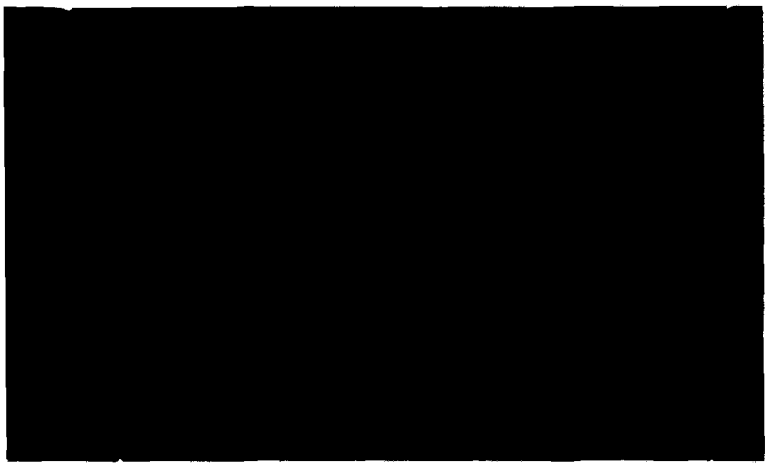
A probabilistic approach to sound classification

van Loon, R.

Award date:
2007

U/e technische universiteit eindhoven



/ faculteit elektrotechniek

Eindhoven University of Technology
Department of Electrical Engineering
Signal Processing Systems

TU/e technische universiteit eindhoven

A Probabilistic approach to sound Classification

by R. van Loon

Master of Science thesis

Project period: augustus 2005 – augustus 2007
Report Number: 18-07

Commissioned by: GN Resound

Supervisors:
(TU/e) Prof.dr. R.M. Aarts
Dr.ir. B. de Vries

Additional Commission members:
(TU/e) Dr.ir. M. Mischi

Contents

List of Figures

List of Tables

Summary

In this thesis it is investigated if a sound classifier that is built around a two-layer perceptron neural network can be improved by using Bayesian inference in the learning process. The sound classifier estimates the power fraction of three different sound classes compared to the total signal power in a segment of an audio signal. The three sound classes are speech, music and noise. Special about this sound classifier is that it uses mixtures of the sound classes in the training process to improve the performance in situations where the applied input audio signal is also a mixture.

During the design process of the sound classifier several choices have to be made about different properties of the neural network. For example about the size of the network and the types and number of features that will be used. A method for making those choices is testing the performance of several trained neural networks, with different initial conditions, on a set of test data. The network with the best performance on the test data set is expected to have the best generalization performance and should therefore be used in the classifier. A disadvantage of such method is that less data becomes available to train the network, because the data that is reserved for testing the performance cannot be used for training the neural network. Another disadvantage is that it is practically impossible to test all combinations of the different properties, which makes it necessary to make subjective choices.

Bayesian inference can help with making more objective choices about the size of the neural network, the weight parameters and reliable features. Also with this approach no separate test data set is needed to find a good network. All available data can thus be used for training the neural network of the classifier. In the Bayesian inference approach the model properties and parameters are not represented by single values, but probability distributions are used to represent the 'knowledge' about them. The training process starts from a situation where the probability distributions represent the prior 'knowledge' that the designer has. During the training process the 'knowledge' about the properties and parameters changes. Posterior probabilities represent the 'knowledge' about the properties and parameters after the training process.

In this thesis an hierarchical Bayesian framework consisting of different levels of inferences is used in the training process of the sound classifier. An approximation method called *Evidence Method* is used to express probability distributions over the network properties and parameters. In the Evidence Method probability distributions are approximated by Gaussian distributions. Before the Bayesian framework is applied to the sound classifier it is first applied to a small synthetic problem that has similarities with the sound classifier. It appears that Bayesian inference with the Evidence Method improves the results on the synthetic problem. For the real sound classifier performance improvements with the applied Bayesian framework are less obvious. We think that these poor improvements are caused by problems in the calculation of the covariance matrices, which are used in the Gaussian approximations. Also the assumption that all probabilities in the model can be expressed in a Gaussian form may be too optimistic in the case of the real sound classifier. For finding a good model for the sound classifier an alternative method is also investigated. This method is called "Pruning with a Laplace prior" and does not make use of large covariance matrices. This method leads to better results.

Chapter 1

Introduction

1.1 Background

In many sound processing applications it is useful to know the composition of the incoming audio signal in terms of different sound classes. For example in hearing aids, different signal processing algorithms can be turned on or off depending on the presence of speech in the incoming signal. Frequently used sound classes are speech, music and noise.

Several examples of sound processing applications that approximate the composition of an audio signal can be found in the literature. Surendran [13] presents a speech detector that estimates the probability of presence of speech in an incoming audio signal. A classifier presented by Lu [5] discriminates five audio classes: silence, music, background sound, pure speech and non-pure speech that includes speech over music and speech over noise. The classifier that will be further investigated in this thesis is presented in Kates [4]. This sound classifier estimates the power fractions of respectively speech, music and noise compared to the total signal power in an audio segment.

What the above examples have in common is that they make use of learning machines. Learning machines try to find and extract rules or regularities from example data and use this information to emulate the underlying generation process of the example data. This is called *machine learning*. It is said that a learning machine is *trained* by examples. Example data for a sound classifier can be made by extracting spectral or temporal information from audio files. Extracting this information to make it ready for the learning machine is called *pre-processing*. An example of temporal information is the *Zero Crossing Rate* (ZCR), which measures how often a signal crosses the zero level in a certain period of time. If the Zero Crossing Rate is constant for a period of time this can indicate that there is a certain rhythm in the signal. A rhythm increases the probability that the signal is music.

A method that has won in popularity over the past few decades in the field of machine learning is called Bayesian inference. Bayesian inference makes use of probabilities to express uncertainties in the training process of a learning machine. Examples of uncertainties are the type of learning machine, the size of the learning machine and the input information that is needed by the learning machine to make a good prediction for unseen data.
Authors who have published work about Bayesian inference in the field of machine learning are Bishop [2], Thodberg [14], Mackay [8], Penny [11] and Tipping [15].

1.2 Problem definition

The goal of this project is to investigate if the sound classifier that is presented by Kates [4] can be improved by making use of Bayesian inference in the learning process. Possible improvements are expected because Kates has mainly focused his paper on the use of mixtures of sound classes and the use of a new feature type in the learning process of the classifier. He did not focus much on the ideal size of the neural network and the number of inputs that are needed for a good performance. The Bayesian inference framework that will be used in this thesis is presented by Mackay in [8]. It is an hierarchical framework where Bayesian inference is used on multiple levels of model optimization.
The following possible benefits of Mackays Bayesian framework in the learning process of the classifier will be investigated:

1. Regularization of the weight parameters in the classifier's neural network.

   Regularization leads to smaller weight parameters, which results in smoother transitions on the network outputs. Smoother transitions on the network outputs prevent the network to fit to noise in the training data. This improves generalization and thus the performance on other unseen data.

2. Automatic Relevance Determination of the inputs (features) presented to the classifier

   The Bayesian framework has the ability to automatically suppress the influences of irrelevant features.

3. Determination of the size of the classifier's neural network.

   A neural network that is too small will not be able to fit to all input-output relations in the training data (*underfitting*). A neural network that is too large will possibly fit to the noise in the training data set too and will then perform poor on other unseen data (*overfitting*). The Bayesian framework has the possibility to automatically find a well sized neural network.

4. Indicating the prediction error on the classifiers outputs.

   Probability distributions can be estimated on the network outputs. A wide distribution on an output means that the network is not very sure about its prediction and a small distribution means that the network is expected to be sure about its predictions.

In Mackays' Bayesian framework an approximation method called *Evidence Method* is used for approximating Gaussian probability distributions. This method approximates Gaussian distributions by second order Taylor expansions around their modes.

1.3 Report outline

Chapter 2: Machine learning,

An introduction is given in how a model can be estimated by observing sample data with given input-output relations. The Bayesian inference principle is introduced here. In Bayesian inference every uncertainty in a model is given a probabilistic interpretation. After a common introduction of model estimation the type of model that is used in the rest of this thesis is introduced. This type of model is called a Neural Network.

chapter 3: Bayesian inference and neural networks,

This chapter describes how the Bayesian inference principle can be used for optimizing neural networks. Bayesian inference is used on different levels of model optimization. A few advantages of Bayesian inference compared to conventional methods are given.

Chapter 4: Tests on a synthetic problem,

The theory, described in chapter 2 and 3, is tested on a synthetic problem. The synthetic problem has a lot of similarities with the sound classification problem. It is also trained on mixtures of three different input patterns. The synthetic problem is used to get practical experience with the theory. The sound classification problem is a very big problem and the simulations are very time consuming.

Chapter 5: Bayesian sound classification,

The theory of Bayesian inference is used here to improve Kates' sound classifier

Chapter 6: Conclusions and Recommendations

In this chapter conclusions are given about the improvements in Kates' classifier by using Bayesian inference methods.

Chapter 2

Machine learning

In machine learning we try to induce patterns, regularities, or rules from data. If those patterns, regularities or rules have been found, a model can be made, which approximates the behavior of the original underlying process of the data. With such a model predictions can be made for new ('future') data points that have not been observed yet. Distinction can be made between regression and classification. In the case of regression the model outputs are continuous and in the case of classification the model outputs are discrete values that assign the inputs to a specific class. Another distinction can be made between supervised and unsupervised learning. In the case of supervised learning a data set is used with known input to output relations. In the case of unsupervised learning the outputs are unknown. The thesis will be focussed on supervised learning of regression problems. This chapter is divided in two parts. The first part explains how a model can be fit to given data in general, without taking into consideration the type of model and its internal structure. The second part of the chapter explains a certain type of model called neural network.

2.1 Model parameter estimation

Given a data set $D = \{(\mathbf{x^n}, t^n)\}_{n=1}^{N}$,consisting of N relations between a vector of input values $\mathbf{x}$ and an output value t of a certain process. We would like to estimate a model that could have generated these outputs given the inputs. Thus approximate the underlying process of the given data set. A model can for example be written as $y(\mathbf{x}; \mathbf{w})$ where an outcome y depends on the input values $\mathbf{x}$ and a set of parameters $\mathbf{w}$. In this thesis we will be consistent with the notation as used in Bishop [2].
Let's start with a simple regression example with a single input and a single output. In Figure 2.1 an example of a regression of a sine function is shown. The circles represent given data points, which can for example come from measurements of an experiment. The horizontal axis represents the input x (x is a single value instead of a vector in this example) of the experiment and vertical axis then represents the measured output t of the experiment. The solid line represents a possible approximation of the underlying process of the given data points.
When the data points at the positions of the circles are used for training a model these data points are called training data. The inputs x are called the features and the measured outputs t are called the targets. During training values for the parameters $\mathbf{w}$ should be found that let the model output y reach the targets t.
In the next subsections three methods will be discussed for estimating the model parameters.

2.1.1 Least Squares Error Estimation

Values for the model parameters $\mathbf{w}$ can be found by minimizing an error function. A standard error function for a regression problem is called the sum-of-squares error function (SSE) and is

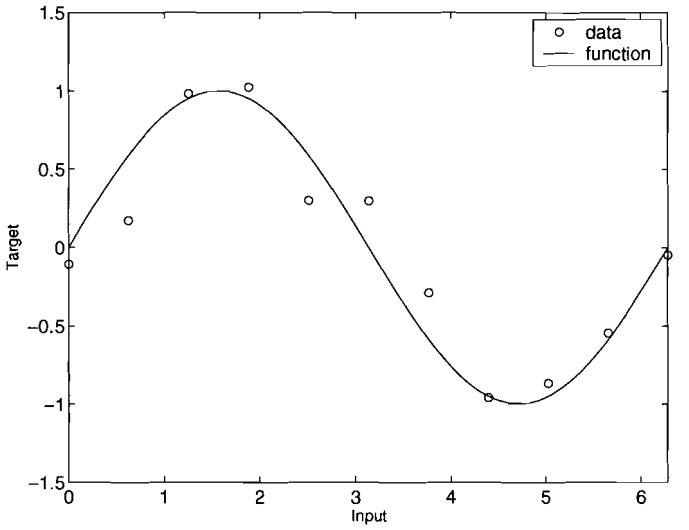


Figure 2.1: *Regression of a sine function. The data points at the positions of the circles can be used to train a model.*

written as

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x^n;\mathbf{w}) - t^n\}^2 \,. \tag{2.1}$$

The better the model output y reaches the target values t the smaller the error will be. Finding optimal values for the parameters by minimizing this error function is called Least Squares Error (LSE) estimation).

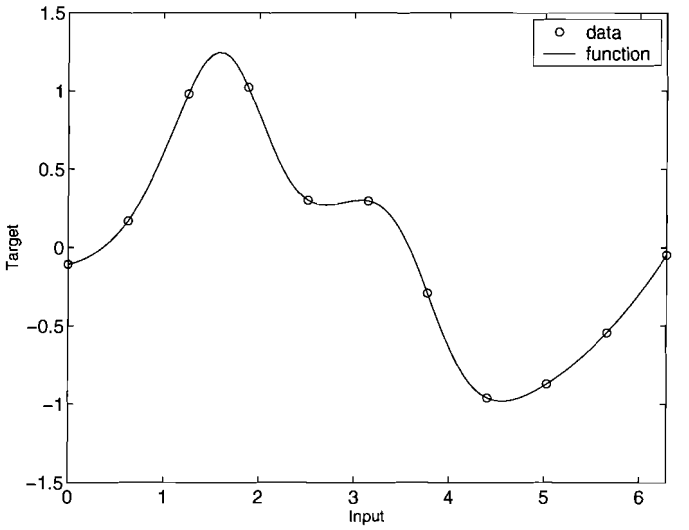


Figure 2.2: *Regression through data points originally coming from a noisy sine function. The used model is too complex. Because of the large complexity of the model it is able to fit the noise too. This results in poor generalization.*

Problems can be expected when a complex model is trained on a small training data set. A complex model is a model that is able to fit to a wide range of input-output relations. Most times

the more parameters the model has the more complex the model is. It is possible in such situation that the model tries to fit too well to the training data and therefore fits also the noise on the training data. In such a situation the model will give poor results on other unseen data from the same process. This effect is called *overfitting*. An example of such situation is shown in Figure 2.2.
A model that can be used to approximate the sine function can for example consist of a set of M basis functions $\phi_m(x)$, which are weighted and then summed to result in a model output. The equation is then given by

$$y(x; \mathbf{w}) = \sum_{m=1}^{M} w_m \phi_m(x) \, . \tag{2.2}$$

Regression of a sine function by using Radial Basis Functions is done in Tipping [15]. Here the type of basis function is not of much importance. What is important here is that we see that with more basis functions a more complex model can be built.
A common and very reasonable assumption is that data is typically generated from a smooth rather than a complex function. Smooth functions will typically have smaller weights. A way to prefer smaller weights over larger weights is to add a penalty term to the sum-of-squares error function. An example of such a penalty term is

$$E_W(\mathbf{w}) = \frac{1}{2} \sum_{m=1}^{M} w_m^2 \, . \tag{2.3}$$

This function is called a *weight decay* function or *regularizer*. The total error function is then given by

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w}) \, . \tag{2.4}$$

It is called the *penalized-least-squares (PLS) error* now. The parameter λ is used to control the strength of the regularization. The larger λ the smoother the network outputs will be. In the next chapter such a parameter will be called a *hyperparameter*, because it is used to scale lower level parameters $\mathbf{w}$ in the model.

2.1.2 Maximum Likelihood

The regression problem in the previous subsection can also be seen from a statistical viewpoint. Each target sample is then interpreted as a random sample coming from an unknown distribution. During training we try to find the probability distribution that would have most likely generated the target samples. The model output y then represents the mean of a probability distribution of the target values. A very common used probability distribution in regression problems is the Gaussian distribution. Several reasons for choosing a Gaussian distribution will be given later on in this subsection. For now we assume the distribution to be Gaussian with zero mean and a variance of $1/\beta$. The relation between a target value t^n and the model output y is then given by

$$t^n = y(\mathbf{x^n}; \mathbf{w}) + \epsilon \, . \tag{2.5}$$

The ϵ term represents the noise, which is drawn from the zero mean Gaussian distribution with variance $1/\beta$, written as

$$\epsilon \sim \mathcal{N}(0, 1/\beta) \, . \tag{2.6}$$

In the situation of the regression of the sine function the zero mean Gaussian in Equation 2.6 can be interpreted as a kind of measurement noise. Because of the zero mean the probability distribution of t^n can be written as

$$p(t^n | \mathbf{x^n}, \mathbf{w}, \beta) = \sqrt{\frac{\beta}{2\pi}} \exp \left\{ -\frac{\beta}{2} [t^n - y(\mathbf{x^n}; \mathbf{w})]^2 \right\} \, . \tag{2.7}$$

If it is also assumed that the data points are independent and identically distributed (IID) then the equation for the probability distribution over all data points becomes

$$p(D|\mathbf{w},\beta) = \prod_{n=1}^{N} p(t^n|\mathbf{x^n},\mathbf{w},\beta) = \left(\frac{\beta}{2\pi}\right)^{N/2} \exp\left\{-\frac{\beta}{2}\sum_{n=1}^{N}[t^n - y(\mathbf{x^n};\mathbf{w})]^2\right\} \equiv \mathcal{L}(\mathbf{w}) \quad . \tag{2.8}$$

The above equation can also be interpreted as a function of $\mathbf{w}$, because D and β are given. This function is then called the likelihood. The optimal values for the parameters $\mathbf{w}$ are the values that maximize this likelihood function. Finding the parameters that maximize the likelihood can be formally written as

$$\hat{\mathbf{w}}_{ML} = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \; . \tag{2.9}$$

A maximum of the likelihood can be found by minimizing the negative logarithm of Equation 2.8. The equation then evaluates to

$$-\ln p(D|\mathbf{w}) \propto \frac{\beta}{2}\sum_{n=1}^{N}[t^n - y(\mathbf{x^n};\mathbf{w})]^2 \; . \tag{2.10}$$

In this case minimizing the negative log-likelihood is the same as finding a minimum for the sum-of-squares error function. Thus by using a Gaussian distribution a statistical interpretation can be given to the parameter optimization process by using conventional methods as least squares error estimation. There are several other reasons for using a Gaussian distribution. Reasons based on the Maximum Entropy Principle are given in Jaynes [3]. Another reason is the Central Limit Theorem, which says that the probability distribution of the average of m independent measurements of some quantity tends to a Gaussian as m increases regardless of the probability distribution of individual measurements of the quantity, provided it has a finite variance, see Yates and Goodman [17]. A last reason is that a Gaussian distribution has some advantages in solving the optimization problem mathematically. In the next chapter the Likelihood is multiplied by another distribution. If both distributions are Gaussians then the result of the multiplication is again Gaussian.

2.1.3 Bayesian inference

The goal of approximating the underlying process of a given training data set D is making a model that can predict the process output related to an unseen input $\mathbf{x}^{\text{new}}$. The probability of a process output is written as $p(t|\mathbf{x}^{\text{new}}, D)$. In the Maximum Likelihood approach the information in D was summarized in a single most probable weight vector after the training process. D can therefore be replaced by $\mathbf{w}$ in the condition, $p(t|\mathbf{x}^{\text{new}}, \mathbf{w})$. In the Bayesian inference method we do not restrict the weight vector to a single most probable vector. Instead of this we interpret $\mathbf{w}$ as a random variable like t. This results in a joint probability $p(t, \mathbf{w}|\mathbf{x}^{\text{new}}, D)$.
A fundamental rule in probability theory is marginalization. Marginalization is used to transform a joint probability into a marginal probability by integrating over the range of the variable we would like to remove from the distribution. A joint probability p(A,B) is transformed into p(A) by

$$p(A) = \int p(A,B)\,\mathrm{d}B \; . \tag{2.11}$$

Applying this marginalization principle to model training results in

$$p(t|\mathbf{x}^{new}, D) = \int p(t, \mathbf{w}|\mathbf{x}^{\text{new}}, D)\,\mathrm{d}\mathbf{w} \; . \tag{2.12}$$

The term in the integral can be rewritten in separate components by making use of another fundamental rule in probability theory. The product rule, which can be written as

$$p(A, B) = p(A|B)p(B) \,. \tag{2.13}$$

Applying the product rule to the term in the integral results in

$$p(t, \mathbf{w}|\mathbf{x}^{\text{new}}, D) \rightarrow p(t, |\mathbf{w}, \mathbf{x}^{\text{new}}, D)p(\mathbf{w}|\mathbf{x}^{\text{new}}, D) \,. \tag{2.14}$$

In the first of the two components D can be removed from the condition, because the information in the data is summarized into the vector $\mathbf{w}$. In the second component $\mathbf{x}^{\text{new}}$ can be removed from the condition. When the term in the integral of Equation 2.12 is substituted with this result the integral is given by

$$p(t|\mathbf{x}^{\text{new}}, D) = \int p(t|\mathbf{w}, \mathbf{x}^{\text{new}})p(\mathbf{w}|D)\,\mathrm{d}\mathbf{w} \,. \tag{2.15}$$

The term $p(t|\mathbf{w}, \mathbf{x}^{\text{new}})$ was already given in Equation 2.7. The new term here is $p(\mathbf{w}|D)$. This is the probability for $\mathbf{w}$ given the data. To explain how we can find this probability distribution we first need to explain another fundamental principle in probability theory called Bayes' Rule. Bayes rule is given by

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)} \,. \tag{2.16}$$

This rule can also be derived from the product rule as follows

$$p(B|A) = \frac{p(B, A)}{p(A)} = \frac{p(A|B)p(B)}{p(A)} \,. \tag{2.17}$$

Applying Bayes' Rule to our problem results in

$$\underbrace{p(\mathbf{w}|D)}_{\text{posterior}} = \frac{\overbrace{p(D|\mathbf{w})}^{\text{likelihood}}\overbrace{p(\mathbf{w})}^{\text{prior}}}{\underbrace{p(D)}_{\text{evidene}}} \,. \tag{2.18}$$

The distribution $p(\mathbf{w}|D)$ can be interpreted as 'the knowledge' we have about the parameters. If we start training a model and no data has been observed the 'state of knowledge' about the parameters is represented by a prior distribution $p(\mathbf{w})$. The prior distribution represents the knowledge that we know 'a priori', in advance. By observing training data this prior distribution is converted into a posterior distribution $p(\mathbf{w}|D)$. Figure 2.3 gives an example of how the prior distribution and the posterior distribution look like in an ideal situation. The prior distribution is most times a broad distribution, because there is little knowledge in advance.
The denominator in Equation 2.17 is the normalization factor and is given by

$$p(D) = \int p(D|\mathbf{w})p(\mathbf{w})d\mathbf{w} \,. \tag{2.19}$$

Regularization by using prior information

In Section 2.1.1 it was shown that we can prefer small parameters over larger parameters by adding a penalty term to the error function. Such a term is called a regularizer. When using Bayesian inference this principle can be given a natural interpretation in the form of the prior distribution. The more we prefer small parameters the smaller the variance of the prior distribution

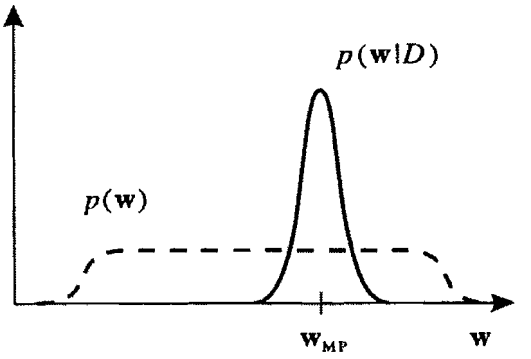


Figure 2.3: *In Bayesian inference a prior distribution is converted into a posterior distribution by learning from data.*

should be. If we take again the Gaussian form, as in Equation 2.7 we see that the term in the exponent is equal to the regularizer used in PLS.

$$p(\mathbf{w}|\alpha) = \left(\frac{\alpha}{2\pi}\right)^{W/2} \exp\left\{-\frac{\alpha}{2}\sum_{i=1}^{W}[0 - w_i]^2\right\} . \tag{2.20}$$

2.2 Neural Networks

There are different models that can be used to approximate the underlying process from training data. The models that will be used in this thesis are neural networks. Neural networks are called this way, because their structure and functionality has a lot of similarities with the human brain. For more information about the biological structure of neurons in a human brain, Chapter 1 of Rojas [12] can be read. In this thesis only the theoretical model will be discussed. In the next subsections neural networks will be explained in a very basic way. Neural networks are built from a number of small elements with the same structure, called neurons. This section starts with explaining a single neuron. After that a neural network with multiple neurons will be explained. In the last subsection the training process of neural networks will be explained

2.2.1 A single neuron

A neuron has one output and one or more inputs (see Figure 2.4). The transfer from the inputs to the output of the neuron can be divided in two steps. First, all inputs x are multiplied by their weight factor w and summed together. An equation for this is given by

$$a = w_0 + \sum_{i=1}^{d} w_i x_i . \tag{2.21}$$

Second, the output of this summation is fed into an activation function, which then gives the output value. Depending on the problem that has to be solved different activation functions can be chosen. Figure 2.5 shows 3 examples of different activation functions.
Let's start with a simple example problem to explain the functionality of neuron. If an appraiser makes an approximation of the price of a house he takes into account different aspects such as the size of the house, the location of the house and facilities in the neighborhood. If all aspects have been taken into account he comes to a price for the house.
This problem can be seen as a regression problem where the different aspect form a feature vector

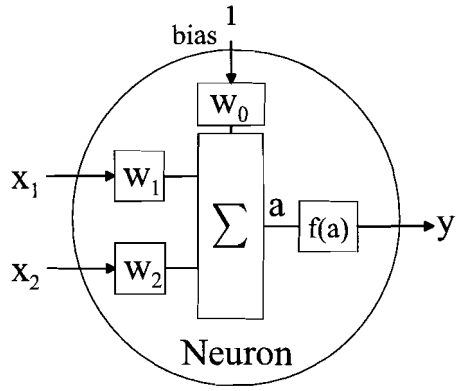
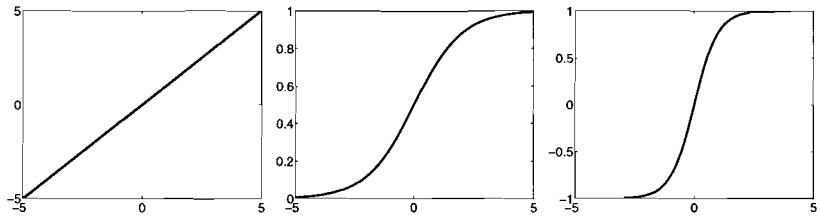


Figure 2.4: *A single neuron. All inputs are weighted and summed together. The result of this summation is fed into an activation function which determines the output value.*

x and the house price is the target vector **t**. All the different aspects have their own contribution to the price. These contributions can be controlled by the weight values of the different inputs. If there are several examples of houses these can be used as training data and we could try to model the decisions of the appraiser with a neuron.



(a) linear (b) sigmoid (c) tanh

Figure 2.5: *examples of activation functions.*

2.2.2 The multilayer perceptron

The output of a neuron can be connected to an input of another neuron. By putting some neurons together a neural network can be formed. An example of such a network is shown in Figure 2.6. The network in Figure 2.6 is called a Multi-Layer Perceptron. In this thesis we will use a multilayer perceptron which consists of two layers, an hidden layer and an output layer. The hidden layer is called this way, because the outputs of the neurons in this layer can not be observed directly on the network outputs.

2.2.3 Training a neural network

Optimal values for the parameters **w** are found by minimizing an error function. Figure 2.7 shows an example of an error function in weight space in the case the weight space is two-dimensional. A minimum error can be found by finding a position in the weight space where the derivative of the error function is zero. In the case of multidimensional weight space it is very difficult to solve this analytically. Optimal values for the parameters are therefore found by iterative optimization algorithms. The network training starts on an initial position in the weight space. This initial position is chosen at random. On each position in the weight space the algorithm makes a decision in which direction and how far in this direction it should go further to become as close as possible

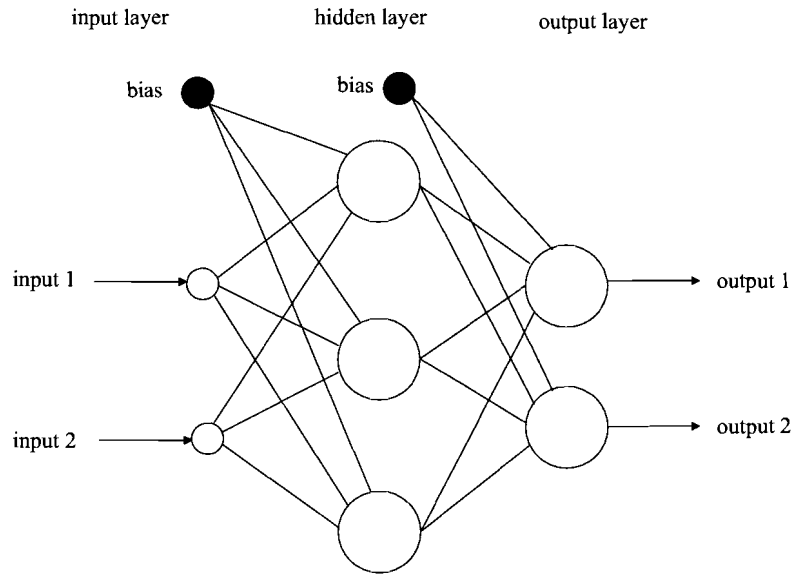


Figure 2.6: *When some neurons are connected to each other they form a neural network.*

to a minimum of the error function. Different algorithms can be used for this task. For example Gradient Descent, Conjugate Gradient or Scaled Conjugate Gradient. More details can be found in Bishop [2]. In this thesis the Scaled Conjugate Gradient (SCG) algorithm will be used.
Many of the optimization algorithms make use of the *Hessian* matrix. This matrix contains the second derivative information of the error on the output to all the weights in the network and is given by

$$\mathbf{H} = \nabla\nabla E = \begin{pmatrix} \frac{\partial^2 E}{\partial w_i^2} & \frac{\partial^2 E}{\partial w_1 w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 w_I} \\ \frac{\partial^2 E}{\partial w_i^2} & \frac{\partial^2 E}{\partial w_1 w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 w_I} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_i^2} & \frac{\partial^2 E}{\partial w_1 w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 w_I} \end{pmatrix} \tag{2.22}$$

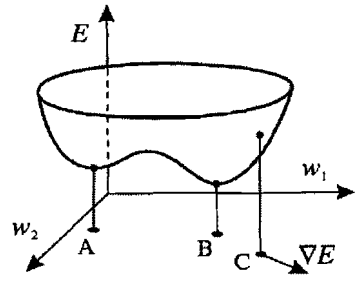


Figure 2.7: *Visualization of an error function above a two-dimensional weight space. An optimization algorithm can be used to find a minimum error.*

In this chapter we have considered how to train a model by examples of input to output relations. We started this chapter with a common used method called least squares error estimation. This method minimizes the "sum-of-squares" error function. It was mentioned that "overfitting" can

occur when the used model is too complex. The overfitting problem was resolved by adding a penalty term to the sum-of-squares error function.
After the explanation of the sum-of-squares function it was shown that it is possible to view the training process from a statistical viewpoint. Maximum Likelihood was introduced. It was shown that the assumption of Gaussian distributed target data in the Maximum Likelihood method can result in the "sum-of-squares" error function . The statistical viewpoint was extended by Bayesain Inference. Bayesian inference was introduced as a method for dealing with uncertainties in a model. A specific model type, called neural network, was discussed.

In the next chapter the Bayesian inference principle is extended to a three level hierarchical framework.

Chapter 3

Bayesian inference and Neural Networks

In this chapter it will be explained how the Bayesian inference principle, as described in the previous chapter, can be used in an hierarchical way in the process of neural network optimization. In the previous chapter was explained that the 'knowledge' about the parameters $\mathbf{w}$ of a model can be described by a posterior probability distribution over the parameter space. The hyperparameters and the model architecture were chosen before the training process was started. What we would like to show in this chapter is that not only the $\mathbf{w}$ parameters, but every uncertainty in our model can be given a Bayesian interpretation. Other uncertainties are for example the state of knowledge about the hyperparameters and the size of the neural network. The Bayesian framework that is used in this chapter is described by Mackay in [7, 6]. This chapter starts with an explanation of the different levels of inference in Mackay's Bayesian framework. After that an approximation method as used by Mackay, called the Evidence method, will be explained. We end this chapter by giving three applications of the Bayesian framework.

3.1 Three levels of Bayesian inference

A Bayesian framework, consisting of three levels of Bayesian inference, can be built for dealing with the different uncertainties in the process of neural network optimization. The first level of inference, the optimization of the weight parameters, was already discussed in the previous chapter. The second and the third level of inference are introduced here. This section is used to emphasize the hierarchy in Mackay's Bayesian framework.

Level 1: Weight parameters

In this subsection the equations for the first level of inference are rewritten for easier use in the remaining of this chapter. The likelihood in Equation 2.8 can be written in the form

$$P(D|\mathbf{w}) = \frac{1}{Z_D(\beta)} \exp\left(-\beta E_D\right) \tag{3.1}$$

where Z_D is a normalization factor that is given by

$$Z_D(\beta) = \left(\frac{2\pi}{\beta}\right)^{N/2} . \tag{3.2}$$

The prior in Equation 2.15 can be written in the form

$$P(\mathbf{w}) = \frac{1}{Z_W(\alpha)} \exp\left(-\alpha E_W\right) \tag{3.3}$$

where Z_W is a normalization factor that is given by

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{W/2} . \tag{3.4}$$

The equation for the posterior distribution can be written in the form

$$P(\mathbf{w}|D) = \frac{1}{Z_S(\alpha,\beta)} \exp\left(-S(\mathbf{w})\right) . \tag{3.5}$$

The term $S(\mathbf{w})$ in the exponent is the regularized error function and is written as

$$S(\mathbf{w}) = \beta E_D + \alpha E_W . \tag{3.6}$$

The normalization factor is calculated by an integration of the unnormalized probability function over the parameter space. This results in the following integral

$$Z_S(\alpha,\beta) = \int \exp\left(-S(\mathbf{w})\right) \mathrm{d}\mathbf{w} . \tag{3.7}$$

This integral can be written in this simple form because $\mathbf{w}$ is written as a vector here. Actually, this integral is a multi-dimensional integral with a dimensionality that is equal to the number of weights (elements) in the vector $\mathbf{w}$. Even a small neural network has most times a number of weights that is much larger than three or four. This often results in highly dimensional integrals that are analytically intractable. To solve this integral an approximation method is needed. In Subsection 3.2.1 it will be shown how to approximate this integral by using a Gaussian approximation for the posterior weight distribution. The normalization factor $Z_S(\alpha,\beta)$ will be used on higher levels.

Level 2: Hyperparameters

In Chapter 2 two types of hyperparameters were introduced: β to control the variance of the likelihood and α to control the variance of the prior distribution. The hyperparameters were treated as constant values during the training proces. A way of finding optimal values for the hyperparameters was to validate the model, after training, on a validation set. If we give the hyperparameters a Bayesian interpretation the hyperparameters can be optimized automatically during the training process. To make the condition on α and β explicit, Equation 2.17 is rewritten as

$$P(\mathbf{w}|D,\alpha,\beta) = \frac{P(D|\mathbf{w},\beta)P(\mathbf{w}|\alpha)}{P(D|\alpha,\beta)} . \tag{3.8}$$

The equation for the second level of Bayesian inference, the one for the inference of the hyperparameters, is given by

$$P(\alpha,\beta|D) = \frac{P(D|\alpha,\beta)P(\alpha,\beta)}{P(D)} . \tag{3.9}$$

The integral for calculating the output distribution will now look like

$$P(t|\mathbf{x},D) = \iiint P(t|\mathbf{w},\mathbf{x},\beta)P(\mathbf{w}|D,\alpha,\beta)P(\alpha,\beta|D)\,\mathrm{d}\mathbf{w}\,\mathrm{d}\alpha\,\mathrm{d}\beta . \tag{3.10}$$

Level 3: Model architecture

Because of its repeating structure a neural network is a very flexible model. The size of a two-layer perceptron neural network can easily by expanded or reduced by adding or removing neurons from

the hidden layer. Different amounts of neurons in the hidden layer result in networks with different complexities. A network that is too small results in a too simple network, which is not able to fit to all input to output relations in the data correctly. This is called *underfitting*. A network that is too large will fit too much to the training data. This results in poor generalization. This is called *overfitting*. If we define H_i to be a neural network with i hidden units in the hidden layer then the Bayesian equation for level 1 will be:

$$P(\mathbf{w}|D, \alpha, \beta, H_i) = \frac{P(D|\mathbf{w}, \beta, H_i)P(\mathbf{w}|\alpha, H_i)}{P(D|\alpha, \beta, H_i)} . \tag{3.11}$$

The equation for level 2 is given by

$$P(\alpha, \beta|D, H_i) = \frac{P(D|\alpha, \beta, H_i)P(\alpha, \beta|H_i)}{P(D|H_i)} . \tag{3.12}$$

and the equation for Bayesian inference on level 3 is

$$P(H_i|D) = \frac{P(D|H_i)P(H_i)}{P(D)} . \tag{3.13}$$

The integral for calculating the output distribution will now look like

$$P(t|\mathbf{x}, D) = \sum_{i=1}^{I} P(H_i|D) \iiint P(t|\mathbf{w}, \mathbf{x}, \beta, H)P(\mathbf{w}|D, \alpha, \beta, H_i)P(\alpha, \beta|D, H_i)\, \mathrm{d}\mathbf{w}\, \mathrm{d}\alpha\, \mathrm{d}\beta . \tag{3.14}$$

A summation is used instead of an integral, because we name a discrete model hypothesis space, indexed by i.

3.2 The Evidence Method

In the previous section it was explained how the Bayesian inference principle can be used on different levels of optimization to find a good model. Large multi-dimensional integrals were shown that represented full Bayesian models. In practice those integrals are analytically intractable and thus almost impossible to solve. To solve the Bayesian framework we need an approximation method. Two approximation methods described in the literature are The Evidence Method, see e.g. Mackay [8] and the Monte Carlo Method, see e.g. Neal [10]. In this thesis the Evidence method will be used. The Evidence Method is based on two approximations: the Laplace approximation for the posterior weight distribution and the assumption that the posterior distribution of the hyperparameters is sharply peaked around the position of its most probable values. We start this chapter with the explanation of the Laplace approximation. After this the Evidence Method will be discussed for hyperparameters and model selection.

3.2.1 Laplace approximation

To resolve the problem of the intractability we make the assumption that the posterior distribution has a Gaussian form. A multidimensional Gaussian distribution can be approximated by a Laplace approximation. Mackay describes this approximation in chapter 27 of his book [9].
The Laplace approximation is based on a second order Taylor expansion around the mode of the posterior weight distribution. The mode of the posterior distribution is located at the position of a minimum of the regularized error function. The position of the minimum of the regularized error is defined by $\mathbf{w_{MP}}$ (MP refers to the value on the most probable position in the parameter space

of $\mathbf{w}$) and can be found with a standard error minimization algorithms such as Scaled Conjugate Gradient (SCG) or Gradient Descent.

A second order Taylor expansion around the minimum of the regularized error results in

$$S(\mathbf{w}) = S(\mathbf{w_{MP}}) + \frac{1}{2}(\mathbf{w} - \mathbf{w_{MP}})^T \mathbf{A} (\mathbf{w} - \mathbf{w_{MP}}) \,. \tag{3.15}$$

The first order term has been left out in this formula, because on the position of a minimum the first derivative is zero. The matrix $\mathbf{A}$ is the Hessian matrix. In chapter 2 the Hessian matrix was introduced as a matrix containing second derivative information of the weights to the error on the model output. If the regularized error function S($\mathbf{w}$) is used the total Hessian matrix is given by

$$\mathbf{A} = \nabla\nabla S_{MP} = \beta \nabla\nabla E_D + \alpha \mathbf{I} \tag{3.16}$$

where S_{MP} represents the value of the regularized error function on the position of $\mathbf{w_{MP}}$.
The Taylor approximation for S($\mathbf{w}$) can be filled in by Equation 3.5. This results in

$$P(\mathbf{w}|D) = \frac{1}{Z_S(\alpha, \beta)} \exp \left[-S(\mathbf{w_{MP}}) - \frac{1}{2}(\mathbf{w} - \mathbf{w_{MP}})^T \mathbf{A} (\mathbf{w} - \mathbf{w_{MP}}) \right] \,. \tag{3.17}$$

If this formula is compared to a standard formula for a W-dimensional Gaussian distribution, which is written as

$$\mathcal{N}(\mu, \Sigma) = (2\pi)^{-W/2} |\Sigma|^{-1/2} \exp \left[-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right] , \tag{3.18}$$

then it is obvious that both equations have the same structure. Equation 3.17 can therefore easily be written in the form of a d-dimensional Gaussian distribution by making the following substitutions

$$\mu \longrightarrow \mathbf{w_{MP}}, \quad x \longrightarrow \mathbf{w}, \quad \Sigma^{-1} \longrightarrow \mathbf{A} \,. \tag{3.19}$$

The approximated normalization factor now is given by

$$Z_S^*(\alpha, \beta) = \exp^{-S(\mathbf{w_{MP}})} (2\pi)^{W/2} |\mathbf{A}|^{-1/2} \,. \tag{3.20}$$

3.2.2 Hyperparameter optimization

In Equation 3.8, 3.9 and 3.10 of the previous section it was shown that the optimization of the hyperparameters can be seen as a second level of Bayesian inference in the hierarchical Bayesian framework. Equation 3.10 can be split in two parts. One part is the integration over the hyperparameters. This results in the following integral

$$P(\mathbf{w}|D) = \iint P(\mathbf{w}|\alpha, \beta, D) P(\alpha, \beta|D) \, \mathrm{d}\alpha \, \mathrm{d}\beta \,. \tag{3.21}$$

The result $P(\mathbf{w}|D)$ can be used in the other part, the integration over the weight parameters. This integration is given by

$$P(t|\mathbf{x}, D) = \int P(t|\mathbf{w}, \mathbf{x}, \beta) P(\mathbf{w}|D) \, \mathrm{d}\mathbf{w} \,. \tag{3.22}$$

One approximation of the Evidence method is that the posterior probability of the hyperparameters $P(\alpha, \beta|D)$ is sharply peaked around its most probable values α_{MP} and β_{MP}. As a result of this approximation we approximate the posterior distribution of the hyperparameters on level 1 as a delta-function at the position of the most probable hyperparameters. Equation 3.21 then simplifies to

$$P(\mathbf{w}|D) \simeq P(\mathbf{w}|\alpha_{MP}, \beta_{MP}, D) \iint P(\alpha, \beta|D)\, \mathrm{d}\alpha\, \mathrm{d}\beta = P(\mathbf{w}|\alpha_{MP}, \beta_{MP}, D)\ . \tag{3.23}$$

With the above approximation level 1 and level 2 can be combined by making use of two steps. Find the most probable values for the hyperparameters on level 2 and fill in the most probable values for the hyperparameters on level 1.
To find the most probable values for the hyperparameters α_{MP} and β_{MP} we need to find the maximum of the posterior distribution $P(\alpha, \beta|D)$. If we assume that the prior distribution is uniform over a large range then we assume that all values have equal probabilities in advance, as discussed in Berger[1]. The choice of this uniform prior and the assumption that the posterior distribution is sharply peaked around its most probable values makes it possible to find the most probable values for the hyperparameters by calculating only the Maximum Likelihood.
The likelihood $P(D|\alpha, \beta)$ is called *the evidence for α and β* and is the normalization factor on level 1. The equation for the likelihood can thus be found by integrating the numerator of level 1 over the weight space. The integral becomes

$$P(D|\alpha, \beta) = \int p(D|\mathbf{w}, \beta) p(\mathbf{w}|\alpha)\, \mathrm{d}\mathbf{w}\ . \tag{3.24}$$

If we write the probability distributions in its exponential form and use the notations as introduced in the beginning of this chapter then the integral can be written in the form

$$P(D|\alpha, \beta) = \frac{1}{Z_D(\beta)} \frac{1}{Z_W(\alpha)} \int \exp\left(-S(\mathbf{w})\right) \mathrm{d}\mathbf{w}\ . \tag{3.25}$$

The integration of -S(**w**) over the weight space results in the normalization factor as approximated by the Laplace approximation of Subsection 3.2.1. The likelihood can now be calculated from the normalization factors.

$$P(D|\alpha, \beta) = \frac{Z_S(\alpha, \beta)}{Z_D(\beta) Z_W(\alpha)} = \frac{e^{-S(w_{MP})} (2\pi)^{W/2} |\mathbf{A}|^{-1/2}}{\left(\frac{2\pi}{\beta}\right)^{N/2} \left(\frac{2\pi}{\alpha}\right)^{W/2}}\ . \tag{3.26}$$

We would like to find the values for the hyperparameters that maximize this likelihood. This calculation is easier for the the log of the likelihood. This log-likelihood is given by

$$\ln P(D|\alpha, \beta) = -\alpha E_W^{MP} - \beta E_D^{MP} - \frac{N}{2} \ln (2\pi) - \frac{1}{2} \ln |\mathbf{A}| + \frac{N}{2} \ln \beta + \frac{W}{2} \ln \alpha\ . \tag{3.27}$$

With the equation of the log likelihood we can find most probable values for α and β by setting the derivative to zero. If we assume that α and β are independent of each other we can take partial differentials. The most difficult part of the equation for differentiation seems to be the determinant of Hessian **A**. We continue this section by taking the partial differentials of ln $|\mathbf{A}|$ to α and β.

If we assume that E_D is a pure quadratic function we can bring $\beta \nabla\nabla E_D$ on a basis of eigenvectors. The matrix **A** then becomes a diagonal matrix with $\lambda_i + \alpha$ on the diagonals.

$$\frac{\partial}{\partial \alpha} \ln |\mathbf{A}| = \frac{\partial}{\partial \alpha} \left(\prod_i (\lambda_i + \alpha) \right) = \frac{\partial}{\partial \alpha} \sum_i \ln (\lambda_i + \alpha) = \sum_i \frac{1}{\lambda_i + \alpha} = \mathrm{Tr}\left[\mathbf{A}^{-1}\right]\ . \tag{3.28}$$

When we fill in this differential in the complete equation and set this equation to zero we get

$$\frac{\partial}{\partial \alpha} \ln p(D|\alpha, \beta) = -E_W^{MP} - \frac{1}{2} \sum_{i=1}^{W} \frac{1}{\lambda_i + \alpha} + \frac{W}{2\alpha} = 0\ . \tag{3.29}$$

This can be written in the form

$$2\alpha E_W^{MP} = W - \sum_{i=1}^{W} \frac{\alpha}{\lambda_i + \alpha} = \gamma \,. \tag{3.30}$$

The variable γ measures the *effective number of weights* whose values are controlled by the data rather than by the prior.

$$\gamma \equiv \sum_{i=1}^{W} \frac{\lambda_i}{\lambda_i + \alpha} \,. \tag{3.31}$$

Equation 3.30 can be solved to find a new value for α,

$$\alpha^{new} = \frac{\sum_{i=1}^{W} \frac{\lambda_i}{\lambda_i + \alpha}}{2E_W^{MP}} = \frac{\gamma}{2E_W^{MP}} \,. \tag{3.32}$$

The α on the left hand side of Equation 3.30 is the new value for α and the α on the right size is the α that was used during the training process on the first level of the inference.
The differentiation to β is implemented in a same way. The update formula for β is given by

$$\beta^{new} = \frac{N - \gamma}{2E_D^{MP}} \,. \tag{3.33}$$

Details can be found in Chapter 10 of Bishop [2].

3.2.3 Model selection

To select the most probable model the maximum of $P(H_i|D)$, the posterior on level 3, needs to be found. In this thesis we assume that each model has an equal probability ('degree of belief') when no data has been observed. This assumption is made explicit by making again use of a uniform prior distribution. As in the previous section the problem of finding a maximum for $P(H_i|D)$ can be limited to finding the maximum likelihood if such a prior is used. The term $P(D|H_i)$ is now called the *evidence for* H_i. This term should not be confused with the term Evidence Method.
The likelihood is found by integrating the nominator of level 2 over the hyperparameter space

$$P(D|H_i) = \iint P(D|\alpha, \beta, H_i) P(\alpha, \beta|H_i) \, \mathrm{d}\alpha \, \mathrm{d}\beta \,. \tag{3.34}$$

The type of hyperparameters used in this thesis are also called *scale parameters*. The hyperparameter α scales the weights and the hyperparameter β scales the noise on the network output. It is common to use a logarithmic parameter space for scale parameters. Scale parameters can only have positive values. If we assume that the hyperparameters α and β are independent of each other we can calculate their likelihoods separately

$$P(D|\ \ln \beta) = P(D|\ \ln \beta_{MP}) \exp\left(-\frac{(\ \ln \beta - \ \ln \beta_{MP})^2}{2\sigma^2_{\ln \beta}} \right) \,. \tag{3.35}$$

A formula with the same form can be constructed for α. From these likelihoods for the hyperparameters the variances can be calculated. More details can be found in Chapter 10 of Bishop [2]. The formulas for the variances are given by

$$\frac{1}{\sigma^2_{\ln \beta}} = \frac{1}{2}(N - \gamma) \tag{3.36}$$

and

$$\frac{1}{\sigma^2_{\ln \alpha}} = \frac{\gamma}{2} . \tag{3.37}$$

The integration if the likelihood times a uniform prior over the hyperparameter space $\ln \beta$ results in

$$P(D|\beta_{MP}) \int P(D|\ln \beta_{MP}) P(\beta) \, \mathrm{d} \ln \beta = p(D|\beta_{MP}) \frac{(2\pi)^{1/2} \sigma_{\ln \beta}}{\ln \Omega} . \tag{3.38}$$

The prior $\frac{1}{\ln \Omega}$ is uniform over a large space. A value for Ω used in many other literature is about 10^{-3}. If the formulas for both hyperparameters are combined the the evidence for H_i is given by

$$P(D|H_i) \simeq P(D|\alpha_{MP}, \beta_{MP}, H_i) 2\pi \frac{\sigma_{\ln \alpha}}{\ln \Omega} \frac{\sigma_{\ln \beta}}{\ln \Omega} . \tag{3.39}$$

The log-likelihood can now be written in the form

$$\begin{aligned} \ln P(D|H_i) = &-\alpha_{MP} E_W^{MP} - \beta_{MP} E_D^{MP} - \frac{1}{2} \ln |\mathbf{A}| + \frac{W}{2} \ln \alpha_{MP} \\ &+ \frac{N}{2} \ln \beta_{MP} + \ln M! + 2 \ln M + \frac{1}{2} \left(\frac{2}{\gamma} \right) + \frac{1}{2} \ln \left(\frac{2}{N - \gamma} \right) . \end{aligned} \tag{3.40}$$

The factor M in the above formula represents the number of hidden units in a neural network and is a correction factor for symmetries in the network.

3.3 Applications of Bayesian inference

In this section four applications are presented that make use of Bayesian inference. The first three where mentioned in the problem definition at the beginning of this thesis as applications of Mackay's Bayesian Framework. The last one uses also Bayesian inference, but does not make use of the same approximations.

3.3.1 Error bars

In Equation 3.22 the integral was given to estimate the distribution of targets t, given an input vector **x** and data D. In Section 2.1.2 a relation between t and y was given, assumed that Gaussian noise with zero mean was added to y. In our case y is a model output and represents the mean of the distribution in Equation 3.22. The variance of this probability distribution on the output can give us information about how sure the model is about its prediction. The larger the variance the less sure the model is about its prediction. In the Bayesian framework that we use the distribution on the output is approximated by a Gaussian probability distribution. We define error bars as the distances of one σ from the model output. The model output represents the mean value. In Figure 3.1 an example is shown. A Gaussian distribution is a symmetric distribution and thus the error bars are symmetric.
Let's start with an example. A model is trained on a small data set of points derived from a noisy sine function (circles in the figure). This can be seen as an incomplete training data set. The dashed lines in the figure represent the error bars. It is obvious that the size of the error bars differs for the different locations in input space x. On the locations where the model has seen more data during training its prediction will be more accurate and it will be more sure about its prediction. On the locations where the model has seen less data during training it is less sure and this will result in a broader distribution on the output.
The equation for calculation of the distribution on the output was given in Equation 2.17. It is obvious that there are two probability distributions that contribute to the output distribution. The first is the likelihood for the input value x and the second is the posterior weight distribution.

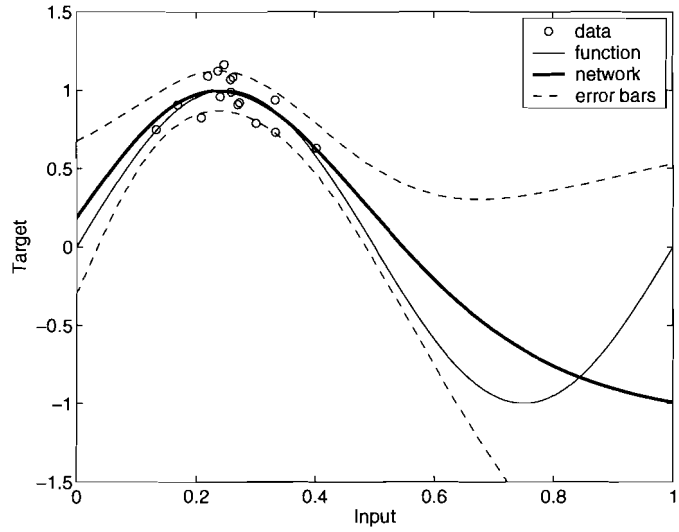
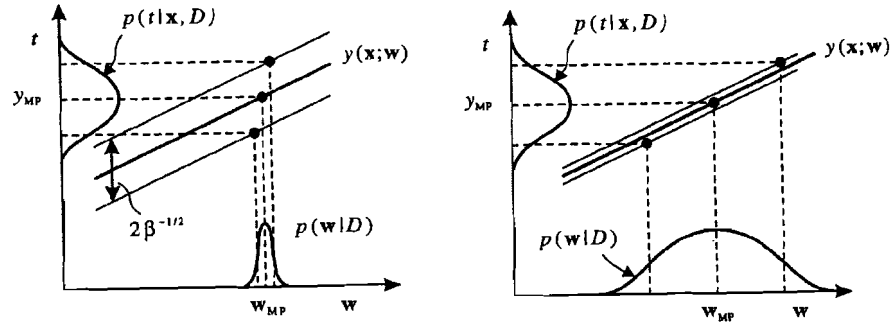


Figure 3.1: *Error bars around an approximated sine function.*

Figure 3.2 gives an indication of the contribution of the two probability distribution to the output distribution. In Figure 3.2a the size of the error bars is dominated by the intrinsic noise. In Figure 3.2b the size of the error bars is dominated by the variance of the posterior distribution of the weights.



(a) The size of the error bars is dominated by the intrinsic noise

(b) The size of the error bars is dominated by the variance of the posterior distribution

Figure 3.2:

Approximation of the error bars

To solve the integral in Equation 3.22 we can make use of earlier shown equations. If for $P(t|x,w)$ Equation 2.7 is used and if for the posterior distribution $P(w|D)$ the Gaussian approximation is used then this results in

$$P(t|\mathbf{x},D) \propto \int \exp\left(-\frac{\beta}{2}\{t-y(\mathbf{x};\mathbf{w})^2\}\right) * \exp\left(-\frac{1}{2}\Delta w^T A \Delta w\right) \mathrm{d}\mathbf{w} \tag{3.41}$$

If the posterior distribution is sufficiently narrow then $y(\mathbf{x};\mathbf{w})$ can be approximated by its linear expansion around $y(\mathbf{x};\mathbf{w_{MP}})$, which can be written as

$$y(\mathbf{x};\mathbf{w}) = y(\mathbf{x};\mathbf{w}_{\mathbf{MP}}) + g^T\Delta\mathbf{w}. \tag{3.42}$$

The first derivative to $\mathbf{w}$ at the point of the most probable weight vector $\mathbf{w}_{\mathbf{MP}}$ is written as:

$$\mathbf{g} \equiv \nabla_{\mathbf{w}} y|_{\mathbf{w}_{MP}}. \tag{3.43}$$

Whe above 3 equations together result in the following equation for error bars

$$\sigma_t^2 = \frac{1}{\beta} + g^T\mathbf{A}^{-1}g, \tag{3.44}$$

For more details refer to Bishop [2], Section 10.2

3.3.2 Automatic Relevance Determination

In cases of large feature vectors (a lot of inputs) it could be that a number of features are not much related to the outputs. Unless they are not very important they can ruin the quality of the prediction. The Bayesian framework has the possibility to control the influence of each input to the output automatically. This is done by associating different prior distributions to each input. Each weight connected to a certain input gets the same prior as the other weights connected to the same input. The variance of that prior controls the regularization of that weight group. In this way we can control the sizes of the weights in that group. Large hyperparameters leads to strong regularization and thus small weights and thus less influence on the output.

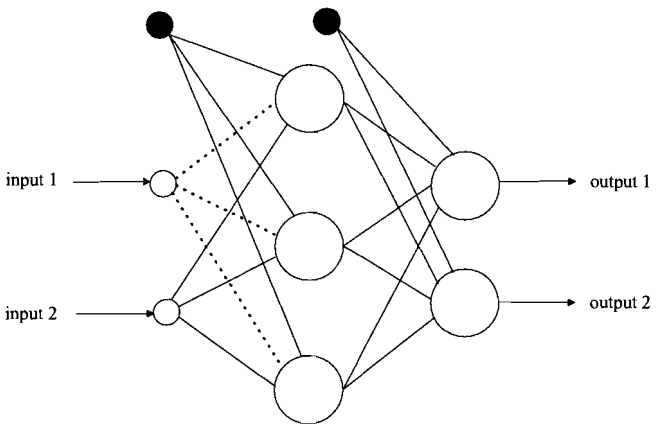


Figure 3.3: *Different prior distributions are associated with each input. The weights connected to the dashed lines share their prior distribution.*

3.3.3 Model selection

Models that are too complex give poor generalization. They perform bad on data other than the training data. This problem is caused by the fact that the model tries to fit to noise on the training data too. On the other hand when a model is too simple the results are also poor, because the model is not able to cover all characteristics of the underlying process. We need a model that is not too simple and not too complex.

Model selection is automatically embodied in the Bayesian framework.

The model evidence $p(D|H_i)$ is build from two components

- the best fit likelihood, which measures how well the tuned model fits the data.

- the ratio of the available parameter volume after and before they are tuned to the data.

The last component is called the *Occam factor* and can be seen as a penalty term. The parameter volume before the training process is given by the volume of the prior distributions. The parameter volume after the training process is given by the volume of the posterior distributions. Very large models have a lot of parameters and thus a large prior volume. A larger prior volume results in a smaller Occam factor and thus smaller model evidence. A smaller model will have a smaller prior volume and will get higher model evidence. Very small models will also get a lower evidence because the parameters need to be finely tuned to the data for a good fit. This results in less posterior space and thus smaller model evidence.

3.3.4 Pruning by using a Laplace prior

An alternative for model selection is *pruning*. When pruning is used we do not compare the performance of networks with different sizes to find a good size for the network. Instead of this we start with a big network and reduce the size of this network by removing connections that have hardly any influence on the errors that the network outputs make with the target values in training data. Williams [16] presents a method that uses Bayesian inference and at the same time contributes to a simple method for pruning. He makes use of a prior distribution with a Laplacian form. Such a prior can be written as

$$\frac{1}{(2/\alpha)^W} \exp\left(-\alpha \sum_{i=1}^{W} |w_i|\right) . \tag{3.45}$$

The hyperparameter α in this formula represents the inverse of the mean of the distribution. The figure below shows a Laplace distribution with α set to 10.

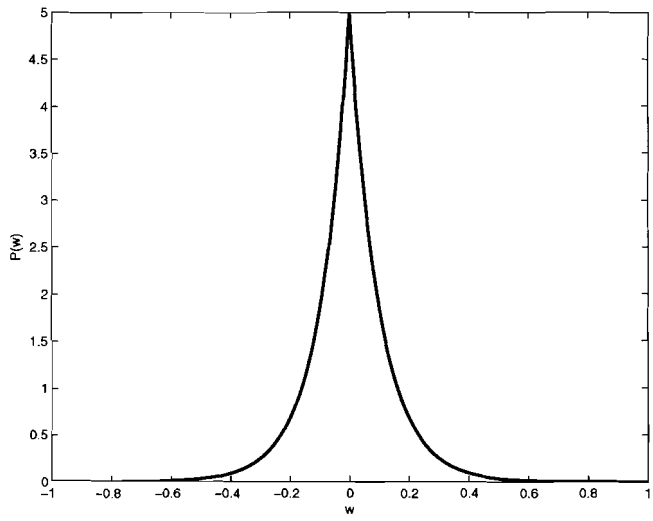


Figure 3.4: *Laplace prior distribution with α set to 10.*

The distribution in Figure 3.4 has a lot of area around the zero point. Because of this the weights are automatically driven to zero as much as possible during the training process. This is known in the literature as *sparse Bayesian learning*. The fact that the weights are driven to zero can be used for pruning. Weight optimization in neural networks is an iterative process. In each iteration step the algorithm starts at a certain point in the weight space and travels in a certain direction to come closer to the position where the error has a minimum. If the algorithm goes to a point where a weight gets the size that is almost zero it can set the weight to an exact zero and test if

it is allowed to keep this weight at zero. We can have two situations for a zero weight. The first situation is that a weight is zero, but that it is on its way to become more positive or negative to result in a smaller total error. The second situation is that the weight should stay at zero, because traveling in both positive or negative direction results in a bigger total error. In the case of the second situation the weight can be removed from the network. With this method the size of the network is reduced during the training process.
The formulas for Bayesian inference with a Laplace prior have a lot in common with the above given formulas for Bayesian inference with a Gaussian prior. The penalty term that is used now is given by

$$E_W = \sum_{i=1}^{W} |w_i| \ . \tag{3.46}$$

This makes the calculation of the Hessian of the regularized error function easier, because the second derivative of this function to w is zero. The Hessain of the total error function can therefore be written as

$$\mathbf{A} = \beta \mathbf{H} = \beta \nabla \nabla E_D \ . \tag{3.47}$$

Another difference with Bayesian inference and a Gaussian prior is the normalization factor of the prior. The factor Z_W is now given by

$$Z_W = \left(\frac{2}{\alpha}\right)^W \ . \tag{3.48}$$

If we look back to Equation 3.26 and use the above two equations then the log evidence becomes

$$-\ln P(D|\alpha,\beta) = \alpha E_W + \beta E_D + \frac{k}{2}\ln(\beta) - W \ln \alpha - \frac{N}{2}\ln \beta + \text{constant} \tag{3.49}$$

with k as the full dimension of $\mathbf{w}$. The update rules for the hyperparameters are now given by

$$\frac{1}{\beta} = \frac{1}{N-k}\sum_{p=1}^{N}(y_p - t_p)^2 \tag{3.50}$$

for β and

$$\frac{1}{\alpha} = \frac{1}{W}\sum_{j=1}^{W}|w_j| \ . \tag{3.51}$$

for α.

$$\tag{3.52}$$

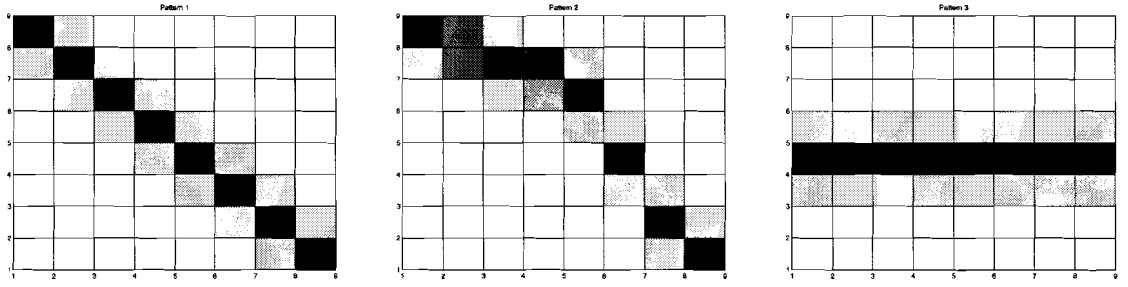
Chapter 4

Tests on a synthetic problem

In this chapter a synthetic problem will be introduced, which is used to test the theory as described in the previous chapters. The synthetic problem has a lot of similarities with the sound classifier that will be described in the next chapter and is therefore used as an intermediate step to get practical experience before running big time consuming simulations.
The chapter starts with a short description of the synthetic problem and a description of how test and training data sets are generated. After this problem description the different facets of the Bayesian framework will be tested on the problem. Those facets are: optimization of the hyperparameters, error bars, Automatic Relevance Determination (ARD) and model selection by model evidence.

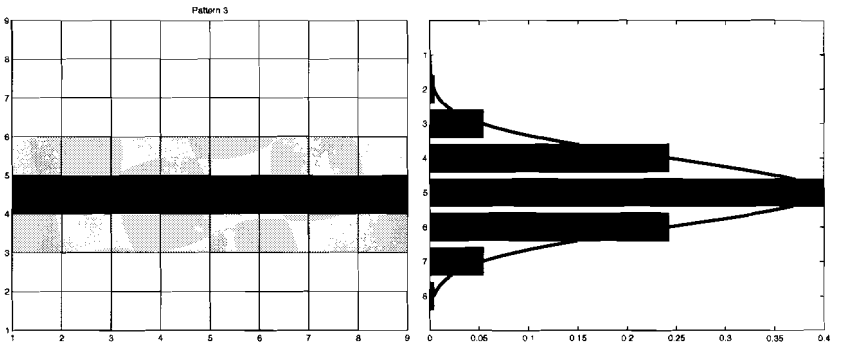
4.1 Description of the problem

The synthetic problem is a mapping from 64 inputs (8 by 8 grid) to 3 outputs. Figure 4.1 shows 3 possible input patterns. The 3 outputs should give respectively the fraction of pattern 1, pattern 2 and pattern 3 in a given test input grid. The patterns in Figure 4.1 can be seen as hill landscapes, seen from above and built from sampled Gaussian probability distributions. Figure 4.2 shows how pattern 3 is built. The right figure represents the Gaussian probability distribution in one column. The darkness of the color indicates the height where the darkest color represents the highest value.



(a) pattern1 (b) pattern 2 (c) pattern 3

Figure 4.1: *Examples of input patterns which are used in the synthetic problem. A trained neural network should estimate the presence of each pattern in a given input pattern*



(a) pattern 3

(b) sampled Gaussian distribution representing the heights in one column of a pattern

Figure 4.2: *Each column in figure a is built from sampled Gaussian distribution. Figure b shows how the distribution looks like.*

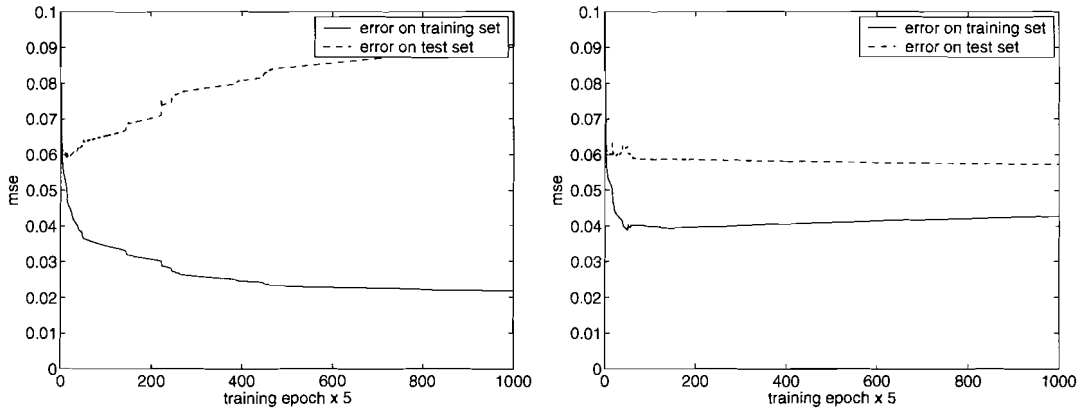
4.2 Generate training and test data

To train and test a neural network shifted versions of the patterns in Figure 4.1 are generated. For each pattern initial values for the means of the distributions are defined. Pattern 2 has for example the following means 8, 7.5, 7, 6.75, 6, 4, 2, and 1 in the columns, seen from left to right. Shifted patterns are generated by adding random noise to the initial mean values. This random noise is represented by a a Gaussian distribution with mean zero and variance 1.
To be consistent with the training procedures of the sound classifier in the next chapter, the training and test data sets are built from mixtures of the three patterns shown in Figure 4.1. The procedure for generating mixtures starts with defining a gain factor for each pattern. One of the patterns is given a gain of 1, another is given a gain between 0 and 1 (drawn from a uniform distribution) and the remaining pattern is given a gain of 0. The choice of which gain is given to which pattern is made at random.

4.3 Updating the hyperparameters

Regularization of the weight parameters and estimation of the correct noise variance on the output of the network can help a lot in improving generalization. Two different simulations are performed to show this. One of the simulations is performed with and the other without regularization and estimation of the noise variance. Figure 4.3 shows the mean squared error measured on the neural network outputs during the training process. Two error values are plotted: the error that the network makes on the training data and the error the network makes on the test data. The left figure, Figure 4.3a, shows an example of a training process with no hyperparameter updates. The right figure, Figure 4.3b, shows an example of a training process where the hyperparameters are updated during training. It is obvious that the Bayesian framework gives a good approach to optimize the hyperparamters. Optimization of the hyperparameters leads to regularization and improves the generalization performance. When Figures 4.3a and 4.3b are compared with each other it is obvious that the network in 4.3b has a much better generalization performance. The network in 4.3a performs better on the training data, but will do worse on the test data.
In Figure 4.4 the results of two longer simulations are plotted. Figure 4.6a shows the results of a simulation where 1000 samples are used and Figure 4.6b shows the results of a simulation where 5000 samples are used during the training process. It is obvious that updating the hyperparameters by Bayesian inference has more effect on the smaller dataset. The Figures 4.5 and 4.6 show how the weight values change during the training process. Also here the regularization effect is visible.

In Figure 4.5a the the range were most of the weights lay is smaller at the end of the simulation than in the beginning of the simulation.



(a) without update of hyper parameters

(b) with update of hyper parameters

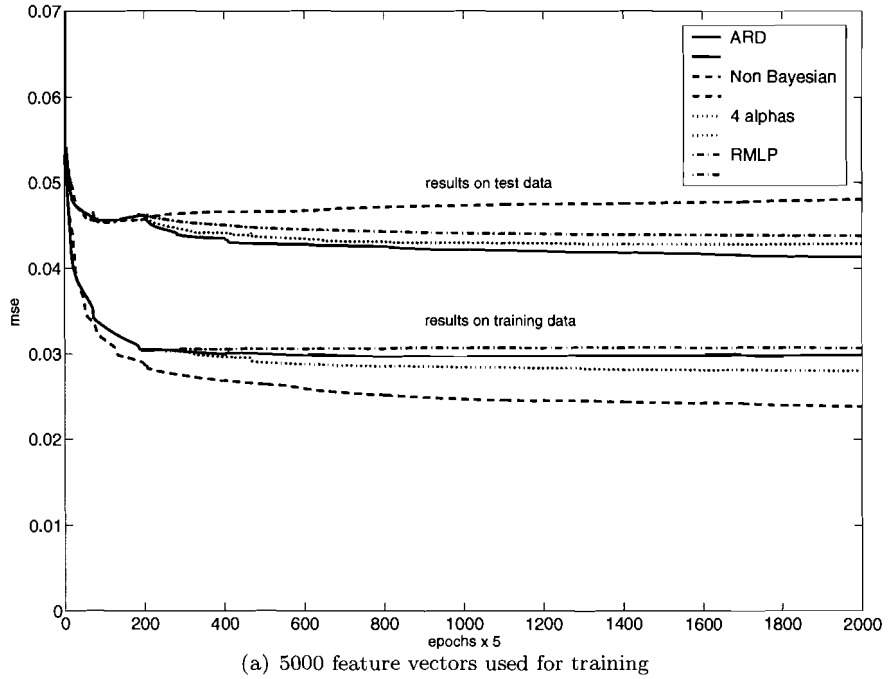
Figure 4.3: *Comparison of two situations during network training. Figure a shows the mse on the training data and the mse on the test data when no hyper parameter update is done during training. Figure b shows a situation where the hyper parameters are updated during training. The generalization effect is very obvious in this picture*

4.4 Automatic Relevance determination

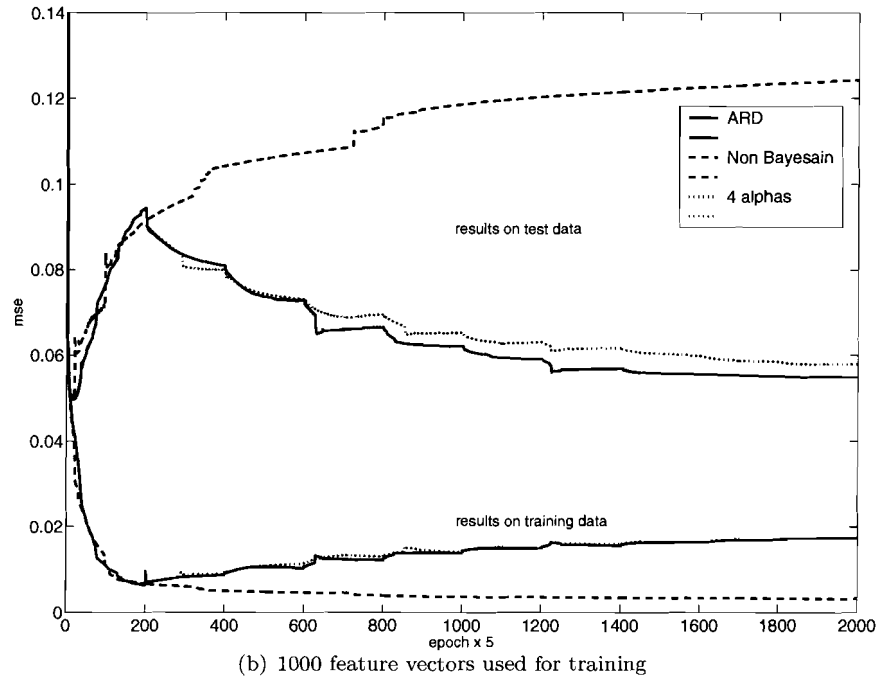
The relevance of each of the 64 inputs can be determined automatically by Automatic Relevance Determination. Results of using ARD can be found in Figure 4.4. Figure 4.4a shows the results of a network trained with 5000 training samples and figure 4.4b shows the results of a network trained with 1000 training samples. In both cases ARD leads to further generalization improvements compared to regularization with 4 hyperparamters.

4.5 Error bars on the outputs of the synthetic problem

In Subsection 2.1.3 it was mentioned that the estimated variances on the network outputs can be used as error bars. Figure 4.7 and 4.8 give an example of how those error bars can look like. A 2 layer perceptron neural network with 8 hidden units is trained with 1000 samples of mixtures of pattern 1 and pattern 3. The first mixture contains 100% of pattern 1 and 0% of pattern 3. The last mixture contains 0% of pattern 1 and 100% of pattern 3 see Figure 4.7a. The transfer functions in the hidden units have a sigmoidal form and the transfer functions in the output layer are linear now. When the trained network is tested on a test set with the same composition as the training set there is not much difference in the size of the error bars for the different test input samples. However, when a network is trained on a training set with the composition as given in Figure 4.7b the error bars will be different as can be seen in Figure 4.8b. The network is less sure about samples with a bigger amount of pattern 3 than pattern 1. This is because it has not seen those cases in the training data.

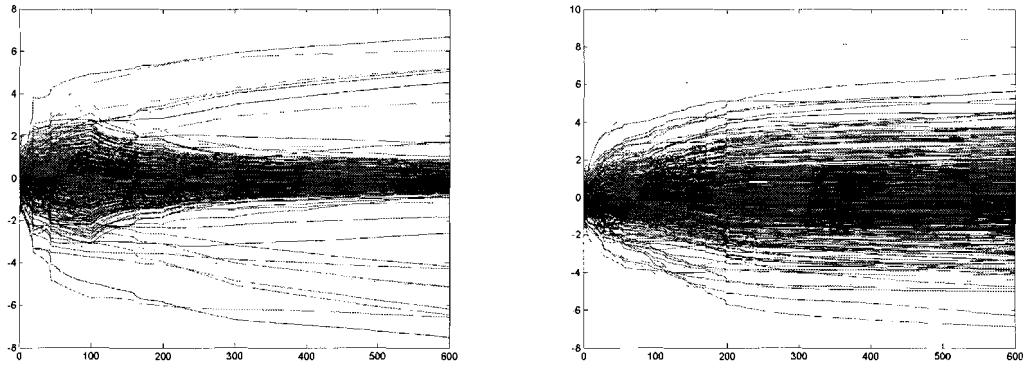


(a) 5000 feature vectors used for training



(b) 1000 feature vectors used for training

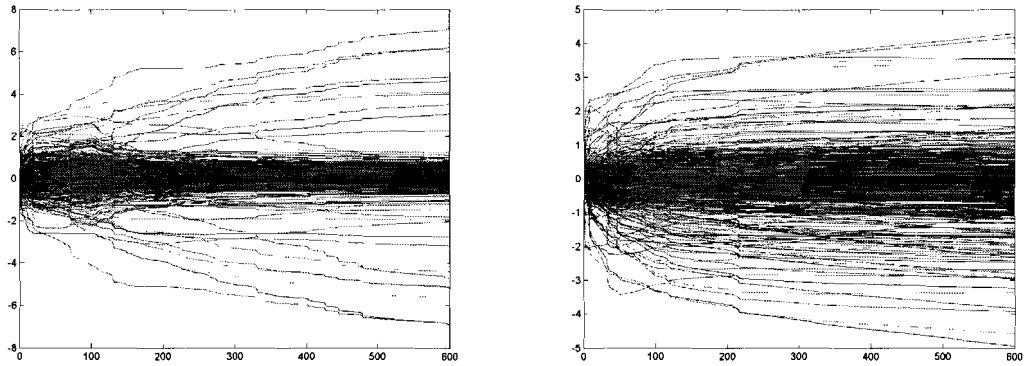
Figure 4.4: *Neural networks trained with different amounts of hyper parameters. In the case of non Bayesian there are no hyper parameters. 4 alphas uses separate alphas for the first layer biases, the first layer weights, the second layer biases and the second layer weights. ARD uses a different prior for each input.*



(a) with optimization of hyperparameters

(b) without optimization of hyperparameters

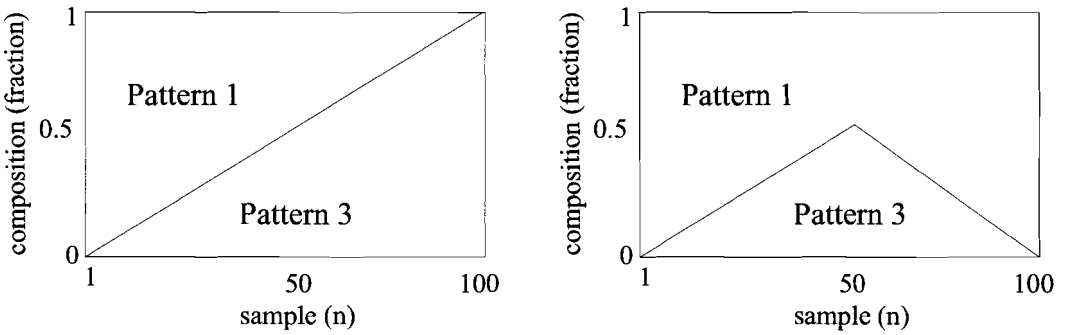
Figure 4.5: *Values of the network weights during the training process. There are 1000 samples used for training. The y-axis represents the weight value and the x-axis represents the number of the training epoch*



(a) with optimization of hyperparameters

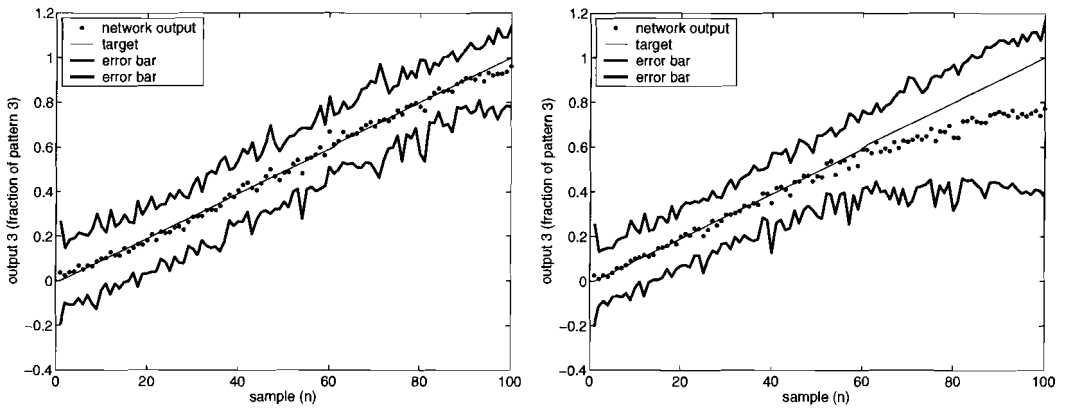
(b) without optimization of hyperparameters

Figure 4.6: *Values of the network weights during the training process. There are 5000 samples used for training. The y-axis represents the weight value and the x-axis represents the number of the training epoch*



(a) complete (b) incomplete

Figure 4.7: *Composition of training and test data sets for testing error bars. Figure a represents the composition of a complete training/test data set. Figure b represents the composition of an incomplete training/test data set. There are no training samples with a bigger amount of pattern 3 than pattern 1.*



(a) output 3 with error bars when trained on full range (b) output 3 with error bars when trained on half range

Figure 4.8: *2 cases of network output 1. In figure a the composition of training data as shown in figure 4.6a is used. In figure b the composition of training data as shown in figure 4.6b is used*

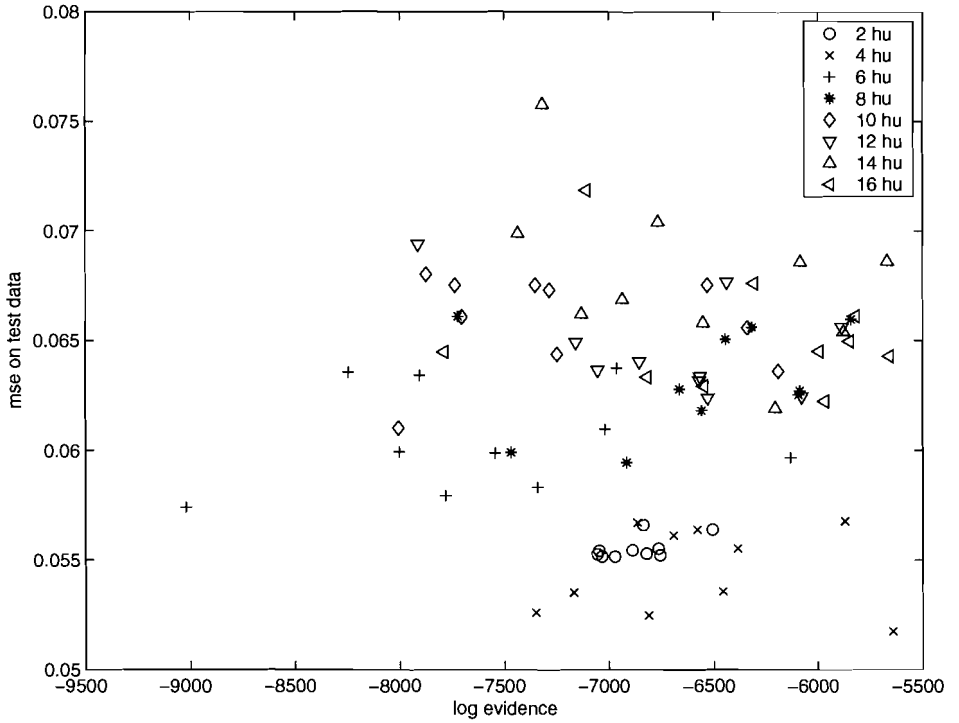
4.6 Using the model Evidence to select a good model

In Equation 3.40 an estimation was given for the model evidence. In this subsection we will investigate if this model evidence can be used to select a good model for our synthetic problem. The term model evidence for model selection should not be confused with the term evidence in Evidence Method. The model evidence $P(D|H_i)$ can also be called marginal likelihood for model H_i, but to be consistent with Penny [11] and Mackay [8] we call it the model evidence for model H_i.
To test how well the model evidence performs as model selection criterion we trained neural networks with different amounts of hidden units. For each network the model evidence was estimated and the mean squared error on an unseen test set was measured. In Figure 4.9 results of simulations with 1000 samples are shown. In Figure 4.10 results of simulations with 5000 samples are shown. It is expected that a network with a higher evidence will perform better on the unseen test set than a network with a lower model evidence. We therefore expect an (anti) proportional relationship between the model evidence and the mean squared error on the test set. This (anti) proportional relation is obvious in the case of 5000 samples, Figure 4.10a. In the case of 1000 samples, Figure 4.9a, this relation is not obvious.
In [11], Pennny says: "Model selection using the evidence or training error is only tenable if the number of training examples exceeds the number of network weights by a factor five or ten.". If we take into account Penny's factor five this means that for networks with 16 hidden units about 5500 samples are needed and that in the case of 1000 samples the results are expected to be reliable for networks with equal or less than 3 hidden units.
To investigate why the model evidence is not reliable in the case of 1000 samples we divided Equation 3.40 in different components to see which component is ruining the model evidence

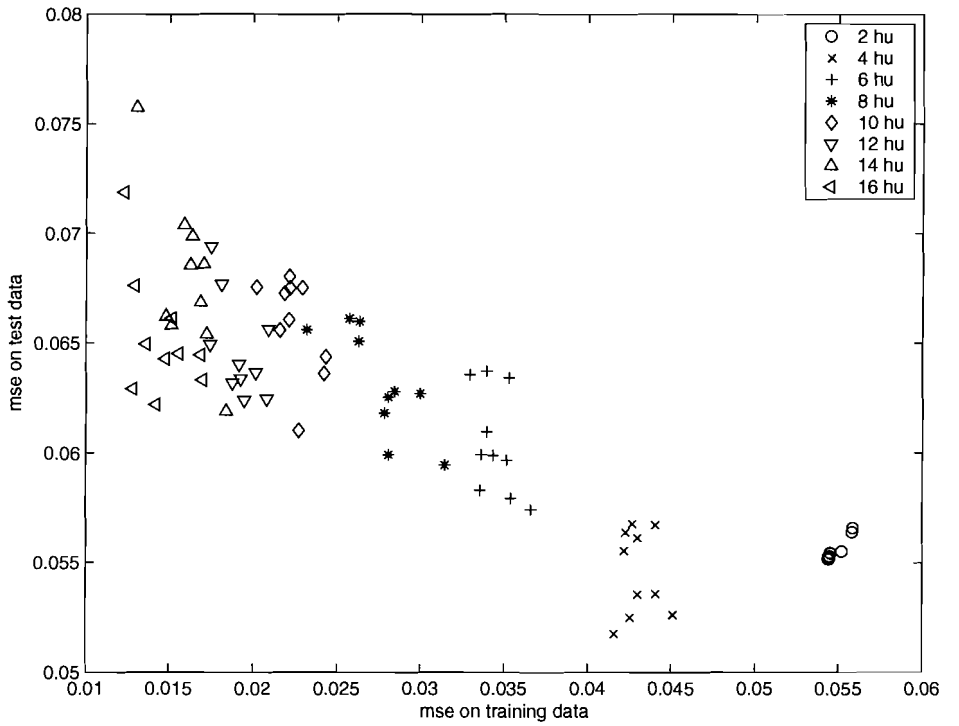
$$\ln p(D|H_i) = \underbrace{-\alpha_{MP}E_W^{MP} - \beta_{MP}E_D^{MP}}_{\text{comp 1}} \underbrace{-\frac{1}{2}\ln|\mathbf{A}|}_{\text{comp 2}} \underbrace{+\frac{W}{2}\ln\alpha_{MP}}_{\text{comp 3}} \underbrace{+\frac{N}{2}\ln\beta_{MP}}_{\text{comp 4}} \underbrace{+\ln M!}_{\text{comp 5}} \underbrace{+2\ln M}_{\text{comp 6}} \underbrace{+\frac{1}{2}\left(\frac{2}{\gamma}\right)}_{\text{comp 7}} \underbrace{+\frac{1}{2}\ln\left(\frac{2}{N-\gamma}\right)}_{\text{comp 8}} \quad (4.1)$$

In the case of 1000 samples component two has a larger contribution to the model evidence. Component two is proportional to the volume under the posterior distribution if this posterior distribution has a Gaussian form. It is confirmed by Mackay that this approximation with the Hessian matrix is very sensitive.

Mackay: "In my experience, evaluating the trace of the Hessian and using it to control hyperparameters is a good and successful idea in real applications. Evaluating the determinant and using it to evaluate the evidence for alternative models has not been so successful, though Thodberg's work is an exception to this observation. So I think that using the derivative of the evidence as a guide in a hyperparameter space is to be highly recommended, but the absolute value of the evidence, although I like it a lot in principle, in computational practice it doesn't seem to be so reliable a guide."

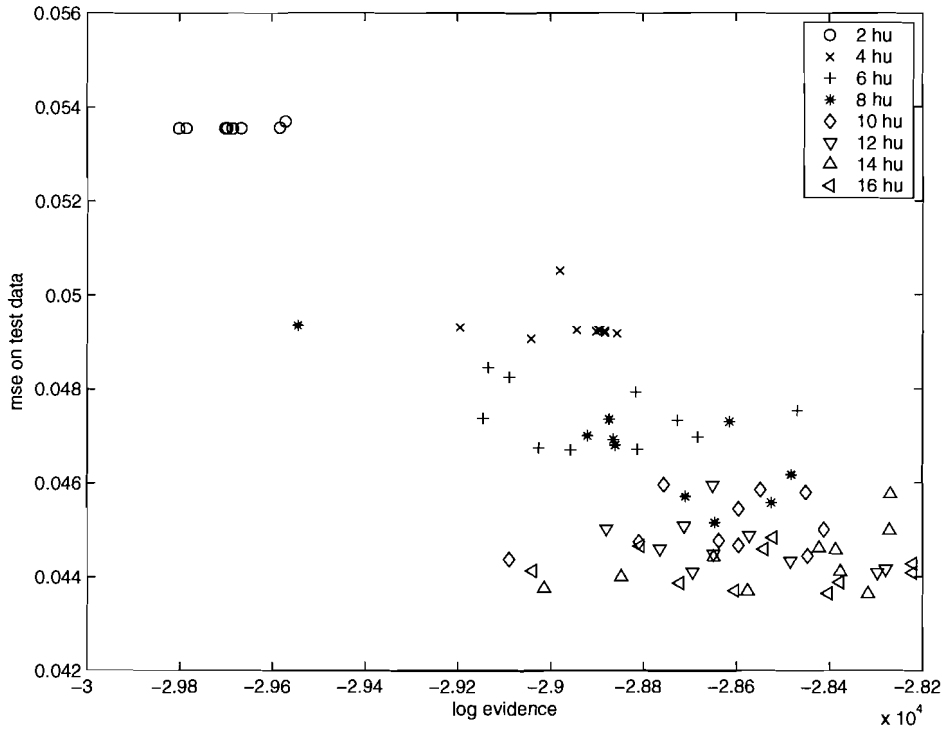


(a) model evidence vs mse on test data

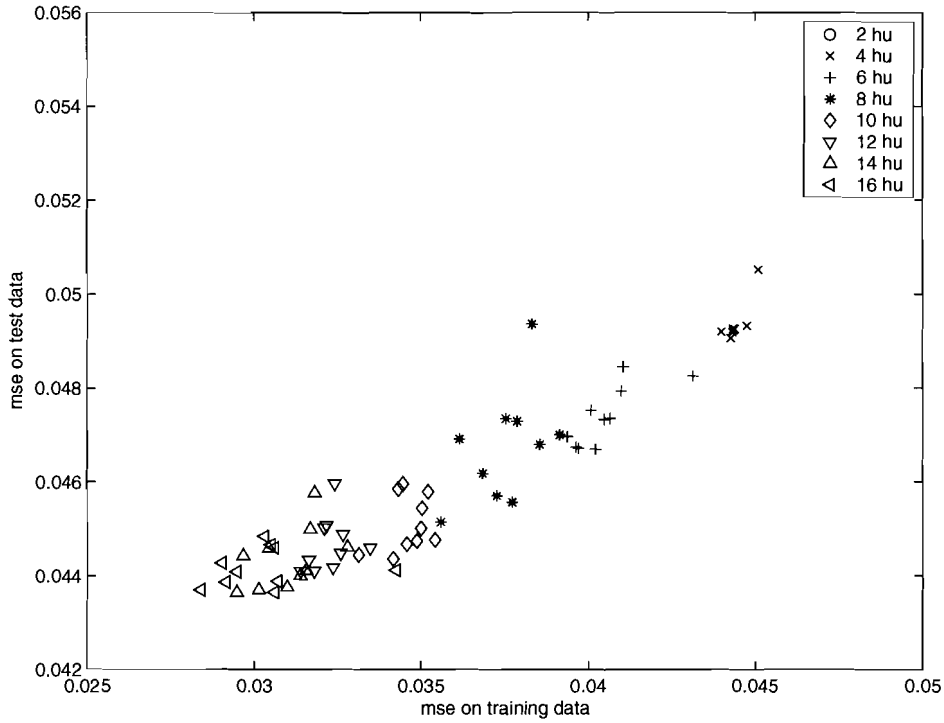


(b) mse on training data vs mse on test data

Figure 4.9: *Simulation results of networks trained with 1000 samples. The networks have different initial conditions.*



(a) model evidence vs mse on test data



(b) mse on training data vs mse on test data

Figure 4.10: *Simulation results of networks trained with 5000 samples. The networks have different initial conditions.*

Chapter 5

Bayesian sound classification

In this chapter the Bayesian framework will be used to improve a sound classifier. A sound classifier is a signal processing application, which is able to predict the composition of a sound signal in terms of different sound classes. Examples of such sound classes are speech, music and noise. This chapter starts with a short explanation of what a sound classifier is. After that the sound classifier that we will try to improve will be presented. The chapter ends with the presentation of the results of simulations that were performed on the sound classifier with the use of the Bayesian framework.

5.1 Sound classification

For many sound processing applications it is useful to know the composition of the incoming signal in terms of different sound classes. For example, in hearing aid applications a beamformer algorithm can be turned on or off depending on the presence of speech in the incoming signal. A beamformer is a method to focus to a specific direction by adapting time delays between multiple microphones. In this thesis the composition of a sound signal will be expressed by three types of sound classes respectively speech, music and noise. A sound classifier predicts for each sound class the probability that the input signal belongs to that class. Figure 5.1 shows a sound classifier represented as a block with one input and three outputs, each representing the conditional probability for a specific class.

Figure 5.1: *A sound classifier. It predicts for three different sound classes the probability that the input signal belongs to that class*

Before a sound classifier can be used it needs to be trained by examples. Training of a sound classifier can be done by extracting features from example sound files.

5.2 Kates' sound classifier

A new approach to sound classification is introduced by Kates [4]. Kates takes two steps to improve the quality of his sound classifier compared to the quality of conventional sound classifiers that are described in the literature.
First, he uses the fact that an incoming signal hardly ever consist of one sound class in the real

world. A very common situation is for example speech with background noise. Conventional sound classifiers that are described in the literature are most times trained with examples consisting of single classes. If the real world signal is a mixture of classes this means that the classifier is trained with single classes and used with mixtures of classes. Kates trains his classifier with mixtures of two sound classes to get possible performance improvements. His results are presented in Section 5.2.6 .
Second, he introduces a new feature type called *multi-channel log-level histogram*. Features used for conventional classifiers will not always be sufficient for training with two class mixtures. For example the Zero Crossing Rate (ZCR) of a signal is dominated by the strongest class in the signal. In the case of 2-class training, features are needed that contain information about both classes. The multi-channel log-level histogram contains information about both the stronger and the weaker class in a 2-class mixture. Kates claims to have found performance improvements with those new features on the following criterions:

- possibility to identify the strongest class in a 2-class mixture
- possibility to identify the weaker class in a 2-class mixture
- estimate the relative amplitude of the weaker class compared to the strongest class in a 2-class mixture

Multi-channel log-level histogram features will be explained in Section 5.2.3. Kates compares the multi-channel log-level histogram features with conventional features that are described in the literature. Figure 5.2 shows three main steps taken to make a sound classifier ready for use.

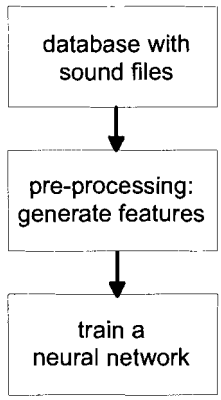


Figure 5.2: *Flow diagram of three main steps taken to make a sound classifier ready for use. First a database with example sound files should be made. Then a pre-processing step is needed to convert the example sound files into information which can be used to train a neural network.*

5.2.1 Generate features

In this subsection it will be explained how information from example files is extracted and converted into data that can be used to train a sound classifier. This process is called pre-processing. This subsection is subdivided in two parts. In the first part is explained how the different example sound files are selected and how they are combined to mixtures for further processing. The second part describes how the sample data of the mixtures is divided into small blocks that can be used for feature extraction.

Extract data from example files

The first step Kates takes to generate the features is extracting information from a database with example sound files. The file database contains:

- 13 speech files from ten native speakers of Swedish (six male and four female), with the files ranging in duration from 12 to 40 sec.
- 9 music files, each 15 sec in duration, taken from commercially-recorded classical music albums.
- 25 noise files, which can be subdivided in 4 groups. multitalker babble: 3 files, duration ranging from 111 to 227 seconds. traffic noise: 14 files, recorded from a sidewalk, duration ranging from 3 to 45 seconds moving automobile: 2 files, recorded in a moving automobile. miscellaneous: 6 files.

From each of the above three classes one file is selected at random. The three selected files are then multiplied by a gain factor. If 1-class features are needed the gain of one class is set to 1 and the gains of the other two classes are set to 0. The class with gain 1 is chosen at random. If 2-class mixtures are needed the gains are first calculated in dB values. The gain of the strongest class in the 2-class mixture is set to 0 dB, the gain of the weaker class is given a value between -30 dB and 0 dB and the class that should not be present in the mixture is given a gain value of -100.000 dB. Before the multiplication the dB values are converted to linear gains. After the multiplication with the gains the signals are added and then normalized. A schematic overview of the procedure is given in Figure 5.3

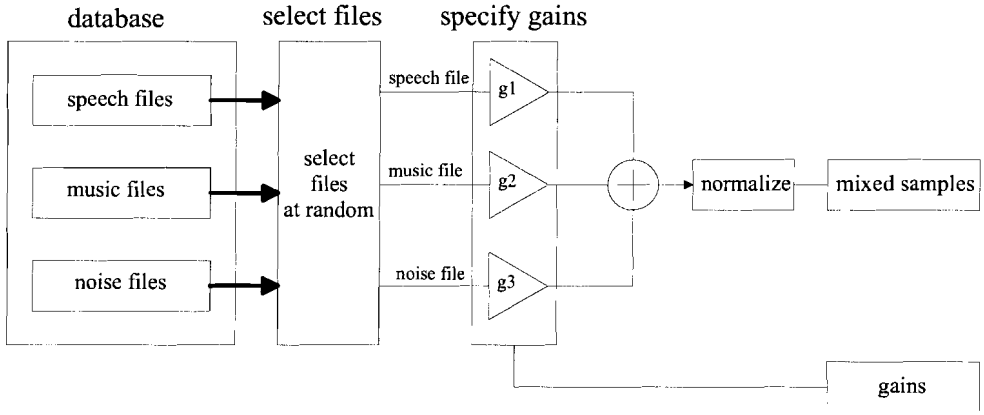


Figure 5.3: *An array of mixed samples is extracted from different sound files in a database. Mixtures are generated by specifying different gains for the files from each class.*

Extract features from the data

Feature vectors are generated by extracting information from the samples gathered in the previous step. The array of mixed samples is divided in blocks of 24 samples. These blocks are used for the calculation of several features. Examples of features are given in the next subsections. After each group of 8 blocks the features are saved. Depending on how much features should be generated with the same gains the complete process is repeated a number of times. Figure 5.4 shows a diagram of the procedure.

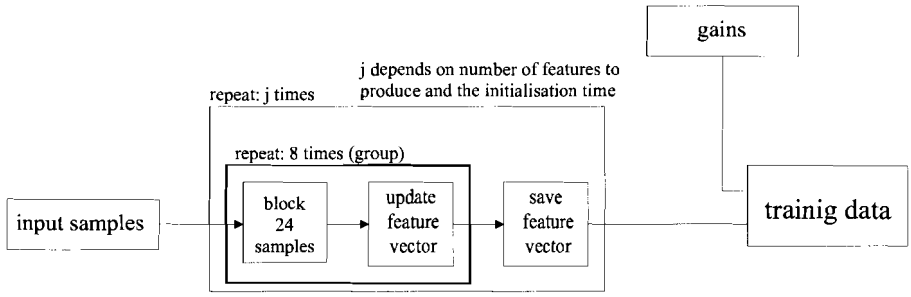


Figure 5.4: *Features can be extracted from an array of input samples. The vector is divided into blocks of 24 samples and a feature vector is updated after each block The feature vector is saved after each group of 8 blocks.*

5.2.2 Conventional features

To test the quality of the new features Kates compares his result with results using conventional features. Table 5.1 below shows the 21 conventional features used by Kates. For more specific information about these features refer to Kates [4].

5.2.3 Multi-channel log-level histograms

Kates' multi-channel log-level histogram feature can be seen as a grid of 14 by 8 squares see Figure 5.5a Each column represents a log-level (dB) histogram of a specific frequency band. In the figure there are 8 frequency bands. The range of the histograms depends on the dynamic range of the system. In this case the dynamic range is 40 dB. The width of each bin is 3 dB, which results in 14 histogram bins. Figure 5.5b shows a histogram of a specific frequency band.
The extraction of multi-channel log-level histograms is done in a few steps, which are described in the next subsubsection. An overview of the steps can be found in Figure 5.6.

Warped delay line

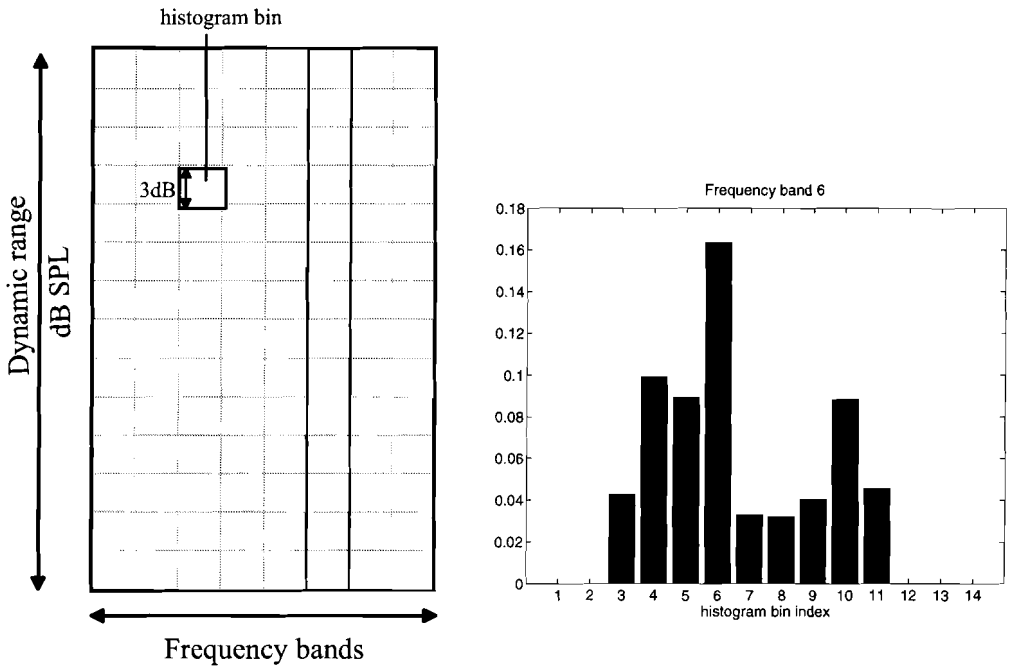
Extraction of log-level histogram features starts with frequency warping. This is a method to match the frequency resolution of a digital system to the frequency resolution of the human auditory system. A common digital system has a constant-bandwidth frequency relation, which gives a uniform spacing in the z-plane. A human auditory system has only a constant bandwidth for the lower frequencies. In the higher frequencies the bandwidth is proportional to the frequency. A good approximation of the human auditory system in digital domain can be achieved by replacing the unit elements in a digital system with all-pass filters. The Equation for an all-pass filter is

$$z \rightarrow A(z) = \frac{z^{-1} - a}{1 - az^{-1}}. \tag{5.1}$$

The frequency warping is implemented with a tapped delay line, which is then called a warped delay line. Figure 5.7 shows a warped delay line.

number	feature
1	Mean-Squared Signal Power
2	Standard Deviation of the Signal Envelope
3	Mel Cepstrum Coefficient 1
4	Mel Cepstrum Coefficient 2
5	Mel Cepstrum Coefficient 3
6	Mel Cepstrum Coefficient 4
7	Delta Cepstrum Coefficient 1
8	Delta Cepstrum Coefficient 2
9	Delta Cepstrum Coefficient 3
10	Delta Cepstrum Coefficient 4
11	Zero Crossing Rate (ZCR)
12	ZCR of the Signal 1st Difference
13	Standard Deviation of the ZCR
14	Power Spectrum Centroid
15	Delta Centroid
16	Standard Deviation of the Centroid
17	Power Spectrum Entropy
18	Broadband Envelope Correlation Lag
19	Broadband Envelope Correlation Peak
20	Four-Band Envelope Correlation Lag
21	Four-Band Envelope Correlation Peak

Table 5.1: *Conventional features used in Kates [4].*



(a) multi-channel log-level histogram grid.

(b) histogram of one frequency band (one column in fig (a).

Figure 5.5: *Overview of the multi-channel log-level histogram feature. In figure a the feature is presented as a grid containing 112 squares. Each column presents the histogram in one frequency band. In figure b the histogram of one frequency band is plotted*

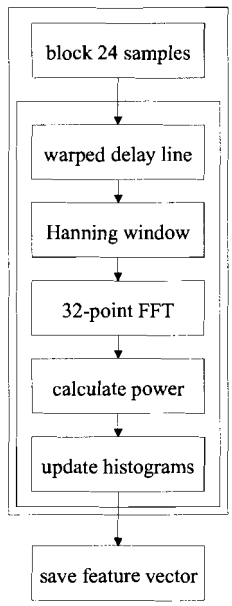


Figure 5.6: *flow diagram of the steps that are followed to extract multi-channel log-level histograms from the blocks of samples.*

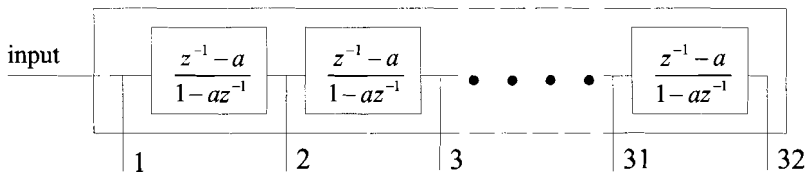


Figure 5.7: *Schematic of a tapped delay line. The delays are substituted with all-pass filters to realize frequency warping. 32 outputs are needed for a 32-point FFT in a later step.*

The number of taps in the delay line corresponds with the the size of the FFT in the third step.

Hanning window

An Hanning window is placed over the 32 output samples from the tapped delay line to smooth he output spectrum of the 32-point FFT in the next step.

$$w[n] = \frac{1}{2}\left[1 + cos\left[\frac{2\pi n}{N}\right]\right] \tag{5.2}$$

32-point FFT

A 32-point FFT (Fast Fourier Transform) is used for the conversion between time-domain and frequency-domain.

calculate power

The power spectrum is found by multiplying the results of the FFT by their complex conjugates. This results in real values. Because the power spectrum is symmetric we get 17 positive frequency bands. The power in each frequency band is accumulated after each block. Only every other frequency band is used. This results in 8 frequency bands.

update histogram

The power in dB SPL in each frequency band is converted into an histogram bin index for the histogram of that frequency band. The values in the bins on the calculated indexes will be increased. When a group of 8 blocks have been passed the histogram will be saved.

5.2.4 The classifier

For the classifier Kates uses a Two Layer Perceptron neural network. For tests with conventional features he uses a network with 16 neurons in one hidden layer and for tests with the log-level histogram features he uses a network with 8 hidden units in one hidden layer. The transfer functions at the hidden units and output units are log-sigmoid functions.

5.2.5 Kates' simulations and results

In this subsection the results of Kates' simulations will be presented. As mentioned earlier, Kates uses two methods two improve the conventional sound classifier. Training with 2-class mixtures instead of single classes and using log-level histogram features instead of other conventional features. Files containing features for the classifier were generated using the methods presented in the previous sections. Kates generates different files for training and testing the network both with single classes and 2-class mixtures. Kates does this for conventional and histogram features. Test files are generated in the same way as the training files. The same database of files and the same methods are used. The difference Kates makes between generating training and test data is that the random generator in Matlab is set to another initial value to guarantee that the data for computing the test features is picked from other locations in the example files than the data for computing the training features. In the left half of Table 5.2 the different combinations of training and test methods can be found.

When using mixtures it will be interesting to identify all classes in the mixture. In the next section the results for respectively identifying the strongest, the weaker and the non-included class in the input signals will be given.

Identifying the Stronger Signal Class

The table below presents results of how well Kates' classifier is able to identify the strongest sound class in a signal.

Training Protocol	Test Protocol	Signal Class			
		Speech	Music	Noise	Average
Conventional					
Separate Classes	Separate Classes	98.6	92.0	95.8	95.4
2-Signal Mixture	Separate Classes	98.1	91.4	86.4	91.9
Separate Classes	2-Signal Mixture	83.7	81.3	86.6	83.6
2-Signal Mixture	2-Signal Mixture	85.4	82.0	80.6	82.7
Histogram					
Separate Classes	Separate Classes	99.6	99.3	99.0	99.3
2-Signal Mixture	Separate Classes	99.6	98.3	95.1	97.7
Separate Classes	2-Signal Mixture	79.0	88.2	84.8	84.0
2-Signal Mixture	2-Signal Mixture	86.8	91.8	86.2	88.3
Histogram + Temporal					
2-Signal Mixture	2-Signal Mixture	87.3	91.2	86.2	88.2

Table 5.2: *from [4]: Percent correct identification of the signal class having the largest gain. Histogram + Conventional combines the log-level histogram with conventional features 11-13 and 18-21 in Table 5.1 .*

For all combinations of training and test methods we see an improvement of the average correct classification when using the log-level histogram features. Kates' new features thus seems to give an improvement. The other way Kates tries to improve the classifier using 2-class mixtures for training seems to give improvements if log-level histogram features are used. For training with conventional features the results become worse.

Identifying the Weaker Signal Class

The table below Table 5.3 presents results of how well Kates' classifier is able to identify the weaker sound class in a signal.

Training Protocol	Test Protocol	Signal Class			
		Speech	Music	Noise	Average
Conventional					
Separate Classes	2-Signal Mixture	23.3	42.4	71.5	45.7
2-Signal Mixture	2-Signal Mixture	46.2	44.8	53.4	48.1
Histogram					
Separate Classes	2-Signal Mixture	16.3	54.3	68.6	46.4
2-Signal Mixture	2-Signal Mixture	56.6	47.4	58.4	54.1
Histogram + Conventional					
2-Signal Mixture	2-Signal Mixture	58.6	48.0	55.9	54.2

Table 5.3: from [4]: Percent correct identification of the weaker signal class of the two-signal mixture. Histogram + Conventional combines the log-level histogram with conventional features 11-13 and 18-21 in Table 5.1.

The accuracy for identifying the weaker class in the signal is about 50 percent. Therefore, identifying the weaker class seems to be a difficult task. Again little improvements can be made by training the classifier with mixtures.

Identifying the Signal Class not included

The table below Table 5.3 presents results of how well Kates' classifier is able to identify the sound class which is not included in the signal.

Training Protocol	Test Protocol	Signal Class			
		Speech	Music	Noise	Average
Conventional					
Separate Classes	2-Signal Mixture	78.8	57.5	18.2	51.5
2-Signal Mixture	2-Signal Mixture	56.3	63.3	47.6	55.7
Histogram					
Separate Classes	2-Signal Mixture	85.4	38.9	23.4	49.2
2-Signal Mixture	2-Signal Mixture	62.8	66.9	56.1	61.9
Histogram + Conventional					
2-Signal Mixture	2-Signal Mixture	61.2	67.5	58.0	62.2

Table 5.4: from [4]: Percent correct identification of the signal class not included in the two-signal mixture. Histogram + Conventional combines the log-level histogram with conventional features 11-13 and 18-21 in Table 5.1.

The same comments as made for identifying the weaker class can be made for identifying the non-included class.

Signal Amplitudes

In the previous paragraphs Kates' results were presented of the sound classifier's capability to identify the sound classes in a signal. Kates also tested how well the classifier was able to estimate the amplitudes of the different sound classes in a mixture. When conventional features are used there is an approximate linear relationship between the input and output amplitudes in the input range of 0 to -10 dB. For input levels lower than -10 dB the neural network assigned most of the amplitudes between -20 and -30 dB. A linear regression performed on the entire data set indicates that a linear input-output model accounts for 54.1 percent of the total variance. The results when using log-level histogram features looks almost similar except for the input range between -5 and -15 dB. The network seems to be a little bit more accurate in this region. A linear regression performed on the entire data set indicates that a linear input-output model accounts now for 66.1 percent of the total variance.

Conclusions

Both methods proposed by Kates seem to give performance improvements. The log-level histogram approach is more accurate than using the conventional features and requires a smaller neural network. Training by 2-class mixtures gives a better performance when the test features given to the network are also mixtures. A more difficult task is estimating the amplitudes of the components from the two classes that make up the mixture. The greatest difficulty appears to be in estimating the amplitudes of the class components at low levels. The ideal sound classifier would identify the signal classes and give their proportions in the composite signal. The methods proposed by Kates result in a sound classifier that reaches the behavior of an ideal sound classifier better than a conventional sound classifier does. Still it is far from ideal. Especially estimating the amplitudes of the components in the mixture is a very difficult task. Therefore we further research it.

5.3 A Bayesian approach for the sound classifier

In this section we present results of simulations on Kates' sound classifier when the Bayesian framework is used in the training process of the sound classifier. Each of the following subsections will describe a different part of the Bayesian framework in the simulations.

5.3.1 Hyperparameters

As mentioned in chapter 3 and 4, in the second level of the Bayesian framework the inference of hyperparameters is performed. A posterior distribution is calculated over the hyperparameter space and the values at the modes are used as hyperparameters. We would like to test if the hyperparameter values that are found by this Bayesian approach will results in generalization performance improvements. The results of two different training processes of a sound classifier are compared with each other. In the first training process no hyperparameter optimization is performed. In the second training process the hyperparameters are interpreted as random variables and optimized by Bayesian inference during the process. The results are plotted in Figure 5.8. On the x-axis a count for the training epochs can be found and the y-axis represents the mean-squared error that the network outputs make with the targets in the data. Both the mean-squared error on the training set and the test set are plotted. The means-squared error on the test set gives an indication of the generalization performance of the network.
For both processes training and test sets are generated with the same procedures as described in Section 5.2.1. The neural networks have 8 hidden units, 112 inputs, 3 outputs and sigmoidal transfer functions in the hidden and output units. Initial values for the weights are drawn from a gaussian distribution with mean zero and variance 1. After this each initial value is divided by the square root of the number of weights in the unit where the initial weight belongs to. This is called scaling and prevents the sigmoid function to start in its saturation regions. The hyperparameters α are given an initial value of 0.01 and β is given an initial value of 10. A common used method is to start with beta larger than expected and alpha smaller, so that the learnt network starts by overfitting. When a reestimation cycle for the hyperparameters is started the Bayesian framework will chose right values for the hyperparameter. Starting with too much regularization can decrease the performance.
In Figure 5.8 can be seen that the results of the two training processes are almost equal. Optimizing the hyperparameters by Bayesian inference did not lead to performance improvements. To validate this we also calculate the classification performance of both situations. The results are presented in Table 5.5. It is obvious that the results do not differ significantly.

	Signal Class			
Gain	Speech	Music	Noise	Average
Without hyperparameter optimization				
Largest	90.1	93.9	88.8	90.9
Middle	53.5	64.2	50.1	55.9
Smallest	70.1	50.6	66.2	62.3
With hyperparameter optimization				
Largest	90.0	93.9	88.8	90.9
Middle	53.6	64.1	50.3	56.0
Smallest	70.1	50.8	66.2	62.4

Table 5.5: Percent correct identification of the signal classses. Two situations are given. One without and one with hyperparameter optimization. The results of the two situations do not show significant differences.

To investigate why the Bayesian Framework did not improve the generalization performance we will look to the propagation of the regularization hyperparameter α, the weight-decay regularizer Ew

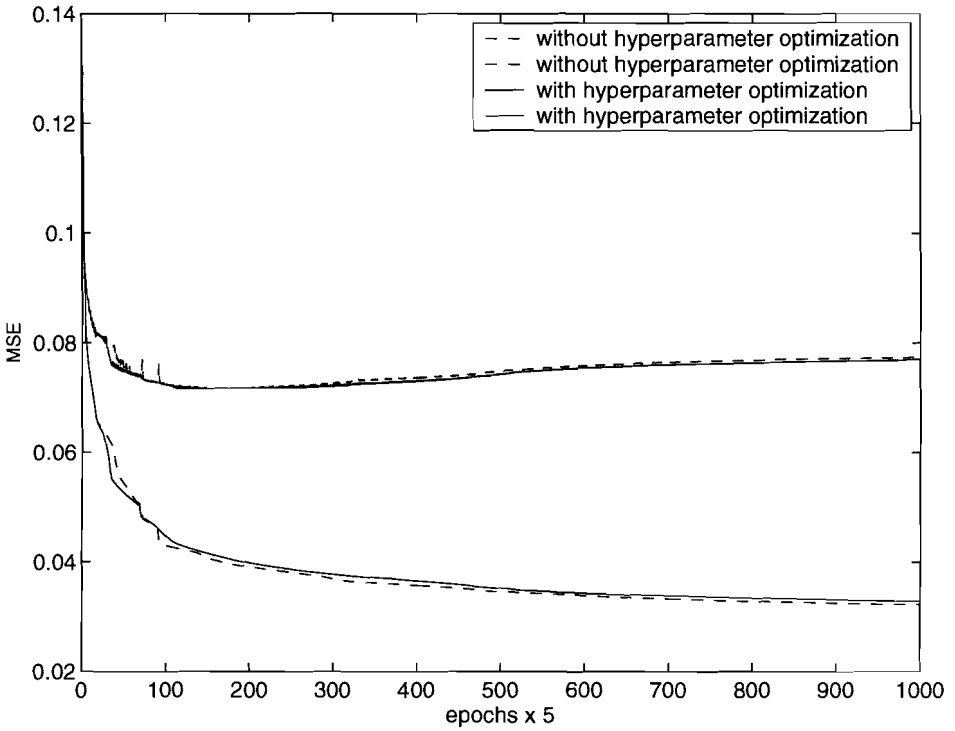
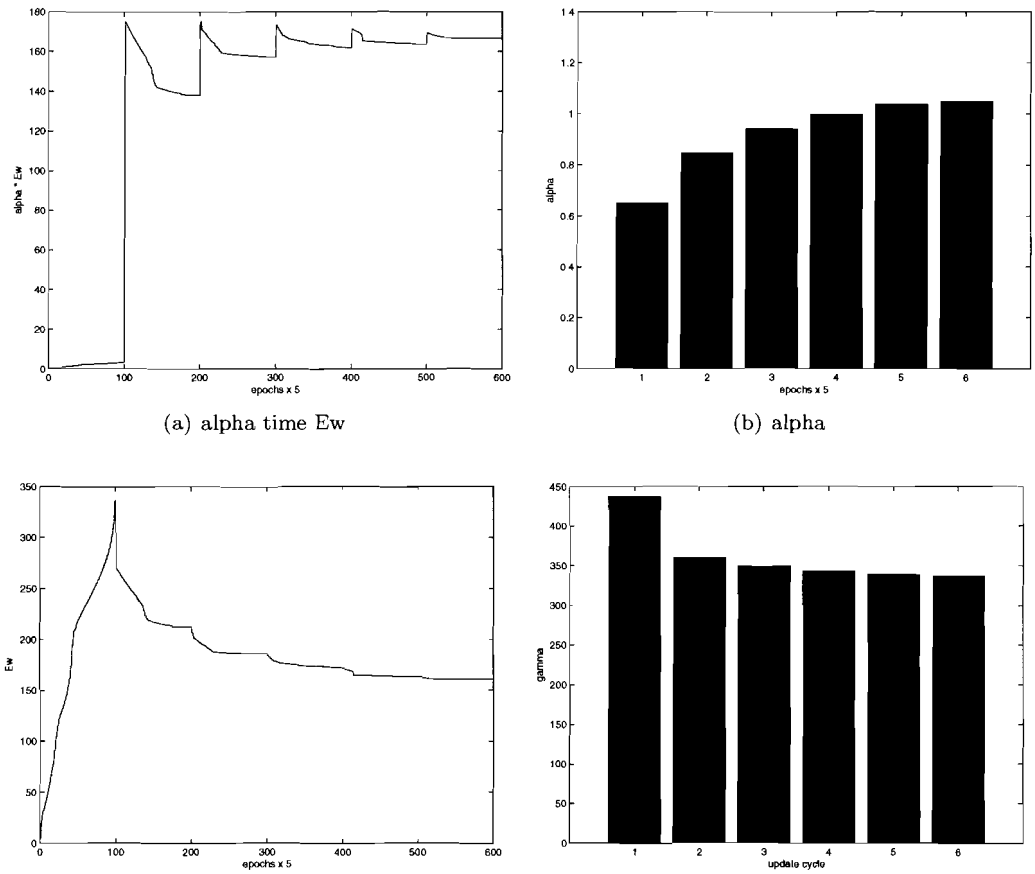


Figure 5.8: *Mean Squared Error during the training process. The solid lines represent the situation with updating of the hyperparameter and the dashed lines represent the situation with no hyperparameter update.*

and γ during the training process. As a reference we first plot these parameters for a simulation of the synthetic problem of which we are sure that regularization takes place. These plots are shown in Figure 5.9. It can be seen that regularization takes place, because E_W becomes smaller which means that the weights become smaller. Also α becomes larger which means more regularization.



(a) alpha time Ew

(b) alpha

(c) Ew

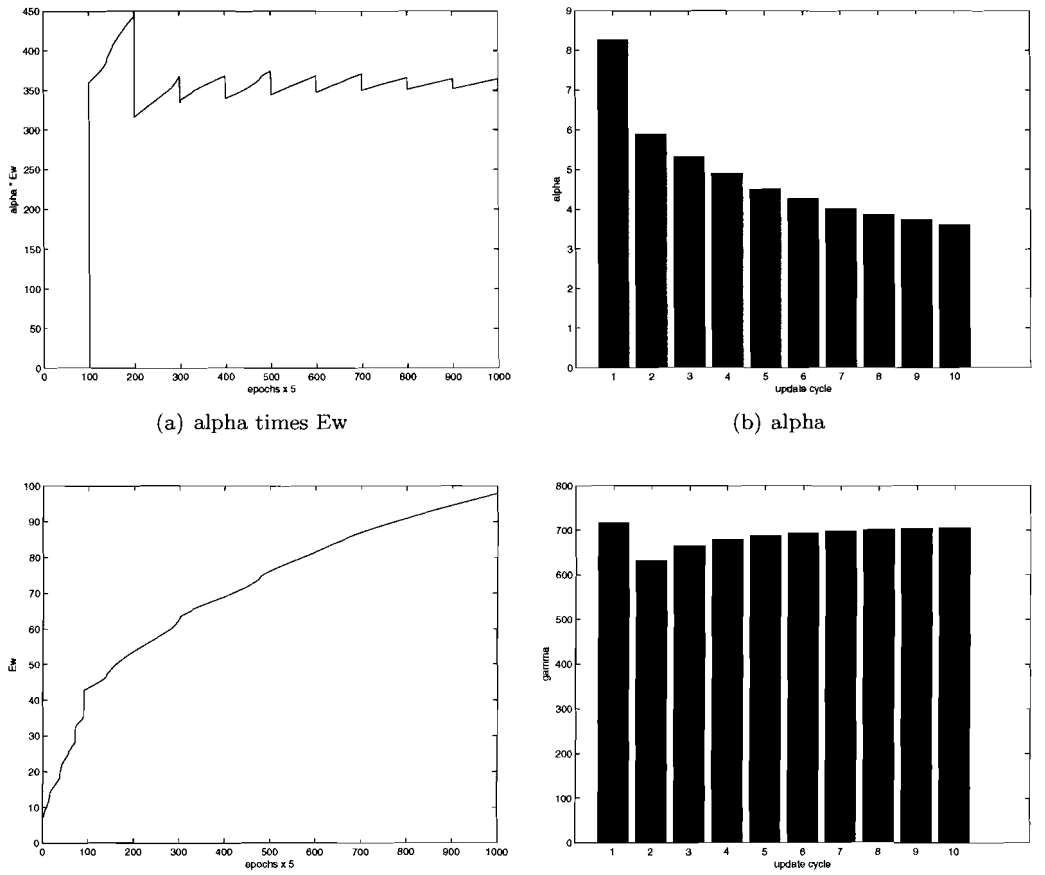
(d) gamma

Figure 5.9: *Evaluations of hyperparameters and errors during the training process of a neural network for the synthetic problem in the case of 1000 training samples.*

The plots for the simulation on Kates' classifier are shown in Figure 5.10. A big difference with the plots in Figure 5.9 is that E_W is now increasing instead of decreasing during the training process. This means that the weights become larger instead of smaller. For an intuitive explanation refer back to Equations 3.30 to 3.33. If we have a dataset with a difficult underlying process the network has to do its very best to fit the data. This results in a tight likelihood function, which results in large eigenvalues in the Hessian matrix. Large eigenvalues result in a large γ, but small αs will not have much influence on γ. If in Equation 3.32 the denominator $2E_W$ increases faster than the nominator γ then α will decrease after each iteration.

5.3.2 Feature selection with ARD

The simulations in the previous subsection were performed with a single hyperparameter for the weight priors. In Automatic Relevance Determination (ARD) every input is given a separate hyperparameter. This can lead to further performance improvements, because the influence of irrelevant inputs on the outputs is further reduced. In the simulations on the sound classifier



(a) alpha times Ew

(b) alpha

(c) Ew

(d) gamma

Figure 5.10: *Evaluations of hyperparameters and errors during the training process of a neural network for Kates' classifier in the case of 40000 training samples.*

ARD did not lead to further performance improvements. We can interpret this as that there are not much irrelevant inputs in the input feature vectors.

5.3.3 Error bars around the outputs of the sound classifier

In this subsection simulations are performed where error bars are predicted around the outputs of the sound classifier. Training and test data sets are generated in the same way as in the previous subsection, except that only speech and noise are used now. The network also consists of the same number of units, but the transfer functions in the output units are now linear. The results on output 1 (speech) are presented in Figure 5.11. The error bars together with the absolute errors between the values on the network output and the targets are plotted in a separate figure, Figure 5.12, for easier evaluation. It can be observed in this figure that for none of the data samples the error bar is larger than the absolute error. The error bars are thus too small.

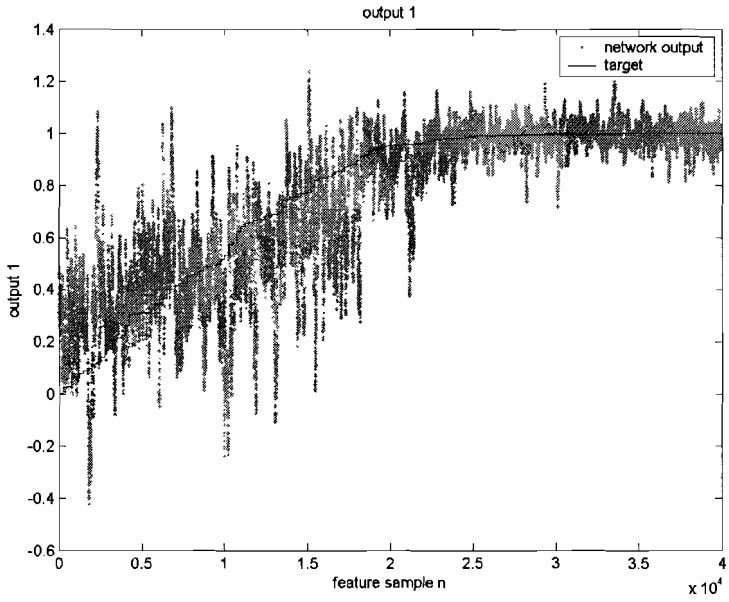


Figure 5.11: *Results on output 1 (speech) when test data is provided to the neural network. The values on the y-axis are converted dB values. The region of -30dB - 0dB is fitted to a region of 0 to 1.*

The equation for calculating error bars was given in Equation 3.44. It was built from two components. The first component represented the error due to the intrinsic noise in the data and was constant. The second component was dependent on the input data provided to the network. In Figure 5.12 the first component can be observed as a certain threshold level around 0.11. The error bars are strongly dominated by this component.

For error bars on the synthetic problem we provided test data to the network from a range in the data space from where no data had been provided to the network during training. For further investigation we will do something similar here now by training the classifier with mixtures of speech and noise and testing it with mixtures of music and noise. We expect that the network is less sure about its predictions if the larger component in the input signal is music. Results can be found in Figure 5.13 and Figure 5.14. It can be observed that also in this situation the error bars are too small. However, there is a clearly observable difference between the error bars for the first half of the test set and the second half.

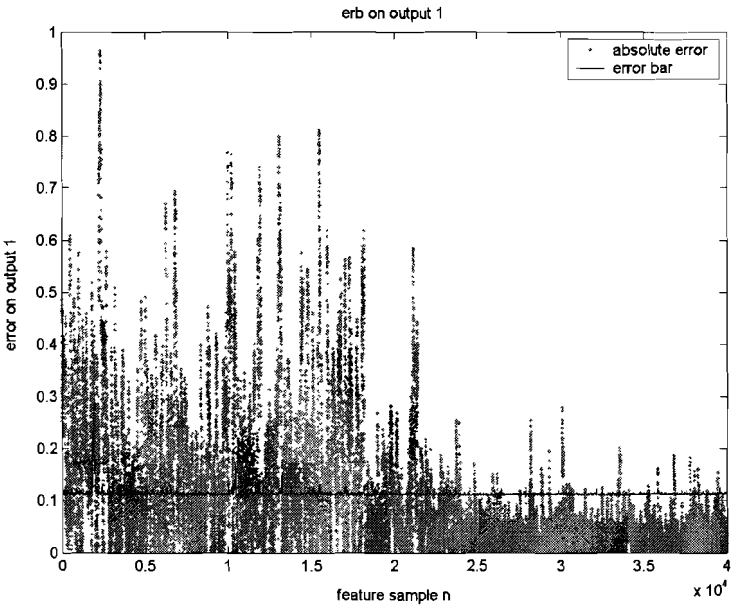


Figure 5.12: *Absolute error versus estimated error bars on output 1 (speech) of the classifier*

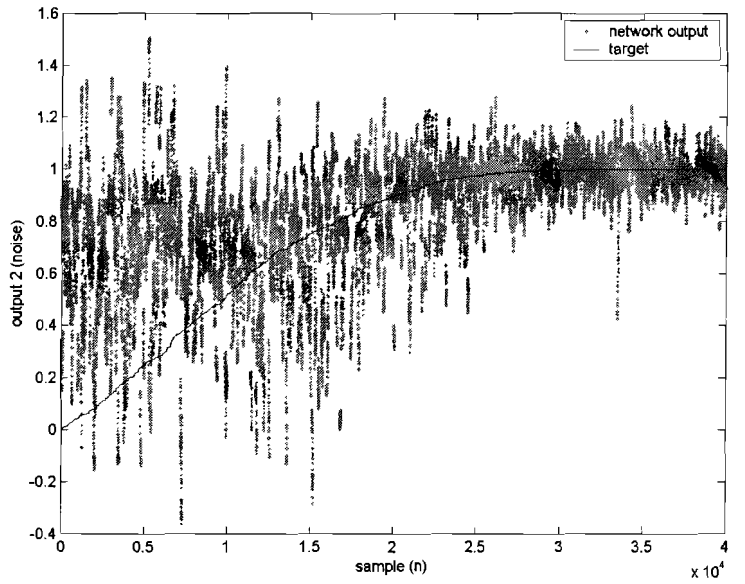


Figure 5.13: *Results on output 1 (speech) when test data is provided to the neural network. The values on the y-axis are converted dB values. The region of -30dB - 0dB is fitted to a region of 0 to 1.*

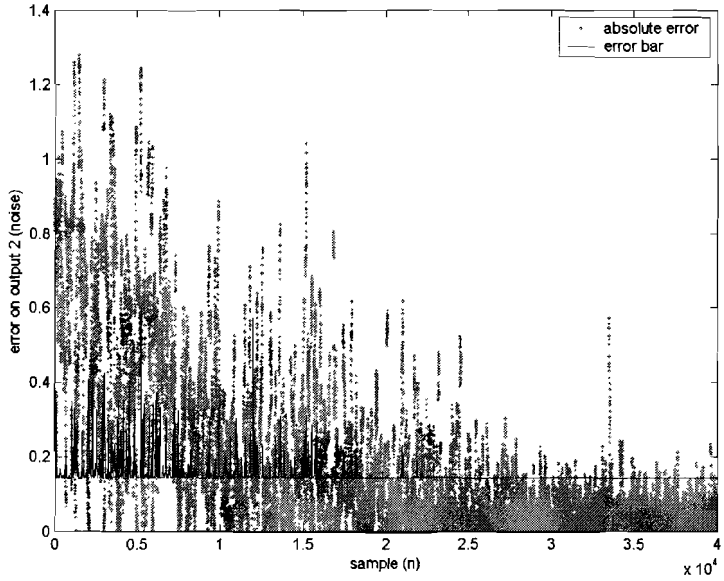


Figure 5.14: *Absolute error versus estimated error bars on output 2 (noise) of the classifier*

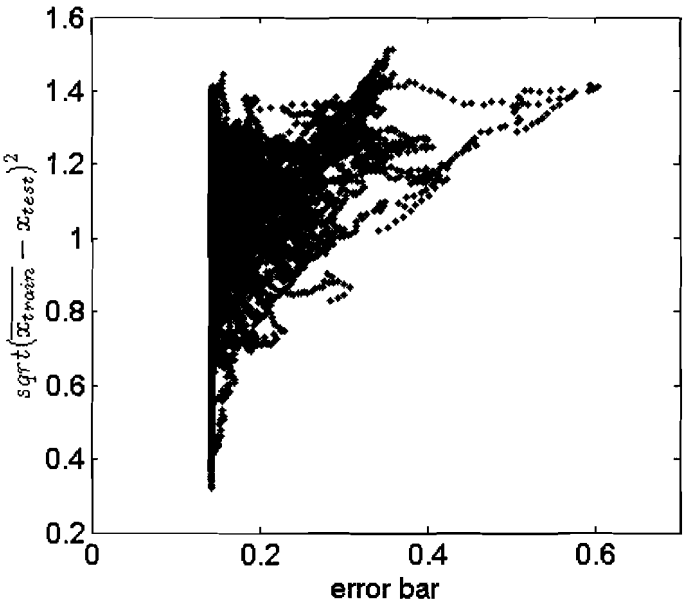


Figure 5.15: *Distance between test input and mean training input versus size of the error bar*

As a last experiment on error bars we will try to find a relation between the distance in data space between the test data and the training data and the size of the error bars. Figure 5.15 shows for each sample the distance between the mean training input data and the test input versus the size of the calculated error bar. It can be observed in this Figure that when the error bar is large the distance to the mean training data is also large. A large distance to the mean training data does not automatically result in a large error bar.
The approximation method for error bars as described in Section 3.3.1 makes several assumptions. A gaussian noise model with zero mean is expected on the outputs. The posterior distribution is also expected to be Gaussian and it is expected that the width of the posterior distribution is sufficiently small such that a linear expansion around $y(\mathbf{x}; \mathbf{w}_{\mathbf{MP}})$ can be used to estimate $y(\mathbf{x}; \mathbf{w})$. We expect that it is somewhere in these assumptions that the approximation fails for the sound classifier, because the input to output relations in the sound classifier are very complex.

5.3.4 Model selection

Kates uses a neural network with 16 neurons in the hidden layer for experiments with conventional features and for the experiments with the log-level histogram features he uses a neural network with 8 hidden units in the hidden layer. The Bayesian framework is able to make a decision about the correct number of hidden units. Overcomplex networks are given a penalty. Because of the bad results with the synthetic problem in the model comparison part pruning will be used in this chapter to find a correct sized network.

5.4 Optimization by using pruning with a Laplace prior

In this subsection simulations will be performed where the pruning method as described in Williams [16] is used for trying to improve the classifier's network. The same type of datasets are used as in Subsection 5.3.1. We also start with a network of the same type and size.
In Figure 5.16 the errors of two simulations are plotted during their training process. As can be seen in the figure the generalization performance has not been increased after the training process. Although, the network size is significantly reduced from 931 to 600 weights. We thus get a smaller networks for the classifier without loosing performance.
To verify that we did not loose classification performance we calculate again the classification performances for both situations. The results are plotted in Table 5.6.

	Signal Class			
Gain	Speech	Music	Noise	Average
Without hyperparameter optimization and pruning				
Largest	91.6	93.5	89.7	91.6
Middle	49.2	62.2	56.6	56.0
Smallest	74.1	54.4	59.7	62.7
With hyperparameter optimization and pruning				
Largest	91.2	95.3	87.6	91.4
Middle	51.3	59.7	58.0	56.3
Smallest	67.4	58.8	60.6	62.3

Table 5.6: Percent correct identification of the signal classses. Two situations are given. One without and one with hyperparameter optimization. The results of the two situations do not show significant differences.

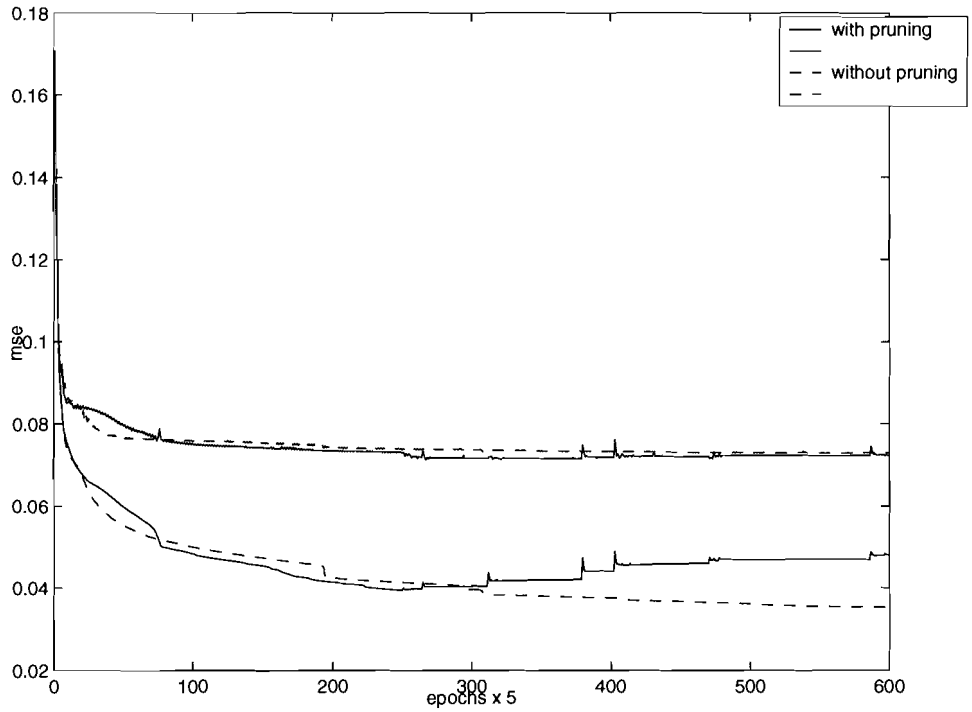


Figure 5.16: *Error during the training processes of two simulations. One with and one without pruning.*

Chapter 6

Conclusions

The possible benefits of using the Bayesian framework in the learning process of the sound classifier have been investigated. We started with simulations on a synthetic problem. After that the real sound classifier was investigated. Here we also start with the conclusions on the synthetic problem. After that the conclusions on the sound classifier will be discussed.

The first subject that we investigated was weight regularization. We compared two situations, with and without optimization of the regularization parameters during the training process. For both situations the mean squared error on the network outputs for the training and for a test set were measured during the training process. The results of two simulation were plotted in Figures 4.4a and 4.4b. These figures show that optimization of the regularization parameters leads to generalization improvements. The difference between the error on the training data and the error on the test data is smaller when the regularization parameters are optimized during the training process. The Bayesian inference framework thus performs well as a method to optimize the regularization parameters in the neural networks for the synthetic problem.
The same figures show that the benefits we get from using Bayesian inference depends on the composition of the training data set. In the situation where we used a small and incomplete training set (1000 samples) the difference between the situation where Bayesian inference was used and the situation where Bayesaian inference was not used was much larger then in the case where a larger and complete training set (5000 samples) was used.

The second subject that has been investigated was Automatic Relevance Determination of the different features. Automatic Relevance Determination did lead to small performance improvements compared to situations where a single or four regularization parameters were used.

The third subject was model comparison by model Evidence. The relation between the model evidence and the mean squared error on a test data set have been investigated. In an ideal situation a (anti-) proportional relation was expected. Results were plotted in Figure 4.7 and Figure 4.8. In Figure 4.8, where 5000 training samples were used, a (anti-) proportional relation was visible. In Figure 4.7, where 1000 samples were used, no obvious relation was visible. These results are conform statements in Penny [11] where Penny says that: "Model selection using the evidence or training error is only tenable if the number of training examples exceeds the number of network weights by a factor five or ten."

The fourth subject was Error bars around the outputs of the neural network. This was tested for the synthetic problem with a simple situation where we left out one half of the dataspace from the training samples. Error bars where larger when data was presented to the neural network that was lying in the half of the dataspace that has not been observed during the training process. Figures with error bars were plotted in Figure 4.7 and Figure 4.8.

In the simulations with Kates' sound classifier Bayesian inference with Mackay's Evidence Framework did not lead to significant improvements. With datasets of the same size as Kates uses no improvements were measured. With smaller datasets some improvements where measured when Bayesian inference was used compared to a situation with no Bayesian inference.

Error bars on the outputs of Kates' classifier did not result to a reliable representation of the error on the network output. In the case that we trained with mixtures of speech and noise and tested with music and noise we observed error bars that were larger than the intrinsic noise when samples containing music were presented to the network. We also saw that there was a relation between the eucledian distance between the test input vector and the mean training input vector and the size of the error bars. In cases of large error bars this distance was also large. A large eucledian distance did not automatically result in a large error bar.

As an alternative for model selection with the Evidence Framework we implemented a method called Pruning with a Laplace prior. This method led to a significant reduction of the network size. Significant classification improvements were not achieved with this method.

Our overall conclusion is that the Evidence Framework as presented by Mackay is a nice theoretical framework for Bayesian learning. On a synthetic problem it has been shown that using Mackay's Bayesian Framework can lead to performance improvements. For Kates' sound classifier the Evidence Framework did not lead to significant performance improvements. Although, the Bayesian method "Bayesian Regularization and Pruning using a Laplace Prior" from [16] led to performance improvements by reducing the network size.

In the Evidence Framework a lot of approximations are used of which the validity in practical situations could not always be easily checked. For future work it is recommended to compare results using the Evidence Framework with results using Markov Chain Monte Carlo simulations, as described in Neal [10].

Bibliography

[1] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, 1985.

[2] C. M. Bishop. *Neural Networks: A Pattern Recognition Perspective.* Oxford University Press, 1996.

[3] E. T. Jaynes. Probability theory as logic. *Maximum-Entropy and Bayesian Methods, P. F. Fouge (ed.), Kluwer, Dordrecht,* 1990.

[4] J. M. Kates. Sound classification using log-level histograms. 1, 2005.

[5] S. Z. Li L. Lu, H. Zhang. Content-based audio classification and segmentation by using support vector machines. *Multimedia Systems,* 8, 2003.

[6] D. J. C. MacKay. Bayesian interpolation. *Neural Computation,* 4(3):415–447, 1992.

[7] D. J. C. MacKay. A practical Bayesian framework for backprop networks. In S. J. Hanson J. E. Moody and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4,* pages 839–846, 1992.

[8] D. J. C. MacKay. Hyperparameters: Optimize, or integrate out? In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Barbara 1993,* pages 43–60, Dordrecht, 1996. Kluwer.

[9] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2005.

[10] R. M. Neal. Bayesian training of backpropagation networks by hybrid monte carlo method. Technical report, Department of Computer Science, University of Toronto, 1992.

[11] W. D. Penny. Bayesian neural networks for classification: how useful is the evidence framework? *Neural Networks,* 12:877–892, 1998.

[12] R. Rojas. *Neural Networks A Systematic Introduction.* Springer-Verlag, 1996.

[13] A. C. Surendran. Logistic discriminative speech detectors using posterior snr. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings 5, pp. V-625-V-628,* 2004.

[14] H. H. Thodberg. A review of bayesian neural networks with an application to near infrared spectroscopy. *IEEE Transactions on Neural Networks,* 7(1):56–72, 1996.

[15] M. E. Tipping. Bayesain inference: An introduction to principles and practice in machine learning. *Advanced Lectures on Machine Learning,* pages 41–62, 2003.

[16] Peter M. Williams. Bayesian regularisation and pruning using a Laplace prior. Technical Report 312, 1994.

[17] R. D. Yates and D. J. Goodman. *Probability And Stochastic Processes A Friendly Introduction for Electrical and Computer Engineers.* Wiley, 2005.