NEURAL COMPUTING
NCRG
RESEARCH GROUP

Neural Computing Research Group
Dept of Computer Science & Applied Mathematics
Aston University
Birmingham B4 7ET
United Kingdom
Tel: +44 (0)121 333 4631
Fax: +44 (0)121 333 4586
http://www.ncrg.aston.ac.uk/

Regularisation of Mixture Density Networks

Lars U Hjorth
Neural Computing Research Group
Aston University, BIRMINGHAM, B4 7ET, UK
email:hjorthl@aston.ac.uk

Technical Report NCRG/99/004 February 12, 1999

Abstract

Mixture Density Networks are a principled method to model conditional probability density functions which are non-Gaussian. This is achieved by modelling the conditional distribution for each pattern with a Gaussian Mixture Model for which the parameters are generated by a neural network. This technical report presents a novel method to introduce regularisation in this context for the special case where the mean and variance of the spherical Gaussian Kernels in the mixtures are fixed to predetermined values. Guidelines for how these parameters can be initialised are given, and it is shown how to apply the evidence framework to mixture density networks to achieve regularisation. This also provides an objective stopping criteria that can replace the 'early stopping' methods that have previously been used. If the neural network used is an RBF network with fixed centres this opens up new opportunities for improved initialisation of the network weights, which are exploited to start training relatively close to the optimum. The new method is demonstrated on two data sets. The first is a simple synthetic data set while the second is a real life data set, namely satellite scatterometer data used to infer the wind speed and wind direction near the ocean surface. For both data sets the regularisation method performs well in comparison with earlier published results. Ideas on how the constraint on the kernels may be relaxed to allow fully adaptable kernels are presented.

Acknowledgements

The first people I would like to thank are Jan-Erik Lundkvist at Linkoping University who enabled me to participate on this MSc program and to all the staff at the University that have given me a high quality eductation during my time there.

I am very grateful to my supervisor, Dr Ian Nabney, who has patiently explained to me what I was unable to understand and who has greatly influenced the quality of this work. He have also given invaluable comments on early drafts and helped me translate my Swenglish writing into English.

I thank Chris Bishop whos work has been a great source of inspiration and understanding.

I am thankful to two fellow students, David Evans and Guillaume Ramage. They have very helpful with information about the wind data set together with Dr Dan Cornford.

I am grateful to Dr Chris Williams, Dr David Saad and all other staff at Aston that have been helpful and provided me with various bits and pieces of advice.

And at last, but not least, I would like to thank Ulf Merell for providing me with a laptop during the project that has been very much appreciated.

Contents

1 Introduction

This project originally had two objectives. The first was to find more efficient ways of training Mixture Density Networks. During the spring of 1998, a fellow student (Evans, 1998a), discovered that much of the slowness in the training was due to an implementation problem in the software. He implemented a new version which is $10 - 50$ times faster. This enhancement made it feasible to train models with c. 10000 patterns, which was the original goal for the problem we had in mind. Therefore my work focussed on the second objective; providing effective regularisation for these models.

The relevant background is reviewed in Section 2. The material is presented in a very compact form and if the concepts are new, I recommend reading of the original papers for a better understanding. This section also serves as an introduction to the notation used throughout this technical report. I have assumed that the reader is familiar with concepts like the likelihood, clustering algorithms (EM algorithms, k-means), basic knowledge about neural networks (multi layer perceptron networks, radial basis function networks) and some knowledge about general multipurpose non-linear optimisation techniques (quasi-Newton, conjugate gradients). Bishop (1995) is a suitable reference for this material.

In the third section I present a novel way to regularise Mixture Density Networks with the constraint that the mean and variance of the Gaussian kernels in the mixture are fixed to predetermined values. I also present an effective procedure to initialise these parameters.

Next, in the fourth section, I show that the regularisation method discussed in Section 3 gives good estimates of the posterior distribution, firstly for a synthetic and secondly for a real life data set involving prediction of the wind speed at the sea surface from radar scatterometer data gathered by a satellite.

Finally, in the last section I discuss the advantages and the disadvantages of this approach compared with earlier work. I also give a brief outline of directions for future work and conclude with a short summary.

1.0.1 Notation Used

Wherever possible I have tried to used a coherent notation

c	number of network outputs; number of classes
d	number of network inputs
$\mathcal{D}$	dataset consisting of input vectors $\mathbf{x}$ and targets $\mathbf{t}$
E	error function
η	mixing coefficient
i,j	mixture labels
k,l	output dimension labels
r,s	hidden unit labels
n	pattern label
M	number of mixtures
$\mathrm{P}(\cdot)$	probability
$\mathrm{p}(\cdot)$	probability density function
$\mathbf{t}$	target value
$\mathbf{w}$	parameter vector for the network weights.
$\mathbf{x}$	network input variable
$\mathbf{z}$	network output variable
ln	logarithm to base e

2 Background

In this section I present a review of the relevant background for this technical report. Since this work is trying to merge conditional mixture models with regularisation theory the literature survey quickly becomes rather broad since in both areas several relevant papers have been written. The idea of modelling both the conditional mean and variance in order to provide error bars for the predictions goes back to Nix and Weigend (1994). First we look at one generalisation of this, the Mixture Density Network (Bishop, 1994), which uses a mixture of spherical Gaussians. I then move on to the evidence approximation (MacKay, 1992a; MacKay, 1992d; MacKay, 1992c) which gives a good method of controlling model complexity for least-squares regression models. This framework has been extended in (Bishop and Qazaz, 1997) to handle input dependent noise for a single (spherical) Gaussian. This leads us to review a paper (Williams, 1996) that deals with a single multivariate Gaussian. Here Williams models the complete covariance matrix and finally we look at a method to initialise our network weights by (Nabney, 1998) for generalised linear models, that I will adapt to the novel theory for Mixture Density Networks in the next section.

2.1 Mixture Density Networks

Before starting to explain what a Mixture Density Network (*MDN*) is, it is useful to look at what happens if we use a standard multi layer perceptron (*MLP*) network on a data set that contains multiple branches, *i.e.* for some x values $f(x)$ has two or more distinct values, so $f(x)$ is not a function but a multi-valued relation. The data set used is taken directly from (Bishop, 1994) and the predictions made by the network can be seen in Figure 1.

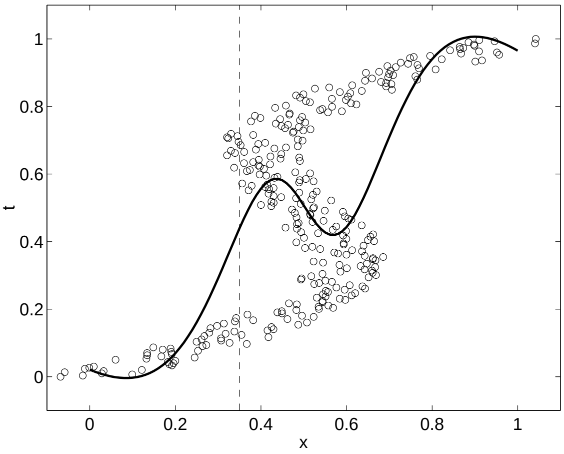


Figure 1: The data was generated by the function $x = t + 0.3\sin(2\pi t) + 0.2\epsilon$ where ϵ is Gaussian noise with zero mean and unit variance. The solid line corresponds to predictions from a *MLP* network with 8 hidden units after training it for 80 iterations with a quasi-Newton optimiser. For some x values the predictions are very poor; this happens because the *MLP* models the average of the conditional probability density which is not a useful statistic for those values of x where the function has more than one branch. The dotted line corresponds to $x = 0.35$; the predicted conditional density for this input is plotted in Figure 2.

It is easy to see that the predictions made are not satisfactory but why does it not work? The answer to that question is that an *MLP* network predicts the conditional average of the target data (Bishop, 1995) and this limited statistic is not appropriate for this kind of data set. One way to solve this problem is to model the complete conditional probability density instead, and this is the approach used by Mixture Density Networks. An example of this can be seen in Figure 2. It is from the same problem as was used for Figure 1 and it shows the conditional probability density

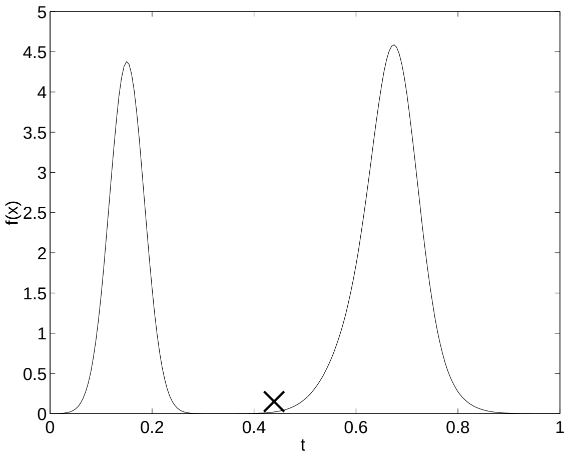


Figure 2: The graph shows the conditional probability distribution for $x = 0.35$ predicted by a Mixture Density Network. The cross corresponds to the prediction made by the *MLP* network and it is clearly not a good prediction, since it is far from both peaks that correspond to the two branches of the relation. Note that the peaks correspond well with t-values that have high data density along the line $x = 0.35$ in Figure 1.

estimate of an *MDN* for $x = 0.35$. The prediction of the *MLP* network for that x value has been included as a cross. It is important to note that instead of making a prediction of a single value we now estimate a whole distribution. The distribution in the example is clearly bi-modal, and the prediction made by the *MLP* is poor since in this case the average of the solutions was not itself a solution. If we want to visualise the results for all x values a contour plot of the conditional probability density is needed, and it can be seen in Figure 3.

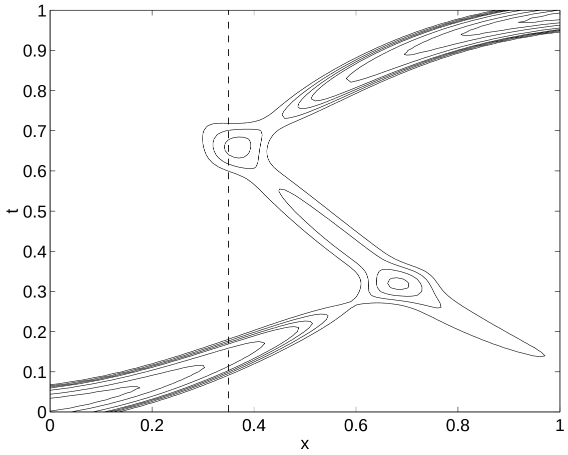


Figure 3: The contour plot shows the conditional probability distribution for $x \in [0, 1]$ predicted by a Mixture Density Network trained on the same problem as in Figure 1. Note that the slice of this plot for which $x = 0.35$ corresponds to the plot in figure 2

The Mixture Density Network was first introduced by (Bishop, 1994) and the review of them in the next section is based on that technical report.

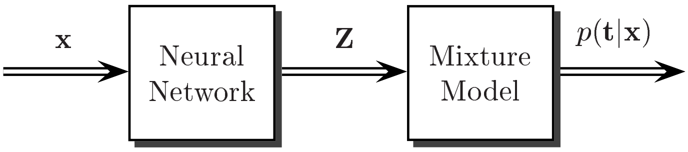


Figure 4: A block diagram showing the structure of a *MDN*. When a pattern $\mathbf{x}$ is presented to the network, a parameter vector $\mathbf{Z}$ is generated as the outputs of the network, which in turn is used as input to the *Mixture Model* to generate the conditional probability, $p(\mathbf{t}|\mathbf{x})$.

2.1.1 The class of Mixture Density Networks

In its broadest meaning, a *MDN* can be seen as *using a Neural Network to generate some parameters for a Mixture Model.* This is illustrated in Figure 4.

Given an input vector, $\mathbf{x}$, the *Neural Network* generates a parameter vector, $\mathbf{Z}$, that is used as input for the *Mixture Model*, which will generate a conditional probability density, $p(\mathbf{t}|\mathbf{x})$, as its output. We can interpret the standard *MLP* as modelling the conditional distribution by a Gaussian with fixed variance and input dependent mean. This distribution can be used to make predictions in a number of ways. One would be to take the average, which would lead us back to the *MLP* model again. Another could be to use non-linear optimisation to find the highest mode.

We now have two black boxes, the *Neural Network* and the *Mixture Model* that need closer examination. In (Bishop, 1994) an *MLP* network was used together with a spherical Gaussian *Mixture Model* and the same choice is made in this review.

Likelihood Suppose that the data set is a collection of independent samples drawn from a fixed distribution and is a set of tuples $\mathcal{D} = \{\mathbf{x}^n, \mathbf{t}^n\}$ where n is a label running over all patterns. The likelihood can then be written as

$$\mathcal{L} = \prod_{n=1}^{N} p(\mathbf{t}^n, \mathbf{x}^n) = \prod_{n=1}^{N} p(\mathbf{t}^n|\mathbf{x}^n)p(\mathbf{x}^n), \tag{1}$$

where N is the number of examples. The factor $p(\mathbf{x}^n)$ can be omitted since it is independent of the model parameters and it will appear as a constant in the subsequent analysis.

The remaining factor can be recognised as the output of the *Mixture Model* when its parameters are generated as in Figure 4. We choose to model $p(\mathbf{t}^n|\mathbf{x}^n)$ by

$$p(\mathbf{t}^n|\mathbf{x}^n) = \sum_{i=1}^{M} \eta_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n). \tag{2}$$

ϕ_i is, in this case, a spherical Gaussian density function, η_i is the *mixing coefficient* for that Gaussian and the index i runs over all M kernels in the mixture. This linear combination of kernels can in principle model any conditional probability density, given a sufficient number of kernels with correctly set parameters (McLachlan and Basford, 1988).

Taking the negative log likelihood of the relevant part of (1) yields the cost function

$$E = -\sum_{n=1}^{N} \ln p(\mathbf{t}^n|\mathbf{x}^n) = -\sum_{n=1}^{N} \ln\left\{\sum_{i=1}^{M} \eta_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n)\right\}. \tag{3}$$

The next thing to ensure is that the network outputs that will be used as the means, variances and mixing coefficients in the mixture model are constrained so that $p(\mathbf{t}^n|\mathbf{x}^n)$ can always be guaranteed to be a probability distribution.

2.1.2 Constraining the outputs of the Network

The *MLP* network has linear output functions, and if we want to be able to interpret the posterior as a probability density function we need to place some constraints on these outputs. First we need the definition of a spherical Gaussian in c dimensions

$$\phi_i(\mathbf{t}|\mathbf{x}) = \frac{1}{(2\pi)^{c/2}\sigma_i(\mathbf{x})^c} \exp\left(-\frac{||\mathbf{t} - \mu_i(\mathbf{x})||^2}{2\sigma_i(\mathbf{x})^2} \right), \tag{4}$$

where μ_i and σ_i^2 are the mean and variance respectively. The mean is a vector of dimension c, and the standard deviation, σ_i, should always have a positive real value.

Means The means of a Gaussian can take on any real value, exactly the same range as for a network output, so the 'connection' between them is straightforward:

$$\mu_{ik} = z_{ik}^{\mu} \qquad \text{for } k = 1, 2, \ldots, c, \tag{5}$$

where z denotes an output from the network. One network output is required for each component of the mean for each kernel, giving a total of Mc outputs.

Variances The standard deviation has the constraint of being positive, implying that this could be modeled by an exponential function,

$$\sigma_i = \exp(z_i^{\sigma}). \tag{6}$$

This also has the advantage that it discourages the variance from going to zero which would cause the likelihood to go to infinity.

Mixing Coefficients For the mixing coefficients we have to ensure that $p(\mathbf{t}^n|\mathbf{x}^n)$ is positive which leads to $\eta_i \geq 0$ and we also have to ensure that $p(\mathbf{t}^n|\mathbf{x}^n)$ integrates to one as a function of $\mathbf{t}^n$. This property is known to hold for a Gaussian kernel and to ensure the same property for our mixture of Gaussians the only additional requirement is that the non negative mixing coefficients also sum to one,

$$\sum_{i=1}^{M} \eta_i = 1. \tag{7}$$

This can be achieved by applying the 'softmax' transformation (Bridle, 1990) to the network outputs.

$$\eta_i = \frac{\exp(z_i^{\sigma})}{\sum_{i'=1}^{M} \exp(z_i^{\sigma})}. \tag{8}$$

Total Number of Outputs After establishing the constraints we can see that Mc outputs are needed for the means and M for the variances and mixing coefficients respectively, leaving us with a total number of $M(c+2)$ outputs compared with c for neural networks used in the conventional way.

2.1.3 Choosing a Network Architecture

To generate these three different kind of outputs Bishop used a standard multi layer perceptron network (*MLP*) with a single hidden layer. To train the network with standard non-linear techniques we need an expression for the gradient of the error function, as this will speed up the optimisation. Appendix A.1 contains the calculations in detail and only the result is stated here. The error function for one pattern $\{\mathbf{x}^n, \mathbf{t}^n\}$ is

$$E^n = -\ln \sum_{i=1}^{M} \eta_i(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n) \tag{9}$$

and the total error can be written as

$$E = \sum_{n=1}^{N} E^n. \tag{10}$$

The gradients are

$$\frac{\partial E^n}{\partial z^{\mu}_{ik}} = -\pi_i(\mathbf{x}^n)\frac{(\mu_{ik}(\mathbf{x}^n) - t^n_k)}{\sigma_i(\mathbf{x}^n)^2} \qquad \text{for } k = 1, 2, \ldots, c, \tag{11}$$

$$\frac{\partial E^n}{\partial z^{\sigma}_{i}} = -\pi_i(\mathbf{x}^n)\left(\frac{||\mathbf{t}^n - \mu_i(\mathbf{x}^n)||^2}{\sigma_i(\mathbf{x}^n)^2} - c\right), \tag{12}$$

$$\frac{\partial E^n}{\partial z^{\eta}_{i}} = \eta_i(\mathbf{x}^n) - \pi_i(\mathbf{x}^n), \tag{13}$$

where $\pi_i(\mathbf{x}^n)$ is defined to be the posterior probability (or 'responsibility') that the ith mixture component generated the data point. For convenience and clarity the dependence on $\mathbf{x}^n$ and $\mathbf{t}^n$ will from now on be omitted where it is unlikely to create confusion, $\eta_i(\mathbf{x}^n)$ will be written as η_i, $\phi_i(\mathbf{t}^n|\mathbf{x}^n)$ as ϕ_i etc. So

$$\pi_i = \frac{\eta_i\phi_i}{\sum_{j=1}^{M}\eta_j\phi_j} \qquad \text{for } i = 1, 2, \ldots, M. \tag{14}$$

Note that the posterior probabilities sum to one,

$$\sum_{i=1}^{M} \pi_i = 1. \tag{15}$$

2.1.4 Initialisation and training

The initial values of the weights in the *MLP* can be sampled from a Gaussian distribution with zero mean and an appropriate variance. My experience is that the variance should be relatively large since a small value is more likely to give rise to symmetries that can slow down the convergence, or even cause the network to be trapped in a poor minimum. The output layer biases are however a special case where we can initialise the weights differently and use the data to improve our initial guess.

Initialisation of the output layer biases The initialisation method starts by estimating the *unconditional* probability density for the *target* space with some clustering algorithm (like k-means, see, for example (Bishop, 1995)). By clustering the data into the same number of clusters as we have mixture components we can use the positions of each cluster as the initial values for the second layer biases corresponding to the means. One of the effects of this is that the kernels are separated, which will normally speed up convergence.

Training The training can now be done with any generic non-linear optimisation method, e.g. Scaled Conjugate Gradients or quasi-Newton. For references, consult (Press *et al.*, 1992). But how do we know when to stop? Often a procedure called 'early stopping' (Baldi and Chauvin, 1991) is used. We start by partitioning the data into three independent sets, that we call the training, validation and test sets. The parameter adjustment is done using the training set and periodically

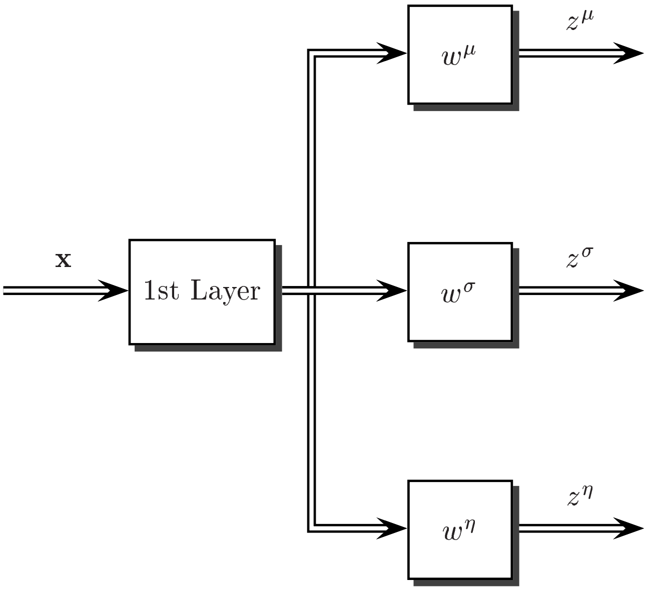


Figure 5: The figure shows how the *MDN* can be seen as three different network sharing a common first layer.

we stop training and measure the error on the independent validation set. In the beginning the validation error normally decreases but as the network starts to over-fit, the validation error starts to increase. The point at which to stop training is therefore when we reach the minimum of the validation error. The generalisation performance of the model can then be evaluated using the third part of the data to compute the test error.

One reason I believe that this procedure is sub-optimal for MDN's is that we implicitly assume that the functions for the mean, variance and mixing coefficients reach their optimum at the same time, and we also assume that they have the same complexity since they all share the first layer, as illustrated by Figure 5. However, it seems very unlikely that we will reach the optimum of the means, variances and mixing coefficients at the same time. The obvious way to reduce the effects of this would be to use *regularisation* (Tikhonov and Arsenin, 1977) and a Bayesian view of this is described in the next section.

2.2 The Evidence Framework

The evidence framework (MacKay, 1992a; MacKay, 1992d; MacKay, 1992c) provides, among other things, a method to deal with the problem of determining model complexity for *RBF* and *MLP* networks, *i.e.* how to choose a model that matches the complexity of the data in an objective Bayesian way. For regression the posterior distribution is modelled as a single Gaussian under the assumption of Gaussian zero mean noise and for classification the outputs are logistics for two class problems and generalised to the softmax transformation for multiple class problems.

For a Bayesian the posterior distribution of the model parameters is the natural starting point and all other quantities are *inferred* from it. The correct approach is to *integrate over all possible weights* for some prior. In the evidence procedure we do this integration (approximately in some cases) assuming the posterior to be distributed as a Gaussian around the *most probable* network weights, $\mathbf{w}_{MP}$. One obvious reason for this is approximation is, efficiency since a full integration of the posterior cannot always be carried out analytically and in that case is very computationally expensive and Monte Carlo techniques have to be considered.

Consider a data set $\mathcal{D} = \{x^n, t^n\}$, where n is an index running over all patterns and x^n and t^n are, for simplicity, scalar input and target vectors respectively. So, by using Bayes' rule we can write the posterior with respect to the network weights $\mathbf{w}$ for some architecture $\mathcal{A}$, as

$$p(\mathbf{w}|\mathcal{D}, \alpha, \beta, \mathcal{A}) = \frac{p(\mathcal{D}|\mathbf{w}, \alpha, \beta, \mathcal{A})p(\mathbf{w}|\alpha, \mathcal{A})}{p(\mathcal{D}|\alpha, \beta, \mathcal{A})} \tag{16}$$

The parameters α and β are going to be explained in the following sections; first we need to interpret the different factors on the right hand side of (16). $p(\mathcal{D}|\mathbf{w}, \alpha, \beta, \mathcal{A})$ is the *likelihood* of the data $\mathcal{D}$, $p(\mathbf{w}|\alpha, \mathcal{A})$ is a *prior distribution* that gives us an opportunity to incorporate prior knowledge about our data into the posterior distribution and finally $p(\mathcal{D}|\alpha, \beta, \mathcal{A})$ is the *evidence* for the model. This term is independent of the network parameters and it can be used to give us an idea of how well a specific model fits the data. This is useful information when dealing with questions concerning model comparison but it is outside the scope of this technical report. From now on the dependence on $\mathcal{A}$ will be taken implicitly in all equations. These interpretations can be summarised by writing the posterior symbolically as

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}.$$

Before we start investigating how we can use the expression for the posterior distribution we have to find a suitable prior distribution. The problem is how to transform any knowledge we have into preferences for specific values of weights in our neural network. What we can do is to express our belief in *different levels of smoothness* for a specific function. A more complex function usually requires larger weights while simpler functions can be represented with smaller weights. This sounds very subjective and the objective solution is to introduce a *hyperparameter* α that controls the prior distribution as follows

$$p(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W), \tag{17}$$

where

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{k/2}, \tag{18}$$

$$E_W = \frac{1}{2}\sum_{i=1}^{k} w_i^2, \tag{19}$$

and k is the number of parameters. This choice of E_W corresponds to a Gaussian prior, but other choices are of course possible. We can then, once again, *infer the most probable value of* α from its posterior distribution. Figure 6 shows the effect that different α values have on fitting a simple sinusoid function using a *MLP*network.

We now move on to show how to find the most probable weights, both for regression (continuous targets) and classification (discrete targets).

2.2.1 Bayesian Regression

Hierarchical Bayesian analysis consists of different levels, and for each level we can infer one or more of the quantities we are interested in. In the first level we infer the posterior of the weights corresponding to the regression parameters and on the second level the value of the regularisation constants can be estimated.

For regression the likelihood of the data set $\mathcal{D}$ is

$$P(\mathcal{D}|\mathbf{w}, \beta) = \frac{\exp(-\beta E_D)}{Z_D(\beta)}, \tag{20}$$

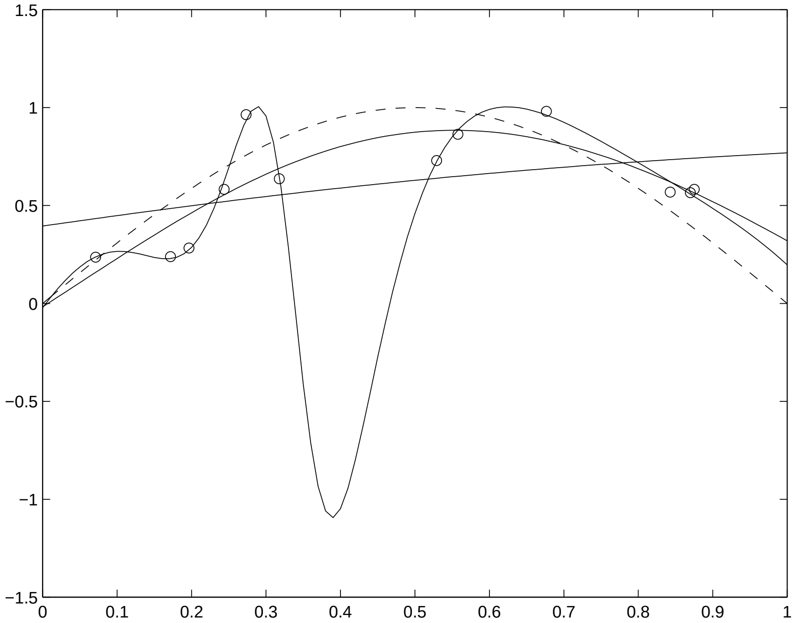


Figure 6: The plot shows the same *MLP* network trained with different values of the weight decay parameter α. The dashed line corresponds to the function that generated the data. For too small α values the networks over-fit and when it is too large they under-fit.

where E_D is normally the sum-of-square error function

$$E_D = \frac{1}{2}\sum_{i=1}^{N}(z_n - t_n)^2. \tag{21}$$

For this E_D the normalisation constant for (20) is

$$Z_D(\beta) = \int \exp(-\beta E_D)dt = \left(\frac{2\pi}{\beta}\right)^{N/2}, \tag{22}$$

and $1/\beta$ is the *estimated average noise* in the targets. This choice of E_D corresponds to the assumption of additive zero mean Gaussian noise in t with variance $\sigma^2 = 1/\beta$.

First level of Inference We are now ready to find an expression for the posterior distribution of the weights $\mathbf{w}$, which we will use to find the most probable parameters $\mathbf{w}_{MP}$. Substituting the expression for the likelihood (20) and the prior (17) into the expression for the posterior (16) gives

$$P(\mathbf{w}|\mathcal{D},\alpha,\beta) = \frac{\exp(-\alpha E_W - \beta E_D)}{Z_M(\alpha,\beta)}, \tag{23}$$

where

$$Z_M(\alpha,\beta) = \int \exp(-\alpha E_W - \beta E_D)d\mathbf{w} \tag{24}$$

is the evidence term mentioned earlier. Maximisation of (23) is equivalent to minimising $\alpha E_W + \beta E_D$ because Z_M is a constant with respect to $\mathbf{w}$. This gives us

$$M(\mathbf{w}) = \alpha E_W + \beta E_D \tag{25}$$

Minimising the *misfit function* $M(\mathbf{w})$ yields the most probable network weights $\mathbf{w}_{MP}$and it corresponds to finding the most probable interpolant. Minimising E_D alone leads us back to the maximum likelihood estimate with weights $\mathbf{w}_{ML}$.

Second level of Inference After we have inferred the most probable values for the weights we want to infer the most probable value of the hyperparameters α and β. To do this we need to calculate the joint posterior distribution with respect to these hyperparameters, which we have earlier implicitly assumed to be independent. This posterior is expressed, with Bayes' rule, as

$$P(\mathcal{D}|\alpha,\beta) = \frac{p(\mathcal{D},\alpha,\beta)P(\alpha,\beta)}{P(\mathcal{D})}. \tag{26}$$

Here we, once again, have to determine a prior distribution and this time it is for the hyperparameters. We choose to use a *flat* prior because this corresponds to our lack of knowledge about their true values. Under these assumptions the posterior (26) can be written in terms of normalisation constants, already given by (24), (18) and (22), as follows

$$P(\mathcal{D}|\alpha,\beta) = \frac{Z_M(\alpha,\beta)}{Z_W(\alpha)Z_D(\beta)}. \tag{27}$$

The only integral left to solve is $Z_M(\alpha,\beta)$, and to do this we use the second order Taylor expansion of $M(\mathbf{w})$ which is

$$M(\mathbf{w}) = M(\mathbf{w}_{MP}) + (\mathbf{w}-\mathbf{w}_{MP})^T\mathbf{A}(\mathbf{w}-\mathbf{w}_{MP}), \tag{28}$$

where $\mathbf{A}$ is the Hessian of $M(\mathbf{w})$ calculated at $\mathbf{w}_{MP}$. This makes (24) a Gaussian integral with the following solution

$$Z_M(\alpha,\beta) = (2\pi)^{k/2}|\mathbf{A}|^{-1/2}\exp\bigl(M(\mathbf{w}_{MP})\bigr) \tag{29}$$

If E_D and E_W are both quadratic this is an exact result, otherwise it is an approximation, but even so the Gaussian approximation is serviceable in practise.

After solving the integrals the negative log of the posterior (26) is written as

$$-\ln P(\mathcal{D}|\alpha,\beta) = -\alpha E_W^{MP} - \beta E_D^{MP} - \frac{1}{2}\ln|\mathbf{A}| + \frac{k}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi \tag{30}$$

We can now find the optimum values of the hyperparameters by setting the the partial derivatives of (30) equal to zero and solving the resultant equations. (For non-quadratic $E_D(E_W)$ this is once again an approximation). This gives

$$\alpha = \frac{\gamma}{2E_W^{MP}} \tag{31}$$

$$\beta = \frac{N-\gamma}{2E_D^{MP}}, \tag{32}$$

where γ is the *effective number of parameters*; it measures how much structure from the data is incorporated into the network parameters or, to rephrase it, how many parameters are well determined by the data. Let λ_a be the k eigenvalues of M, calculated in the natural basis for the prior, *i.e.* the basis where the Hessian for the prior is the identity matrix, and $E_W = \frac{1}{2}\sum_{i\in\mathbf{w}} w_i^2$. γ is defined as

$$\gamma = \sum_{a=1}^{k} \frac{\lambda_a}{\lambda_a+\alpha}, \tag{33}$$

where each term in the sum is a number between 0 and 1; thus γ ranges from 0 to k.

This means that evaluating γ requires the calculation of the Hessian of M. If this is not possible or too computationally expensive some numerical approximation can be used, for example the following approximate re-estimators

$$\alpha_c = \frac{k_c}{2E_W^c} \tag{34}$$

$$\beta = \frac{N}{2E_D}, \tag{35}$$

where k_c is the number of parameters in regularisation class c. We interpret this as no longer distinguishing between well and poorly determined parameters. The advantage with these expressions are that since they are very cheap to compute it is affordable to update the regularisation frequently whereas in the case where we calculate the effective number of parameters we have to find the balance between updating frequency and the computational cost.

Regularisation Classes MacKay has shown that for some problems it can be beneficial to introduce different regularisation classes, which means that we partition the weights into different disjoint groups (it does not, however, need to be exhaustive, *i.e.* some weights, like biases, may not belong to any regularisation class). A common grouping is to have different regularisation groups for first and second layer weights and distinguish between weights and biases. The motivation behind this is that these groups can have weights of different magnitudes due to, for example, scaling differences between input vectors and target vectors. The misfit function (25) is then modified to be

$$M(\mathbf{w}) = \sum_{c=1}^{C} \alpha_c E_{W_c} + \beta E_D \qquad \text{where } E_{W_c} = \frac{1}{2}\sum_{i \in \mathbf{w}_c} w_i^2, \tag{36}$$

and the re-estimation formulae for the hyperparameters change into

$$\alpha_c = \frac{\gamma_c}{2E_{W_c}^{MP}} \tag{37}$$

$$\beta = \frac{N - \sum_{c=1}^{C} \gamma_c}{2E_D^{MP}}, \tag{38}$$

where C is the number of regularisation classes and γ_c is the sum of the eigenvalues corresponding to regularisation class c.

2.2.2 Applying the Evidence Procedure to Classification

Consider a data set $\mathcal{D} = \{x^n, \mathbf{t}^n\}$ where x is the input vector as in the previous section and $\mathbf{t}^n$ is a vector with 1-out-of-M coding with a 1 in column corresponding to the correct class and zeros for all other columns. For classification the objective function is no longer the sum-of-squares but some information based measure like the cross-entropy (Bishop, 1995). For the two class case this corresponds to a logistic distribution. In the context of *MDN*'s, a mixture of two kernels is, however, hardly very useful, so we focus on the case with more than two classes, which uses the generalised logistic, the softmax ((Bridle, 1990)).

The cross-entropy for M classes is defined as

$$G(\mathcal{D}|\mathbf{w}) = -\sum_{n=1}^{N}\sum_{i=1}^{M} t_i^n \ln \eta_i(\mathbf{x}^n; \mathbf{w}), \tag{39}$$

where $\eta_i(\mathbf{x}^n; \mathbf{w})$ is the softmax transformation of the network outputs z_i as follows

$$\eta(\mathbf{x}^n; \mathbf{w}) = \frac{\exp(z_i(\mathbf{x}^n; \mathbf{w}))}{\sum_{j=1}^{M} \exp(z_j(\mathbf{x}^n; \mathbf{w}))}. \tag{40}$$

The misfit function from (25) transforms into

$$M(\mathbf{w}) = -G + \alpha E_W. \tag{41}$$

Here we can see that the term βE_D has been replaced by $-G$. This is valid when the targets are organised in a 1-out-of-M coding.

We can now use (41) to find the most probable weights analogously to the regression case but note that the expression for the Hessian also changes since it depends on $M(\mathbf{w})$. The regularisation constant α can be evaluated as before using the new misfit function and (31).

For regression, the evidence procedure, is exact for generalised linear regression (*GLR*), for example a *RBF* network with fixed centres and linear outputs, because for these models M is quadratic. In in the classification case it is an approximation because G is non-quadratic and this will slow down the convergence since we are approximating a non-quadratic error function with a quadratic one. MacKay motivates the validity of this approximation with the central limit theorem; 'We expect the posterior to converge to a set of locally Gaussian peaks with increasing quantities of data.'

2.2.3 Training the network

The values of the weights are determined by non-linear optimisation of M. To do this we need an initial estimate of α. The evidence framework does not give us a method to systematically determine the initial value. If similar networks have been trained on the same data their final α can be used to get an initial guess of the right magnitude. The initial choice is not critical to the convergence of the algorithm. After a few iterations of the optimisation algorithm we stop to re-estimate the regularisation parameter. In theory the re-estimation formula is only valid at a local minimum of the cost function but it is still serviceable away from the minimum and as the optimisation of M proceeds the quality of the α update will increase until we reach a minimum. MacKay re-estimated α and β every other iteration. The efficency of the quasi-Newton optimiser relies on its estimate of the inverse Hessian. The optimiser starts with the identity matrix and for every iteration the approximation is refined with an update rule. For frequent updates of the regularisation parameters it is important to maintain the estimated Hessian in the optimiser otherwise the convergence rate will decrease to a similar rate of the gradient descent. In the same way care has to be taken with other quadratic optimisation routines in order to not decrease their convergence rate.

2.3 A Bayesian Approach to Input Dependent Noise

The evidence framework described in Section 2.2 assumes that the noise on the targets is constant. It is easy to invent situations where this assumption is not true. (MacKay, 1992b, Section 6) gives an outline for a solution to this problem and (Qazaz, 1996; Bishop and Qazaz, 1997) have investigated this for the case of a single Gaussian kernel at the output using generalised linear models (*RBF* networks with fixed centres in the hidden layer) and showed that this approach reduces the bias in the estimated variance compared with a traditional *ML* approach. The rest of this section is a review of their work.

2.3.1 The Model

If we want to be able to model input dependent noise we need to let β, the inverse variance, be a function of the input vector. One way to achieve this is by using another network to estimate β. Using this idea we have two networks, one that models the mean and one that models the inverse variance. The networks are chosen to be *RBF* networks with fixed basis functions because this simplifies the analysis but they could in principle be *MLP* networks (but that requires further approximations in the following calculations).

The likelihood for the data set can be written as

$$p(\mathcal{D}|\mathbf{u}, \mathbf{w}) = \frac{1}{Z_D} \exp\big(-\sum_{n=1}^{N} \beta(\mathbf{x}^n; \mathbf{u}) E^n\big), \tag{42}$$

where Z_D is a normalisation constant

$$Z_D = \prod_{n=1}^{N} \Big(\frac{2\pi}{\beta^n}\Big)^{1/2}, \tag{43}$$

E_n is the sum-of-squares error

$$E^n = \frac{1}{2}\big(y(\mathbf{x}^n, \mathbf{w}) - t^n\big)^2, \tag{44}$$

and the two networks are defined to be

$$y(\mathbf{x}^n, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}^n) \tag{45}$$

$$\beta(\mathbf{x}^n, \mathbf{u}) = \exp\big(\mathbf{u}^T \psi(\mathbf{x}^n)\big), \tag{46}$$

where $\mathbf{u}$ and $\mathbf{w}$ are the weights for the respective networks. The design matrices $\phi(\mathbf{x}^n)$ and $\psi(\mathbf{x}^n)$ are the output of the first (non-linear) layer of the *RBF* networks where one of the basis functions is a constant, $\phi_0 = \psi_0 = 1$, so the corresponding weights represent bias parameters. Note that the inverse variance β is modelled as an exponential to prevent it from being negative. This will also discourage β from going to zero, which would cause the likelihood to approach infinity.

2.3.2 Hierarchical Bayesian analysis

This is the same approach as in section 2.2.1 but the generalisation that allows β to be a function of the inputs increases the hierarchy with one extra layer.

First Level On this first level of inference we want to infer the most probable weights, $\mathbf{w}_{MP}$. We can do this by finding an expression for the posterior of $\mathbf{w}$ and maximising that. At this stage we temporarily assume that the value of all other relevant parameters, $\mathbf{u}_{MP}$ and α_w are known. The posterior of $\mathbf{w}$ can then be written as

$$p(\mathbf{w}|\mathcal{D}, \mathbf{u}_{MP}, \alpha_w) = \frac{p(\mathcal{D}|\mathbf{w}, \mathbf{u}_{MP})p(\mathbf{w}|\alpha_w)}{p(\mathcal{D}|\mathbf{u}_{MP}, \alpha_w)} \tag{47}$$

by using Bayes' rule. The denominator of (47) is insignificant at this stage of inference and can be omitted since it is independent of $\mathbf{w}$. Before we can do any inference we need to define a prior over $\mathbf{w}$ and as in Section 2.2, we choose a Gaussian distribution $p(\mathbf{w}|\alpha_w)$

$$p(\mathbf{w}|\alpha_w) = \left(\frac{\alpha_w}{2\pi}\right)^{k_w/2} \exp(-\frac{\alpha_w}{2}||\mathbf{w}||^2), \tag{48}$$

where k_w is the length of the parameter vector $\mathbf{w}$. Substituting (42) and (48) in (47) yields

$$p(\mathbf{w}|\mathcal{D}, \mathbf{u}_{MP}, \alpha_w) = \frac{1}{Z_D(\mathbf{u}_{MP})} \exp\Big(-\sum_{n=1}^{N} \beta^n E^n\Big) \frac{1}{Z_\alpha(\alpha_w)} \exp\Big(-\alpha_w E_w(\mathbf{w})\Big), \tag{49}$$

where all quantities have been defined in Section 2.3.1. Taking the negative log of (49) and omitting constant terms with respect to $\mathbf{w}$ gives

$$M(\mathbf{w}) = \sum_{n=1}^{N} \beta^n E^n + \frac{\alpha_w}{2}||\mathbf{w}||^2. \tag{50}$$

The most probable weights, $\mathbf{w}_{MP}$ can now be found by maximising $M(\mathbf{w})$ for a fixed $\mathbf{u}$ and α_w. This can be done by solving a set of linear equations.

Second Level After finding the most probable weights for the regression network we can now move on to infer the most probable weights for the network estimating the inverse variance. The correct Bayesian approach is then to *marginalise* over all possible regression weights. This makes our estimate of the inverse variance independent of the mean. If we do not use marginalisation the mean will inevitably fit some of the noise, since it is indistinguishable from the true data which would cause the variance to be underestimated.

The posterior with respect to the inverse variance is written, using Bayes' rule, as

$$p(\mathbf{u}|\mathcal{D}, \alpha_w, \alpha_u) = \frac{p(\mathcal{D}|\mathbf{u}, \alpha_w)p(\mathbf{u}|\alpha_u)}{p(\mathcal{D}|\alpha_w, \alpha_u)}. \tag{51}$$

Once again the denominator of the posterior can be discarded due to its independence of $\mathbf{u}$. The prior $p(\mathbf{u}|\alpha_u)$ is chosen to be Gaussian in the same way as for the prior over $\mathbf{w}$,

$$p(\mathbf{u}|\alpha_u) = \left(\frac{\alpha_u}{2\pi}\right)^{k_u/2} \exp(-\frac{\alpha_u}{2}||\mathbf{u}||^2) \tag{52}$$

where k_u denotes the length of the vector $\mathbf{u}$.

The factor $p(\mathcal{D}|\mathbf{u}_{MP}, \alpha_w)$, from (51) can be recognised as the denominator from (47) and it can be written as

$$p(\mathcal{D}|\mathbf{u}, \alpha_w) = \int p(\mathcal{D}|\mathbf{w}, \mathbf{u}) p(\mathbf{w}|\alpha_w) d\mathbf{w}. \tag{53}$$

The integral in (53) can be solved analytically for *GLR* and after taking the negative log and discarding constants with respect to $\mathbf{u}$ we get

$$S(\mathbf{u}) = \sum_{n=1}^{N} \beta^n E^n + \frac{\alpha_u}{2} ||\mathbf{u}||^2 - \frac{1}{2} \sum_{n=1}^{N} \ln \beta^n + \frac{1}{2} \ln |\mathbf{A}|, \tag{54}$$

where $\mathbf{A}$ is the Hessian of $S(\mathbf{u})$ which can be written as

$$\mathbf{A} = \sum_{n=1}^{N} \beta^n \Phi(\mathbf{x}^n) \Phi(\mathbf{x}^n)^T + \alpha_w \mathbf{I}. \tag{55}$$

The term $\ln |\mathbf{A}|$ is interesting since it is the only term that differs from what we would get with a penalised *ML* approach and it originates from the marginalisation over $\mathbf{w}$.

To find $\mathbf{u}_{MP}$ we cannot use linear techniques in the same way as we solved $\mathbf{w}_{MP}$ because $S(\mathbf{u})$ is non-linear in $\mathbf{u}$. For this case (slower) non-linear optimisation algorithms have to be employed.

Third Level Thus far the most probable values for $\mathbf{u}$ and $\mathbf{w}$ have been inferred and the remaining task is to find the most probable values for the regularisation constants, α_w and α_u.

The posterior for the alphas is

$$p(\alpha_u, \alpha_w|\mathcal{D}) = \frac{p(\mathcal{D}|\alpha_u, \alpha_w) p(\alpha_u, \alpha_w)}{p(\mathcal{D})}. \tag{56}$$

The denominator can be discarded since it is independent of the regularisation constants. The regularisation constants are assumed to be independent so $p(\alpha_u, \alpha_w) = p(\alpha_u)p(\alpha_w)$ and if we chose a flat prior over these, which corresponds to our lack of knowledge about their correct values, (56) reduces to

$$p(\alpha_u, \alpha_w|\mathcal{D}) = p(\mathcal{D}|\alpha_u, \alpha_w) = \int p(\mathcal{D}, \mathbf{u}|\alpha_u, \alpha_w) d\mathbf{u} = \int p(\mathcal{D}|\mathbf{u}, \alpha_w) p(\mathbf{u}|\alpha_u) d\mathbf{u} \tag{57}$$

The term $p(\mathcal{D}, |\alpha_w)$ can be recognised from the second level of inference as (53). We now continue to marginalise over the $\mathbf{u}$ but this integral is not tractable. However by Taylor expanding a Gaussian approximation of $S(\mathbf{u})$ and some algebraic manipulation we get

$$p(\mathcal{D}|\alpha_u, \alpha_w) \propto (\alpha_w)^{k_w/2} (\alpha_u)^{k_u/2} |\mathbf{H}|^{-1/2} \exp\Big(-\alpha_w E_W\Big) \exp\Big(-M(\mathbf{u}_{MP})\Big) \tag{58}$$

where $|\mathbf{H}|$ is the determinant of the Hessian with respect to $\mathbf{u}$ at $\mathbf{u}_{MP}$, which is written as

$$\mathbf{H} = \frac{\partial^2 S(\mathbf{u})}{\partial \mathbf{u}^2} \Big|_{\mathbf{u}_{MP}} \tag{59}$$

We can then form a misfit function, $W(\alpha_u, \alpha_w)$, by taking the negative log of the posterior approximation in (58) as follows

$$\begin{aligned} W(\alpha_w, \alpha_u) &= -\ln p(\mathcal{D}|\alpha_u, \alpha_w) \\ &= \alpha_w E_W + \alpha_u E_U - \frac{k_w}{2} \ln \alpha_w - \frac{k_u}{2} \ln \alpha_u + \frac{1}{2} \ln |\mathbf{A}| + \frac{1}{2} \ln |\mathbf{H}| \end{aligned} \tag{60}$$

At the minimum of W the most probable hyperparameters satisfy

$$\alpha_{wMP} = \frac{\gamma_w}{2E_W} \tag{61}$$

$$\alpha_{uMP} = \frac{\gamma_u}{2E_U} \tag{62}$$

where γ_w and γ_u are the effective number of parameters as explained in Section 2.2.1. And it is also possible to use the cheaper approximative re-estimation formulaes from (31) and (32).

Training the Model The training is divided into three different stages that coincide with the levels of inference described earlier, as follows:

1. Optimise $\mathbf{u}$ with non-linear techniques for a given $\mathbf{w}$.
2. Optimise $\mathbf{w}$ with linear techniques for a given $\mathbf{u}$.
3. Re-estimate the regularisation constants

We repeat these steps until convergence is reached. Note that $\mathbf{u}$ is optimised for several iterations with non-linear algorithms while the optimal $\mathbf{w}$ is calculated by solving a set of linear equations, for a given $\mathbf{u}$, which is computationally cheaper. At every new step the last value of the parameters are used as initialisation.

2.4 Modelling the Complete Covariance Matrix

Another interesting approach to modelling both the conditional mean and variance has been carried out for Gaussians. Williams (1996) has investigated a model where the full covariance matrix is modelled and demonstrated good results with this method on both synthetic data and real life financial time series.

The density function for a c dimensional Gaussian with covariance matrix Σ and mean, μ can be written as

$$p(\mathbf{t}|\mathbf{x}) = \frac{1}{(2\pi)^{c/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{t} - \mu)^T \Sigma^{-1} (\mathbf{t} - \mu) \right). \tag{63}$$

We now want to use a neural network to generate the parameters of a Gaussian that models the target vector, when we propagate the input vector through the neural network. This is the same idea as for the *MDN* in Section 2.1. The only extension we need is to include the non-diagonal elements in the covariance matrix.

The covariance matrix is always a symmetric positive definite matrix. This means that we can use the Cholesky decomposition to find a triangular matrix that satisfies

$$\Sigma^{-1} = \mathbf{S}^T \mathbf{S}. \tag{64}$$

It is then possible to model the elements of $\mathbf{S}$ as follows

$$s_{ii} = \exp(z_{ii}) \qquad i = 1, 2, \ldots, c \tag{65}$$

$$s_{ij} = z_{ij} \qquad i = 1, 2, \ldots, c-1 \quad j = 2, 3, \ldots, c, \quad i < j. \tag{66}$$

The mean is simply modelled as

$$\mu_i = z_i^{\mu} \qquad i = 1, 2, \ldots, c. \tag{67}$$

For the normalisation term we need to evaluate the square root of the determinant of the covariance matrix and this becomes very simple since $\mathbf{S}$ is a triangular matrix. The square root of the determinant of the covariance matrix can be written as

$$|(\Sigma)|^{-1/2} = \left(|\mathbf{S}^T||\mathbf{S}|\right)^{1/2} = \left(\left(\sum_{i=1}^{c} s_{ii}\right)\left(\sum_{i=1}^{c} s_{ii}\right)\right)^{1/2} = \sum_{i=1}^{c} s_{ii}. \tag{68}$$

2.4.1 The Likelihood

The negative log likelihood for this model can be written as

$$E = \sum_{n=1}^{N} E_n \tag{69}$$

where the error for each pattern is

$$E_n \propto \frac{1}{2}\log|\Sigma_n| + \frac{1}{2}(\mathbf{t}_n - \mu_n)^T|\Sigma|^{-1}(\mathbf{t}_n - \mu_n) \tag{70}$$

under the assumption that the observations are jointly independent. In order to obtain a ML estimate we optimise the weights to minimise the error and for more efficient optimisation the gradient of the error function can be used. To calculate the gradient we first introduce two new definitions to simplify the calculations

$$\delta_i = \mu_i - t_i \qquad i = 1, 2, \ldots, c \tag{71}$$

$$\xi_i = \sum_{j=i}^{c} \sigma_{ij}\delta_{ij} \qquad i = 1, 2, \ldots, c. \tag{72}$$

The error function for one pattern can then be written as

$$E = \sum_{i=1}^{c}\left(\frac{1}{2}\xi_i^2 - z_{ii}^{\sigma}\right) \tag{73}$$

where we have omitted the indices on the patterns. The gradient of (73) can the be written as:

$$\frac{\partial E}{\partial y_i^{\mu}} = \sum_{j=1}^{i} \xi_j s_{ji} \qquad i = 1, 2, \ldots, c, \tag{74}$$

$$\frac{\partial E}{\partial y_i^{\sigma}} = \xi_i\delta_i s_{ii} - 1 \qquad i = 1, 2, \ldots, c, \tag{75}$$

$$\frac{\partial E}{\partial y_{ij}^{\sigma}} = \xi_i\delta_j \qquad i = 1, 2, \ldots, c-1, \quad j = 2, 3, \ldots, c, \quad i < j. \tag{76}$$

2.4.2 Training and Regularisation

The model can then be trained with ordinary non-linear optimisation, with or without regularisation. The regularisation can be achieved by adding a regularisation term (penalised likelihood) or by pruning (Williams, 1994), *i.e.* where we start training with a large network and during the learning process we remove hidden units with some objective criteria until we have matched model complexity with the complexity of the data.

2.5 Training of *RBF* networks for classification

In Nabney (1998), an effective method for training RBF networks in classification tasks is introduced. Here the position of the centres in the *RBF* network are fixed and the model is effectively a *generalised linear model.* For the theory of *GLM* see McCullagh and Nelder (1983). If the *RBF* networks have *linear outputs* the likelihood $\mathcal{L}$, is a quadratic form with respect to the weights $\mathbf{w}$,

$$\mathcal{L} = (\mathbf{T} - \Phi\mathbf{w})^T(\mathbf{T} - \Phi\mathbf{w}), \tag{77}$$

where $\mathbf{T}$ denotes the target matrix and Φ is the design matrix. Equating the derivative of (77) to zero gives the normal equations

$$(\Phi^T\Phi)\mathbf{w} = \Phi^T\mathbf{T} \tag{78}$$

which we solve by computing the pseudo-inverse $\Phi^\dagger$ of Φ and setting $\mathbf{w} = \Phi^\dagger\mathbf{T}$. This is numerically more stable than computing explicitly the inverse of the square matrix $\Phi^T\Phi$.

The drawback with linear outputs is that the outputs cannot be interpreted as probabilities. If we want this interpretation we once more need to use the logistic and softmax functions on the outputs but this will lead to a non-quadratic likelihood function and we have to resort to iterative methods for the solution. The Fisher scoring method updates the parameter estimates $\mathbf{w}$ at the r th step by

$$\mathbf{w}_{r+1} = \mathbf{w}_r - \{\mathrm{E}[\mathbf{H}]\}^{-1}\frac{\partial\mathcal{L}}{\partial\mathbf{w}} \tag{79}$$

where the Hessian is $\mathbf{H} = \partial^2\mathcal{L}/\partial\mathbf{w}\partial\mathbf{w}^T$. In our case, the expected value of the Hessian coincides with the Hessian and this update rule is equivalent to the Newton-Raphson algorithm. This is normally not a good idea since it is easy to over-shoot the maximum, but for the logistic model there are two special features that make it work well in practise. First $\mathcal{L}$ has a single maximum and second it is possible to initialise $\mathbf{w}$ reasonably close to that maximum.

For the logistic model the Hessian is equal to $-\Phi^T\mathbf{D}\Phi$, where $\mathbf{D}$ is a diagonal *weight* matrix whose elements are $\pi^n(1-\pi^n)$. The gradient is equal to $\Phi^T\mathbf{D}\mathbf{e}$ where the nth row of $\mathbf{e}$ is given by

$$\mathbf{e}^n = (\mathbf{t}^n - \pi^n)/f' \tag{80}$$

where f' is the derivative of the link function, which is $\pi^n(1-\pi^n)$ for the logistic model. We can now write the linearisation of the link function around the mean to be $\mathbf{z}_r = \Phi^T\mathbf{w}_r + \mathbf{e}$ and the update formula (79) changes into

$$(\Phi^T\mathbf{D}\Phi)\mathbf{w}_{r+1} = \Phi^T\mathbf{D}_r\mathbf{z}_r \tag{81}$$

which is the normal format equation for an input matrix $\Phi^T\mathbf{D}_r^{1/2}$ and dependent variable $\mathbf{D}_r^{1/2}\mathbf{z}_r$. The weights $\mathbf{D}$ change each iteration, since they are a function of $\mathbf{w}$. The algorithm is known as *iteratively re-weighted least squares* (*IRLS*).

For the initial step we set $\pi^n = (\mathbf{t}^n + 0.5)/2$ as a first estimate of π^n and this enables us to derive the other quantities required.

For the multiple class case things are a bit more complicated. If we assume independence of the outputs we achieve the same update rule for the logistic case, but this is a poor approximation of the Hessian. It is however good enough for the initialisation step, with the small modification that the initial values of the π^n should now be $\pi^n = (\mathbf{t}^n + \psi)/2$ where ψ is a constant vector of the same dimension as π^n with all elements equal to $1/M$ where M is the number of classes and for this case the gradient and the Hessian for a single pattern are given by

$$\frac{\partial\mathcal{L}}{\partial w_{ik}} = (\pi_i - t_k^n)\phi_k^n \tag{82}$$

$$\frac{\partial^2\mathcal{L}}{\partial w_{jl}\partial w_{ik}} = (\pi^n\delta_{ij} - \pi_i^n\pi_j^n)\phi_k^n\phi_l^n. \tag{83}$$

Without the independence assumption the Hessian normally becomes very ill-conditioned but using singular value decomposition it is still possible to solve the Fisher score equations. The expression for the gradient and Hessian can be found in Nabney (1998), together with more details on the softmax case.

3 Regularisation of MDNs with Fixed Kernels

In this section I present a procedure for regularising Mixture Density Networks under the constraint that the mean and variance of the spherical Gaussian kernels are fixed. Under this assumption I show how to derive an expression for the error function and how the regularisation hyperparameters can be estimated following the evidence framework. I then give guidelines for setting sensible values for the mean and variance of each kernel in the Gaussian Mixture, which is required if the model is to perform well. Finally I describe how to modify the results for the case when a RBF network generates the mixture coefficients.

3.1 Modifying the Mixture Density Network Model.

Let us assume that the output mixture model consists of M normalised Gaussians, $\{\phi_1, \phi_2, \ldots, \phi_M\}$ for which the mean and variance are known and fixed. This assumption simplifies the optimisation task compared with the *MDN* from section 2.1 where these parameters were also found by optimisation at the same time as the mixing coefficients.

The likelihood, with respect to the network weights $\mathbf{w}$, is now

$$p(\mathcal{D}|\mathbf{w}) = G = \prod_{n=1}^{N} \sum_{i=1}^{M} \eta_i(\mathbf{x}^n; \mathbf{w}) \phi_i(\mathbf{t}^n) \tag{84}$$

where the mixing coefficients $\eta_i(\mathbf{x}^n; \mathbf{w})$ have been normalised with the 'softmax' transformation (Bridle, 1990) as in (8).

The difference between (84) and the corresponding likelihood for the standard *MDN* is that the kernels $\phi_i(\mathbf{t}^n)$ are no longer dependent on the input vector or the network parameters $\mathbf{w}$.

Our task is now to *infer* the most probable network parameters, $\mathbf{w}_{MP}$, by using the evidence procedure from Section 2.2 with the new objective function G from (84).

3.1.1 The New Misfit Function

If we assume a Gaussian prior on the weights as in (17), with α as the hyperparameter controlling the distribution, and use (84) as the likelihood function the misfit function from (41) becomes

$$M(\mathbf{w}) = \alpha_w E_W - \ln G = \alpha E_W(\mathbf{w}) - \sum_{n=1}^{N} \ln\left(\sum_{i=1}^{M} \eta_i^n \phi_i^n \right) \tag{85}$$

Minimising (85) will give us the locally most probable weights $\mathbf{w}_{MP}$.

The update formula for the hyperparameter α is still given by (31) where γ is calculated from the Hessian of the modified misfit function.

3.1.2 Initialisation of the Fixed Parameters

As mentioned earlier, the *MDN* optimises all parameters in the Mixture Model but when we fix the mean and variance of each kernel we need to determine them in the initialisation step. If we do not choose these values sensibly the model will not be able to adjust the mixing coefficients in order to model the data well. We need to find guidelines on how to set these values in order to ensure good general approximation abilities.

In theory we would like to use an infinite number of kernels and truly have a mixture that could model any density to arbitrary precision. The number of kernels needed grows exponentially with the dimension of the target space and this is an example of the 'curse of dimensionality'. In practice we can only afford to use a small number of kernels, even with state-of-the-art computers since the computational task becomes very expensive. In the following section a simple and general way of initialising the mean and variance is presented.

The Means Here my suggestion is to place the kernels equidistantly in the range of the targets. The only prior knowledge needed is in what range the targets reside. Computationally, this is a very cheap solution.

One alternative would be to use some clustering algorithm, as we used to initialise the second layer bias weights for the standard *MDN* in Section 2.1.4. This will distribute the kernels according to the density of the targets. It is however not clear that we can always represent multiple branches in the whole of the target space since the number of kernels are finite, and in practice only relatively few kernels can be used before the model gets too computationally demanding. The real drawback is that we have to set the variance of each kernel independently since the distance between kernels varies. I believe that if this kind of advanced initialisation is needed it is probably better to use a *MDN* with *movable kernels* and optimise the mean and variance for each kernel.

The Variances The problem of setting the variance can best be described with a Swedish word, 'lagom', which means not too much nor too little, but just the right amount. Strangely enough many languages do not have this word but following sections will describe a procedure to set the variance of the kernels to 'lagom' values.

If the variance is set to too small a value we will get very sharp peaks at the kernel means but if we attempt to move the peak to an arbitrary point between two kernels by changing the mixing coefficients it goes wrong. Instead of getting one single peak in the middle we get two peaks, one at each kernel's mean. The variance has to be set large enough to generate one single peak anywhere in the interval between the kernels and this value serves as a lower bound on the choice of sigma.

On the other hand; what happens for too large a variance value? This is not as serious as too small a value since it will always generate a single peak. What can happen is that peak becomes flatter, which will decrease the likelihood and this gives a (soft) upper limit on the variance.

Between the lower and upper limit there ought to be an optimum choice. The criterion for this optimum is found by maximising the curvature for the point between the kernels with the lowest curvature. Figure 3.1.2 illustrates this.

To calculate the optimum variance we introduce $f(\eta, x, \mu_1, \mu_2, \sigma)$ to be a mixture of two Gaussians with the same variance as follows

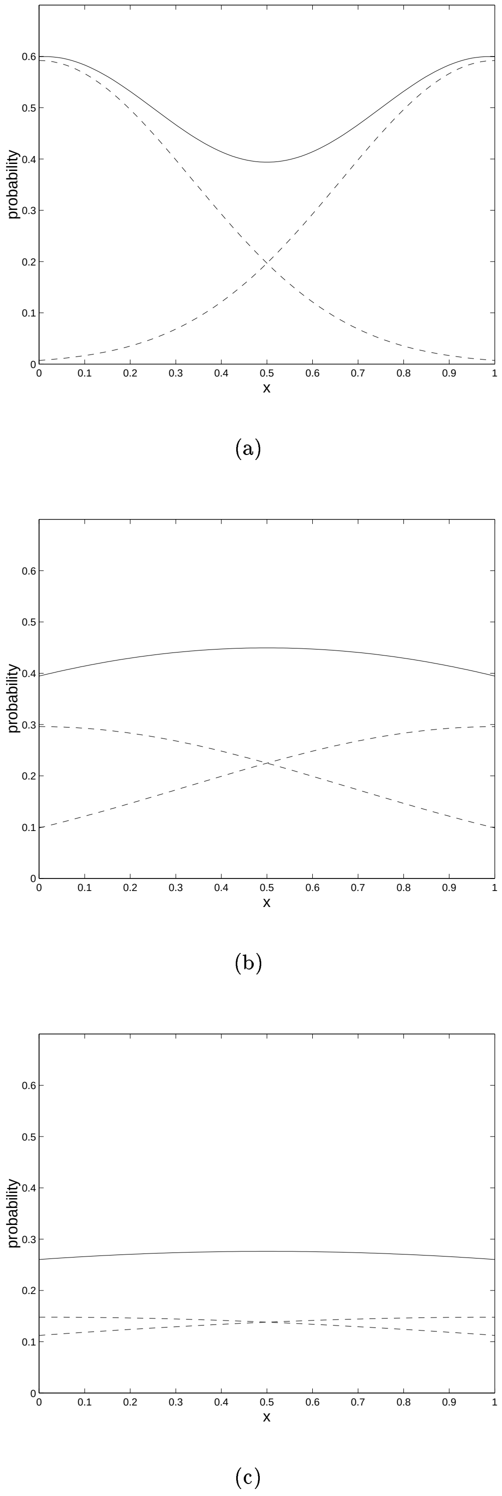
$$f(x; \eta, \mu_1, \mu_2, \sigma) = \eta \frac{1}{(2\pi)^{1/2}\sigma} \exp\left(-\frac{||x-\mu_1||^2}{2\sigma^2}\right) + (1-\eta)\frac{1}{(2\pi)^{1/2}\sigma} \exp\left(-\frac{||x-\mu_2||^2}{2\sigma^2}\right) \qquad 0 \le \eta \le 1 \tag{86}$$

where η is the mixing coefficient. In these calculations $\mu_1 = 0$ and $\mu_2 = 1$ will be used but the calculations can be carried out for arbitrary kernel means.

Differentiating f two times with respect to x gives the curvature of f which we denote

$$h = \frac{\partial^2 f}{\partial x^2} = \frac{1}{2\sqrt{2\pi}}\Big(-\frac{\exp\big(-\frac{x^2}{2\sigma^2}\big)}{\sigma^3} + \frac{x^2\exp\big(-\frac{x^2}{2\sigma^2}\big)}{\sigma^5} - \frac{\exp\big(-\frac{(x-1)^2}{2\sigma^2}\big)}{\sigma^3} + \frac{(x-1)^2\exp\big(-\frac{(x-1)}{2\sigma^2}\big)}{\sigma^5}\Big) \tag{87}$$

The point with the lowest curvature is the point exactly in the middle between the kernels and that means that to calculate the optimal variance we can set $x = (\mu_1 + \mu_2)/2 = 0.5$ and $\eta = 0.5$.



(a)

(b)

(c)

Figure 7: The figure illustrates the effect different choices of the variance σ^2 have on a mixture of two Gaussians. In (a) the variance is half of the optimum. In (b) the variance is the optimum value, $\sigma = 0.674$ and in (c) twice the optimal value.

By using these values for η and x and differentiating (87) with respect to the variance we get

$$\frac{\partial h}{\partial \sigma} = \frac{1}{32}\exp\bigl(-\frac{1}{8\sigma^2}\bigr)\bigl(48\sigma^4 - 24\sigma^2 + 1\bigr) \tag{88}$$

The optimal variance can now be calculated by solving (88). This gives two solutions where the first, $\sigma^2 = (3 - \sqrt{6})/12$ can be discarded because it is below the lower limit of the variance. The second, $\sigma^2 = (3 + \sqrt{6})/12$, is the optimal solution.

To conclude, for a mixture of two kernels the maximum curvature for the point with the minimum curvature is achieved with a variance of $\sigma^2 = ((\mu_1 + \mu_2)/2)^2(3 + \sqrt{6})/12$ and this is the only maximum where the constraint that the two mixing coefficients sum to unity holds and we have a single peak solution.

The result is adequate for more than two kernels because we can always choose to set all but two mixing coefficients to zero which will lead us back to the special case but the analysis cannot be carried out in full detail for more than two kernels.

3.1.3 Using a RBF Network to Generate the Mixture Coefficients.

The theory in Section 3.1 does not make any assumptions about the structure of the network used to generate the remaining parameters for the mixture model. We have however chosen to use an *RBF* network instead of the *MLP* network that was originally used in the *MDN*. One reason for this is that the calculations for the Hessian and gradient with respect to the (second layer) weights can be carried out analytically without the use of back propagation. It will also give us the possibility to initialise our weights a lot closer to the solution than with the *MLP* case.

For the non-linear optimisation of the misfit function (85) we need the gradient, and the new part is the gradient of G which, for a pattern n, is

$$\frac{\partial G}{\partial w_{ir}} = \Psi_r\bigl[\eta_i - \pi_i\bigr], \tag{89}$$

where π denotes the posterior distribution from (14) and Ψ is a $N \times M$ matrix where each element Ψ_{ij} is the activation of kernel j for pattern i. r ranges over all hidden units and i over all kernels. Note that the dependencies on n have been left out. The Hessian for one pattern n is given by

$$\mathbf{H} = \frac{\partial^2 G}{\partial w_{js}\partial w_{ir}} = \Psi_r\Psi_s\Bigl(\delta_{ij}(\eta_i - \pi_i) - \eta_i\eta_j + \pi_i\pi_j\Bigr), \tag{90}$$

which is for the re-estimation of α. The full derivations of the gradient and the Hessian can be found in Appendix A. Note that the Hessian, $\mathbf{H}$, is not positive semi-definite and caution has to be exercised when calculating the number of effective parameters, γ. We have chosen to set negative eigenvalues to zero in order to ensure $\mathbf{H}$ is positive semi-definite.

3.1.4 Initialisation of the Second Layer Weights

One advantage of using *RBF* networks in *MDN* is that we can use the linear structure of the second layer to get much better initialisation than for the original *MDN* with an *MLP* network. In section 2.5 we looked at a way to do this initialisation, but this technique does not take any penalty terms for regularisation into account. It is not possible to use this method directly since it will normally lead to weights with a large magnitude. This solution is in most cases far from the minimum of the cost function we are trying to optimise.

Since we are developing an initialisation method the result does not have to be exact; it will only be used as a first estimate of the weights. For this reason we approximate the softmax transformation of the network outputs with independent logistic output functions to simplify the procedure.

Under these assumptions the expressions for the gradient and the Hessian have an additional term corresponding to the weight decay term of the cost function. The gradient changes into $\Phi\mathbf{De}$ and the Hessian is equal to $-\Phi^T\mathbf{D}\Phi$.

Substituting the expression for the gradient and the Hessian into (79) yields

$$\mathbf{w} = (1+\alpha)\mathbf{w}_0 + \left(\Phi^T\mathbf{D}\Phi + \alpha\mathbf{I}\right)^{-1}\Phi^T\mathbf{De}, \tag{91}$$

where $\mathbf{w}$ denotes the weights after the initialisation and $\mathbf{w}_0$ is the initial value of the weights.

This is no longer the normal form equations and we have to carry out a full inversion of $(\Phi^T\mathcal{D}\Phi+\alpha\mathbf{I})$ in order to solve for $\mathbf{w}$. However, the fact that α is positive and adds to the diagonal improves the condition number of the matrix which will to some extent reduces the numerical errors that are introduced by the matrix inversion. By substituting $\mathbf{z} = \Phi^T\mathbf{w} + \mathbf{e}$ into (91) we obtain

$$\mathbf{w} = \left(\Phi^T\mathbf{D}\Phi + \alpha\mathbf{I}\right)^{-1}\Phi^T\mathbf{D}(\mathbf{z} + \alpha\Phi\mathbf{w}_0) + (1+\alpha)\alpha\mathbf{w}_0. \tag{92}$$

The value of $\mathbf{w}_0$ is set to $\mathbf{w}_0 = \Phi^\dagger\mathbf{a}$ where $\mathbf{a}$ denotes the activations of the outputs (since $a = \Phi\mathbf{w}_0$). The initial estimate of the outputs used in section 2.5 is sufficient to derive the activations.

Due to a mistake in the algebra, (92) was not used in the experiments. Instead

$$\mathbf{w} = \left(\Phi^T\mathbf{D}\Phi + \alpha\mathbf{I}\right)^{-1}\Phi^T\mathbf{Dz}. \tag{93}$$

In (93) some terms were omitted. The effect of this error is small for α close to 0 and in the experiments small α values have been used during the initialisation. For example, the difference in the cost function M between the two solutions, for a network with 18 kernels and 36 hidden units with a single regularisation class was less than five percent. After training the network parameters the effect of the error ought to be even smaller since it does not affect the training in any way. I believe the final performance of the networks was not sufficiently affected by this mistake to motivate a complete re-run of the experiments.

4 Experimental Results

In this Section the regularisation method presented in the previous section is tested on two data sets. The first data set is a synthetic set and the second is a real life data set. I compare the regularised and unregularised case and discuss the results.

4.1 The S-curve experiment

4.1.1 The Data Set

The data used in this experiment comes directly from (Bishop, 1994) The data was generated by the following model

$$x = t + 0.3 \sin(2\pi t) + 0.2\epsilon, \tag{94}$$

where ϵ is Gaussian noise with zero mean and unit variance. The training and validation sets consist of 300 patterns each and the test set of 900 patterns. This is a multi-valued function, since for some x values the conditional density $p(t|x)$ has up to three separate modes.

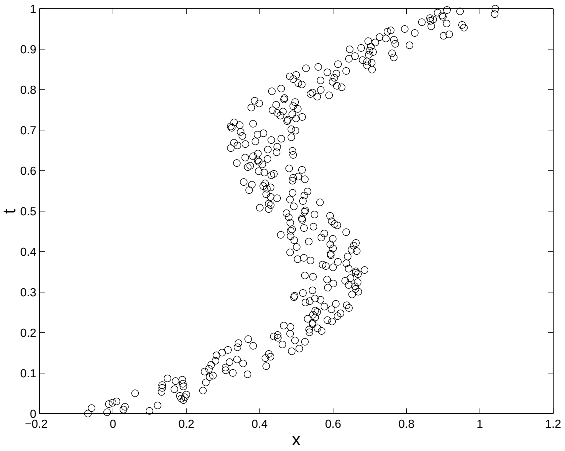


Figure 8: A scatter plot of the 300 data points in the training data that was generated with $x = t + 0.3\sin(2\pi t) + 0.2\epsilon$ where ϵ is Gaussian noise with zero mean and unit variance. For some x values there exists multiple t values, for example, $x = 0.6$ gives two different modes of the posterior distribution, one at $t \approx 0.35$ and one at $t \approx 0.8$.

4.1.2 Configurations

The models are divided into three different categories that will be compared with each other in the results section.

MDN(MLP) The first category is the standard *MDN* as described by (Bishop, 1994). Networks were trained with 3 kernels and different numbers of hidden units (5, 10 and 15). The training was done with the quasi-Newton optimiser for up to 2000 iterations with ‘early stopping’. This was repeated three times with different random seeds resulting in a total of 9 networks.

MDN(RBF, single-reg) In the second category the kernel means and variances were fixed. The means were positioned uniformly in the interval from 0 to 1 and the variance was calculated as in

Section 3.1.2. For this model, it is interesting to vary, in addition to the hidden units and random seed, the number of kernels (10, 30, 50). In total 27 networks were trained with the quasi-Newton optimiser for up to 300 iterations or until the error function converged with a change smaller than 0.0001. In the *RBF* networks *thin plate spline* (*TPS*) activation functions were used, and their positions determined by 20 iterations of the EM-algorithm. The second layer weights were initialised with the method described in section 3.1.4 for $\alpha = 0.05$. The regularisation consists of one parameter class for all second layer weights, with the biases excluded, which were updated every 4 iterations with the modified evidence procedure using the update rule which requires the Hessian to be calculated. Note that when the number of flops is given the actual calculation of the Hessian is excluded since this part is written in C for efficency reasons.

MDN(RBF, multi-reg) The final category is a small modification of the previous category where there is now one regularisation class for each network output, *i.e.* each mixing coefficient.

4.1.3 Results

After training the networks their performance was evaluated on the test set with the objective criteria that will be explained later in this section. The best network of each type was chosen and will represent its type in all plots in this section. The parameters for the best networks of each type can be seen below.

Model Type	Hidden units	Kernels	Run
MDN(MLP)	15	3	1
MDN(RBF, single-reg)	10	30	1
MDN(RBF, multi-reg)	10	30	1

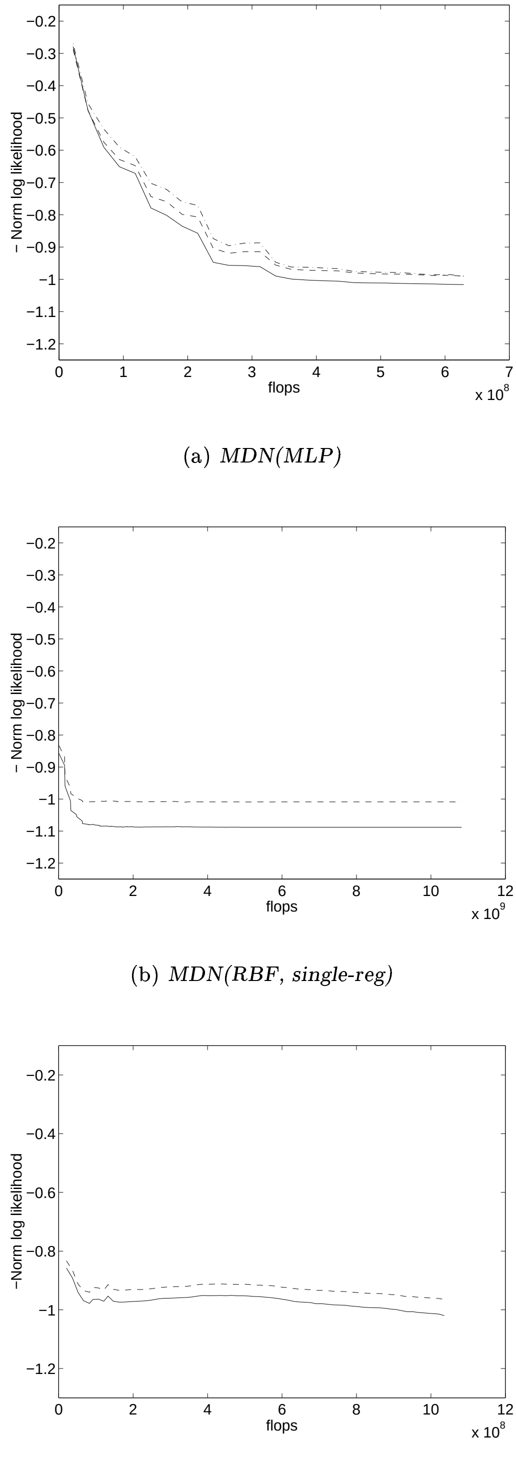
The following subsections discuss interesting features of the results and the main differences between the different categories.

Errors The error for the different partitions of the data set can be seen in Figure 9.

In (a) we can see that the network reaches a point where the validation and test errors stop decreasing but the training error continue to decrease. Interestingly enough this does not lead to much poorer generalisation, even if the validation error has some strange behaviour, probably because the data set only consits of 300 data points. The test set error has much smaller fluctuations.

(b) shows the characteristic signs of a regularised network where all errors are well correlated and the difference between the training error and the test set error is small. The frequent updates of the regularisation constant cause small variations in the regularisation constant and these fluctuations propagate forward into the error function. A large amount of the total computation time is spent on a phase where these fluctuations prevents the convergence criterion to be met, with very small impact on the log likelihood and this is the reason for the different scale on the x axis compared with (a) and (c).

(c) has well correlated errors but the shape is somewhat odd. The whole error function does have the normal monotonic decreasing form. This means that the balance between E_D and E_W changes during training and this causes the non-monotonic behaviour of E_D. It looks like the network has not converged when the optimisation criteria is reached but this is the network with the best performance of its category — and other runs of the same network converge on similar log likelihoods.

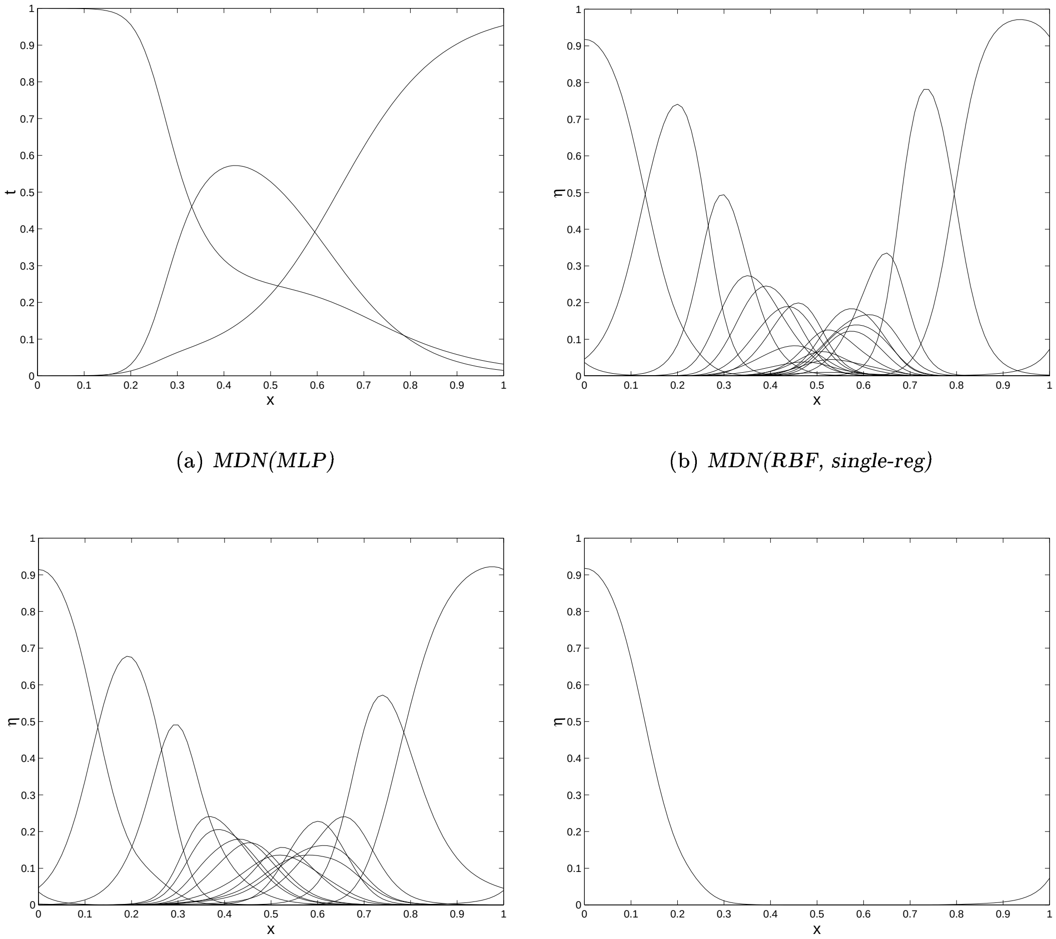


(a) *MDN(MLP)*

(b) *MDN(RBF, single-reg)*

(c) *MDN(RBF, multi-reg)*

Figure 9: Figure of the data part of the error for the training (solid), and the test (dashed) set during the training. For (a) the validation (dot-dashed) error has also been plotted.



(a) *MDN(MLP)*

(b) *MDN(RBF, single-reg)*

(c) *MDN(RBF, multi-reg)*

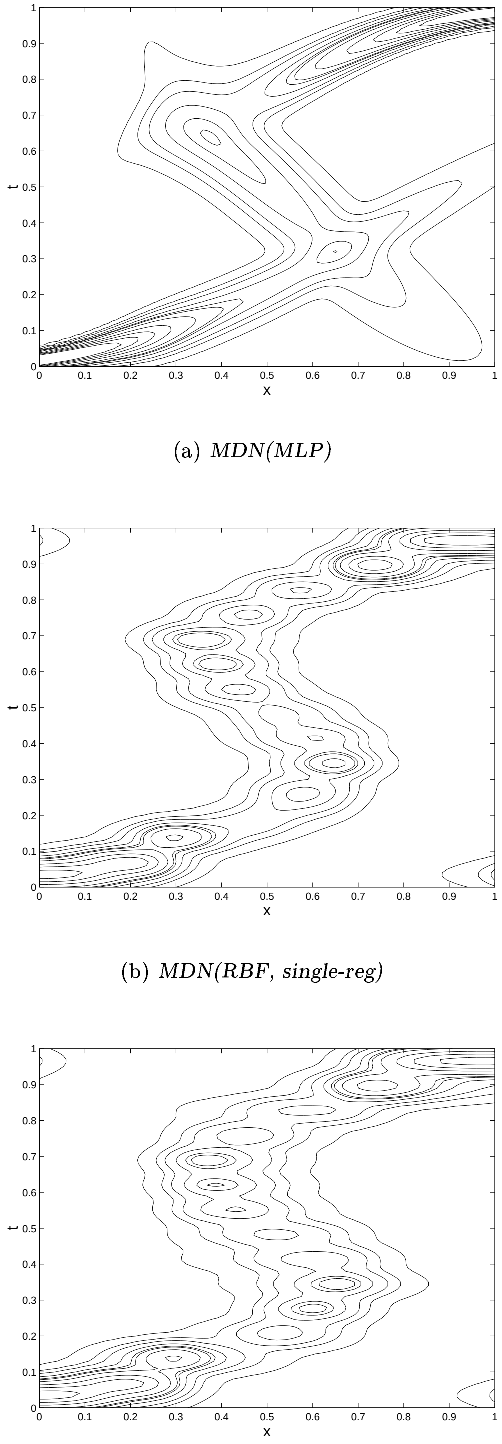
(d) Single kernel from (b)

Figure 10: (a), (b) and (c) shows the mixing coefficients for the categories. In (d) a single kernel from (b) is plotted. Note that the mixing coefficient is turned on in a regon close to $x = 1$ which is from the kernels 'responsibility' since the mean of the kernel is 0.0345.

Mixing coefficients Figure 10 shows the mixing coefficients for all categories. They all more or less look as expected and the only thing to point out is the behaviour at the edges of the interval. (d) is a plot of a single kernel, with its mean at 0.0345, from (b). We would expect the kernel to have a non-zero mixing coefficient only for x values that corresponds to small t values. The result is however different; the kernel has a non-zero mixing coefficient for *both* small x values close to zero and large x values near 1 which is as far as possible from its 'responsibility' area. This inaccuracy is caused by the small number of data points that are actually close to the end points of the interval and therefore E_D is only marginally effected by this 'error' while the extra magnitude of the weights required to avoid the 'error' will be penalised in E_W.

The Conditional Probability Density Contour plots of the conditional probability density have been plotted in Figure 11.

The major difference between the plots is the preference for continous functions for the moving kernel models. They imply a belief that if a kernel fits some data well it is likely that the position for new data outside the range of the training data can be extrapolated by assuming that the kernel means are a smooth function. This is illustrated in Figure 12 where the means of the kernels have



(a) *MDN(MLP)*

(b) *MDN(RBF, single-reg)*

(c) *MDN(RBF, multi-reg)*

Figure 11: The plot shows the conditional probability density for the networks. (a) has very smooth contours but the mixing coefficients have not 'turned off' unnecessary branches enough everywhere. (b) and (c) on the other hand have fitted the probability density well but have more ragged contours.

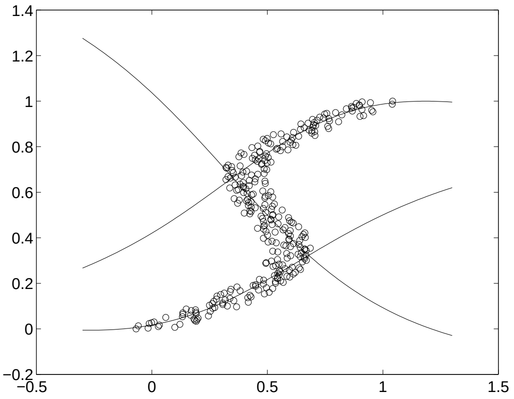


Figure 12: The kernel means for a *MDN(MLP)* are superimposed on the training data. Note that the means for x values outside the region of 0 and 1 have been extrapolated without any training data in that region.

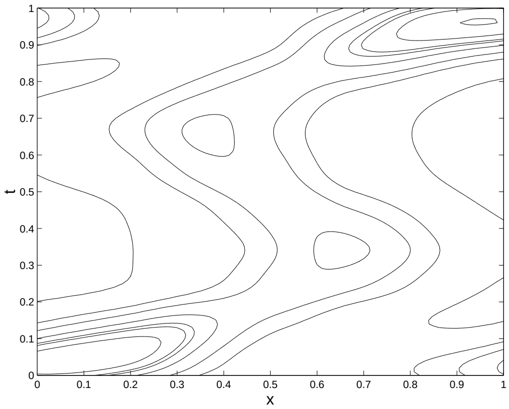
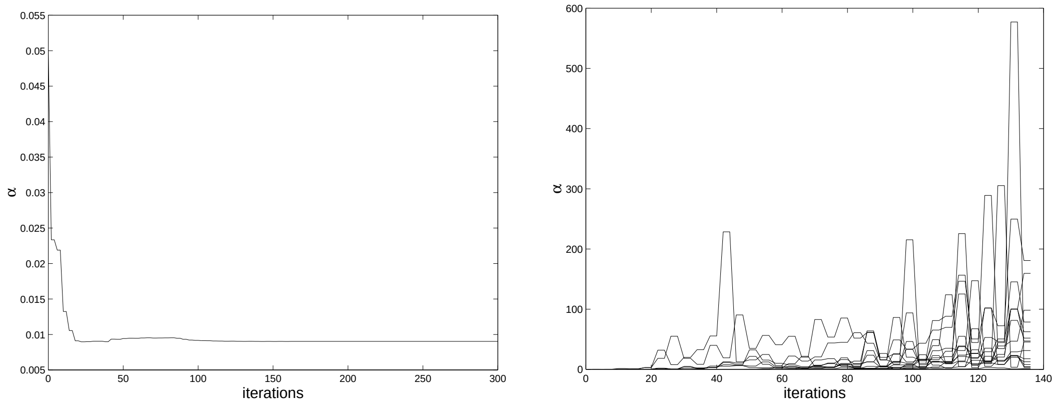


Figure 13: The plot shows the conditional probability density for a *MDN(RBF, single-reg)* that has only been initialised and is still to be trained with non-linear optimisation techniques. Most of the probability is well correlated with the density of the training set but some of it is smeared out in other areas of the target space.

been superimposed on the original data set for x values ranging from -0.3 to 1.3. (b) and (c) do not exhibit this behaviour. The mixing coefficients are optimised to fit the data well without any constraints on continuity for fixed kernels. However this approach also has its problems; two small islands of probability appear in the upper left and lower right corner. This can be seen as over-fitting at the edge of the interval where the data points are sparse for the reasons given in the discussion about the mixture coefficients earlier in this section.

Effects of initialisation The networks with fixed centres were initialised with the algorithm described in Section 3.1.4 and the results for the *MDN(RBF, single-reg)* is in Figure 13. The result of the initialisation is very good. To train a *MDN(MLP)* to the same stage of the training would take quite a few iterations. Note that some probability mass is more diffuse and the S-shape of the density is not as sharp as for a network that has been optimised — but for an initialisation, the result is surprisingly good.

Weight Decay Parameters The regularisation constants are updated as described in section 3.1.1. Figure 14 (a) shows a text book example of how we want the parameters to evolve — a very quick convergence to the final value. The results for the multiple regularisation classes in (b) are surprisingly different from the single regularisation classes case in (a). Since we now have one



(a) *MDN(RBF, single-reg)*

(b) *MDN(RBF, multi-reg)*

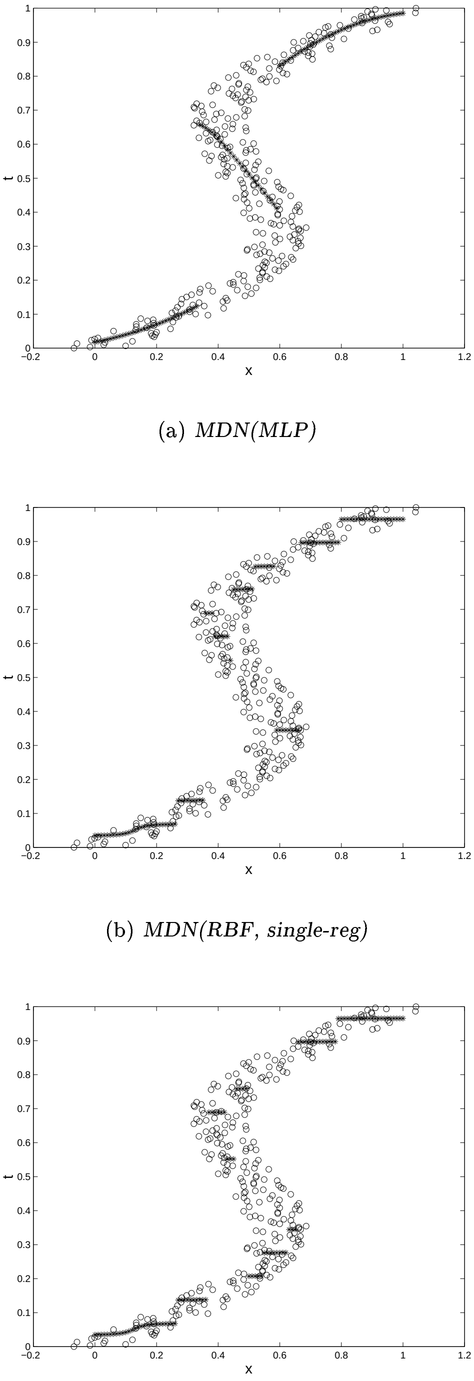
Figure 14: The figure shows how the regularisation constants change during the training. The regularisation constant was re-estimated every fourth iteration. For (a) it quickly stabilises close to the final value where in (b) it starts to oscillate for a few of the 30 regularisation classes. More and more classes seem to get involved in this oscillation

regularisation class per kernel, the most likely explanation is that for kernels responsible only for a few patterns, the evidence framework breaks down as there are too few patterns compared with the number of parameters. However, increasing the size of the training data set from 300 patterns to 1000 did not help. The phenomenon remained and the oscillation occured for most networks independently of the number of kernels and hidden units. The problem did not occur for a single regularisation class. The other explanation I can think of is a software problem but I have not been able to identify any such problem.

Making Predictions with the Model Since the output from the model is a whole distribution there are several possible methods for making predictions. In many cases we are however restricted to just making a prediction of a single point in target space and the normal choice is then to choose the most probable value. For the plots in Figure 15 non-linear optimisation on the one-dimensional conditional distribution $p(t|x)$ has been used to find the most probable value, the highest mode, which has been plotted together with the training data set.

For (a) the predictions seem like a logical choice, whereas the predictions of (b) and (c) are not equally obvious. Movable kernel models are more likely to produce interpretable results because of their preference for continuous functions. For the fixed kernel centres the predictions are simply where the model has allocated the highest probability. This leads to quantification into a few discrete output values, whereas the movable kernel models produce smoother outputs where almost every t value can be generated for a suitable input vector. This raises the question whether larger variances should be used for the kernels to increase the smoothness inspite of the results in the theory. However if we look back at the initialisation in Figure 13 this plot is very smooth and shows that the variances are sufficiently big for smooth output distributions. This suggests that the problem is the value of the second layer weights and that the likelihood is greater for the quantised case.

Error Residuals By constructing histograms of the residuals of the prediction error of the test set and the true targets, $(y - t)$ we can study the error distribution. The residuals are plotted in figure 4.1.3. The sub-plot in (c) shows that most of the predictions are a solution but it is sometimes the wrong solution — which is inevitable, and this is why the residual distribution has three modes. For the two fixed centre models this tri-modal structure is very clear.

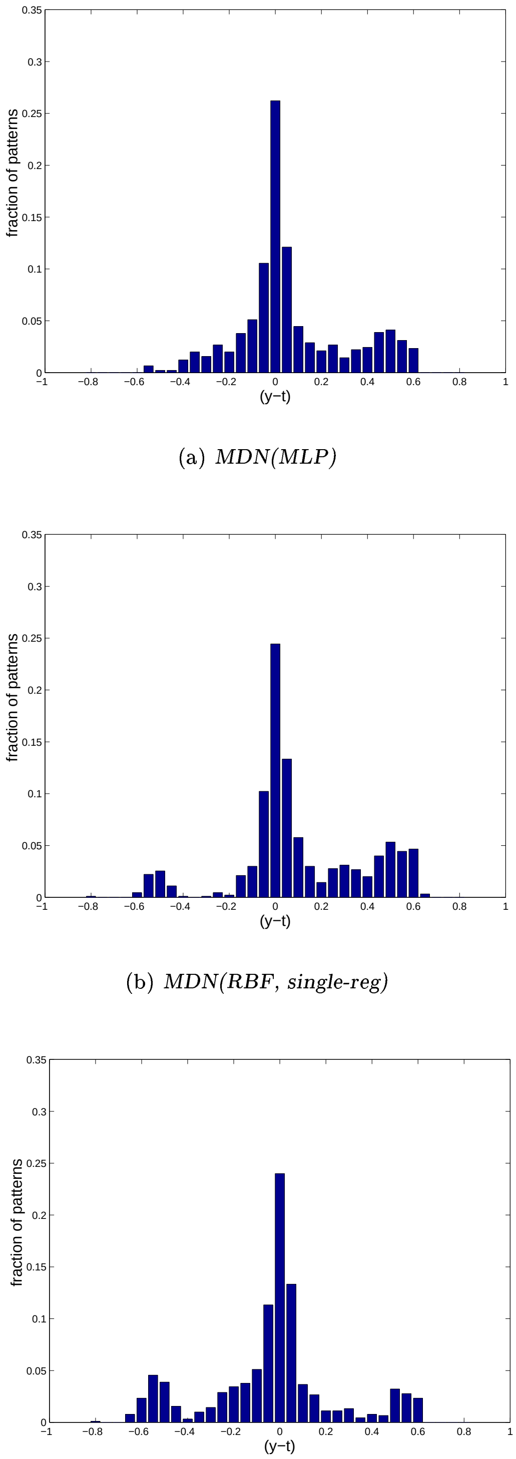


(a) *MDN(MLP)*

(b) *MDN(RBF, single-reg)*

(c) *MDN(RBF, multi-reg)*

Figure 15: The plots shows the most probable t-value (highest peak of the output distribution). (a) represents the original function well. (b) and (c) on the other hand produces good predictions but without preference for continuous functions.



(a) *MDN(MLP)*

(b) *MDN(RBF, single-reg)*

(c) *MDN(RBF, multi-reg)*

Figure 16: Histograms of the test set residuals $(y - t)$.

Model	Hid. Units	Kernels	Avg. Norm. *RMS* (opt)	Avg. *RMS* (maxη)
MDN(MLP)	15	3	0.00458	0.00548
MDN(RBF, single-reg)	10	30	0.00073	0.00096
MDN(RBF, multi-reg)	5	30	0.00130	0.00150

Table 1: Normalised *RMS* test set error averaged over the different runs with the same configuration for the best networks out of each the category. The first column of *RMS* errors is when the predictions are based on the highest mode, where the second is based on the largest mixing coefficient (the kernel with most probability mass).

Root Mean Square error (RMS) The drawback with the method of error residual histograms is that it requires someone to look at the graphs, and sometimes it is hard to tell which network is performing best. What we would like to have is an objective criterion that measures the performance as a single figure. The criterion should encourage models with multiple solutions and penalise models that only predict the average of the individual solutions. When we have access to the true generator function of the data this can be used to achieve the criteria by transforming the predictions back to the input space and then calculating the RMS error (and normalise it by dividing by the number of patterns).

This figure does not, however, say anything about how good the second and third solutions are, because we are always using the first (most probable) solution in the calculation of the error.

The normalised rms error was calculated for all the networks in each category on the test set, which consists of three times more data points than the training set. The full tables are available in Appendix B.1 and a summary can be seen in table 1 where the best average error out of each category has been selected. The two models with fixed centres perform slightly better than the adaptable kernel models. One reason for this could be that the number of branches of the data varies, which is better modelled by the configurations with fixed kernels since they have less preference for continuous functions. The differences are however marginal; all models are working well.

4.1.4 Discussion

Fixed or Adaptive Kernels? A key question is whether the kernels should be fixed or adaptive? There are advantages with both but there are some key points to consider. An adaptive kernel model will model the variance directly and this can easily be extracted; the accuracy of these estimates is however questionable since the maximum likelihood principle encourages small variances for unregularised models. These models also have a strong preference for continuous functions and that can affect the decision of which model to use. If, for example, we want to predict a time series, a moving kernel approach would probably do well where a fixed centre one would fail miserably.

One of the problems with the fixed kernel approach is that the number of parameters in the model grows rapidly. For a network with 10 hidden units and 30 kernels, which is what performed best on this problem, the model already has 340 weights. Out of these weights 300 are in the second layer, if we exclude the biases, leaving us with a Hessian consisting of 90000 elements! To get the recommended ratio between the number of parameters and the patterns in the training set mentioned in the evidence framework we need a training set at least three times as big as the current one. The effect of increasing the size of the training set that much is likely to also improve the results for the unregularised model, which for 10 hidden units and 3 kernels only has 119 parameters. What we would like is to combine the effect of regularisation with adaptive kernels so as to reduce the number of model parameters.

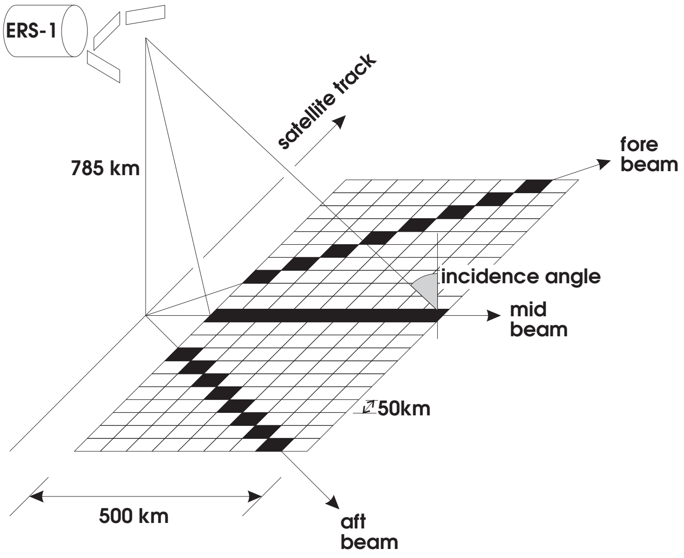


Figure 17: Scatterometer measurement process.

Consistency Another important issue is consistency. Since only three different seeds have been used it is hard to draw any definite conclusions. It does seem as though the *MDN(RBF, single-reg)* has smaller variations of the performance due to differences in the initialisation. For example, if we look at the best *MDN(MLP)* it has a normalised RMS error of 0.00060 while a different seed produces an normalised RMS error of 0.01158, approximately 20 times worse performance. For the best *MDN(RBF, single-reg)* the difference in performance between the best and the worst seed was only 0.00031. We need to train networks with several more seeds in order to evaluate the consistency with better accuracy.

4.2 Application to Radar Scatterometer Data

This is a geophysical application where the final goal is to improve weather predictions. This is currently done by a numeric weather prediction (*NWP*) model that, given the current 'state' (*i.e.* measurements of relevant variables), estimates weather conditions for the future. One of these state variables is the wind field near the ocean surface. It is not feasible to measure these winds directly and we have to resort to indirect measurements. This is done by measuring radar backscatter with a scatterometer on the ERS-1 satellite. These measurements have been shown to be correlated with the wind vector near the sea surface. Our task is to build the inverse mapping from the scatterometer data back to wind speed and direction that can be used by the *NWP*. Figure 17 shows a sketch of how the data is collected. The satellite samples a swathe of the ocean surface in a single pass. Each swathe is divided into nineteen tracks. For each track we receive a stream of three-tuples, one sample from each scatterometer, per cell. A sample from one scatterometer corresponds to to the average wind speed and direction in that cell. Each tuple is denoted σ_n^0. The target data comes from a *NWP* model. (This is very computationally intensive to run and generates overly smooth wind fields — hence the interest in using satellite data for more direct measurement.)

Handling Aliases in the Wind Direction The main problem is that the measurements from three different antennae on the satellite are ambiguous, *i.e.* certain measurements do not have a one-to-one correspondence with a unique wind vector. The problem exists for the wind direction and we often get one or more *aliases*, often 180 degrees from the true wind direction; this is illustrated in Figure 18. This ambiguity leads to a multi-modal posterior distribution and is the

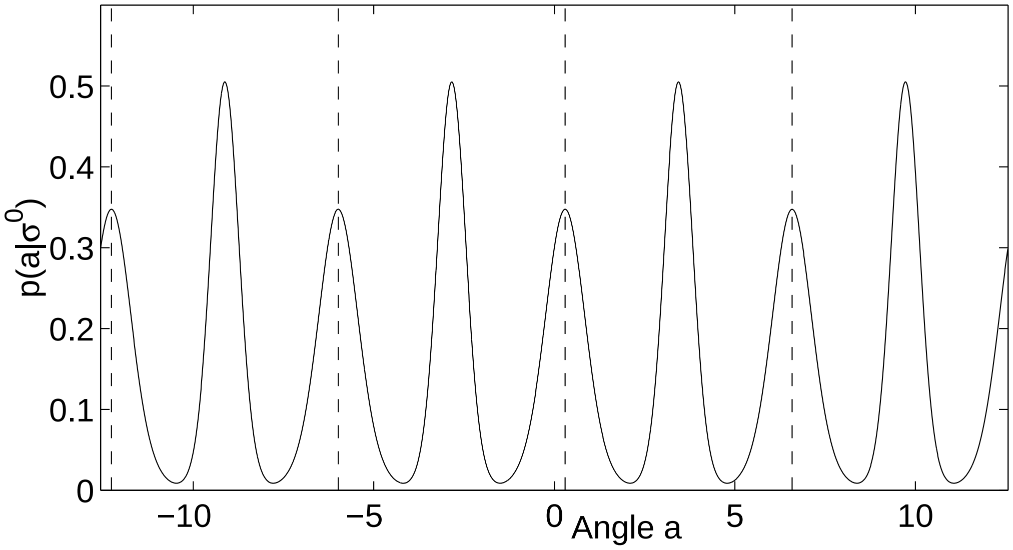


Figure 18: Sketch illustrating the ambiguity of the posterior distribution. The x axis corresponds to the angle and the y axis corresponds to the conditional probability of the angle given a specific σ^0 value. Peaks with dashed lines going through them correspond to the true wind direction whereas the others correspond to alias directions. Note that the posterior is periodic with period 2π.

reason why a *MDN* is used to model the inverse mapping, instead of a simpler network.

The Data Set For training and model evaluation we need labelled data and for this data set the labelling was made using *NWP* models. This means that the target may not correspond to the backscatter σ^0 because of errors in the model. According to Dr Cornford (personal communication) there exists a substantial number of outliers in this data set and in future work these will be removed to improve the quality of the data. A further restriction on the data is that all patterns corresponding to wind speeds below 4 meters per second have been removed. The greatest wind speed is around 25 meters per second.

In this study the training set consisted of 3000 patterns from track 11, which is a track of average difficulty. For the training set and the validation set (consisting of 1000 patterns) the number of patterns with high wind speeds was increased because they are rarer than low wind speeds and they are also of more interest for meteorologists. The test set also consisted of 1000 patterns but was sampled from the natural distribution. The data used is exactly the same data set as in (Evans, 1998b). Note that the test set was *only* used to measure the model performance and it was not used in any way during the training.

Other Relevant Research Three different approaches have been used in earlier work. The first approach was to model the wind speed and direction separately. The speed was modelled with a *MLP* network under a Gaussian noise assumption and the angle was modelled with a *MDN* using several different kernel functions (Evans, 1998a).

The second is to model *wind vectors* with components u and v (Evans, 1998b). From this vector the speed and angle can easily be calculated. Both these methods use only one output prediction for each pattern and relies on an independence assumption between samples.

The third approach relies on these sample correlations and uses the values from eight surrounding cells in addition to the current cell for making predictions (Richaume *et al.*, 1998). The predictions are made separately for the speed and direction in this model and there is one model for each track. This is different from the wind vector approach where one single model can be trained on data from all tracks and make predictions for the whole swathe.

In this study we are only going to model the wind direction since the wind speed is well modelled by conventional *MLP*s and we do this in the same way as the first approach described. The Gaussian kernels we have used earlier are not appropriate for periodic functions; we need to look at a different kernel that is better suited to this problem.

4.2.1 Modelling Probability Distributions for Periodic Functions

In order to make predictions in direction space we have to take the periodicity of the target variable into account. We choose to do this by using *Circular Normal* (also known as *von Mises* distribution) kernels (Mardia, 1972). These kernel functions can be motivated by considering a two dimensional vector $\mathbf{v}$ for which the probability distribution $p(\mathbf{v})$ is a symmetric Gaussian. By using the transformation $v_x = ||v|| \cos(\theta)$ and $v_y = ||v|| \sin(\theta)$ the conditional distribution of the direction θ, given the vector magnitude, is equal to

$$\phi(\theta|\mathbf{x}) = \frac{1}{2\pi I_0(m)} \exp\{m \cos(\theta - \varphi)\} \tag{95}$$

where φ is the centre of each kernel which corresponds to the mean and m corresponds to the inverse variance of a traditional Gaussian distribution. $I_0(m)$ is a zeroth order modified Bessel function of the first kind. The m and φ can now be initialised in an analogous way as for kernels of Gaussians. Because $I_0(m)$ is asymptotically an exponential function of m, care must be taken to avoid numerical problems with overflow in the result of intermediate calculations. The method presented in section 3 can now be used to estimate the mixing coefficients.

4.2.2 Configurations

For this experiment we are using three different models on the scatterometer data set. Two of the models are regularised with a single regularisation class was used and one unregularised model was included for comparison.

Unregularised *MDN* with Early Stopping To see how regularisation affects the performance of the final model we have trained a few networks with early stopping for comparison. The networks had $15, 25$ or 40 centres in *RBF* network and 36 Circular Normal kernels equidistantly positioned in the range from $-\pi$ to π. The variance was set using a similar method to that described in section 3.1.2 to ensure a sufficient overlap between the kernels. The validation set was used to determine the stopping criteria and the results evaluated on the test set. The centres of the *RBF* network were initialised with a maximum of 100 iterations with the EM-algorithm. The remaining weights in the second layer were initialised using the method described in Section 2.5.

Regularised *MDN* Using γ to Re-estimate the Regularisation Constant This model is the computationally most expensive because the Hessian needs to be evaluated in order to estimate γ which will in turn be used to re-estimate the regularisation parameter α. The mean and variance of the kernels were initialised as for the unregularised case. The initial value of α was subjectively chosen to a low value 0.001. α was re-estimated every 20 iterations. The models had 15 or 25 centres and 18 or 36 kernels, giving a total of four networks. The initialisation of the second layer weights now uses the modified method from Section 3.1.4. Note that when the number of flops is given the actual calculation of the Hessian is excluded since this part is written in C for efficiency reasons.

Regularised *MDN* with Approximate Re-estimation of the Regularisation Constant The only difference between this and the previously described model is that the Hessian is no longer needed for the update of α, since we no longer distinguish between well and poorly determined parameters (see Section 2.2.1). Since the update is now computationally cheap α was updated every other iteration. The models had $15, 25$ or 40 hidden units permutated with 18 and 36 kernels. For each combination two different random seeds were used for initialisation of the model parameters.

For the regularised models the validation set was not used and the performance was evaluated on the test set as for the unregularised models. The training was done with up to 2000 iterations with the quasi-Newton optimiser. These modelss will be referred to as *MDN(RBF, unreg)*, *MDN(RBF, Hess-reg)* and *MDN(RBF, noHess-reg)* respectively.

4.2.3 Results

The Normalised Error for the Data Set. Since the likelihood depends on the width of the kernel functions, it is hard to compare the performance of models with different numbers of kernels. To illustrate the effect regularisation has on the different models we look at Figure 19. This figure contains the result of training the different models with 25 centres and 36 kernels. We see that the initialisation gives us a good start; for all the models the likelihood of the test set only decreases by approximately 10 percent over the whole training period. For the unregularised network we can see the characteristic u-shaped curve showing how the network overfits the data causing decreased generalisation performance.

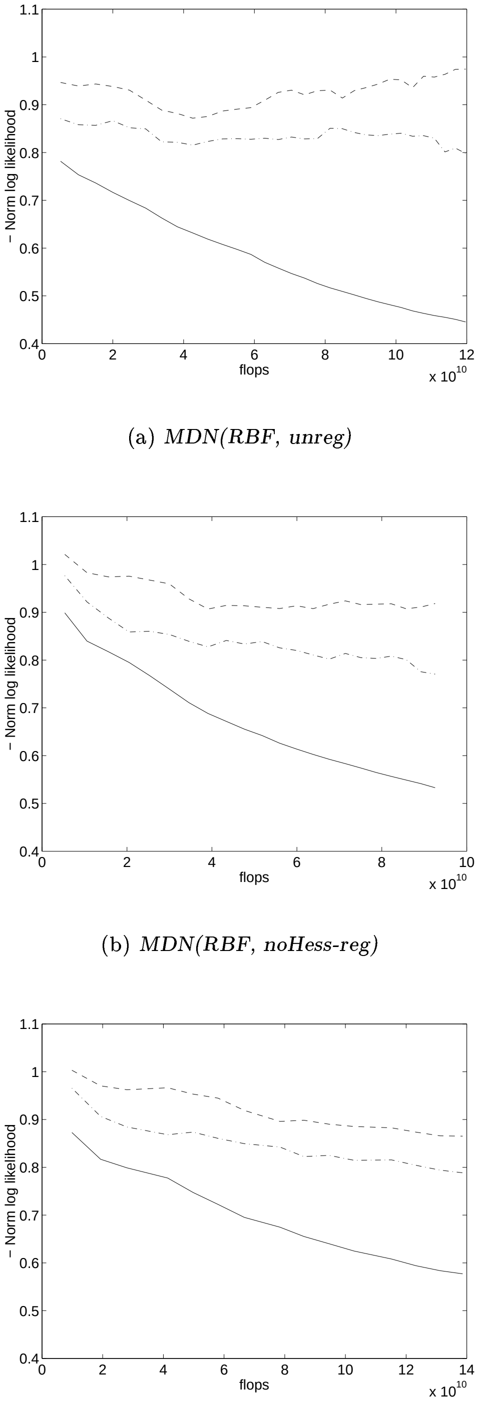
In the plots it also seems like training has not finished since the training error is still decreasing but this is not the case. The optimisation algorithm has not reached its convergence criteria and the cause is likely to be that it gets stuck in a local minimum. For the regularised networks no increase in the test error can be detected with time. The small fluctuations are due to the relatively small size of the test sets, just one third of the training set. A better size would probably be at least 6000 patterns (Having the test set to be at least twice the training set size was mentioned by (MacKay, 1992b). Note that the model using the analytical Hessian has slightly better performance.

The regularisation causes a big increase in the computation time when the Hessian needs to be evaluated. For a network with 36 kernels and 25 centres it takes approximately 50 minutes of CPU time [1] to calculate the Hessian in Matlab when the most computationally expensive code is written in C. The approximate update seem to provide a computationally cheap solution (basically one extra evaluation of M that takes less than 5 seconds to calculate), together with an objective stopping criteria that can replace ‘early stopping’. If the performance is critical or the model complexity is mis-matched the update rule requiring the Hessian normally performs better.

Convergence of the Regularisation Constant Another interesting aspect of the regularisation is how the regularisation parameter α changes over time; see Figure 20. For the majority of the models the value of α evolves roughly along the dashed line in Figure 20 and the variations for a given model with different seeds are small and they mostly converge to the same value. One reason for the slightly odd behaviour of the model using the analytical Hessian may well be that the number of parameters in the model is getting too large in comparison with the size of the training set. In this case the model had 936 parameters for a training set of 3000 patterns which is on the limit of where the evidence approximation may break down according to MacKay. For models with fewer parameters the behaviour of the regularisation constant is more predictable.

The Posterior Distribution It is important that the posterior distribution of the wind direction is a good approximation because all our predictions originate from it. For five different models of all categories we plotted the posterior distribution for the same 6 patterns, which were chosen at random from the test set. Two of these plots can be seen in figure 21 and the rest of them can be found in Section B.2. The shape of the posterior seems to depend more on the number of kernels than on the regularisation method.

[1] Using a 200 MHz R10000 Silicon Graphics Challenge.



(a) *MDN(RBF, unreg)*

(b) *MDN(RBF, noHess-reg)*

(c) *MDN(RBF, Hess-reg)*

Figure 19: The Normalised Log Likelihood for the different categories with 25 hidden units and 36 kernels. The unregularised model starts to 'over-fit' the test set. In (b) the regularisation prevents over-fitting but the results are not as good as for the model using the analytical Hessian in (c).

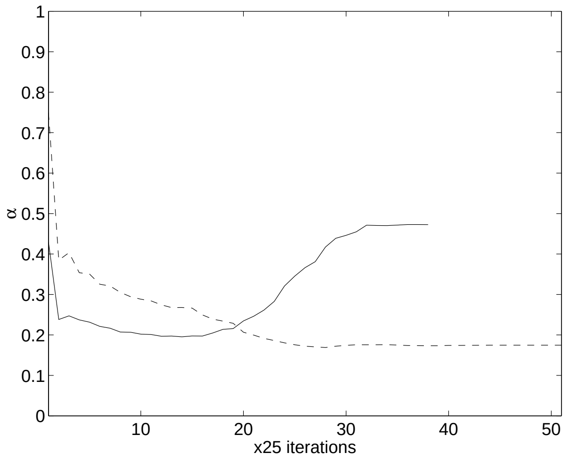
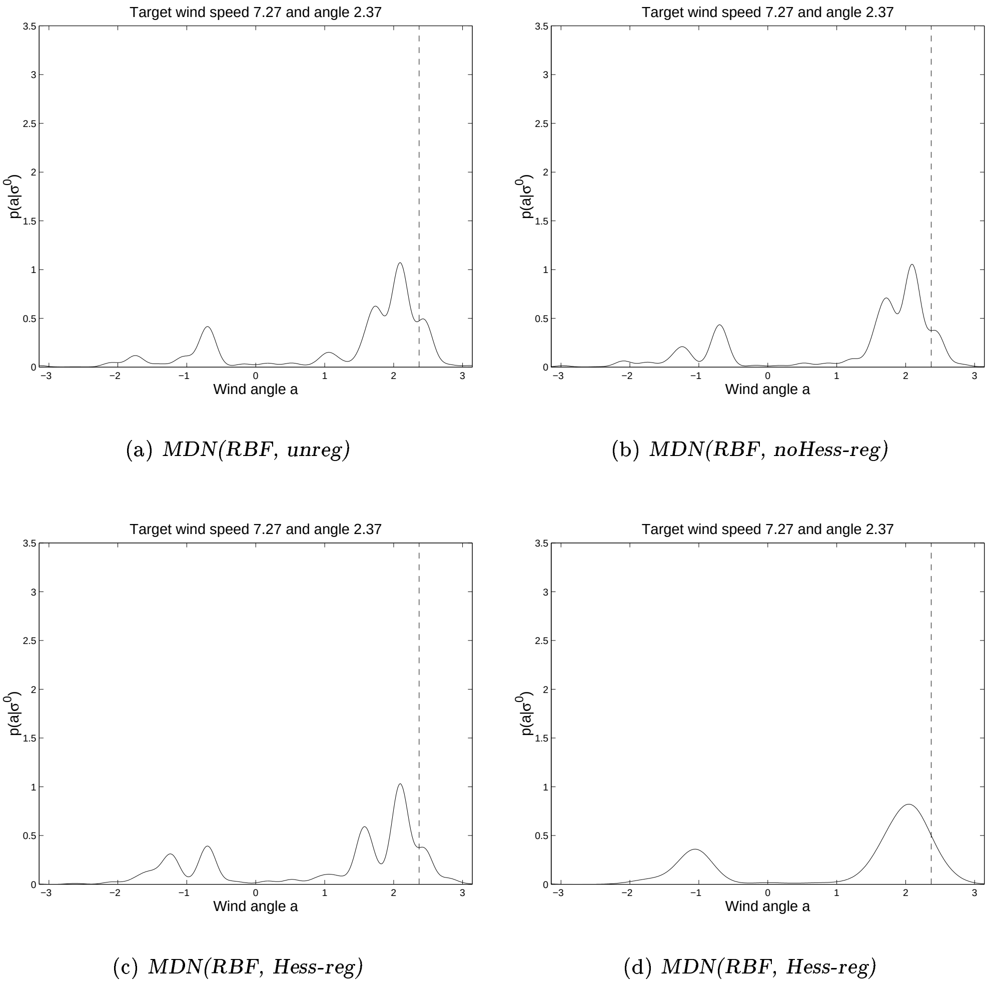


Figure 20: The value of the regularisation constant over time for the *MDN(RBF, Hess-reg)* (solid) and *MDN(RBF, noHess-reg)* (dashed) line.



(a) *MDN(RBF, unreg)*

(b) *MDN(RBF, noHess-reg)*

(c) *MDN(RBF, Hess-reg)*

(d) *MDN(RBF, Hess-reg)*

Figure 21: The effect of regularisation on the posterior distribution for the four models with 25 hidden units and 36 kernels except (d) which has only 18 kernels. The dashed line corresponds to the true target. The regularised model in (d) with fewer kernels gives a posterior close to what we believe is the true posterior with a bimodal structure. The other models have more modes and the effect of regularisation seem to be small when the number of kernels is too large.

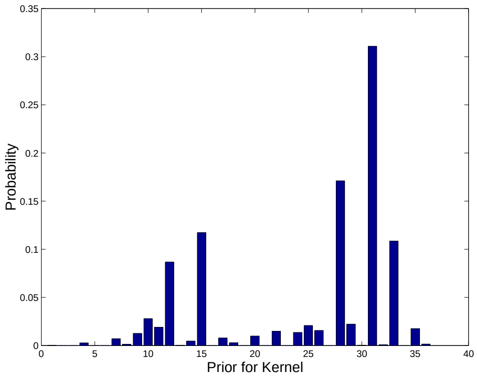


Figure 22: The mixing coefficients generated by the network for the same pattern as in figure 21. Some kernels between kernels with a high prior have a low prior themselves which seems sub-optimal.

Figure 22 shows the mixing coefficients for the same pattern as Figure 21. We can see that mixing coefficients corresponding to kernels between kernels with large mixing coefficients may have a *low* mixing coefficient themselves. This is not the behaviour we would expect. It is worrying that the regularisation does not solve this problem. It could either imply that the optimisation is stuck in a local minimum or that the difference in likelihood between the expected smooth curve and the actual result is very small. If we decrease the number of kernels to 18 this problem disappears and the posterior distribution becomes bimodal. This network has a slightly lower likelihood than the 36 kernel network and this indicates that the likelihood alone is not enough to compare models with different numbers of kernels since the variance depends on the number of kernels and the likelihood depends on the variance of the kernels.

The posteriors for the *MDN(RBF)* look very similar to posterior distributions in previous work (Cornford *et al.*, 1997) with *MDN(MLP)* that used the same kernel function. This indicates that replacing the *MLP* network with a *RBF* network does not significantly affect the models performance on this data set.

Comparing the Results with Previous Research One of the benchmarks used in previous work to compare different approaches is to measure the percentage of predictions that fall within ±20 degrees of the solution. The problem with this is how to resolve which mode that is the true solution from the multi modal distribution, a process called disambiguation. The normal approach when evaluting model performance is to pick the best solution out of the most probable modes in the posterior distribution which is normally referred to as *perfect disambiguation*. The motivation for this 'cheating' is that we want to see how well our model performs without being affected by the disambiguation procedure.

In this study the two most probable modes were found by optimisation; for each kernel the starting point was set to its mean and the optimisation was carried out. From the resulting extreme values the most probable one was chosen. The second was then calculated by taking the second most probable under the condition that it was at least at 0.1 radians from the first solution. No checks were made to ensure that the two solutions were approximately 180 degrees apart and this may have affected the results for two solutions because sometimes the estimated posterior distribution had more than two modes.

Table 2 contains the results for the unregularised networks for reference. Table 3 and 4 shows the result for the two regularised configurations. These tables show that there is a slight improvement for the regularised models but that the type of regularisation does not make a big difference with respect to this criteria. Since only one or two networks with the same parametrisation have been trained it is hard to estimate how much the performance depends on the values of the initial weights.

Percentage of predictions within 20 degrees.			
Network Parameters		Seed 2	
Centres	Kernels	1 Solution	2 Solutions
15	36	48.8	70.5
25	36	**53.6**	72.2
40	36	52.8	**73.8**

Table 2: Results from *MDN(RBF, unreg)* networks trained with 'early stopping' for three different configurations. The first solution is the maximum of the posterior. For 2 solutions the solution closest to the true target was chosen.

Percentage of predictions within 20 degrees.			
Network Parameters		Seed 2	
Centres	Kernels	1 Solution	2 Solutions
15	18	52.9	70.8
15	36	51.2	71.9
25	18	54.9	70.9
25	36	**56.3**	**72.2**

Table 3: Results after training a *MDN(RBF)* with single class regularisation where the regularisation constant was updated every twenty iterations with the analytical Hessian. The columns have been calculated in the same way as for Table 2.

Percentage of predictions within 20 degrees.					
Network Parameters		Seed 2		Seed 3	
Centres	Kernels	1 Solution	2 Solutions	1 Solution	2 Solutions
15	18	48.1	65.8	47.6	68.3
15	36	49.3	69.0	49.6	71.4
25	18	53.8	67.9	55.0	71.4
25	36	**56.8**	73.7	55.6	**74.1**
40	18	56.3	70.7	55.6	69.8
40	36	56.2	**75.0**	**56.1**	73.6

Table 4: Results from unregularised *MDN(RBF)* networks trained with 'early stopping' for three different configurations. The first solution is the maximum of the posterior. For 2 solutions the solution closest to the true target was chosen.

Percentage of predictions within 20 degrees.			
Nr	Model description	1 Solution	2 Solutions
1	*MDN*[a]	53.0	72.0
2	*MDN*[b], 4 net committee	53.8	74.2
3	*MDN*[c], trained per trace in (u, v) space	–	75.7
4	*MDN*[d], trained on all traces in (u, v) space	–	76.5
5	A-NN$_5$ [e]	–	**87.8**
6	*MDN(RBF)*, Unregularised.	52.8	73.8
7	*MDN(RBF)*, Regularised with analytical Hessian	**56.3**	72.2
8	*MDN(RBF)*, Regularised with approximative re-estimation	56.2	75.0

Table 5: The table contains previously published results together with the best results using *MDN(RBF)*. Note that configuration only configuration $3, 4, 6, 7$ and 8 have been trained on the same data set. A-NN$_5$ uses the surrounding patterns in additions to the current sample to make predictions, *i.e.* it is doing some smoothing and disambiguation.

[a] (Cornford *et al.*, 1997)
[b] (Cornford *et al.*, 1997)
[c] (Evans, 1998b)
[d] (Evans, 1998b)
[e] (Richaume *et al.*, 1998)

A comparison of the results with earlier published results is given in table 5. We can see that the regularised models performs at about the same level as all other networks using a single sample only for predictions. The A-NN$_5$ model is superior because it uses the spatial correlation between samples which effectively does some smoothing and disambiguation. Once again note that there are no error bars on this figures and the results may change depending on the initialisation of the weights. Note that different test sets were used for the different methods (different samples and different data set size).

4.2.4 Discussion

Regularisation provides an alternative to 'early stopping' The experiment has showed that this regularisation method is an alternative to 'early stopping' with similar results, where we can use our data more effectively since we do not have to allocate a validation set. It does however not really present an alternative to training a whole set of networks with systematically varying number of hidden units and kernels because having a more complex network than is actually needed gives big computational cost, especially for the regularisation using the analytical Hessian, due to the fact that we need a large number of kernels in order to represent the posterior distribution with to same precision as in the case with adaptable kernels. The results suggests that even larger networks may give better performance.

But is it not wrong to sample the training and test set from different distribution? This was something I found strange when I first learned about the current research of wind vector retrieval. Strictly speaking, it is wrong. The network should be optimised on the same distribution as it will be operating on. In this case there are three factors behind the choice to have different distributions.

Firstly, meteorologists are most interested in high wind speeds because they are the ones that can cause damage and create weather conditions that it is important to predict well in advance.

Secondly, these high wind speeds are quite rare in the atmosphere and in order to predict them accurately we need a very large data set if we don't manipulate the distribution to increase the fraction of high wind speeds. The models are computationally demanding to train this is a way to decrease it a bit.

Lastly, the signal to noise ratio is relatively low for low wind speeds. It is still possible to infer the wind speed but the wind angle is very difficult to infer at 4 meters per second.

How often should the regularisation constants be re-estimated? This is a difficult question with several factors to consider

- There is no point in fine tuning network weights if the regularisation constant is going to change much.
- Changing the regularisation constant will change the curvature and this affects the estimate of the inverse Hessian that quasi-Newton uses for the optimisation and we can expect worse performance until the estimate has adapted to the changes in the error function.
- Calculating the analytical Hessian is very computationally expensive and we have to balance this against how much it changes with time.

I do not have any objective criteria for finding a balance to this delicate problem. When using the approximate, cheap updates, it can be done frequently without cost and every two iterations seem plausible, giving smooth changes in the error function. For the expensive update rules one idea could be to ensure it is updated approximately 10 times during the progress of the training to give it a 'fair chance' of converging without calculating it too often.

Software All the experiments were carried out with Matlab 5.1 using the Netlab neural network toolbox [2] for which we wrote additional software to implement the theory in this technical report. Only one part, the computationally intensive calculation of the Hessian, was written in C.

[2] Available from `http://www.ncrg.aston.ac.uk`.

5 Discussion

In this section I briefly summarise the project and try to clarify a few of the key points. I review how the project has fulfilled the initial objectives and conclude by discussing a few possible directions for future research.

5.1 Summary

This work builds on the *MDN*, which in its standard form used an *MLP* to generate the parameters for a Gaussian Mixture Model and gives us a conditional Gaussian Mixture Model which models an arbitrary (possibly multi-modal) distribution without assuming Gaussian noise.

In the other corner we have the evidence framework, which provides a means of regularising networks where the posterior weight distribution can be approximated reasonably well with a single Gaussian and the noise is assumed to be additive Gaussian.

To make the unification of these methods easier we fixed the mean and variance of each kernel. This requires some additional initialisation for the fixed parameters and we have presented an adequate solution for this problem.

We then applied the evidence framework to this simplified model. The form of the error function is such that some integrals in the evidence framework cannot be calculated analytically but need to be approximated.

The method was applied to two data sets. For the synthetic data the regularised networks performed better than the *MDN* because they ‘turned off’ unnecessary kernels more accurately. Since the variance is fixed it cannot be decreased as for the *MDN* to increase the likelihood instead of fine tuning the mixture coefficients. For the second data set the regularised and unregularised results were similar but ‘early stopping’ can be replaced by an objective criterion. We detected a problem where some mixture coefficients were ‘turned off’ even if its ‘neighbours’ had large values. This indicated that some other form of regularisation is needed. A regulariser that penalises large differences between adjacent kernels could be appropriate for fixed kernels. This regularisation would however be difficult to generalise to moving kernels where the order of kernels can change. We have not investigated this further yet, and it is unclear if it is feasible at all.

In the experiments an *RBF* network was used to generate the Mixture Model parameters because they can be initialised much closer to the solution. We developed a method for taking the regularisation parameter into account in the initialisation.

5.2 Conclusions

Can the Fixed Kernel Approach Be Used in Practice? One problem with fixed kernels is the ‘curse of dimensionality’. All experiments in this technical report are done on one dimensional targets for this reason. I briefly looked at the robot arm kinematics problem (Bishop, 1994) where the targets are two dimensional. To achieve similar performance as a standard *MDN* at least 100 kernels were needed to model a bi-modal distribution. Even so, it did not quite match the performance of an adaptive kernel model even though the training cost was far higher.

But if we restrict ourselves to one-dimensional targets the method has similar computational cost to the standard *MDN* and has the advantage of an objective criterion for when to stop training which I believe makes the method feasible to use in practice.

Making Predictions with the Model During the experiments, predictions were made from a *maximum a posteri* (*MAP)* parameter estimates where each maximum was found by nonlinear optimisation for each posterior distribution. This is not a fully Bayesian procedure but the focus in this project was on regularisation, not prediction. MacKay (1992c) introduced the concept of *moderated outputs* where we take the weight uncertainty into account when making predictions since the *MAP* is usually over confident.

The Number of Parameters in the Model One obvious problem with this approach is that the number of parameters to optimise grows very quickly when we increase the model complexity, namely as $M(H+1)$ where M is the number of kernels and H is the number of hidden units. If we have $M = 30$ and $H = 30$, which is not unreasonable, the weight vector will be 900 elements long and the Hessian will be 900 × 900. This makes the Hessian very expensive to compute.

How Often Should the Regularisation Constants Be Updated? This problem was discussed in 4.2.4 but I mention it again here since I believe it to be a key issue. The choice of the frequency and the method for re-estimation of the regularisation constant will greatly influence both the result and the computational demand. I have tried to initialise the regularisation constant to a relatively small value to enable the initialisation to fit the data well and not get too constrained by the penalty term, and this seemed to work well in practice. Unfortunately the data set and the model complexity influences the update frequency. This area certainly could be investigated further in order to achieve more effective regularisation.

Reviewing the Objectives of This Project As stated in the introduction, there were two main objectives. It is possible to train models with about 10000 patterns for a reasonably complex network if the approximate, non-Hessian update rule is used since this hardly requires any extra computation, though changing α (and therefore changing the cost function) probably does delay convergence. The regularisation itself provides an alternative to 'early stopping' with good performance. The drawback is that the number of parameters grows rapidly and this slows down the training and makes the method infeasible for other than one-dimensional targets.

5.3 Some possible directions for future work

Extending the Theory to Adaptive Kernels The next step is certainly to loosen the restriction on the kernels and find the mean and variance by optimisation as in the original model. The main advantage is that this is a way to lessen the effect of the 'curse of dimensionality' and to reduce the number of parameters in the model.

A reduction in the number of parameters would shrink the Hessian, which in addition can be split into three parts, for the mean, variances and mixing coefficients, by using different regularisation classes for each group. The eigenvalues and inversion could be computed separately which will decrease computational cost.

Some preliminary experiments with adaptive kernels where all parameters are optimised simultaneously indicate that the variance is underestimated, which leads to a high likelihood but not necessarily to good density estimations. When the variances do 'collapse' to small values, changes in the parameters related to the mixture coefficients and the mean are only causing small changes in the likelihood in comparison with the big increase in the likelihood caused by the decrease in variance of the kernels. Changes in these parameters are therefore only marginally effecting the likelihood; further improvements on the likelihood can only be made by decreasing the variance even more. This can partly be avoided by setting the regularisation parameter for the variance to a large value during the early stages of the training until the mean and mixing coefficients have started to converge we have noticed improvement in the results where the variances does not collapse as frequently. Further investigation is needed to clarify this.

If we have a data set for which we can estimate which data points are going to belong to each centre and we use a *RBF* network the kernel means can also be initialised effectively. For an example, consider using the adaptive kernel approach for the wind data in (u, v) space. Assuming that the distribution is bimodal, which we model with two kernels, we can partition the data set in two parts: One part that has positive u components, and one with the negative u components. For each of these data subsets we calculate the design matrix and for the second layer weights of the *RBF* the corresponding kernel weights can then be solved with a matrix pseudo-inverse to give us an initial estimate of the means. This gives us an input dependent initialisation compared with the *MDN* where the initialisation only uses the unconditional mean of the targets,

Another advantage with adaptive kernels is that the variances are once again modelled explicitly and they can be used in the next step, prediction.

Semi-Adaptive Kernels One idea to overcome this overestimation of the likelihood is to divide the training into two steps

- Optimise the mixing coefficients keeping the mean and variance of the kernels fixed.
- Use the mixing coefficient calculated in the previous step, which are now fixed, to *weight* each pattern and train each kernel *separately* with the method described in Section 2.3 which provides regularisation for the mean and variance but needs to be modified to take the weighting into account.

This model will not optimise the mean and variance jointly whereas in the more straightforward generalisation to adaptive kernels all parameters are optimised at the same time. It seems likely that the convergence rate is different for the different type of parameters, for example the means often converge faster than the variances and with this approach this can be taken in to account for efficiency. It also seems possible to develop efficient initialisation for this approach.

References

Baldi, P. and Y. Chauvin 1991. Temporal evaluation of generalisation during learning in linear networks. *Neural Computation* **4** (3), pp589–603.

Bishop, C. M. 1994. Mixture Density Networks. Technical report, Department of Computer Science and Applied Mathematics, Aston University, UK.

Bishop, C. M. 1995. *Neural Networks for Pattern Recoginition.* Oxford Press.

Bishop, C. M. and C. S. Qazaz 1997. Regression with Input-Dependent Noise: A Bayesian treatment. *Advances in Neural Information Processing Systems.*

Bridle, J. S. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing: Algorithms, Architectures and Applications Ed. F Fogelman Souliè and J Hérault, Springer-Verlag*, pp. 227–236.

Cornford, D., I. T. Nabney, and C. M. Bishop 1997. Neural Network Based Wind Vector Retrieval from Satellite Scatterometer Data. Technical report, Neural Computing Research Group, Aston University, Birmingham B4 7ET, UK.

Evans, D. J. 1998a. Mixture Density Network Training by Computation in Parameter Space. Technical report, Neural Computing Research Group, Aston University, Birmingham B4 7ET, UK. NCRG/98/016.

Evans, D. J. 1998b. Neural Networks for Extracting Wind Vectors from Satellite Scatterometer Data. Master's thesis, Neural Computing Research Group, Aston University, Birmingham B4 7ET, UK.

MacKay, D. J. C. 1992a. Bayesian Interpolation. *Neural Computation* **4** (3), 415–447.

MacKay, D. J. C. 1992b. *Bayesian Methods for Adaptive Models.* Ph.D. thesis, California Institute of Technology, Pasadena, California.

MacKay, D. J. C. 1992c. The Evidence Framework Applied to Classification Networks. *Neural Computation* **4** (5), 698–714.

MacKay, D. J. C. 1992d. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation* **4** (3), 448–472.

Mardia, K. V. 1972. *Statistics of Directional Data.* Academic Press, London.

McCullagh, P. and J. A. Nelder 1983. *Generalized Linear Models.* London:Chapman and Hall.

McLachlan, G. J. and K. E. Basford 1988. *Mixture Models: Inference and Applications to clustering.* Marcel Dekker, New York.

Nabney, I. T. 1998. Efficient training of RBF Networks for Classification. to be published.

Nix, A. D. and A. S. Weigend 1994. Estimating the Mean and Variance of the Target Probability Distribution. In *Proceeding of the IEEEE International Conference on Neural Networks*, pp. pp 55–60. IEEE.

Press, W. H., A. S. Teukolsky, T. W. Vetterling, and B. P. Flemming 1992. *Numerical Recipes in C* (Second ed.). Cambridge University Press.

Qazaz, C. S. 1996. *Bayesian Error Bars for Regression.* Ph.D. thesis, Aston University.

Richaume, P., F. Badran, M. Crepon, C. Mejia, H. Roquet, and S. Thiria 1998. Neural Network Wind Retrieval from ERS-1 Scatterometer Data. Submitted to: Journal of Geophysical Research.

Tikhonov, A. N. and V. Y. Arsenin 1977. *Solutions of Ill-Posed Problems.* Washington DC: V. H. Winston.

Williams, P. M. 1994. Bayesian Regularisation and Pruning using a Laplace Prior. Technical report, School of Cognitive and Computing Sciences, University of Sussex.

Williams, P. M. 1996. Using Neural Networks to Model Conditional Multivariate Densities. *Neural Computation* **8**, 843–854.

A Calculations

The calculations in this section are for Mixture Density Networks with spherical Gaussian kernels defined as follows

Definition 1
When the covariance matrix of a Gaussian can be written as a constant times the unit matrix ($\Sigma = \sigma\mathbf{I}$) *it is called* spherical *and for d dimensions it can be written as*

$$\phi(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left(-\frac{||\mathbf{x} - \mu||^2}{2\sigma^2} \right)$$

where μ is a d dimensional vector with the means and σ^2 is the variance.

We start by calculating the gradient and then we continue to calculate the Hessian for *RBF* networks with fixed centres. The gradients are however valid for all *MDN*s with spherical Gaussian kernels.

A.1 Calculating the gradient

The error function for one pattern is

$$E^n = -\ln \sum_{i=1}^{M} \eta(\mathbf{x}^n)\phi_i(\mathbf{t}^n|\mathbf{x}^n). \tag{96}$$

For convenience and clarity the dependence on x^n and t^n will be omitted. $\eta_i(\mathbf{x}^n)$ will be written as η_i, $\Psi_i(\mathbf{x}^n)$ as Ψ_i and $\Phi_i(\mathbf{t}^n|\mathbf{x}^n)$ as ϕ_i.

The gradient with respect to the means Expanding the error function (96) with the chain rule gives

$$\frac{\partial E}{\partial z^{\mu}_{ik}} = \frac{\partial E}{\partial \phi_i}\frac{\partial \phi_i}{\partial \mu_{ik}}\frac{\partial \mu_{ik}}{\partial z^{\mu}_{ik}}. \tag{97}$$

The first factor can be calculated by applying the quotient rule on (96):

$$\frac{\partial E}{\partial \phi_i} = -\frac{\eta_i}{\sum_{j'=1}^{m} \eta_{j'}\phi_{j'}} \tag{98}$$

The second factor is the derivative of Definition 1 with respect to the mean as follows

$$\frac{\partial \phi(\mathbf{x})}{\partial \mu_k} = \phi(\mathbf{x})\frac{(x_k - \mu_k)}{\sigma^2} \qquad \text{for } k = 1, 2, \ldots, d. \tag{99}$$

The last factor of (97) is 1 since $\mu_{ik} = z^{\mu}_{ik}$. Substituting these three equations into (97) gives

$$\frac{\partial E}{\partial z^{\mu}_{ik}} = -\frac{\eta_i\phi_i}{\sum_{j'=1}^{m} \eta_{j'}\phi_{j'}}\frac{(\mu_{ik} - t_k)}{\sigma_i^2} \cdot 1 \tag{100}$$

It is now suitable to define the posterior π_i, for kernel i to be

$$\pi_i = \frac{\eta_i\phi_i}{\sum_{j'=1}^{m} \eta_{j'}\phi_{j'}} \qquad \text{for } i = 1, 2, \ldots, M. \tag{101}$$

and to substitute this into (100) and get the following Lemma.

Lemma 1
The gradient with respect to the network outputs for the means are

$$\frac{\partial E}{\partial z_{ik}^{\mu}} = \pi_i \frac{(\mu_{ik} - t_k)}{\sigma_i^2}. \tag{102}$$

for all components $i = 1, 2, \ldots, M$ over all dimensions in target space $k = 1, 2, \ldots, d$.

The gradient with respect to the standard deviations Applying the chain rule on the error function in (96) gives

$$\frac{\partial E}{\partial z_i^{\sigma}} = \frac{\partial E}{\partial \phi_i} \frac{\partial \phi_i}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial z_i^{\eta}}. \tag{103}$$

The second factor is found by differentiating Definition 1 with respect to the standard deviation which gives

$$\frac{\partial \phi(\mathbf{x})}{\partial \sigma} = \phi(\mathbf{x})\Big(\frac{||\mathbf{t} - \mu||^2}{\sigma^3} - \frac{d}{\sigma}\Big) \tag{104}$$

and the third factor is

$$\frac{\partial \sigma_i}{\partial z_i^{\eta}} = \exp(z_i^{\eta}) = \eta_i. \tag{105}$$

Substituting (98), (104) and (105) into (103) gives

$$\frac{\partial E}{\partial z_i^{\sigma}} = -\frac{\eta_i}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} \phi_i \Big(\frac{||\mathbf{t} - \mu_i||^2}{\sigma_i^3} - \frac{d}{\sigma_i}\Big) \sigma_i \tag{106}$$

This can be simplified by using the definition of π from (101) to get the following lemma:

Lemma 2
The gradient with respect to the network outputs for the standard deviation parameters are

$$\frac{\partial E}{\partial z_i^{\sigma}} = -\pi_i \left(\frac{||\mathbf{t} - \mu_i||^2}{\sigma_i^2} - d\right).$$

for all mixtures $i = 1, 2, \ldots, M$.

The gradient with respect to the mixing coefficients Expanding (96) with the chain rule gives

$$\frac{\partial E}{\partial z_i^{\eta}} = \sum_{m=1}^{M} \frac{\partial E}{\partial \eta_m} \frac{\partial \eta_m}{\partial z_i^{\eta}} \tag{107}$$

Differentiating of the first factor with the quotient rule gives

$$\frac{\partial E}{\partial \eta_m} = -\frac{\phi_m}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} \frac{\eta_m}{\eta_m} = -\frac{\pi_m}{\eta_m} \tag{108}$$

The second term of (107) is a bit more complicated and we start by recalling the definition of the mixing coefficients to be

$$\eta_m = \frac{\exp(z_m^{\eta})}{\sum_{j=1}^{M} \exp(z_j^{\eta})}. \tag{109}$$

Differentiating this expression when $m = i$ gives

$$\begin{aligned}\frac{\partial \eta_m}{\partial z_i^\eta} &= \frac{\exp(z_m^\eta)\sum_{j=1}^M \exp(z_j^\eta) - \exp(z_m^\eta)\exp(z_i^\eta)}{\left(\sum_{j=1}^M \exp(z_j^\eta)\right)^2} \\ &= \frac{\exp(z_m^\eta)}{\sum_{j=1}^M \exp(z_j^\eta)} - \frac{\exp(z_m^\eta)}{\sum_{j=1}^M \exp(z_j^\eta)}\frac{\exp(z_i^\eta)}{\sum_{j=1}^M \exp(z_j^\eta)} = \eta_m - \eta_m\eta_i\end{aligned} \tag{110}$$

and for $m \neq i$ we get

$$\frac{\partial \eta_m}{\partial z_i^\eta} = -\frac{\exp(z_m^\eta)\exp(z_i^\eta)}{\left(\sum_{j=1}^M \exp(z_j^\eta)\right)^2} = -\eta_m\eta_i. \tag{111}$$

$$\tag{112}$$

Combining the cases together gives us this lemma:

Lemma 3
The derivative of the network output with respect to the mixing coefficient is:

$$\frac{\partial \eta_m}{\partial z_i^\eta} = \delta_{mi}\eta_m - \eta_m\eta_i$$

for all $m, i = 1, 2, \ldots, M.$

Using this Lemma and (108) to substitute into (107) gives

$$\frac{\partial E}{\partial z_i^\eta} = -\sum_{m=1}^M \frac{\pi_m}{\eta_m}\left(\delta_{mi}\eta_m - \eta_m\eta_i\right) = -\frac{\pi_i}{\eta_i}\eta_i + \sum_{m=1}^M \frac{\pi_m}{\eta_m}\eta_m\eta_i \tag{113}$$

$$= -\pi_i + \eta_i \underbrace{\sum_{m=1}^M \pi_m}_{=1} = -\pi_i + \eta_i \tag{114}$$

where we have used the property that the posterior should sum up to one in the last step. This gives the final expression for the gradient.

Lemma 4
The gradient with respect to the network outputs for the mixing coefficients are

$$\frac{\partial E}{\partial z_i^\eta} = \eta_i - \pi_i$$

for all mixtures $i = 1, 2, \ldots, M.$

The gradient with respect to the second layer weights Lemma 1, 2 and 4 give the derivative with respect to the network outputs. This far the results are independent on the network in the *MDN*. If the network is a *MDN(MLP)* the gradient with respect to the network weights are calculated by back-propagation. When we have a RBF network with fixed centres the situation is a bit different and we can calculate the gradients for the second layer weights analytically. (The derivative of the first layer weights are zero for fixed centres.)

Since all hidden units are of the same type this can be done for all second layer weights in one step. The network output is:

$$z_i = \sum_{r=1}^R \Psi_r w_{ri} \qquad \text{where R is the number of hidden units.} \tag{115}$$

Differentiating this gives

$$\frac{\partial z_i}{\partial w_{ir}} = \Psi_r \tag{116}$$

for all three kinds of outputs.

Using lemma 1, 2 4 and (116) the gradient can be expressed as the following proposition:

Proposition 1
The gradient for the MDN(RBF) is a vector $\mathbf{G} = \{\mathbf{G}^{\mu}, \mathbf{G}^{\sigma}, \mathbf{G}^{\eta}\}$ *where each component can be written as*

$$G^{\mu}_{ikr} = \Psi_r \pi_i \frac{(\mu_{ik} - t_k)}{\sigma_i^2} \tag{117}$$

$$G^{\sigma}_{ir} = -\Psi_r \pi_i \left(\frac{||\mathbf{t} - \mu_i||^2}{\sigma_i^2} - d \right) \tag{118}$$

$$G^{\eta}_{ir} = \Psi_r \big[\eta_i - \pi_i \big] \tag{119}$$

for $i = 1, 2, \ldots, M$, $k = 1, 2, \ldots, d$ *and* $r = 1, 2, \ldots, R$.

A.2 Calculating the Hessian

The Hessian for the means Expanding (117) gives

$$\frac{\partial G^{\mu}_{ikr}}{\partial w_{jls}} = \frac{\partial G^{\mu}_{ikr}}{\partial z^{\mu}_{jl}} \frac{\partial z^{\sigma}_{jl}}{\partial w_{jls}}. \tag{120}$$

Calculating the first term

$$\frac{\partial G^{\mu}_{ikr}}{\partial z^{\mu}_{jl}} = \Psi_r \left[\frac{\partial \pi_i}{\partial z^{\mu}_{jl}} \frac{(\mu_{ik} - t_k)}{\sigma_i^2} + \frac{\pi_i}{\sigma_i^2} \frac{\partial \mu_{ik}}{\partial z^{\mu}_{jl}} \right] \tag{121}$$

where we first need to calculate the following derivative

$$\frac{\partial \pi_i}{\partial z^{\mu}_{jl}} = \frac{\partial \pi_i}{\partial \phi_j} \frac{\partial \phi_j}{\partial z^{\mu}_{jl}} \tag{122}$$

Differentiating (101) with respect to ϕ_i, starting with the case where $i = j$

$$\frac{\partial \pi_i}{\partial \phi_j} = \frac{\eta_i (\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}) - \eta_i \phi_i \eta_j}{\left(\sum_{j'=1}^{m} \eta_{j'} \phi_{j'} \right)^2} = \frac{\eta_i}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} - \frac{\eta_j \pi_i}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} \tag{123}$$

otherwise, when $i \neq j$

$$\frac{\partial \pi_i}{\partial \phi_i} = -\frac{\eta_i \phi_i \eta_j}{\left(\sum_{j'=1}^{m} \eta_{j'} \phi_{j'} \right)^2} = -\frac{\eta_j \pi_i}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} \tag{124}$$

The cases are combined back together with a Kronecker delta.

$$\frac{\partial \pi_i}{\partial \phi_i} = \delta_{ij} \frac{\eta_i}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} - \pi_i \frac{\eta_j}{\sum_{j'=1}^{m} \eta_{j'} \phi_{j'}} \tag{125}$$

The second factor of equation (122) is

$$\frac{\partial \phi_j}{\partial z^{\mu}_{jl}} = \Phi_j \frac{(\mu_{jl} - t_k)}{\sigma^2} \tag{126}$$

Equation (125) and (126) can now be substituted into (122) to give

$$\frac{\partial \pi_i}{\partial z^\mu_{jl}} = \Big(\delta_{ij}\frac{\eta_i}{\sum_{j'=1}^m \eta_{j'}\phi_{j'}} - \pi_i \frac{\eta_j}{\sum_{j'=1}^m \eta_{j'}\phi_{j'}}\Big)\Phi_j \frac{(\mu_{jl}-t_k)}{\sigma^2} = \delta_{ij}\pi_i - \pi_i\pi_j \tag{127}$$

which in turn can be substituted into (121) as follows

$$\frac{\partial G^\mu_{ikr}}{\partial z^\mu_{jl}} = \Psi_r\left[\Big(\delta_{ij}\pi_i - \pi_i\pi_j\Big)\frac{(\mu_{jl}-t_l)}{\sigma_j^2}\frac{(\mu_{ik}-t_k)}{\sigma_i^2} - \delta_{ij}\delta_{kl}\frac{\pi_i}{\sigma_i^2}\right] \tag{128}$$

Substituting this expression and (116) into (119) gives the final expression.

Lemma 5
The Hessian with respect to the second layer weights for outputs generating mean parameters in the mixture model are

$$\frac{\partial G^\mu_{ikr}}{\partial w_{jls}} = \Psi_r\Psi_s\left[\Big(\delta_{ij}\pi_i - \pi_i\pi_j\Big)\frac{(\mu_{jl}-t_l)}{\sigma_j^2}\frac{(\mu_{ik}-t_k)}{\sigma_i^2} - \delta_{ij}\delta_{kl}\frac{\pi_i}{\sigma_i^2}\right]$$

The Hessian for the standard deviations The second derivative can be written as

$$\frac{\partial G^\sigma_{ik}}{\partial w^\sigma_{jl}} = \frac{\partial G^\sigma_{ik}}{\partial z^\sigma_{jl}}\frac{\partial z^\sigma_{jl}}{\partial w^\sigma_{jl}} \tag{129}$$

The first term is the derivative for (118) can be calculated by splitting into two cases, first for $i = j$

$$\begin{aligned}\frac{\partial G^\sigma_{ik}}{\partial z^\sigma_{jl}} &= -\Psi_r\left[\frac{\partial \pi_i}{\partial z^\sigma_{jl}}\left(\frac{\|t-\mu_i\|^2}{\sigma_i^2} - d\right) + \pi_i\frac{\|t-\mu_i\|^2}{\sigma_i^3}(-2)\underbrace{\frac{\partial \sigma_i}{\partial z^\sigma_{jk}}}_{=\sigma_i}\right] \\ &= -\Psi_r\left[(\delta_{ij}\pi_i - \pi_i\pi_j)\left(\frac{\|t-\mu_j\|^2}{\sigma_j^2} - d\right)\left(\frac{\|t-\mu_i\|^2}{\sigma_i^2} - d\right) - 2\pi_i\frac{\|t-\mu_i\|^2}{\sigma_i^3}\right]\end{aligned} \tag{130}$$

and otherwise

$$\frac{\partial G^\sigma_{ik}}{\partial z^\sigma_{jl}} = -\Psi_r\left[(\delta_{ij}\pi_i - \pi_i\pi_j)\left(\frac{\|t-\mu_j\|^2}{\sigma_j^2} - d\right)\left(\frac{\|t-\mu_i\|^2}{\sigma_i^2} - d\right)\right] \tag{131}$$

which can be combined into

$$\frac{\partial G^\sigma_{ik}}{\partial z^\sigma_{jl}} = -\Psi_r\left[(\delta_{ij}\pi_i - \pi_i\pi_j)\left(\frac{\|t-\mu_j\|^2}{\sigma_j^2} - d\right)\left(\frac{\|t-\mu_i\|^2}{\sigma_i^2} - d\right) - 2\delta_{ij}\pi_i\frac{\|t-\mu_i\|^2}{\sigma_i^2}\right] \tag{132}$$

Substituting (132) and (116) into (129) gives us the following lemma:

Lemma 6
The Hessian with respect to the second layer weights for outputs generating variance parameters in the mixture model are

$$\frac{\partial G^\sigma_{ik}}{\partial w^\sigma_{jl}} = -\Psi_r\Psi_s\left[(\delta_{ij}\pi_i - \pi_i\pi_j)\left(\frac{\|t-\mu_j\|^2}{\sigma_j^2} - d\right)\left(\frac{\|t-\mu_i\|^2}{\sigma_i^2} - d\right) - 2\delta_{ij}\pi_i\frac{\|t-\mu_i\|^2}{\sigma_i^2}\right]$$

The Hessian for mixing coefficients The expression for the Hessian is

$$\frac{\partial G^{\eta}_{ir}}{\partial w_{js}} = \sum_{m=1}^{M} \frac{\partial G^n}{\partial \eta_m} \frac{\partial \eta_m}{\partial z_j} \frac{\partial z_j}{\partial w_{js}}. \tag{133}$$

Starting with the first factor we get

$$\frac{\partial G^n}{\partial \eta_m} = \frac{\partial}{\partial \eta_m}\big(\Psi_r(\eta_k - \pi_k)\big) = \Psi_r\Big(\frac{\partial \eta_k}{\partial \eta_m} - \frac{\partial \pi_k}{\partial \eta_m}\Big). \tag{134}$$

The derivative of η_k with respect to m is only non-zero if $k = m$. This gives

$$\frac{\partial G^n}{\partial \eta_m} = \Psi_r\Big(\delta_{im} + \frac{\pi_i\pi_m - \delta_{im}\pi_i}{\eta_m}\Big). \tag{135}$$

The next step is to calculate the sum from equation (133) by substitution from (135) and (3) which yields

$$\begin{aligned}
\sum_{m=1}^{M} \frac{\partial G^n}{\partial \eta_m}\frac{\partial \eta_m}{\partial z_j} &= \sum_{m=1}^{M} \Psi_r\Big(\delta_{im} + \frac{\pi_i\pi_m - \delta_{im}\pi_i}{\eta_m}\Big)(\delta_{mj}\eta_m - \eta_m\eta_j)\\
&= \Psi_r\Big[(\delta_{ij}\eta_i - \eta_i\eta_j) + \pi_i\Big(\sum_{m=1}^{M} \frac{\pi_m - \delta_{im}}{\eta_m}(\delta_{mj} - \eta_j)\eta_m\Big)\Big]\\
&= \Psi_r\Big[\delta_{ij}\eta_i - \eta_i\eta_j + \pi_i\Big(\sum_{m=1}^{M} \pi_m\delta_{mj} - \pi_m\eta_j - \delta_{im}\delta_{mj} + \delta_{im}\eta_j\Big)\Big]\\
&= \Psi_r\big[\delta_{ij}\eta_i - \eta_i\eta_j + \pi_i(\pi_j - \eta_j - \delta_{ij} + \eta_j)\big]\\
&= \Psi_r\big[\delta_{ij}\eta_i - \eta_i\eta_j + \pi_i\pi_j - \pi_i\delta_{ij}\big]\\
&= \Psi_r\big[\delta_{ij}(\eta_i - \pi_i) - \eta_i\eta_j + \pi_i\pi_j\big].
\end{aligned} \tag{136}$$

Substituting (116) and (136) into (133) gives

$$\begin{aligned}
\sum_{m=1}^{M} \frac{\partial G^n}{\partial \eta_m}\frac{\partial \eta_m}{\partial z_j}\frac{\partial z_j}{\partial w_{js}} &= \Psi_r\left[\delta_{ij}(\eta_i - \pi_i) - \eta_i\eta_j + \pi_i\pi_j\right]\Psi_s\\
&= \Psi_r\Psi_s\big[\delta_{ij}(\eta_i - \pi_i) - \eta_i\eta_j + \pi_i\pi_j\big].
\end{aligned} \tag{137}$$

Lemma 7
The Hessian with respect to the network outputs for the mixing coefficients are

$$\frac{\partial G^{\eta}_{ir}}{\partial w_{js}\partial w_{ir}} = \Psi_r\Psi_s\Big(\delta_{ij}(\eta_i - \pi_i) - \eta_i\eta_j + \pi_i\pi_j\Big). \tag{138}$$

Proposition 2
The Hessian with respect to the outputs for the different kinds of outputs can be expressed as

$$\begin{aligned}
\frac{\partial^2 E^n}{\partial w_{ki}\partial w_{lj}} &= -\Psi_r\Psi_s\left[\big(\delta_{ij}\pi_i - \pi_i\pi_j\big)\left(\frac{||t-\mu_j||^2}{\sigma_j^2} - d\right)\left(\frac{||t-\mu_i||^2}{\sigma_i^2} - d\right) - 2\delta_{ij}\pi_i\frac{||t-\mu_i||^2}{\sigma_i^2}\right], \quad \textit{(means)}\\
\frac{\partial^2 E^n}{\partial w_{ki}\partial w_{lj}} &= -\Psi_r\Psi_s\left[\big(\delta_{ij}\pi_i - \pi_i\pi_j\big)\left(\frac{||t-\mu_j||^2}{\sigma_j^2} - d\right)\left(\frac{||t-\mu_i||^2}{\sigma_i^2} - d\right) - 2\delta_{ij}\pi_i\frac{||t-\mu_i||^2}{\sigma_i^2}\right], \quad \textit{(variances)}\\
\frac{\partial^2 E^n}{\partial w_{ki}\partial w_{lj}} &= \Psi_r\Psi_s\Big(\delta_{ij}(\eta_i - \pi_i) - \eta_i\eta_j + \pi_i\pi_j\Big), \quad \textit{(mixing coefficients)}
\end{aligned}$$

Note that for the full Hessian several cross products need to be evaluated but they are not needed in the current application and have therefore been left out.

B Results

B.1 The S-curve Results

B.1.1 Results for the standard *MDN*

Network		RMS (opt) for network				RMS ($\max \eta$) for network			
Cen	Kern	1	2	3	Avg	1	2	3	Avg
5	3	0.00204	0.00925	0.00664	**0.00598**	0.00209	0.00957	0.00740	**0.00635**
10	3	0.00224	0.01832	0.00172	**0.00743**	0.00423	0.01833	0.00201	**0.00819**
15	3	0.00060	0.00157	0.01158	**0.00458**	0.00025	0.00413	0.01205	**0.00548**

(opt) means that the prediction was made from the mode of the posterior which was found by non-linear optimisation. For the ($\max \eta$) columns the prediction simply the mean of the kernel with the largest mixing coefficient.

B.1.2 Results for *MDN(RBF, single-reg)*

Network		RMS (opt) for network				RMS ($\max \eta$) for network			
Cen	Kern	1	2	3	Avg	1	2	3	Avg
5	10	0.00972	0.00980	0.01016	**0.00989**	0.00539	0.00558	0.00584	**0.00560**
10	10	0.02602	0.02562	0.02627	**0.02597**	0.02211	0.02149	0.02245	**0.02202**
15	10	0.00639	0.02603	0.00585	**0.01276**	0.00214	0.02227	0.00192	**0.00877**
5	30	0.00081	0.00122	0.00101	**0.00101**	0.00113	0.00157	0.00145	**0.00138**
10	30	0.00055	0.00078	0.00086	**0.00073**	0.00068	0.00090	0.00129	**0.00096**
15	30	0.00234	0.00127	0.00251	**0.00204**	0.00293	0.00202	0.00334	**0.00277**
5	50	0.00444	0.00306	0.00587	**0.00446**	0.00492	0.00346	0.00527	**0.00455**
10	50	0.00311	0.00273	0.00432	**0.00339**	0.00316	0.00268	0.00450	**0.00344**
15	50	0.00303	0.00313	0.00610	**0.00409**	0.00316	0.00309	0.00630	**0.00418**

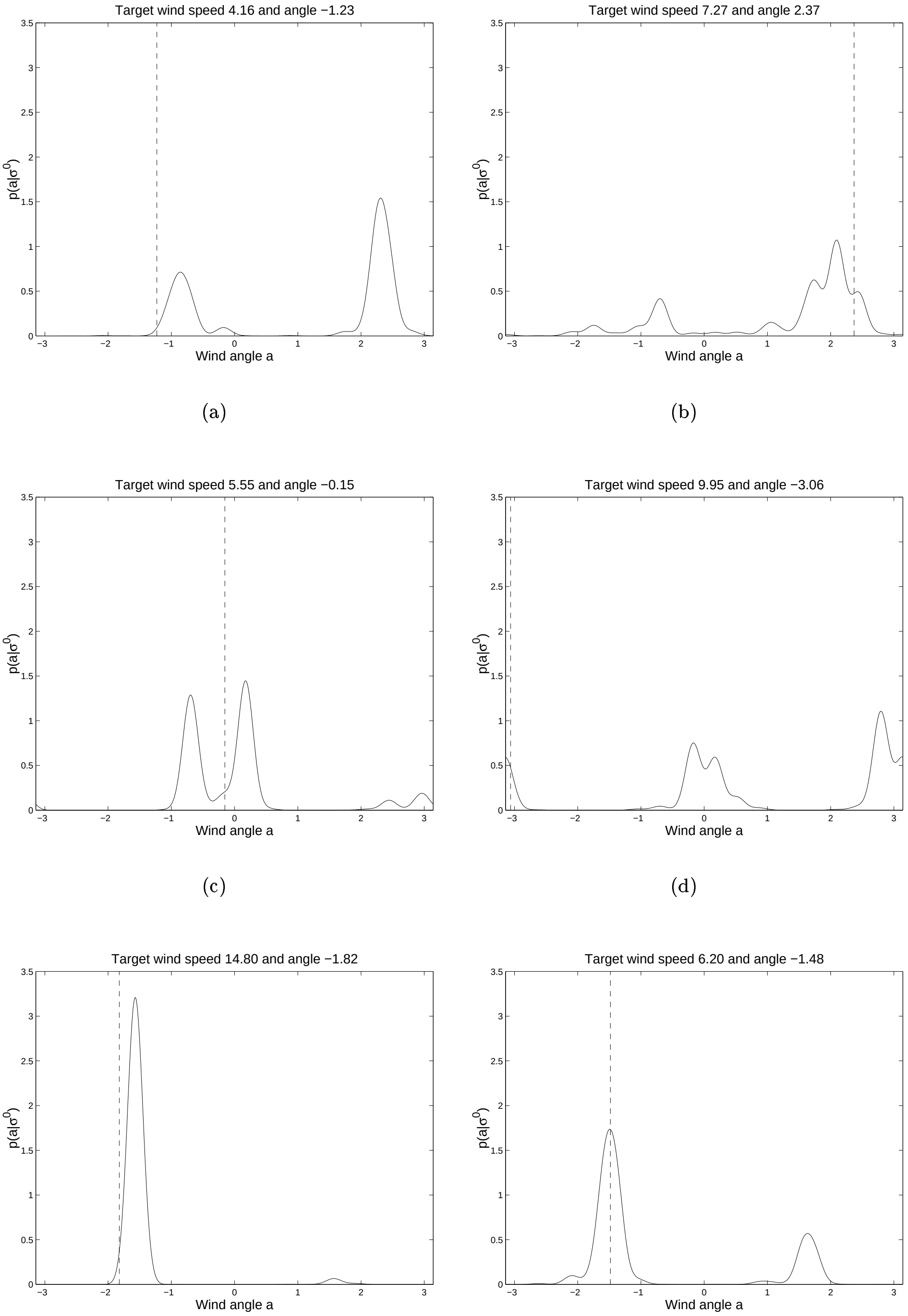
B.1.3 Results for *MDN(RBF, multi-reg)*

Network		RMS (opt) for network				RMS ($\max \eta$) for network			
Cen	Kern	1	2	3	Avg	1	2	3	Avg
5	10	0.02363	0.00251	0.00264	**0.00959**	0.02047	0.00075	0.00487	**0.00870**
10	10	0.02029	0.02818	0.02463	**0.02437**	0.01602	0.02586	0.02077	**0.02088**
15	10	0.00348	0.01768	0.01717	**0.01277**	0.00051	0.01414	0.01516	**0.00994**
5	30	0.00028	0.00165	0.00196	**0.00130**	0.00056	0.00160	0.00233	**0.00150**
10	30	0.00010	0.00393	0.00048	**0.00150**	0.00011	0.00417	0.00061	**0.00163**
15	30	0.00393	0.00171	0.00229	**0.00264**	0.00406	0.00132	0.00238	**0.00259**
5	50	0.00704	0.00756	0.00157	**0.00539**	0.00619	0.00695	0.00161	**0.00492**
10	50	0.00286	0.00351	0.00251	**0.00296**	0.00284	0.00353	0.00263	**0.00300**
15	50	0.00525	0.00076	0.00441	**0.00348**	0.00496	0.00125	0.00432	**0.00351**

B.2 The Wind Data Results

Examples of the posterior distribution

From the test set 6 different patterns were selected and their posterior distribution calculated for networks, from all categories, with 25 hidden units and 18 or 36 kernels (not 18 for the unregularised network). The posterior distributions are in figure 23, 24, 25, 26 and 27.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 23: Examples of the the posterior distribution for six random patterns from the test set for a *MDN(RBF, unreg)* with 25 hidden units and 36 kernels.

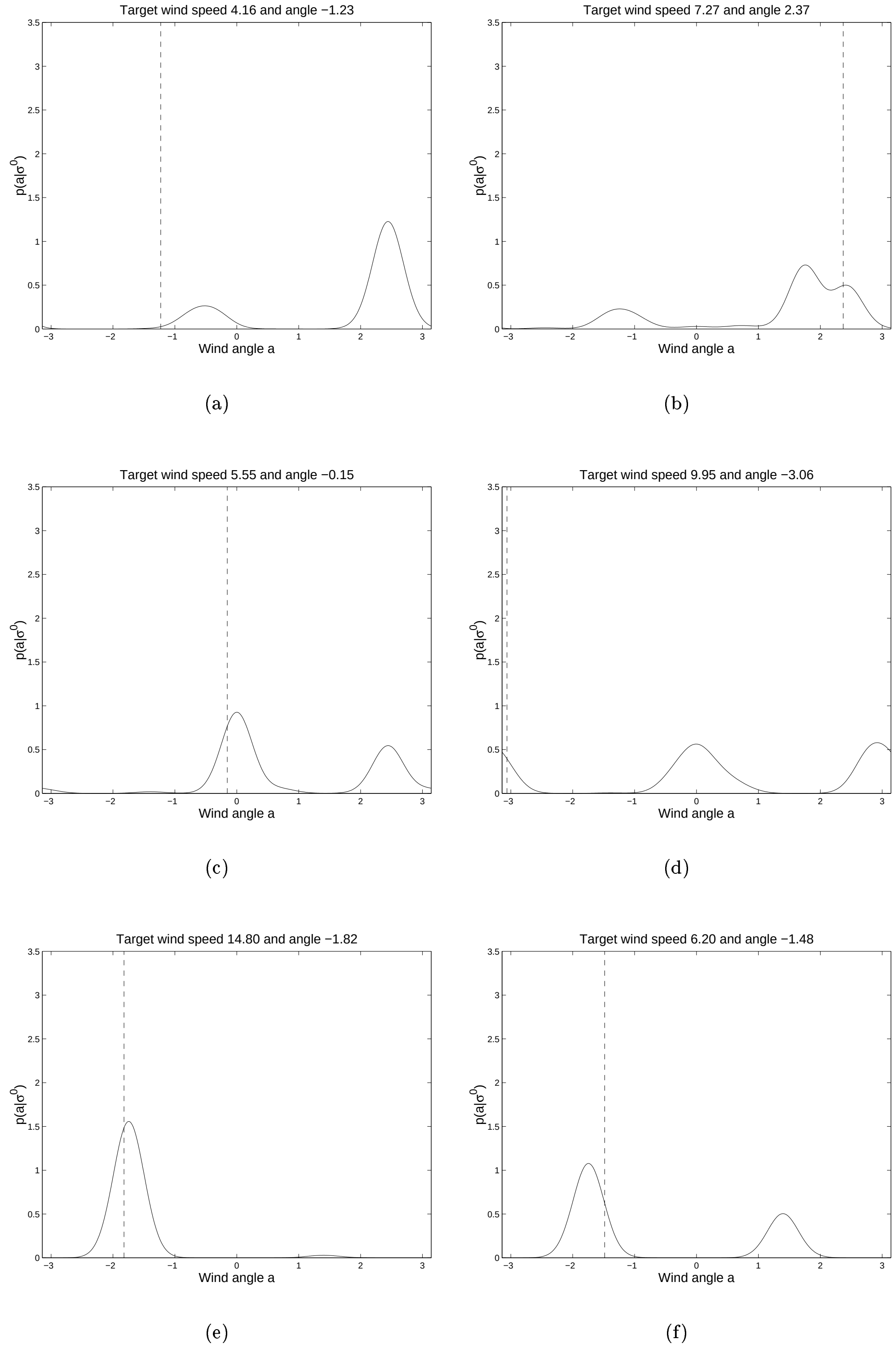
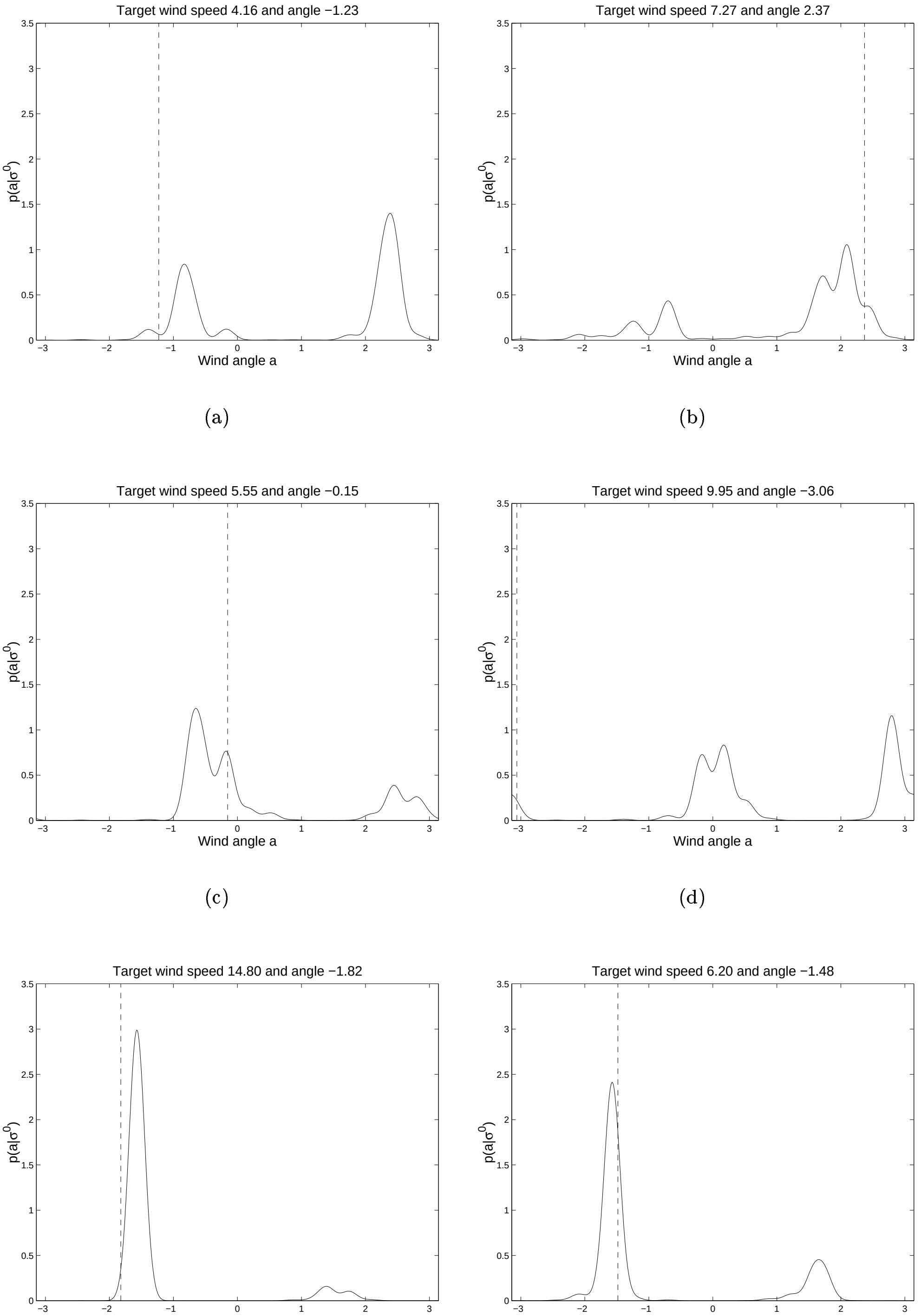


Figure 24: Examples of the the posterior distribution for six random patterns from the test set for a *MDN(RBF, noHess-reg)* with 25 hidden units and 18 kernels.



Target wind speed 4.16 and angle −1.23

(a)

Target wind speed 7.27 and angle 2.37

(b)

Target wind speed 5.55 and angle −0.15

(c)

Target wind speed 9.95 and angle −3.06

(d)

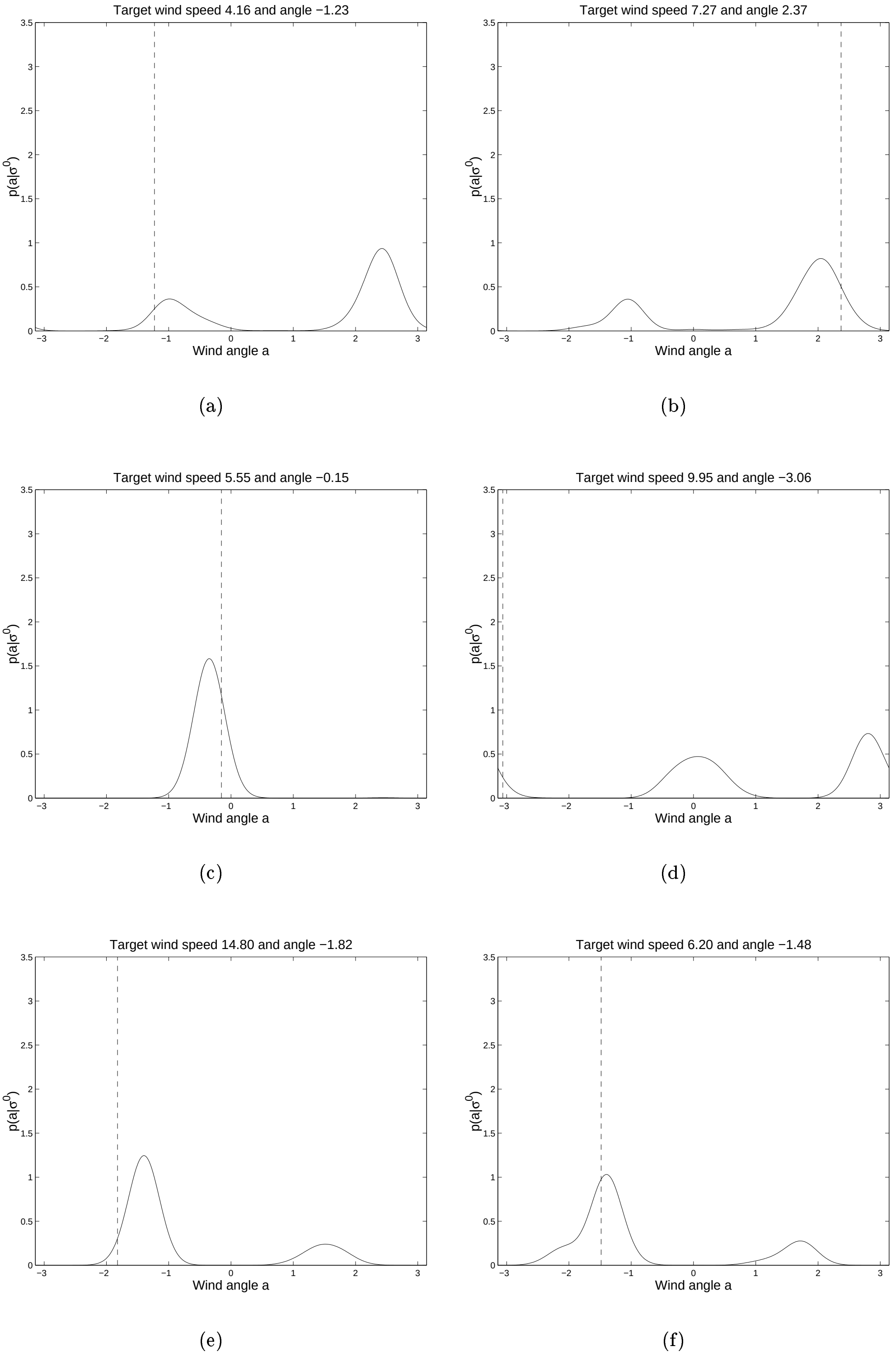
Target wind speed 14.80 and angle −1.82

(e)

Target wind speed 6.20 and angle −1.48

(f)

Figure 25: Examples of the the posterior distribution for six random patterns from the test set for a *MDN(RBF, noHess-reg)* with 25 hidden units and 36 kernels.



(a) (b)

(c) (d)

(e) (f)

Figure 26: Examples of the the posterior distribution for six random patterns from the test set for a *MDN(RBF, Hess-reg)* with 25 hidden units and 18 kernels.

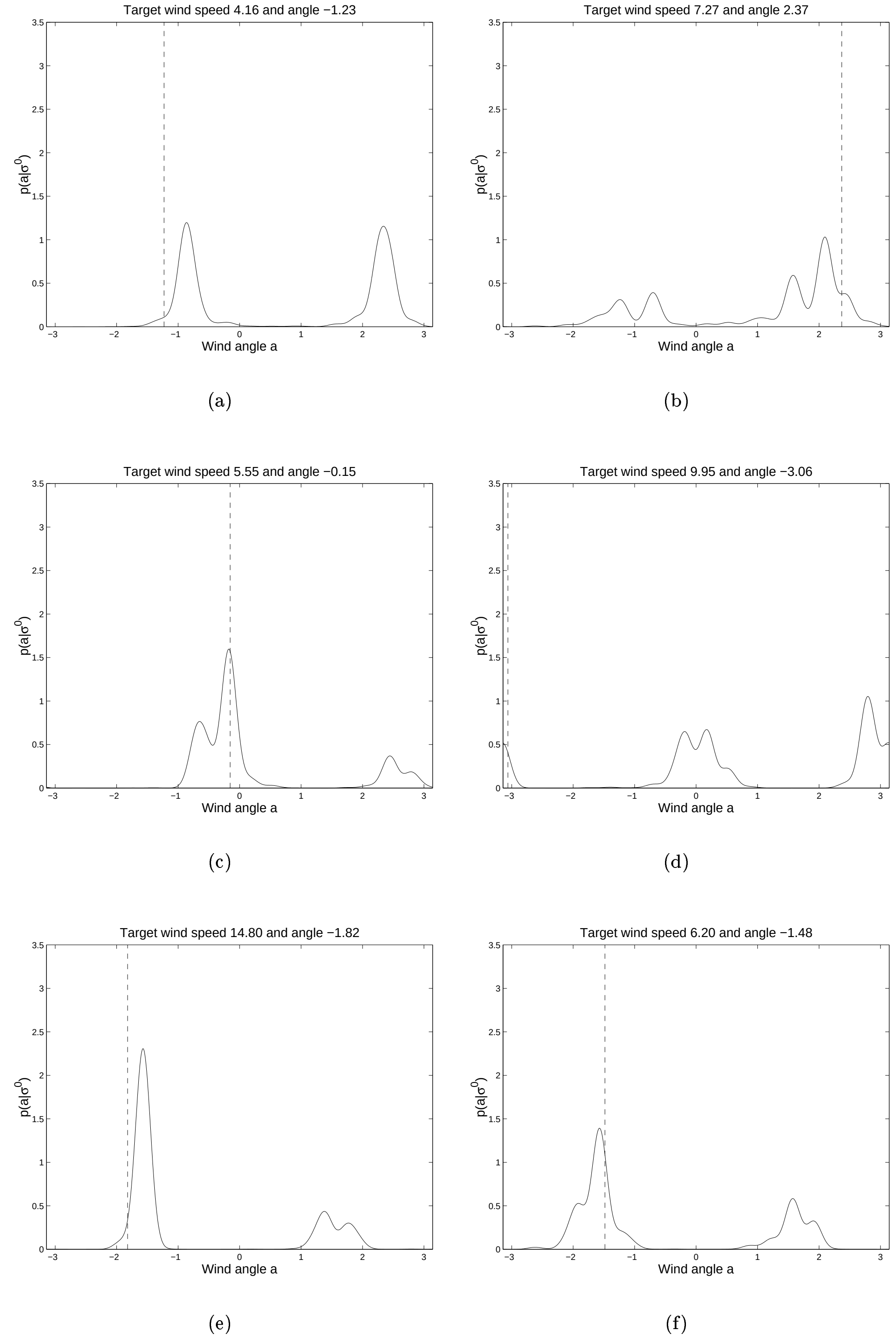


Figure 27: Examples of the the posterior distribution for six random patterns from the test set for a *MDN(RBF, Hess-reg)* with 25 hidden units and 36 kernels.