Comparison of Bayesian learning and conjugate gradient descent training of neural networks

by

Willem Daniel Nortje

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Electronics)

in the

Faculty of Engineering

UNIVERSITY OF PRETORIA

October 2001

Summary

Keywords: neural networks, sampled optimisation, Bayesian learning, Bayesian neural networks

Neural networks are used in various fields to make predictions about the future value of a time series, or about the class membership of a given object. For the network to be effective, it needs to be trained on a set of training data combined with the expected results. Two aspects to keep in mind when considering a neural network as a solution, are the required training time and the prediction accuracy.

This research compares the classification accuracy of conjugate gradient descent neural networks and Bayesian learning neural networks. Conjugate gradient descent networks are known for their short training times, but are not very consistent and results are heavily dependant on initial training conditions. Bayesian networks are slower, but much more consistent.

The two types of neural networks are compared, and some attempts are made to combine their strong points in order to achieve shorter training times while maintaining a high classification accuracy.

Bayesian learning outperforms the gradient descent methods by almost 1%, while the hybrid method acieves results between those of Bayesian learning and gradient descent. The drawback of the hybrid method is that there is no speed improvement above that of Bayesian learning.

Samevatting

Sleutelwoorde: neurale netwerke, gemonsterde optimering, Bayes afrigting, Bayes neurale netwerke

Neurale netwerke word in verskeie velde gebruik om voorspellings oor die toekomstige waarde van 'n tydreeks te maak, of oor die klas waartoe 'n gegewe voorwerp behoort. Vir die netwerk om effektief te wees, moet dit afgerig word met 'n stel afrigtings data, tesame met die verwagte waardes. Twee aspekte om in gedagte te hou by die oorweging van 'n neurale netwerk as oplossing, is die tyd wat nodig is vir afrigting en die akkuraatheid van die voorspelling.

Hierdie navorsing vergelyk die klassifikasie akkuraatheid van toegevoegde gradiënt ('conjugate gradient descent') neurale netwerke en neurale netwerke wat met Bayes tegnieke afgerig word. CGD netwerke is bekend vir hul kort afrigtingstye, maar is nie besonder konsekwent nie, en resultate is hoogs afhanklik van aanvanklike afrigtingstoestande. Bayes netwerke is stadiger, maar baie meer konsekwent.

Hierdie twee tipes netwerke word vergelyk, en daar word gepoog om hul sterk punte te kombineer om sodoende korter afrigtingstye te verkry, terwyl hoë klassifikasie akkuraatheid behoue bly.

Die Bayes metode oortref CGD met ongeveer 1%, terwyl die hibriede metode resultate verkry wat val tussen die van Bayes en CGD. Die nadeel van die hibriede metode is dat dit geen verbetering ten opsigte van spoed inhou teenoor die Bayes metode nie.

Contents

Chapter 1

Introduction

A neural network is a mathematical tool used to predict a function's output value from a given input value. The mappings between the input values and the output values can be highly non-linear, as in cases where it is not possible to represent the output value as a simple function of the input.

Neural networks can be used to predict a single output value according to time as an input value. This is one form of regression. Examples of possible uses for neural networks predicting regression functions include: process controllers in factories, stock values on stock exchanges or the electricity consumption of a city.

Another use of neural networks is to predict the class membership of a given input vector. This is commonly known as classification. Applications of this include: optical character recognition or personal identification. In short, neural networks are mathematical tools that try to accomplish what humans continuously do without ever thinking about it.

A feed-forward neural network consists of a number of layers of neurons, that is, an input layer, zero or more hidden layers and an output layer. The hidden layers and the output layer neurons have connections, or weights, to one or more neurons in the

preceding layers. Except for the input neurons, the value of a neuron is calculated by weighing all the input values to the specific neuron with their associated weights and adding these results. This combined value is then passed through a transfer function to obtain the neuron's final value. These values are fed through the network until a value is obtained for each output neuron. No feedback exists, hence the name 'feed-forward'. In contrast to this, the input neurons simply take on the value of the corresponding input.

Obviously the output of a neural network depends heavily on the values of the network weights. In order for a network to implement the correct input to output mapping, the weights must be chosen carefully. The process of finding the best weights for the problem at hand is known as training.

Conceptually the training process can be compared to standing blindfolded on a rather big surface full of hills and valleys. The object is then to find the lowest point of the surface in as few steps as possible. The most frequently used methods of training neural networks attempt to accomplish this using numerical optimisation algorithms. Another, very recent, method is known as Bayesian learning. Bayesian learning searches for the optimal solution using statistical methods.

In this study two methods of training neural networks are compared in terms of their classification performance. As such the neural networks are only used as classifiers and not as time series predictors. The two methods compared were Bayesian learning and a form of conjugate gradient descent.

1.1 Background and related work

Since the fields of neural networks and conjugate gradient descent learning are well established by now, information on and software for training with conjugate gradient descent is freely available. On the other hand, the Bayesian learning method is still

new and literature on the subject is limited.

Gradient descent involves finding the minimum value of an error function for a specific set of training data, using the gradient of the error function The error is a function in weight space that represents an error between the actual network output and the desired network output. It is a function of the network weights and the network inputs. The idea behind the gradient descent method is that the error and the gradient of the error are calculated for each input vector to indicate in which direction the network's weight adjustment should be made. The weight adjustments are made after each iteration in such a way that the error value decreases. An iteration may be defined as after each input vector has been evaluated (called 'on-line' or stochastic training), after the complete training set has been evaluated ('batch' training) or anything in between. Although the concept is simple, the method is very computationally intensive because the gradient has to be calculated numerically. For this reason a lot of research in neural networks has been done to improve the training methods.

In this research the book *Neural Networks for Pattern Recognition* [1] by Christopher M. Bishop was used as the main reference for neural networks and training methods. It was also used as a secondary reference regarding Bayesian learning. The gradient descent training software used was written by Johann E.W. Holm during his Ph.D. studies. Dr. Holm's thesis [2] was also used as a reference for conjugate gradient descent algorithms.

Bayesian learning uses Bayesian statistics to sample the weight space for solutions. The process is as follows: select the prior probability distribution for the network weights, calculate the posterior distribution, select new weights and finally check if the network performs well enough. The first two steps and the last step of this process are straightforward and easy to calculate, however, it is the selection of new weights that makes the Bayesian learning method difficult to implement and slow to use.

Selection of new weights in itself consists of a few steps: calculate a gradient, sample weights in the determined direction and decide whether or not the weights are accept-

able. Once again, calculating the gradient is a numerical procedure. The weights are sampled in a much more efficient way than by purely moving in steps in a certain direction. A Monte Carlo method is used to determine whether the new weights are accepted or not according to some probability. This method may sometimes force a weight update with weights that are inferior to the previous weights or lead to the rejection of weights that are superior to the previous weights. Radford M. Neal's Ph.D. thesis [3] was used as the main reference regarding Bayesian learning, Markov Chain Monte Carlo sampling and the associated topics. Dr. Neal's Bayesian learning software[1] was also used to train the Bayesian networks.

1.2 Problem statement

Extensive research in the field of neural networks has focused on finding different methods for training a network using the gradient information provided by some error function, as well as minimising the time it takes to train networks using these methods. These gradient calculations and optimisation algorithms are all numerical methods that do not take the statistics of the problem into account. Despite this drawback, a number of very effective and fast algorithms have been developed. One such an algorithm, ALECO-2 [2], was used in this research.

Recently a completely different method of training neural networks has been introduced. The new method, Bayesian learning, is based on Bayesian statistics. Using Bayesian learning is relatively slow compared to the gradient methods, but it seems to be more accurate and consistent, and less influenced by poor initial conditions.

The method of Bayesian learning has been evaluated on several benchmark problems [4], and with various modifications to the training process [5]. In some studies, vague comparisons are made between the performance of Bayesian learning for neural net-

[1] The Bayesian learning software can be found on the Internet at `http://www.cs.toronto.edu/~radford/`

works and other training methods. As far as could be established, no direct comparison of classification performance or training time exists between gradient descent methods and Bayesian learning. It was the purpose of this research to compare the classification performance of the Bayesian learning method with that of a good conjugate gradient descent method. Attempts were also made to improve the classification performance of the gradient descent method using data resampling techniques, as well as testing a hybrid method which combines the strong points of the gradient descent method with those of the Bayesian learning method in order to speed up training with Bayesian learning.

1.3 Approach

The method of Bayesian learning uses statistical methods to sample the weight space for a solution to a given problem. This process means that the weight space is extensively sampled - to a much larger degree than in the case of gradient methods. In addition, results from the Bayesian networks include *variance* on each output, which may be used as a measure of confidence in the result. Taking all this into account, one may easily jump to the conclusion that Bayesian learning is superior to any gradient descent method. To determine if this is indeed the case, this research aimed to compare these two methods and to suggest possible improvements.

The first focus was on comparing the performance of Bayesian learning with the performance of conjugate gradient descent. This was done using readily available implementations of these methods. Bayesian networks were trained using software by Neal [3] and conjugate gradient descent training was done using an implementation of ALECO-2 by Holm [2].

The second part of the work consisted of trying to improve the performance of the gradient descent method by resampling the training data set and then using the created subsets to train the neural networks. A number of networks were trained on unique

subsets of the original training set with the idea that the networks should find different solutions to the same problem. These different networks were then combined in a committee in order to include the opinions of all the networks in the result, as well as to average the variance introduced by resampling the training set. Obviously, training more networks for use in a committee results in longer training times. This time penalty may be reduced by using smaller data sets to train each network. In the experiments for this comparison, networks were trained on subsets of varying size, with all the networks in a committee using the same size of subset. Also, the number of member networks were varied in the committees to determine the effect of including poor networks in a committee. A second experiment on the committees was done, with the difference that the neural networks used in the committee were overtrained on the data. This attempts to use the variance averaging effect of committees more effectively, since overtrained neural networks have much more variance than networks that use early stopping in their training.

Finally, a hybrid technique was introduced that tried to combine the performance benefit of using Bayesian learning with the training speed of conjugate gradient descent.

It was found in the trial runs that the Bayesian software always finds a near optimal solution regardless of the initial network weights. It seemed that the only penalty for choosing poor initial weights was that it took more iterations for the weights to stabilise, in effect lengthening the training time. By first using a gradient descent method, an initial set of weights might be obtained that is closer to the solution than a random set of weights. Using the weights resulting from the gradient descent step as the initial weights for the Bayesian method, the number of iterations required by Bayesian learning may be reduced, which will reduce the training time significantly.

1.4 Contribution

This research contributes mainly by giving a quantitative comparison of classification performance between the method of Bayesian learning and the conjugate gradient learning method. Further, the possibility of improving conjugate gradient descent's performance using committees and data resampling will be reported on. And lastly, the improvement or degradation of the Bayesian learning method's training time using a simple hybrid technique will be shown.

Some of the results obtained were reported in [6].

1.5 Organisation

Chapter 2 gives the theoretical background of the conjugate gradient descent training method and background on the Bayesian learning method.

Chapter 3 describes the experimental work. Firstly, the data used in experimentation is described and the derivation of the theoretical minimum classification error is given. Thereafter, the experimental protocol and the results of the experiments are provided and discussed. Some results on a benchmark problem is also provided.

Finally, in Chapter 4, a summary is given and the conclusion is drawn.

Chapter 2

Theoretical background

In this chapter the theoretical background of gradient descent methods and Bayesian learning are discussed. For both the Bayesian learning and the gradient descent methods, only the theory for feed-forward networks are discussed.

2.1 Gradient descent methods

2.1.1 Neural network structure

The neural networks most commonly used in engineering applications are multilayer perceptron networks, also known as feed-forward neural networks. These networks take an input vector of real values and produces an output vector. A typical example of a feed-forward network is shown in Figure 2.1, where the neurons labelled x_i are the input layer, the neurons labelled z_j are the hidden layer and the neurons labelled y_k are the output layer. Neurons x_0 and z_0 are the bias neurons which always have an output value of 1 [1].

It is possible to have any number of hidden layers and connections between non-

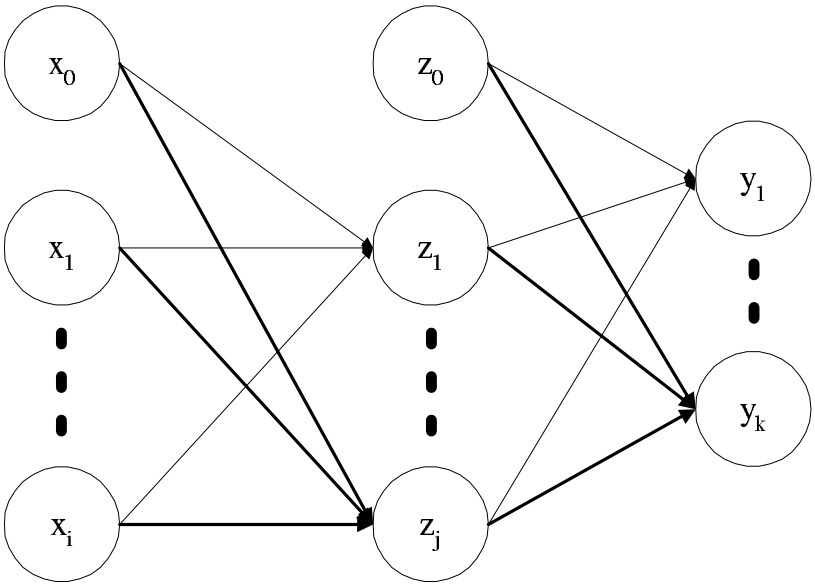

Figure 2.1: A typical feed-forward neural network structure.

consecutive layers, although this complex type of network does not necessarily result in an improvement on classification performance. Neural networks with the simple topology shown in Figure 2.1 are capable of approximating any continuous functional mapping to arbitrary accuracy [1], leaving little reason to use a complex topology in general applications. The process of calculating a neural network's output is described by the equations below.

The first step is to find the activations, a_j, of the hidden neurons. This is done using

$$a_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i \tag{2.1}$$

where:

- a_j is the activation of hidden neuron j,
- d is the dimensionality of the input vector,
- w_{ji} is the weight between input i and hidden neuron j,

- x_i is input i,
- and $x_0 = 1$.

Equation (2.2) is used to calculate the output values of the hidden layer:

$$z_j = g(a_j) \tag{2.2}$$

where:

- z_j is the output value of hidden neuron j, and
- $g(\cdot)$ is the activation (or transfer) function.

A number of possible activation functions may be used and even different functions for each neuron may be utilised by simply using the correct function at the right time. Usually though, a sigmoidal or *tanh* function is used for all the hidden neurons and sigmoidal, *tanh* or linear functions for the output layer. When using the sigmoidal or *tanh* function, the network outputs may be interpreted as probabilities, although strictly they are not.

Equation (2.3) is used to calculate the activations of the output layer neurons:

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{2.3}$$

where:

- a_k is the activation of output neuron k,
- w_{kj} is the weight from hidden neuron j to output neuron k, and

- z_j is the output of hidden neuron j,

while Equation (2.4) gives the output values of the output neurons:

$$y_k = g(a_k). \tag{2.4}$$

2.1.2 Error gradient

In order to train a neural network using a numerical optimisation method, one must know in which direction the network's weights should be modified to be able to reach a solution quickly and efficiently. It is obvious that it would be easy to determine in which direction and by what amount the weights should be modified if one knew the individual weight's influence on the network's output, however, until the introduction of error back-propagation, there was no way to find the correct direction in which the weight change should be made.

Error back-propagation is the process whereby each weight's influence on the output error, i.e. the derivative of the error in terms of the weights, can be determined. To calculate this derivative, an input vector is first presented to the neural network. These values are then fed through the network to generate an output vector. The output is compared to the required output and an error function is calculated. Although any suitable function may be used as an error function, the sum-of-squares error (Equation (2.5)) is often used, as this function performs well in a broad range of neural network applications:

$$E = \frac{1}{2}\sum_n \|\mathbf{y}(\mathbf{x}^n;\mathbf{w}) - \mathbf{t}^n\|^2 \tag{2.5}$$

where $\mathbf{t}$ is the vector of target outputs and x^n represents the n^{th} training vector. The individual errors are summed over the number of training vectors.

The second step of the error back-propagation process is to evaluate δ_k for all the output units using Equation (2.6):

$$\begin{aligned}\delta_k &\equiv \frac{\partial E^n}{\partial a_k} \\ &= \frac{\partial E^n}{\partial z_k} \cdot \frac{\partial z_k}{\partial a_k} \\ &= g'(a_k)\frac{\partial E^n}{\partial y_k}.\end{aligned} \tag{2.6}$$

Next the δ_j's are calculated using

$$\delta_j = g'(a_j)\sum_k w_{kj}\delta_k \tag{2.7}$$

for each hidden unit. In the case of a network containing multiple hidden layers, Equation (2.7) is used to calculate the δ_j's for each successive layer.

The total error gradient is obtained by using Equation (2.8)

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E^n}{\partial w_{ji}} \tag{2.8}$$

where:

- n = number of current input vector,
- E = error value,
- w = weight vector,
- E_n = error value for input vector n, and
- w_{ji} = weight vector from neuron i to neuron j.

The error gradient gives an indication of the magnitude and direction of the required weight change in order to decrease the error. This is called *steepest descent*. The more accurately the error gradient can be calculated, the better the indication of the required weight change, which in turns results in fewer changes and gradient calculations, leading to shorter training times.

2.1.3 Updating the weights

For the fixed step-by-step steepest gradient descent technique, weights are updated using Equation (2.9) when learning on-line and Equation (2.10) when training in batches:

$$\Delta w_{ji} = -\eta \delta_j x_i \tag{2.9}$$

$$\Delta w_{ji} = -\eta \sum_n \delta_j^n x_i^n. \tag{2.10}$$

2.1.4 Sampled optimisation

As mentioned earlier, better error gradient estimations can be beneficial, and from Equation (2.8) one sees that by using more input vectors in each gradient calculation we get better estimations. The down side to using more input vectors per weight update is that the gradient has to be calculated for each input vector used. In the training process, it is the gradient calculations that take the longest. Using multiple input vectors quickly lengthens the training time to intolerable levels.

When the complete training set is used to calculate the error gradient, it is known as *batch* training. This technique is the better option when training times are not crucial and storage space is not a problem.

For the cases where either training times should be short or storage space is limited, stochastic training should be used. Stochastic training, also known as *on-line* training, is when only one input vector is used to calculate the error gradient before each weight update. A typical system that would use stochastic training is one where the network weights are updated as data becomes available. The main advantage of stochastic training is that the input vector may be discarded as soon as the error gradient has been calculated. This implies that little storage space is needed. Another advantage of this method is that it is a lot faster than batch training.

On the other hand, the calculated gradient is at best an approximation of the true gradient, which is heavily dependant on the number of vectors used to calculate it. Furthermore, the gradient approximation quickly deteriorates as the number of possible classes increase, because this makes a single input vector an even worse representation of the true distribution of the training data. Poor gradient approximations usually cause oscillations around the optimal solution, resulting in either long training times or sub-optimal solutions.

Batch training is slow, but achieves good results because it is more consistent when used for gradient calculations. Stochastic training is a lot faster, but does not perform as well, because it is inconsistent in its gradient calculations.

Pure stochastic training may be modified by using multiple input vectors for each update. This method, termed sampled optimisation [7], involves resampling the training set to obtain the smaller batch of training vectors and then using a batch-mode algorithm to train the neural network. Sampled optimisation takes longer than stochastic training to perform a single weight update, but the use of larger samples results in more accurate gradient calculations which usually means fewer weight updates and shorter training times, as well as better solutions than those obtained with stochastic training.

Another method of sampled optimisation is known as stratified sampling [7]. Stratified sampling ensures that each possible class has the same *a priori* distribution in the sample that it has in the complete training set. This makes the error gradient calculated

from the sample much more representative of the true gradient, because the sample is larger and the classes' distribution information is taken into account as well. Using this better estimate, the solution vector tends to oscillate a lot less, making the decision to stop training much easier.

2.1.5 Conjugate gradients

Using larger batches of samples to calculate the error gradient does not result in comparable reductions in the number of iterations required to find a solution. The reason for this lack of increased performance is that plain steepest gradient descent algorithms do not use the more accurate gradient calculations effectively [7].

Steepest gradient descent algorithms use the last calculated gradient value, and descends directly in that direction according to Equation (2.11):

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E|_{\mathbf{w}^{(\tau)}} \tag{2.11}$$

where

- η is the learning rate, and
- τ is the step number.

A problem with gradient descent is to determine the optimal value for η. If η is too large, the algorithm may overshoot the minimum and if it is too small, it will take very long to reach the minimum. Furthermore, the optimal value for η changes during the course of the optimisation. Another big problem with gradient descent is that almost everywhere on the error surface, the local gradient does not point directly to the minimum error.

An alternative implementation is to perform a line search in the specified direction after each gradient calculation in order to find the minimum error. On termination of the line search, the error gradient is zero in the direction of the search.

Conjugate gradient algorithms use line searches and are a large improvement on plain gradient algorithms. They use the current gradient information, as well as previous gradients' information. In this method, the new search direction is constructed so that it is conjugate to the previous search direction. A direction $\mathbf{d}^{\tau+1}$ is conjugate to direction $\mathbf{d}^{\tau}$ if the component of the gradient parallel to $\mathbf{d}^{\tau}$, which has just been made zero by the previous line search, remains zero as we move in direction $\mathbf{d}^{\tau+1}$.

One very effective conjugate gradient descent algorithm is ALECO-2, and was proposed by Perantonis and Karras [8]. Using this method, the weights are updated using Equation (2.12):

$$\Delta \mathbf{w}^{(k)} = -\frac{\lambda_1}{2\lambda_2}\mathbf{g}^{(k)} + \frac{1}{2\lambda_2}\mathbf{w}^{(k-1)} \tag{2.12}$$

where λ_1 and λ_2 need to be computed at each weight update. λ_1 and λ_2 are calculated by substituting

$$I_{EE} = \mathbf{g}^{(k)\mathrm{T}}\mathbf{g}^{(k)} \tag{2.13}$$

$$I_{E\Phi} = \mathbf{g}^{(k)\mathrm{T}}\Delta\mathbf{w}^{(k-1)} \tag{2.14}$$

$$I_{\Phi\Phi} = \Delta\mathbf{w}^{(k-1)\mathrm{T}}\Delta\mathbf{w}^{(k-1)} \tag{2.15}$$

into

$$\lambda_2 = \frac{1}{2}\left[\frac{I_{EE}(\delta P)^2 - (\delta E)^2}{I_{\Phi\Phi}I_{EE} - I_{E\Phi}^2}\right]^{-\frac{1}{2}} \tag{2.16}$$

$$\lambda_1 = \left[\frac{I_{E\Phi} - 2\lambda_2\delta E}{I_{EE}}\right]. \tag{2.17}$$

Suitable values for the parameters δP and δE are determined during trial runs.

The ALECO-2 method's classification performance was tested on synthetic data (the parity problem) and real-world data (English speech data) by Holm and Botha [9]. It was found that ALECO-2 performed favourably compared to other, well known, gradient based algorithms.

2.1.6 Committees of networks

It is standard practice to train a number of neural networks on the same problem, using different starting positions in weight space and different numbers of hidden neurons, to ensure that the selected network is effective. Usually only the best network is retained while the others are discarded, which is a waste of processing time. As the chances are good that there are at least a few networks that will perform almost as well as the best one, these extra networks may be utilised by combining them all in a committee of neural networks.

Using Cauchy's inequality [1], it can be proved that the committee's classification performance will at least be up to the same standard as the average of individual networks' performances. This may seem to degrade the maximum attainable performance because the best network will perform better than the average, however, one must keep in mind that it is possible for noise in the testing data to favour the best network, even though it may perform noticeably worse on other data sampled from the same distribution.

Committees are normally implemented using some form of averaging of the member networks' predictions. Because of this averaging, committees are more consistent in their performance on different sets of data sampled from the same distribution. The averaging reduces the variance contained in the member networks.

2.1.7 Bias and variance

Training a neural network is the process of trying to approximate an unknown function, $h(x)$, by fitting a non-parametric function, $g(x)$, to a given set of data. If we stop training too early, the network will not fit $g(x)$ close enough to $h(x)$, resulting in a network with high bias and small variance (Figure 2.2). If we continue to train the network for too long, it will begin to specialise on the training data. This will result in a network that fits the decision boundary exactly to the training set, which causes high variance and small bias (Figure 2.3). The process of specialising on the training set is known as overtraining.

In general, one tries to find the optimal balance between bias and variance. To achieve this, it is necessary to stop training the neural network just before it starts to specialise on the training set. Figure 2.4 shows a typical network's performance on its training set and on a separate sample from the true distribution as training progresses.

A method often used to stop training at the correct time is known as cross-validation. Cross-validation involves using a second set of data - the cross-validation set - in the training process. The cross-validation set is normally about 10% of the size of the training set to prevent unnecessary long computation times. When cross-validation is used for early stopping, the network is tested on the cross-validation set after each weight update. When the performance on the cross-validation set starts to decrease, training is stopped (X number of iterations in Figure 2.4).

Cross-validation balances the bias-variance trade-off by limiting the variance. Another method to accomplish this, is to minimise the bias while allowing the variance to

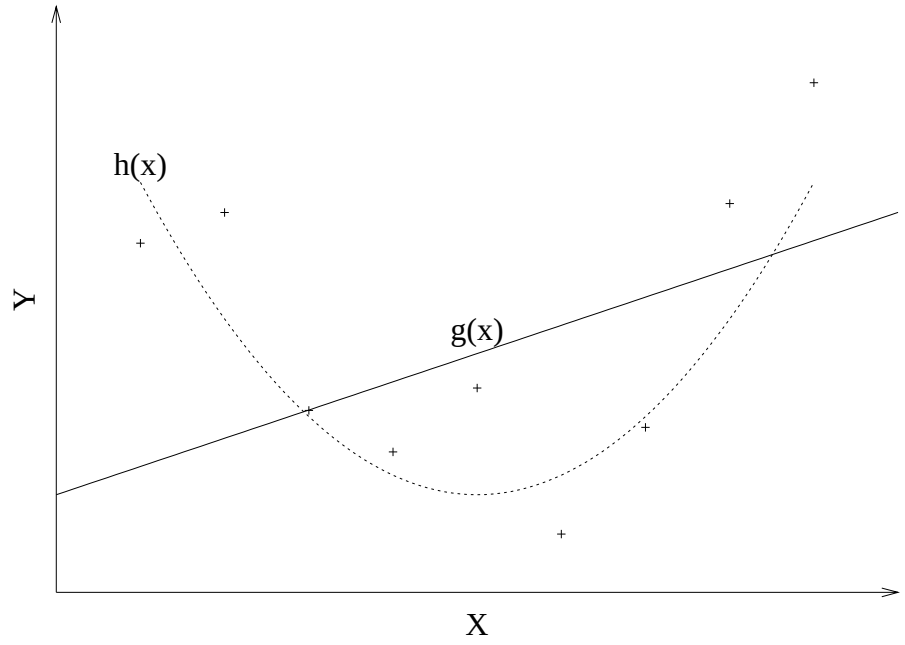

Figure 2.2: An illustration of bias and variance. The crosses denote the data points, which were generated by adding random noise to the underlying function $h(x)$. Using an insufficent model, $g(x)$ is not fitted close enough to the data, resulting in a solution with high bias and low variance.

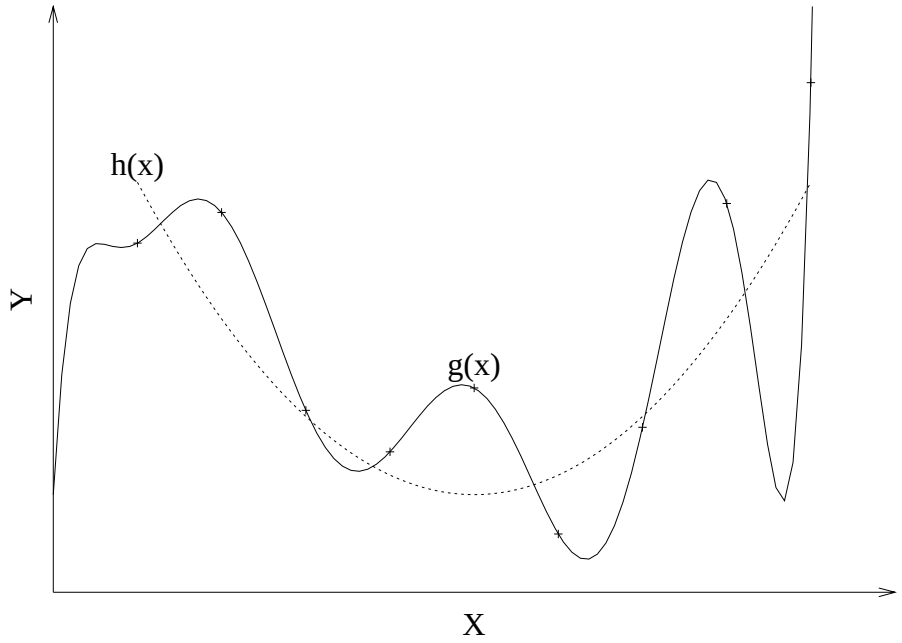

Figure 2.3: As in Figure 2.2, but using too complex a model. An exact fit of the data is obtained, resulting in a solution with low bias and high variance. This is known as overfitting.

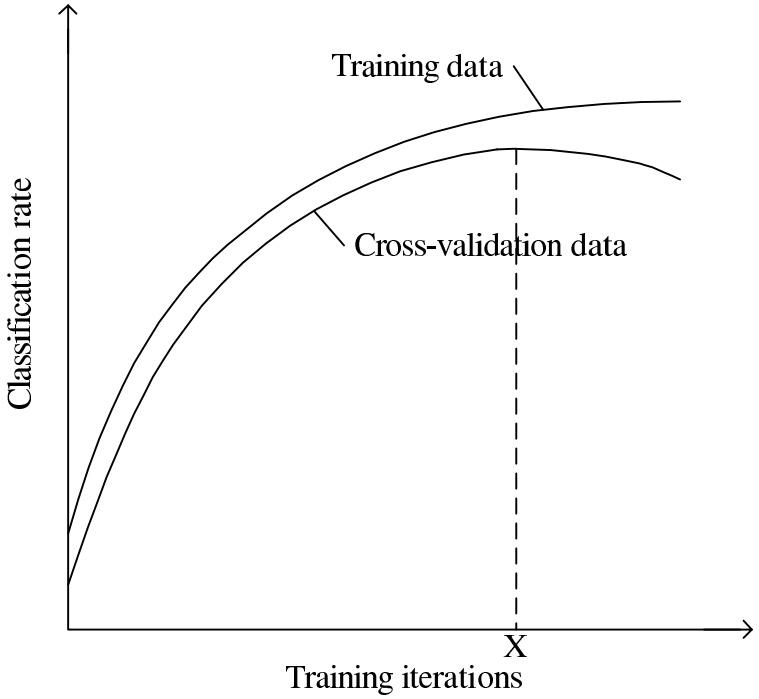

Figure 2.4: Performance of a neural network on its training set and the cross-validation set.

increase. When the bias is minimised, the next step is to decrease the variance. The process is implemented by overtraining some neural networks, which minimises the bias, and then combining a number of the networks in a committee, which minimises the variance.

2.2 Bayesian learning

2.2.1 Bayesian inference

When predicting the outcome of a random event, one of two statistical methods may be used. The first method is the well-known frequentist method. The second is known as Bayesian inference.

The frequentist method uses an estimator to predict unknown values. Such an estimator can only be calculated from historical data. A well known example where an estimator would produce good results is in the tossing of a coin, where the purpose is to predict the number of times a head will show. A good estimator for this problem is to perform an experiment of a number of tosses and then to divide the number of times a head was shown by the total number of tosses. Future predictions are made by using the estimator as the probability of tossing a head.

The method of Bayesian inference expresses all uncertainties as probability distributions over the possible results, in stead of single valued estimators. In Bayesian learning, these probability distributions start out as an *a priori* distribution which is updated each time new data is received.

In the coin tossing example, one might start with a uniform prior distribution for the probability of a head showing. Before any coins are tossed, this prior distribution may be used to predict the outcome. As the experiment progresses and new data become available, the likelihood of a head showing will be updated. The likelihood, combined

with the prior distribution, is used to calculate the posterior distribution, which is used to estimate the probability of tossing a head.

Prediction with Bayesian networks

The result of Bayesian learning is a probability distribution over the model parameters that expresses the beliefs regarding how likely the different parameters' values are.

Training starts by selecting a prior distribution, $p(\theta)$, for the parameters θ. The prior distribution expresses our beliefs about the parameters' values before any data are seen. After observing n data points, $x^{(1)}, \ldots, x^{(n)}$, the posterior distribution is calculated using

$$p(\theta|\, x^{(1)}, \ldots, x^{(n)}) = \frac{p(x^{(1)}, \ldots, x^{(n)}|\, \theta)\, p(\theta)}{p(x^{(1)}, \ldots, x^{(n)})} \tag{2.18}$$

$$\propto L(\theta|\, x^{(1)}, \ldots, x^{(n)}) p(\theta). \tag{2.19}$$

To predict a value using the Bayesian method, one integrates the predictions of the model with respect to the posterior distribution of the parameters. The predictive distribution of x^{n+1} is given by

$$p(x^{(n+1)}|\, x^{(1)}, \ldots, x^{(n)}) = \int p(x^{(n+1)}|\, \theta) p(\theta|\, x^{(1)}, \ldots, x^{(n)}) d\theta. \tag{2.20}$$

The ability to produce the distribution of the predicted values is one of the advantages of using the Bayesian approach.

Neal tested different implementations of Bayesian learning on a robot arm problem [3, 10], and found that the hybrid Monte Carlo method performed the best. However, the implementation of Bayesian learning is likely to change in the future, as research in this subject continues.

Hierarchical models

In Equation (2.20), θ is used to model the distribution of many observable quantities $x^{(i)}$. If different groups of quantities have different parameters describing their distributions, i.e. $\theta = \{\theta_1, \ldots, \theta_p\}$, it might be useful to specify the distribution of the different θ's using a common hyperparameter. Schemes using hyperparameters are known as hierarchical models.

Similar θ's are usually grouped together under the same hyperparameter. A typical hierarchical model, and the one used in this research, is to have a hyperparameter controlling all the weights in a layer, and one hyperparameter to control the bias of the layer. Applying this model to a neural network consisting of an input layer, a single hidden layer and an output layer, would result in four hyperparameters.

A hyperparameter is used to obtain a value which is used as a parameter for the controlled quantity's probability distribution. An example of this is where the controlled quantity has a Gaussian distribution with a fixed mean. The hyperparameter's distribution would then be sampled to obtain a value that will be used as the standard deviation of the controlled quantity, before sampling the controlled quantity's distribution.

The usual distributions, and again, as used in this research, is Gamma for the hyperparameters and Gaussian for the controlled quantities.

2.2.2 Calculating Bayesian neural network outputs

For feed-forward type Bayesian neural networks, the values of the output neurons are calculated in exactly the same way as the gradient descent methods. To obtain the probability that a given input belongs to class k, Equation (2.21) is used:

$$p(y = k|\,\mathbf{x}) = \exp(f_k(\mathbf{x}))/\sum_{k'} \exp(f_{k'}(\mathbf{x})), \tag{2.21}$$

where $f_k(\mathbf{x})$ is the value of output neuron k.

As the objective of the Bayesian approach is to find a predictive distribution for the target values of each new input, Equation (2.21) needs to be integrated over the range of acceptable models. This integration is performed with a Markov chain.

2.2.3 Training a Bayesian neural network

Defining the model

A model for a Bayesian neural network consists of the number of hidden layers, the number of neurons in each layer and the grouping of the weights. The grouping of the weights determines the number of hyperparameters and their associated weights.

According to Neal [3] there is no need to constrain the size of a Bayesian network based on the amount of available data, because the networks will not overfit the data. In some cases very large networks will even give substantially better results than smaller ones. However, it may be practical to limit the size of a network based on the available computing power.

Selecting priors

Bayesian inference starts with the selection of suitable prior distributions of the model parameters. These prior distributions should capture our prior knowledge about the parameters. In the application of neural networks, it is usually not obvious what the relation between the model parameters (the weights and biases) and the data distribution is.

Unless the data definitely indicate a specific distribution, the selection of a suitable prior remains an *ad hoc* procedure. For this reason Gaussian priors are often used for their mathematical simplicity, even though it has been shown that they frequently are not the best choice.

Updating the weights

In the process of Bayesian learning, the network weights are updated according to the posterior distribution of the network parameters. Using the simplest method, weights are drawn randomly from the prior distribution. The decision on whether the drawn weights are accepted or not, is made using the likelihood function, given as Equation (2.22):

$$L(\theta|\,(x^{(1)},y^{(1)}),\ldots,(x^{(n)},y^{(1)}) = \prod_{i=1}^{n} p(y^{(i)})|\,x^{(i)},\theta). \tag{2.22}$$

An obvious method of making the decision, is to accept the new weights if their likelihood is larger than the current weights' likelihood. Training which uses this brute force method is very inefficient, as the new knowledge about the parameters' distribution, contained in the likelihood, is ignored when drawing new weights.

A much more efficient method of updating the network parameters is to use Markov Chains, constructed from the weights. The Markov Chain decision method is part of the hybrid Monte Carlo method, which is discussed below.

Hybrid Monte Carlo

Discussion of the hybrid Monte Carlo method uses terminology that refers to physical units such as energy and momentum. The reason for this is that the hybrid Monte Carlo method has its origins in physics. Two variables used in this discussion that

have equivalents in the context of neural networks are q, which represents position, and $E(q)$, which represents potential energy. In the neural network environment, position is equivalent to the network parameters and potential energy is equivalent to the error value. A number of other physical values are used. These do not have equivalents in neural networks, as they are introduced only to allow the use of dynamical methods.

The hybrid Monte Carlo method proceeds as follows: select a random point, (p, q), in phase space, take L steps into a direction determined by the gradient of the phase space at the current position and momentum, and then decide whether the result of the move was acceptable using Markov Chains. If the resulting phase position is accepted, the network's weights are updated to the new values, otherwise the previous values are kept.

The potential energy, $E(q)$, is defined as

$$E(q) = -\log p(q) - \log Z \tag{2.23}$$

for probability density $p(q)$ and any convenient Z.

The initial values for p and q are drawn randomly from a canonical distribution defined by Equation (2.24):

$$p(q, p) \propto \exp\left(-H(q, p)\right) \tag{2.24}$$

with

$$H(q, p) = E(q) + K(p) \tag{2.25}$$

where:

- $H(q,p)$ is the Hamiltonian (the total system energy),
- $E(q)$ is the potential energy, and
- $K(p)$ is the kinetic energy.

$K(p)$ is given by Equation (2.26):

$$K(p) = \sum_{i=1}^{n} \frac{p_i^2}{2m_i} \tag{2.26}$$

where m_i is the mass associated with element i. These mass values may be adjusted to improve efficiency, but are normally chosen to all be one. Recall that p is Gaussian distributed, which makes p^2 chi-distributed. As $K(p)$ is a summation of these chi-distributions, it approaches a Gaussian distribution.

Sampling from the distribution given in Equation (2.24) is done in two phases: p and q are sampled for (1) a fixed total energy $H(p,q)$ and (2) for different values of $H(p,q)$.

Hamiltonian dynamics is simulated with the leapfrog method described by Equations (2.27) through (2.29), where the leapfrog method computes new values for p and q at time $\tau+\epsilon$, from p and q at time τ, and the gradient of E ($\frac{\partial E}{\partial q}$) is calculated using standard backpropagation:

$$\hat{p}_i(\tau+\frac{\epsilon}{2}) = \hat{p}_i(\tau) - \frac{\epsilon}{2}\frac{\partial E}{\partial q_i}\big(\hat{q}(\tau)\big) \tag{2.27}$$

$$\hat{q}_i(\tau+\epsilon) = \hat{q}_i(\tau) + \epsilon\frac{\hat{p}_i(\tau+\frac{\epsilon}{2})}{m_i} \tag{2.28}$$

$$\hat{p}_i(\tau+\epsilon) = \hat{p}_i(\tau+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial E}{\partial q_i}\big(\hat{q}(\tau+\epsilon)\big). \tag{2.29}$$

After L repetitions of Equations (2.27) through (2.29), the new state, $H(q^*,p^*)$, is accepted with probability

$$\min\big(1, \exp\big(-\left(H(q^*,p^*) - H(q,p)\right)\big)\big). \tag{2.30}$$

After the decision has been made on the acceptance of the calculated state, new values for the momentum variables are calculated using Gibbs sampling before the next leapfrog routine is started.

Gibbs sampling

Gibbs sampling is used when a multidimensional value must be sampled from a probability distribution. This method picks a random value from the distribution for one of the dimensions while keeping the other dimensions constant. To illustrate, suppose a parameter $\theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ has to be sampled, the process would proceed as follows:

- Pick $\theta_1^{(t+1)}$ from the distribution of θ_1 given $\theta_2^{(t)}, \theta_3^{(t)}, \ldots, \theta_m^{(t)}$.
- Pick $\theta_j^{(t+1)}$ from the distribution of θ_j given $\theta_1^{(t+1)}, \ldots, \theta_{j-1}^{(t+1)}, \theta_{j+1}^{(t)}, \ldots, \theta_m^{(t)}$.
- Pick $\theta_m^{(t+1)}$ from the distribution of θ_m given $\theta_1^{(t+1)}, \theta_2^{(t+1)}, \ldots, \theta_{m-1}^{(t+1)}$.

In our experiments, θ represents the network weights.

Algorithm

Summarised, the algorithm for the hybrid Monte Carlo method is as follows:

1. Define the network architecture (number of inputs, outputs, hidden layers, hidden neurons in each layer and the hyperparameter structure).
2. Select the prior distribution for the parameters (weights) or hyperparameters. For the weights, we choose Gaussian distributions with zero means and variances determined by the controlling hyperparameter, and for the hyperparameters we choose Gamma distributions.

3. Define the energy function $E(\theta)$, which in this case, is the sum of squares error.
4. Use Gibbs sampling to generate random values for the initial position (weights).
5. Use Gibbs sampling to generate random values for the momentum variables.
6. Use the leapfrog Equations (2.27) through (2.29) to compute a new state. This is repeated L times, where L and ϵ are determined during trial runs.
7. Accept or reject the new state according to Equation (2.30).
8. Repeat steps 5 to 7 until the hyperparameter values stabilise.

In the next chapter, we describe experiments performed using the gradient descent and Bayesian learning methods for neural network training.

Chapter 3

Experiments

This chapter describes the topology of the neural networks used, the data used for experimentation, the maximum theoretical performance of the neural networks and the experiments. We describe the experimental method followed, the results obtained and discuss the results.

3.1 Topology

The same architecture was used for all the networks and the training methods. This was done in order to be able to make a direct comparison between the different training methods. Each network consisted of five input neurons, five hidden neurons and five output neurons. This totals to a number of 55[1] weights in the neural network.

The number of input and output neurons was dictated by the problem to be solved. It was decided to use five partially overlapping classes from a five dimensional space to make the problem challenging.

[1] (5 input + 1 bias) * 5 hidden + 5 hidden * 5 output = 55

Only one layer of hidden neurons was used, as this is enough for a neural network to represent any continuous surface [1]. The number of neurons in the hidden layer was determined by trial runs using both the Bayesian method and the gradient descent method. It was found that five hidden neurons were the minimum number that would give both training methods the ability to perform well on the given data. Both training methods would gain nothing from more hidden units, although the Bayesian method might have performed equally well with fewer hidden neurons.

3.2 Data

Because of the need to calculate the theoretical maximum classification rate, the probability distribution of the data has to be known. Using synthetic data gives us the ability to control the data's distribution as well as the amount of available data.

For a network consisting of 55 weights, the optimal number of training vectors is 1760, i.e. 32 times the weights count [11], although it is usually considered sufficient if the number of training vectors is as little as ten times the total number of weights.

In practical problems where training data is limited, one may easily find that the available number of training vectors is only about five times more that the number of network weights. To create a training environment similar to these kinds of conditions, the size of the training data set was restricted to 300 vectors.

A cross-validation data set consisting of 30 vectors was used by the gradient descent method to determine when to stop training for a particular network.

In order to test the neural networks thoroughly, the test data set consisted of 10,000 vectors.

To create the data sets, data was drawn randomly from Gaussian distributions. Data was generated for five partially overlapping classes from a five dimensional space, with

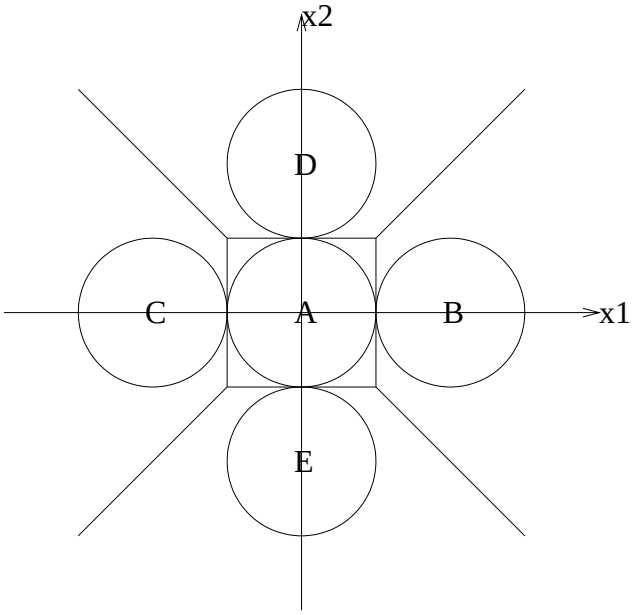

Figure 3.1: A 2-D projection of the input space with optimal decision surfaces.

all the dimensions independent. All five classes have a variance of 0.7^2 and respective means of:

- $\mu_A = (\ 0, 0, 0, 0, 0\ )$
- $\mu_B = (\ 1, 0, 0, 0, 0\ )$
- $\mu_C = (\text{-}1, 0, 0, 0, 0\ )$
- $\mu_D = (\ 0, 1, 0, 0, 0\ )$
- $\mu_E = (\ 0,\text{-}1, 0, 0, 0\ )$

In some of the experiments only subsets of the training set were used to train the networks. These subsets were created by randomly drawing the required number of vectors from the original training set and storing it separately.

All classes are equally likely, resulting in *a priori* class probabilities of 0.2.

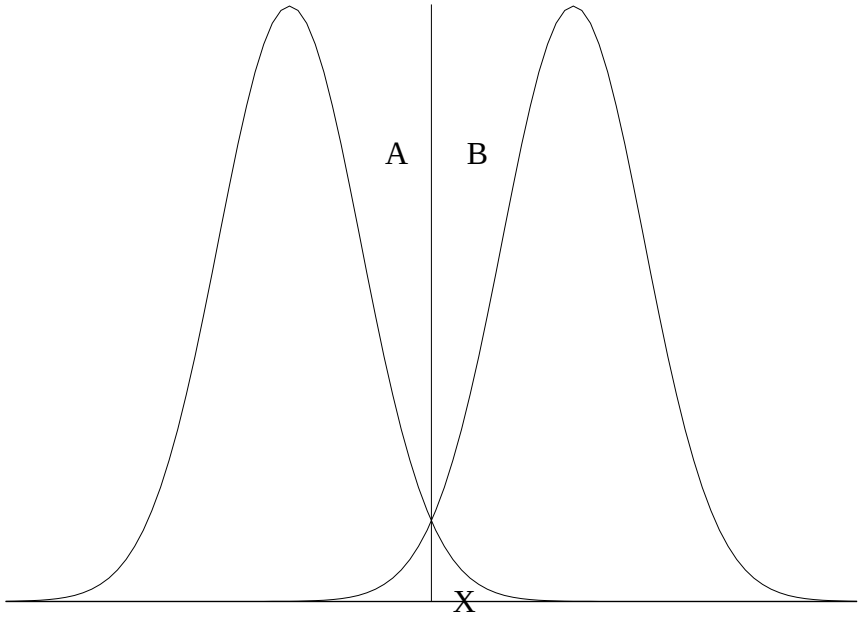

Figure 3.2: An illustration of the optimal decision surface for a two-class problem given the probability distribution functions of both classes.

Figure 3.1 shows a 2-D projection of the input space. The circles indicate contours of distributions of equal probability, and the line segments indicate the decision surfaces.

3.3 Maximum theoretical performance

To classify unknown data as belonging to a specific class, one devides the whole range of possible values into regions. Lets call the regions 'Γ' and the boundaries between the regions 'decision surfaces'. Figure 3.2 is an illustration of these two concepts for a two class problem.

If the point X in Figure 3.2 is to be classified, we would say that it belongs to class B because it lies in the region Γ_B. To calculate the maximum classification rate possible, the probability of correctly classifying each class is integrated over all the classes.

To calculate the probability of an input belonging to a class, one needs the probability

distribution of the data. For a Gaussian distribution, the probability distribution is completely defined by the mean and the variance.

As described before, the data used consisted of five partially overlapping classes in a five dimensional space.

The maximum classification rate is given by Equation (3.1):

$$P_{max} = 0.2(P_{\mathrm{A}} + P_{\mathrm{B}} + P_{\mathrm{C}} + P_{\mathrm{D}} + P_{\mathrm{E}}) \tag{3.1}$$

where P_{X} is the probability of classifying a data vector correctly as class X.

To calculate the values used in Equation (3.1), the first step is to find al the decision surfaces. For a five-dimensional variable with statistically independent dimensions and equal variance, σ^2, in all dimensions, the Gaussian distribution is

$$p(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{\frac{5}{2}}} e^{-\frac{\|\mathbf{x}-\mu\|^2}{2\sigma^2}}. \tag{3.2}$$

If class X should be selected in stead of class Y, Γ_{X} is defined as the region where

$$p_X(\mathbf{x}) \geq p_Y(\mathbf{x}). \tag{3.3}$$

Substituting Equation (3.3) into Equation (3.2), we get:

$$\frac{1}{(2\pi\sigma^2)^{\frac{5}{2}}} e^{-\frac{\|\mathbf{x}-\mu_X\|^2}{2\sigma^2}} \geq \frac{1}{(2\pi\sigma^2)^{\frac{5}{2}}} e^{-\frac{\|\mathbf{x}-\mu_Y\|^2}{2\sigma^2}}. \tag{3.4}$$

After some manipulation

$$(x_1 - \mu_{X1})^2 + \cdots + (x_5 - \mu_{X5})^2 \leq (x_1 - \mu_{Y1})^2 + \cdots + (x_5 - \mu_{Y5})^2 \tag{3.5}$$

results.

Using Equation (3.5) and the classes' means to calculate $A \geq B$, $A \geq C$, $A \geq D$ and $A \geq E$, we obtain the limits of Γ_A:

$$-\frac{1}{2} \leq x_1 \leq \frac{1}{2} \tag{3.6}$$
$$-\frac{1}{2} \leq x_2 \leq \frac{1}{2} \tag{3.7}$$
$$-\infty \leq x_3 \leq \infty \tag{3.8}$$
$$-\infty \leq x_4 \leq \infty \tag{3.9}$$
$$-\infty \leq x_5 \leq \infty \tag{3.10}$$

Following the same procedure for the other four classes, we obtain the following decision regions:

Γ_B:

$$\frac{1}{2} \leq x_1 \leq \infty \tag{3.11}$$
$$-x_1 \leq x_2 \leq x_1 \tag{3.12}$$

Γ_C:

$$-\infty \leq x_1 \leq -\frac{1}{2} \tag{3.13}$$
$$x_1 \leq x_2 \leq -x_1 \tag{3.14}$$

Γ_D:

$$-x_2 \leq x_1 \leq x_2 \tag{3.15}$$
$$\frac{1}{2} \leq x_2 \leq \infty \tag{3.16}$$

Γ_E:

$$x_2 \leq x_1 \leq -x_2 \tag{3.17}$$

$$-\infty \leq x_2 \leq -\frac{1}{2}. \tag{3.18}$$

Regions B through E all have the same limits as region A for dimensions x_3, x_4 and x_5.

The second step in finding the values used in Equation (3.1), is to integrate each class' probability distribution over the limits for the class. This integral is

$$P_Y = \int_{x_1}\int_{x_2}\int_{x_3}\int_{x_4}\int_{x_5} p(x_1)p(x_2)p(x_3)p(x_4)p(x_5)\; dx_5 dx_4 dx_3 dx_2 dx_1. \tag{3.19}$$

As

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1, \tag{3.20}$$

and neither x_1 nor x_2 is dependant on x_3, x_4 or x_5, Equation (3.19) becomes

$$P_Y = \int_{x_1}\int_{x_2} p(x_1)p(x_2) dx_2 dx_1. \tag{3.21}$$

Using Equation (3.21) and the results of Equations (3.6) through (3.18), we obtain Equations (3.23) through (3.27), which are solved numerically. These equations use the function $N(x, \mu, \sigma)$, which is defined as

$$N(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{3.22}$$

$$P_A = \int_{-\frac{1}{2}}^{\frac{1}{2}} N(x_1, 0, 0.7)\, dx_1 \int_{-\frac{1}{2}}^{\frac{1}{2}} N(x_2, 0, 0.7)\, dx_2$$
$$= 0.276 \tag{3.23}$$
$$P_B = \int_{\frac{1}{2}}^{\infty} N(x_1, 1, 0.7) \int_{-x_1}^{x_1} N(x_2, 0, 0.7)\, dx_2 dx_1$$
$$= 0.663 \tag{3.24}$$
$$P_C = \int_{-\infty}^{-\frac{1}{2}} N(x_1, -1, 0.7) \int_{x_1}^{-x_1} N(x_2, 0, 0.7)\, dx_2 dx_1$$
$$= 0.663 \tag{3.25}$$
$$P_D = \int_{\frac{1}{2}}^{\infty} N(x_2, 1, 0.7) \int_{-x_2}^{x_2} N(x_1, 0, 0.7)\, dx_1 dx_2$$
$$= 0.663 \tag{3.26}$$
$$P_E = \int_{-\infty}^{-\frac{1}{2}} N(x_2, -1, 0.7) \int_{x_2}^{-x_2} N(x_1, 0, 0.7)\, dx_1 dx_2$$
$$= 0.663. \tag{3.27}$$

Substituting Equations (3.23) through (3.27) into Equation (3.1), we find that the maximum classification performance is

$$P_{max} = 58.53\%. \tag{3.28}$$

3.4 Experimental procedure

Neural networks were trained using a number of different methods and then compared afterwards. The training was executed in the form of five experiments.

3.4.1 Experiment 1 : Bayesian learning

A neural network was trained using the method of Bayesian learning. This training was continued for 200 iterations on all the available training data (batch training). Only the last 100 iterations were used to test the network's classification performance.

3.4.2 Experiment 2 : CGD and committees

One hundred networks were trained using the gradient descent method on all the available training data (batch training). On completion the networks were added to a committee, while varying its size from one member to 100 members. The committee was started with only the best network as member and each time adding the next best network. After each addition of a network, the committee's performance on the test set was noted. This whole process was performed twice. The first time the networks used the cross-validation set to prevent overtraining. The second time no cross-validation was used and the networks were forced to overtrain.

3.4.3 Experiment 3 : Sampled CGD and committees

One hundred networks were trained using the gradient descent method on unique subsets of the training data. The trained networks were added to a committee in the same manner as described for experiment 2. Again, this experiment was also performed twice: once training was stopped according to the cross-validation set, and once the networks were forced to overtrain. In experiment 3 each individual neural network started training from a different random position (possibly not unique) in weight space.

3.4.4 Experiment 4 : Sampled CGD from trained starting position

This experiment is the same as described in experiment 3 except that the individual neural networks all started training from the same position in weight space. This starting position was found by training 100 neural networks on the whole training set while using cross-validation to stop the training process. The final weights of the best network from the group of 100 were used as the starting weights.

3.4.5 Experiment 5 : Hybrid training with CGD and Bayesian learning

For the final experiment a hybrid method was used to train the neural networks. Firstly a network was trained on all the whole training set to about 70% of the expected maximum classification performance using the gradient descent algorithm. The network's training was then completed using the Bayesian learning algorithm, which started training from the position where the gradient descent phase stopped.

3.5 Results

3.5.1 Experiment 1 : Bayesian learning

The Bayesian networks require a few iterations to settle down before useable networks are produced. A graph of the network's hyperparameters is plotted to determine which iterations are good enough to employ in the final network. Figure 3.3 is an example of such a graph, where the iteration number is given on the horizontal axis and the third hyperparameter's value on the vertical axis. The third hyperparameter's value is the standard deviation of the distribution of the second layer weights.

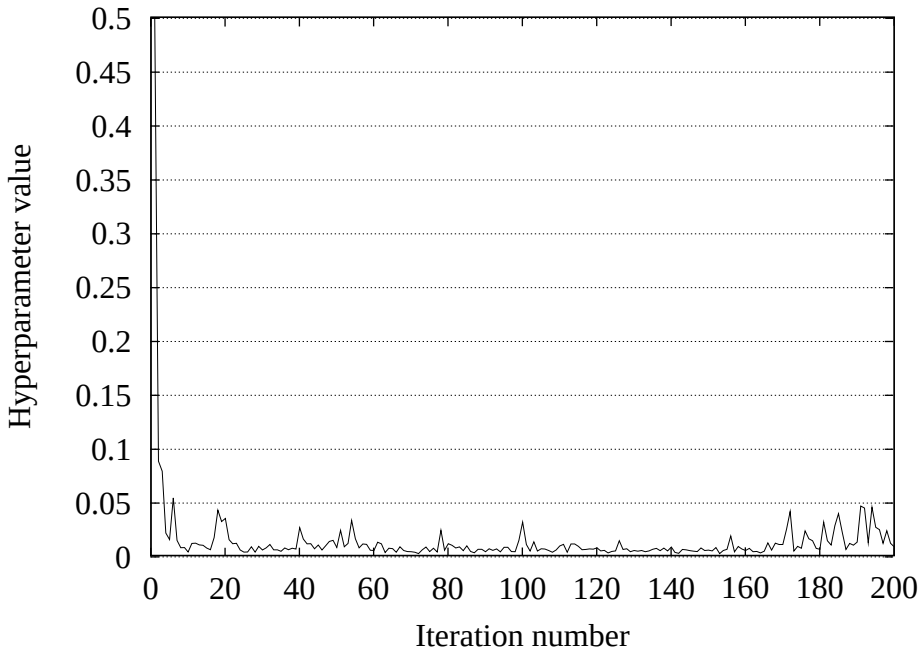

Figure 3.3: Hyperparameter values for a Bayesian network.

For these experiments a hyperparameter value of 0.4 was considered to be 'settled', although this is arbitrary and varies for different problems. Using the 0.4 level as criterium, all the iterations from two and upwards could have been included in the final network. One must keep in mind that a few poor iterations may be included without any serious performance degradation if this is required in order to include many qualifying iterations.

Using the last hundred iterations, the Bayesian network achieved 57.76 % classification accuracy. This is within 1 % of the theoretical maximum, which is very good if one considers that the training was done on only 300 samples.

3.5.2 Experiment 2 : CGD and committees

Figure 3.4 shows the results obtained in experiment 2 as classification performance on the test set versus number of member networks in the committee.

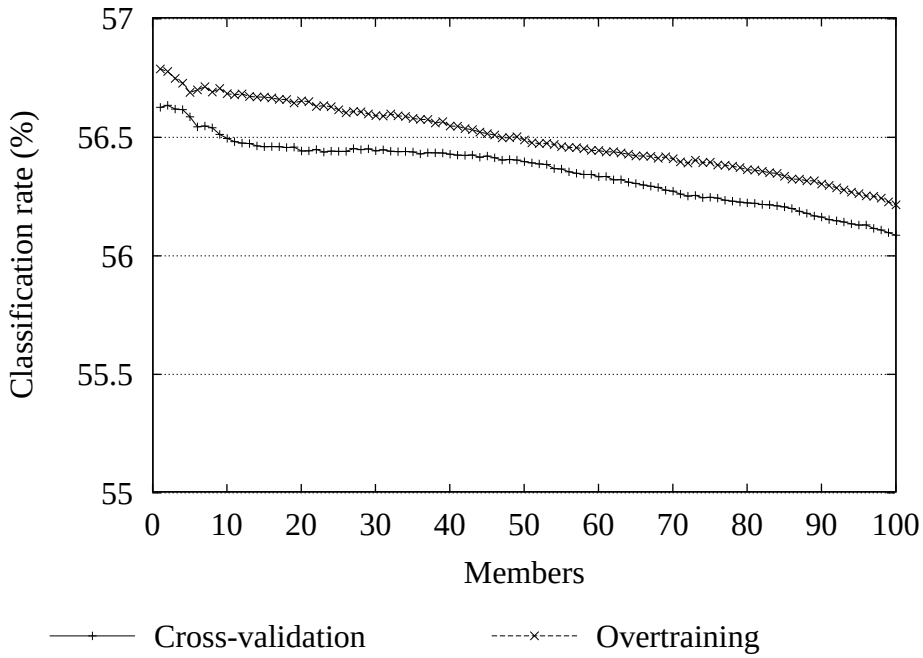

Figure 3.4: Classification results as a function of the number of member networks in a committee. The networks were trained on the whole training set.

The curves are shown for both the committee using cross-validation as well as for the committee using overtraining. We see that the overtraining method consistently performed better than cross-validation by about 0.25 %. The best committee of CGD networks with cross-validation achieved 56.82% and with overtraining, 56.97%.

The trend in the results is consistent with the theory that committees average variance and that they will perform better when the member networks are overtrained.

As more networks are added to the committee, the performance slowly decreases. This performance reduction is also to be expected, as the newer networks are always inferior to the networks already contained in the committee (by nature of our experimental design). It is also worth to note that although using a 100 networks in a committee results in about 1% loss of classification performance, it is still much better than the individual performance of the inferior networks: The worst networks only achieve performances in the 40% when evaluated individually.

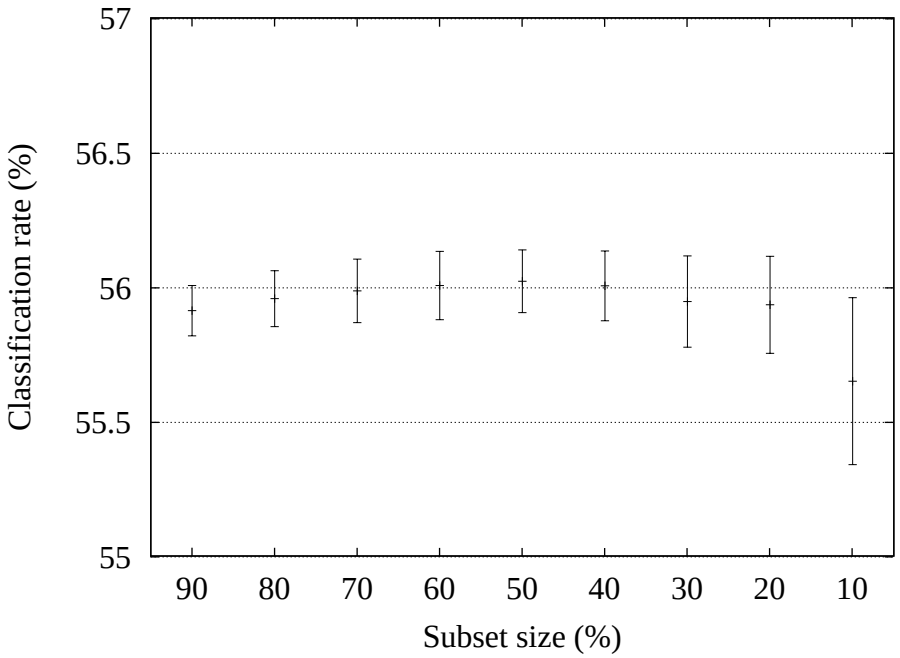

Figure 3.5: Results for individual CGD networks trained using cross-validation and different initial weights. The mean and standard deviation are shown as the size of the training data subset changes.

3.5.3 Experiment 3 : Sampled CGD and committees

The results of experiment 3 is graphed in Figures 3.5 through 3.8. The performances of the individual networks were noted, and graphed in Figure 3.5 and Figure 3.6. The plot was made to illustrate the performance of the networks over the range of training subset sizes tested. The subset size is given as a percentage (of 300 samples). Figure 3.5 is graphed for the case where the networks were trained using cross-validation, while Figure 3.6 is graphed for the instance where the networks were overtrained. In both Figure 3.5 and Figure 3.6 the average is shown with bars indicating the standard deviation.

In Figure 3.5 the classification performance starts out low at 90% of the data. It then steadily increases to a maximum at 50% of the data. After 50% the performance decreases again. The curve ends lower at 10% data than where it started out from at 90%. The best performance achieved with this sampled optimisation is 56.14%,

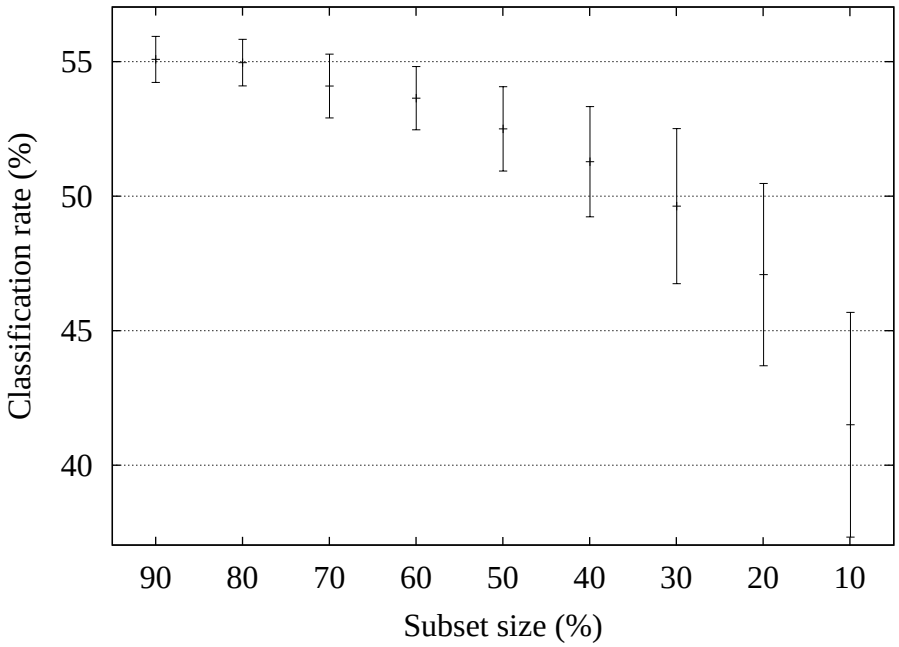

Figure 3.6: Results for individual CGD networks trained using overtraining and different initial weights. The mean and standard deviation are shown as the size of the training data subset changes.

which is well below the best performance of the CGD network using cross-validation of 56.82% (Experiment 2). In Figure 3.6 the classification performance starts at an average of 55% and constantly decreases as the size of the subset is decreased. The best network here performed at 55.78%. This is well below the performance of the best overtrained CGD network at 56.97% (Experiment 2).

At first glance one would expect to simply extrapolate the graphs in Figures 3.5 and 3.6 to estimate a performance value for networks using 100% of the training data. Doing this would lead one to believe that the CGD networks using cross-validation should achieve performances of about 55.8% (subset size of 100% in Figure 3.5), while the overtrained CGD network should achieve 55% (subset size of 100% in Figure 3.6). As mentioned in the previous paragraph, this is not the case. The reason for the increased performance at 100% for both cases, is that in experiment 2, all the networks used the same training data and in experiment 3, each network used a different subset of training data. Experiment 3's networks were not trained on the same subsets, as this

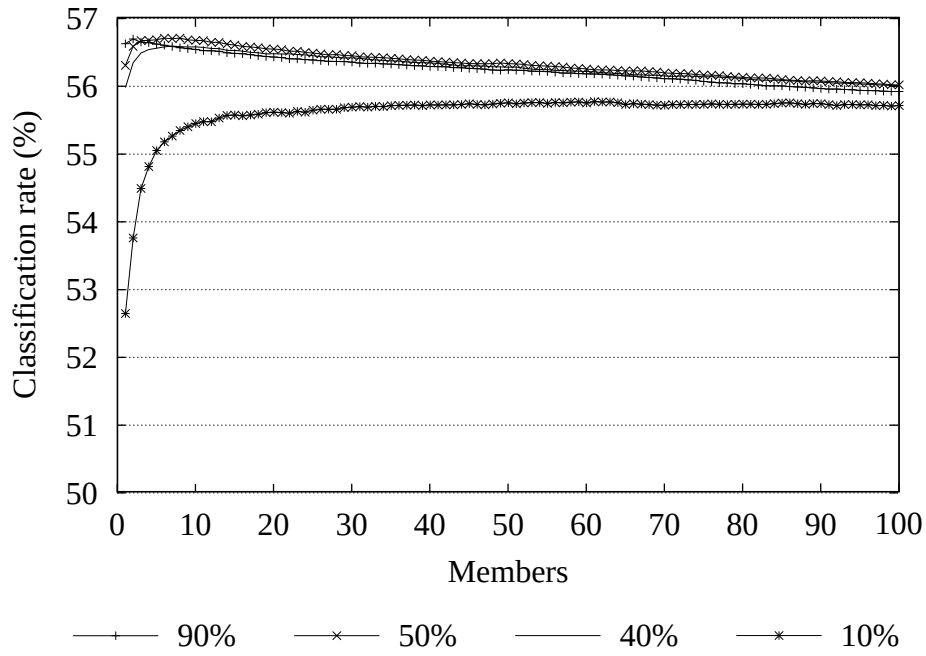

Figure 3.7: Results for committees using cross-validation and different initial weights.

would not be sampled optimisation, but only an optimisation problem with less data.

For both Figure 3.5 and Figure 3.6 the variances increase as the data sets shrink. Comparing these two figures, we also note that the cross-validation trained networks consistently outperform the overtrained networks. This is to be expected, as the idea behind cross-validation is to stop training when performance starts to decrease. Using cross-validation is also the reason why the variances in Figure 3.5 are much smaller than the variances in Figure 3.6. Figure 3.6's larger variance is caused by small data sets that have local minima that are far removed from the large training set's minimum. Furthermore, the large variance is also caused by the overtraining that causes the networks to specialise on these smaller sets. The maximum reached in Figure 3.5 can be explained with the bias-variance trade-off which balances out at approximately 50% of the training data.

Figure 3.7 and Figure 3.8 show the performance of committees as the number of member networks varies. In these graphs the neural networks all started with random initial

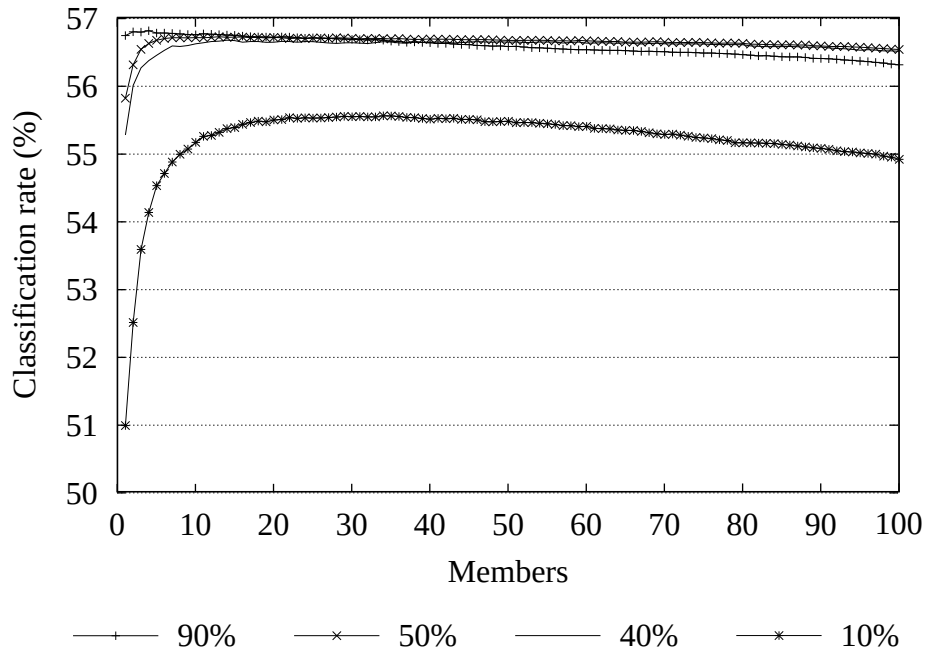

Figure 3.8: Results for committees using overtraining and different initial weights.

weight vectors, Figure 3.7's networks used cross-validation and Figure 3.8's networks used overtraining. For clarity's sake, only the results for networks trained on 90%, 50%, 40% and 10% of the training data are shown.

Comparing Figure 3.7 to Figure 3.8 we see the same trend in both graphs; the performance starts out low at one network in the committee, it then increases as networks are added and, after a maximum, starts to decrease. It is also evident that by using more than 50% of the data, very little performance increase is gained. Using less than 40% of the available data, however, rapidly decreases the performance.

The committee of overtrained networks also outperforms the cross-validation trained committee by a small margin. This was expected, as overtraining results in networks that have very low bias with high variance, i.e. the individual networks specialise on their training sets. This high variance is then mostly removed using the averaging effect that a committee has on network variance, resulting in a committee with both low bias and low variance. The cross-validation trained committee has poorer performance

because cross-validation balances the individual networks' bias-variances, which means that the committee's variance averaging effect has little impact on the total variance. In the end the committee has about the same bias-variance properties as a single network.

The graph in Figure 3.4 actually represents the plots in Figure 3.7 and Figure 3.8 for 100% data sets. However, it was placed on a different graph as it would be meaningless to put multiple neural networks in a committee when they all started training with the same initial weights on the exact same data, as would have been the case in Figure 3.9 and Figure 3.10.

3.5.4 Experiment 4 : Sampled CGD from trained starting position

As in Figure 3.5 and Figure 3.6, Figure 3.9 and Figure 3.10 show results where classification performance is given for the individual neural networks. The results shown were obtained in experiment 4, and, as previously mentioned, all the networks trained in experiment 4 started with the same initial weight vector. Figure 3.9 is given for the case where the neural networks were using cross-validation while Figure 3.10 is for the case where the neural networks were overtrained.

Figure 3.9 has the same form and other characteristics as Figure 3.5 while Figure 3.10 looks the same as Figure 3.6. This shows that the form of the curves is dependant on whether cross-validation or overtraining was used, while the initial weight vector mainly influences the networks' bias and variance. As was the case in experiment 3, the cross-validation trained networks outperform the overtrained networks regardless of the size of the training sets. Also, the cross-validation method's variances are smaller (note the scale differences in the figures) than the overtraining method. Larger variances would decrease the neural network's performance, which would cause the cross-validation algorithm to stop training.

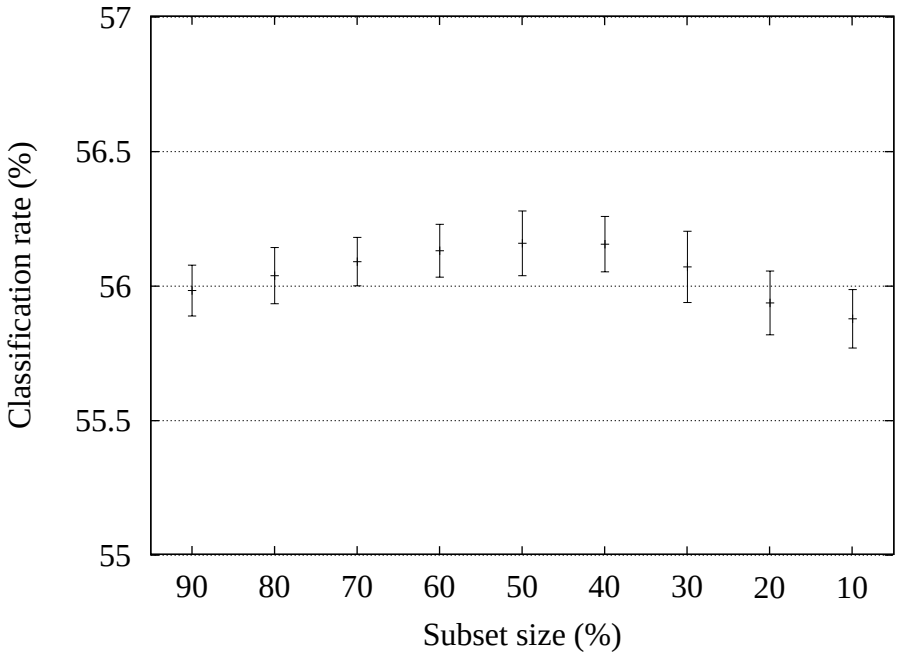

Figure 3.9: Results for networks using cross-validation and the same initial weights.

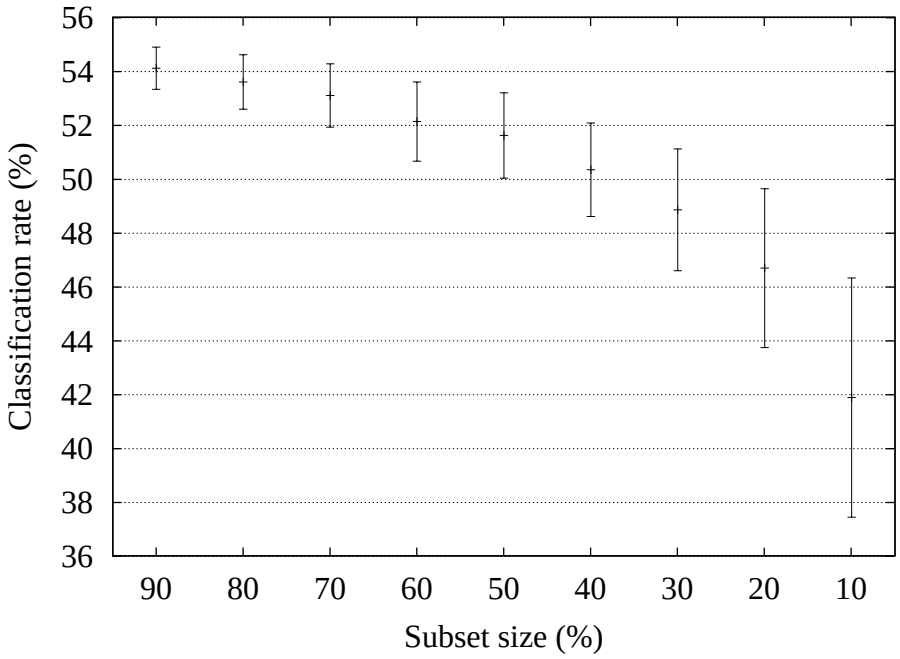

Figure 3.10: Results for networks using overtraining and the same initial weights.

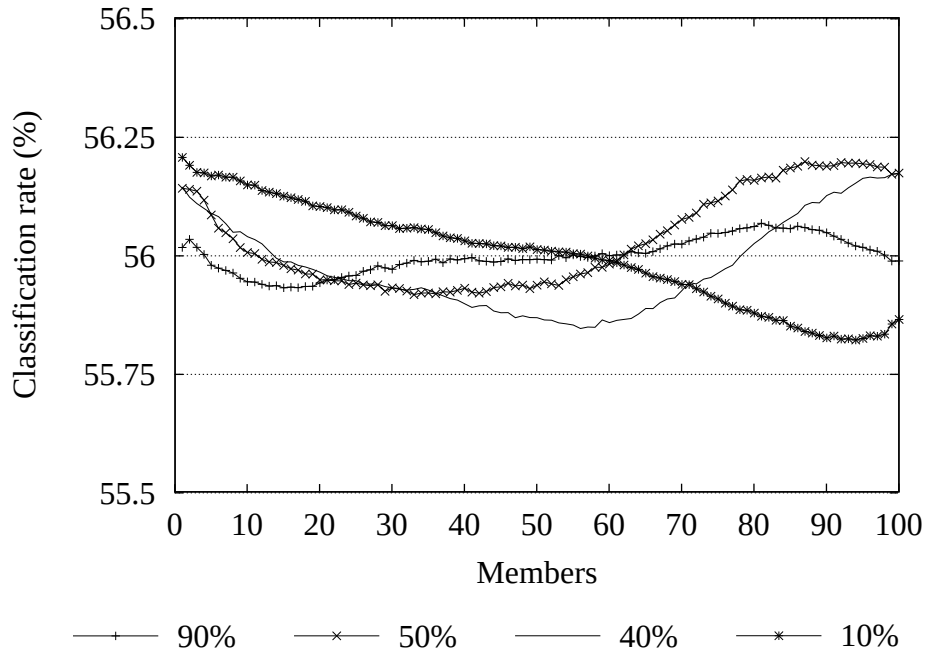

Figure 3.11: Results for committees using cross-validation and the same initial weights.

Figure 3.11 and Figure 3.12 are the results from experiment 4, where classification performance is shown as a function of the number of member networks in the committee. Figure 3.11 is for the case where cross-validation was used, while Figure 3.12 is given for the overtraining case.

In Figure 3.11, we see that in all the cases shown, the curves oscillate around 56%. In all cases the curve starts at its peak with few members, it then decreases until it reaches the minimum after which it again starts to rise. We note that as the size of the subsets decrease, the amplitude of the oscillation increases and the frequency decreases. This leads to the conclusion that by using less training data, better results may be achieved, but also inferior results if the number of committee members is wrong. The fact that the small data sets' plot has a low frequency means that the 10% curve crosses the 56% line for the first time at around 60 members. Using this knowledge, one may safely create a committee that will perform at as well as, or better than, the 90% case by using fewer committee members. In short, if neural networks are trained using cross-validation and same initial weight vectors, it is much better to use committees with

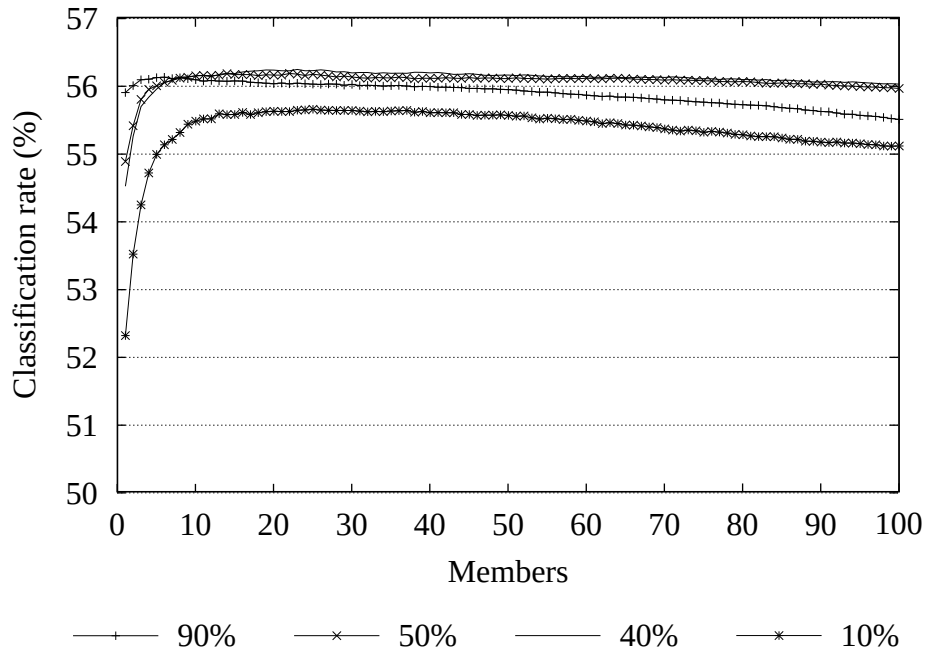

Figure 3.12: Results for committees using overtraining and the same initial weights.

few members, where the members are trained on little data.

All these networks start training from the same initial position. As the size of the subsets decrease, the subset's minimum moves further away from the original set's minimum. This greater distance between the two minima causes larger variance on the original distribution and better generalisation ability, which leads to better performance (larger amplitude). As more of the poorer networks are added, the performance obviously decrease. The networks with the better generalisation takes longer to decrease, leading to oscillations with lower frequency.

The reason for the rising of the plot after reaching the minimum, is not quite clear. It might be that the networks that perform the worst found a different mode in the error function that gives good results on a small piece of the data, but poor results on the largest group of data.

The graph of the committees using overtraining and starting from the same initial

position, Figure 3.12, has basically the same characteristics as the graph in Figure 3.8, which is plotted for committees using overtraining and different initial positions. The main difference being that in Figure 3.12 the performance degrades noticeably when more than 50% data is used. As in Figure 3.8, the curves of 40% and 50% data stay almost constant after the performance peak has been reached.

Comparing Figure 3.11 and Figure 3.12, we see that although Figure 3.11 oscillates, it stays within 0.25% of 56%. This is approximately the same as the best curve (50% data) in Figure 3.12. According to this result, there is no definite winner according to the achieved performances. More or less the same results may be achieved using either overtraining and 50% data, or cross-validation with any amount of data. By using cross-validation with 10% data, however, it is possible to achieve the same results as when using overtraining, with much shorter training times.

Comparing Figures 3.5 and Figure 3.9, i.e. the graphs for classification performance vs. subset size for cross-validation, we find that the curves look very similar. Both start out low, reach a peak at 50% and decrease again. The difference between the two is that the whole graph for the same initial weights is shifted up by about 0.2%. In addition, as subset size decreases, the variance for the graph of different initial weights increases much more than it does for the graph of same initial weights. The networks that trained from the same initial position have lower variance because the cross-validation does not allow them to move very far from the optimised initial position. The networks trained from different initial positions each find a different minimum, of which some may be very bad, giving the large variance. For the same reason the networks of Figure 3.9 perform better, because they have almost the same solution, giving a much better average performance.

When we compare Figure 3.7 and Figure 3.11, we see that the plots in Figure 3.11 consistently stay at 56%±0.25%. In Figure 3.7 the plots from 40% data and more all begin at 56% or higher, after which they quickly improve to around 56.75%. After this peak, the plots of 40% and more decrease slowly to a minimum of 56%±0.2%. These

plots all start to cross the 56.25% line at around 45 member networks. This implies that for data subset sizes of 40% or more, any number of member networks below 45 may be used, and the committee will outperform the method of cross-validation with same initial training positions.

Figure 3.6 and Figure 3.10 have the same form, but there are a few differences. The plot for the same initial weights, Figure 3.10, starts about 1% lower than the different initial weights' curve. Both curves end at roughly the same performance value. This manifestation of equal performance with small data sets can be explained by the fact that these small subsets normally have probability distributions that differ considerably from the real distribution. Because of this difference, the initial weights used for the networks that train from the same initial position, may not even be close to the subset's minimum error. The effect is that these initial weights are the same as random initial weights, resulting in networks that produce the same results as networks that all started training from random values. The variances on these classification performances are about the same.

When we compare Figure 3.8 with Figure 3.12 we see that the committees trained with 10% data perform almost the same when the same initial weights are used for the networks than it does when random initial weights are used. However, if we use 40% or more data, the random initial positions give consistently better results. The classification using few member networks are better, the peak performance is far better and the tail does not fall as far as it does for the corresponding same initial weights plots. The networks of Figure 3.8 for 40% data and more perform better than Figure 3.12's, because they explore a larger area of weight space, resulting in networks that perform well on different parts of the training data. Figure 3.12's networks all have more or less the same solution, leaving the possibility that all the networks perform poorly on some part of the data.

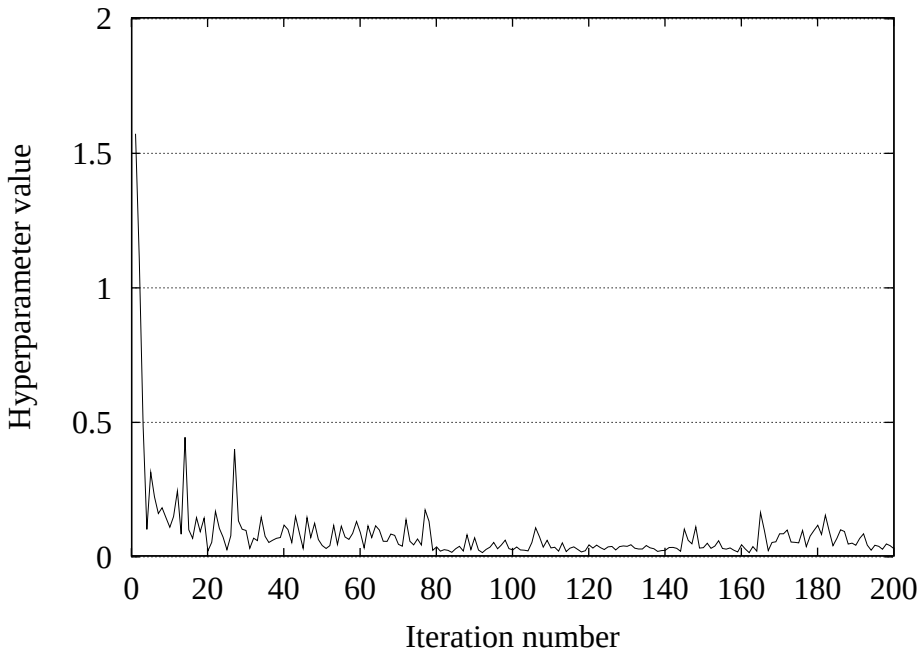

Figure 3.13: Results for the hybrid Bayesian method.

3.5.5 Experiment 5 : Hybrid training with CGD and Bayesian learning

As explained in Section 3.4.5, the hybrid method tested in experiment 5 initially uses a conjugate gradient descent training method to find suitable initial weights after which the Bayesian learning method is used to complete the network's training. The value of the third hyperparameter (standard deviation of the second layer weights) of the Bayesian learning is given in Figure 3.13. As for experiment 1, iteration number is given on the horizontal axis while the hyperparamer's value is on the vertical axis.

The first thing we notice about the graph, is that the hyperparameter for this experiment starts at 1.6 as opposed to the 0.5 of experiment 1's hyperparameter. This is because 0.5 was specified as the initial value for the hyperparameters of experiment 1, while the initial weights were specified for experiment 5. The specified value of 0.5 for the hyperparameters, is an arbitrary value that has proven to work quite well for Bayesian learning. The large initial value for experiment 5's hyperparameters is the

result of the distribution of the initial weights calculated by the conjugate gradient descent method.

The initial value of the hyperparameters is not really relevant in determining whether the network will make accurate predictions. A much greater indication is the stability of the hyperparameters. Another important factor is how fast the value stabilises below 0.4. Fewer iterations used to stabilise the hyperparameters, means that fewer iterations need to be completed, which may reduce training times significantly.

Comparing Figure 3.3 with Figure 3.13, we see that the plots on both graphs decrease very rapidly during the first few iterations. In Figure 3.3 the iterations from about 3 and onwards may be used effectively. There are a few larger values further down the line, but these are all well below the 0.4 limit. The iterations from experiment 5, are usable from about iteration 5. Around iterations 15 and 25, there are two peaks above 0.4, but as explained earlier, including a few networks that break this limit does not necessarily do any harm. One must also keep in mind that the training data used were well defined. In real world data, it is quite possible to get networks whose hyperparameters never settle below 5, and still produce accurate results.

Experiment 5 was repeated a number of times to get a representative average for the training times. The conjugate gradient descent algorithm took about 6 seconds and the Bayesian learning about 14 minutes. On average, Bayesian learning took about 4 seconds per iteration. The average classification accuracy obtained with the hybrid method was 57.33% correct, with a maximum of 57.48% correct. Compared to experiment 1's result of 57.76%, it shows that the hybrid method does not improve the Bayesian learning's performance.

3.5.6 Results summary

Figure 3.14 is a summary of all the results obtained, plotted as a mean and with variance. For experiments 3 and 4 only the best results are shown. The key is as

follows:

1. Experiment 1: Bayesian learning.
2. Experiment 2: Conjugate gradient descent training, use all the training data, use cross-validation.
3. Experiment 2: Conjugate gradient descent training, use all the training data, use overtraining.
4. Experiment 3: Conjugate gradient descent training, use subsets of the training set, use cross-validation, random initial positions.
5. Experiment 3: Conjugate gradient descent training, use subsets of the training set, use overtraining, random initial positions.
6. Experiment 4: Conjugate gradient descent training, use subsets of the training set, use cross-validation, same initial positions.
7. Experiment 4: Conjugate gradient descent training, use subsets of the training set, use overtraining, same initial positions.
8. Experiment 5: Hybrid method

In Figure 3.14 we see that the Bayesian network achieves the best classification performance. The hybrid method is second best, while all the methods using gradient descent perform much worse.

As expected, number 3 performs better than 2. This performance increase is the result of the variance averaging effect that committees have on overtrained networks. For this same reason number 5 is better than 4. Numbers 3 and 5 are better than numbers 2 and 4 by less than 0.1%. This may be expected if one extrapolates the plots of Figure 3.7 and Figure 3.8 to 100% data sets.

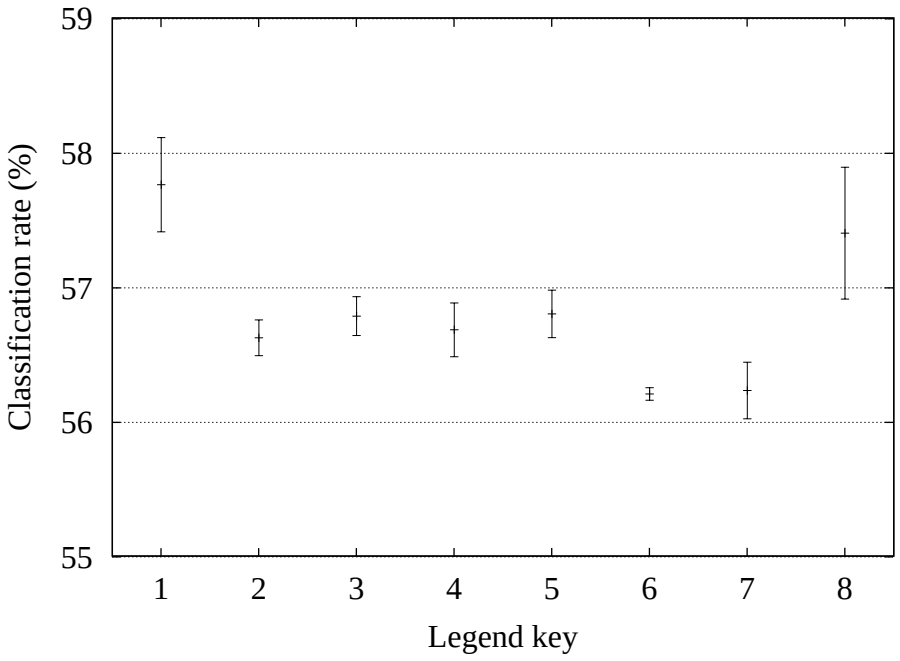

Figure 3.14: Summary of the best results obtained in each experiment.

As in experiments 2 and 3, the overtraining case of experiment 4 also outperforms the cross-validation case, although only slightly. Another point to note is that the variance for the cross-validation case (number 6) is very small. Resampling the training data resulted in subsets with slightly changed distributions, whose minima had small offsets from the original set's. As the networks started training from an already optimised position, the cross-validation prevented them from straying too far from the original set's minimum. This same change in distribution caused the overtrained committees to have much larger variance, as there is no mechanism to prevent the networks from moving far from the original set's minimum. Even so, the variance does not differ much from the variances of numbers 2 to 5.

The two methods using Bayesian learning (1 and 8) have much larger variances than the gradient descent methods. These larger variances are the result of searching a much larger area of the weight space for solutions, i.e. the wider search produces more diverse networks. When gradient descent is used, the variance is caused by networks that have solutions mostly around the same local minimum, while the Bayesian networks have results at different minima.

Although resampling does give a performance improvement, this improvement is not significant enough to warrant the time and effort required to implement it. What does give a useable improvement, is using overtrained neural networks in a committee. Implementing overtrained networks in a committee is actually easier than implementing a single network using cross-validation. The drawback is that overtraining a network takes much longer than using cross-validation. A second disadvantage is that an overtrained neural network will perform much worse than one using cross-validation should it become necessary to use a single network.

The main reason for Bayesian learning's superior performance is that the weight space is explored much more completely than with gradient methods. Bayesian learning accomplishes this by using the weights' distribution to sample new network parameters from. Resampling attempts to accomplish the same by removing data from the training set in order to introduce variance in the network parameters.

3.5.7 Trained decision boundaries

Figures 3.15 through 3.17 show the decision boundaries for the five dimensional data as 2-D projections. The boundaries were created by Bayesian learning, CGD using cross-validation on all the data and CGD overtraining on all the data respectively. The boundaries shown for conjugate gradient descent are those for individual neural networks, rather than those for committees.

When we compare the three figures, it is obvious that the Baysian method comes much closer to implementing the theoretical boundaries than the gradient descent method does. We can also see that Bayesian learning has much sharper boundaries, while those created by gradient descent are more fuzzy. This is caused by the fact that the three features not plotted in the figures, are irrelevant in determining a sample's class (see section 3.3). Bayesian learning manages to mostly ignore these inputs, while CGD is confused by them.

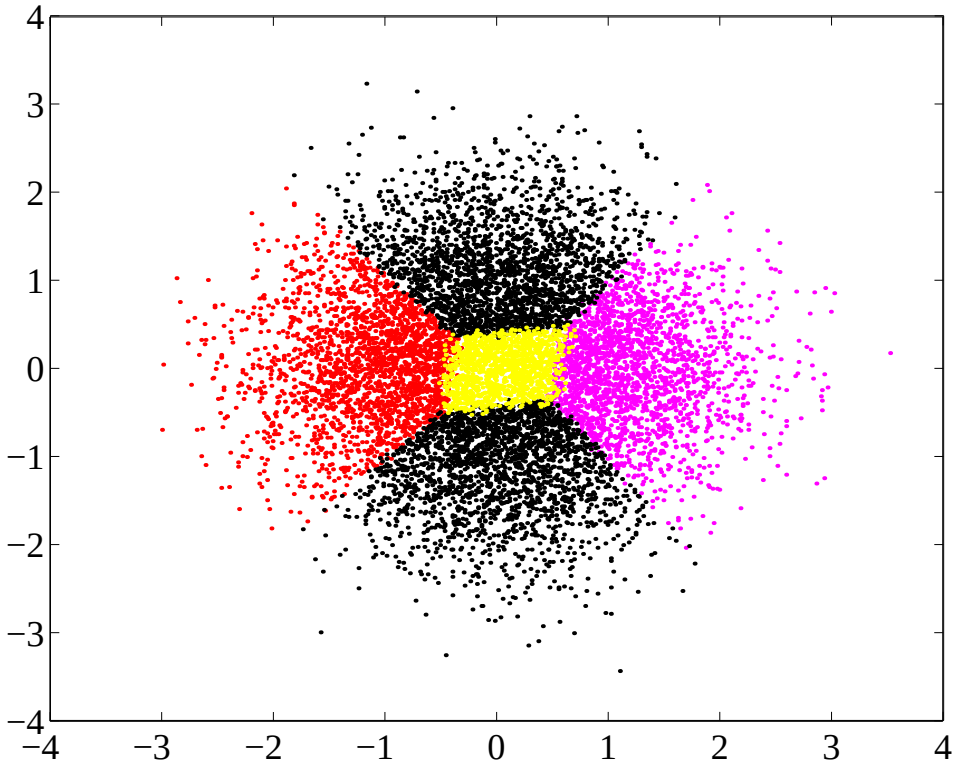

Figure 3.15: A 2-D projection of the decision boundaries produced by Bayesian learning in experiment 1. Different shades represent different classes.

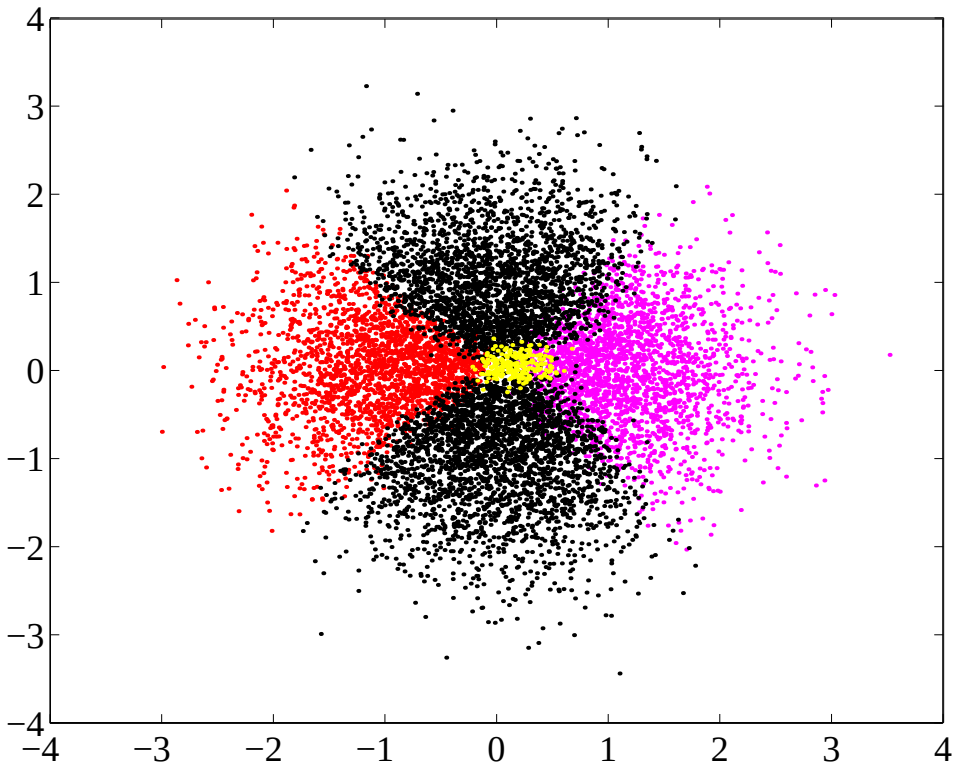

Figure 3.16: A 2-D projection of the decision boundaries produced by CGD in experiment 2 for cross-validation without committees. Different shades represent different classes.

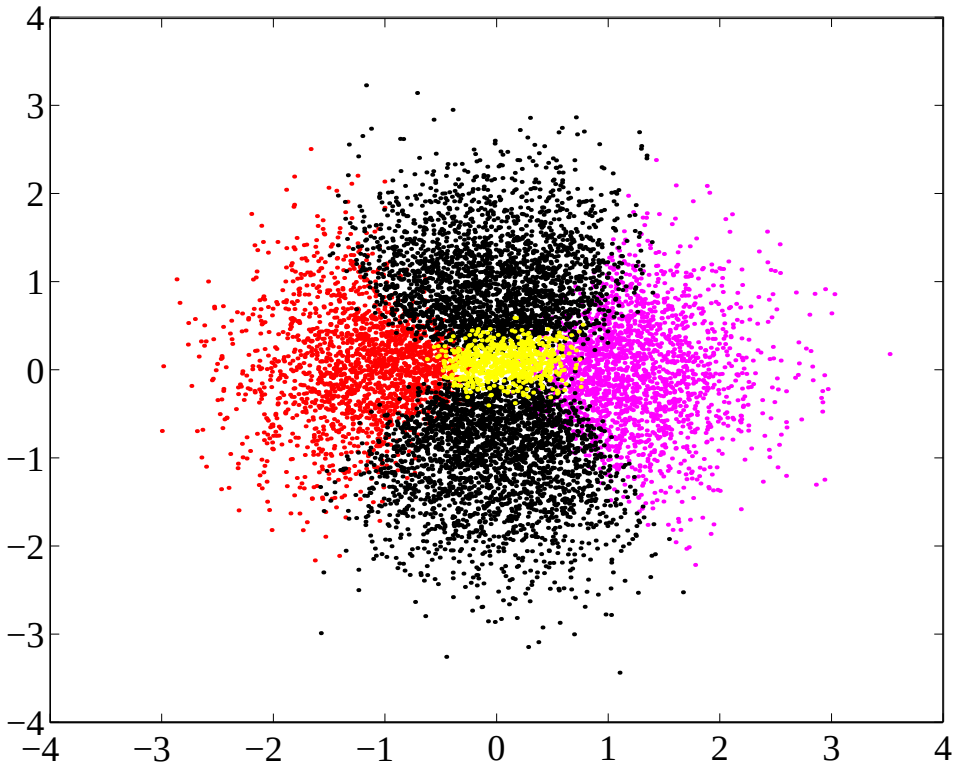

Figure 3.17: A 2-D projection of the decision boundaries produced by CGD in experiment 2 for overtraining without committees. Different shades represent different classes.

If we look at the centre region of Figure 3.16 and Figure 3.17, we see that this region is very small compared to the centre region in Figure 3.15. This small centre region means that while the conjugate gradient descent method may seem to perform well on the whole, it performs poorly on data from the centre class. This situation is far from ideal as one would normally prefer that the network gives each class a reasonable chance.

3.6 Verfication

To test the results obtained in the experiments on the five dimensional Gaussian data, some experiments were repeated on a more difficult problem.

3.6.1 Topology

Again, to make direct comparison possible, the same architecture was used for all the networks. Each network consisted of two input neurons, ten hidden neurons and two output neurons. Compared to network topologies usually used to solve the problem, the chosen number of hidden neurons is small. Few hidden neurons are used to limit the number of weights, as the training set is already small, and it becomes extremely small when subsets contain only 10% of the samples.

3.6.2 Data

The problem chosen for verification was the two intertwined spirals proposed by A. Wieland[2]. This is a two class problem, consisting of two spirals coiling aroung the origin and around each other. Each class is represented by 97 samples, where a sample consists of the x and y coordinates of a point.

The purpose of the problem is to obtain 100% classification accuracy, and as such, the testing set is identical to the training set.

3.6.3 Experimental procedure

The verification tests repeated some experiments that achieved the best results on the five dimensional data.

Experiment 6: Bayesian learning on spirals

A neural network was trained using the method of Bayesian learning. This training was continued for 100 iterations on all the available training data. Iterations 10 through

[2]Posted to "connectionists" mailing list by Alexis Wieland of MITRE Corporation.

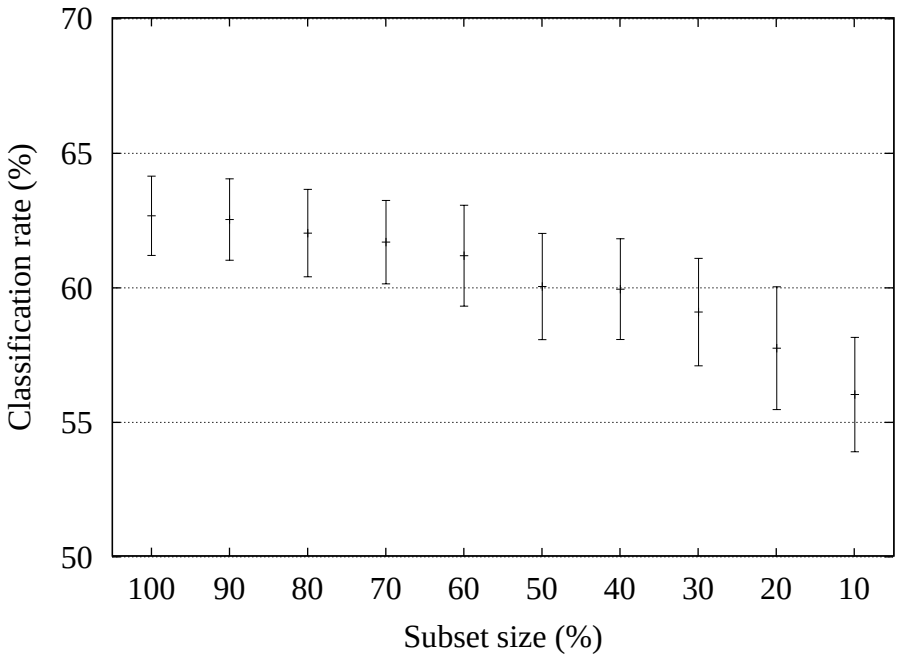

Figure 3.18: Results for CGD networks overtrained on the spiral data. The average and standard deviation of the networks' performances are shown as a function of the size of the training subset used.

100 were used to test the network's classification performance.

Experiment 7: CGD on spirals

Neural networks were trained using the gradient descent method on subsets of the available training data. The size of the training subset was varied from 100% to 10% of the complete training set in 10% steps, and 100 networks were trained for each subset size. Overtraining was used for all the networks because the training and testing sets are identical.

All the networks were tested on the complete testing set.

3.6.4 Results

Figure 3.18 shows the results of training a conjugate gradient descent network on the two spiral problem. The graph has the same form as those obtained in the experiments with the Gaussian data when the networks were overtrained.

Table 3.1: Results for the spiral problem.

	Classification rate (%)
Bayesian learning	64.43 ± 3.5
Conjugate gradient descent	62.64 ± 1.5

Table 3.1 lists the results of the Bayesian learning and the best performance from Figure 3.18. The results are not very good and one expects much better. The poor performance can be ascribed to an insufficient network model, which was chosen to give the networks training on small subsets a reasonable chance for learning the problem. Network topologies with more hidden units can be expected to increase the performance significantly if the network is to be trained on the complete training set, especially for Bayesian learning, as Neal claims that the complexity of a Bayesian network should be dictated by the time available for training, rather than by the number of available training samples[3].

Figure 3.19 shows the trained decision boundary of the Bayesian method for the two spiral problem, and Figure 3.20 shows the decision boundary created for the same problem by the gradient method.

The results obtained on the spiral problem are consistent with those obtained on the Gaussian data, i.e. Bayesian learning outperforms conjugate gradient descent by a small margin, and its standard deviation is about double the standard deviation of the gradient method. Also, the decision boundaries created by Bayesian learning seem more natural than those created by the gradient descent networks, although they are not necessarily better.

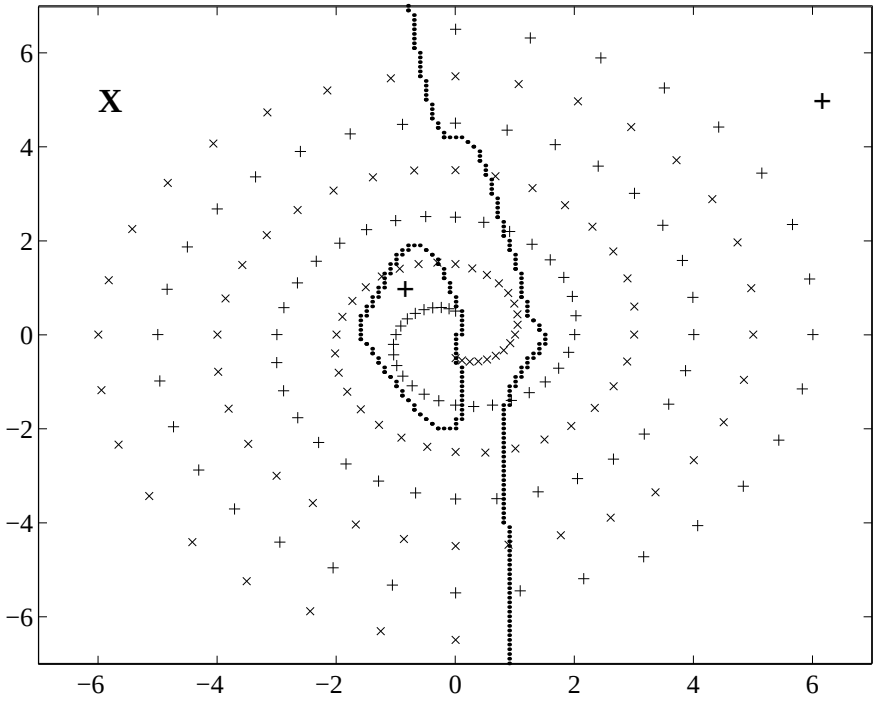

Figure 3.19: Decision boundaries produced by the Bayesian method for the two spirals problem.

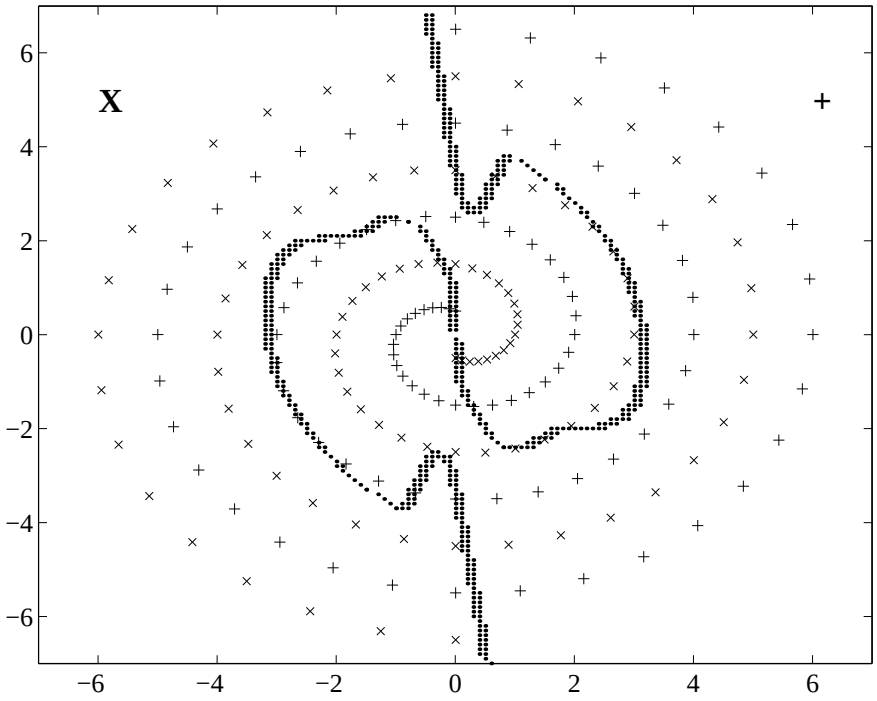

Figure 3.20: Decision boundaries produced by the CGD method for the two spirals problem when training on all the data.

3.7 Statistical significance

To be more confident in accepting the results showing that Bayesian learning is better than conjugate gradient descent, a test for statistical significance was performed. The test was performed for a 99% level of significance (that is, we are 99% sure it is not an error to accept that Bayesian learning is better than conjugate gradient descent).

To reach this level of certainty, we have to reject a hypothesis H_0 according to a test statistic:

$$t > t_\alpha, \tag{3.29}$$

where

$$t = \frac{(\overline{C}_{BL} - \overline{C}_{CGD})}{s_{\overline{C}_{BL} - \overline{C}_{CGD}}}. \tag{3.30}$$

$\overline{C}_{BL}$ is the sample mean for the performances obtained from Bayesian learning, and $\overline{C}_{CGD}$ is the sample mean for CGD's performances.

H_0 states that gradient descent will be accepted as the better method as long as the following condition persists:

$$H_0 : \mu_{BL} - \mu_{CGD} \leq 0 \tag{3.31}$$

Equation (3.30) is solved using equations (3.32) and (3.33) and constants (3.34) through (3.38):

$$s_{\overline{C}_{BL} - \overline{C}_{CGD}} = \sqrt{s_{pd}^2 (\frac{2}{n})} \tag{3.32}$$

where

$$s_{pd}^2 = \frac{(n-1)(s_{BL}^2 + s_{CGD}^2)}{2n-2}, \tag{3.33}$$

and

$$n = 100 \tag{3.34}$$

$$\overline{C}_{BL} = 0.5776 \tag{3.35}$$

$$s_{BL} = 0.0035 \tag{3.36}$$

$$\overline{C}_{CGD} = 0.5678 \tag{3.37}$$

$$s_{CGD} = 0.0014. \tag{3.38}$$

A standard t distribution table gives t_α for $\alpha = 0.01$ as:

$$t_\alpha = 2.326, \tag{3.39}$$

and t is calculated as:

$$t = 25.997. \tag{3.40}$$

Equation (3.40) indicates that the results obtained are significant.

Chapter 4

Summary and conclusion

In this dissertation, experiments were performed to compare the classification accuracy of MLP neural networks trained with the Bayesian learning method with those trained using conjugate gradient descent methods. To implement the Bayesian learning, software written by R. M. Neal [3] was used. The software implementation used for the conjugate gradient descent was written by J. E. W. Holm [2]. This software implements the ALECO-2 algorithm, which is known as a good method of doing conjugate gradient descent learning.

The second goal was to try and improve the classification performance of the conjugate gradient descent method by resampling the training data set and combining networks trained on the resampled data in a committee. Finally an experiment was done to try and speed up the Bayesian learning method while still keeping its classification performance by combining the gradient descent method with the Bayesian method.

These experiments were performed because no quantitative comparison between Bayesian learning and conjugate gradient descent training methods for neural networks could be found in the literature.

Comparing the Bayesian learning method with the plain conjugate gradient descent

method showed that the Bayesian learning achieves between 1 and 1.5% better classification accuracy on the same training data, however, the Bayesian learning does take a lot longer to train than the gradient descent, so the decision of which method to use would depend, as most other decisions about neural networks, on the importance of prediction accuracy and available training time. It must be noted that in the data used, the classes overlapped extensively. If classification needs to be done on classes that do not overlap as much, the gradient descent method may come even closer to the performance of the Bayesian learning.

Resampling the training set and using a number of neural networks in a committee proved to perform slightly better than just merely using conjugate gradient descent on the whole training set. This method of improvement seems viable, since the only extra work apart from using the gradient descent method, is to create smaller data sets from the complete training set, and to combine the resulting neural networks in a committee. Training more networks shouldn't be a problem, because it is standard procedure to train a number of networks anyway in order to find the best one. Another advantage is that the smaller training sets result in shorter training times.

An experiment was done where the data was resampled, as above, but the individual neural networks were forced to start training from a specific position in weight space. This position was found by training a network on all the data, and using the final weights as the starting position for the other networks. The idea was that the data resampling would cause small shifts in the minimum represented by the data, which would make the networks find almost the same solution for the problem. These shifted solutions would then cancel some bias contained in the other network's solutions, which could be expected to improve the combined ability to make predictions. It proved not to be the case. The networks trained with the same initial weights performed worse than the gradient descent method trained on all the data.

The three methods discussed above, the conjugate gradient descent method, the resampling method, and the resampling method using identical initial weights, were all

repeated with the difference that the neural networks were allowed to overtrain on their training data. The reasoning is that the variance averaging effect of using neural networks in a committee is lost when using cross-validation to stop the networks' training early. The overtraining would cause each network to have more variance, but the combined networks would have less variance than non-overtrained networks combined. This was expected to increase the performance of the committee of networks, and it did indeed increase the performance. In all three cases the classification performance increased by about 0.15%. However, the gain in accuracy was offset by the much longer training times. Using overtraining resulted in training times that were 6 to 10 times longer than for cross-validation on the same data set. Very few problems would justify such an increase in training time for only 0.15% performance increase.

The final experiment used both the gradient descent and the Bayesian learning methods in an effort to create a hybrid method that would outperform both methods. It was noted during preliminary tests that the Bayesian method's performance was not much influenced by the initial conditions. Poor initial conditions only caused it to use more iterations to find the solution. The intention was to use the gradient descent method to find weights that were closer to a solution than just choosing random initial weights. It was hoped that this would give the Bayesian network an advantage to settle quickly on a solution, while still achieving the same performance. The results of the hybrid method proved to be much better than the gradient descent's, but not as good as the Bayesian method's. The hybrid method also takes as long as, or longer than the Bayesian method to find a solution.

Should the classification performance of Bayesian learning be required in a problem, it would be best to use pure Bayesian learning. If speed is required, conjugate gradient descent is the better option. The proposed hybrid method only decreases the performance of Bayesian learning without adding any advantages. It must be said that the experiment on the hybrid method used here did not include tests with the weights' probability distribution after the initial weights were specified. By specifying the initial weights and experimenting with the hyperparameters, the hybrid method may reach

the same performance as Bayesian learning, or may find solutions faster.

The theoretical maximum classification rate possible on the data used, is 58.53 %. With only 60 examples of each classes' distribution used for training, the networks performed rather well, achieving up to 57.76%, considering that few points may not represent the real distribution very well.

For verification of the results obtained, the Bayesian learning experiment and a conjugate gradient descent experiment were repeated on the two spiral benchmark problem. The CGD experiment used in the verification started with random initial weights and overtrained on the spiral data. More tests on benchmark problems will have to be performed before any claims can be made, but from the verification it appears as if the results obtained on the five dimensional data may be generalised to other problems for which MLP neural networks are effective.

The results obtained are statistically significant, which means that one can be reasonably sure that Bayesian learning is indeed better than conjugate gradient descent. Whether Bayesian learning is the best option to use for a specific problem, will be determined by other factors, and not by the statistical significance of the results.

Bibliography

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press Inc., New York, USA, 1995.

[2] J. E. W. Holm, *Sampled optimization for training perceptron neural networks*, Ph.D. thesis, University of Pretoria, Pretoria, South Africa, 1999.

[3] R. M. Neal, *Bayesian learning for neural networks*, Springer-Verlag, New York, USA, 1996.

[4] D. Husmeier, W.D. Penny and S.J. Roberts, "An empirical evaluation of Bayesian sampling with hybrid Monte Carlo for training neural network classifiers," *Neural Networks*, vol. 12, no. 6, pp. 677–705, 1999.

[5] W.D. Penny and S.J. Roberts, "Bayesian neural networks for classification: how usefull is the evidence framework?," *Neural Networks*, vol. 12, no. 9, pp. 877–892, 1999.

[6] W. D. Nortje, J.E.W. Holm and E.C. Botha, "Performance comparison between different sampling methods for multi-layer perceptron neural networks," in *Proceedings of the eleventh annual symposium of the Pattern Recognition Association of South Africa*, Z. Fazekas, G.R.J. Cooper and F. Aghdasi, Ed., Broederstroom, South Africa, November 2000, University of the Witwatersrand, Johannesburg.

[7] J.E.W. Holm and E.C. Botha, "Sampled optimization improves on purely stochastic neural network training," in *Proceedings of the second ICSC symposium on*

Neural Computation, H. Bothe and R. Rojas, Eds., Berlin, Germany, May 23 - 26 2000.

[8] D.A. Karras and S.J. Perantonis, "An efficient constrained training algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1420–1434, Nov 1995.

[9] J.E.W. Holm and E.C. Botha, "Leap-frog is a robust algorithm for training neural networks," in *Network: Computation in Neural Systems*, vol. 10, no. 1, pp. 1–13. Journal of Physics Publishing, 1999.

[10] R.M. Neal, "Bayesian training of backpropagation networks by the hybrid Monte Carlo method," Tech. Rep. CRG-TR-92-1, Connectionist Research Group, Department of Computer Science, University of Toronto, April 1992.

[11] D. Cohn and G. Tesauro, "How tight are the Vapnik-Chervonenkis bounds?," *Neural Computation*, vol. 4, pp. 249–269, 1992.