature space for a pattern recognition problem [Takenaga *et al* 1991].

5.2 Neural Network Pruning

A too small NN architecture does not have enough free parameters to satisfactorily approximate the true function - the network is said to underfit the data. A too large architecture has too many free parameters which cause the training data to be overfitted, or memorized. It is therefore important to optimize the NN architecture for a specific problem, by finding a balance between the underfitting and overfitting effects. Pruning is one approach to solve this problem. Training starts on a very large architecture, which is successively reduced through elimination of irrelevant network parameters. Section 5.1.1 reviewed several approaches to network pruning. This section concentrates on pruning design issues, including pruning heuristics, stopping criteria, how much to prune and how to repair the network after pruning. These issues are discussed in section 5.2.2. First, a general pruning algorithm is presented in section 5.2.1. This thesis concentrates on NN sensitivity analysis applications, and therefore on sensitivity analysis pruning. Section 5.2.3 presents a short comparison of the two main sensitivity analysis pruning approaches.

5.2.1 General Pruning Algorithm

Neural network architecture reduction through pruning can be summarized by the following, very general, algorithm:

1. Initialize the NN architecture and learning parameters
2. Repeat
   (a) Train the NN until a pruning indicator is triggered
   (b) Compute the approximate importance of each parameter considered for pruning
   (c) Apply a pruning heuristic
   (d) Repair weights of the pruned network

   until the architecture is satisfactory
3. Train the final pruned NN architecture

The algorithm above refers to concepts which need further explanation at this point. These concepts are discussed in more detail in section 5.2.2. Step 2.(a) refers to a pruning indicator. The pruning indicator addresses the important question of when to start pruning, i.e. when have the network learned sufficiently to ensure the correct removal of only irrelevant parameters? Also key to the efficiency of a pruning method is the calculation of the approximate importance of parameters (i.e. input and hidden units, and weights). The estimated importance of parameters is used to decide whether a parameter should be removed or not. It is therefore essential that the method used to calculate parameter importance results in accurate estimations of the true importance of parameters. A pruning heuristic, or pruning decision, is applied to the estimated importance values to decide when a parameter is viewed as being irrelevant or not. All irrelevant parameters are then removed.

After pruning, the network's state has changed, and the network is no longer in the same position in the search space. The network needs to be re-trained on the smaller architecture. For this purpose a weights repair algorithm is applied to either reset weights to new random

values, or to redistribute the values of the removed weights over the remaining weights. After application of the weight repair algorithm, the pruned network is trained on the same data, and the pruning process is restarted. When no more parameters can be pruned, the last step is to train the final pruned network.

5.2.2 Pruning Design Issues

The design of a pruning algorithm should be attempted cautiously. Careful consideration must be given to design issues to ensure that pruning does not unacceptably deteriorate generalization performance, and does not incorrectly remove parameters. Such design issues are discussed in this section.

One of the questions that needs to be addressed is when to start pruning. If pruning is invoked prematurely, important parameters may be eliminated from the NN model. On the other hand, if pruning is applied too late, overfitting may cause potentially irrelevant parameters to learn noise, losing their irrelevance. Most pruning algorithms require the network to be well trained. Pruning approaches based on objective function sensitivity analysis, for example OBD, OBS and OCD [Le Cun 1990, Hassibi *et al* 1993, Cibas *et al* 1996], is very strict on the requirement that the training process must reach a minimum of the objective function before pruning starts. This requirement is due to the extremal approximation assumption as discussed in section 2.4.1. The correctness of the output sensitivity analysis model, developed in this thesis, does not rely on this assumption. However, to ensure efficiency, the learned derivatives should accurately approximate the true derivatives of the underlying function. For this purpose the network should be in a local minimum. A validation set can be used to indicate when a local minimum is reached, and to start pruning before the network overfits. As soon as the validation error deteriorates, training stops and the pruning algorithm is executed. Non-convergent pruning algorithms have been developed, which can be applied without requiring that the network should have converged to a local minimum. For example, Finnoff, Hergert and Zimmermann propose a statistical weight pruning algorithm that can be used to prune weights at any time during training [Finnoff *et al* 1993]. Prechelt developed a similar approach to weight pruning [Prechelt 1995].

The method used to calculate the approximate importance of parameters is very important to the success of any pruning algorithm. Initial approaches to pruning use heuristics to decide on the importance of parameters. For example, magnitude based pruning of weights assumes that small weights are unimportant. This is not always true. In cases where units saturate, the large weights that cause the saturation lead to constant outputs of the units and are therefore rather unimportant. More accurate approximations to parameter importance are based on the statistical relationships among parameters. For example, information approaches base parameter importance on parameter covariance matrices which statistically convey important information on the principle components of a model [Cottrell *et al* 1994, Fletcher *et al* 1998, Levin *et al* 1994, Xue *et al* 1990]. Sensitivity analysis approaches use the NN's learned knowledge to accurately approximate parameter importance. OBD, OBS and OCD define a saliency measure based on the sensitivity of the network error to parameter perturbations [Le Cun 1990, Hassibi *et al* 1993, Cibas *et al* 1996], while output sensitivity analysis pruning computes parameter significance as the sensitivity of the NN output function to parameter perturbations [Cloete *et al* 1994c, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Zurada *et al* 1994, Zurada *et al* 1997]. Such theoretically justified approaches should be preferred as measures of parameter importance.

A pruning algorithm applies a heuristic or rule to decide whether a parameter should be eliminated or not. Based on the parameter importance values, the heuristic first implements a mechanism to suggest candidate parameters for pruning, and then implements a test to determine if those parameters can in fact be pruned. Overly optimistic heuristics may cause too many parameters to be pruned, while pessimistic heuristics may prune too few - if any. The simplest heuristic is to select the least important parameter and to remove it, as is done in OBD [Le Cun 1990], skeletonization [Mozer *et al* 1989], and the pruning algorithm proposed by Steppe, Bauer and Rogers [Steppe *et al* 1996]. If network performance is degraded too much, the parameter is restored. Fletcher *et al* uses likelihood-ratio tests to find the number of hidden units that can be pruned during one iteration of the pruning algorithm, and to identify exactly which units to prune [Fletcher *et al* 1998].

Heuristics have been developed which orders the parameter importance values, and from this ordered list try to find a gap which indicates a division into relevant and irrelevant parameters.

All parameters below the gap are then removed in one step. OCD orders the input parameter saliencies ζ_i in decreasing order $\zeta_{i_1} \geq \zeta_{i_2} \geq \cdots \geq \zeta_{i_N}$, and finds the gap position n such that $\sum_{l=1}^{n} \zeta_{i_l} \geq q \sum_{l=1}^{N} \zeta_{i_l}$ [Cibas *et al* 1994b]. All parameters from position $n+1$ to N are then removed. The efficiency of this approach depends on the value of q, which is definitely problem dependent. The most appropriate value of q therefore needs to be calculated for each different problem using cross-validation, which can be time-consuming.

Zurada, Malinowski and Cloete define a gap measure as $g_{i_m} = \frac{\Phi_{i_m}}{\Phi_{i_{m+1}}}$, where input parameter significance values are ordered such that $\Phi_{i_{m+1}} \geq \Phi_{i_m}$ [Zurada *et al* 1994, Zurada *et al* 1997]. The largest gap $g_{max} = \max_{i_m}\{g_{i_m}\}$ is found, and $m_{cut} = m$ such that $g_{i_m} = g_{max}$. Then, if the pruning condition $Cg_{max} > \max_{i_m \neq i_{m_{cut}}}\{g_{i_m}\}$ is satisfied, all input parameters from position i_{m+1} are pruned. This heuristic also includes a problem dependent constant C which needs to be determined through cross-validation. It can be shown that this heuristic does not prune irrelevant input parameters for all situations, even when there is a very clear large gap between relevant and irrelevant parameters. If we have two inputs, with one totally insignificant, a large gap will occur between the significances of these two inputs. No second largest gap can be found to satisfy the pruning condition, and the redundant input will not be pruned. Even with three inputs, of which one is irrelevant, no pruning will occur. The largest gap, g_{max}, is then located between the second and third ordered input, while $\max_{i_m \neq i_{m_{cut}}}\{g_{i_m}\}$ will not satisfy the pruning condition and the redundant input is retained. Another scenario where this heuristic fails is if we have I inputs with a very large gap, g_{max}, between say Φ_{i_m} and $\Phi_{i_{m+1}}$, and a second largest gap g_{i_s} is found with $s > m+1$, but with $Cg_{max} \not> g_{i_s}$. Then, no inputs will be pruned, where the gap g_{max} may, however, be large enough to clearly suggest pruning everything from position $m+1$.

As an alternative, Engelbrecht, Cloete and Zurada [Engelbrecht *et al* 1995b] and Engelbrecht and Cloete [Engelbrecht *et al* 1996] use the rule of thumb that when $\Phi_{i_{m+1}}$ is a factor C less than Φ_{i_m}, then all parameters from position $m+1$ are pruned. Again, the value of C is problem dependent.

Finnoff, Hergert and Zimmermann [Finnoff *et al* 1993] and Prechelt [Prechelt 1995] use hypothesis testing to find those weights that satisfy the null hypothesis that the expected value of weight changes is equal to zero [Finnoff *et al* 1993]. They define $\xi_h^{(p)} = w_h + \Delta w_h^{(p)}$, for

each pattern p, and a test variable $\mathcal{L}_h$ for each weight w_h

$$\mathcal{L}_h = \frac{|\sum_{p=1}^{P} \xi_h^{(p)}|}{\sqrt{\sum_{p=1}^{P} (\xi_h^{(p)} - \overline{\xi}_h)^2}} \tag{5.5}$$

where $\overline{\xi}_h$ denotes the average weight change over all patterns. Setting as null hypothesis that the expected value of $\xi_h^{(p)}$ is zero, amounts to testing the significance of the deviation of weight w_h from zero using the test variable $\mathcal{L}_h$.

The heuristics reviewed above does not test the change in performance of the network after pruning to determine whether the pruning does not unacceptably degrade performance. If performance deteriorates too much, the pruned parameters should be restored. The simplest way to test whether the pruned network can be accepted, is to retrain the network and to evaluate the generalization of the pruned network with that of the original network. If generalization performance of the pruned network is better than, or at least comparable to the unpruned network, the pruned network can be accepted. Statistical tests can be be used to test whether a pruned architecture can be accepted or not. Fletcher, Katkovnik, Steffens and Engelbrecht [Fletcher *et al* 1998] and Steppe, Bauers and Rogers [Steppe *et al* 1996] use the likelihood-ratio test statistic to accept or reject a pruned network. Cibas, Soulié, Gallinari and Raudys use an F-test of the null hypothesis that the parameters selected for pruning are in fact unimportant [Cibas *et al* 1996]. Alternatively, the Network Information Criterion (NIC), derived from Akaike's Information Criterion (AIC), can be used as model selection criterion [Murata *et al* 1991, Murata *et al* 1994a, Murata *et al* 1994b]. Given a pruned network $\mathcal{F}_{pruned}$ and the corresponding unpruned network $\mathcal{F}_{unpruned}$, if $NIC(\mathcal{F}_{pruned}) < NIC(\mathcal{F}_{unpruned})$, the generalization performance of $\mathcal{F}_{pruned}$ is expected to be better than that of $\mathcal{F}_{unpruned}$. The pruned network can therefore be accepted. In the same way Moody's Generalized Prediction Error (GPE) can be used to compute the expected generalization error of a pruned model [Moody 1992, Moody 1994a, Moody *et al* 1996]. These models require some knowledge about the distribution of the target values, since an estimation of the *noise covariance matrix* of the target values is needed. Such information is not always available. Also, these approaches are computationally expensive since they require the inversion of the Hessian of the objective function to compute the *influence matrix*.

When inputs, hidden units or weights are pruned from a NN, that NN no longer represents the

same function as before pruning. After pruning, the remaining weights need to be repaired to reflect the structural changes. After repair of the weights, the smaller NN is retrained - if necessary. The simplest method to do this is to simply assign new random values to the remaining weights. This will cause the network to move out of its current local minimum - with no guarantee that the network will converge during the retraining step. Alternatively, the pruned weight values can be redistributed over the remaining weights, keeping the network in its current minimum. Better local minima may exist for the pruned network, and retraining after weight redistribution may not necessarily mean that the network will jump out of the current minimum and move towards the better solution. Furthermore, weight redistribution as proposed by Hassibi and Stork is complicated and time consuming due to the calculation of the inverse Hessian matrix [Hassibi *et al* 1993]. Simpler weight redistribution methods have been developed [Fukumizu 1996, Jasić *et al* 1995, Pelillo *et al* 1993, Sietsma *et al* 1991].

Pruning should stop before too many parameters are removed to ensure that the underlying function can still be fitted accurately by the reduced model. The stopping decision should form part of the pruning heuristic. When parameters are identified for elimination, the performance of the reduced model should first be evaluated. If performance is unacceptably degraded, the parameters are not pruned and the pruning algorithm terminates. The performance of the reduced model is typically the error on a test set, i.e. the generalization.

Another issue is the order of pruning the different NN parameters. This chapter endorses the strategy proposed by Moody and Utans [Moody *et al* 1995]. The hidden layer is pruned first and then the input layer. When no more hidden or input units can be removed, weights are pruned. The basic idea of this pruning ordering is to start with those parameters that will cause the largest reduction in the number of free parameters. Pruning of one hidden unit causes the elimination of more weights than pruning one input unit.

5.2.3 Sensitivity Analysis Pruning

Neural network sensitivity analysis has been introduced in section 2.4, discussing the two main approaches to sensitivity analysis: with regard to the objective function and with regard to the NN output function - resulting in the development of two sensitivity analysis pruning

approaches. Equation (2.13) derives a general relationship between the two sensitivity analysis approaches, under the assumption of one output unit. This relationship shows that the two approaches are conceptually the same. Using equation (2.13), the saliency measure of OBD (from equation (2.8)) can be written as

$$S_\theta \approx \frac{1}{2}\frac{\partial^2 \mathcal{E}}{\partial \theta^2}\theta^2 = \frac{1}{2}(\frac{\partial o_k}{\partial \theta})^2\theta^2 \tag{5.6}$$

Therefore, under the assumptions listed in section 2.4.1, and under the assumption of one output networks, OBD and output sensitivity analysis are functionally the same, differing by a scaling factor of $\frac{1}{2}\theta^2$. The two pruning algorithms will remove the same parameters, since the parameter ordering according to significance values is the same. A similar relationship can be derived for more than one output unit as illustrated in equation (2.14).

Output sensitivity analysis has the advantages that it is not based on assumptions to simplify complexity, and that it is computationally less complex than objective function sensitivity analysis. The next section presents a pruning method based on output sensitivity analysis.

5.3 Pruning using Sensitivity Analysis wrt NN Output Function

This section presents the output sensitivity analysis pruning model for the removal of irrelevant parameters, including input units, hidden units and individual weights. The pruning algorithm computes the significance of parameters as a function of the first order derivatives of the NN outputs with respect to the parameters to be pruned. The parameter significance values measure the influence small perturbations to the parameters have on the outcome of the network. If a parameter has a low significance, it means that small changes to the value of that parameter causes insignificant changes to the network output.

The NN output sensitivity analysis algorithm is derived from a first order Taylor expansion of the output unit o_k around the parameter θ_i:

$$f_{o_k}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) \approx f_{o_k}(\vec{\theta}) + \Delta\theta_i \frac{\partial f_{o_k}}{\partial \theta_i} \tag{5.7}$$

From equation (5.7), the change in NN output due to the perturbation $\Delta\theta_i$ is approximated

by

$$f_{o_k}(\theta_1, \cdots, \theta_i + \Delta\theta_i, \cdots, \theta_I) - f_{o_k}(\vec{\theta}) \approx \Delta\theta_i \frac{\partial f_{o_k}}{\partial \theta_i} \tag{5.8}$$

The first order derivative $\frac{\partial f_{o_k}}{\partial \theta_i}$ can therefore be used as a measure of the change in the output due to perturbations of θ_i.

This section presents the NN output function sensitivity analysis algorithm for the pruning of input units, hidden units and individual weights. In addition to serving as a pruning tool, the model can be used to determine the importance of input parameters, that is, for causal inferencing. The work presented in this section has been published in [Cloete *et al* 1994c, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Engelbrecht *et al* 1999e], and applied to reduce the complexity of rule extraction algorithms [Viktor *et al* 1995, Viktor *et al* 1998a, Viktor 1998b].

The presentation of the output sensitivity analysis model is outlined as follows. Section 5.3.1 discusses the assumptions of the model. Two approaches to determine the relevance of parameters are developed. The first approach computes the significance of each parameter as a norm defined over the entire training set. These norms are discussed in section 5.3.2. The second approach computes a statistical test variable, based on variance analysis, for each parameter which expresses the variance in pattern sensitivities for that parameter, as presented in section 5.3.3. Pruning heuristics based on these two measures of relevance are explained in section 5.3.4. Complete pruning algorithms for the parameter significance and statistical test variable approach are given in section 5.3.5, and design issues are discussed.

5.3.1 Assumptions

NN output sensitivity analysis is not based on any assumptions to reduce the complexity of the model - which is the case for objective function sensitivity analysis. While the model does assume the network to be well trained, this assumption is not crucial to the theoretical validity of the model. However, output sensitivity analysis does require the derivatives of the underlying function to be well approximated. Hence the assumption that the network converged to a local minimum.

Output sensitivity analysis assumes activation functions that are at least once differentiable.

For this presentation sigmoidal activation functions are assumed. No assumptions are made with regard to the number of layers in the network. Although this section presents the model using a three layer architecture, it can easily be extended to any number of layers as explained in appendix E.1.1. The model also makes no assumptions with regard to the objective function or optimization method. Output sensitivity analysis is totally independent of the objective function and optimization method, and therefore independent of any regularization terms included in the objective function.

5.3.2 Parameter Significance

The first approach to determine the relevance of parameters for the output sensitivity analysis model rests on the concept of parameter significance. A parameter with low significance has little, or no influence on any of the outputs of the network and can therefore be removed.

Definition 5.1 Parameter Significance: *Define the significance of a parameter θ_i, which can be an input unit, hidden unit or weight, as the sensitivity of the NN output vector to small perturbations in that parameter. Let Φ_{θ_i} denote the significance of parameter θ_i. Then,*

$$\Phi_{\theta_i} \doteq ||\vec{S}_o|| \tag{5.9}$$

where $\vec{S}_o$ is the output sensitivity vector defined over the entire training set, and $||\bullet||$ is any suitable norm.

This study uses the maximum-norm to define parameter significance,

$$\Phi_{\theta_i} = ||\vec{S}_o||_\infty = \max_{k=1,\cdots,K}\{S_{o\theta,ki}\} \tag{5.10}$$

where $S_{o\theta}$ refers to the sensitivity matrix of the output vector $\vec{o}$ with respect to the parameter vector $\vec{\theta}$, and individual elements $S_{o\theta,ki}$ refers to the sensitivity of output o_k to perturbations in parameter θ_i over all patterns.

Different norms can be used to compute the output sensitivity matrix $S_{o\theta}$, e.g. the sum-norm, Euclidean-norm or the maximum-norm. Since each pattern results in a different sensitivity matrix, whatever norm is used, the norm has to be applied over the entire training set to

reflect the aggregated effect of all patterns. This study uses the Euclidean-norm, where for each element $S_{o\theta,ki}$ of the sensitivity matrix,

$$S_{o\theta,ki} = ||S_{o\theta}||_2 = \sqrt{\frac{\sum_{p=1}^{P}[S_{o\theta,ki}^{(p)}]^2}{P}} \tag{5.11}$$

where $S_{o\theta,ki}^{(p)}$ refers to the sensitivity of output o_k to changes in parameter θ_i for a single pattern p.

The formula to calculate $S_{o\theta,ki}^{(p)}$ depends on the type of parameter and activation function. For input units, from (E.6),

$$S_{oz,ki}^{(p)} = f_{o_k}^{(p)'} \sum_{j=1}^{J} w_{kj} f_{y_j}^{(p)'} v_{ji} \tag{5.12}$$

which can be written in matrix notation as

$$S_{oz}^{(p)} = O^{(p)'} W Y^{(p)'} V \tag{5.13}$$

where W $(K \times J)$ and V $(J \times I)$ are respectively the output and hidden layer weight matrices, and $O^{(p)'}$ $(K \times K)$ and $Y^{(p)'}$ $(J \times J)$ are defined as

$$O^{(p)'} \doteq diag(o_1^{(p)'}, \cdots, o_k^{(p)'}, \cdots, o_K^{(p)'}) \tag{5.14}$$

$$Y^{(p)'} \doteq diag(y_1^{(p)'}, \cdots, y_j^{(p)'}, \cdots, y_J^{(p)'}) \tag{5.15}$$

For hidden layer pruning, from (E.7),

$$S_{oy,kj}^{(p)} = f_{o_k}^{(p)'} w_{kj} \tag{5.16}$$

or in matrix notation

$$S_{oy}^{(p)} = O^{(p)'} W \tag{5.17}$$

Considering the pruning of weights, from (E.8) and (E.9),

$$S_{w,kj}^{(p)} = f_{o_k}^{(p)'} y_j^{(p)} \tag{5.18}$$

$$S_{v,ji}^{(p)} = f_{o_k}^{(p)'} w_{kj} f_{y_j}^{(p)'} z^{(p)} \tag{5.19}$$

or in matrix notation

$$S_{w}^{(p)} = O^{(p)'} Y^{(p)} \tag{5.20}$$

$$S_{v}^{(p)} = O^{(p)'} W Y^{(p)'} Z^{(p)} \tag{5.21}$$

where

$$Y^{(p)} \doteq diag(y_1^{(p)}, \cdots, y_j^{(p)}, \cdots, y_J^{(p)}) \tag{5.22}$$

$$Z^{(p)} \doteq diag(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) \tag{5.23}$$

Full derivations of these sensitivity equations are given in appendix E.

To allow for accurate results it is required that the range of the output vector and that of the parameter vector be the same. If this is not the case, scaling of the sensitivity matrix $S_{o\theta}$ is required as follows:

$$S_{o\theta,ki} = S_{o\theta,ki} \frac{(\max_{p=1,\cdots,P}\{\theta_i^{(p)}\} - \min_{p=1,\cdots,P}\{\theta_i^{(p)}\})}{(\max_{p=1,\cdots,P}\{o_k^{(p)}\} - \min_{p=1,\cdots,P}\{o_k^{(p)}\})} \tag{5.24}$$

5.3.3 Statistical Test Variable

A second approach to determine the relevance of parameters, based on parameter sensitivity information, is developed in this section (this work is published in [Engelbrecht *et al* 1999e]). A variance nullity measure is computed for each parameter, based on ideas borrowed from the non-convergent tests of Finnoff, Hergert and Zimmermann [Finnoff *et al* 1993]. The basic idea of the variance nullity measure is to test whether the variance in parameter sensitivity for the different patterns is significantly different from zero. If the variance in parameter sensitivities is not significantly different from zero, it indicates that the corresponding parameter has little or no effect on the output of the NN over all patterns considered. A hypothesis testing step, described in section 5.3.4, uses these variance nullity measures to statistically test if a parameter should be pruned, using the χ^2 distribution.

Definition 5.2 Parameter Variance Nullity: *Define the statistical nullity in the parameter sensitivity variance of a NN parameter θ_i over patterns $p = 1, \cdots, P$ as*

$$\Upsilon_{\theta_i} = \frac{(P-1)\sigma_{\theta_i}^2}{\sigma_0^2} \tag{5.25}$$

where $\sigma_{\theta_i}^2$ is the variance of the sensitivity of the network output to perturbations in parameter θ_i, and σ_0^2 is a value close to zero (the characteristics of this value are explained in section 5.3.4).

The *variance* in parameter sensitivity, $\sigma^2_{\theta_i}$, is computed as

$$\sigma^2_{\theta_i} = \frac{\sum_{p=1}^{P}(\aleph^{(p)}_{\theta_i} - \overline{\aleph}_{\theta_i})^2}{P-1} \tag{5.26}$$

where

$$\aleph^{(p)}_{\theta_i} = \frac{\sum_{k=1}^{K} S^{(p)}_{o\theta,ki}}{K} \tag{5.27}$$

and $\overline{\aleph}_{\theta_i}$ is the *average* parameter sensitivity

$$\overline{\aleph}_{\theta_i} = \frac{\sum_{p=1}^{P} \aleph^{(p)}_{\theta_i}}{P} \tag{5.28}$$

In equation (5.27), $\aleph^{(p)}_{\theta_i}$ is the *average* sensitivity of the NN output to perturbations in parameter θ_i for pattern p, and $S^{(p)}_{o\theta,ki}$ is the sensitivity of output o_k to perturbations in parameter θ_i for pattern p. If θ_i is an input parameter, equation (E.6) is used to calculate $S^{(p)}_{o\theta,ki}$, if it is a hidden unit equation (E.7) is used instead, equation (E.8) is used for hidden to output layer weights, and equation (E.9) is used for input to hidden layer weights.

The analysis of variance approach is followed here instead of an analysis of means as is done by Finnoff *et al* [Finnoff *et al* 1993]. In this study, an analysis of means is not appropriate since large negative and positive sensitivities may cancel each other, or produce a sum close to zero, indicating that the parameter is insignificant - which is not true. Using the variance nullity measure defined in definition 5.2, statistical theory prescribes the use of the $\chi^2(P-1)$ distribution to determine if a parameter can be pruned [Steyn *et al* 1995]. The next section illustrates how the variance nullity measure is used in hypothesis testing.

5.3.4 Pruning Heuristics

Two pruning heuristics are proposed. The first is a rule of thumb based on the parameter significance values calculated from equation (5.10), and the second a hypothesis testing procedure based on the statistical test variable calculated from equation (5.25).

The initial implemented pruning heuristic first orders the significance values in decreasing order [Cloete *et al* 1994c, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Viktor *et al* 1995]. It then finds a large enough gap between consecutive significance values, pruning all parameters following the gap - if such a gap could be found. A large enough gap is said to exist

whenever the significance $\Phi_{\theta_{m+1}}$ is a factor C less than Φ_{θ_m}. All the parameters from position $m+1$ are then removed. Unfortunately, the choice of C is problem dependent, and its optimal value should be found through cross-validation which immediately increases the time-complexity of the pruning algorithm.

While this rule of thumb appeared to work well for the simulations in [Cloete *et al* 1994c, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Viktor *et al* 1995], more robust and efficient problem independent heuristics are needed. For this purpose a new statistical pruning heuristic is proposed next.

The statistical pruning heuristic is based on proving or disproving the null hypothesis that the variance in parameter sensitivity is approximately zero. For this purpose, the null hypothesis that the variance in parameter sensitivity is approximately zero is tested, where the null hypothesis

$$\mathcal{H}_0 : \quad \sigma^2_{\theta_i} = \sigma^2_0 \tag{5.29}$$

is defined for each parameter θ_i. Unfortunately, from equation (5.25), $\sigma^2_0 \neq 0$, and we cannot hypothesize that the variance in parameter sensitivity over all patterns is exactly zero, i.e. $\sigma^2_{\theta_i} = 0$. Instead, a small value close to zero is chosen for σ^2_0, and the alternative hypothesis,

$$\mathcal{H}_1 : \quad \sigma^2_{\theta_i} < \sigma^2_0 \tag{5.30}$$

is tested. Using the fact that under the null hypothesis the variance nullity measure has a $\chi^2(P-1)$ distribution in the case of P patterns [Steyn *et al* 1995], the critical value Υ_c is obtained from χ^2 distribution tables,

$$\Upsilon_c = \chi^2_{v;1-\alpha} \tag{5.31}$$

where $v = P - 1$ is the number of degrees of freedom, and α is the level of significance. A significance level $\alpha = 0.01$, for example, means that we are satisfied with incorrectly rejecting the hypothesis once out of 100 times.

Using the critical value defined in equation (5.31), if $\Upsilon_{\theta_i} \leq \Upsilon_c$, the alternative hypothesis $\mathcal{H}_1$ is accepted and parameter θ_i is pruned.

The value of σ^2_0 is crucial to the success of this pruning heuristic. If σ^2_0 is too small, no parameters will be pruned. On the other hand, if σ^2_0 is too large, important parameters will

be pruned. The algorithm presented in section 5.3.5 therefore starts with a small value of σ_0^2, and increases the value if no parameters can be pruned under the smaller value of σ_0^2. After each pruning step, the performance of the pruned network is first tested to see if performance is not degraded too much. If the deterioration in performance is unacceptable, the original network is restored, and pruning stops.

To reduce computation time during the hypothesis testing phase, the variance nullity measures Υ_{θ_i} are arranged in increasing order. Hypothesis tests start on the smallest Υ_{θ_i} value and continue until no more parameters can be identified for pruning.

5.3.5 Pruning Algorithm

This section presents a complete output sensitivity analysis pruning algorithm for each of the pruning heuristics. Design issues are then briefly discussed.

The first algorithm is for the original gap method based on parameter significance values:

1. Initialize the NN architecture and learning parameters. Initialize the gap constant $C = 0.5$ (suggested value only).

2. Repeat

   (a) Train the NN until a pruning indicator is triggered

   (b) for each θ_i

      i. for each $p = 1, \cdots, P$
      calculate pattern sensitivity matrices $S_{o\theta,ki}$ using equations (5.12), (5.16), (5.18) or (5.19) depending on the parameter type

      ii. Calculate the Euclidean norm $S_{o\theta,ki} = ||S_{o\theta,ki}||_2$ using equation (5.11)

      iii. Calculate the parameter significance $\Phi_{\theta_i} = ||S_{o\theta}||_\infty$ using equation (5.10)

   (c) Apply the pruning heuristic:

      i. order Φ_{θ_i} in increasing order such that $\Phi_{\theta_{i_{m+1}}} \leq \Phi_{\theta_{i_m}}$,
      $m = 1, \cdots, I$

      ii. find a gap such that $\Phi_{\theta_{i_m}} \geq C\Phi_{\theta_{i_{m+1}}}$

      iii. prune parameters from position $m + 1$

   until no gap is found or the reduced network is not accepted due to an unacceptable deterioration in generalization performance

3. Train the final pruned NN architecture

In step 2.(b).(i) pruning is started with the hidden units. When no more hidden units can be pruned, the input layer is pruned. It is proposed that weights are pruned after pruning of the hidden and output layers.

Next, an algorithm for the statistical pruning heuristic is given:

1. Initialize the NN architecture and learning parameters
2. repeat
   (a) train the NN until a pruning indicator is triggered
   (b) let $\sigma_0^2 = 0.0001$
   (c) for each θ_i
      i. for each $p = 1, \cdots, P$, calculate $\aleph_{\theta_i}^{(p)}$ using equation (5.27)
      ii. calculate the average $\overline{\aleph}_{\theta_i}$ using equation (5.28)
      iii. calculate the variance in parameter sensitivity using $\sigma_{\theta_i}^2$ from equation (5.26)
      iv. calculate the test variable Υ_{θ_i} using equation (5.25)
   (d) apply the pruning heuristic
      i. arrange Υ_{θ_i} in increasing order
      ii. find Υ_c using equation (5.31)
      iii. for each θ_i, if $\Upsilon_{\theta_i} \leq \Upsilon_c$, then prune θ_i
      iv. if $\Upsilon_{\theta_i} > \Upsilon_c$ for all θ_i, let $\sigma_0^2 = \sigma_0^2 \times 10$

   until no θ_i is pruned, or the reduced network is not accepted due to an unacceptable deterioration in generalization performance
3. Train the final pruned NN architecture

The variance nullity algorithm starts pruning the hidden layer first, then the input layer. After pruning of the hidden and input layers, it is proposed that irrelevant weights are pruned. Calculation of the variance nullity measures can be done on the training, validation or test sets. For the experimental results reported in section 5.4.2, a data set of maximum 100 patterns (due to limitations of the available χ^2 tables) was created to calculate variance nullity measures. The patterns of this data set, referred to as the nullity set, were randomly selected from the original available data.

During training, the error on a validation set is monitored to detect when overfitting starts,

at which point pruning is initiated. When a network is pruned, the pruning model starts retraining of the reduced model on new initial random weights. The values of pruned weights are not redistributed over the remaining weights. The pruning stopping criterion is encapsulated in the pruning heuristic. If no more parameters can be identified for pruning, or if a reduced model is not accepted, the pruning process terminates. A pruned architecture is accepted if generalization performance is not unacceptably deteriorated.

5.4 Experimental Results

This section illustrates the application of the output sensitivity analysis pruning algorithms on classification and function approximation problems. Since successful results of the parameter significance pruning heuristic presented in section 5.4.1 have already been published in detail [Cloete *et al* 1994c, Engelbrecht *et al* 1995b, Engelbrecht *et al* 1996, Viktor *et al* 1995], some of these results are simply repeated in section 5.4.1. New results using the statistical variance nullity measure and hypothesis testing are presented in section 5.4.2 to illustrate the efficacy of this new pruning heuristic. Section 5.2.3 presented a mathematical model to prune input units, hidden units and individual weights. The results reported in this section are for input and hidden unit pruning only. Empirical studies into weight pruning are proposed for future work.

5.4.1 Parameter Significance: Pruning Results

Results of the application of the parameter significance pruning heuristic to three problems are reported in this section. The XOR problem and the time series TS2 defined in equation (4.42) are used to illustrate pruning of the input layer. Hidden unit pruning is illustrated on six parity problems ranging from 2 dimensions to 7 dimensions. Each experiment is described separately next. For these experiments it was assumed that when the significance of one parameter is half of that of the next parameter, a large enough gap is existed justify pruning of the lower significant parameters. The gap constant was therefore $C = 0.5$.

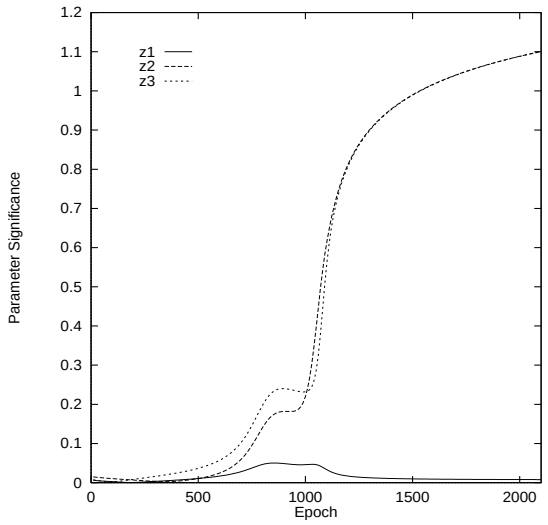
A 3-2-1 architecture was used for the XOR problem with one of the input units irrelevant

[Cloete *et al* 1994c]. Three simulations were executed, where each simulation differed in the position of the irrelevant input unit as the first, second or third input unit. The training set consisted of only 4 patterns, where the irrelevant input has small random values. These values were chosen as small deviations from 0, which is one of the actual input values of the XOR problem. The objective is to identify the irrelevant parameter based on the parameter significance values as calculated from equation (5.9). Training stopped when a 100% correct classification on the training set was achieved. Figure 5.1 illustrates the evolution of the input parameter significances during training. These parameter significance profiles show that for all three experiments a definite distinction was made between input units relevant for the classification problem and the one irrelevant input. For a 0.5 gap constant, the gap between the sensitivities of the relevant and irrelevant inputs was large enough to suggest that the irrelevant input unit bore little significance to the classification, and could therefore be pruned.

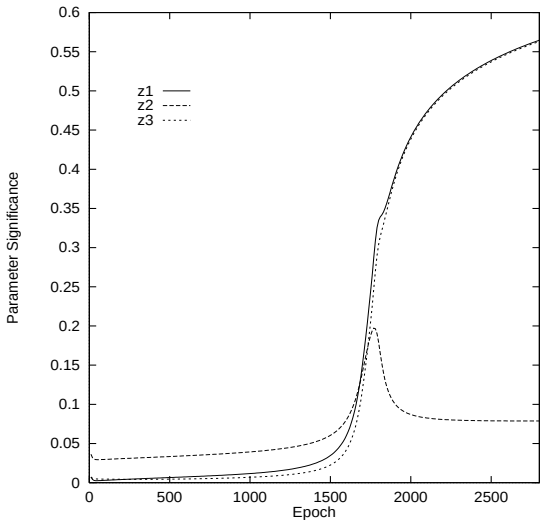
Next, the parameter significance pruning heuristic was applied to time series TS2 defined in equation (4.42) [Cloete *et al* 1994c]. The definition of this time series shows that only 3 input units are relevant, i.e. z_5, z_7 and z_{10}. The parameter significance profiles for the 10 input units are illustrated in figure 5.2. This figure shows a distinct grouping of the seven irrelevant input units with significance values in the range $[0.07, 0.23]$, and the relevant input units z_5, z_7 and z_{10} with higher significances in the range $[0.59, 1.12]$. A large enough gap exists between the two groups of input units which allows pruning of the seven irrelevant inputs.

The last set of experiments in this section is used to illustrate the applicability of the parameter significance pruning heuristic to the pruning of hidden units. The pruning heuristic was applied to six N-bit parity problems, with $N = 2, 3, \cdots, 7$. The reason for selecting parity problems to illustrate hidden unit pruning is to compare the results of the pruning algorithm with that published by Rumelhart and McClelland [Rumelhart *et al* 1986]. It is well-known from [Rumelhart *et al* 1986] that only N hidden units are necessary and sufficient to learn the N-bit parity problem. For all the simulations training stopped when all the examples in the training set were classified correctly. Training sets consisted of 224 patterns, created randomly. An N-10-1 architecture was used for each simulation.

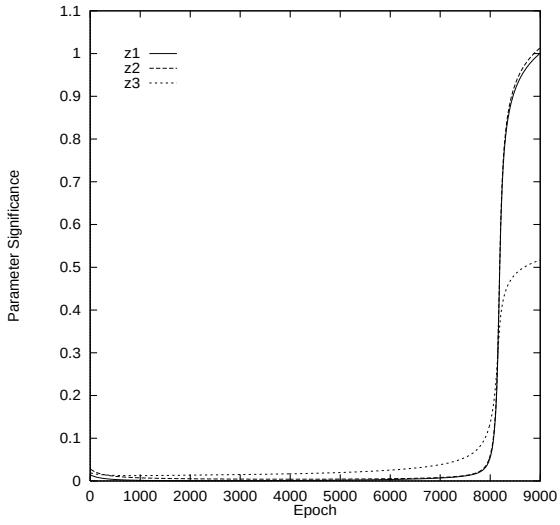
Figure 5.3 depicts the significance profiles for the hidden units for each simulation. For each



(a) Input z_1 irrelevant



(b) Input z_2 irrelevant



(c) Input z_3 irrelevant

Figure 5.1: Significance profiles for the XOR problem

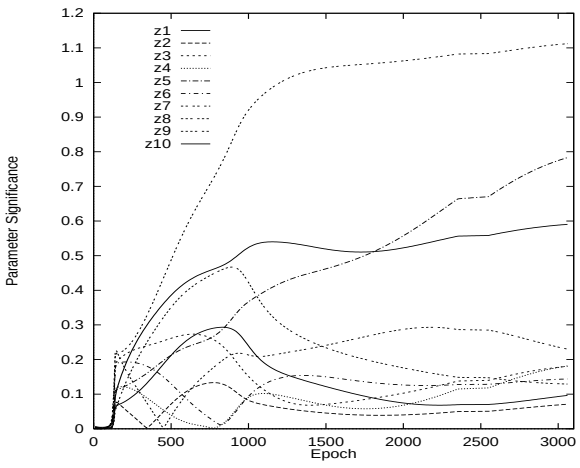


Figure 5.2: Significance profile for time series TS2

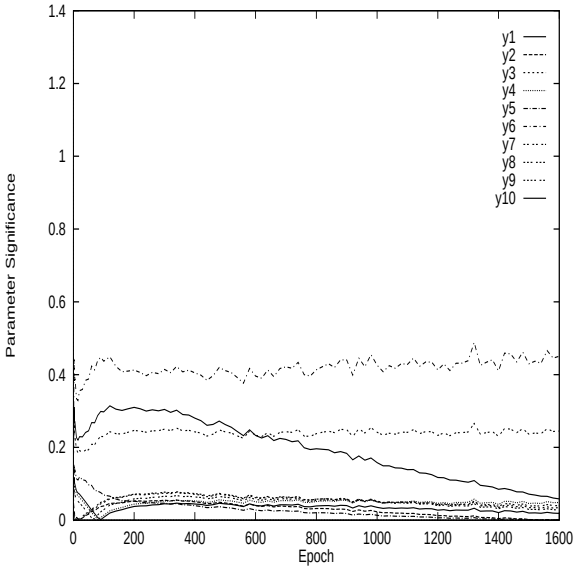
problem a large gap formed between relevant and irrelevant hidden units. At the time of convergence, a large enough gap existed between the group of relevant units and the group of irrelevant units to eliminate $10 - N$ hidden units in the case of N-parity. For example, 6 hidden units were pruned for the 4-bit parity problem (refer to figure 5.3(c)).

The results illustrated by figure 5.3 agree with the conclusion by Rumelhart and McClelland that an N-bit parity problem requires at least N units to be solved [Rumelhart *et al* 1986].

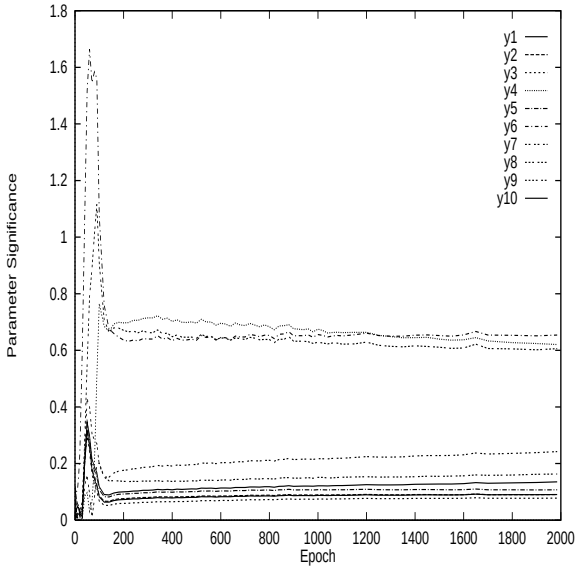
Although the experiments reported in this section showed parameter significance to be an accurate descriptor of the actual significance of parameters, the gap pruning heuristic is problematic. For these experiments a value of $C = 0.5$ for the gap constant worked quite well, but may be insufficient for other problems. Therefore requiring an optimal value to be computed for each new problem using methods such as cross-validation. For this reason, a statistically based method was developed to eliminate the problem of parameter pruning. Results of this statistical pruning heuristic are reported in the next section.

5.4.2 Parameter Variance Nullity: Pruning Results

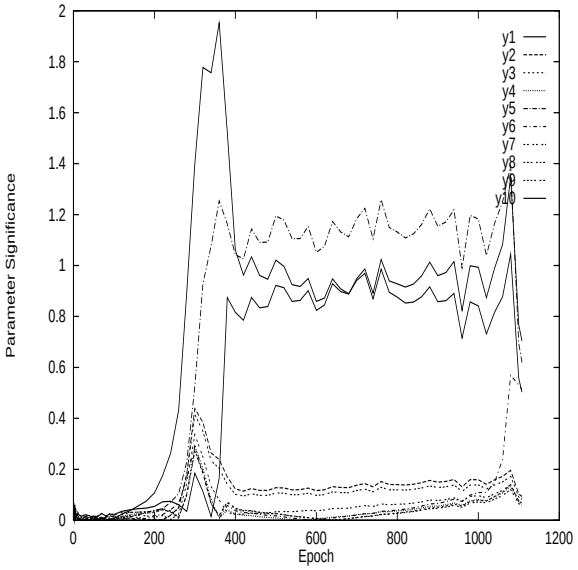
In this section the pruning heuristic based on the parameter variance nullity measure (defined in section 5.3.3) and hypothesis testing (described in section 5.3.4) is applied to problems for which the optimal architecture is known, either from the definition of the problem, or previous published results. The pruning algorithm is also applied to real-world problems for which the



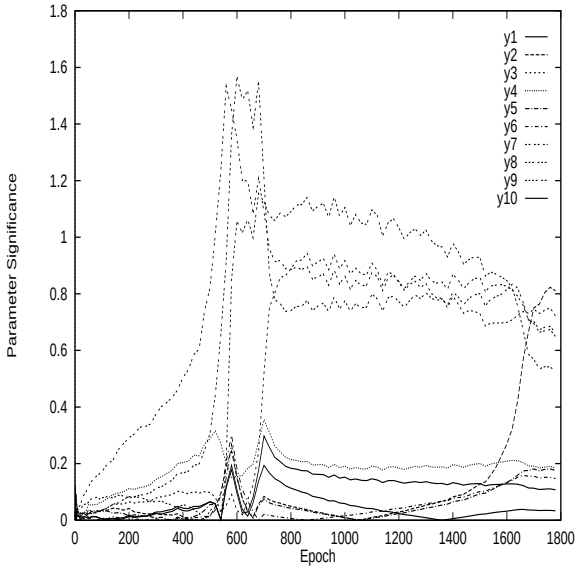
(a) 2-bit parity



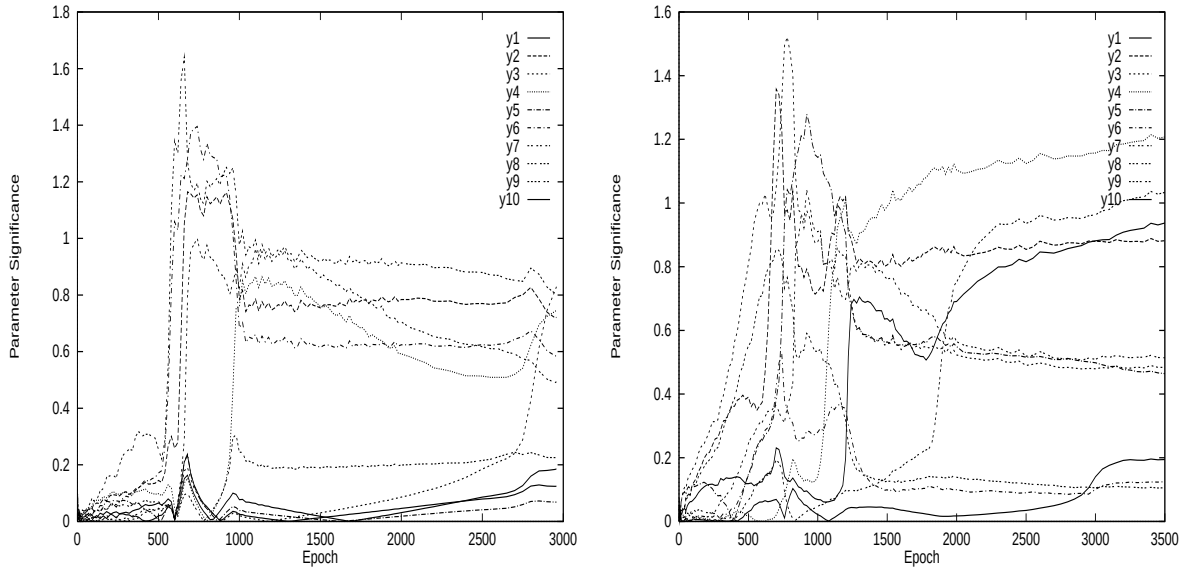
(b) 3-bit parity



(c) 4-bit parity



(d) 5-bit parity



(e) 6-bit parity

(f) 7-bit parity

Figure 5.3: Hidden unit significance profiles for N-bit parity problems

optimal architecture is unknown. In such cases, the generalization error is used as indication of whether a pruned network can be accepted or not. Table 5.1 summarizes the problems used to illustrate the correctness of this pruning algorithm.

The results presented in this section show that pruning is an iterative process: most of the problems investigated required more than one pruning step to optimize the architecture. For each problem a table is given to summarize the outcome of each pruning step. These tables contain for each pruning step the current architecture, the training and generalization (test) errors at the point where overfitting starts, the value of σ_0^2 and the critical value Υ_c for a significance of $\alpha = 0.01$, the number of patterns in the nullity set used for pruning purposes (i.e. the number of free parameters), and a reference to the figure that illustrates which parameters were pruned. The error on the test set serves as an indication of the performance of the pruned network compared to the oversized network.

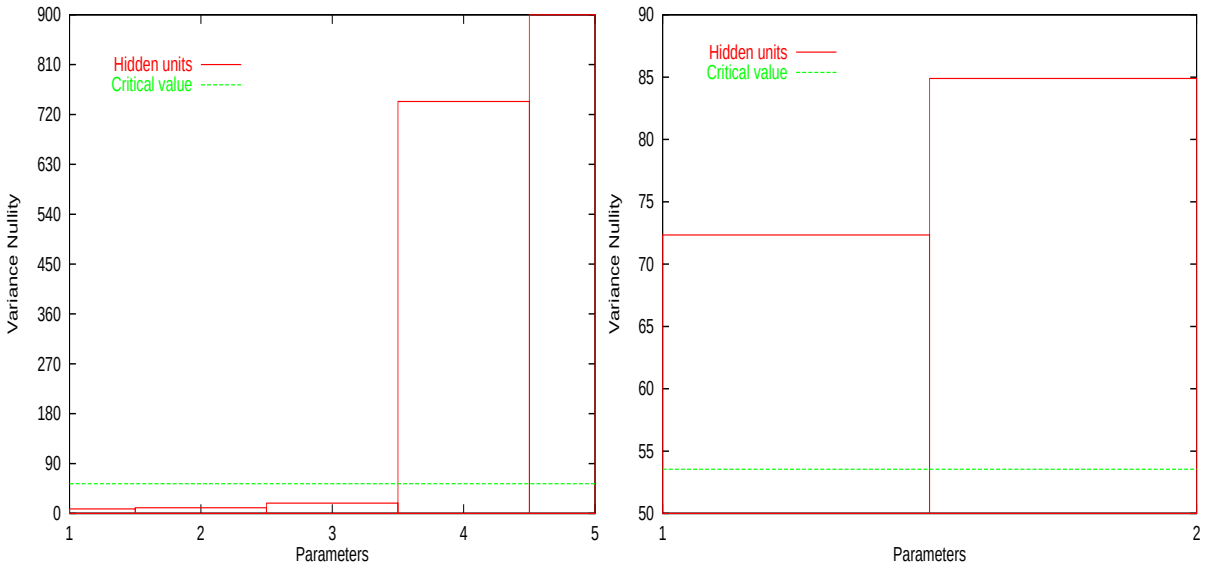
Problem	Defined in	To Prune	Initial NN	Optimized NN	Optimality Reference
Function F1	equation (4.38)	hidden units	1-5-1	1-2-1	figure 4.11(a)
Function F2	equation (4.39)	hidden units	1-10-1	1-5-1	figure 4.11(b)
Artificial Classification	equation (3.5)	hidden units	2-10-1	2-3-1	figure 3.1 equation (3.5)
Four Class Artificial Classification	equation (3.6)	hidden units input units	2-10-4	1-3-4	figure 3.8 equation (3.6)
iris	section 4.3.5	hidden units input units	4-10-3	2-2-3	UCI repository [UCI]
wine	section 4.3.5	hidden units input units	13-10-3	6-3-3	–
hepatitis	section 4.3.5	hidden units input units	19-25-1	4-4-1	–
diabetes	section 4.3.5	hidden units input units	8-40-1	6-8-1	–
breast cancer	section 4.3.5	hidden units input units	9-10-1	3-1-1	–

Table 5.1: Problems used to test the statistical pruning heuristic

For one dimensional function approximation problems experience has shown that the number of turning points in the target function, plus one, is sufficient to learn that function (if sigmoid activation functions are used). An activation function is fitted for each inflection point. The minimum number of hidden units for functions F1 and F2 are therefore respectively 2 and 5 (refer to figures 4.11(a) and 4.11(b)). The pruning results for function F1 are summarized in table 5.2, while table 5.3 contains the pruning results of function F2. These results illustrate that the statistical pruning heuristic correctly removed unnecessary hidden units. In figures 5.4 and 5.5, which illustrate the variance nullity of each hidden unit, only parameters with variance nullity smaller than the critical value (the dotted line) were removed. For function F1 only one pruning step was needed, while two pruning steps were needed for function F2. Note that the generalization error of the final architecture is very similar to that of the original unpruned network.

Pruning Step	Current Architecture	MSE Training Set	MSE Test Set	Patterns Used	σ_0^2	Υ_c	Pruned Parameters
1	1-5-1	0.040315	0.036176	80	0.1	53.54	3 hiddens figure 5.4(a)
2	1-2-1	0.040240	0.036191	80	1.0	53.54	pruning stops figure 5.4(b)

Table 5.2: Pruning results for function F1 using hypothesis testing



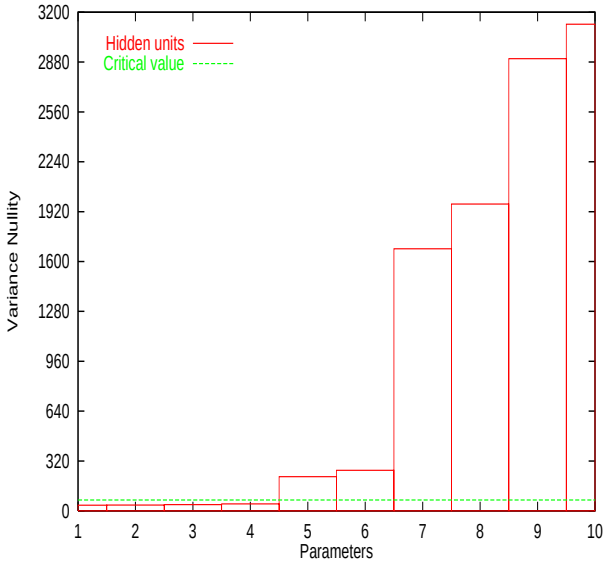
(a) Pruning Step 1

(b) Pruning Step 2

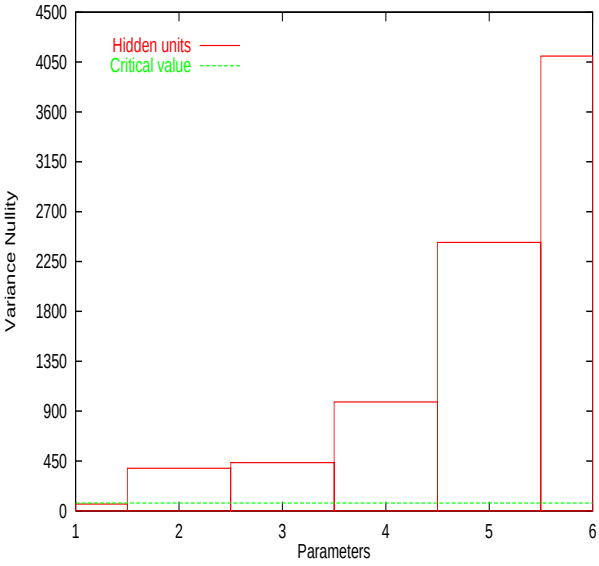
Figure 5.4: Hidden unit variance nullity for function F1

Pruning Step	Current Architecture	MSE Training Set	MSE Test Set	Patterns Used	σ_0^2	Υ_c	Pruned Parameters
1	1-10-1	0.021542	0.022668	100	0.1	70.065	4 hiddens figure 5.5(a)
2	1-6-1	0.025451	0.027519	100	0.1	70.065	1 hidden figure 5.5(b)
3	1-5-1	0.027835	0.029995	100	0.1	70.065	pruning stops figure 5.5(c)

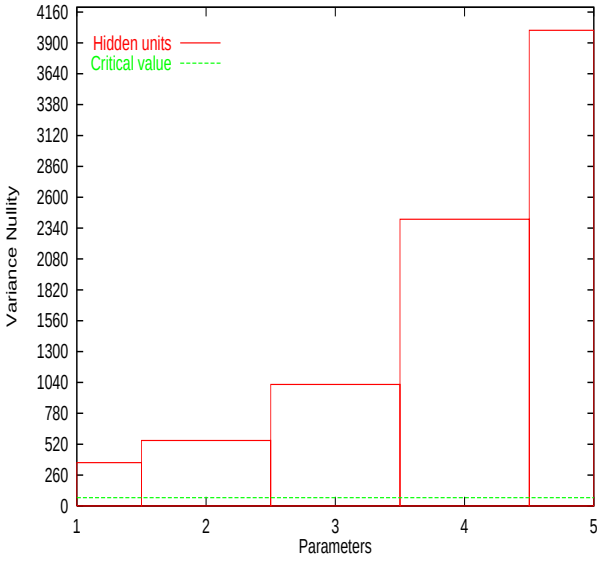
Table 5.3: Pruning results for function F2 using hypothesis testing



(a) Pruning Step 1



(b) Pruning Step 2



(c) Pruning Step 3

Figure 5.5: Hidden unit variance nullity for function F2

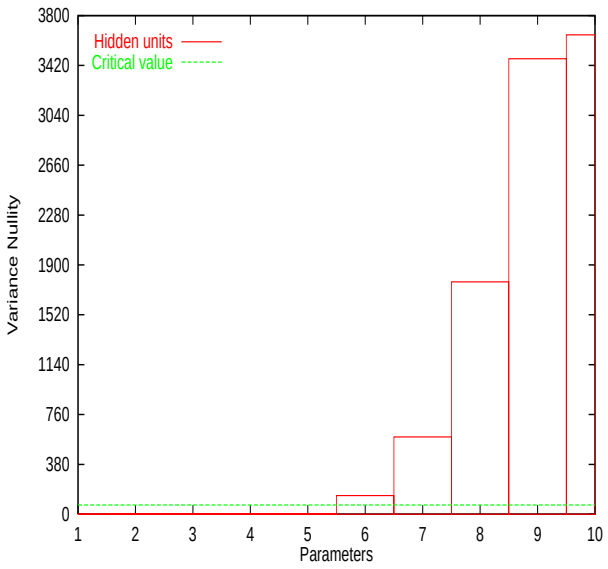
Pruning Step	Current Architecture	Accuracy Training Set	Accuracy Test Set	Patterns Used	σ_0^2	Υ_c	Pruned Parameters
1	2-10-1	95.5%	95%	100	0.01	70.065	5 hiddens figure 5.6(a)
2	2-5-1	96.2%	95%	100	0.01	70.065	2 hiddens figure 5.6(b)
3	2-3-1	96%	95%	100	1.0	70.065	pruning stops figure 5.6(c)

Table 5.4: Pruning results for artificial classification problem (3.5) using hypothesis testing

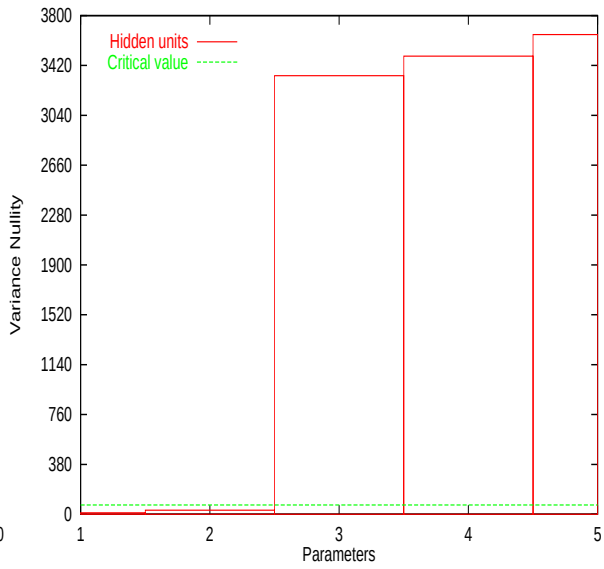
The next problems are classification tasks. Firstly, the artificial classification problem defined in equation (3.5) is used to further illustrate hidden unit pruning. From the definition of the problem in equation (3.5) and the illustration in figure 3.1, three hidden units are sufficient for this problem. The pruning results for this artificial problem are summarized in table 5.4. Two pruning steps were needed to find the optimal architecture. The last application of the pruning algorithm, with $\sigma_0^2 = 1.0$ suggested pruning of the remaining three hidden units. Since performance then greatly deteriorated, the pruning process was stopped and the current 2-3-1 architecture was accepted. The generalization performance of the accepted pruned network was the same as the original network.

The four-class artificial problem defined in equation (3.6) is used next to illustrate pruning of both hidden and input units. Equation (3.6) shows the second input unit to have no influence on the classification, and figure 3.8 shows that only 3 hidden units are required to separate the four classes. Only two pruning steps were needed, where the required number of hidden units was obtained after step one, and the irrelevant input unit was removed during step 2. Table 5.5 illustrates the successful pruning results for this problem. Note that the hidden layer was pruned first, and then the input layer when no more hidden units could be pruned.

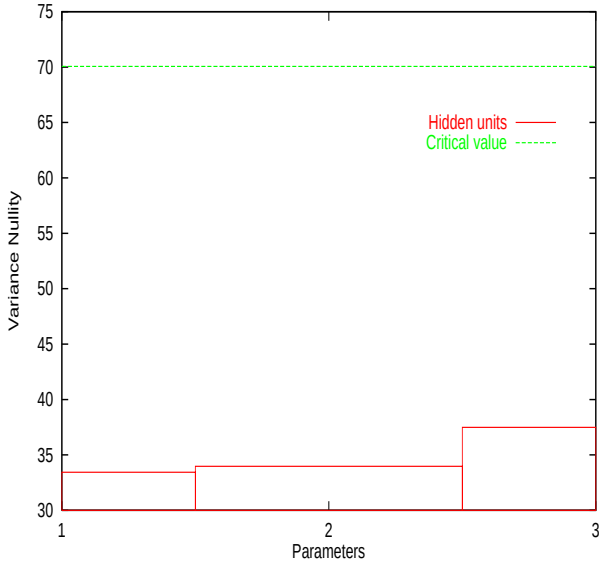
The last problem for which a detailed illustration of pruning is given is the iris problem used in section 4.3.5. From the statistical results given by the UCI repository, the first two input



(a) Pruning Step 1

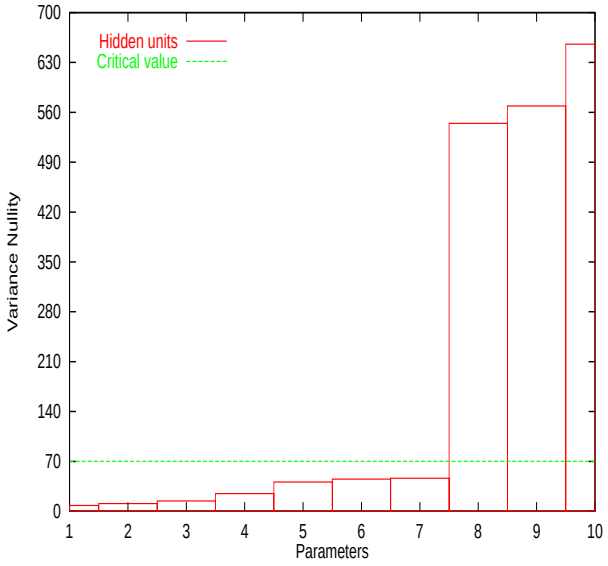


(b) Pruning Step 2

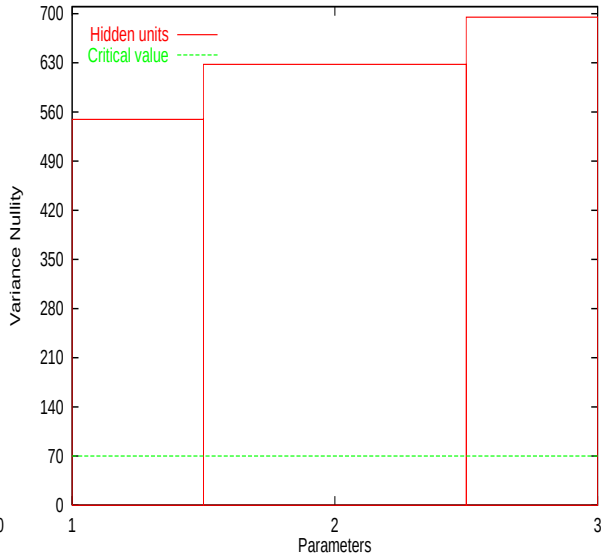


(c) Pruning Step 3

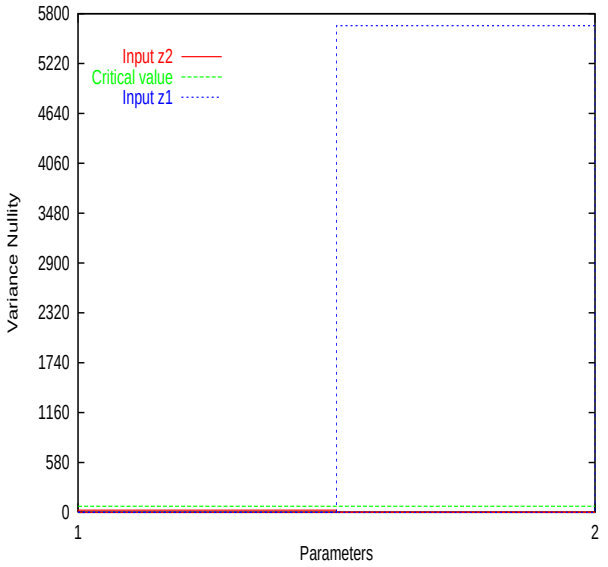
Figure 5.6: Hidden unit variance nullity for artificial classification problem (3.5)



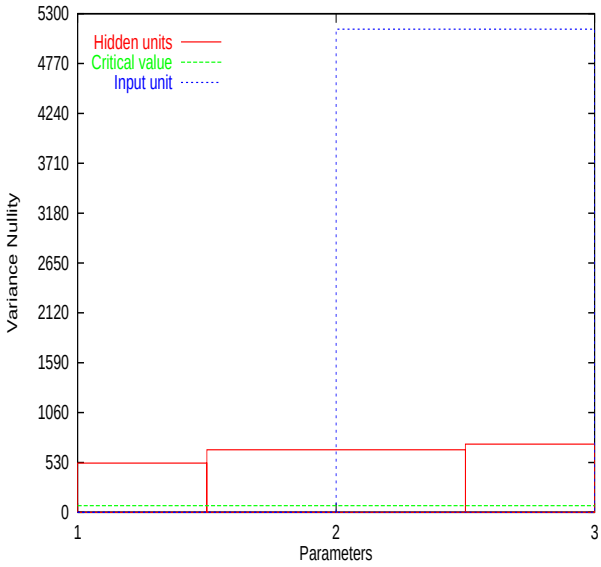
(a) Pruning Step 1



(b) Pruning Step 2



(c) Pruning Step 3



(d) Pruning Step 4

Figure 5.7: Parameter variance nullity for four-class artificial classification problem (3.6)

Pruning Step	Current Architecture	Accuracy Training Set	Accuracy Test Set	Patterns Used	σ_0^2	Υ_c	Pruned Parameters
1	2-10-4	95.3%	93%	100	0.01	70.065	7 hiddens figure 5.7(a)
2	2-3-4	95.5%	93%	100	0.01	70.065	hidden pruning stops figure 5.7(b) input unit z_2 figure 5.7(c)
3	1-3-4	94.7%	93%	100	1.0	70.065	pruning stops figure 5.7(d)

Table 5.5: Pruning results for four-classs artificial classification problem (3.6) using hypothesis testing

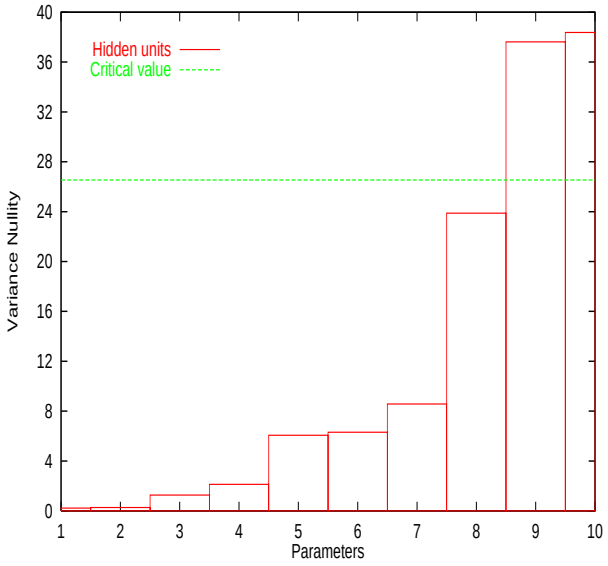
Pruning Step	Current Architecture	Accuracy Training Set	Accuracy Test Set	Patterns Used	σ_0^2	Υ_c	Pruned Parameters
1	4-10-3	97.1%	97.7%	45	0.1	26.54	8 hiddens figure 5.8(a)
2	4-2-3	96.2%	97.7%	45	0.1	26.54	hidden pruning stops figure 5.8(b) input units z_1, z_2 figure 5.8(c)
3	2-2-3	96.2%	97.7%	45	0.1	26.54	pruning stops figure 5.8(d)

Table 5.6: Pruning results for the iris classification problem using hypothesis testing

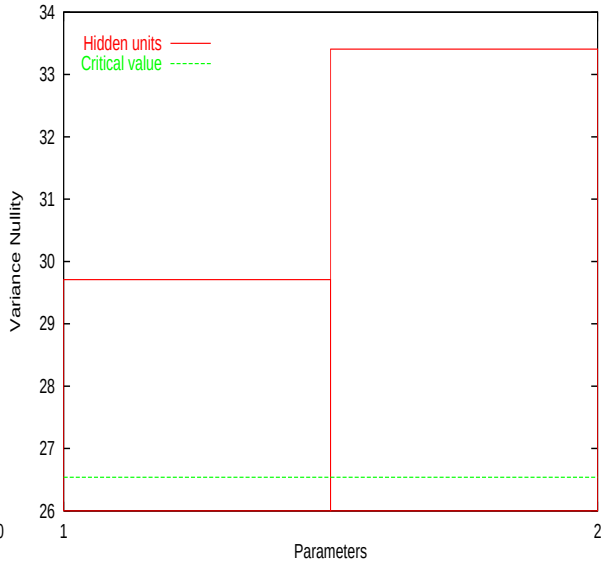
units, which correspond to the parameters *sepal length* and *sepal width*, can be removed since they have a low class correlation [1]. In addition to pruning of these two irrelevant input units, table 5.6 also illustrates that two hidden units were sufficient for the iris problem. The pruned network for this problem also retained the generalization performance of the original network.

Table 5.7 summarizes for other real-world classification problems the reduced architectures, obtained after application of the hypothesis test pruning heuristic. A detailed illustration of each step is not given, only the original and final architectures, and the percentage correctly

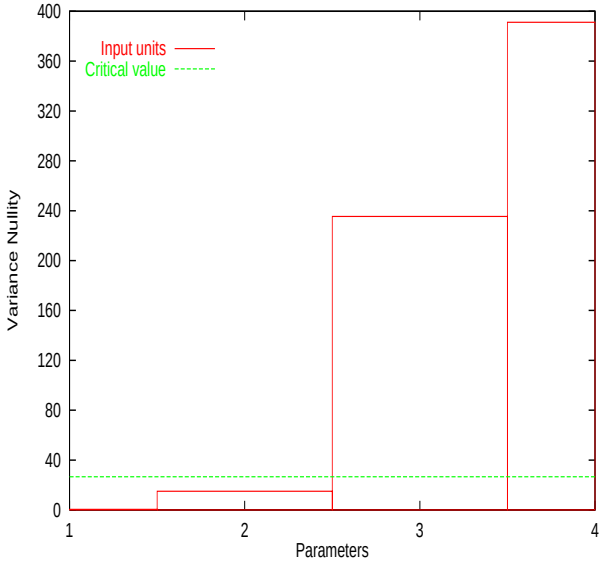
[1] Refer to the UCI machine learning repository at http://www.ics.uci.edu/ ~mlearn/MLRepository.html



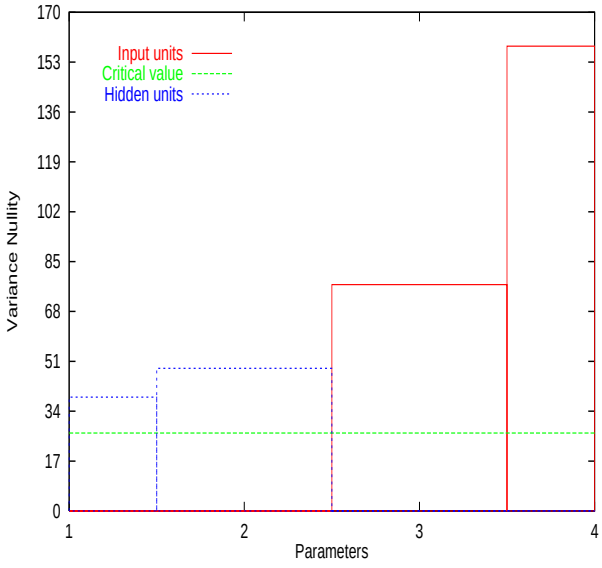
(a) Pruning Step 1



(b) Pruning Step 2



(c) Pruning Step 3



(d) Pruning Step 4

Figure 5.8: Parameter variance nullity for the iris problem

	Original Network			Reduced Network			Pruning
Problem	**NN**	$\mathcal{E}_T$	$\mathcal{E}_G$	**NN**	$\mathcal{E}_T$	$\mathcal{E}_G$	**Steps**
wine	13-10-1	100%	98%	6-3-3	96.1%	95.9%	4
hepatitis	19-25-1	94.3%	80%	4-4-2	78.9%	83.3%	7
diabetes	8-40-1	72%	68%	6-8-2	70.5%	69.1%	3
breast cancer	9-10-1	96.4%	98.1%	3-1-2	96.2%	97.8%	7

Table 5.7: Pruning results on real-world classification problems using hypothesis testing

classified patterns for these architectures for the training and test sets. The number of pruning steps to reach these optimal architectures is also given. The percentage correctly classified patterns on the training and test sets give an indication of the correctness of the pruning algorithm: the classification accuracies of the original and pruned networks were very similar, indicating that the pruned architectures can be accepted.

5.5 Conclusions

This chapter presented an approach to the pruning of multilayer feedforward NNs using output sensitivity analysis with respect to the parameters to be pruned. Two methods were proposed to quantify the relevance of network parameters, and a pruning heuristic was developed for each measure of parameter relevance:

- The parameter significance method computes the relevance of a parameter as the squared average of the sensitivity of the output to changes in that parameter over all patterns in the training set. The pruning heuristic orders the parameter significances and finds a large enough gap between consecutive parameters to prune lower significant parameters. The success of this approach depends on a gap constant which is problem dependent.

- The statistical variance nullity method quantifies the importance of a parameter as the variance in the parameter sensitivity over a set of patterns. The pruning heuristic is based on formal hypothesis tests, using a strict significance level of $\alpha = 0.01$.

Results for both approaches illustrated the success of the pruning algorithms in removing irrelevant input and hidden units. Correctness of the pruning results was illustrated by using problems for which the minimal number of input and hidden units were known, and by evaluating the accuracy of the pruned architectures with that of the original architectures.

It is proposed that future research beyond this thesis includes

- empirical studies of the presented pruning methods to prune weights;
- the inclusion of an F-test statistic to statistically test if the performance of the pruned architecture is significantly less than that of the original architecture, instead of using the rule of thumb "stop pruning when generalization deteriorates too much";
- empirical studies to investigate application of the pruning methods to different multi-layer NN types, using activation functions other than sigmoid functions; and
- a comparative study of different pruning techniques, including pruning based on NN output sensitivity analysis.

The pruning algorithms presented in this section addressed the thesis sub objectives by reducing the complexity of the NN model, thereby reducing the number of free parameters. Consequently, the effects of overfitting is reduced. The pruning algorithm also reduces the complexity of NN rule extraction algorithms, and enables the extraction of more accurate rules with no redundant input parameters [Viktor *et al* 1995, Viktor 1998b].

Chapter 6

Conclusion

"...on the other hand,
we cannot ignore efficiency"
- J Bentley

This thesis proposed NN output sensitivity analysis as an approach to learn more about the inner working of feedforward NNs, and the data being modeled. New sensitivity analysis techniques were developed to probe the knowledge embedded in the weights, and to use this knowledge within specialized algorithms to improve generalization performance, to reduce learning and model complexity, and to improve convergence characteristics.

The main contributions of this study were

- a comparison of objective function and output sensitivity analysis;
- a sensitivity analysis approach to the visualization of decision boundaries;
- a selective learning algorithm for classification problems;
- an incremental learning algorithm for function approximation problems;
- a pruning algorithm which includes a pruning heuristic based on variance analysis and hypothesis testing; and
- a self-scaling learning algorithm which dynamically adapts sigmoid activation functions.

NN output function sensitivity analysis was presented as an approach to compute the sensitivity of output units to small perturbations in network parameters, which include input units, hidden units and weights. A general mathematical model was developed for output sensitivity analysis from a first order Taylor expansion of the NN output with respect to NN parameters. This model was compared with objective function sensitivity analysis. It was shown that these two approaches to sensitivity analysis are conceptually the same, and will result in the same order of parameter significances. However, output sensitivity analysis is less complex than objective function sensitivity analysis, which requires the calculation of the Hessian matrix. Most of the information needed by output sensitivity analysis is already available from the learning equations. The output sensitivity analysis approach presented in this thesis does not rely on simplifying assumptions as is the case with objective function sensitivity analysis. Complete sensitivity analysis equations for standard feedforward and product unit NNs were derived. It was also shown that the output sensitivity analysis approach is applicable to different multilayer NN types, including feedforward, product unit and functional link NNs. Experimental results showed that the analytical equations for standard feedforward NNs accurately approximate the true first order derivatives of the function being modeled.

Algorithms were developed to visualize and analyze decision boundaries formed in the input space. For this purpose two types of decision boundaries were defined, viewed from one dimension only. These definitions were used to develop algorithms to visually inspect the position of decision boundaries, using output sensitivity information with respect to input perturbations. It was shown how these visualization algorithms can be used to

- **identify irrelevant input parameters:** No boundaries are formed for irrelevant parameters, which will be indicated by sensitivity values of approximately zero over the entire input space.
- **analyze the functioning of hidden units:** Hidden units that implement no boundary can be detected, as well as hidden units that duplicate the function of other hidden units, i.e. units that implements the same boundary. Also, it can be determined which boundary is implemented by which hidden unit(s).

- **assign a measure of informativeness to patterns:** Patterns that lie close to decision boundaries have very high sensitivity values, and convey the most information about the position of boundaries.
- **aid in the extraction of accurate symbolic rules from trained NNs:** The visualized boundaries can be used as threshold values for continuous valued input parameters in rule clauses [Engelbrecht *et al* 1999a, Viktor *et al* 1995, Viktor 1998b]. It was not the objective of this thesis to explore this application further. The reader is, however, referred to [Viktor 1998b] where this application is explored in detail.

The decision boundary visualization and detection algorithms addressed the sub objectives of this thesis by providing a mechanism through which we gain a better understanding of the functioning of multilayer feedforward NNs and greater comprehensibility of the numerically encoded knowledge for classification problems.

Two new active learning algorithms were developed:

- **A selective learning algorithm** for classification problems, where the training set is pruned by removing uninformative patterns during training. Training is done on the most informative patterns, which are those patterns that lie closest to decision boundaries. For this purpose, the sensitivity analysis decision boundary detection algorithm was used to locate all patterns near boundaries.
- **An incremental learning algorithm**, where patterns are incrementally selected from a candidate set of patterns and added to the actual training set. The training set therefore incrementally grows with the most informative patterns being added to it. The most informative patterns are those that cause the largest change in NN output, and therefore the largest weight changes.

For the purposes of these active learning algorithms, the concept of pattern informativeness was defined as the influence small perturbations in any of the input values of that pattern have on the NN output.

Both the selective learning and incremental learning algorithms resulted in a substantial saving in computational cost during training. In general, these two approaches resulted in improved

convergence properties, and had less overfitting than normal fixed set learning (FSL). The selective learning algorithm illustrated similar generalization performances compared to FSL, while the incremental learning algorithm showed substantially improved generalization for most of the problems investigated.

A sensitivity analysis tool was developed to prune feedforward NN architectures. Theoretical models were developed to prune input units, hidden units and weights. Experimental results illustrated the application of the sensitivity analysis pruning algorithm to the pruning of input and hidden units. The thesis defined two mechanisms to determine the relevance of NN parameters, which resulted in the development of two pruning heuristics. The first approach calculates the significance of parameters as a norm over the sensitivity of output units to perturbations in that parameters, over the entire training set. A pruning heuristic was proposed that orders the parameter significance values and finds a large enough gap between consecutive parameters to allow pruning of low significance parameters. This heuristic introduced a problem dependent gap constant for which an optimal value needs to be determined.

For this reason a new statistical measure based on variance analysis was defined to express the importance of parameters. The parameter variance nullity measure quantifies the variance in output sensitivity due to parameter perturbations over a given set of patterns. The pruning heuristic consists of formal hypothesis tests to determine if the variance in output sensitivity with respect to a network parameter is significantly different from zero. If not, the parameter is pruned. This new measure of parameter importance and pruning heuristic provide a statistically justified mechanism to prune irrelevant parameters, with no problem dependent parameters for which optimal values need to be determined.

The pruning algorithms developed in this thesis addresses the generalization and complexity sub objectives of the thesis. Through removal of excess parameters, overfitting is avoided, and the complexity of the NN model is reduced. Also, pruning of NN architectures has been shown to improve the accuracy of rules extracted from trained networks, and to reduce the complexity of rule extraction algorithms [Viktor *et al* 1995, Viktor 1998b].

6.1 Future Work

The sensitivity analysis techniques presented in this thesis were developed for feedforward NNs using sigmoid activation functions. It is proposed that future research include empirical investigations into the applicability of these sensitivity analysis techniques to other NN types, including recurrent, product unit and functional link NNs, using differentiable activation functions other than sigmoid functions.

With reference to the individual sensitivity analysis techniques developed in this thesis, future research could include

- an extension of the selective learning algorithm to allow for dynamic values of the subset selection constant;
- a study of the performance of the incremental learning algorithm under different subset sizes;
- an empirical investigation into the effect of outliers on both the selective and incremental learning algorithms;
- an investigation into the applicability of the incremental learning algorithm to classification problems;
- application of the pruning algorithm to the removal of irrelevant weights; and
- extending the pruning heuristic to include an F-test to statistically determine if a pruned NN architecture can be accepted.
- application of the developed sensitivity analysis tools to other NN types, e.g. recurrent, functional link, and product unit NNs.

The empirical results of this thesis showed output sensitivity analysis to be an efficient approach to improve the performance of multilayer feedforward NNs. A key advantage of the NN output sensitivity analysis approach is that no simplifying assumptions are made, and that the approach is simple to use and by no means computationally complex. Most of the information needed is already available from learning equations. The approach is very flexible

being independent of the objective function and optimization method, and it is applicable to any differentiable, monotonic increasing function.

"By perseverance the snails reached the ark."

-Charles Haddon Spurgeon

Bibliography

[Abu-Mostafa 1989] YS Abu-Mostafa, *The Vapnik-Chervonenkis Dimension: Information versus Complexity in Learning*, Neural Computation, Vol 1, 1989, pp 312-317.

[Abu-Mostafa 1993] YS Abu-Mostafa, *Hints and the VC Dimension*, Neural Computation, Vol 5, 1993, pp 278-288.

[Adejuma 1999] A Adejuma, *A Comparative Study of Active Learning Techniques in Multilayer Feedforward Neural Networks*, M.Sc Thesis in preparation, Department of Computer Science, University of Pretoria, South Africa, 1999.

[Ahmad *et al* 1989] S Ahmad, G Tesauro, *Scaling and Generalization in Neural Networks: A Case Study*, Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 1989, pp 3-10.

[Akaike 1974] H Akaike, *A New Look at Statistical Model Identification*, IEEE Transactions on Automatic Control, 19(6), 1974, pp 716-723.

[Alippi *et al* 1995] C Alippi, V Piuri, M Sami, *Sensitivity to Errors in Artificial Neural Networks: A Behavioral Approach*, IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications, 42(6), 1995, pp 358-361.

[Amari *et al* 1995] S Amari, N Murata, K-R Müller, M Finke, H Yanh, *Asymptotic Statistical Theory of Overtraining and Cross-Validation*, Technical Report METR 95-06, Department of Mathematical Engineering and Information, University of Tokyo, 1995.

[Amari *et al* 1996] S Amari, N Murata, K-R Müller, M Finke, H Yang, *Statistical Theory of Overtraining - Is Cross-Validation Asymptotically Effective?*, DS Touretzky, MC

Mozer, ME Hasselmo (eds), Advances in Neural Information Processing Systems, Vol 8, 1996, pp 176-182.

[Barnard 1991] E Barnard, *Performance and Generalization of the Classification Figure of Merit Criterion Function*, IEEE Transactions on Neural Networks, 2(2), 1991, pp 322-325.

[Basson *et al* 1999] E Basson, AP Engelbrecht, *Approximation of a Function and Its Derivatives in Feedforward Neural Networks*, IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999, paper 2152.

[Battiti 1992] R Battiti, *First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method*, Neural Computation, Vol 4, 1992, pp 141-166.

[Baum 1988] EB Baum, *On the Capabilities of Multilayer Perceptrons*, Journal of Complexity, Vol 4, 1988, pp 193-215.

[Baum *et al* 1989] EB Baum, D Haussler, *What Size Net Gives Valid Generalization?*, DS Touretzky (ed), Advances in Neural Information Processing Systems, Vol 1, 1989, pp 81-90.

[Baum 1991] EB Baum, *Neural Net Algorithms that Learn in Polynomial Time from Examples and Queries*, IEEE Transactions on Neural Networks, 2(1), 1991, pp 5-19.

[Baxt 1992] WG Baxt, *Improving the Accuracy of an Artificial Neural Network using Multiple Differently Trained Networks*, Neural Computation, Vol 4, 1992, pp 772-780.

[Becker *et al* 1988] S Becker, Y Le Cun, *Improving the Convergence of Back-Propagation Learning with Second Order Methods*, DS Touretzky, GE Hinton, TJ Sejnowski (eds), Proceedings of the 1988 Connectionist Summer School, Morgan Kaufmann:Los Angeles, 1988.

[Belue *et al* 1995] LM Belue, KW Bauer, *Determining Input Features for Multilayer Perceptrons*, Neurocomputing, Vol 7, 1995, pp 111-121.

[Bishop 1992] C Bishop, *Exact Calculation of the Hessian Matrix for the Multilayer Perceptron*, Neural Computation, Vol 4, 1992, pp 494-501.

[Blanning 1974] RW Blanning, *The Sources and Uses of Sensitivity Information*, Interfaces, 4(4), 1974, pp 32-38.

[Bös 1996] S Bös, *Optimal Weight Decay in a Perceptron*, International Conference on Artificial Neural Networks, 1996, pp 551-556.

[Buntine *et al* 1994] WL Buntine, AS Weigend, *Computing Second Order Derivatives in Feed-Forward Networks: A Review*, IEEE Transactions on Neural Networks, 5(3), 1994, pp 480-488.

[Burrascano 1993] P Burrascano, *A Pruning Technique Maximizing Generalization*, Proceedings of the 1993 International Joint Conference on Neural Networks, Vol 1, 1993, pp 347-350.

[Cao 1985] X-R Cao, *Convergence of Parameter Sensitivity Estimates in a Stochastic Experiment*, IEEE Transactions on Automatic Control, 30(9), 1985, pp 845-853.

[Chauvin 1989] Y Chauvin, *A Back-Propagation Algorithm with Optimal use of Hidden Units*, DS Touretzky (ed), Advances in Neural Information Processing Systems, Vol 1, 1989, pp 519-526.

[Chauvin 1990] Y Chauvin, *Dynamic Behavior of Constrained Back-Propagation Networks*, DS Touretzky (ed), Advances in Neural Information Processing Systems, Vol 2, 1990, pp 642-649.

[Choi *et al* 1992] JY Choi, C-H Choi, *Sensitivity Analysis of Multilayer Perceptron with Differential Activation Functions*, IEEE Transactions on Neural Networks, 3(1), 1992, pp 101-107.

[Cibas *et al* 1994a] T Cibas, F Fogelman Soulié, P Gallinari, S Raudys, *Variable Selection with Neural Networks*, International Conference on Artificial Neural Networks, 1994, pp 1464-1469.

[Cibas *et al* 1994b] T Cibas, F Fogelman Soulié, P Gallinari, S Raudys, *Variable Selection with Optimal Cell Damage*, International Conference on Artificial Neural Networks, 1994, pp 727-730.

[Cibas *et al* 1996] T Cibas, F Fogelman Soulié, P Gallinari, S Raudys, *Variable Selection with Neural Networks*, Neurocomputing, Vol 12, 1996, pp 223-248.

[Cloete *et al* 1993] I Cloete, J Ludik, *Increased Complexity Training*, International Workshop on Artificial Neural Networks, in "New Trends in Neural Computation," J Mira, J Cabestany, A Prieto (eds), in series Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993, pp 267-271.

[Cloete *et al* 1994a] I Cloete, J Ludik, *Incremental Training Strategies*, Proceedings of the International Conference on Artificial Neural Networks, Sorrento, Italy, Vol 2, May 1994, pp 743-746.

[Cloete *et al* 1994b] I Cloete, J Ludik, *Delta Training Strategies*, IEEE World Congress on Computational Intelligence, Orlando, USA, Vol 1, 1994, pp 295-298.

[Cloete *et al* 1994c] I Cloete, AP Engelbrecht, *New Tools for Decision Support*, AMSE International Conference on Intelligent Systems, Pretoria, 1994.

[Cohn *et al* 1991] D Cohn, G Tesauro, *Can Neural Networks do Better than the Vapnik-Chervonenkis Bounds?*, R Lippmann, J Moody, DS Touretzky (eds), Advances in Neural Information Processing Systems, Vol 3, 1991, pp 911-917.

[Cohn 1994a] DA Cohn, *Neural Network Exploration using Optimal Experiment Design*, AI Memo No 1491, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1994.

[Cohn *et al* 1994b] D Cohn, L Atlas, R Ladner, *Improving Generalization with Active Learning*, Machine Learning, Vol 15, 1994, pp 201-221.

[Cohn *et al* 1996] DA Cohn, Z Ghahramani, MI Jordan, *Active Learning with Statistical Models*, Journal of Artificial Intelligence Research, Vol 4, 1996, pp129-145.

[Cohen *et al* 1997] B Cohen, D Saad, E Marom, *Efficient Training of Recurrent Neural Networks with Time Delays*, Neural Networks, 10(1), 1997, pp 51-59.

[Cosnard *et al* 1992] M Cosnard, P Koiran, H Paugam-Moisy, *Complexity Issues in Neural Network Computations*, LATIN92, Sau Paulo, Brazil, 1992.

[Cottrell *et al* 1994] M Cottrell, B Girard, Y Girard, M Mangeas, C Muller, *SSM: A Statistical Stepwise Method for Weight Elimination*, International Conference on Artificial Neural Networks, Vol 1, 1994, pp 681-684.

[Craven *et al* 1993] MW Craven, JW Shavlik, *Learning Symbolic Rules using Artificial Neural Networks*, PE Utgoff (ed), Proceedings of the 10th International Conference on Machine Learning, Morgan Kaufmann, 1993.

[Crespo *et al* 1993] JL Crespo, E Mora, *Tests of Different Regularization Terms in Small Networks*, International Workshop on Artificial Neural Networks, in "New Trends in Neural Computation," J Mira, J Cabestany, A Prieto (eds), in series Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993, pp 284-289.

[Czernichow 1996] T Czernichow, *Architecture Selection through Statistical Sensitivity Analysis*, International Conference on Artificial Neural Networks, 1996, pp 179-184.

[Darken *et al* 1992] C Darken, J Moody, *Towards Faster Stochastic Gradient Search*, J Moody, SJ Hanson, R Lippmann (eds), Advances in Neural Information Processing Systems, Vol 4, 1992, pp 1009-1016.

[Denoeux *et al* 1993] T Denoeux, R Lengellé, *Initializing Back Propagation Networks using Prototypes*, Neural Networks, 6(3), 1993, pp 351-363.

[Depenau *et al* 1994] J Depenau, M Møller, *Aspects of Generalization and Pruning*, World Congress on Neural Networks, Vol 3, 1994, pp 504-509.

[Dorizzi *et al* 1996] B Dorizzi, G Pellieux, F Jacquet, T Czernichow, A Muñoz, *Variable Selection using Generalized RBF Networks: Application to the Forecast of the French T-Bonds*, Computational Engineering in Systems Applications, Lille, France, 1996, pp 122-127.

[Durbin *et al* 1989] R Durbin, DE Rumelhart, *Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks*, Neural Computation, Vol 1, 1989, pp 133-142.

[Engelbrecht *et al* 1995a] AP Engelbrecht, I Cloete, J Geldenhuys, JM Zurada, *Automatic Scaling using Gamma Learning for Feedforward Neural Networks*, International Workshop on Artificial Neural Networks, Torremolinos, Spain, June 1995, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing," in the series Lecture Notes in Computer Science, Vol 930, pp 374-381.

[Engelbrecht *et al* 1995b] AP Engelbrecht, I Cloete, JM Zurada, *Determining the Significance of Input Parameters using Sensitivity Analysis*, International Workshop on Artificial Neural Networks, Torremolinos, Spain, June 1995, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing," in the series Lecture Notes in Computer Science, Vol 930, pp 382-388.

[Engelbrecht *et al* 1996] AP Engelbrecht, I Cloete, *A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks*, IEEE International Conference in Neural Networks, Washington, Vol 2, 1996, pp 1274-1277.

[Engelbrecht *et al* 1998a] AP Engelbrecht, I Cloete, *Selective Learning using Sensitivity Analysis*, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp 1150-1155.

[Engelbrecht *et al* 1998b] AP Engelbrecht, I Cloete, *Feature Extraction from Feedforward Neural Networks using Sensitivity Analysis*, International Conference on Systems, Signals, Control, Computers, Durban, South Africa, Vol 2, 1998, pp 221-225.

[Engelbrecht *et al* 1999a] AP Engelbrecht, HL Viktor, *Rule Improvement through Decision Boundary Detection using Sensitivity Analysis*, the International Work Conference on Neural Networks (IWANN'99), 2-4 June, Alicante: Spain, Lecture Notes in Computer Science (Volume 1607), Springer-Verlag, Berlin: Germany, pp 78-84.

[Engelbrecht 1999c] AP Engelbrecht, *Sensitivity Analysis for Decision Boundaries*, Neural Processing Letters, Vol 10, 1999, pp 1-14.

[Engelbrecht *et al* 1999d] AP Engelbrecht, I Cloete, *Incremental Learning using Sensitivity Analysis*, IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999, paper 380.

[Engelbrecht *et al* 1999e] AP Engelbrecht, L Fletcher, I Cloete, *Variance Analysis of Sensitivity Information for Pruning Feedforward Neural Networks*, IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999, paper 379.

[Finnoff *et al* 1993] W Finnoff, F Hergert, HG Zimmermann, *Improving Model Selection by Nonconvergent Methods*, Neural Networks, Vol 6, 1993, p 771-783.

[Fletcher *et al* 1994] GP Fletcher, CJ Hinde, *Learning the Activation Function for the Neurons in Neural Networks*, International Conference on Artificial Neural Networks, Vol 1, 1994, pp 611-614.

[Fletcher *et al* 1998] L Fletcher, V Katkovnik, FE Steffens, AP Engelbrecht, *Optimizing the Number of Hidden Nodes of a Feedforward Artificial Neural Network*, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp 1608-1612.

[Fritzke 1995] B Fritzke, *Incremental Learning of Local Linear Mappings*, International Conference on Artificial Neural Networks, Paris, October 1995.

[Fu *et al* 1993] L Fu, T Chen, *Sensitivity Analysis for Input Vector in Multilayer Feedforward Neural Networks*, IEEE International Conference on Neural Networks, Vol 1, 1993, pp 215-218.

[Fujita 1992] O Fujita, *Optimization of the Hidden Unit Function in Feedforward Neural Networks*, Neural Networks, Vol 5, 1992, pp 755-764.

[Fukumizu 1996] K Fukumizu, *Active Learning in Multilayer Perceptrons*, DS Touretzky, MC Mozer, ME Hasselmo (eds), Advances in Neural Information Processing Systems, Vol 8, 1996, pp 295-301.

[Funahashi 1989] K-I Funahashi, *On the Approximate Realization of Continuous Mappings by Neural Networks*, Neural Networks, Vol 2, 1989, pp 183-192.

[Gallant *et al* 1992] AR Gallant, H White, *On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks*, Neural Networks, Vol 5, 1992, pp 129-138.

[Gaynier *et al* 1995] RJ Gaynier, T Downs, *Sinusoidal and Monotonic Transfer Functions: Implications for VC Dimension*, Neural Networks, 8(6), 1995, pp 901-904.

[Gedeon *et al* 1995] TD Gedeon, PM Wong, D Harris, *Balancing Bias and Variance: Network Topology and Pattern Set Reduction Techniques*, International Workshop on Artificial Neural Networks, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing," in the series Lecture Notes in Computer Science, Vol 930, 1995, pp 551-558.

[Geman *et al* 1992] S Geman, E Bienenstock, R Dousart, *Neural Networks and the Bias/Variance Dilemma*, Neural Computation, Vol 4, 1992, pp 1-58.

[Ghosh *et al* 1992] J Ghosh, Y Shin, *Efficient Higher-Order Neural Networks for Classification and Function Approximation*, International Journal of Neural Systems, 3(4), 1992, pp 323-350.

[Girosi *et al* 1995] F Girosi, M Jones, T Poggio, *Regularization Theory and Neural Networks Architectures*, Neural Computation, Vol 7, 1995, pp 219-269.

[Goh 1993] T-H Goh, *Semantic Extraction using Neural Network Modelling and Sensitivity Analysis*, Proceedings of the 1993 International Joint Conference on Neural Networks, 1993, pp 1031-1034.

[Gorodkin *et al* 1993b] J Gorodkin, LK Hansen, A Krogh, C Svarer, *A Quantitative Study of Pruning by Optimal Brain Damage*, International Journal of Neural Systems, 4(2), 1993, pp 159-169.

[Gu *et al* 1996] H Gu, H Takahashi, *Towards more Practical Average Bounds on Supervised Learning*, IEEE Transactions on Neural Networks, 7(4), 1996, pp 953-986.

[Gu *et al* 1997] H Gu, H Takahashi, *Estimating Learning Curves of Concept Learning*, Neural Networks, 10(6), 1997, pp 1089-1102.

[Guo *et al* 1992] Z Guo, RE Uhrig, *Sensitivity Analysis and Applications to Nuclear Power Plant*, Proceedings of the IEEE, Vol 2, 1992, pp 453-458.

[Hagiwara 1993] M Hagiwara, *Removal of Hidden Units and Weights for Back Propagation Networks*, Proceedings of the 1993 International Joint Conference on Neural Networks, Vol 1, 1993, pp 351-354.

[Hampshire *et al* 1990] JB Hampshire, AH Waibel, *A Novel Objective Function for Improved Phoneme Recognition using Time-Delay Neural Networks*, IEEE Transactions on Neural Networks, 1(2), 1990, pp 216-228.

[Hanson *et al* 1989] SJ Hanson, LY Pratt, *Comparing Biases for Minimal Network Construction with Back-Propagation*, DS Touretzky (ed), Advances in Neural Information Processing Systems, Vol 1, 1989, pp 177-185.

[Hashem 1992] S Hashem, *Sensitivity Analysis for Feedforward Artificial Neural Networks with Differential Activation Functions*, Proceedings of the IEEE, Vol 1, 1992, pp 419-424.

[Hassibi *et al* 1993] B Hassibi, DG Stork, *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon*, C Lee Giles, SJ Hanson, JD Cowan (eds), Advances in Neural Information Processing Systems, Vol 5, 1993, pp 164-171.

[Hassibi *et al* 1994] B Hassibi, DG Stork, G Wolff, *Optimal Brain Surgeon: Extensions and Performance Comparisons*, JD Cowan, G Tesauro, J Alspector (eds), Advances in Neural Information Processing Systems, Vol 6, 1994, pp 263-270.

[Hashem *et al* 1994] S Hashem, B Schmeiser, Y Yih, *Optimal Linear Combinations of Neural Networks: An Overview*, IEEE International Joint Conference on Neural Networks, 1994, pp 1507-1512.

[Hassoun 1995] MH Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, 1995.

[Haussler *et al* 1992] D Haussler, M Kearns, M Opper, R Schapire, *Estimating Average-Case Learning Curves using Bayesian, Statistical Physics and VC Dimension Method*, J Moody, SJ Hanson, R Lippmann (eds), Advances in Neural Information Processing Systems, Vol 4, 1992, pp 855-862.

[Hayashi 1993] M Hayashi, *A Fast Algorithm for the Hidden Units in a Multilayer Perceptron*, Proceedings of the 1993 International Joint Conference on Neural Networks, Vol 1, 1993, pp 339-342.

[Hirose *et al* 1991] Y Hirose, K Yamashita, S Hijiya, *Back-Propagation Algorithm which Varies the Number of Hidden Units*, Neural Networks, Vol 4, 1991, pp 61-66.

[Ho 1987] Y-C Ho, *Performance Evaluation and Perturbation Analysis of Discrete Time Dynamic Systems*, IEEE Transactions on Automatic Control, 32(7), 1987, pp 563-572.

[Ho 1988] Y-C Ho, *Perturbation Analysis Explained*, IEEE Transactions on Automatic Control, 33(8), 1988, pp 761-763.

[Hoaglin *et al* 1983] DC Hoaglin, F Mosteller, JW Tukey, *Understanding Robust and Exploratary Data Analysis*, John Wiley & Sons, 1983.

[Hole 1996] A Hole, *Vapnik-Chervonenkis Generalization Bounds for Real Valued Neural Networks*, Neural Computation, Vol 8, 1996, pp 1277-1299.

[Holmström *et al* 1992] L Holmström, P Koistinen, *Using Additive Noise in Back-Propagation Training*, IEEE Transactions on Neural Networks, Vol 3, 1992, pp 24-38.

[Holtzman 1992] JM Holtzman, *On using Perturbation Analysis to do Sensitivity Analysis: Derivatives versus Differences*, IEEE Transactions on Automatic Control, 37(2), 1992, pp 243-247.

[Hornik 1989] K Hornik, *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, Vol 2, 1989, pp 359-366.

[Hornik *et al* 1990] K Hornik, M Stinchcombe, H White, *Universal Approximation of an Unknown Mapping and Its Derivatives using Multilayer Feedforward Networks*, Neural Networks, Vol 3, 1990, pp 551-560.

[Huang 1994] K-Y Huang, *Sequential Classification by Perceptrons and Application to Net Pruning of Multilayer Perceptron*, World Congress on Neural Networks, Vol 3, 1994, pp 510-515.

[Huber 1981] PJ Huber, *Robust Statistics*, John Wiley & Sons, 1981.

[Hüning 1993] H Hüning, *A Node Splitting Algorithm that Reduces the Number of Connections in a Hamming Distance Classifying Network*, International Workshop on Artificial Neural Networks, in "New Trends in Neural Computation," J Mira, J Cabestanyy, A Prieto (eds), in the series Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993, pp 102-107.

[Hussain *et al* 1997] A Hussain, JJ Soraghan, TS Durbani, *A New Neural Network for Non-linear Time-Series Modelling*, NeuroVest Journal, Jan 1997, pp 16-26.

[Hwang *et al* 1990] J-N Hwang, JJ Choi, S Oh, RJ Marks II, *Query Learning Based on Boundary Search and Gradient Computation of Trained Multilayer Perceptrons*, International Joint Conference on Neural Networks, Vol 3, 1990, pp 57-62.

[Hwang *et al* 1991] J-N Hwang, JJ Choi, S Oh, RJ Marks II, *Query-Based Learning Applied to Partially Trained Multilayer Perceptrons*, IEEE Transactions on Neural Networks, 2(1), January 1991, pp 131-136.

[Hunt *et al* 1995] SD Hunt, JR Deller (jr), *Selective Training of Feedforward Artificial Neural Networks using Matrix Perturbation Theory*, Neural Networks, 8(6), 1995, pp 931-944.

[Iwatsuki *et al* 1989] M Iwatsuki, M Kawamata, T Higuchi, *Statistical Sensitivity and Minimum Sensitivity Structures with Fewer Coefficients in Discrete Time Linear Systems*, IEEE Transactions on Circuits and Systems, 37(1), 1989, pp 72-80.

[Jabri *et al* 1991] M Jabri, B Flower, *Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks*, Neural Computation, Vol 3, 1991, pp 546-565.

[Jacobs 1989] RA Jacobs, *Increased Rates of Convergence through Learning Rate Adaptation*, Neural Networks, 4(1), 1989.

[Jacobs *et al* 1997] RA Jacobs, F Peng, MA Tanner, *A Bayesian Approach to Model Selection in Hierarchical Mixtures-of-Experts Architectures*, Neural Networks, 10(2), 1997, pp 231-241.

[Janson *et al* 1993] DJ Janson, JF Frenzel, *Training Product Unit Neural Networks with Genetic Algorithms*, IEEE Expert, Oct 1993, pp 26-33.

[Jasić *et al* 1995] T Jasić, HL Poh, *Analysis of Pruning in Backpropagation Networks for Artificial and Real World Mapping Problems*, International Workshop on Artificial Neural Networks, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing," in the series Lecture Notes in Computer Science, Vol 930, 1995, pp 239-245.

[Jutten *et al* 1995b] C Jutten, P Chentouf, *A New Scheme for Incremental Learning*, Neural Processing Letters, 2(1), 1995, pp 1-4.

[Kambhatla *et al* 1994] N Kambhatla, TK Leen, *Fast Non-Linear Dimension Reduction*, JD Cowan, G Tesauro, J Alspector (eds), Advances in Neural Information Processing Systems, Vol 6, 1994, pp 152-159.

[Kamruzzaman *et al* 1992] J Kamruzzaman, Y Kumagai, H Hikita, *Study on Minimal Net Size, Convergence Behavior and Generalization Ability of Heterogeneous Backpropagation Network*, I Aleksander, J Taylor (eds), Artificial Neural Networks, Vol 2, 1992, pp 203-206.

[Kamimura 1993] R Kamimura, *Principal Hidden Unit Analysis: Generation of Simple Networks by Minimum Entropy Method*, Proceedings of the 1993 International Joint Conference on Neural Networks, Vol 1, 1993, pp 317-320.

[Kamimura *et al* 1994a] R Kamimura, S Nakanishi, *Weight Decay as a Process of Redundancy Reduction*, World Congress on Neural Networks, Vol 3, 1994, pp 486-491.

[Kamimura *et al* 1994b] R Kamimura, T Takagi, S Nakanishi, *Minimum Information Principle: Improving Generalization Performance by Information Minimization*, World Congress on Neural Networks, Vol 3, 1994, pp 680-685.

[Karnin 1990] ED Karnin, *A Simple Procedure for Pruning Back-Propagation Trained Neural Networks*, IEEE Transactions on Neural Networks, 1(2), 1990, pp 239-242.

[Kehagias *et al* 1997] A Kehagias, V Petridis, *Predictive Modular Neural Networks for Time Series Classification*, Neural Networks, 10(1), 1997, pp 31-49.

[Kibsgaard 1992] S Kibsgaard, *Sensitivity Analysis - The Basis for Optimization*, International Journal for Numerical Methods in Engineering, Vol 34, 1992, pp 901-932.

[Koda 1995] M Koda, *Stochastic Sensitivity Analysis Method for Neural Network Learning*, International Journal of System Science, 26(3), 1995, pp 703-711.

[Koda 1997] M Koda, *Neural Network Learning based on Stochastic Sensitivity Analysis*, IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, 27(1), 1995, pp 132-135.

[Kohara 1995] K Kohara, *Selective Presentation Learning for Forecasting by Neural Networks*, International Workshop on Applications of Neural Networks in Telecommunications, Stockolm, Sweden, 1995, pp 316-323.

[Krogh *et al* 1992] A Krogh, JA Hertz, *A Simple Weight Decay can Improve Generalization*, J Moody, SJ Hanson, R Lippmann (eds), Advances in Neural Information Processing Systems, Vol 4, 1992, pp 950-957.

[Kuscu *et al* 1994] I Kuscu, C Thornton, *Design of Artificial Neural Networks using Genetic Algorithms: Review and Prospect*, Technical Report, Cognitive and Computing Sciences, University of Sussex, 1994.

[Kwok *et al* 1995] T-Y Kwok, D-Y Yeung, *Constructive Feedforward Neural Networks for Regression Problems: A Survey*, Technical Report HKUST-CS95-43, Department of Computer Science, The Hong Kong University of Science & Technology, 1995.

[Lamers *et al* 1998] MH Lamers, JN Kok, *A Multilevel Nonlinearity Study Design*, IEEE International World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp 730-734.

[Lange *et al* 1994] R Lange, R Männer, *Quantifying a Critical Training Set Size for Generalization and Overfitting using Teacher Neural Networks*, International Conference on Artificial Neural Networks, Vol 1, 1994, pp 497-500.

[Lange *et al* 1996] S Lange, T Zeugmann, *Incremental Learning from Positive Data*, Journal of Computer and System Sciences, Vol 53, 1996, pp 88-103.

[Laskey 1995] KB Laskey, *Sensitivity Analysis for Probability Assessments in Bayesian Networks*, IEEE Transactions on Systems, Man, and Cybernetics, 25(6), 1995, pp 901-909.

[Le Cun 1990] Y Le Cun, JS Denker, SA Solla, *Optimal Brain Damage*, D Touretzky (ed), Advances in Neural Information Processing systems, Vol 2, 1990, pp 598-605.

[Lee 1991] T-C Lee, *Structure Level Adaptation for Artificial Neural Networks*, Kluwer Academic Publishers, 1991.

[Lee *et al* 1992] C Lee, DA Landgrebe, *Decision Boundary Feature Extraction for Neural Networks*, Proceedings of the IEEE, 1992, pp 1053-1058.

[Leerink *et al* 1995] LR Leerink, C Lee Giles, BG Horne, MA Jabri, *Learning with Product Units*, Advances in Neural Information Processing Systems, Vol 7, 1995, pp 537-544.

[Leung *et al* 1996] CS Leung, KW Wong, PF Sum, LW Chen, *On-line Training and Pruning for Recursive Least Square Algorithms*, Electronic Letters, 32(23), 1996, pp 2152-2153.

[Levin *et al* 1994] AU Levin, TK Leen, JE Moody, *Fast Pruning using Principal Components*, JD Cowan, G Tesauro, J Alspector (eds), Advances in Neural Information Processing Systems, Vol 6, 1994, pp 35-42.

[Liang *et al* 1994] X Liang, S Xia, *Principle and Methods of Structure Variation in Feedforward Neural Networks*, World Congress on Neural Networks, Vol 3, 1994, pp 522-527.

[Lim *et al* 1994] S-F Lim, S-B Ho, *Dynamic Creation of Hidden Units with Selective Pruning in Backpropagation*, World Congress on Neural Networks, Vol 3, June 1994, pp 492-497.

[Ludik *et al* 1993] J Ludik, I Cloete, *Training Schedules for Improved Convergence*, International Joint Conference on Neural Networks, Nagoya, Japan, Vol 1, 1993, pp 561-564.

[Ludik *et al* 1994] J Ludik, I Cloete, *Incremental Increased Complexity Training*, European Symposium on Artificial Neural Networks, Brussels, Belgium, April 1994, pp 161-165.

[Ludik 1995a] J Ludik, *Training, Dynamics, and Complexity of Architecture-Specific Recurrent Neural Networks*, PhD Thesis, Department of Computer Science, University of Stellenbosch, February 1995.

[Ludik *et al* 1995b] J Ludik, I Cloete, *Training Strategies for Architecture-Specific Recurrent Neural Networks*, The South African Computer Journal, Number 14, 1995, pp 48-58.

[Ludik *et al* 1996] J Ludik, I Cloete, *Bounds for Hidden Units of Simple Recurrent Networks*, IEEE International Joint Conference on Neural Networks, Washington DC, 1996, pp 323-328.

[MacKay 1992a] DJC MacKay, *Bayesian Methods for Adaptive Models*, PhD Thesis, California Institute of Technology, 1992.

[MacKay 1992b] DJC MacKay, *Information-Based Objective Functions for Active Data Selection*, Neural Computation, Vol 4, 1992, pp 590-604.

[Magoulas *et al* 1997] GD Magoulas, MN Vrahatis, GS Androulakis, *Effective Backpropagation Training with Variable Stepsize*, Neural Networks, 10(1), 1997, pp 69-82.

[Maillard *et al* 1994] E Maillard, G Gueriot, *Learning the Sigmoid Slopes to Increase the Convergence speed of the Multilayer Perceptron*, World Congress on Neural Networks, Vol 3, 1994, pp 539-544.

[Malinowski *et al* 1993] A Malinowski, J Zurada, *Minimal Training Set Size for Neural Network-Based Function Encoding*, Intelligent Engineering Systems through Artificial Neural Networks, 1993, pp 149-154.

[Mitchell 1997] TM Mitchell, *Machine Learning*, MacGraw Hill, 1997.

[Modai *et al* 1995] I Modai, NI Saban, M Stoler, A Valevski, N Saban, *Sensitivity Profile of 41 Psychiatric Parameters Determined by Neural Network in Relation to 8-Week Outcome*, Computers in Human Behavior, 11(2), 1995, pp 181-190.

[Møller 1993] MF Møller, *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, Vol 6, 1993, pp 525-533.

[Moody 1992] JE Moody, *The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems*, J Moody, SJ Hanson, R Lippmann (eds), Advances in Neural Information Processing Systems, Vol 4, 1992, pp 847-854.

[Moody 1994a] JE Moody, *Prediction Risk and Architecture Selection for Neural Networks*, V Cherkassky, JH Friedman, H Wechsler (eds), From Statistics to Neural Networks: Theory and Pattern Recognition Applications, Springer, 1994, pp 147-165.

[Moody *et al* 1995] J Moody, J Utans, *Architecture Selection Strategies for Neural Networks: Application to Corporate Bond Rating Prediction*, AN Refenes (ed), Neural Networks in the Capital Markets, John Wiley & Sons, 1995.

[Moody *et al* 1996] J Moody, PJ Antsaklis, *The Dependence Identification Neural Network Construction Algorithm*, IEEE Transactions on Neural Networks, 7(1), 1996, pp 3-15.

[Morgan *et al* 1991] DP Morgan, CL Scofield, *Neural Networks and Speech Processing*, Kluwer Academic Publishers, 1991.

[Mozer *et al* 1989] MC Mozer, P Smolensky, *Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment*, DS Touretzky (ed), Advances in Neural Information Processing Systems, Vol 1, 1989, pp 107-115.

[Müller *et al* 1995] K-R Müller, M Finke, N Murata, K Schulten, S Amari, *A Numerical Study on Learning Curves in Stochastic Multi-Layer Feed-Forward Networks*, Neural Computation, 8(5), 1995, pp 1085-1106.

[Murata *et al* 1991] N Murata, S Yoshizawa, S Amari, *A Criterion for Determining the Number of Parameters in an Artificial Neural Network Model*, T Kohonen, K Mäkisara, O Simula, J Kangas (eds), Artificial Neural Networks, Elsevier Science Publishers, 1991, pp 9-14.

[Murata *et al* 1994a] N Murata, S Yoshizawa, S Amari, *Learning Curves, Model Selection and Complexity of Neural Networks*, C Lee Giles, SJ Hanson, JD Cowan (eds), Advances in Neural Information Processing Systems, Vol 5, 1994, pp 607-614.

[Murata *et al* 1994b] N Murata, S Yoshizawa, S Amari, *Network Information Criterion - Determining the Number of Hidden Units for an Artificial Neural Network Model*, IEEE Transactions on Neural Networks, 5(6), 1994, pp 865-872.

[Nowlan *et al* 1992] SJ Nowlan, GE Hinton, *Simplifying Neural Networks By Soft Weight-Sharing*, Neural Computation, Vol 4, 1992, pp 473-493.

[Oh *et al* 1995] S-H Oh, Y Lee, *Sensitivity Analysis of Single Hidden-Layer Neural Networks with Threshold Functions*, IEEE Transactions on Neural Networks, 6(4), 1995, pp 1005-1007.

[Ohnishi *et al* 1990] N Ohnishi, A Okamoto, N Sugiem, *Selective Presentation of Learning Samples for Efficient Learning in Multi-Layer Perceptron*, International Joint Conference on Neural Networks, Vol 1, 1990, pp 688-691,

[Opper 1994] M Opper, *Learning and Generalization in a Two-Layer Neural Network: The Role of the Vapnik-Chervonenkis Dimension*, Physical Review Letters, 72(13), 1994, pp 2133-2166.

[Orr *et al* 1993] GB Orr, TK Leen, *Momentum and Optimal Stochastic Search*, Proceedings of the 1993 Connectionist Models Summer School, MC Mozer, P Smolensky, DS Touretzky, JL Elman, AS Weigend (eds), Erlbaum Associates, 1993.

[Pedersen *et al* 1996] MW Pedersen, LK Hansen, J Larsen, *Pruning with Generalization Based Weight Saliencies: γ OBD, γ OBS*, DS Touretzky, MC Mozer, ME Hasselmo (eds), Advances in Neural Information Processing Systems, Vol 8, 1996, pp 521-528.

[Pelillo *et al* 1993] M Pelillo, AM Fanelli, *A Method of Pruning Layered Feed-Forward Neural Networks*, International Workshop on Artificial Neural Networks, 1993, pp 278-283.

[Plutowski *et al* 1993] M Plutowski, H White, *Selecting Concise Training Sets from Clean Data*, IEEE Transactions on Neural Networks, 4(2), March 1993, pp 305-318.

[Pratt *et al* 1994] LY Pratt, AN Christensen, *Relaxing the Hyperplane Assumption in the Analysis and Modificatioon of Back-Propagation Neural Networks*, R Trappl (ed), Cybernetics and Systems, World Scientific, Singapore, 1994, pp 1711-1718.

[Prechelt 1994] L Prechelt, *PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules*, Technical Report 21/94, Fakultät für Informatik, Universität Karslruhe.

[Prechelt 1995] L Prechelt, *Adaptive Parameter Pruning in Neural Networks*, Technical Report TR-95-009, International Computer Science Institute, Berkeley, California, 1995.

[Reed 1993] R Reed, *Pruning Algorithms - A Survey*, IEEE Transactions on Neural Networks, 4(5), 1993, pp 740-747.

[Rigler *et al* 1991] AK Rigler, JM Irvine, TP Vogl, *Rescaling of Variables in Back Propagation Learning*, Neural Networks, Vol 4, 1991, pp 225–229.

[Röbel 1994a] A Röbel, *Dynamic Pattern Selection: Effectively Training Backpropagation Neural Networks*, International Conference on Artificial Neural Networks, Vol 1, 1994, pp 643-646.

[Röbel 1994b] A Röbel, *Dynamic Pattern Selection for Faster Learning and Controlled Generalization of Neural Networks*, European Symposium on Artificial Neural Networks, 1994.

[Röbel 1994c] A Röbel, *The Dynamic Pattern Selection Algorithm: Effective Training and Controlled Generalization of Backpropagation Neural Networks*, Technical Report, Institute für Angewandte Informatik, Technische Universität Berlin, March 1994, pp 497-500.

[Rosen *et al* 1997] BE Rosen, JM Goodwin, *Optimizing Neural Networks using Very Fast Simulated Annealing*, Neural, Parallel & Scientific Computations, 5(3), 1997, pp 383-392.

[Rumelhart *et al* 1986] DE Rumelhart, GE Hinton, RJ Williams, *Learning Internal Representation by Error Propagation*, DE Rumelhart, JL McClelland and the PDP Research Group (eds), Parallel Distributed Processing, Vol 1, MIT Press, Cambridge, Mass, 1986.

[Sakurai *et al* 1992a] A Sakurai, M Yamasaki, *On the Capacity of n-h-s Networks*, I Aleksander, J Taylor (eds), Artificial Neural Networks, Vol 2, 1992, pp 237-240.

[Sakurai 1992b] A Sakurai, *n-h-1 Networks Use No Less than nh+1 Examples, but Sometimes More*, Proceedings of the IEEE, Vol 3, 1992, pp 936-941.

[Salomon *et al* 1996] R Salomon, JL Van Hemmen, *Accelerating Backpropagation through Dynamic Self-Adaptation*, Neural Networks, 9(4), 1996, pp 589-601.

[Sarle 1995] WS Sarle, *Stopped Training and Other Remedies for Overfitting*, Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics, 1995, pp 352-360.

[Sartori *et al* 1991] MA Sartori, PJ Antsaklis, *A Simple Method to Derive Bounds on the Size and to Train Multilayer Neural Networks*, IEEE Transactions on Neural Networks, 2(4), 1991, pp 467-471.

[Schittenkopf *et al* 1997] C Schittenkopf, G Deco, W Brauer, *Two Strategies to Avoid Overfitting in Feedforward Neural Networks*, Neural Networks, 10(30), 1997, pp 505-516.

[Schwartz *et al* 1990] DB Schwartz, VK Samalam, SA Solla, JS Denker, *Exhaustive Learning*, Neural Computation, Vol 2, 1990, pp 374-385.

[Seung *et al* 1992] HS Seung, M Opper, H Sompolinsky, *Query by Committee*, Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, 1992, pp 287-299.

[Sietsma *et al* 1991] J Sietsma, RJF Dow, *Creating Artificial Neural Networks that Generalize*, Neural Networks, Vol 4, 1991, pp 67-79.

[Slade *et al* 1993] P Slade, TD Gedeon, *Bimodal Distribution Removal*, International Workshop on Artificial Neural Networks, in "New Trends in Neural Computation," J Mira, J Cabestany, A Prieto (eds), in the series Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993, pp 249-254.

[Sollich *et al* 1996] P Sollich, A Krogh, *Learning with Ensembles: How Over-fitting can be useful*, DE Touretzky, MC Mozer, ME Hasselmo (eds), Advances in Neural Information Processing Systems, Vol 8, 1996, pp 190-196.

[Steppe *et al* 1996] JM Steppe, KW Bauer, SK Rogers, *Integrated Feature and Architecture Selection*, IEEE Transactions on Neural Networks, 7(4), 1996, pp 1007-1014.

[Stevenson *et al* 1990] M Stevenson, R Winter, B Widrow, *Sensitivity of Feedforward Neural Networks to Weight Errors*, IEEE Transactions on Neural Networks, 1(1), 1990, pp 71-80.

[Steyn *et al* 1995] AGW Steyn, CF Smit, SHC du Toit, C Strasheim, *Moderne Statistiek vir die Praktyk*, JL van Schaik Academic Publishers, 1995.

[Sung *et al* 1996] KK Sung, P Niyogi, *A Formulation for Active Learning with Applications to Object Detection*, AI Memo No 1438, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1996.

[Takahashi 1993] T Takahashi, *Principal Component Analysis is a Group Action of SO(N) which Minimizes an Entrpoy Function*, Proceedings of the 1993 International Joint Conference on Neural Networks, Vol 1, 1993, pp 355-358.

[Takenaga *et al* 1991] H Takenaga, S Abe, M Takamoto, M Kayama, T Kitamura, Y Okuyama, *Input Layer Optimization of Neural Networks by Sensitivity Analysis and Its Application to Recognition of Numerals*, Electrical Engineering in Japan, 111(4), 1991, pp 130-138.

[Tamura *et al* 1993] S Tamura, M Tateishi, M Matumoto, S Akita, *Determination of the Number of Redundant Hidden Units in a Three-layer Feed-Forward Neural Network*, Proceedings of the 1993 International Joint Conference on Neural Networks, Vol 1, 1993, pp 335-338.

[Tang *et al* 1994] Z Tang, GJ Koehler, *Deterministic Global Optimal FNN Training Algorithm*, Neural Networks, 7(2), 1994, pp 301-311.

[Thimm *et al* 1996] G Thimm, P Moerland, E Fiesler, *The Interchangeability of Learning Rate and Gain in Backpropagation Neural Networks*, Neural Computation, 8(2), 1996, pp 451-460.

[Thodberg 1991] HH Thodberg, *Improving Generalization of Neural Networks through Pruning*, International Journal of Neural Systems, 1(4), 1991, pp 317-326.

[Towell *et al* 1993] GG Towell, JW Shavlik, *Extracting Refined Rules from Knowledge-Based Neural Networks*, Machine Learning, Vol 13, 1993, pp 71-101.

[Tumer *et al* 1996] K Tumer, J Gosh, *Analysis of Decision Boundaries in Linearly Combined Neural Classifiers*, Pattern Recognition, 29(2), 1996, pp 341-348.

[UCI] University California Irvene Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html

[Viktor *et al* 1995] HL Viktor, AP Engelbrecht, I Cloete, *Reduction of Symbolic Rules from Artificial Neural Networks using Sensitivity Analysis*, IEEE International Conference in Neural Networks, Perth, Australia, 1995, pp 1788-1793.

[Viktor *et al* 1998a] HL Viktor, AP Engelbrecht, I Cloete, *Incorporating Rule Extraction from ANNs into a Cooperative Learning Environment*, Neural Networks & Their Applications, Marseilles, France, March 1998, pp 386-391.

[Viktor 1998b] HL Viktor, *Learning by Cooperation: An Approach to Rule Induction and Knowledge Fusion*, PhD thesis in preparation, Department of Computer Science, University of Stellenbosch, South Africa, 1998.

[Vogl *et al* 1988] TP Vogl, JK Mangis, AK Rigler, WT Zink, DL Alken, *Accelerating the Convergence of the Back-Propagation Method*, Biological Cybernetics, Vol 59, 1988, pp 257-263.

[Wang *et al* 1994] K Wang, AN Michel, *Robustness and Perturbation Analysis of a Class of Artificial Neural Networks*, Neural Networks, 7(2), 1994, pp 251-259.

[Weigend *et al* 1991] AS Weigend, DE Rumelhart, BA Huberman, *Generalization by Weight-Elimination with Application to Forecasting*, R Lippmann, J Moody, DS Touretzky (eds), Advances in Neural Information Processing Systems, Vol 3, 1991, pp 875-882.

[Weigl *et al* 1994] M Weigl, W Kosiński, *A Neural Network System for Approximation and its Implementation*, Proceedings of I Krajowa Konferencja "Sieci Neuronowe i Ich Zastosowania", 1994.

[Weir 1990] MK Weir, *A Method for Self-Determination of Adaptive Learning Rates in Back Propagation*, Neural Networks, 1990, pp 371-379.

[Wessels *et al* 1992] LFA Wessels, E Barnard, *Avoiding False Local Minima by Proper Initialization of Connections*, IEEE Transactions on Neural Networks, 3(6), 1992, pp 899-905.

[Williams 1995] PM Williams, *Bayesian Regularization and Pruning Using a Laplace Prior*, Neural Computation, Vol 7, 1995, pp 117-143.

[Whitley *et al* 1990] D Whitley, C Bogart, *The Evolution of Connectivity: Pruning Neural Networks using Genetic Algorithms*, International Joint Conference on Neural Networks, Vol 1, 1990, pp 134-137.

[White *et al* 1993] D White, P Ligomenides, *GANNet: A Genetic Algorithm for Optimizing Topology and Weights in Neural Network Design*, International Workshop on Artificial Neural Networks, in "New Trends in Neural Computation," J Mira, J Cabestany, A

Prieto (eds), in the series Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993, pp 332-327.

[Wynne-Jones 1992] M Wynne-Jones, *Node Splitting: A Constructive algorithm for Feed-Forward Neural Networks*, J Moody, SJ Hanson, R Lippmann (eds), Advances in Neural Information Processing Systems, Vol 4, 1992, pp 1072-1079.

[Xue *et al* 1990] Q Xue, Y Hu, WJ Tompkins, *Analyses of the Hidden Units of Back-Propagation Model*, International Joint Conference on Neural Networks, Vol 1, 1990, pp 739-742.

[Yasui 1997] S Yasui, *Convergence Suppression and Divergence Facilitation: Minimum and Joint Use of Hidden Units by Multiple Outputs*, Neural Networks, 10(2), 1997, pp 353-367.

[Yu *et al* 1997] X-H Yu, G-A Chen, *Efficient Backpropagation Learning using Optimal Learning Rate and Momentum*, Neural Networks, 10(3), 1997, pp 517-527.

[Zhang 1994] B-T Zhang, *Accelerated Learning by Active Example Selection*, International Journal of Neural Systems, 5(1), March 1994, pp 67-75.

[Zurada 1992a] J Zurada, *Lambda Learning Rule for Feedforward Neural Networks*, Proceedings of the IEEE International Conference in Neural Networks, San Francisco, 1992.

[Zurada 1992b] JM Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, 1992.

[Zurada *et al* 1994] JM Zurada, A Malinowski, I Cloete, *Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network*, IEEE International Symposium on Circuits and Systems, London, May 30 - June 3, 1994.

[Zurada *et al* 1997] JM Zurada, A Malinowski, S Usui, *Perturbation Method for Deleting Redundant Inputs of Perceptron Networks*, Neurocomputing, Vol. 14, 1997, pp 177-193.

Appendix A

Symbols and Notation

The notation and symbols used in this thesis assume a three layer neural network (NN) architecture with one input layer, one hidden layer, and one output layer. This appendix summarizes the symbols used throughout this thesis with reference to the three layer architecture depicted by figure A.1

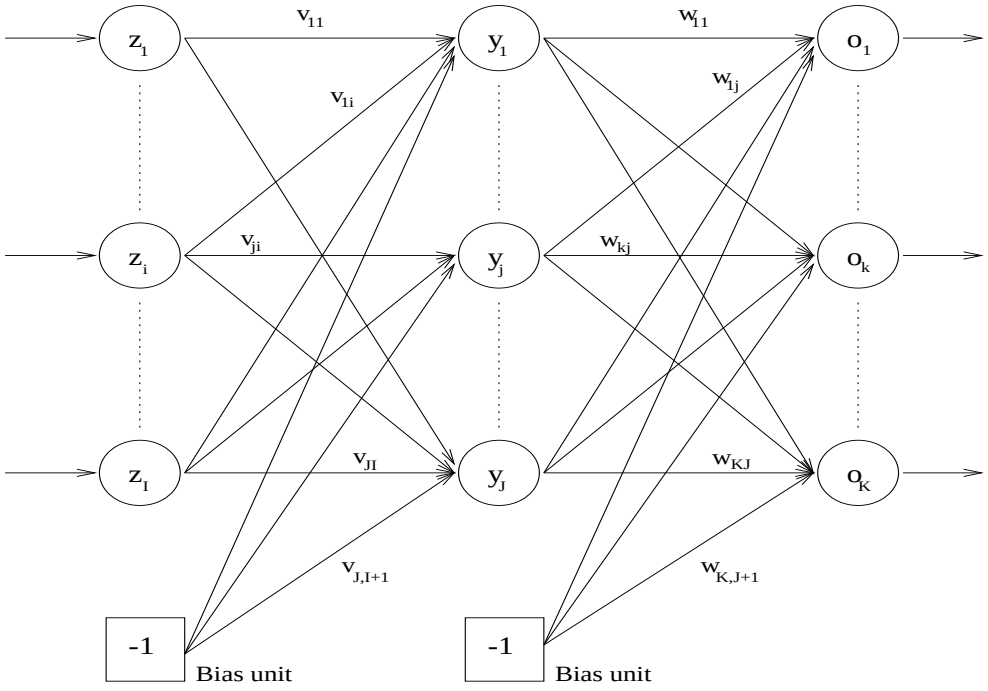


Figure A.1: Illustration of three layer NN architecture

The symbols are listed alphabetically.

Symbol	Interpretation
Φ_{θ_i}	significance of parameter θ_i
$\Phi^{(p)}$	informativeness of pattern p
$\Phi_{\infty}^{(p)}$	informativeness of pattern p using maximum norm
$\overline{\Phi}_{\infty}$	average pattern informativeness using maximum norm
Ψ	selective learning pattern selection rule
Ω	distribution of input space
Υ_{θ_i}	variance nullity measure for parameter θ_i
Υ_c	critical variance nullity value obtained from distribution tables
$\aleph_{\theta_i}^{(p)}$	absolute average sensitivity of parameter θ_i over all outputs for pattern p
$\overline{\aleph}_{\theta_i}$	average sensitivity of parameter θ_i over all patterns
α	momentum in learning context
	level of significance when used with χ^2 distribution
η	learning rate
γ	range coefficient for γ learning algorithm
λ	steepness of sigmoid activation function
	regularization parameter in case of penalty objective functions
ρ	Röbel's generalization factor
ρ_j	$\rho_j = \sum_{i=1}^{I} v_{ji} \ln \lvert z_i \rvert$
ϕ_j	$\phi_j = \sum_{i=1}^{I} v_{ji} \mathcal{I}_i$
σ	standard deviation
σ^2	variance
$\sigma_{\theta_i}^2$	variance in parameter sensitivity of parameter θ_i
σ_0^2	zero variance, a value close to zero
ζ	denotes noise
θ	any NN parameter
$\Delta\theta$	a small perturbation of θ
τ	denotes a subset selection interval
ξ	denotes an epoch

Symbol	Interpretation
$\mathcal{A}^+$	incremental learning operator
$\mathcal{A}^-$	selective learning operator
$\mathcal{C}$	denotes a computational cost as number of calculations
$\mathcal{E}_G$	error on test set / generalization error
$\mathcal{E}_T$	training error
$\mathcal{E}_V$	validation error
$\overline{\mathcal{E}}_V$	average validation error
$\mathcal{F}_{NN}$	NN output function
$\mathcal{H}$	denotes hypothesis space
$\mathcal{H}_0$	the null hypothesis
$\mathcal{H}_1$	the alternative hypothesis
$\mathcal{P}$	a performance measure
$\mathcal{S}$	subset size function
$\mathcal{T}$	number of pattern presentations during learning
C	pruning gap constant
D_C	set of candidate training patterns
D_G	test set / generalization set
D_{S_s}	subset at s^{th} subset selection interval
D_T	actual training set
D_V	validation set
I	total number of input units
J	total number of hidden units
K	total number of output units
$N(a, b)$	normal distribution with mean a and variance b
P_C	number of candidate training patterns
P_G	number of test / generalization patterns
P_{S_s}	number of patterns in subset D_{S_s}

Symbol	Interpretation
P_T	number of patterns in actual training set
P_V	number of validation patterns
S	total number of subset selection intervals
$\vec{S}_o^{(p)}$	output sensitivity vector for pattern p
$\vec{S}_o$	output sensitivity vector as a norm over training set
$S_{oz}^{(p)}$	output-input sensitivity matrix for pattern p
$S_{o,k}^{(p)}$	sensitivity of output unit o_k for pattern p
$S_{o\theta,ki}^{(p)}$	the sensitivity of output o_k to perturbations in parameter θ_i for pattern p
$S_{o\theta,ki}$	the sensitivity of output o_k to perturbations in parameter θ_i as a norm over all training patterns
$S_{v,ji}^{(p)}$	the sensitivity of output o_k to perturbations in weight v_{ji} for pattern p
$S_{w,kj}^{(p)}$	the sensitivity of output o_k to perturbations in weight w_{kj} for pattern p
$S_{oy,kj}^{(p)}$	the sensitivity of output o_k to perturbations in hidden unit y_j for pattern p
$S_{oz,ki}^{(p)}$	the sensitivity of output o_k to perturbations in input z_i for pattern p
$U(-a,a)$	uniform distribution of values $[-a,a]$
f_{o_k}	activation function of k^{th} output unit
f'_{o_k}	derivative of activation function of k^{th} output unit
f_{y_j}	activation function of j^{th} hidden unit
f'_{y_j}	derivative of activation function of j^{th} hidden unit
i	index for input units when used with z
	general parameter index when used with θ
j	index for hidden units
k	index for output units
net_{o_k}	netto input to k^{th} output unit
net_{y_j}	netto input to j^{th} hidden unit
o_k	the k^{th} output unit

Symbol	**Interpretation**
p	pattern index
s	subset interval index
t_k	target value for k^{th} output unit
u_{ki}	direct weight between output o_k and input z_i
v_{ji}	weight between hidden y_j and input z_i
w_{kj}	weight between output o_k and hidden y_j
y_j	the j^{th} hidden unit
z_i	the i^{th} input unit

Appendix B

Definitions

This appendix summarizes definitions of key terms used in this thesis. The terms are defined in alphabetical order.

Active Learning: Active learning is any form of learning in which the learning algorithm has some deterministic control during training over what part of the input space it receives information (page 60).

Active Learning Operator: An active learning operator is a mechanism/algorithm used to dynamically select patterns during training from a candidate training set (page 67).

Architecture Selection: Architecture selection is the process of selecting a suitable NN architecture through regularization, pruning or growing (network construction) (page 143).

Axis-parallel Decision Boundary: Under the assumption that a NN implements a nonlinear differentiable function, and that monotonic increasing activation functions are used in the hidden and output layers, if there exists an input parameter value $z_i^{(p)}$ and a small perturbation Δz_i of $z_i^{(p)}$ such that, for any output unit o_k, $f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) \neq f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{(p)})$, then an axis-parallel decision boundary is located in the range $[z_i^{(p)}, z_i^{(p)} + \Delta z_i]$ of input parameter z_i (page 38).

Non-axis-parallel Decision Boundary: Under the assumption that a NN implements a nonlinear differentiable function, and that monotonic increasing activation functions are used in the hidden and output layers, if there exist two input parameter values $z_{i_1}^{(p)}$ and $z_{i_2}^{(p)}$ with $z_{i_1}^{(p)} < z_{i_2}^{(p)}$ such that for $z_i^{(p)} \in [z_{i_1}^{(p)}, z_{i_2}^{(p)}]$, a small perturbation Δz_i of $z_i^{(p)}$ and any output unit o_k, $f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)}, \cdots, z_I^{(p)}) \neq f_{o_k}(z_1^{(p)}, \cdots, z_i^{(p)} + \Delta z_i, \cdots, z_I^{(p)})$, then a non-axis-parallel decision boundary spans over the range $[z_{i_1}^{(p)}, z_{i_2}^{(p)}]$ of input parameter z_i (page 38).

Decision Boundary: A decision boundary is a region in input space of maximum ambiguity in classification. A decision boundary forms a separation between two different classes (page 33).

Dynamic Learning: Refer to the definition of active learning.

Fixed Set Learning: In fixed set learning, the NN learner passively receives a fixed set of information to learn (page 59).

Generalization Factor: The generalization factor as defined by Röbel is the ratio of the validation set error versus the training set error, used as measure of overfitting (page 64).

Model Selection: Model selection is the process of designing an optimal NN architecture and the optimal selection of training patterns from a pool of candidate patterns (page 54).

Incremental Learning: Incremental learning is an active learning algorithm where the training set is incrementally grown from a candidate set, by dynamically selecting and removing patterns from the candidate set and adding them to the training set (page 57).

Objective Function Sensitivity Analysis: A study of the influence that small parameter perturbations have on the function being optimized (usually the sum squared error) by means of the derivatives of the objective function with respect to the perturbed parameters is referred to as objective function sensitivity analysis (page 17).

Output Sensitivity Analysis: A study of the influence that small parameter perturbations have on the NN output function being approximated by means of the derivatives of the

NN output function with respect to the perturbed parameters is referred to as output sensitivity analysis (page 20).

Mean Squared Error: In the context of neural networks, the mean squared error (MSE) is defined as the mean of the squared sum of the error between target values $t_k^{(p)}$ and the actual NN output values $o_k^{(p)}$:

$$MSE = \frac{\sum_{p=1}^{P}\sum_{k=1}^{K}(t_k^{(p)} - o_k^{(p)})^2}{PK}$$

where P is the total number of patterns and K is the number of output units (page 25).

Parameter Significance: Define the significance of a NN parameter θ_i, which can be an input unit, hidden unit or weight, as the sensitivity of the NN output vector to small perturbations in that parameter (page 161).

Parameter Variance Nullity: Define the statistical nullity in the parameter sensitivity variance of a NN parameter θ_i over patterns $p = 1, \cdots, P$ as

$$\Upsilon_{\theta_i} = \frac{(P-1)\sigma_{\theta_i}^2}{\sigma_0^2}$$

where $\sigma_{\theta_i}^2$ is the variance of the sensitivity of the network output to perturbations in parameter θ_i, and σ_0^2 is a value close to zero (page 163).

Pattern Informativeness: The informativeness of a pattern is defined as the sensitivity of the NN output vector to small perturbations in the input vector. Pattern informativeness quantifies the amount of information a pattern has about the problem being learned (page 74).

Perturbation Analysis: Perturbation analysis is the study of the influence that small perturbations $\Delta\theta$ of system parameters θ has on a performance function $\mathcal{P}$. The change in performance due to perturbations is described by the Taylor series (page 14)

$$\mathcal{P}(\theta + \Delta\theta) = \mathcal{P}(\theta) + \frac{\Delta\theta}{1!}\mathcal{P}'(\theta) + \frac{\Delta\theta^2}{2!}\mathcal{P}''(\theta) + \cdots$$

Selective Learning: Selective learning is an active learning algorithm for classification problems which starts training on the entire candidate set and removes patterns that become uninformative during training (page 57).

Sensitivity Analysis: Sensitivity analysis is the study of the influence that small parameter perturbations have on the result of a performance function $\mathcal{P}$ through calculation of the derivatives of the performance function with respect to the perturbed parameters (page 14).

Sensitivity Analysis Incremental Learning: Sensitivity analysis incremental learning is an approach to active learning where pattern informativeness is used as measure to select patterns from a candidate training set that convey the most information on the function being approximated. The most informative patterns are removed from the candidate training set and added to the current training subset (page 109).

Sensitivity Analysis Selective Learning: Sensitivity analysis selective learning is an approach to active learning where pattern informativeness is used as measure to select patterns for learning which are located close to decision boundaries. Selective learning prunes uninformative information from the training set (page 76).

Training Strategy: A training strategy is viewed as preprocessing of the training set to determine an order according to which patterns will be presented to the network, without using the knowledge of the learner (page 60).

Appendix C

Automatic Scaling using γ Learning

This appendix is a reproduction of a paper published in the proceedings of the International Workshop on Artificial Neural Networks, 1995 [Engelbrecht *et al* 1995a]. The research presented is a study of the effects of the scaling of output values on training time and generalization, and the development of an automatic scaling algorithm.

C.1 Abstract

Standard error back-propagation requires output data that is scaled to lie within the active area of the activation function. We show that normalizing data to conform to this requirement is not only a time-consuming process, but can also introduce inaccuracies in modeling of the data. In this paper we propose the gamma learning rule for feedforward neural networks which eliminates the need to scale output data before training. We show that the utilization of "self-scaling" units results in faster convergence and more accurate results compared to the rescaled results of standard back-propagation.

C.2 Introduction

Many artificial neural networks trained with the popular error back propagating training algorithm, also called the *delta rule*, contain units having the well-known sigmoid activation

function where λ is a positive constant [Zurada 1992a]:

$$f(\lambda, y) = \frac{1}{1 + e^{-\lambda y}} \tag{C.1}$$

A problem with this squashing function is that its output is always in the range $[0, 1]$, thus requiring scaling of the desired output before training to fit into this range. In addition to scaling of the output data, the input data is normally scaled to lie within the active area of the sigmoid activation function (e.g. to the range $[-\sqrt{3}, \sqrt{3}]$). In this paper we investigate the effects that scaling of the output data has on the learning process.

In practice, the data set presented to a neural network often contains values which lie outside the active range of the sigmoid activation function. If the delta learning rule with sigmoid activation functions is used to learn the data set, the data must be pre-processed before training. During pre-processing, the data is compressed to fit into the active range of the sigmoid function. This scaled data set is then used for training purposes. To interpret the results obtained from the neural network, the outputs must be rescaled to the original range. From the user's viewpoint the accuracy obtained by the neural network refers to this rescaled data set. We show that the scaling of outputs into a smaller range than the original unscaled range leads to longer training times to reach a specified accuracy on the rescaled data.

In this paper we extend the delta rule to the so-called *gamma learning rule* which adjusts the output range of the sigmoid activation function during learning. Thus, the gamma rule is effectively performing automatic scaling – a property applicable to almost all applications. Recently, Zurada proposed the *lambda learning rule* where the constant λ in (C.1) is treated as a variable and also adapted during training [Zurada 1992a]. Thimm *et al* has shown that there exists a relationship between the gain λ and learning rate η, as well as initial weights [Thimm *et al* 1996]. The same reduced training times achieved by lambda learning can be obtained through appropriate scaling of the learning rate. In our research we develop gamma learning as a mechanism to automatically scale the range of the sigmoid activation functions, with the purpose to investigate the effect of scaling on training time and error. A similar relationship between gamma values and learning rate does not exist. We denote the combination of lambda and gamma learning as the *lambda-gamma learning rule*, which is more general than the delta rule.

The gamma rule is reminiscent of biological neurons which are able to adjust to signals of various natures through transmitter depletion and contrast enhancement. For instance, cells in the auditory system exhibit "stimulus selectivity" [Morgan *et al* 1991], becoming attuned to a characteristic frequency.

In the next section we investigate the effects of scaling of the output data, and show the advantages of self-scaling output units. The lambda-gamma rule is derived for a single neuron in section C.4, and extended to single layer learning in section C.5. Section C.6 generalizes the rule for hidden layer learning. A complete general learning algorithm is presented in section C.7, and experimental results are reported in section C.8.

C.3 Effects of scaling

For the purpose of this exposition assume an output layer which consists of one neuron[1]. Without loss of generality, assume that the desired output data is scaled into the range $[0, 1]$ using linear scaling:

$$t_s = c_1 t_u + c_2 \tag{C.2}$$

where t_u is the original unscaled desired output data (i.e. raw data), and t_s is the corresponding scaled desired output data to be used for training. To scale data to the range $[0, 1]$ the scaling factors c_1 and c_2 are the following:

$$c_1 = \frac{1}{\max_{p=1,\ldots,P}\{t_u^{(p)}\} - \min_{p=1,\ldots,P}\{t_u^{(p)}\}} \qquad c_2 = \frac{-\min_{p=1,\ldots,P}\{t_u^{(p)}\}}{\max_{p=1,\ldots,P}\{t_u^{(p)}\} - \min_{p=1,\ldots,P}\{t_u^{(p)}\}} \tag{C.3}$$

with P the total number of patterns. Then, from (C.2) the rescaled desired output t_r is

$$t_r = \frac{1}{c_1} t_s - \frac{c_2}{c_1} \tag{C.4}$$

Let o_s denote the actual output of output neuron o. Then, similarly to (C.4), o_r is the actual output rescaled to the original output range. Assume it is possible to learn original unscaled data, and let o_u denote the actual unscaled output of neuron o. Since the values of desired output data are not changed during training, it is clear that $t_u = t_r$, and under ideal conditions we will also have that $o_u = o_r$. However, this will require perfect learning with

[1] The derivations in this section can easily be extrapolated to an output layer with more than one neuron.

zero error which is in practice not realizable. Let MSE_s and MSE_r respectively denote the mean square error for the scaled and rescaled data over the entire training set. Then,

$$MSE_s = \sum_{p=1}^{P}(t_s^{(p)} - o_s^{(p)})^2/P \tag{C.5}$$

$$MSE_r = \sum_{p=1}^{P}(t_r^{(p)} - o_r^{(p)})^2/P \tag{C.6}$$

By substitution of equation (C.4) in (C.6) we obtain

$$MSE_r = \sum_{p=1}^{P}[(\frac{1}{c_1}t_s^{(p)} - \frac{c_2}{c_1}) - (\frac{1}{c_1}o_s^{(p)} - \frac{c_2}{c_1})]^2/P = (\frac{1}{c_1})^2 MSE_s \tag{C.7}$$

Equation (C.7) illustrates a clear relation between the scaled and rescaled error. If

$$|c_1| < 1 \qquad (c_1 \neq 0) \tag{C.8}$$

then (C.7) indicates that the rescaled error is a factor of $(\frac{1}{c_1})^2$ larger than the scaled error, where condition (C.8) corresponds to the compression of data into a smaller range than the original range.

For the following, assume it is possible to learn the original unscaled data. Let MSE_u denote the mean square error for the unscaled data. The relationship illustrated above indicates that in order to obtain a rescaled accuracy MSE_r which is equal to MSE_u, the network must be trained longer until

$$MSE_s = (c_1)^2 MSE_r \tag{C.9}$$

On the other hand, if

$$|c_1| > 1 \tag{C.10}$$

we have from (C.7) that the rescaled error is a factor of $(\frac{1}{c_1})^2$ smaller than the scaled error. This corresponds to our claim that training on data which is expanded over a wider range will lead to faster convergence, since a scaling factor c_1 which conforms to condition (C.10) represents the scaling of data to a larger range than the original unscaled range.

From this investigation into the effects of scaling we conclude that it is preferable to use "self-scaling" output units and to learn original unscaled data when the range is greater than $[0, 1]$. This will significantly decrease the number of training cycles compared to learning

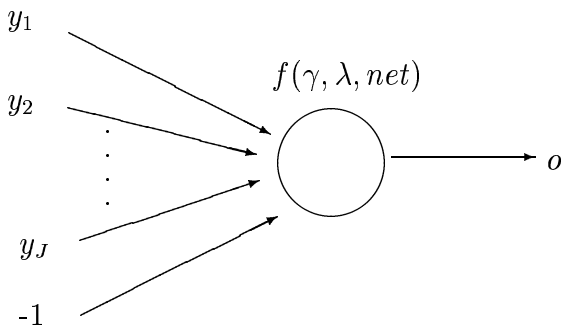


Figure C.1: Lambda-gamma learning for a single neuron

scaled data, especially when $|c_1|$ is very small. Currently only linear self-scaling output units are available. In the next sections we propose the use of sigmoid self-scaling output units where the output range of the sigmoid activation function is dynamically adapted to span the original output range. The online adjustment of the output range enables the learning of unscaled data.

C.4 Lambda-gamma single neuron learning

The customary sigmoid function (C.1) for a single neuron is modified to include a *range coefficient* γ and a *steepness coefficient* λ, to give:

$$f(\gamma, \lambda, net) = \frac{\gamma}{1 + e^{-\lambda net}} \tag{C.11}$$

where the activation value is $net = \vec{w}^t \vec{y}$, the augmented input vector is $\vec{y} = [y_1 \; y_2 \; \ldots \; y_{n-1} \; -1]^t$ and the weight vector is $\vec{w} = [w_1 \; w_2 \; \ldots \; w_n]^t$. In the classical delta rule the neuron therefore learns in $(n-1)$-dimensional non-augmented weight space in which n weights are adjustable. The lambda and gamma learning rules expand the learning space to $(n+1)$ dimensions, while the lambda-gamma learning rule expands it to $(n+2)$ dimensions. In addition to weight learning, both the steepness λ and the range γ undergo adjustments in the negative gradient direction. Referring to Figure C.1 and using the customary expression for error between the desired value t and the actual output of the neuron o,

$$E(\gamma, \lambda, \vec{w}) = \frac{1}{2}[t - o(\gamma, \lambda, \vec{w})]^2$$

where

$$o(\gamma, \lambda, \vec{w}) = f(\gamma, \lambda, net(\vec{w}, \vec{y}))$$

we obtain the following weight adjustments for single neuron learning:

$$\Delta w_j = -\eta_1 \frac{\partial E}{\partial w_j} = -\eta 1 \frac{\partial E}{\partial o} \frac{\partial o}{\partial net} \frac{\partial net}{\partial w_j} = \eta_1 (t - o) \frac{\lambda}{\gamma} o(\gamma - o) y_j \tag{C.12}$$

$$\Delta \lambda = -\eta_2 \frac{\partial E}{\partial \lambda} = -\eta_2 \frac{\partial E}{\partial o} \frac{\partial o}{\partial \lambda} = \eta_2 (t - o) \frac{1}{\gamma} o(\gamma - o) net \tag{C.13}$$

$$\Delta \gamma = -\eta_3 \frac{\partial E}{\partial \gamma} = -\eta_3 \frac{\partial E}{\partial o} \frac{\partial o}{\partial \gamma} = \eta_3 (t - o) \frac{1}{\gamma} o \tag{C.14}$$

where η_1, η_2 and η_3 are positive learning constants usually selected as arbitrarily small values. Inspection of expression (C.12) coincides with the delta learning rule [Zurada 1992b] when $\lambda = \gamma = 1$, and the lambda learning rule [Zurada 1992a] when $\gamma = 1$. Expression (C.13) similarly reduces to the lambda rule when $\gamma = 1$. The extension to the lambda-gamma learning rule where a neuron's activation value can be "self-scaling" to the desired range is represented by expression (C.14).

The next section illustrates single layer learning for the lambda-gamma learning rule.

C.5 Single layer learning

Assume that the neurons in the output layer $\vec{o}$ undergo training. In addition to a net input and activation value, each neuron o_k $(k = 1, \ldots, K)$ has a range coefficient γ_{o_k} and a steepness coefficient λ_{o_k}. The range and steepness coefficients are trained along with the weights w_{kj} for all hidden neurons y_j $(j = 1, \ldots, J)$ shown as the rightmost two layers in Figure A.1. The usual definitions for error and error signal terms are used:

$$E = \frac{1}{2} \sum_{k=1}^{K} [t_k - o_k(\gamma_{o_k}, \lambda_{o_k}, net_{o_k})]^2 \tag{C.15}$$

$$\delta_{o_k} = \frac{\partial E}{\partial net_{o_k}} = -\frac{\lambda_{o_k}}{\gamma_{o_k}} (t_k - o_k) o_k (\gamma_{o_k} - o_k) \tag{C.16}$$

where $\vec{d}$ and $\vec{o}$ are respectively the desired output and the actual output vectors, and $\vec{\delta}_o$ and $\vec{\delta}_y$ are respectively the error signal term vectors for the output and hidden layers. The

adjustments to the learning variables are given below:

$$\Delta w_{kj} = -\eta_1 \frac{\partial E}{\partial w_{kj}} = -\eta_1 \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial w_{kj}} = -\eta_1 \delta_{o_k} y_j \tag{C.17}$$

$$\Delta \lambda_{o_k} = -\eta_2 \frac{\partial E}{\partial \lambda_{o_k}} = -\eta_2 \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial \lambda_{o_k}} = -\eta_2 \delta_{o_k} \frac{net_{o_k}}{\lambda_{o_k}} \tag{C.18}$$

$$\Delta \gamma_{o_k} = -\eta_3 \frac{\partial E}{\partial \gamma_{o_k}} = -\eta_3 \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial \gamma_{o_k}} = \eta_3 (t_k - o_k) \frac{1}{\gamma_{o_k}} o_k \tag{C.19}$$

where

$$o_k = f(\gamma_{o_k}, \lambda_{o_k}, net_{o_k}(\vec{w}_k, \vec{y})) \quad \text{and} \quad net_{o_k}(\vec{w}_k, \vec{y}) = \sum_{j=1}^{J} w_{kj} y_j$$

Hidden layer learning for the lambda-gamma learning rule is described in the next section.

C.6 Hidden layer learning

To train the neurons y_j $(j = 1, \ldots, J)$ in the hidden layer, the weights v_{ji} $(i = 1, \ldots, I)$, the steepness coefficient λ_{y_j} and the range coefficient γ_{y_j} must be adjusted during each iteration of the learning algorithm, using respectively

$$\Delta v_{ji} = -\eta_1 \frac{\partial E}{\partial v_{ji}} \tag{C.20}$$

$$\Delta \lambda_{y_j} = -\eta_2 \frac{\partial E}{\partial \lambda_{y_j}} \tag{C.21}$$

$$\Delta \gamma_{y_j} = -\eta_3 \frac{\partial E}{\partial \gamma_{y_j}} \tag{C.22}$$

Using the error (C.15) and error signal term

$$\delta_{y_j} = \frac{\partial E}{\partial net_{y_j}} = \frac{\lambda_{y_j}}{\gamma_{y_j}} (\gamma_{y_j} - y_j) y_j \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{C.23}$$

we obtain

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial net_{y_j}} \frac{\partial net_{y_j}}{\partial v_{ji}} = \delta_{y_j} z_i \tag{C.24}$$

$$\frac{\partial E}{\partial \lambda_{y_j}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \lambda_{y_j}} = \delta_{y_j} \frac{net_{y_j}}{\lambda_{y_j}} \tag{C.25}$$

$$\frac{\partial E}{\partial \gamma_{y_j}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \gamma_{y_j}} = \frac{1}{\gamma_{y_j}} f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{C.26}$$

where we have from (C.15)

$$\frac{\partial E}{\partial y_j} = \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{C.27}$$

Substitution of equations (C.24), (C.25) and (C.26) into equations (C.20), (C.21) and (C.22) respectively yields the following adjustments for the hidden layer neurons:

$$\begin{aligned}
\Delta v_{ji} &= -\eta_1 \delta_{y_j} z_i \\
\Delta \lambda_{y_j} &= -\eta_2 \delta_{y_j} \frac{net_{y_j}}{\lambda_{y_j}} \\
\Delta \gamma_{y_j} &= -\eta_3 \frac{1}{\gamma_{y_j}} f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj}
\end{aligned}$$

where

$$y_j = f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}(\vec{v}_j, \vec{z})) \quad \text{and} \quad net_{y_j}(\vec{v}_j, \vec{z}) = \sum_{i=1}^{I} v_{ji} z_i$$

with δ_{o_k} and δ_{y_j} respectively the error signal term of the k-th output neuron and the j-th hidden neuron by equations (C.16) and (C.23). As mentioned previously, the adjustments reduce to that of the delta rule when λ and γ are constants equal to 1, the lambda rule when γ is equal to 1 and the gamma rule when λ is equal to 1. For training, the complete back-propagation algorithm in [Zurada 1992a] is updated to reflect the changes given above. The updated algorithm is presented in the next section.

C.7 Complete lambda-gamma learning algorithm

The algorithm presented in [Zurada 1992a] is modified below to reflect the lambda-gamma learning rule which is more general than the delta and lambda learning rules. Changes correspond to the adjustments to weights, steepness and range coefficients.

Begin: Given P training pairs of vectors of inputs and desired outputs $\{(\vec{z}_1, \vec{d}_1), (\vec{z}_2, \vec{d}_2), \ldots, (\vec{z}_p, \vec{d}_p)\}$ where $\vec{z}_i$ is $(I \times 1)$, $\vec{d}_i$ is $(K \times 1)$ and $i = 1, \ldots, P$; $\vec{y}$ is $(J \times 1)$ and $\vec{o}$ is $(K \times 1)$.

Step 1: Choose the values of the learning rates η_1, η_2 and η_3 according to the learning rule:

Delta learning rule	$\eta_1 > 0,\ \eta_2 = 0,\ \eta_3 = 0$
Lambda learning rule	$\eta_1 > 0,\ \eta_2 > 0,\ \eta_3 = 0$
Gamma learning rule	$\eta_1 > 0,\ \eta_2 = 0,\ \eta_3 > 0$
Lambda-gamma learning rule	$\eta_1 > 0,\ \eta_2 > 0,\ \eta_3 > 0$

Choose an acceptable training error E_{max}. Weights W $(K \times J)$ and V $(J \times I)$ are initialized to small random values. Initialize the number of cycles q and the training pairs counter p to $q = 1,\ p = 1$. Let $E = 0$ and initialize the steepness and range coefficients

$$\lambda_{y_j} = \gamma_{y_j} = 1 \quad \forall\, j = 1, \ldots, J \quad \text{and} \quad \lambda_{o_k} = \gamma_{o_k} = 1 \quad \forall\, k = 1, \ldots, K$$

Step 2: Start training. Input is presented and the layers' outputs are computed using $f(\gamma, \lambda, net)$ as in equation (C.11):

$$\vec{z} = \vec{z}_p, \quad \vec{t} = \vec{t}_p \quad \text{and} \quad y_j = f(\gamma_{y_j}, \lambda_{y_j}, \vec{v}_j^t \vec{z}) \quad \forall\, j = 1, \ldots, J$$

where $\vec{v}_j$, a column vector, is the j-th row of V and

$$o_k = f(\gamma_{o_k}, \lambda_{o_k}, \vec{w}_k^t \vec{y}) \quad \forall\, k = 1, \ldots, K$$

where $\vec{w}_k$, a column vector, is the k-th row of W.

Step 3: The error value is computed:

$$E = E + \frac{1}{2}(t_k - o_k)^2 \quad \forall\, k = 1, \ldots, K$$

Step 4: The error signal vectors $\vec{\delta}_o$ $(K \times 1)$ and $\vec{\delta}_y$ $(J \times 1)$ of both the output and hidden layers are computed

$$\delta_{o_k} = -\frac{\lambda_{o_k}}{\gamma_{o_k}}(t_k - o_k) o_k (\gamma_{o_k} - o_k) \quad \forall\, k = 1, \ldots, K$$

$$\delta_{y_j} = \frac{\lambda_{y_j}}{\gamma_{y_j}} y_j (\gamma_{y_j} - y_j) \sum_{k=1}^{K} \delta_{o_k} w_{kj} \quad \forall\, j = 1, \ldots . J$$

Step 5: Output layer weights and gains are adjusted:

$$w_{kj} = w_{kj} + \eta_1 \delta_{o_k} y_j \qquad \lambda_{o_k} = \lambda_{o_k} + \eta_2 \delta_{o_k} \frac{net_{o_k}}{\lambda_{o_k}} \qquad \gamma_{o_k} = \gamma_{o_k} + \eta_3 (t_k - o_k) \frac{1}{\gamma_{o_k}} o_k$$

for all $k = 1, \ldots, K$ and $j = 1, \ldots, J$.

Step 6: Hidden layer weights and gains are adjusted:

$$v_{ji} = v_{ji} + \eta_1 \delta_{y_j} z_i \qquad \lambda_{y_j} = \lambda_{y_j} + \eta_2 \frac{1}{\lambda_{y_j}} \delta_{y_j} net_{y_j} \qquad \gamma_{y_j} = \gamma_{y_j} + \eta_3 \frac{1}{\gamma_{y_j}} f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj}$$

for all $j = 1, \ldots, J$ and $i = 1, \ldots, I$.

Step 7: If $p < P$ then let $p = p + 1$ and go to Step 2; otherwise go to Step 8.

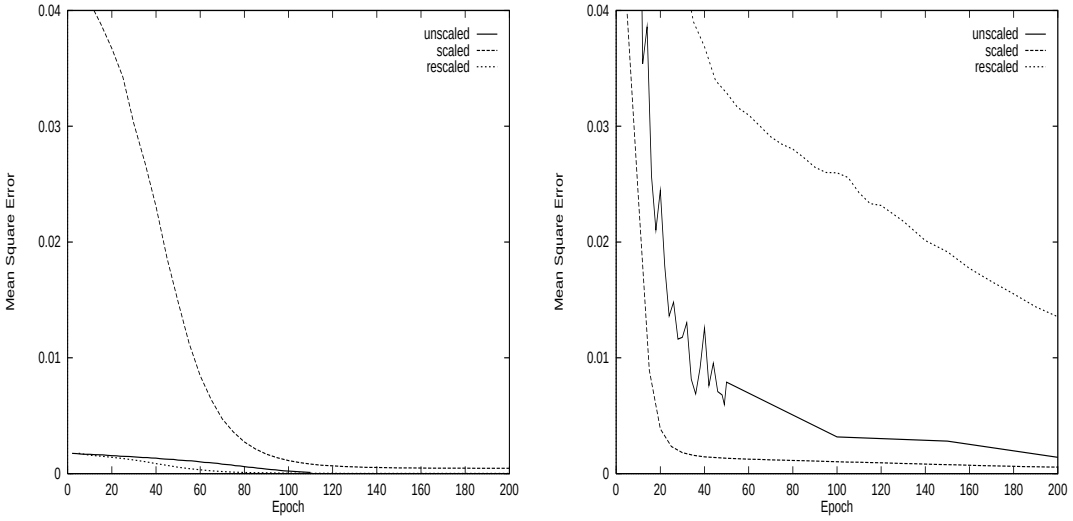
Step 8: One training cycle is completed. If $E < E_{max}$ then terminate the training session. Output the cycle counter q and error E; otherwise let $E = 0,\ p = 1,\ q = q + 1$ and initiate a new training cycle by going to Step 2.

C.8 Experimental results

We have used a simple function approximation experiment to substantiate our claims to the effects of scaling. A 1-10-1 network architecture was used to approximate the function $f(z) = |z|$. For the experiments described below, we have used the lambda-gamma learning algorithm to train on original unscaled data, and the delta learning algorithm to train on the scaled data. For illustration purposes, Figure C.2 also shows the learning profile on the rescaled output data. The mean square error MSE_r on the rescaled output data is calculated form equations (C.4) and (C.6) after each epoch. Both experiments use the same initial weights, which are initialized as random values in the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$.

- **Experiment 1:** For this experiment we have $z \in [0.4, 0.6]$, and $t \in [0.4, 0.6]$. The desired outputs t are linearly scaled to $[0, 1]$ using (C.2). From (C.3) we have $|c_1| = 5$, which illustrates the effect when output data is scaled to a larger range than the original. Figure C.2(a) shows that the mean square error for the rescaled data is smaller than the mean square error of the scaled data for each epoch when $|c_1| > 1$. For example, from Figure C.2(a) we see that a required error of 0.0005 on the scaled data has already been reached at epoch 55 on the rescaled data compared to epoch 150 on the scaled data.

- **Experiment 2:** For this experiment we have $z \in [-5, 5]$ and $t \in [0, 5]$. The desired outputs t are linearly scaled to $[0, 1]$ using (C.2). Then, from (C.3) we have $|c_1| = \frac{1}{5}$ which corresponds to the compression of data. From Figure C.2(b) we observe that an



(a) Experiment 1

(b) Experiment 2

Figure C.2: Learning profiles for unscaled, scaled and rescaled data.

	Experiment 1			Experiment 2		
Epoch	MSE_s	MSE_r	$(\frac{1}{c_1})^2 MSE_s$	MSE_s	MSE_r	$(\frac{1}{c_1})^2 MSE_s$
5	0.043914	0.001757	0.0017566	0.037712	0.94279	0.9428
50	0.013873	0.000555	0.0005549	0.001315	0.03288	0.032875
100	0.001086	0.00043	0.00004344	0.001039	0.025983	0.025975
150	0.000489	0.00002	0.00001956	0.000766	0.01915	0.01915
200	0.000455	0.000018	0.0000182	0.000542	0.013551	0.01355

Table C.1: Comparison of MSE_s with MSE_r.

error of 0.02, which is reached at epoch 20 using lambda-gamma learning on unscaled data, is reached at epoch 140 on the rescaled data. Longer training is therefore required on scaled data to obtain a specified error equivalent on the original data.

Figure C.2 also shows the learning profile for the lambda-gamma rule on unscaled data. Table C.1 shows that condition (C.7) holds for arbitrarily selected epochs: for any given epoch, the error MSE_r on the rescaled data is a factor of $(\frac{1}{c_1})^2$ larger than the error MSE_s on the scaled data when $|c_1| < 1$, and a factor $(\frac{1}{c_1})^2$ smaller when $|c_1| > 1$.

The results presented in this section confirm our conclusion that training on output data that is scaled into a smaller range causes longer training times to reach a required accuracy on the rescaled output data (training accuracy is normally specified in terms of the rescaled data). The problem escalates as $|c_1|$ becomes very small. With the lambda-gamma learning rule, the same accuracy is obtained in less training cycles.

C.9 Conclusions

We have derived a relationship between the mean square errors for scaled and rescaled data when output data is linearly scaled. This relationship has indicated that the compression of data causes longer training times compared to training on the unscaled data. We have presented the lambda-gamma learning algorithm which utilizes self-scaling sigmoid output units.

In order to perform scaling, the maximum and minimum ranges of the input and output must be known. For incremental learning systems this is difficult to obtain, since all training pairs are not available before training. Upper and lower bounds need to be determined beforehand. Gamma learning eliminates this problem since the output range of the sigmoid activation function is dynamically adjusted during training. The lambda-gamma learning rule further seems to eliminate the need for internal rescaling within the units as reported by Rigler, Irvine and Vogl [Rigler *et al* 1991].

Appendix D

Gradient Descent Learning Equations

Complete learning equations for the feedforward NN type used in this thesis are derived in this appendix. The derivations presented are for gradient descent optimization, using on-line learning. The notations and symbols listed in appendix A are used throughout this appendix, assuming a three layer NN architecture (one input, one hidden and one output layer, with a bias unit in the input and hidden layers) with sigmoid activation functions.

D.1 Feedforward Neural Network

The MSE objective function, and sigmoid activation functions in both the hidden and output layers are assumed.

Objective function:

$$E = \frac{\sum_{p=1}^{P} E^{(p)}}{P} \tag{D.1}$$

where P is the total number of patterns in the training set, and $E^{(p)}$ is the error of pattern p, defined as

$$E^{(p)} = \frac{1}{2}\frac{\sum_{k=1}^{K}(t_k^{(p)} - o_k^{(p)})}{K} \tag{D.2}$$

where K is the number of output units, $t_k^{(p)}$ and $o_k^{(p)}$ are respectively the target and actual

output values of the k-th output unit.

The rest of the derivations refer to individual patterns. The pattern superscript (p) is therefore omitted for notational convenience.

The output of the k-th output unit is

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-net_{o_k}}} \tag{D.3}$$

where

$$net_{o_k} = \sum_{j=1}^{J+1} w_{kj} y_j \tag{D.4}$$

J is the number of hidden units; the $(J+1)^{th}$ unit represents the bias to each output unit; w_{kj} is the weight between the j-th hidden and k-th output units; y_j is the output of the j-th hidden unit, defined as

$$y_j = f_{y_j}(net_{y_j}) = \frac{1}{1 + e^{-net_{y_j}}} \tag{D.5}$$

with

$$net_{y_j} = \sum_{i=1}^{I+1} v_{ji} z_i \tag{D.6}$$

I is the number of input units; the $(I+1)^{th}$ unit represents the bias to each hidden unit; v_{ji} is the weight between the i-th input and hidden j-th hidden units; z_i is the value of the i-th input unit.

Weights are updated according to the following equations:

$$w_{kj}(t) = \Delta w_{kj}(t) + \alpha w_{kj}(t-1) \tag{D.7}$$

$$v_{ji}(t) = \Delta v_{ji}(t) + \alpha v_{ji}(t-1) \tag{D.8}$$

where α is the momentum.

In the rest of this section the equations for calculating $\Delta w_{kj}(t)$ and $\Delta v_{ji}(t)$ are derived. The reference to time, t, is omitted for notational convenience.

From (D.3),

$$\frac{\partial o_k}{\partial net_{o_k}} = \frac{\partial f_{o_k}}{\partial net_{o_k}} = (1 - o_k) o_k = f'_{o_k} \tag{D.9}$$

From (D.4),

$$\frac{\partial net_{o_k}}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} (\sum_{j=1}^{J+1} w_{kj} y_j) = y_j \tag{D.10}$$

From (D.9), (D.10),

$$\begin{aligned}\frac{\partial o_k}{\partial w_{kj}} &= \frac{\partial o_k}{\partial net_{o_k}}\frac{\partial net_{o_k}}{\partial w_{kj}} \\ &= (1-o_k)o_k y_j \\ &= f'_{o_k} y_j \end{aligned} \tag{D.11}$$

From (D.2),

$$\frac{\partial E}{\partial o_k} = \frac{\partial}{\partial o_k}(\frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2) = -(t_k - o_k) \tag{D.12}$$

Define the output error that needs to be back-propagated as $\delta_{o_k} = \frac{\partial E}{\partial net_{o_k}}$. Then, from (D.12) and (D.9),

$$\begin{aligned}\delta_{o_k} &= \frac{\partial E}{\partial net_{o_k}} \\ &= \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_{o_k}} \\ &= -(t_k - o_k)(1-o_k)o_k = -(t_k - o_k)f'_{o_k} \end{aligned} \tag{D.13}$$

Then, the changes in the hidden-to-output weights are computed from (D.12), (D.11) and (D.13),

$$\begin{aligned}\Delta w_{kj} &= \eta(-\frac{\partial E}{\partial w_{kj}}) \\ &= -\eta\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}} \\ &= -\eta\delta_{o_k} y_j \end{aligned} \tag{D.14}$$

Continuing with the input-to-hidden weights, from (D.5),

$$\frac{\partial y_j}{\partial net_{y_j}} = \frac{\partial f_{y_j}}{\partial net_{y_j}} = (1-y_j)y_j = f'_{y_j} \tag{D.15}$$

From (D.6),

$$\frac{\partial net_{y_j}}{\partial v_{ji}} = \frac{\partial}{\partial v_{ji}}(\sum_{i=1}^{I+1} v_{ji} z_i) = z_i \tag{D.16}$$

From (D.15) and (D.16),

$$\begin{aligned}\frac{\partial y_j}{\partial v_{ji}} &= \frac{\partial y_j}{\partial net_{y_j}}\frac{\partial net_{y_j}}{\partial v_{ji}} \\ &= (1-y_j)y_j z_i = f'_{y_j} z_i \end{aligned} \tag{D.17}$$

From (D.4),

$$\frac{\partial net_{o_k}}{\partial y_j} = \frac{\partial}{\partial y_j}(\sum_{j=1}^{J+1} w_{kj} y_j) = w_{kj} \tag{D.18}$$

From (D.13) and (D.18),

$$\begin{aligned}
\frac{\partial E}{\partial y_j} &= \frac{\partial}{\partial y_j}(\frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2) \\
&= \sum_{k=1}^{K} \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_{o_k}}\frac{\partial net_{o_k}}{\partial y_j} \\
&= \sum_{k=1}^{K} \frac{\partial E}{\partial net_{o_k}}\frac{\partial net_{o_k}}{\partial y_j} \\
&= \sum_{k=1}^{K} \delta_{o_k} w_{kj}
\end{aligned} \tag{D.19}$$

Define the hidden layer error which needs to be back-propagated from (D.19) and (D.15),

$$\begin{aligned}
\delta_{y_j} &= \frac{\partial E}{\partial net_{y_j}} \\
&= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_{y_j}} \\
&= \sum_{k=1}^{K} \delta_{o_k} w_{kj} f'_{y_j}
\end{aligned} \tag{D.20}$$

Finally, the changes to input-to-hidden weights are calculated from (D.19), (D.17) and (D.20),

$$\begin{aligned}
\Delta v_{ji} &= \eta(-\frac{\partial E}{\partial v_{ji}}) \\
&= -\eta\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}} \\
&= -\eta\delta_{y_j} z_i
\end{aligned} \tag{D.21}$$

If direct weights from the input to the output layer are included, the following additional weight updates are needed:

$$\begin{aligned}
\Delta u_{ki} &= \eta(-\frac{\partial E}{\partial u_{ki}}) \\
&= -\eta\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial u_{ki}} \\
&= -\eta\delta_{o_k} z_i
\end{aligned} \tag{D.22}$$

where u_{ki} is a weight from the i-th input unit to the k-th output unit.

Appendix E

Sensitivity Analysis Derivations

Complete sensitivity analysis derivations are presented in this appendix for the feedforward and product unit NN types, with differentiable activation functions. Unless otherwise specified, this appendix assumes a three layer architecture with one input, one hidden and one output layer, with a bias unit in the input and hidden layers (as illustrated in figure A.1). The notations and symbols listed in appendix A are used throughout. The sensitivity analysis presented in the first part of this appendix is that of the output layer with regard to NN parameters, for example input units, hidden units and weights. The last part of the appendix gives the equations for the second-order sensitivity analysis of the objective function with regard to NN parameters. The reader should note the independence of output sensitivity analysis to the objective function. For the purposes of this exposition, sigmoidal hidden and output activation functions are assumed. Extension to other differentiable activation functions is straightforward. Sensitivity analysis equations for FFNN and PUNN are given in sections E.1.1 and E.1.2 respectively. For each network type, sensitivity analysis of the output with regard to each layer and weight matrix is presented.

E.1 Output Sensitivity Analysis

E.1.1 Feedforward Neural Network

This section starts by assuming an unrestricted number of hidden layers, and presents a general sensitivity analysis formulation using matrix and vector notation as presented by Engelbrecht and Cloete [Engelbrecht *et al* 1996]. A complete derivation of the different sensitivity analysis types are then presented with reference to individual matrix and vector elements, using the notation in Engelbrecht and Cloete [Cloete *et al* 1994c, Engelbrecht *et al* 1995b] and Zurada, Malinowski and Cloete [Zurada *et al* 1994].

Consider a FFNN which consists of the $H+1$ layers $L_0, L_1, \ldots, L_H$, where L_0 and L_H respectively represent the input and output layers. Let the weight matrix between layer L_h and L_{h+1} be denoted by W_h. Without loss of generality, assume that the sensitivity of layer L_l with respect to layer L_h needs to be computed, with $l > h$. The sensitivity matrix $S_{lh}^{(p)}$ defines the sensitivity of the units in layer L_l to small perturbations in the units of layer L_h, for a specific training pattern p:

$$\begin{aligned} S_{lh}^{(p)} &= \frac{\partial O_l}{\partial O_h} \\ &= O_l' W_{l-1} O_{l-1}' W_{l-2} \cdots O_{h+1}' W_h \\ &= \prod_{m=h}^{l-1} O_{m+1}' W_m \end{aligned} \tag{E.1}$$

where O_m' is the diagonal matrix

$$O_m' \doteq \text{diag}(o_{1m}', o_{2m}', \cdots, o_{Nm}') \tag{E.2}$$

with o_{nm}' the derivative of the output of unit n in layer L_m. The product $O_{m+1}' W_m$ refers to matrix multiplication. In the case of output-input layer sensitivity analysis, the dimension of the resulting sensitivity matrix $S_{H0}^{(p)}$ for pattern p will be (number of outputs $\times$ number of inputs).

The derivations presented in the rest of this appendix refer to a single pattern. The parameter superscript p is therefore omitted for notational convenience. This section continues by defining equations for the sensitivity analysis of output units with respect to input units,

hidden units and weights with reference to individual parameters. For notational purposes, a three layer network is assumed.

Output-Input layer analysis

Let $S_{oz,ki} = \frac{\partial o_k}{\partial z_i}$ be the sensitivity of output unit o_k with respect to input unit z_i for a single pattern. The first part of the subscript indicates the layers involved (o for the output layer, and z for the input layer), and the second part indicate the respective unit of each layer. The following derives equations for the calculation of $S_{oz,ki}$.

From (D.6),

$$\frac{\partial net_{y_j}}{\partial z_i} = \frac{\partial}{\partial z_i}(\sum_{i=1}^{I+1} v_{ji} z_i) = v_{ji} \tag{E.3}$$

From (D.15) and (E.3),

$$\frac{\partial y_j}{\partial z_i} = \frac{\partial y_j}{\partial net_{y_j}} \frac{\partial net_{y_j}}{\partial z_i} = (1 - y_j) y_j v_{ji} \tag{E.4}$$

From (D.4) and (E.4),

$$\begin{aligned} \frac{\partial net_{o_k}}{\partial z_i} &= \frac{\partial}{\partial z_i}(\sum_{j=1}^{J+1} w_{kj} y_j) \\ &= \sum_{j=1}^{J} \frac{\partial net_{o_k}}{\partial y_j} \frac{\partial y_j}{\partial z_i} \\ &= \sum_{j=1}^{J} w_{kj}(1 - y_j) y_j v_{ji} \end{aligned} \tag{E.5}$$

Then, from (D.9) and (E.5), the sensitivity of output o_k to perturbations in input z_i is defined as

$$\begin{aligned} S_{oz,ki} &= \frac{\partial o_k}{\partial z_i} = \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial z_i} \\ &= (1 - o_k) o_k \sum_{j=1}^{J} w_{kj}(1 - y_j) y_j v_{ji} \\ &= f'_{o_k} \sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji} \end{aligned} \tag{E.6}$$

Output-Hidden layer analysis

Denote by $S_{oy,kj} = \frac{\partial o_k}{\partial y_j}$ the sensitivity of output unit o_k to small perturbations in hidden unit y_j for a single pattern. Then, from (D.9) and (D.18),

$$\begin{aligned} S_{oy,kj} &= \frac{\partial o_k}{\partial y_j} = \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial y_j} \\ &= (1 - o_k) o_k w_{kj} \\ &= f'_{o_k} w_{kj} \end{aligned} \tag{E.7}$$

Output-Weights analysis

First consider the sensitivity of an output unit o_k to perturbations in a weight w_{kj} between hidden unit y_j and output unit o_k. From (D.11),

$$\begin{aligned} S_{w,kj} &= \frac{\partial o_k}{\partial w_{kj}} = \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial w_{kj}} \\ &= (1 - o_k) o_k y_j \\ &= f'_{o_k} y_j \end{aligned} \tag{E.8}$$

where the first part of the subscript, w, indicates the weights between the hidden and output layers.

Similarly, the sensitivity of output o_k to perturbations in the weight v_{ji} between input unit z_i and hidden unit y_j is defined from (E.7) and (D.17) as

$$\begin{aligned} S_{v,ji} &= \frac{\partial o_k}{\partial v_{ji}} = \frac{\partial o_k}{\partial y_j} \frac{\partial y_j}{\partial v_{ji}} \\ &= (1 - o_k) o_k w_{kj} (1 - y_j) y_j z_i \\ &= f'_{o_k} w_{kj} f'_{y_j} z_i \end{aligned} \tag{E.9}$$

where the v part of the subscript indicates the weights between the input and hidden layers.

Hidden-Input analysis

Denote by $S_{yz,ji} = \frac{\partial y_j}{\partial z_i}$ the sensitivity of hidden unit y_j to small perturbations in input unit z_i for a single pattern. Then,

$$\begin{aligned} S_{yz,ji} &= \frac{\partial y_j}{\partial z_i} = \frac{\partial y_j}{\partial net_{y_j}} \frac{\partial net_{y_j}}{\partial z_i} \\ &= (1 - y_j) y_j v_{ji} \\ &= f'_{y_j} v_{ji} \end{aligned} \tag{E.10}$$

Direct Input-to-Output connections

If the NN also has direct input to output connections, all the sensitivity analysis equations derived previously remain the same, except for the output-input analysis. In this case equation (E.6) becomes

$$S_{oz,ki} = f'_{o_k} \sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji} + u_{ki} \tag{E.11}$$

where u_{ki} is the direct weight between input z_i and output o_k. Additionally, sensitivity analysis with respect to a direct weight is formulated as

$$S_{u,ki} = \frac{\partial o_k}{\partial u_{ki}} = z_i \tag{E.12}$$

E.1.2 Product Unit Neural Network

This section develops equations for the sensitivity analysis of product unit neural networks (PUNN). The sensitivity of output units with respect to input units and weights, and the sensitivity of hidden units with respect to input units are derived. The sensitivity equations not repeated here are the same as for the FFNN presented in section E.1.1. For the purpose of this exposition, it is assumed that there are product units in the single hidden layer only (considering a three-layer architecture), and that linear activation functions are used in the hidden layer.

For each hidden unit y_j, the netto input to that hidden unit is

$$net_{y_j} = \prod_{i=1}^{I} z_i^{v_{ji}} \tag{E.13}$$

Output-Input layer analysis

From equation (E.5),

$$\frac{\partial net_{o_k}}{\partial z_i} = \sum_{j=1}^{J} \frac{\partial net_{o_k}}{\partial y_j} \frac{\partial y_j}{\partial z_i} \tag{E.14}$$

where

$$\frac{\partial y_j}{\partial z_i} = \frac{\partial y_j}{\partial net_{y_j}} \frac{\partial net_{y_j}}{\partial z_i} \tag{E.15}$$

Because linear activation functions are used in the hidden layer,

$$\frac{\partial y_j}{\partial net_{y_j}} = 1 \tag{E.16}$$

Equation (E.13) is rewritten as

$$\begin{aligned} net_{y_j} &= \prod_{i=1}^{I} z_i^{v_{ji}} \\ &= \prod_{i=1}^{I} e^{v_{ji} \ln(z_i)} \\ &= e^{\sum_i v_{ji} \ln(z_i)} \end{aligned} \tag{E.17}$$

If $z_i < 0$, then z_i can be written as the complex number $z_i = \imath^2 |z_i|$ which, substituted in (E.17), yields

$$net_{y_j} = e^{\sum_i v_{ji} \ln |z_i|} e^{\sum_i v_{ji} \ln \imath^2} \tag{E.18}$$

Let $c = 0 + \imath = a + b\imath$ be a complex number representing $\imath$. Then,

$$\ln c = \ln r e^{\imath\theta} = \ln r + \imath\theta + 2\pi k\imath \tag{E.19}$$

where $r = \sqrt{a^2 + b^2} = 1$.

Considering only the main argument, $arg(c)$, $k = 0$ which implies that $2\pi k\imath = 0$. Furthermore, $\theta = \frac{\pi}{2}$ for $\imath = (0, 1)$. Therefore, $\imath\theta = \imath\frac{\pi}{2}$, which simplifies equation (E.19) to $\ln c = \imath\frac{\pi}{2}$, and consequently,

$$\ln \imath^2 = \imath\pi \tag{E.20}$$

Substitution of (E.20) in (E.18) gives

$$\begin{aligned} net_{y_j} &= e^{\sum_i v_{ji} \ln |z_i|} e^{\sum_i v_{ji} \pi \imath} \\ &= e^{\sum_i v_{ji} \ln |z_i|} [\cos(\sum_i v_{ji}\pi) + \imath \sin(\sum_i v_{ji}\pi)] \end{aligned} \tag{E.21}$$

The imaginary part of (E.21) is omitted, since Durbin and Rumelhart empirically discovered that no substantial improvements were achieved by including the imaginary part [Durbin *et al* 1989]. Equation (E.21) is therefore simplyfied as:

$$net_{y_j} = e^{\sum_i v_{ji} \ln |z_i|} \cos(\pi \sum_i v_{ji})$$

Now, let

$$\rho_j = \sum_{i=1}^{I} v_{ji} \ln |z_i| \tag{E.22}$$

$$\phi_j = \sum_{i=1}^{I} v_{ji} \mathcal{I}_i \tag{E.23}$$

with

$$\mathcal{I}_i = \begin{cases} 0 & \text{if } z_i > 0 \\ 1 & \text{if } z_i < 0 \end{cases} \tag{E.24}$$

and $z_i \neq 0$.

Then,

$$net_{y_j} = e^{\rho_j} \cos(\pi \phi_j) \tag{E.25}$$

From (E.25),

$$\begin{aligned} \frac{\partial net_{y_j}}{\partial z_i} &= \frac{\partial}{\partial z_i}(e^{\rho_j} \cos(\pi \phi_j)) \\ &= \frac{v_{ji}}{|z_i|} e^{\rho_j} \cos(\pi \phi_j) \end{aligned} \tag{E.26}$$

Substitution of (E.26) in (E.15), and from (E.6),

$$\frac{\partial o_k}{\partial z_i} = f'_{o_k} \sum_{j=1}^{J} w_{kj} \frac{v_{ji}}{|z_i|} e^{\rho_j} \cos(\pi \phi_j) \tag{E.27}$$

Input-Hidden analysis

From (E.16) and (E.26),

$$\begin{aligned} \frac{\partial y_j}{\partial z_i} &= \frac{\partial y_j}{\partial net_{y_j}} \frac{\partial net_{y_j}}{\partial z_i} \\ &= \frac{v_{ji}}{|z_i|} e^{\rho_j} \cos(\pi \phi_j) \end{aligned} \tag{E.28}$$

Output-Weights analysis

Only the equation to calculate the sensitivity of the outputs to changes in the weights between the input and hidden layer changes. From (E.9),

$$\begin{aligned} \frac{\partial o_k}{\partial v_{ji}} &= \frac{\partial o_k}{\partial y_j}\frac{\partial y_j}{\partial net_{y_j}}\frac{\partial net_{y_j}}{\partial v_{ji}} \\ &= f'_{o_k} w_{kj} \frac{v_{ji}}{|z_i|} e^{\rho_j} \cos(\pi\phi_j) \end{aligned} \tag{E.29}$$

E.2 Objective Function Sensitivity Analysis

This section gives complete derivations of the sensitivity of the mean squared error (MSE) objective function with respect to NN parameters. Only three layer FFNNs are considered, and sigmoid activation functions are assumed. Sensitivity analysis of the objective function with regard to each layer and weight matrix is presented, giving the equations to compute $\frac{\partial^2 \mathcal{E}}{\partial \theta^2}$, with θ a NN parameter. For the equations below, the reader should note the dependence of this sensitivity analysis approach on the objective function.

Define the objective function as in equation (D.1). Then,

$$\frac{\partial^2 E}{\partial \theta^2} = \frac{1}{2P}\sum_{k=1}^{K} \frac{\partial^2 E^{(p)}}{\partial \theta^2} \tag{E.30}$$

For the rest of this section, the error $E^{(p)}$ for one pattern (p) is considered. For notational convenience the pattern superscript p is omitted.

Error-Input layer analysis

From equation (D.2),

$$\begin{aligned} \frac{\partial^2 E}{\partial z_i^2} &= \frac{\partial}{\partial z_i}\left(\frac{\partial E}{\partial z_i}\right) \\ &= \frac{\partial}{\partial z_i}\left(\sum_{k=1}^{K} \frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial z_i}\right) \\ &= \sum_{k=1}^{K}\left[\frac{\partial E_k}{\partial o_k}\frac{\partial^2 o_k}{\partial z_i^2} + \frac{\partial o_k}{\partial z_i}\frac{\partial^2 E_k}{\partial z_i \partial o_k}\right] \end{aligned} \tag{E.31}$$

From (E.6),

$$\begin{aligned}\frac{\partial^2 o_k}{\partial z_i^2} &= \frac{\partial}{\partial z_i}(\frac{\partial o_k}{\partial z_i}) \\ &= \frac{\partial}{\partial z_i}[(1-o_k)o_k\sum_{j=1}^{J} w_{kj}(1-y_j)y_j v_{ji}] \end{aligned} \tag{E.32}$$

Let $f = (1-o_k)o_k$ and $g = \sum_{j=1}^{J} w_{kj}(1-y_j)y_j v_{ji} = \sum_{j=1}^{J} h_j$. Then, equation (E.32) becomes

$$\frac{\partial^2 o_k}{\partial z_i^2} = f\frac{\partial g}{\partial z_i} + g\frac{\partial f}{\partial z_i} \tag{E.33}$$

But, from (E.6)

$$\begin{aligned}\frac{\partial f}{\partial z_i} &= \frac{\partial f}{\partial o_k}\frac{\partial o_k}{\partial z_i} \\ &= (1-2o_k)(1-o_k)o_k\sum_{j=1}^{J} w_{kj}(1-y_j)y_j v_{ji} \\ &= f''_{o_k}\sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji} \end{aligned} \tag{E.34}$$

and

$$\begin{aligned}\frac{\partial g}{\partial z_i} &= \sum_{j=1}^{J}\frac{\partial h_j}{\partial z_i} \\ &= \sum_{j=1}^{J}\frac{\partial h_j}{\partial y_j}\frac{\partial y_j}{\partial z_i} \\ &= \sum_{j=1}^{J} w_{kj}(1-2y_j)v_{ji}(1-y_j)y_j v_{ji} \\ &= \sum_{j=1}^{J} w_{kj} f''_{y_j} v_{ji}^2 \end{aligned} \tag{E.35}$$

Substitute (E.34) and (E.35) in (E.33) to get

$$\frac{\partial^2 o_k}{\partial z_i^2} = f'_{o_k}\sum_{j=1}^{J} w_{kj} f''_{y_j} v_{ji}^2 + f''_{o_k}[\sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji}]^2 \tag{E.36}$$

Now, let $f = o_k - t_k$. Then, from (D.12) and (E.6)

$$\frac{\partial^2 E_k}{\partial z_i \partial o_k} = \frac{\partial f}{\partial o_k}\frac{\partial o_k}{\partial z_i} = f'_{o_k}\sum_{j=1}^{J} w_{kj} f'_{y_j} v_{ji} \tag{E.37}$$

Substitute (E.36) and (E.37) in (E.31) and simplify:

$$\begin{aligned}\frac{\partial^2 E}{\partial z_i^2} &= \sum_{k=1}^{K}[-(t_k-o_k)(f'_{y_j}\sum_{j=1}^{J} w_{kj}f''_{y_j}v_{ji}^2 + f''_{o_k}[\sum_{j=1}^{J} w_{kj}f'_{y_j}v_{ji}]^2) + [f'_{o_k}\sum_{j=1}^{J} w_{kj}f'_{y_j}v_{ji}]^2] \\ &= \sum_{k=1}^{K}[-(t_k-o_k)f'_{o_k}\sum_{j=1}^{J} w_{kj}f'_{y_j}v_{ji}^2 + [\sum_{j=1}^{J} w_{kj}f'_{y_j}v_{ji}]^2((f'_{o_k})^2-(t_k-o_k)f''_{o_k})] \quad \text{(E.38)}\end{aligned}$$

Using the Levenberg-Marquardt assumption (refer to section 2.4.1), equation (E.38) becomes

$$\frac{\partial^2 E}{\partial z_i^2} \approx (f'_{o_k})^2[\sum_{j=1}^{J} w_{kj}f'_{y_j}v_{ji}]^2 \quad \text{(E.39)}$$

Error-Hidden layer analysis

From equation (D.2),

$$\begin{aligned}\frac{\partial^2 E}{\partial y_j^2} &= \frac{\partial}{\partial y_j}(\frac{\partial E}{\partial y_j}) \\ &= \sum_{k=1}^{K}\frac{\partial}{\partial y_j}(\frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial y_j}) \\ &= \sum_{k=1}^{K}[\frac{\partial E_k}{\partial o_k}\frac{\partial^2 o_k}{\partial y_j^2} + \frac{\partial o_k}{\partial y_j}\frac{\partial^2 E_k}{\partial y_j \partial o_k}] \quad \text{(E.40)}\end{aligned}$$

Let $a = (1-o_k)o_k w_{kj}$. Then, from (E.7)

$$\begin{aligned}\frac{\partial a}{\partial y_j} &= \frac{\partial a}{\partial o_k}\frac{\partial o_k}{\partial y_j} \\ &= (1-2o_k)(1-o_k)o_k w_{kj}^2 = f''_{o_k}w_{kj}^2 \quad \text{(E.41)}\end{aligned}$$

Then,

$$\frac{\partial^2 o_k}{\partial y_j^2} = \frac{\partial}{\partial y_j}(\frac{\partial o_k}{\partial y_j}) = \frac{\partial a}{\partial y_j} = f''_{o_k}w_{kj}^2 \quad \text{(E.42)}$$

Now, let $a = o_k - t_k$. Then, from (D.12) and (E.7)

$$\begin{aligned}\frac{\partial^2 E_k}{\partial y_j \partial o_k} &= \frac{\partial}{\partial y_j}(\frac{\partial E_k}{\partial o_k}) \\ &= \frac{\partial a}{\partial y_j} \\ &= \frac{\partial a}{\partial o_k}\frac{\partial o_k}{\partial y_j} \\ &= (1-o_k)o_k w_{kj} = f'_{o_k}w_{kj} \quad \text{(E.43)}\end{aligned}$$

Substitute (E.42) and (E.43) in (E.40) to get

$$\begin{aligned}\frac{\partial^2 E}{\partial y_j^2} &= \sum_{k=1}^{K}[-(t_k - o_k)f''_{o_k}w_{kj}^2 + (f'_{o_k}w_{kj})^2] \\ &= \sum_{k=1}^{K}[((f'_{o_k})^2 - (t_k - o_k)f''_{o_k})w_{kj}^2] \end{aligned} \tag{E.44}$$

After application of the Levenberg-Marquardt assumption, equation (E.44) becomes

$$\frac{\partial^2 E}{\partial y_j^2} \approx \sum_{k=1}^{K}[f'_{o_k}w_{kj}]^2 \tag{E.45}$$

Error-Weights analysis

First consider the hidden-to-output weights, and compute the sensitivity of the objective function with respect to perturbations from (D.2):

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{kj}^2} &= \frac{\partial}{\partial w_{kj}}(\frac{\partial E_k}{\partial w_{kj}}) \\ &= \frac{\partial}{\partial w_{kj}}(\frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}}) \\ &= \frac{\partial E_k}{\partial o_k}\frac{\partial^2 o_k}{\partial w_{kj}^2} + \frac{\partial^2 E_k}{\partial w_{kj}\partial o_k}\frac{\partial o_k}{\partial w_{kj}} \end{aligned} \tag{E.46}$$

From (E.37),

$$\begin{aligned}\frac{\partial^2 o_k}{\partial w_{kj}} &= \frac{\partial}{\partial w_{kj}}(\frac{\partial E_k}{\partial w_{kj}}) \\ &= \frac{\partial}{\partial w_{kj}}(\frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}}) \\ &= \frac{\partial E_k}{\partial o_k}\frac{\partial^2 o_k}{\partial w_{kj}^2} + \frac{\partial^2 E_k}{\partial w_{kj}\partial o_k}\frac{\partial o_k}{\partial w_{kj}} \end{aligned} \tag{E.47}$$

From (E.8),

$$\begin{aligned}\frac{\partial^2 o_k}{\partial w_{kj}} &= \frac{\partial}{\partial w_{kj}}(\frac{\partial o_k}{\partial w_{kj}}) \\ &= (1 - 2o_k)(1 - o_k)o_k y_j^2 \\ &= f''_{o_k} y_j^2 \end{aligned} \tag{E.48}$$

If $a = o_k - t_k$, then from (D.12) and (E.8)

$$\begin{aligned}\frac{\partial^2 E_k}{\partial w_{kj}\partial o_k} &= \frac{\partial a}{\partial w_{kj}} \\ &= \frac{\partial a}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}} \\ &= (1-o_k)o_k y_j = f'_{o_k} y_j \end{aligned} \tag{E.49}$$

Substitute (E.48) and (E.49) in (E.46):

$$\frac{\partial^2 E}{\partial w_{kj}^2} = [(f'_{o_k})^2 - (t_k - o_k) f''_{o_k}] y_j^2 \tag{E.50}$$

and after applying the Levenberg-Marquardt assumption,

$$\frac{\partial^2 E}{\partial w_{kj}^2} \approx [f'_{o_k} y_j]^2 \tag{E.51}$$

Next, consider the sensitivity of the objective function to perturbations in the input-hidden layer weights:

$$\begin{aligned}\frac{\partial^2 E}{\partial v_{ji}^2} &= \frac{\partial}{\partial v_{ji}}(\frac{\partial E}{\partial v_{ji}}) \\ &= \frac{\partial}{\partial y_i}(\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}}) \\ &= \frac{\partial E}{\partial y_j}\frac{\partial^2 y_j}{\partial v_{ji}^2} + \frac{\partial y_j}{\partial v_{ji}}\frac{\partial^2 E}{\partial v_{ji}\partial y_j} \end{aligned} \tag{E.52}$$

Let $a = (1-y_j)y_j z_i$. Then, from equation (D.17),

$$\begin{aligned}\frac{\partial^2 y_j}{\partial v_{ji}^2} &= \frac{\partial}{\partial v_{ji}}(\frac{\partial y_j}{\partial v_{ji}}) \\ &= \frac{\partial a}{\partial v_{ji}} = \frac{\partial a}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}} \\ &= (1-2y_j)(1-y_j)y_j z_i^2 = f''_{y_j} z_i^2 \end{aligned} \tag{E.53}$$

Now, let $a_k = o_k - t_k$ and $b_k = (1-o_k)o_k w_{kj}$. Then, from equation (D.19),

$$\begin{aligned}\frac{\partial^2 E}{\partial v_{ji}\partial y_j} &= \frac{\partial}{\partial v_{ji}}(\frac{\partial E}{\partial y_j}) \\ &= \sum_{k=1}^{K}\frac{\partial}{\partial v_{ji}}(a_k b_k) \\ &= \sum_{k=1}^{K}[a_k\frac{\partial b_k}{\partial v_{ji}} + g_k\frac{\partial f_k}{\partial v_{ji}}] \end{aligned} \tag{E.54}$$

From (E.9),

$$\begin{aligned}\frac{\partial b_k}{\partial v_{ji}} &= \frac{\partial b_k}{\partial o_k}\frac{\partial o_k}{\partial v_{ji}} \\ &= (1-2o_k)(1-o_k)o_k w_{kj}^2(1-y_j)y_j z_i \\ &= f_{o_k}^{''} w_{kj}^2 f_{y_j}^{'} z_i \end{aligned} \tag{E.55}$$

and

$$\begin{aligned}\frac{\partial a_k}{\partial v_{ji}} &= \frac{\partial a_k}{\partial o_k}\frac{\partial o_k}{\partial v_{ji}} \\ &= (1-o_k)o_k w_{kj}(1-y_j)y_j z_i \\ &= f_{o_k}^{'} w_{kj} f_{y_j}^{'} z_i \end{aligned} \tag{E.56}$$

Substitute (E.55) and (E.56) in (E.54) and simplify to get

$$\frac{\partial^2 E}{\partial v_{ji}\partial y_j} = f_{y_j}^{'} z_i \sum_{k=1}^{K}[(f_{o_k}^{'})^2 - (t_k - o_k)f_{o_k}^{''}]w_{kj}^2 \tag{E.57}$$

Now substitute (D.19), (D.17), (E.53) and (E.57) in (E.52) and simplify:

$$\begin{aligned}\frac{\partial^2 E}{\partial v_{ji}^2} &= [\sum_{k=1}^{K} -(t_k - o_k)f_{o_k}^{'} w_{kj}]f_{y_j}^{''} z_i^2 + (f_{y_j}^{'} z_i)^2 \sum_{k=1}^{K}[(f_{o_k}^{'})^2 - (t_k - o_k)f_{o_k}^{''}]w_{kj}^2 \\ &= (f_{y_j}^{'})^2[\sum_{k=1}^{K}[(f_{o_k}^{'})^2 - (t_k - o_k)f_{o_k}^{''}]w_{kj}^2 - (t_k - o_k)f_{o_k}^{'} w_{kj} f_{y_j}^{''}]z_i^2 \end{aligned} \tag{E.58}$$

After applying the Levenberg-Marquardt assumption, (E.58) becomes

$$\frac{\partial^2 E}{\partial v_{ji}^2} \approx (f_{y_j}^{'})^2 \sum_{k=1}^{K}(f_{o_k}^{'} w_{kj})^2 z_i^2 \tag{E.59}$$

Appendix F

Publications based on this Research

F.1 Accepted Papers

The following papers related to this study have been authored or co-authored by the author of this thesis:

E Basson, AP Engelbrecht, *Approximation of a Function and Its Derivatives in Feedforward Neural Networks*, IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999, paper 2152.

I Cloete, AP Engelbrecht, *New Tools for Decision Support*, AMSE International Conference on Intelligent Systems, Pretoria, 1994.

AP Engelbrecht, I Cloete, J Geldenhuys, JM Zurada, *Automatic Scaling using Gamma Learning for Feedforward Neural Networks*, International Workshop on Artificial Neural Networks, Torremolinos, Spain, June 1995, in J Mira, F Sandoval (eds), "From Natural to Artificial Neural Computing" in the series 'Lecture Notes in Computer Science,' Vol 930, pp 374-381.

AP Engelbrecht, I Cloete, JM Zurada, *Determining the Significance of Input Parameters using Sensitivity Analysis*, International Workshop on Artificial Neural Networks, Torremolinos, Spain, June 1995, in J Mira, F Sandoval (eds), "From Natural to Artificial

Neural Computing" in the series 'Lecture Notes in Computer Science,' Vol 930, pp 382-388.

AP Engelbrecht, I Cloete, *A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks*, IEEE International Conference in Neural Networks, Washington, Vol 2, 1996, pp 1274-1277.

AP Engelbrecht, I Cloete, *Selective Learning using Sensitivity Analysis*, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp1150-1155.

AP Engelbrecht, I Cloete, *Feature Extraction from Feedforward Neural Networks using Sensitivity Analysis*, International Conference on Systems, Signals, Control, Computers (invited session), Durban, South Africa, Vol 2, 1998, pp 221-225.

AP Engelbrecht, HL Viktor, *Rule Improvement through Decision Boundary Detection using Sensitivity Analysis*, the International Work Conference on Neural Networks (IWANN'99), 2-4 June, Alicante: Spain, Lecture Notes in Computer Science (Volume 1607), Springer-Verlag, Berlin: Germany, pp 78-84.

AP Engelbrecht, *Sensitivity Analysis for Decision Boundaries*, Neural Processing Letters, Vol 10, 1999, pp 1-14.

AP Engelbrecht, I Cloete, *Incremental Learning using Sensitivity Analysis*, IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999, paper 380.

AP Engelbrecht, L Fletcher, I Cloete, *Variance Analysis of Sensitivity Information for Pruning Feedforward Neural Networks*, IEEE International Joint Conference on Neural Networks, Washington DC, USA, 1999, paper 379.

L Fletcher, V Katkovnik, FE Steffens, AP Engelbrecht, *Optimizing the Number of Hidden Nodes of a Feedforward Artificial Neural Network*, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, Anchorage, Alaska, 1998, pp 1608-1612.

HL Viktor, AP Engelbrecht, I Cloete, *Reduction of Symbolic Rules from Artificial Neural Networks using Sensitivity Analysis*, IEEE International Conference in Neural Networks,

Perth, 1995, pp 1788-1793.

HL Viktor, AP Engelbrecht, I Cloete, *Incorporating Rule Extraction from ANNs into a Co-operative Learning Environment*, NEURAP98, Neural Networks & Their Applications, Marseilles, France, March 1998, pp 386-391.

F.2 Submitted Papers

The following papers based on this research have been submitted for publication:

AP Engelbrecht, I Cloete, *Sensitivity Analysis for Incremental Learning*, submitted for publication.

AP Engelbrecht, L Fletcher, I CLoete, *A new Pruning Heuristic Based on Variance Analysis of Sensitivity Information*, submitted for publication.

AP Engelbrecht, *Selective Learning for Classification Problems: Experimental Results*, submitted for publication.

Index