Mixture Density Networks

Christopher M. Bishop

Neural Computing Research Group
Dept. of Computer Science and Applied Mathematics
Aston University
Birmingham. B4 7ET, U.K.
C.M.Bishop@aston.ac.uk

February, 1994

Neural Computing Research Group Report: NCRG/94/004[1]
Available from: http://www.ncrg.aston.ac.uk/

Abstract

Minimization of a sum-of-squares or cross-entropy error function leads to network outputs which approximate the conditional averages of the target data, conditioned on the input vector. For classifications problems, with a suitably chosen target coding scheme, these averages represent the posterior probabilities of class membership, and so can be regarded as optimal. For problems involving the prediction of continuous variables, however, the conditional averages provide only a very limited description of the properties of the target variables. This is particularly true for problems in which the mapping to be learned is multi-valued, as often arises in the solution of inverse problems, since the average of several correct target values is not necessarily itself a correct value. In order to obtain a complete description of the data, for the purposes of predicting the outputs corresponding to new input vectors, we must model the conditional probability distribution of the target data, again conditioned on the input vector. In this paper we introduce a new class of network models obtained by combining a conventional neural network with a mixture density model. The complete system is called a Mixture Density Network, and can in principle represent arbitrary conditional probability distributions in the same way that a conventional neural network can represent arbitrary functions. We demonstrate the effectiveness of Mixture Density Networks using both a toy problem and a problem involving robot inverse kinematics.

[1]Previously issued as NCRG/94/4288

1 Introduction

Neural network models are widely used in applications involving associative mappings in which the aim is to learn a transformation from a set of input variables $\mathbf{x} \equiv \{x_1, \ldots, x_d\}$ to a set of output variables $\mathbf{t} \equiv \{t_1, \ldots, t_c\}$. In practice, such networks are trained using a finite set of examples, which we denote by $\{\mathbf{x}^q, \mathbf{t}^q\}$, where $q = 1, \ldots, n$, and q labels the particular training pattern. The central goal in network training is not to memorize the training data, but rather to model the *underlying generator* of the data, so that the best possible predictions for the output vector $\mathbf{t}$ can be made when the trained network is subsequently presented with a new value for $\mathbf{x}$. The most general and complete description of the generator of the data is a statistical one (White, 1989), and can be expressed in terms of the probability density $p(\mathbf{x}, \mathbf{t})$ in the joint input-target space. This density function specifies that the probability of a data point $(\mathbf{x}, \mathbf{t})$ falling in a small region $(\Delta\mathbf{x}, \Delta\mathbf{t})$ is given by $p(\mathbf{x}, \mathbf{t})\Delta\mathbf{x}\Delta\mathbf{t}$, and is normalized to give unit probability for the data point to lie somewhere in the input-target space: $\int p(\mathbf{x}, \mathbf{t})\, d\mathbf{x}\, d\mathbf{t} = 1$. Note that, if the generator of the data itself evolves with time, then we must consider time as an additional variable in the joint probability density. In this paper we shall limit our attention to static distributions, although the models which we shall introduce can be extended to non-stationary problems, provided the models are treated as continuously adaptive.

For associative mapping problems of the kind we are considering, it is convenient to decompose the joint probability density into the product of the conditional density of the target data, conditioned on the input data, and the unconditional density of input data

$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{t} \mid \mathbf{x})p(\mathbf{x}) \tag{1}$$

where $p(\mathbf{t} \mid \mathbf{x})$ denotes the probability density of $\mathbf{t}$ *given* that $\mathbf{x}$ takes a particular value. The density $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{t})\, d\mathbf{t}$ of input data plays an important role in validating the predictions of trained networks (Bishop, 1994). However, for the purposes of making predictions of $\mathbf{t}$ for new values of $\mathbf{x}$, it is the conditional density $p(\mathbf{t} \mid \mathbf{x})$ which we need to model.

As we shall see in the next section, the conventional neural network technique of minimizing a sum-of-squares error leads to network functions which approximate the conditional average of the target data. A similar result holds for the cross-entropy error function. For classification problems in which the target variables have a 1-of-N coding scheme, these conditional averages represent the posterior probabilities of class membership, and so can be regarded as providing an optimal solution. For problems involving the prediction of continuous variables, however, the conditional average represents a very limited description of the statistical properties of the target data, and for many applications will be wholly inadequate. Such applications include the important class of *inverse* problems, for which the target data is frequently multi-valued. In such cases the conventional least-squares approach can give completely erroneous results, as we shall show.

In this paper we introduce a new class of neural network models, called Mixture Density Networks (MDNs), which overcome these limitations and which provide a completely general framework for modelling conditional density functions. These networks, obtained by combining a conventional neural network with a mixture density model, contain the conventional least-squares approach as a special case. Indeed, their implementation in software represents a straightforward modification of standard neural network models. We demonstrate the effectiveness of MDNs using both a toy problem in two variables, and a problem involving robot inverse kinematics.

2 Conventional Least Squares

The usual approach to network training involves the minimization of a sum-of-squares error, defined over a set of training data, of the form

$$E^{\mathrm{S}}(\mathbf{w}) = \frac{1}{2} \sum_{q=1}^{n} \sum_{k=1}^{c} [f_k(\mathbf{x}^q; \mathbf{w}) - t_k^q]^2 \tag{2}$$

where t_k represent the components of the target vector, and $f_k(\mathbf{x}; \mathbf{w})$ denote the corresponding outputs of the network mapping function, which is parametrized by an array $\mathbf{w}$ adaptive parameters (the weights and biases).

We begin by considering the nature of the solutions found by least-squares techniques. In a practical application we must deal with a finite data set. This in turn means that we must limit the complexity of the network model (for instance by limiting the number of hidden units or by introducing regularization terms) in order to control the balance between bias and variance (Geman *et al.*, 1992; Bishop, 1995). The use of successively larger data sets allows the models to be more flexible (i.e. have less bias) without over-fitting the data (i.e. without leading to increased variance). In the limit of an infinite data set both bias and variance can be reduced to zero, and the optimal least squares solution is obtained. In this limit we can replace the finite sum over data points in the sum-of-squares error function (2) by an integral over the joint probability density

$$E^{\mathrm{S}} = \lim_{n \to \infty} \frac{1}{2n} \sum_{q=1}^{n} \sum_{k=1}^{c} [f_k(\mathbf{x}^q, \mathbf{w}) - t_k^q]^2 \tag{3}$$

$$= \frac{1}{2} \sum_{k=1}^{c} \iint [f_k(\mathbf{x}, \mathbf{w}) - t_k]^2 \, p(\mathbf{t}, \mathbf{x}) \, d\mathbf{t} \, d\mathbf{x} \tag{4}$$

where an extra overall factor of $1/n$ has been introduced into (3) for convenience. Since the corresponding network model $f_k(\mathbf{x}, \mathbf{w})$ is permitted to be very flexible, we can formally minimize the error function by functional differentiation with respect to $f_k(\mathbf{x}, \mathbf{w})$

$$\frac{\delta E^{\mathrm{S}}}{\delta f_k(\mathbf{x}, \mathbf{w})} = 0 \tag{5}$$

Substituting (4) into (5), and using (1), we obtain the following expression for the minimizing function

$$f_k(\mathbf{x}, \mathbf{w}^*) = \langle t_k \mid \mathbf{x} \rangle \tag{6}$$

where $\mathbf{w}^*$ represents the corresponding set of weight values. We have defined the conditional average of a quantity $Q(\mathbf{t})$ by

$$\langle Q \mid \mathbf{x} \rangle \equiv \int Q(\mathbf{t}) \, p(\mathbf{t} \mid \mathbf{x}) \, d\mathbf{t} \tag{7}$$

Thus, the network function is given by the conditional average of the target data, conditioned on the input vector. This key result is illustrated in Figure 1, and has many important implications for practical applications of neural networks (Bishop, 1995).

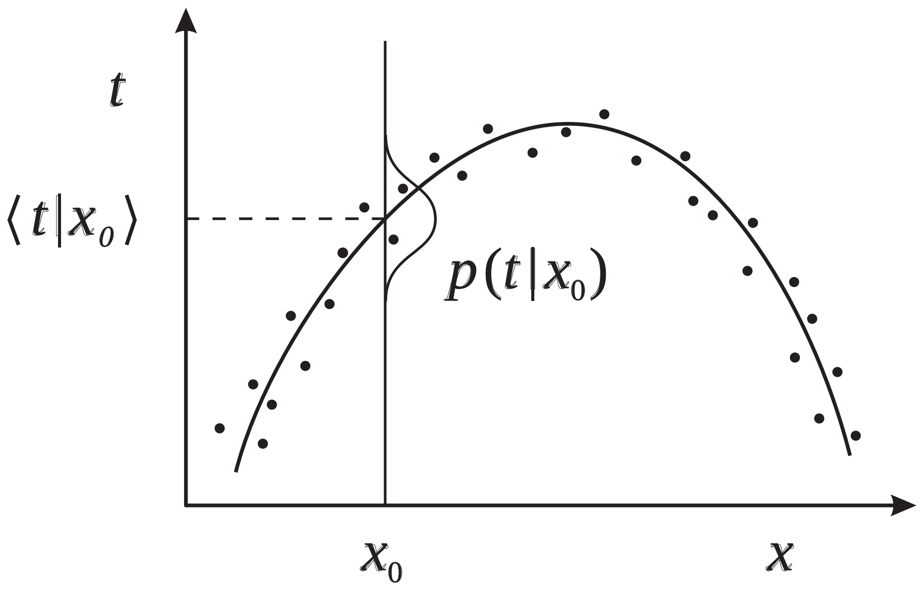

Figure 1: A schematic illustration of a set of data points (black dots) consisting of values of the input variable x and corresponding target variable t. Also shown is the optimal least-squares function (solid curve), which is given by the conditional average of the target data. Thus, for a given value of x, such as the value x_0, the least-squares function $\langle t \mid x\rangle$ is given by the average of t with respect to the probability density $p(t \mid x)$ at the given value of x.

We can also derive this result by rewriting the sum-of-squares error in a different form. By adding and subtracting $\langle t_k \mid \mathbf{x}\rangle$ inside the square brackets in (4), and again using (1), we obtain

$$E^{\mathrm{S}} = \frac{1}{2}\sum_{k=1}^{c}\int [f_k(\mathbf{x},\mathbf{w}) - \langle t_k \mid \mathbf{x}\rangle]^2\, p(\mathbf{x})\, d\mathbf{x} \tag{8}$$

$$+\frac{1}{2}\sum_{k=1}^{c}\int [\langle t_k^2 \mid \mathbf{x}\rangle - \langle t_k \mid \mathbf{x}\rangle^2]\, p(\mathbf{x})\, d\mathbf{x} \tag{9}$$

We note that the error function only depends on the network weights through the first term (8), whose minimum occurs when (6) is satisfied. Thus, we again see the result that the global minimum of the error function is given by the conditional average of the target data. The second term (9) gives the residual value of the error function at its global minimum, and is seen to correspond to the average variance of the target data around its conditional average value.

Thus, the result of training a standard neural network by least squares is that we have approximated two statistics of the target data, namely the conditional average as a function of the input vector $\mathbf{x}$, given by $f_k(\mathbf{x};\mathbf{w}^*)$, and the average variance of the data around this conditional average, given by the residual value of the error function at its minimum. If we know these two statistics, then we can represent the conditional distribution of the target data by a Gaussian function having a centre (which depends on $\mathbf{x}$) given by $f_k(\mathbf{x};\mathbf{w}^*)$ and a global variance parameter determined by the residual error. The use of a least squares approach does not assume or require the distribution of the target data to be Gaussian, but neither can it distinguish between a Gaussian distribution and any other distribution having the same (conditional) mean and (global) variance.

Conversely, if we assume that the conditional distribution of the target data is indeed Gaussian, then we can obtain the least-squares formalism using maximum likelihood, as follows. We assume that the target data is governed by the following conditional probability density function

$$p(t_k \mid \mathbf{x}) = \frac{1}{(2\pi)^{1/2}\sigma} \exp\left\{ -\frac{[F_k(\mathbf{x}) - t_k]^2}{2\sigma^2} \right\} \tag{10}$$

where σ is a global variance parameter, and where the output variables are treated as independently distributed. Here $F_k(\mathbf{x})$ is the mean of the target variable t_k and is taken to be a general function of $\mathbf{x}$. The conditional density of the complete target vector is then given by

$$p(\mathbf{t} \mid \mathbf{x}) = \prod_{k=1}^{c} p(t_k \mid \mathbf{x}) = \frac{1}{(2\pi)^{c/2}\sigma^c} \exp\left\{ -\frac{1}{2\sigma^2} \sum_{k=1}^{c} [F_k(\mathbf{x}) - t_k]^2 \right\} \tag{11}$$

The underlying generator function $F_k(\mathbf{x})$ is unknown, and is the basic quantity we seek to determine, since knowledge of $F_k(\mathbf{x})$, together with the value of the unknown parameter σ, gives us a complete description of the data generation process (within the framework of the Gaussian assumption). We therefore model $F_k(\mathbf{x})$ by a parametrized functional form $f_k(\mathbf{x};\mathbf{w})$. Feed-forward neural networks offer a particularly powerful choice for the parametrized function since they can efficiently represent non-linear multivariate functions with in principle no serious limitations on the functional forms which they can approximate (Hornik *et al.*, 1989).

The values for the parameters $\mathbf{w}$ in $f_k(\mathbf{x};\mathbf{w})$ must be determined from the finite set of training examples $\{\mathbf{x}^q, \mathbf{t}^q\}$. This can be achieved by maximizing the likelihood that the model gave rise to the particular set of data points. If we assume that the training data are drawn independently from the distribution given by (11), then the likelihood of the data set is given by the product of the likelihoods for each of the data points

$$\mathcal{L} = \prod_{q=1}^{n} p(\mathbf{t}^q, \mathbf{x}^q) = \prod_{q=1}^{n} p(\mathbf{t}^q \mid \mathbf{x}^q) p(\mathbf{x}^q) \tag{12}$$

where we have used (1). The quantity $\mathcal{L}$ is a function of the parameters $\mathbf{w}$, and we can determine appropriate values for $\mathbf{w}$ by maximization of $\mathcal{L}$. In practice, it is convenient instead to minimize the negative logarithm of $\mathcal{L}$, which is usually called an error function

$$E = -\ln \mathcal{L} \tag{13}$$

Minimizing E is equivalent to maximizing $\mathcal{L}$ since the negative logarithm is a monotonic function. Using (11), (12) and (13), and modelling $F_k(\mathbf{x})$ by $f_k(\mathbf{x};\mathbf{w})$, we can write E in the form

$$E = nc \ln \sigma + \frac{nc}{2} \ln(2\pi) + \frac{1}{2\sigma^2} \sum_{q=1}^{n} \sum_{k=1}^{c} [f_k(\mathbf{x}^q;\mathbf{w}) - t_k^q]^2 + \sum_{q=1}^{n} \ln p(\mathbf{x}^q) \tag{14}$$

Note that only the third term in (14) depends on the parameters $\mathbf{w}$ and so their values can be determined by minimizing only this term. In addition, the factor $1/\sigma^2$ can be omitted since it has no effect on the minimization with respect to $\mathbf{w}$. This gives rise to the standard sum-of-squares error function commonly used in neural network training

$$E^{\mathrm{S}} = \frac{1}{2} \sum_{q=1}^{n} \sum_{k=1}^{c} [f_k(\mathbf{x}^q; \mathbf{w}) - t_k^q]^2 \tag{15}$$

where the pre-factor of $1/2$ has been retained for convenience when computing derivatives of E^{S}. In general, $f_k(\mathbf{x}; \mathbf{w})$ will be a non-linear function of the parameters $\mathbf{w}$, as would be the case for a multi-layer perceptron network for instance. Thus, the minimization of E^{S} represents a problem in non-linear optimization, for which there exists a range of standard techniques (Press *et al.*, 1992; Bishop, 1995).

Having found values for the parameters $\mathbf{w}^*$, the optimum value for σ can then by found by minimization of E in (14) with respect to σ. This minimization is easily performed analytically with the explicit, and intuitive, result

$$\sigma^2 = \frac{1}{nc} \sum_{q=1}^{n} \sum_{k=1}^{c} [f_k(\mathbf{x}^q; \mathbf{w}^*) - t_k^q]^2 \tag{16}$$

which says that the optimum value of σ^2 is given by the residual value of the sum-of-squares error function at its minimum, as shown earlier.

For classification problems the target values are generally chosen to have a 1-of-N coding scheme whereby $t_k^q = \delta_{kl}$ for an input pattern $\mathbf{x}^q$ belonging to class $\mathcal{C}_l$. The probability distribution of target values is then given by

$$p(t_k \mid \mathbf{x}) = \sum_{l=1}^{c} \delta(t_k - \delta_{kl}) P(\mathcal{C}_l \mid \mathbf{x}) \tag{17}$$

where $P(\mathcal{C}_l \mid \mathbf{x})$ is the probability that $\mathbf{x}$ belongs to class $\mathcal{C}_l$. Substituting (17) into (6) then gives

$$f_k(\mathbf{x}, \mathbf{w}^*) = P(\mathcal{C}_k \mid \mathbf{x}) \tag{18}$$

and so the network outputs represent the Bayesian posterior probabilities of membership of the corresponding classes. In this sense the network outputs can be regarded as optimal since the posterior probabilities allow minimum risk classifications to be made, once the appropriate loss matrix has been specified. For instance, if the goal is to minimize the number of misclassifications, corresponding to loss matrix given by the unit matrix, then each new input should be assigned to the class having the largest posterior probability.

It should be noted that the cross-entropy error function also leads to network outputs which approximate the conditional average of the target data. For a finite data set, the cross-entropy error function can be written as

$$E^{\Omega} = -\sum_{q=1}^{n} \sum_{k=1}^{c} \{t_k^q \ln f_k(\mathbf{x}^q; \mathbf{w}) + (1 - t_k^q) \ln(1 - f_k(\mathbf{x}^q; \mathbf{w}))\} \tag{19}$$

In the infinite-data limit this can be written as

$$E^{\Omega} = -\sum_{k=1}^{c} \iint \{t_k \ln f_k(\mathbf{x}; \mathbf{w}) + (1 - t_k) \ln(1 - f_k(\mathbf{x}; \mathbf{w}))\}\, p(\mathbf{t}, \mathbf{x})\, d\mathbf{t}\, d\mathbf{x} \tag{20}$$

By functional differentiation as before, and making use of (1) and (7), we obtain

$$f_k(\mathbf{x}, \mathbf{w}^*) = \langle t_k \mid \mathbf{x} \rangle \tag{21}$$

so that the network outputs again represent the conditional averages of the target data.

Most conventional applications of neural networks only make use of the prediction for the mean, given by $f_k(\mathbf{x}; \mathbf{w}^*)$, which approximates the conditional average of the target data, conditioned on the input vector. We have seen that, for classification problems, this represents the optimal solution. However, for problems involving the prediction of continuous variables, the conditional average represents only a very limited statistic. For many applications, there is considerable benefit in obtaining a much more complete description of the probability distribution of the target data. We therefore introduce the Mixture Density Network which can in principle represent arbitrary conditional distributions, in the same way that a conventional neural network can represent arbitrary non-linear functions.

3 Mixture Density Networks

As we have already seen, the conventional least-squares technique can be derived from maximum likelihood on the assumption of Gaussian distributed data. This motivates the idea of replacing the Gaussian distribution in (11) with a *mixture model* (McLachlan and Basford, 1988), which has the flexibility to model completely general distribution functions. The probability density of the target data is then represented as a linear combination of kernel functions in the form

$$p(\mathbf{t} \mid \mathbf{x}) = \sum_{i=1}^{m} \alpha_i(\mathbf{x}) \phi_i(\mathbf{t} \mid \mathbf{x}) \tag{22}$$

where m is the number of components in the mixture. The parameters $\alpha_i(\mathbf{x})$ are called *mixing coefficients*, and can be regarded as *prior* probabilities (conditioned on $\mathbf{x}$) of the target vector $\mathbf{t}$ having been generated from the i^{th} component of the mixture. Note that the mixing coefficients are taken to be functions of the input vector $\mathbf{x}$. The functions $\phi_i(\mathbf{t} \mid \mathbf{x})$ represent the conditional density of the target vector $\mathbf{t}$ for the i^{th} kernel. Various choices for the kernel functions are possible. For the purposes of this paper, however, we shall restrict attention to kernel functions which are Gaussian of the form

$$\phi_i(\mathbf{t} \mid \mathbf{x}) = \frac{1}{(2\pi)^{c/2} \sigma_i(\mathbf{x})^c} \exp \left\{ - \frac{\| \mathbf{t} - \boldsymbol{\mu}_i(\mathbf{x}) \|^2}{2 \sigma_i(\mathbf{x})^2} \right\} \tag{23}$$

where the vector $\boldsymbol{\mu}_i(\mathbf{x})$ represents the centre of the i^{th} kernel, with components μ_{ik}. In (23) we have assumed that the components of the output vector are statistically independent within each component of the distribution, and can be described by a common variance $\sigma_i(\mathbf{x})$. This assumption can be relaxed in a straightforward way by introducing full covariance matrices for each Gaussian kernel, at the expense of a more complex formalism. In principle, however, such a complication is not necessary, since a Gaussian mixture model, with kernels given by (23), can approximate any given density function to arbitrary accuracy, provided the mixing coefficients and the Gaussian parameters (means and variances) are correctly chosen (McLachlan and Basford, 1988). Thus, the representation given by (22) and (23) is completely general. In particular, it does not assume that the components of $\mathbf{t}$ are statistically independent, in contrast to the single-Gaussian representation in (11).

For any given value of $\mathbf{x}$, the mixture model (22) provides a general formalism for modelling an arbitrary conditional density function $p(\mathbf{t} \mid \mathbf{x})$. We now take the various parameters of the mixture model, namely the mixing coefficients $\alpha_i(\mathbf{x})$, the means $\boldsymbol{\mu}_i(\mathbf{x})$ and the variances $\sigma_i(\mathbf{x})$, to be general (continuous) functions of $\mathbf{x}$. This is achieved by modelling them using the outputs of a conventional neural network which takes $\mathbf{x}$ as its input. The combined structure of a feed-forward network and a mixture model we refer to as a *Mixture Density Network* (MDN), and its basic structure is indicated in Figure 2. By choosing a mixture model with a sufficient number of kernel functions, and a neural network with a sufficient number of hidden units, the MDN can approximate as closely as desired any conditional density $p(\mathbf{t} \mid \mathbf{x})$.

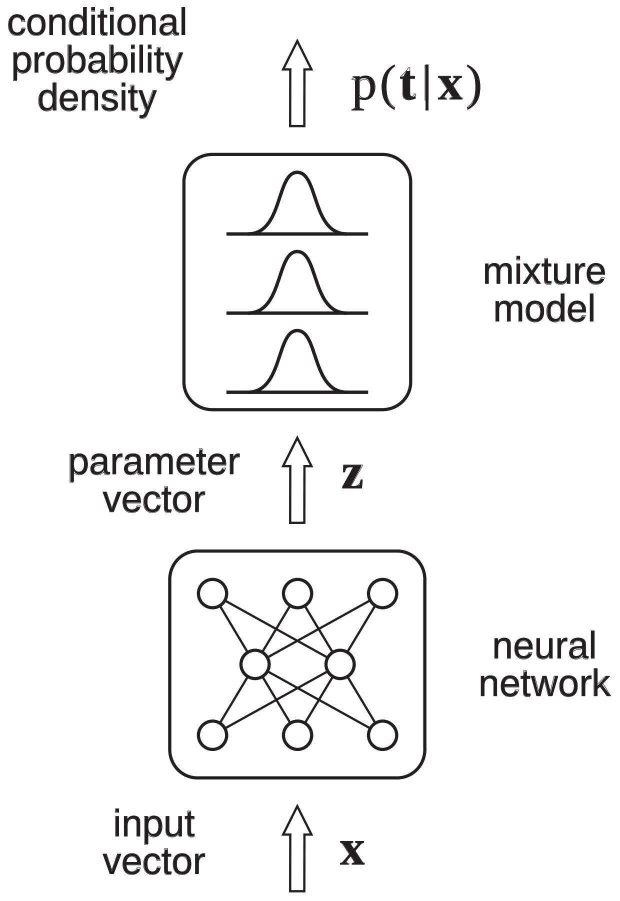

Figure 2: The Mixture Density Network consists of a feed-forward neural network whose outputs determine the parameters in a mixture density model. The mixture model then represents the conditional probability density function of the target variables, conditioned on the input vector to the neural network.

The neural network element of the MDN can be any standard feed-forward structure with universal approximation capabilities. In this paper we consider a standard multi-layer perceptron, with a single hidden layer of sigmoidal units and an output layer of linear units, and we shall use z_j to denote the output variables. Note that the total number of network outputs is given by $(c + 2) \times m$, as compared with the usual c outputs for a network used in the conventional manner.

It is important to note that the mixing coefficients $\alpha_i(\mathbf{x})$ must satisfy the constraint

$$\sum_{i=1}^{m} \alpha_i(\mathbf{x}) = 1 \tag{24}$$

This is achieved by choosing $\alpha_i(\mathbf{x})$ to be related to the networks outputs by a 'softmax' function (Bridle, 1990; Jacobs *et al.*, 1991)

$$\alpha_i = \frac{\exp(z_i^{\alpha})}{\sum_{j=1}^{M} \exp(z_j^{\alpha})} \tag{25}$$

where z_i^{α} represent the corresponding network outputs. This can be regarded as a generalization of the usual logistic sigmoid, and ensures that the quantities α_i lie in the range $(0, 1)$ and sum to unity, as required for probabilities.

The variances σ_i represent *scale* parameters and so it is convenient to represent them in terms of the exponentials of the corresponding network outputs

$$\sigma_i = \exp(z_i^{\sigma}) \tag{26}$$

which, in a Bayesian framework, would correspond to the choice of an un-informative Bayesian prior, assuming the corresponding network outputs z_i^{σ} had uniform probability distributions (Jacobs *et al.*, 1991; Nowlan and Hinton, 1992). This representation also has the additional benefit of avoiding pathological configurations in which one or more of the variances goes to zero. The centers $\boldsymbol{\mu}_i$ represent *location* parameters, and the notion of an un-informative prior suggests that these be represented directly by the network outputs

$$\mu_{ik} = z_{ik}^{\mu} \tag{27}$$

As before, we can construct a likelihood function using (12), and then define an error function by taking the negative logarithm of the likelihood, as in (13), to give the error function for the Mixture Density Network in the form

$$E = \sum_q E^q \tag{28}$$

where the contribution to the error from pattern q is given by

$$E^q = -\ln \left\{ \sum_{i=1}^{m} \alpha_i(\mathbf{x}^q) \phi_i(\mathbf{t}^q \mid \mathbf{x}^q) \right\} \tag{29}$$

with $\phi_i(\mathbf{t} \mid \mathbf{x})$ given by (23). We have dropped the term $\sum_q p(\mathbf{x}^q)$ as it is independent of the parameters of the mixture model, and hence is independent of the network weights. Note that (29) is formally equivalent to the error function used in the 'competing local experts' model of Jacobs et al. (Jacobs *et al.*, 1991). The interpretation presented here, however, is quite different. Instead of seeking to impose soft competition between a number of competing simpler network modules, the aim is to model the complete conditional probability density of the output variables. From this density function, any desired statistic involving the output variables can in principle be computed.

In order to minimize the error function, we need to calculate the derivatives of the error E with respect to the weights in the neural network. These can be evaluated by using the standard 'back-propagation' procedure, provided we obtain suitable expressions for the derivatives of the error with respect to the activations of the output units of the neural network. Since the error function (28) is composed of a sum of terms, one for each pattern, we can consider the derivatives $\delta_k^q = \partial E^q / \partial z_k$ for a particular pattern q and then find the derivatives of E by summing over all patterns. The derivatives δ_k^q act as 'errors' which can be back-propagated through the network to find the derivatives with respect to the network weights. This is discussed further in Section 4. Standard optimization algorithms, such as conjugate gradients or quasi-Newton methods, can then be used to find a minimum of E. Alternatively, if an optimization algorithm such as stochastic gradient descent is to be used, the weight updates can be applied after presentation of each pattern separately.

We have already remarked that the ϕ_i can be regarded as conditional density functions, with prior probabilities α_i. It is convenient to introduce the corresponding *posterior* probabilities, which we obtain using Bayes theorem

$$\pi_i(\mathbf{x}, \mathbf{t}) = \frac{\alpha_i \phi_i}{\sum_{j=1}^{m} \alpha_j \phi_j} \tag{30}$$

as this leads to some simplification of the subsequent analysis. Note that the posterior probabilities sum to unity:

$$\sum_{i=1}^{m} \pi_i = 1 \tag{31}$$

Consider first the derivatives with respect to those network outputs which correspond to the mixing coefficients α_i. Using (29) and (30) we obtain

$$\frac{\partial E^q}{\partial \alpha_i} = -\frac{\pi_i}{\alpha_i} \tag{32}$$

We now note that, as a result of the softmax activation function (25), the value of α_i depends on all of the network outputs which contribute to the priors, and so we have

$$\frac{\partial \alpha_i}{\partial z_k^{\alpha}} = \delta_{ik}\alpha_i - \alpha_i \alpha_k \tag{33}$$

From the chain rule we have

$$\frac{\partial E^q}{\partial z_k^{\alpha}} = \sum_i \frac{\partial E^q}{\partial \alpha_i} \frac{\partial \alpha_i}{\partial z_k^{\alpha}} \tag{34}$$

Combining (32), (33) and (34) we then obtain

$$\frac{\partial E^q}{\partial z_k^{\alpha}} = \alpha_k - \pi_k \tag{35}$$

where we have used (31).

For the derivatives corresponding to the σ_i parameters we make use of (29) and (30), together with (23), to give

$$\frac{\partial E^q}{\partial \sigma_i} = -\pi_i \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_i\|^2}{\sigma_i^3} - \frac{c}{\sigma_i} \right\} \tag{36}$$

Using (26) we have

$$\frac{\partial \sigma_i}{\partial z_i^\sigma} = \sigma_i \tag{37}$$

Combining these together we then obtain

$$\frac{\partial E^q}{\partial z_i^\sigma} = -\pi_i \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_i\|^2}{\sigma_i^2} - c \right\} \tag{38}$$

Finally, since the parameters μ_{ik} are given directly by the z_{ik}^μ network outputs, we have, using (29) and (30), together with (23),

$$\frac{\partial E^q}{\partial z_{ik}^\mu} = \pi_i \left\{ \frac{(\mu_{ik} - t_k)}{\sigma_i^2} \right\} \tag{39}$$

The derivatives of the error function with respect to the network outputs, given by (35), (38) and (39), can be used in standard optimization algorithms to find a minimum of the error. For the results presented in this paper, the optimization of the network weights was performed using the BFGS quasi-Newton algorithm (Press *et al.*, 1992).

In the previous section we considered the properties of the standard least-squares network model in the limit of an infinite data set. We now perform the corresponding analysis for the Mixture Density Network. Taking the infinite data set limit of (28) and (29), we can write the error function in the form

$$E = -\int\!\!\int \ln \left\{ \sum_{i=1}^{m} \alpha_i(\mathbf{x}) \phi_i(\mathbf{t} \mid \mathbf{x}) \right\} p(\mathbf{x}, \mathbf{t}) \, d\mathbf{x} \, d\mathbf{t} \tag{40}$$

If we set the functional derivatives of E with respect to $z_i^\alpha(\mathbf{x})$, $z_i^\sigma(\mathbf{x})$ and $z_i^\mu(\mathbf{x})$ to zero we obtain, after some algebraic rearrangement, the following conditions which are satisfied by the mixture model parameters at the minimum of the error function

$$\alpha_i(\mathbf{x}) = \langle \pi_i \mid \mathbf{x} \rangle \tag{41}$$

$$\boldsymbol{\mu}_i(\mathbf{x}) = \frac{\langle \pi_i \mathbf{t} \mid \mathbf{x} \rangle}{\langle \pi_i \mid \mathbf{x} \rangle} \tag{42}$$

$$\sigma_i^2(\mathbf{x}) = \frac{\langle \pi_i \|\boldsymbol{\mu}_i(\mathbf{x}) - \mathbf{t}\|^2 \mid \mathbf{x} \rangle\rangle}{\langle \pi_i \mid \mathbf{x} \rangle} \tag{43}$$

where $\pi_i \equiv \pi_i(\mathbf{x}, \mathbf{t})$, and where the conditional averages are defined by (7) as before. These results have a very natural interpretation. For each value of the input vector $\mathbf{x}$, (41) shows that the priors $\alpha_i(\mathbf{x})$ are given by the corresponding posterior probabilities, averaged over the conditional density of the target data. Similarly, the centers (42) are given by the conditional average of the target data, weighted by the corresponding posterior probabilities. Finally, the variance parameters (43) are given by the conditional average of the variance of the target data around the corresponding kernel centre, again weighted by the posterior probability of that kernel.

Once an MDN has been trained it can predict the conditional density function of the target data for any given value of the input vector. This conditional density represents a complete description of the generator of the data, so far as the problem of predicting the value of the target vector is concerned. From this density function we can calculate more specific quantities which may be of interest in different applications. Here we discuss some of the possibilities.

One of the simplest statistics is the mean, corresponding to the conditional average of the target data, given by

$$\begin{aligned} \langle \mathbf{t} \mid \mathbf{x} \rangle &= \sum_i \alpha_i(\mathbf{x}) \int \mathbf{t}\, \phi_i(\mathbf{t} \mid \mathbf{x})\, d\mathbf{t} && (44) \\ &= \sum_i \alpha_i(\mathbf{x}) \boldsymbol{\mu}_i(\mathbf{x}) && (45) \end{aligned}$$

where we have used (23). This is equivalent to the function computed by a standard network trained by least-squares. Thus, MDNs contain the conventional least-squares result as a special case. We can likewise evaluate the variance of the density function about the conditional average, to give

$$\begin{aligned} s^2(\mathbf{x}) &= \left\langle \|\mathbf{t} - \langle \mathbf{t} \mid \mathbf{x} \rangle\|^2 \mid \mathbf{x} \right\rangle && (46) \\ &= \sum_i \alpha_i(\mathbf{x}) \left\{ \sigma_i(\mathbf{x})^2 + \left\| \boldsymbol{\mu}_i(\mathbf{x}) - \sum_j \alpha_j(\mathbf{x}) \boldsymbol{\mu}_j(\mathbf{x}) \right\|^2 \right\} && (47) \end{aligned}$$

which is more general than the corresponding least-squares result since this variance is allowed to be a general function of $\mathbf{x}$. Similar results can be obtained for other moments of the conditional distribution.

For many problems we might be interested in finding one specific value for the output vector. The most *likely* value for the output vector, for a given input vector $\mathbf{x}$, is given by the maximum of the conditional density $p(\mathbf{t} \mid \mathbf{x})$. Since this density function is represented by a mixture model, the location of its global maximum is a problem in non-linear optimization. While standard techniques exist for solving such problems (Press *et al.*, 1992), these are iterative in nature and are therefore computationally costly. For applications where speed of processing for new data is important, we may need to find a faster, approximate, approach.

If we assume that the component kernels of the density function are not too strongly overlapping, then to a very good approximation the most likely value of $\mathbf{t}$ will be given by the centre of the highest component. From (22) and (23), we see that the component with the largest central value is given by

$$\max_i \left\{ \frac{\alpha_i(\mathbf{x})}{\sigma_i(\mathbf{x})^c} \right\} \qquad (48)$$

and the corresponding centre $\boldsymbol{\mu}_i$ represents the most likely output vector, to a good approximation. Alternatively, we may wish to consider the total 'probability mass' associated with each of the mixture components. This would be appropriate for applications involving multi-valued mappings with a finite number of distinct branches, in which we are interested in finding a representative vector corresponding to the most *probable* branch.

(This approach is also less susceptible to problems due to artificially small values of σ_i arising from regions of sparse data). An example of such a problem, involving the kinematics of robot arms, is discussed in the next section. Since each component of the mixture model is normalized, $\int \phi_i(\mathbf{t} \mid \mathbf{x})\, d\mathbf{t} = 1$, the most probable branch of the solution, assuming the components are well separated and have negligible overlap, is given by

$$\max_i \{\alpha_i(\mathbf{x})\} \tag{49}$$

The required value of $\mathbf{t}$ is then given by the corresponding centre $\boldsymbol{\mu}_i$.

A whole variety of other statistics can be computed from the conditional probability density, as appropriate to the particular application.

4 Software Implementation

The implementation of the MDN in software is very straightforward, and for large-scale problems will typically not lead to a significant computational overhead compared with the standard least-squares approach. Consider a multi-layer perceptron network trained by minimizing a sum-of-squares error function using a standard optimization procedure (such as gradient descent or quasi-Newton). The only modification to the software which is required arises from the modified definition of the error function, with all other aspects of the implementation remaining unchanged. In general, we can regard the error function as a 'module' which takes a network output vector $\mathbf{z}^q$ (for a particular pattern q) and a corresponding target vector $\mathbf{t}^q$ and which can return the value of the error E^q for that pattern, and also the derivatives $\boldsymbol{\delta}^q$ of the error with respect to the network outputs $\mathbf{z}^q$. This is illustrated in Figure 3. The derivatives of the error function with respect to one of the weights w in the network is obtained by use of the chain rule

$$\frac{\partial E^q}{\partial w} = \sum_i \frac{\partial E^q}{\partial z_i}\frac{\partial z_i}{\partial w} = \sum_i \delta_i^q \frac{\partial z_i}{\partial w} \tag{50}$$

where the quantities $\delta_i^q = \partial E^q / \partial z_i$ can be interpreted as 'errors' which are to be back-propagated through the network. For the particular case of the sum-of-squares error function we have

$$E^q = \frac{1}{2}\left|\mathbf{z}^q - \mathbf{t}^q\right|^2 \tag{51}$$

$$\boldsymbol{\delta}^q = \mathbf{z}^q - \mathbf{t}^q \tag{52}$$

In order to modify the software to implement the MDN, (51) and (52) must be replaced by the appropriate expressions. The error function for a particular pattern is given by (29), while the elements of the vector $\boldsymbol{\delta}$ are given by (35), (38) and (39). The implementation of an MDN is particularly simple and natural in an object oriented language such as C++, since the error module can be represented as a class, with methods to set the mixture parameters for a given set of network outputs, and to return the error function or its derivatives. The error function class can also be provided with methods to return the value of the conditional probability density for given values of $\mathbf{x}$ and $\mathbf{t}$, or to return

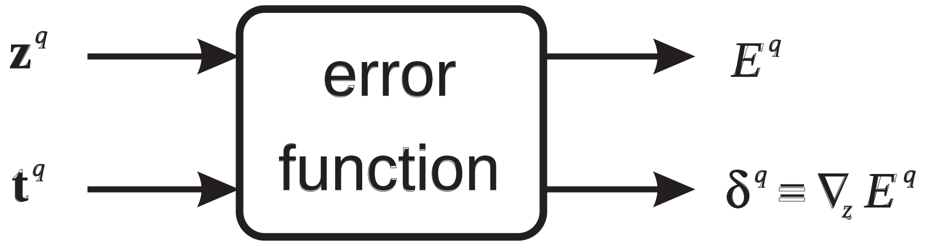

Figure 3: For the purposes of software implementation, an error function can be regarded as a module which takes a network output vector $\mathbf{z}^q$ (for a particular pattern q) and a corresponding target vector $\mathbf{t}^q$ and which can return the value of the error E^q for that pattern, as well as the derivatives of the error with respect to the network outputs, $\boldsymbol{\delta}^q = \nabla_{\mathbf{z}} E^q$.

other statistics derived from the conditional probability density (such as the centre vector corresponding to the most probable kernel).

For applications involving large numbers of input variables, the computational requirements for the MDN need not be significantly greater than with a standard network trained using a sum-of-squares error function, since much of the computational cost lies in the forward and backward propagation of signals through the network itself. For networks with a large number of inputs (and hence a large number of weights in the first layer) this will exceed the cost of evaluating the error function and its derivatives.

In any algorithm which uses gradient-based methods to perform error minimization, a very powerful check on the software can be made by comparing the error derivatives obtained from the analytic expressions with those calculated using finite differences. Close agreement between these two approaches demonstrates that a high proportion of the code has been implemented correctly. Note that substantially improved accuracy is obtained if symmetric central differences are used, rather than simple finite differences, since in the latter case we have

$$\frac{E(w+\epsilon) - E(w)}{\epsilon} = \frac{\partial E}{\partial w} + \mathcal{O}(\epsilon) \tag{53}$$

where ϵ is a small parameter, whereas central differences give

$$\frac{E(w+\epsilon) - E(w-\epsilon)}{2\epsilon} = \frac{\partial E}{\partial w} + \mathcal{O}(\epsilon^2) \tag{54}$$

for which the correction terms are much smaller. Of course, for use in error minimization, the analytic expressions should be used in preference to the finite difference formulae since, not only are they more accurate, but they are substantially more computationally efficient (Bishop, 1995).

5 A Simple Inverse Problem

Many potential applications of neural networks fall into the category of *inverse* problems. Examples include the control of industrial plant, analysis of spectral data, tomographic reconstruction, and robot kinematics. For such problems there exists a well defined *forward* problem which is characterized by a functional (i.e. single-valued) mapping. Often this corresponds to causality in a physical system. In the case of spectral reconstruction, for example, the forward problem corresponds to the prediction of the spectrum when the parameters (locations, widths and amplitudes) of the spectral lines are prescribed. For practical applications, however, we generally have to solve the corresponding inverse problem in which the roles of input and output variables are interchanged. In the case of spectral analysis, this corresponds to the determination of the spectral line parameters from an observed spectrum. For inverse problems, the mapping can be often be multi-valued, with values of the input for which there are several valid values for the output. For example, there may be several choices for the spectral line parameters which give rise to the same observed spectrum (corresponding, for example, to the exchange of width parameters for two co-located lines). If a standard neural network is applied to such inverse problems, it will approximate the conditional average of the target data, and this will frequently lead to extremely poor performance. (The average of several solutions is not necessarily itself a solution). This problem can be overcome in a natural and effective way by appropriate use of a Mixture Density Network instead.

In order to illustrate the application of the MDN, we begin by considering a simple example of an inverse problem involving a mapping between a single input variable and a single output variable. Consider the mapping from t (regarded here as an input variable) to x (regarded as an output variable) defined by

$$x = t + 0.3\sin(2\pi t) + \epsilon \qquad (55)$$

where ϵ is a random variable with uniform distribution in the interval $(-0.1, 0.1)$. The mapping from t to x provides an example of a *forward* problem. In the absence of the noise term ϵ, this mapping is single-valued, so that each value of t gives rise to a unique value of x. Figure 4 shows a data set of 1,000 points generated by sampling (55) at equal intervals of t in the range $(0.0, 1.0)$. Also shown is the mapping represented by a standard multi-layer perceptron after training using this data set. The network had 1 input, 5 hidden units with 'tanh' activation functions, and 1 linear output unit, and was trained for 1,000 complete cycles of the BFGS quasi-Newton algorithm. It can be seen that the network, which is approximating the conditional average of the target data, gives an excellent representation of the underlying generator of the data. This result is insensitive to the choice of network structure, the initial values for the network weights, and the details of the training procedure.

We now consider the corresponding inverse problem in which we use exactly the same data set as before, but we try to find a mapping from the x variable to the t variable. The result of training a neural network using least-squares is shown in Figure 5. Again the network tries to approximate the conditional average of the target data, but now this corresponds to a very poor representation of the process (55) which generated the data. The precise form of the neural network mapping is now more sensitive to network architecture, weight initialization, etc., than was the case for the forward problem. The mapping shown in Figure 5 is the best result obtained after some careful optimization (with the network often finding significantly poorer solutions). The network in this case had 20 hidden units and was trained for 1,000 cycles of the BFGS algorithm. It is clear that a conventional network, trained by minimizing a sum-of-squares error function, cannot give a good representation of the generator of this data.

We next apply an MDN to the same inverse problem, using the same data set as before.

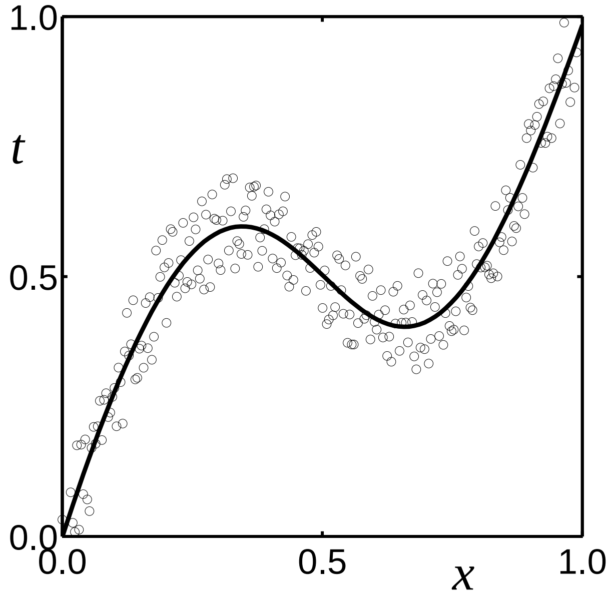

Figure 4: A simple example of a forward problem, showing 1,000 data points (the circles) generated from the mapping $x = t + 0.3\sin(2\pi t) + \epsilon$ where ϵ is a random variable having a uniform distribution in the range $(-0.1, 0.1)$. The solid curve shows the result of training a multi-layer perceptron network with 5 hidden units using a sum-of-squares error function. The network approximates the conditional average of the target data, which gives a good representation of the generator of the data.

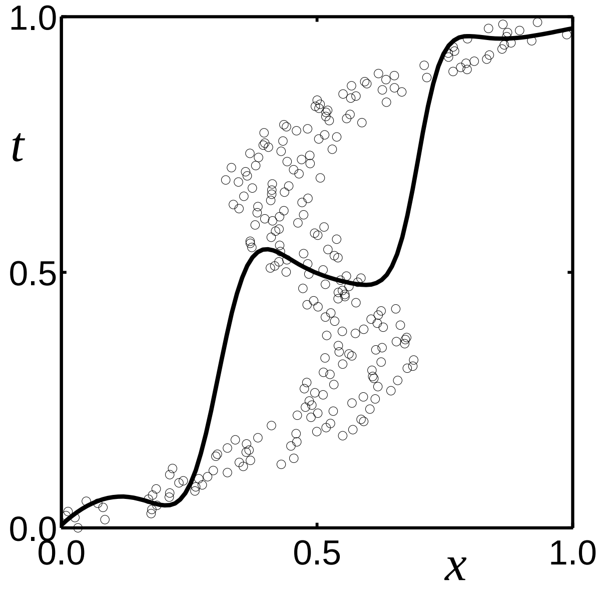

Figure 5: This shows precisely the same data set as in Figure 4, but with the roles of input and target variables interchanged. The solid curve shows the result of training a standard neural network using a sum-of-squares error. This time the network gives a very poor fit, as it tries to represent the conditional average of the target data.

For clarity we restrict attention to MDNs with 3 kernel functions, as this is the minimum number needed to give good solutions for this problem (since the inverse mapping has 3 branches at intermediate values of x, as is clear from Figure 5). In practice, the appropriate number of kernels will not be known in advance and must be addressed as part of the model order selection problem. Experiments with 5 kernel functions on this same problem give almost identical results to those obtained using 3 kernels. We shall discuss the problem of selecting the appropriate number of kernel functions in Section 7. The network component of the MDN was a multi-layer perceptron with 1 input, 20 hidden units with 'tanh' activation functions, and 9 output units (corresponding to the 3 parameters for each of the 3 Gaussian kernel functions). This network structure has not been optimized to any degree since the main purpose of this exercise is to illustrate the operation of the MDN. The MDN was trained with 1,000 cycles of the BFGS algorithm. Once trained, the MDN predicts the conditional probability density of t for each value of x presented to the input of the network. Figure 6 shows contours of $p(t \mid x)$ as a function of t and x. It is clear that the MDN has captured the underlying structure in the data set, despite the multi-valued nature of the inverse problem. Notice that the contour values are much higher in regions of x where the data is single-valued in t. This is a consequence of the fact that $p(t \mid x)$ satisfies $\int p(t \mid x) dt = 1$ at each value of x, and can be seen more clearly in Figure 7 which shows plots of $p(t \mid x)$ versus t for 3 values of x. Note particularly that, for $x = 0.5$, the MDN has correctly captured the tri-modal nature of the mapping.

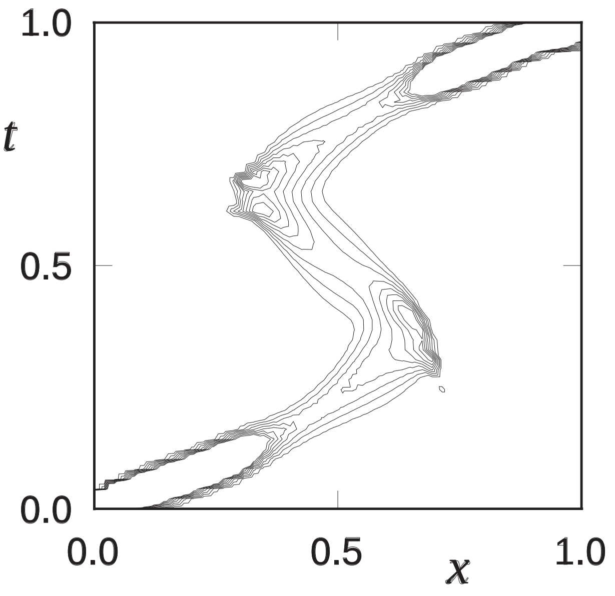

Figure 6: Plot of the contours of the conditional probability density of the target data obtained from a Mixture Density Network trained using the same data as in Figure 5. The network has 3 Gaussian kernel functions, and 5 sigmoidal units in the hidden layer.

The outputs of the neural network part of the MDN, and hence the parameters in the mixture model, are necessarily continuous single-valued functions of the input variables. The MDN is able to produce a conditional density which is unimodal for some values of x and trimodal for other values, as in Figure 6, by modulating the amplitudes of the mixture components. This can be seen in Figure 8 which shows plots of the 3 priors $\alpha_i(x)$ as functions of x. It can be seen that for $x = 0.2$ and $x = 0.8$ only one of the 3 kernels has a significant prior probability. At $x = 0.5$, however, all 3 kernels have comparable priors.

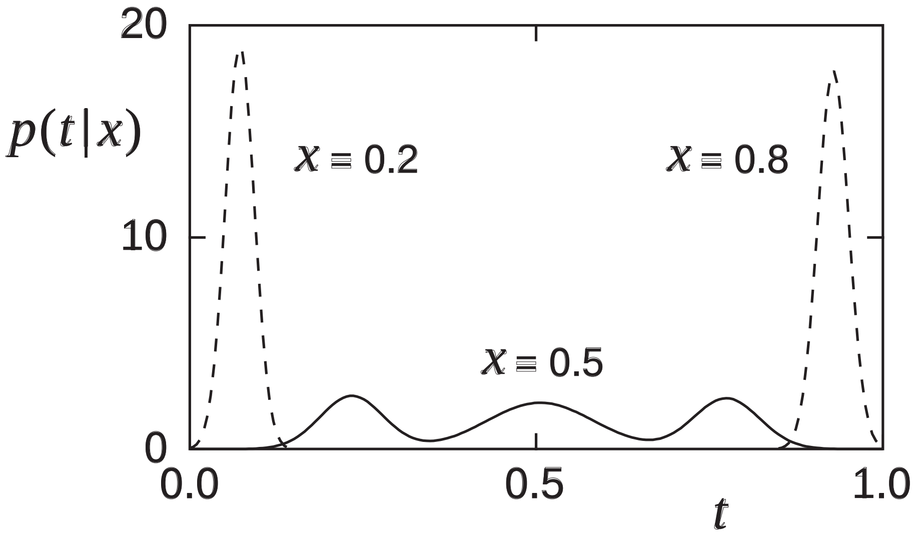

Figure 7: Plot of the conditional probability densities of the target data, for various values of x, obtained by taking vertical slices through the contours in Figure 6, for $x = 0.2$, $x = 0.5$ and $x = 0.8$. It is clear that the Mixture Density Network is able to capture correctly the multi-modal nature of the target data density function at intermediate values of x.

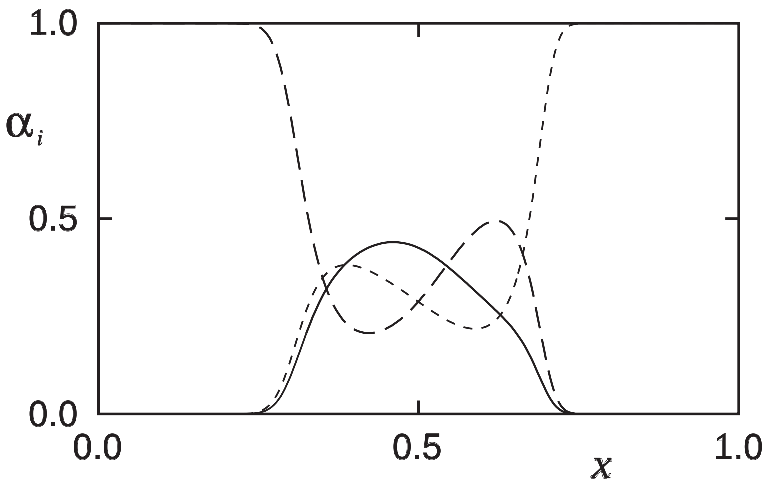

Figure 8: Plot of the priors $\alpha_i(x)$ as a function of x for the 3 kernel functions from the same Mixture Density Network as was used to plot Figure 6. At both small and large values of x, where the conditional probability density of the target data is unimodal, only one of the kernels has a prior probability which differs significantly from zero. At intermediate values of x, where the conditional density is tri-modal, the three kernels have comparable priors.

Having obtained a good representation for the conditional density of the target data, it is then in principle straightforward to calculate any desired statistic from that distribution. We consider first the evaluation of the conditional mean of the target data $\langle t \mid x \rangle$, given by (45), and the squared variance $s^2(x)$ of the target data around this mean, given by (47). Figure 9 shows a plot of $\langle t \mid x \rangle$ against x for the MDN used to plot Figure 6, together with plots of $\langle t \mid x \rangle \pm s(x)$. This representation corresponds to the assumption of a single Gaussian distribution for the target data, but with a variance parameter which is a function of x. While this is more general that the standard least-squares approach (which assumes a constant variance) it still gives a poor representation of the data in the multi-valued region. Notice that, in the regions where the data is single valued, the MDN gives a much smoother and more accurate representation of the conditional average of the target data than was obtained from the standard least-squares neural network as shown in Figure 5. This can be attributed to the fact that the standard network is having to make a single global fit to the whole data set, whereas the MDN uses different kernels for the different branches of the mapping.

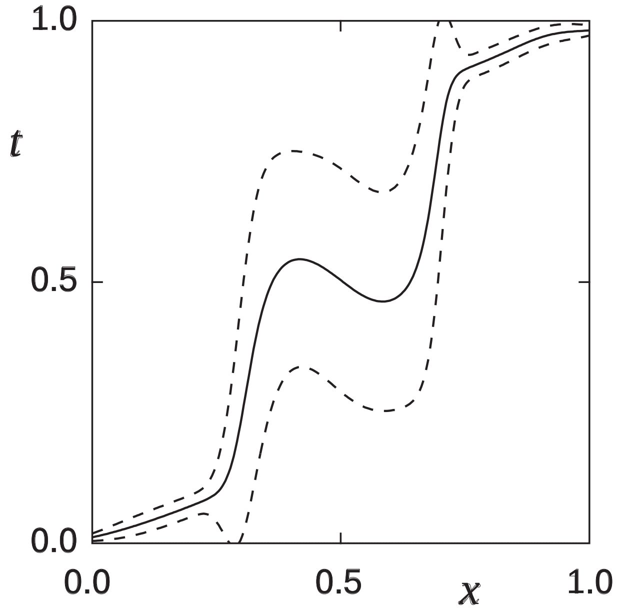

Figure 9: This shows a plot of $\langle t \mid x \rangle$ against x (solid curve) calculated from the MDN used to plot Figure 6, together with corresponding plots of $\langle t \mid x \rangle \pm s(x)$ (dashed curves). Notice that for small and large values of x, where the mapping is single-valued, the MDN actually gives a better representation of the conditional average than the standard least-squares approach, as can be seen by comparison with Figure 5. This can be attributed to the fact that the standard network is having to make a single global fit to the whole data set, whereas the MDN uses different kernels for the different branches of the mapping.

We can also consider the evaluation of the centre of the most probable kernel according to (49), which gives the result shown in Figure 10. This now represents a discontinuous functional mapping from x to t, such that, at each value of x, the MDN make a good prediction for the value of t, which lies well within one of the branches of the data. It can be seen that the discontinuities correspond to the crossing points in Figure 8 which separate the regions in which different priors have the largest value. Comparison with the corresponding mapping obtained with the standard neural network approach, given in Figure 5, shows that the MDN gives substantially improved predictions for the inverse mapping.

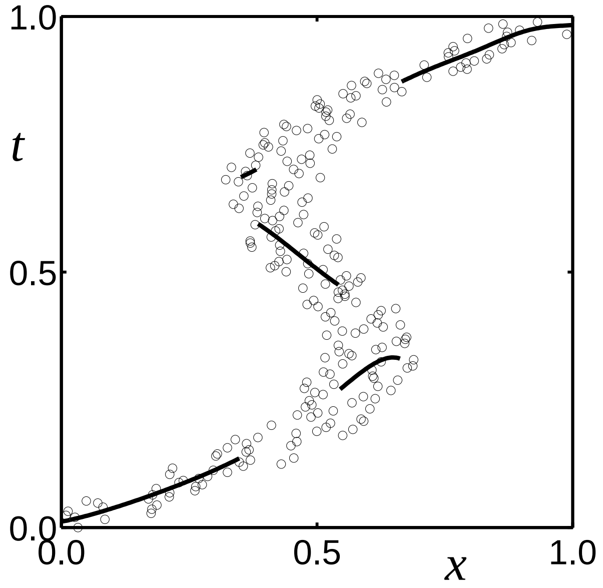

Figure 10: Plot of the central value of the most probable kernel as a function of x from the Mixture Density Network used to plot Figure 6. This gives a (discontinuous) functional mapping from x to t which at every value of x gives an accurate representation of the data. The diagram should be compared with the corresponding result obtained from a conventional neural network, shown in Figure 5.

6 Robot Kinematics

As our second application of Mixture Density Networks, we consider the kinematics of a simple 2-link robot arm, as shown in Figure 11. For given values of the joint angles (θ_1, θ_2), the end effector is moved to a position given by the Cartesian coordinates

$$x_1 = L1\cos(\theta_1) - L2\cos(\theta_1 + \theta_2) \tag{56}$$

$$x_2 = L1\sin(\theta_1) - L2\sin(\theta_1 + \theta_2) \tag{57}$$

where $L1$ and $L2$ are the lengths of the two links of the robot arm. Here we consider a particular configuration of robot for which $L1 = 0.8$ and $L2 = 0.2$ and where θ_1 is restricted to the range $(0.3, 1.2)$ and θ_2 is restricted to the range $(\pi/2, 3\pi/2)$. The mapping from (θ_1, θ_2) to (x_1, x_2) is known as the *forward kinematics*, and is single-valued. However, for practical robot control, the end effector must be moved to prescribed locations and it is therefore necessary to find corresponding values for the joint angles. This is called the *inverse kinematics* and corresponds to the mapping from (x_1, x_2) to (θ_1, θ_2). In general, the inverse kinematics is not a single-valued mapping. This is illustrated in Figure 12, where we see that there are two configurations of the joint angles, known as 'elbow up' and 'elbow down', which both give rise to the same end effector position. The extent of the elbow up and elbow-down regions, for the particular configuration of robot arm considered here, is shown in Figure 13. We see that there are regions (A and C) which are accessible using only one of the two configurations, and for end effector positions in either of these regions, the inverse kinematics will be single valued. There is also a region (B) in which end effector positions can be accessed by both elbow-up and elbow-down configurations, and in this region the inverse kinematics is double-valued.

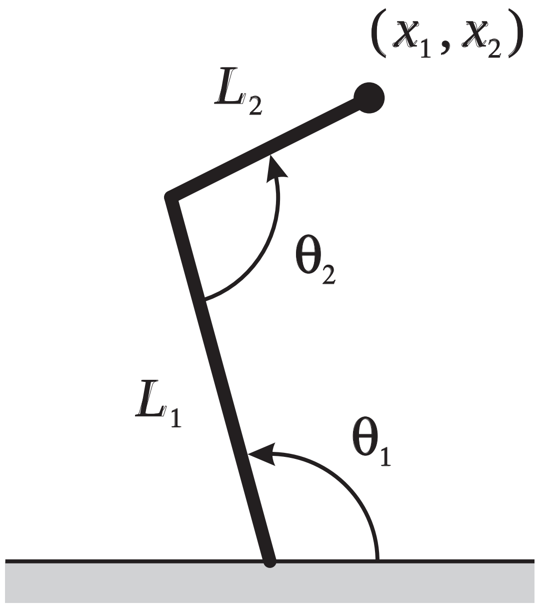

Figure 11: Schematic illustration of a two-link robot arm in two dimensions. For given values of the joint angles θ_1 and θ_2, the position of the end effector, described by the Cartesian coordinates (x_1, x_2), is uniquely determined. In practice, control of the robot arm requires the solution of the *inverse kinematics* problem in which the end effector position is specified and the joint angles θ_1 and θ_2 must be found.

We first consider the use of a standard neural network, trained by minimizing a sum-of-squares error function, to learn the inverse kinematics mapping. A training set of 1,000 points was generated by selecting pairs of joint angles at random with uniform distribution within the allowed limits, and computing the corresponding end effector coordinates using (56) and (57). A test set, also of 1,000 points, was generated in a similar way, but with a different random selection of joint angles. A standard multi-layer perceptron network having 2 inputs, N hidden units with 'tanh' activation functions, and 2 linear output units was used. Here N was set to 5, 10, 15, 20, 25 and 30, and in each case the network was trained for 3,000 complete cycles of the BFGS algorithm. The performance of the network was assessed by presenting the test set end effector coordinates as inputs and using the corresponding values of joint angles predicted by the network to calculate the achieved end effector position using (56) and (57). The RMS Euclidean distance between the desired and achieved end effector positions is used as a measure of performance. This measure is evaluated using the test set, after every 10 cycles of training using the training set, and the network having the smallest RMS error is saved. All the networks gave very similar performance. Figure 14 shows the positioning errors achieved by the best network (20 hidden units) for all of the points in the test set. Comparison with Figure 13 shows that the positioning errors are largest in region B where the inverse kinematics mapping is double valued. In this region the end effector positions achieved by the robot lie at the outer boundary of the accessible region, corresponding to a value of $\theta_2 = \pi$.

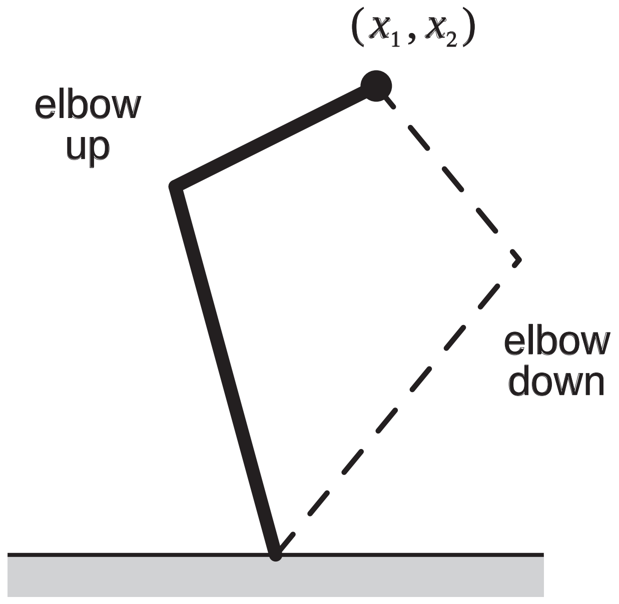

Figure 12: This diagram shows why the inverse kinematics mapping for the robot arm is multi-valued. For the given position (x_1, x_2) of the end effector, there are two solutions for the joint angles, corresponding to 'elbow up' and 'elbow down'.

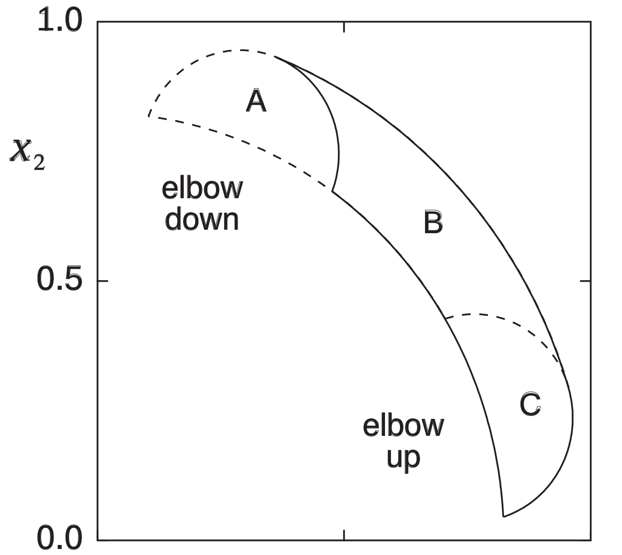

Figure 13: For the particular geometry of robot arm considered, the end effector is able to reach all points in regions A and B in the 'elbow down' configuration, and all points in regions B and C in the 'elbow up' configuration. Thus, the inverse kinematics will correspond to a single-valued mapping for positions in regions A and C, and to a double-valued mapping for positions in region B. The base of the robot arm is at (0.0, 0.0).

Examination of Figure 12 shows that this is indeed the result we would expect, since the average of the joint angles for an elbow-up configuration and the corresponding elbow-down configuration always gives this value for θ_2.

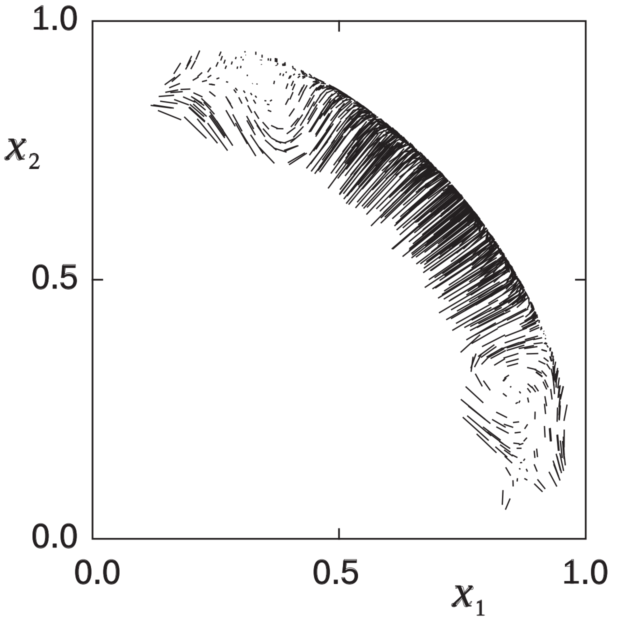

Figure 14: This shows the result of training a conventional neural network, using a sum-of-squares error function, on the inverse kinematics problem for the robot arm corresponding to Figure 13. For each of the 1,000 points in the test set, the positioning error of the end effector is shown as a straight line connecting the desired position (which forms the input to the network) to the actual position achieved by the robot when the joint angles are set to the values predicted by the outputs of the network. Note that the errors are everywhere large, but are smaller for positions corresponding to regions A and C in Figure 13 where the inverse kinematics is single valued, and larger for positions corresponding to the double valued region B.

The same datasets were also used to train an MDN having two kernel functions in the mixture model. The network component of the MDN was a standard multi-layer perceptron having two inputs, N hidden units and 8 output units, and the same optimization procedure was used as for the previous network trained by least squares. In this case the best network had 10 hidden units, and the corresponding position errors are plotted in Figure 15. It is clear that the positioning errors are reduced dramatically compared to the least-squares results shown in Figure 14. A comparison of the RMS positioning errors for the two approaches is given in Table 1, which shows that the MDN gives an order of magnitude reduction in RMS error compared to the least-squares approach.

Model	**RMS positioning error**
Least squares	0.0578
MDN	0.0053

Table 1: Comparison of RMS positioning errors for the robot end effector, measured using the test set, for a standard neural network trained by least-squares, and for a Mixture Density Network.

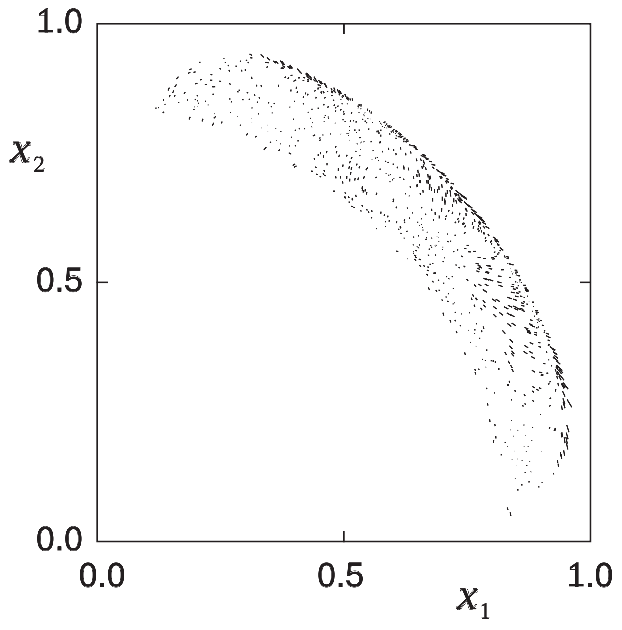

Figure 15: As in Figure 14, but showing the corresponding results obtained using a Mixture Density Network. The RMS error in positioning the end effector is reduced by an order of magnitude compared with the conventional network, and the errors remain small even in the region where the inverse kinematics is double-valued.

7 Discussion

In this paper we have introduced a new class of networks, called Mixture Density Networks, which can model general conditional probability densities. By contrast, the conventional network approach, involving the minimization of a sum-of-squares error function, only permits the determination of the conditional average of the target data, together with a single global variance parameter. We have illustrated the use of Mixture Density Networks for a simple 1-input 1-output mapping, and for a robot inverse kinematics problem. In both of these examples the required mapping is multi-valued and so is poorly represented by the conditional average.

There are many other approaches to dealing with the problem of learning multi-valued mappings from a set of example data. In general, however, these are concerned with generating one specific solution (i.e. one branch of the multi-valued mapping). The Mixture Density Network, by contrast, is concerned with modelling the complete conditional density function of the output variables, and so gives a completely general description of the required mapping. From the conditional density, more specific information can be extracted. In particular, we have discussed methods for evaluating moments of the conditional density (such as the mean and variance), as well as for selecting a particular branch of a multi-valued mapping.

Implementation of Mixture Density Networks is straightforward, and corresponds to a modification of the error function, together with a different interpretation for the network outputs. One aspect of the MDN which is more complex than with standard models is the problem of model order selection. In applying neural networks to finite data sets, the degree of complexity of the model must be optimized to give the best generalization performance. This might be done by varying the number of hidden units (and hence the number of adaptive parameters) as was done for the simulations in this paper. It could also be done through the use of regularization terms added to the error function, or through the use of 'early stopping' during training to limit the effective number of degrees of freedom in the network. The same problem of model complexity must also be addressed for MDNs. However, there is in addition the problem of selecting the appropriate number of kernel functions. Changes to the number of kernels leads to changes in the number of adaptive parameters in the network through changes to the number of output units (for a given number of hidden units), and so the two problems are somewhat interrelated. For problems involving discrete multi-valued mappings it is important that the number of kernel functions is at least equal to the maximum number of branches of the mapping. However, it is likely that the use of a greater number of kernel functions than this will have little ill effect, since the network always has the option either of 'switching off' redundant kernels by setting the corresponding priors to small values, or of 'combining' kernels by giving them similar $\boldsymbol{\mu}_i$ and σ_i parameters. Preliminary experiments on the problems discussed in this paper involving a surplus of kernels indicates that there is no significant reduction in network performance. Future research will be concerned with the automation of model order selection for Mixture Density Networks, as well as with the performance of these networks in a range of large-scale applications.

Acknowledgements

I would like to thank Pierre Baldi, David Lowe, Richard Rohwer and Andreas Weigend for providing helpful comments on an earlier draft of this report.

References

Bishop, C. M. (1994). Novelty detection and neural network validation. *IEE Proceedings: Vision, Image and Signal Processing* **141** (4), 217–222. Special issue on applications of neural networks.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

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

Geman, S., E. Bienenstock, and R. Doursat (1992). Neural networks and the bias/variance dilema. *Neural Computation* **4** (1), 1–58.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* **2** (5), 359–366.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton (1991). Adaptive mixtures of local experts. *Neural Computation* **3** (1), 79–87.

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

Nowlan, S. J. and G. E. Hinton (1992). Simplifying neural networks by soft weight sharing. *Neural Computation* **4** (4), 473–493.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992). *Numerical Recipes in C: The Art of Scientific Computing* (Second ed.). Cambridge University Press.

White, H. (1989). Learning in artificial neural networks: a statistical perspective. *Neural Computation* **1** (4), 425–464.