Bayesian Neural Networks and Density Networks

David J.C. MacKay
University of Cambridge
Cavendish Laboratory
Madingley Road
Cambridge CB3 0HE
mackay@mrao.cam.ac.uk

To appear in Proceedings of Workshop on Neutron Scattering Data Analysis 1994

Abstract

This paper reviews the Bayesian approach to learning in neural networks, then introduces a new adaptive model, the density network. This is a neural network for which target outputs are provided, but the inputs are unspecified. When a probability distribution is placed on the unknown inputs, a latent variable model is defined that is capable of discovering the underlying dimensionality of a data set. A Bayesian learning algorithm for these networks is derived and demonstrated.

1 Introduction to the Bayesian view of learning

A binary classifier is a parameterized mapping from an input $\mathbf{x}$ to an output $y \in [0,1])$; when its parameters $\mathbf{w}$ are specified, the classifier states the probability that an input $\mathbf{x}$ belongs to class $t = 1$, rather than the alternative $t = 0$. Consider a binary classifier which models the probability as a sigmoid function of $\mathbf{x}$:

$$P(t = 1|\mathbf{x}, \mathbf{w}, \mathcal{H}) = y(\mathbf{x}; \mathbf{w}, \mathcal{H}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}} \tag{1}$$

This form of model is known to statisticians as a linear logistic model, and in the neural networks field as a single neuron.

For convenience let us study the case where the input vector $\mathbf{x}$ and the parameter vector $\mathbf{w}$ are both two-dimensional. Figures 1(a1-8) show the output of the classifier as a function of the input vector, for a selection of different values of the parameter vector $\mathbf{w}$. Figure 1b shows the *points* in $\mathbf{w}$ space that correspond to these functions of $\mathbf{x}$. Notice that the ramp of each sigmoid function is oriented in the plane perpendicular to the corresponding vector $\mathbf{w}$. The gain of the ramp is proportional to the magnitude of $\mathbf{w}$.

Now imagine that we receive some data as shown in the left column of figure 2. Each data point consists of a two dimensional input vector $\mathbf{x}$ and a t value indicated by $\times$ $(t = 1)$ or $\Box$ $(t = 0)$. The second column shows the *likelihood* as a function of $\mathbf{w}$; this is the probability assigned to the observed t values by the model with parameters set to $\mathbf{w}$. It is a product of functions of the form (1).

In the traditional view of learning, a single parameter vector $\mathbf{w}$ evolves under the learning rule from an initial starting point $\mathbf{w}^0$ to a final optimum $\mathbf{w}^*$, in such a way as to minimize an *objective function* equal to minus the log likelihood plus a *regularizer* such as $\alpha \sum_i w_i^2/2$. The product of learning is the estimator $\mathbf{w}^*$. In contrast, in the Bayesian view, the product of learning is an *ensemble* of plausible parameter values (bottom right of figure 2). This posterior ensemble is obtained by multiplying the

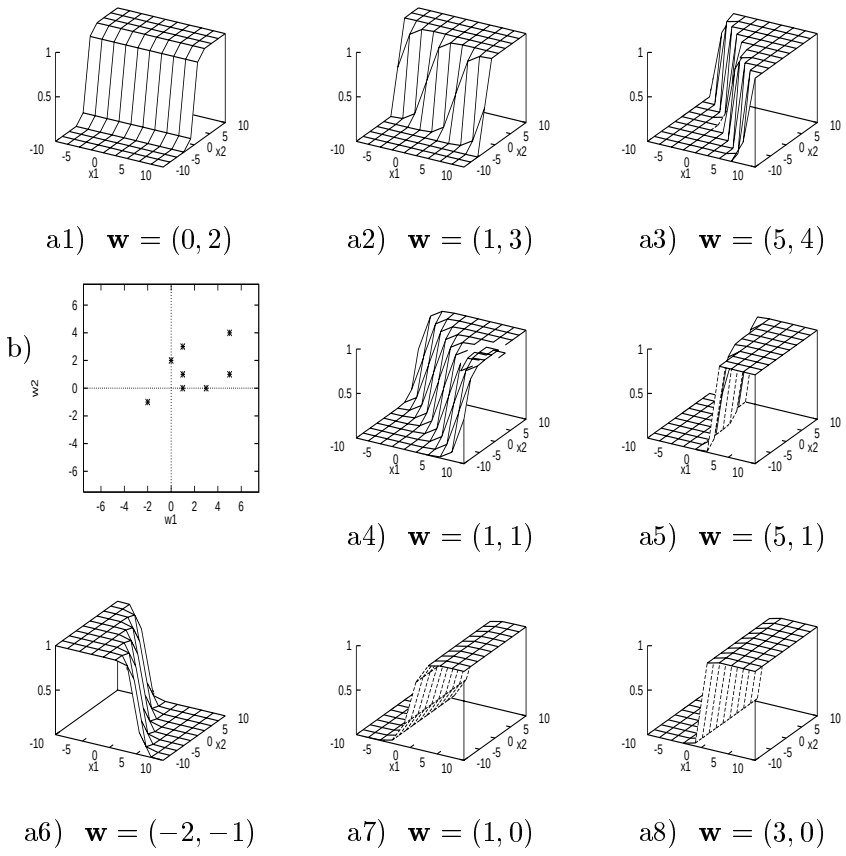


Figure 1: **(a1-8) The output of the classifier as a function of the input, for a variety of different values of its parameters. (b) The parameter space of the model, showing each of the functions in (a1-8) as a point.**

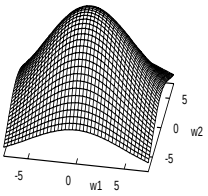
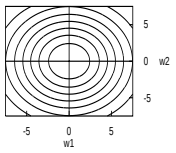
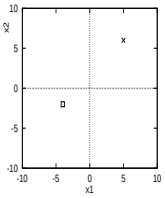
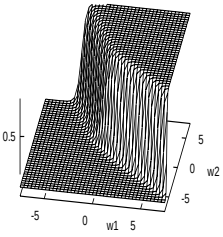
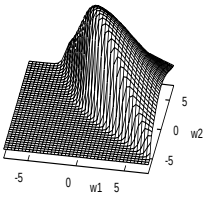
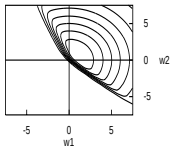
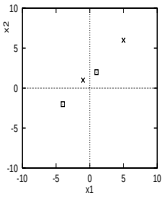
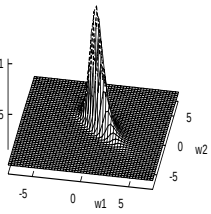
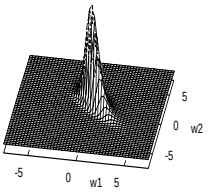
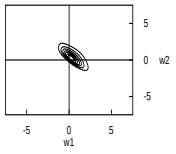
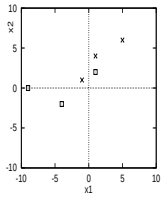
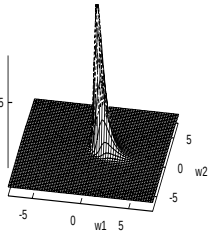
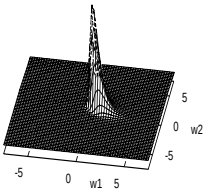
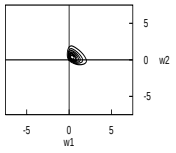
N	Data set	Likelihood	Probability of parameters	
0		*(constant)*		
2				
4				
6				

Figure 2: **Evolution of the probability distribution over parameters as data arrives.**

likelihood by a prior ensemble over $\mathbf{w}$ space (shown as a broad Gaussian at upper right of figure 2). The third column of figure 2 shows the posterior ensemble (within a multiplicative constant), obtained by multiplying the prior by the likelihood. The same function is shown in a contour plot in the fourth column. We obtain a correspondence between traditional learning and the maximum of the Bayesian posterior probability distribution if the prior is defined by setting its logarithm equal to the regularizer; the regularizer $\alpha \sum_i w_i^2/2$ then defines a Gaussian prior with variance $\sigma_w^2 = 1/\alpha$, $P(\mathbf{w}|\alpha, \mathcal{H}) = \exp(-\alpha \sum_i w_i^2/2)/Z_w$, with $Z_w = (2\pi/\alpha)^{k/2}$.

Note therefore that the contrast with traditional learning methods is not the Bayesian's prior — this just corresponds to the regularizer already used in traditional learning; the contrast is that the Bayesian obtains a probability distribution over parameter space, rather than a point estimate in that space.

The Bayesian viewpoint allows the following advantages to be gained by simple application of the rules of probability theory.

1. By viewing the product of learning as an ensemble of probable classifiers, we can take our uncertainty into account when making predictions. This improves the quality of the predictions.

2. By viewing the regularizer with regularization constant α as defining a prior probability, we can solve the 'overfitting problem', *i.e.*, the problem of setting hyperparameters such as α that control the 'complexity' of the model. With probability theory we can effectively infer the appropriate value of the regularization constant α from the data. This removes the need to waste resources on cross-validation, which is traditionally used to set such hyperparameters. This is an especially important advantage of the Bayesian approach when there are many such regularization constants.

The Bayesian inference of $\mathbf{w}$ given the data $D = \{t^{(n)}\}_{n=1}^N$ (the classifications of the N examples) can be written in symbols as follows, using Bayes' theorem:

$$P(\mathbf{w}|D, \alpha, \mathcal{H}) = \frac{P(D|\mathbf{w}, \mathcal{H})P(\mathbf{w}|\alpha, \mathcal{H})}{P(D|\alpha, \mathcal{H})}. \tag{2}$$

Here $\mathcal{H}$ and α denote the assumed model and its hyperparameters. We infer α given the data by writing Bayes' theorem again:

$$P(\alpha|D, \mathcal{H}) = \frac{P(D|\alpha, \mathcal{H})P(\alpha|\mathcal{H})}{P(D|\mathcal{H})}. \tag{3}$$

The data-dependent term $P(D|\alpha, \mathcal{H})$ in this second inference is the normalizing constant of the first inference, and we call it the *evidence* for α and $\mathcal{H}$.

$$\text{Evidence} = P(D|\alpha, \mathcal{H}) = \int d^k\mathbf{w}\, P(D|\mathbf{w}, \mathcal{H})P(\mathbf{w}|\alpha, \mathcal{H}) \tag{4}$$

We make predictions (of a new datum $t^{(N+1)}$, for example) by *marginalizing* over the unknown parameters $\mathbf{w}$ and hyperparameters α:

$$P(t^{(N+1)}|D, \mathcal{H}) = \int d^C\alpha\, d^k\mathbf{w}\, P(t^{(N+1)}|\mathbf{w}, \mathcal{H})P(\mathbf{w}|D, \alpha, \mathcal{H})P(\alpha|D, \mathcal{H}). \tag{5}$$

The Bayesian framework has been implemented in two ways: first by approximating the posterior distribution of $\mathbf{w}$ (2) by a Gaussian fitted at the optimum $\mathbf{w}^*$ [1, 2]; and by methods that represent the posterior distribution by a set of Monte Carlo samples from it [3]. The former approach has been successfully applied to practical problems, as described elsewhere [4, 5]. See ref. [6] for a review.

In the general case of a classification problem with multiple classes $i = 1 \ldots I$, a 'softmax' classifier is a natural form of model. This assigns probabilities to the alternative classes i thus:

$$P(t = i|\mathbf{x}, \mathbf{w}, \mathcal{H}) = y_i(\mathbf{x}; \mathbf{w}) = \frac{e^{a_i(\mathbf{x};\mathbf{w})}}{\sum_{i'} e^{a_{i'}(\mathbf{x};\mathbf{w})}}, \tag{6}$$

where $\{a_i(\mathbf{x}; \mathbf{w})\}_{i=1}^I$ are linear or non-linear functions of $\mathbf{x}$ parameterized by $\mathbf{w}$. It is convenient to define the log of the likelihood $P(D|\mathbf{w}, \mathcal{H}) = \prod_{n=1}^N P(t = t^{(n)}|\mathbf{x}, \mathbf{w}, \mathcal{H})$ thus:

$$G(\mathbf{w}) \equiv \log P(D|\mathbf{w}, \mathcal{H}) = \sum_{n=1}^{N} \log y_{t^{(n)}}(\mathbf{x}; \mathbf{w}). \tag{7}$$

The derivative of this function with respect to $\mathbf{w}$ is simple to evaluate for the models described above using the 'backpropagation' algorithm (chain rule) [7].

This paper now describes a new Bayesian neural network model.

2 Density Modelling

The most popular supervised neural networks, multilayer perceptrons (MLPs), are well established as probabilistic models for regression and classification, both of which are *conditional* modelling tasks: the input variables are assumed given, and we *condition* on their values; no model of the density over input variables is constructed. Density modelling (or generative modelling), on the other hand, is a name for modelling tasks in which a density over all the observable quantities is constructed. Multilayer perceptrons have not conventionally been used to create density models (though belief networks [8] and other neural networks such as the Boltzmann machine [9] do define density models). Various interesting research problems in this field relate to the difficulty of defining a full probabilistic model with an MLP. For example, if some inputs in a regression problem are 'missing', then traditional methods offer no principled way of filling the gaps. This paper discusses how one can use an MLP as a density model.

Traditional density models

A popular class of density models are *mixture models*, which define the density as a sum of simpler densities.

Mixture models might however be viewed as inappropriate models for high-dimensional data spaces such as images or genome sequences. The number of components in a mixture model has to scale exponentially as we add independent degrees of freedom. Consider, for example, a protein in which there is a strong correlation between the amino acids in the first and second columns — they are either both hydrophobic, or both hydrophilic, say — and there is an independent correlation between two other amino acids later in the protein chain — when one of them has a large residue the other has a small residue, say. A mixture model would have to use four categories to capture all four combinations of these binary attributes, whereas only two independent degrees of freedom are really present.

These observations motivate the development of density models that have *components* rather than *categories* as their 'latent variables' [10]. Let us denote the observables by $\mathbf{t}$. If a density is defined on the latent variables $\mathbf{x}$, and a parameterized mapping is defined from these latent variables to a probability distribution over the observables $P(\mathbf{t}|\mathbf{x}, \mathbf{w})$, then a non-trivial density over $\mathbf{t}$ is defined. Simple linear models of this form in the statistics literature come under the labels of 'factor analysis' and 'principal components analysis'. Here I allow $P(\mathbf{t}|\mathbf{x}, \mathbf{w})$ to be a non-linear parameterized mapping, and use interesting priors on $\mathbf{w}$. I suggest the name 'density networks' for these models.

The model

The 'latent inputs' of the model are a vector $\mathbf{x}$ indexed by $h = 1 \ldots H$ ('h' mnemonic for 'hidden'). The dimensionality of this hidden space is H but the effective dimensionality assigned by the model in the output space may be smaller, as some of the hidden dimensions may be effectively unused by the model. The relationship between the latent inputs and the observables, parameterized by $\mathbf{w}$, has the form of a mapping from inputs to outputs $\mathbf{y}(\mathbf{x}; \mathbf{w})$, and a probability of targets given outputs, $P(\mathbf{t}|\mathbf{y})$. The observed data are a set of target vectors $D = \{\mathbf{t}^{(n)}\}_{n=1}^{N}$. To complete the model we assign a prior $P(\mathbf{x})$ to the latent inputs (an independent prior for each vector $\mathbf{x}^{(n)}$) and a prior $P(\mathbf{w})$ to the unknown parameters. [In the applications that follow the priors over $\mathbf{w}$ and $\mathbf{x}^{(n)}$ are assumed to be spherical Gaussians; other distributions could easily be implemented and compared, if desired.] In summary, the probability of everything is:

$$P(D, \{\mathbf{x}^{(n)}\}, \mathbf{w}|\mathcal{H}) = \prod_n \left[P(\mathbf{t}^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}, \mathcal{H}) P(\mathbf{x}^{(n)}|\mathcal{H}) \right] P(\mathbf{w}|\mathcal{H}) \tag{8}$$

It will be convenient to define the 'error functions' $G^{(n)}(\mathbf{w}; \mathbf{x})$ as follows:

$$G^{(n)}(\mathbf{x}; \mathbf{w}) \equiv \log P(\mathbf{t}^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}) \tag{9}$$

The function G depends on the nature of the problem. If $\mathbf{t}$ consists of real variables then G might be a sum-squared error between $\mathbf{t}$ and $\mathbf{y}$; in a 'softmax' classifier, it is a 'cross entropy'. In general we may have many output groups of different types. The following derivation applies to all cases. If you wish to have a concrete example in mind, think of the observable $\mathbf{t}$ as consisting of four attributes that are believed to be correlated. Each attribute can take one of a range of discrete values, a probability over which is modelled with a softmax group. This corresponds to the toy example of the following section.

Having written down the probability of everything we can now make any desired inferences by turning the handle of probability theory. Let us aim towards the inference of the parameters $\mathbf{w}$ given the data D, $P(\mathbf{w}|D, \mathcal{H})$. We can obtain this quantity conveniently by distinguishing two levels of inference.

Level 1

Given $\mathbf{w}$ and $\mathbf{t}^{(n)}$, infer $\mathbf{x}^{(n)}$. The posterior distribution of $\mathbf{x}^{(n)}$ is

$$P(\mathbf{x}^{(n)}|\mathbf{t}^{(n)}, \mathbf{w}, \mathcal{H}) = \frac{P(\mathbf{t}^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}, \mathcal{H}) P(\mathbf{x}^{(n)}|\mathcal{H})}{P(\mathbf{t}^{(n)}|\mathbf{w}, \mathcal{H})}, \tag{10}$$

where the normalizing constant is:

$$P(\mathbf{t}^{(n)}|\mathbf{w}, \mathcal{H}) = \int d^H \mathbf{x}^{(n)} \; P(\mathbf{t}^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}, \mathcal{H}) P(\mathbf{x}^{(n)}|\mathcal{H}). \tag{11}$$

Level 2

Given $D = \{\mathbf{t}^{(n)}\}$, infer $\mathbf{w}$.

$$P(\mathbf{w}|D, \mathcal{H}) = \frac{P(D|\mathbf{w}, \mathcal{H}) P(\mathbf{w}|\mathcal{H})}{P(D|\mathcal{H})} \tag{12}$$

The data-dependent term here is a product of the normalizing constants of the level 1 inferences:

$$P(D|\mathbf{w}, \mathcal{H}) = \prod_{n=1}^{N} P(\mathbf{t}^{(n)}|\mathbf{w}, \mathcal{H}) \tag{13}$$

The evaluation of the evidence $P(\mathbf{t}^{(n)}|\mathbf{w}, \mathcal{H})$ for a particular n is a problem similar to the evaluation of the evidence for a supervised neural network (cf. equation 4). In a supervised neural network, the

inputs $\mathbf{x}$ are given, and the parameters $\mathbf{w}$ are unknown; we obtain the evidence by integrating over $\mathbf{w}$. In the present problem, on the other hand, the hidden vector $\mathbf{x}$ is unknown, and the parameters $\mathbf{w}$ are conditionally fixed, for the purposes of the evidence evaluation. This is an example of the duality discussed in ref. [11].

Learning: the derivative of the evidence with respect to w

The derivative of the log of the evidence (equation 11) is:

$$\frac{\partial}{\partial \mathbf{w}} \log P(\mathbf{t}^{(n)}|\mathbf{w},\mathcal{H}) = \frac{1}{P(\mathbf{t}^{(n)}|\mathbf{w},\mathcal{H})} \int d^H\mathbf{x}\ \exp(G^{(n)}(\mathbf{x};\mathbf{w}))P(\mathbf{x}|\mathcal{H})\frac{\partial}{\partial \mathbf{w}}G^{(n)}(\mathbf{x};\mathbf{w}) \quad (14)$$

$$= \int d^H\mathbf{x}\, P(\mathbf{x}|\mathbf{t}^{(n)},\mathbf{w},\mathcal{H})\frac{\partial}{\partial \mathbf{w}}G^{(n)}(\mathbf{x};\mathbf{w}) \quad (15)$$

This gradient can thus be written as an expectation of the traditional 'backpropagation' gradient $\frac{\partial}{\partial \mathbf{w}}G^{(n)}(\mathbf{x};\mathbf{w})$, averaging over the posterior distribution of $\mathbf{x}^{(n)}$ found in equation (10).

Higher levels — priors on w

We can continue up the hierarchical model, putting a prior on $\mathbf{w}$ with hyperparameters $\{\alpha\}$ which are inferred by integrating over $\mathbf{w}$. These priors are important from a practical point of view to limit overfitting of the data by the parameters $\mathbf{w}$. These priors will also be used to bias the solutions towards ones that are easier for humans to interpret.

Evaluation of the evidence and its derivatives using simple Monte Carlo sampling

The evidence and its derivatives with respect to $\mathbf{w}$ both involve integrals over the hidden components $\mathbf{x}$. For a hidden vector of sufficiently small dimensionality, a simple Monte Carlo approach to the evaluation of these integrals can be effective.

Let $\{\mathbf{x}^{(r)}\}_{r=1}^R$ be random samples from $P(\mathbf{x})$. Then we can approximate the log evidence by:

$$\log P(\{\mathbf{t}^{(n)}\}|\mathbf{w},\mathcal{H}) = \sum_n \log \int d^H\mathbf{x}\ \exp(G^n(\mathbf{x};\mathbf{w}))P(\mathbf{x})$$

$$\simeq \sum_n \log \left[\frac{1}{R}\sum_r \exp(G^n(\mathbf{x}^{(r)};\mathbf{w}))\right].$$

Similarly the derivative can be approximated by:

$$\frac{\partial}{\partial \mathbf{w}} \log P(\{\mathbf{t}^{(n)}\}|\mathbf{w},\mathcal{H}) \simeq \sum_n \frac{\sum_r \exp(G^n(\mathbf{x}^{(r)};\mathbf{w}))\frac{\partial}{\partial \mathbf{w}}G^n(\mathbf{x}^{(r)};\mathbf{w})}{\sum_r \exp(G^n(\mathbf{x}^{(r)};\mathbf{w})).}$$

Some practical details

I have optimized $\mathbf{w}$ by evaluating the evidence and its gradient and feeding them into a conjugate gradient routine [12]. The random points $\{\mathbf{x}^{(r)}\}$ are kept fixed, so that the objective function and its gradient are deterministic functions during the optimization. This also has the advantage of allowing one to get away with a smaller number of samples R than might be thought necessary, as the parameters $\mathbf{w}$ can adapt to make the best use of the empirical distribution over $\mathbf{x}$.

More efficient evaluation of the evidence using importance sampling

If we create a sampling distribution $Q^n(\mathbf{x})$ that is similar to the posterior distribution $P(\mathbf{x}|\mathbf{t}^{(n)}, \mathbf{w}, \mathcal{H})$ then the evidence can be approximated in terms of $\{\mathbf{x}^{(r)}\}_{r=1}^R$, which are random samples from $Q(\mathbf{x})$:

$$\begin{aligned} \log P(\mathbf{t}^{(n)}|\mathbf{w}, \mathcal{H}) &= \log \int d^H\mathbf{x} \, \exp(G^n(\mathbf{x}; \mathbf{w}))P(\mathbf{x}) \\ &\simeq \log \left[\frac{1}{R} \sum_r \exp(G^n(\mathbf{x}; \mathbf{w})) \frac{P(\mathbf{x})}{Q^n(\mathbf{x})} \right] \end{aligned}$$

This approach may have the advantage of better convergence than plain sampling with $Q(\mathbf{x}) = P(\mathbf{x})$, but the adaptation of the sampler $Q(\mathbf{x})$ may require considerable computational effort.

3 A componential density model for a protein family

A protein is a sequence of amino acids. A protein family is a set of proteins believed to have the same physical structure but not necessarily having the same sequence of amino acids. Each location in the sequence of amino acids is known as a 'column'. There are twenty different amino acids, and columns can often be characterised by a predominance of particular amino acids.

The development of models for protein families is useful for two reasons. The first is that a good model might be used to identify new members of an existing family, and discover new families too, in data produced by genome sequencing projects. The second reason is that a sufficiently complex model might be able to give new insight into the properties of the protein family; for example, properties of the proteins' tertiary structure might be elucidated by a model capable of discovering suspicious long-range correlations.

The only probabilistic model that has so far been applied to protein families is a hidden Markov model [13]. This model is not inherently capable of discovering long-range correlations, as Markov models, by definition, produce no correlations between observables, given a hidden state sequence.

The next-door neighbour of proteins, RNA, has been modelled with a 'covariance model' capable of capturing correlations between base-pairs in anti-parallel RNA strands [14].

Here I model the protein families using a density network containing one softmax group for each column. Toy data is shown in table 1. Real data describing 400 proteins in the globin family was received in aligned form courtesy of Sean Eddy (modelling of unaligned data is possible in principle, but harder), with $S = 208$ columns each containing one of $I = 21$ symbols (twenty amino acids and 'deletion'), or else a 'no measurement' symbol. The density network maps from H latent inputs to SI outputs, grouped in S softmax groups of I units each. I used a single layer network that maps linearly from the latent inputs $\mathbf{x}$ to the functions $a(\mathbf{x}; \mathbf{w})$ appearing in equation (6). When $a(\mathbf{x}; \mathbf{w})$ have been computed, each softmax group is normalized separately so as to define a probability over the amino acids in one column. With $H = 20$ latent dimensions this model has about 80,000 parameters. The special case of $H = 0$ latent inputs creates a model with independent probabilities over amino acids at each column, which is roughly equivalent to the hidden Markov model.

Regularization schemes

A human prejudice towards comprehensible solutions gives an additional motivation for regularizing the model, beyond the usual reasons for having priors. We can encourage the model to be comprehensible in various ways:

1. There is a redundancy in the model regarding where it gets its randomness from. Assume that a particular output is actually random and uncorrelated with other outputs. This could be modelled in two ways: its weights from the latent inputs could be set to zero, and the biases

could be set to the log probabilities; or alternatively the biases could be fixed to arbitrary values, with appropriate connections to unused latent inputs being used to create the required probabilities, on marginalization over the latent variables. In predictive terms, these two models would be identical, but we prefer the first solution, finding it more intelligible. To encourage such solutions we can use a prior which weakly regularizes the biases, so that they are 'cheap', while strongly regularizing the other parameters.

2. If the distribution $P(\mathbf{x})$ is rotationally invariant, then the predictive distribution is invariant under corresponding transformations of the parameters $\mathbf{w}$. If a solution can be expressed in terms of parameter vectors aligned with some of the axes (*i.e.* so that some parameters are zero), then we would prefer that. A non-spherical prior on the parameters can be created in two ways:

   (a) Non-Gaussian priors of the form, *e.g.*, $P(\mathbf{w}) \propto \exp(-\alpha \sum w^\beta/\beta)$, with $\beta < 2$, encourage $\mathbf{w}$ to align with the axes.

   (b) Multiple regularizers. Another way to make entire collections of weights assume small values is to use multiple undetermined regularization constants $\{\alpha_c\}$, each one associated with a class of weights (cf. the automatic relevance determination model [5, 15]). For example, a weight class could consist of all the weights from one latent input to one softmax group. This prior would then favour solutions in which one latent input has non-zero connections to all the units in some softmax groups (corresponding to small α_c), and negligible connections to other softmax groups (large α_c). This approach has the advantage that it encourages whole blocks of weights to go to zero together in an way that can be interpreted in terms of correlations between columns.

In the results described here, the final method above was used; for a protein with S columns modelled using H latent variables, I introduced SH regularization constants $\{\alpha_c\}$, each specifying whether a particular latent variable has an influence of a particular column. Given α_c, the prior on the parameters in class c is Gaussian with variance $1/\alpha_c$. For given values of $\{\alpha_c\}$, the parameters $\mathbf{w}$ were optimized to locally maximize the posterior probability. No explicit Gaussian approximation was made to the posterior distribution of $\mathbf{w}$, but the hyperparameters $\{\alpha_c\}$ were adapted during the optimization of the parameters $\mathbf{w}$, using a cheap and cheerful method motivated by Gaussian approximations [1], thus:

$$\alpha_c := f \frac{k_c}{\sum_{i \in c} w_i^\beta}. \tag{16}$$

Here k_c is the number of parameters in class c, f is a 'fudge factor' incorporated to imitate the effect of integrating over $\mathbf{w}$ (set to a value between 0.1 and 1.0), and β is the exponent of the prior (set to 2.0 in this work). The biases' regularization constant was not adapted, but was set to a weak value throughout in order to enforce the prejudice explained above.

This algorithm could be converted to a correct 'stochastic dynamics' Monte Carlo method [3] by adding an appropriate amount of noise to gradient descent on $\mathbf{w}$ and setting $f = 1$.

Toy data

A toy data set was created imitating a protein family with four columns each containing one of five amino acids. The 27 data (table 1) were constructed to exhibit two correlations between the columns: the first and second columns have a tendency both to be amino acid E together. The third and fourth columns are correlated such that if one is amino acid B, then the other is likely to be A, B or C; the if one is C, then the other is likely to be B, C or D; and so forth, with an underlying single dimension running through the amino acids A,B,C,D. The model is given no prior knowledge of the 'spatial

```
EEAB EECB EEBC EECC EEAA EEBA EEBB EECD
EEDC EEDD AACD DDDC CBDD CCAB BDCB ABBC
CBCC EDAA ABBA BCBB DBAB AECB EBBC BDCC
BCAA DABA BCBB
```

Table 1: Toy data for a protein family

	Input 1	Input 2	Input 3	Input 4
Column 1	100	100	0.502863	100
Column 2	100	100	0.423623	100
Column 3	0.733805	100	100	100
Column 4	0.718757	100	100	100

Table 2: Regularization constants inferred for toy protein family

relationship' of the columns, or of the ordering of the amino acids. A model that can identify the two correlations in the data is what we are hoping for.

Both regularized and unregularized density networks having four latent inputs were adapted to this data. Unregularized density networks give solutions that successfully predict the two correlations, but the parameters of those models are hard to interpret. The regularized models, in which all the parameters connecting one input to one softmax group are put in a regularization class with an unknown hyperparameter α_c, interpretable solutions are obtained that clearly identify the two correlated groups of columns. Table 2 shows the regularization constants inferred in a typical solution. An upper bound of 100 was set on all the regularization constants; this value is interpreted as signifying that there is no influence of the input on the residue in question. Notice that two of the inputs are unused in this solution. Of the other two inputs, one has an influence on columns 1 and 2 only, and the other has an influence on columns 3 and 4 only. Thus this model has successfully revealed the underlying 'structure' of the proteins in this family.

Results on real data

A density network with $H = 2$ latent inputs was adapted to data describing 400 globins of various sorts. This leads to a componential representation of these globins that is easy to visualize. The posterior mean of the latent components (estimated by Monte Carlo) is displayed for each globin in figure 3. The type of each globin is identified by the point style. It is evident that the components have identified relevant properties of the proteins, as the types are cleanly separated in this representation.

In work in progress, $H = 20$ latent inputs have been used, and it is believed that some of the correlations discovered by the model may relate to the physical structure of the protein.

More complex models under development will include additional layers of processing between the latent variables and the observables. If some of the parameters of a second layer were communal to all columns of the protein, the model would be able to generalize amino acid equivalences from one column to another.

A poor assumption in this work is that, given the parameters $\mathbf{w}$, the observed sequences $\mathbf{t}$ are independent. In fact the sequences form an evolutionary tree (of which several main branches are identified in figure 3). A better model would represent this evolutionary tree as part of the probabilistic structure. It would be interesting to attempt to represent the evolution as taking place in the latent

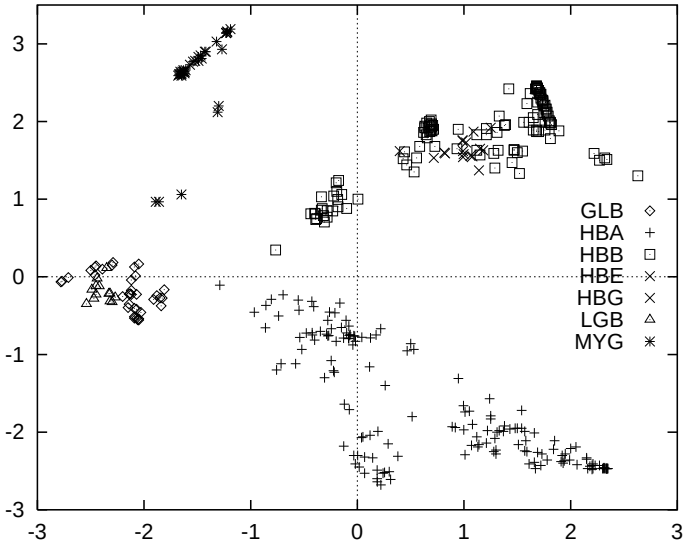


Figure 3: **Componential space for globins.**
The posterior mean of the latent inputs is shown for each globin in the training set. Note the clean separation of the main classes of globin.

variable space of a density network.

4 Discussion

The missing inputs problem

In this paper we have seen that one can 'train' an MLP without knowing any of its inputs at all. The intermediate case in which some inputs are given for some examples should also be solvable with these methods. The distribution over the inputs, fixed by fiat in this paper, would become an adaptive part of the model.

Relationship to autoencoders

One class of MLPs relates closely to density networks: the *autoencoding network* is trained to reproduce the input vector at its output after mapping it through a low-dimensional bottleneck [16, 17]. The density network is like the second 'generative' half of the autoencoder — from the bottleneck to the output. The first mapping in an autoencoder, the 'recognition' mapping from the input to the hidden layer, plays no role in the probabilistic model, but in some applications of density networks it might serve a useful computational purpose; for example the first mapping could be used to learn the appropriate parameters of an importance sampler. Hinton and Zemel describe the use of the recognition mapping to compute an approximate distribution over the latent variables that is used to train an autoencoder by an elegant free energy minimization method [18]. The connection between the 'MDL' approach that they use and the Bayesian viewpoint is explained in ref. [6]. The main differences between this work and Hinton and Zemel's are the types of network studied, and the inclusion in this work of an additional level in the hierarchical model (the hyperparameters $\{\alpha_c\}$), so that it can discover for itself the appropriate dimensionality of the latent space.

Conclusion

By turning autoencoders into explicit probabilistic models, this paper has reaped the benefits of hierarchical Bayesian modelling — automatically inferring from the data what the appropriate dimensionality of the hidden representation is; and thereby (in a toy example at least!) discovering structural properties of a protein from sequence data.

References

[1] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[2] D. J. C. MacKay. The evidence framework applied to classification networks. *Neural Computation*, 4(5):698–714, 1992.

[3] Radford M. Neal. Bayesian learning via stochastic dynamics. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 475–482, San Mateo, California, 1993. Morgan Kaufmann.

[4] H. H. Thodberg. Ace of Bayes: application of neural networks with pruning. Technical Report 1132 E, Danish meat research institute, 1993.

[5] D. J. C. MacKay. Bayesian non-linear modelling for the 1993 energy prediction competition. In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Barbara 1993*, Dordrecht, 1994. Kluwer.

[6] D. J. C. MacKay. Bayesian methods for backpropagation networks. In E. Domany, J. L. van Hemmen, and K. Schulten, editors, *Models of Neural Networks III*, chapter 6. Springer-Verlag, New York, 1994.

[7] D.E. Rumelhart, G. E. Hinton, and R.J. Williams. Learning representations by back–propagating errors. *Nature*, 323:533–536, 1986.

[8] D. J. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.

[9] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In Rumelhart *et al.*, editor, *Parallel Distributed Processing*, pages pp. 282–317. MIT Press, 1986.

[10] B. S. Everitt. *An Introduction to Latent Variable Models.* Chapman and Hall, London, 1984.

[11] J.-P. Nadal and N. Parga. Duality between learning machines: a bridge between supervised and unsupervised learning. *Neural Computation*, 6(3):489–506, 1994.

[12] W.H. Press, B.P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C.* Cambridge, 1988.

[13] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[14] S. R. Eddy and R. Durbin. RNA sequence analysis using covariance models. NAR, in press, 1994.

[15] D. J. C. MacKay and Radford M. Neal. Automatic relevance determination for neural networks. Technical Report in preparation, Cambridge University, 1994.

[16] A. J. Robinson and F. Fallside. Static and dynamic error propagation networks with application to speech coding. In D. Z. Anderson, editor, *Neural Information Processing Systems.* American Institute of Physics, 1988.

[17] E. Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, 1989.

[18] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and Helmholtz free energy. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, San Mateo, California, 1994. Morgan Kaufmann.

I thank Radford Neal, Geoff Hinton, Sean Eddy, Richard Durbin, Tim Hubbard and Graeme Mitchison for invaluable discussions.