UNIVERSITY OF CAMBRIDGE
PROGRAMME FOR INDUSTRY

Modelling Phase Transformations in Steels

Bayesian Non-Linear Modelling with Neural Networks

David J.C. MacKay
Cavendish Laboratory,
Cambridge, CB3 0HE.
`mackay@mrao.cam.ac.uk`

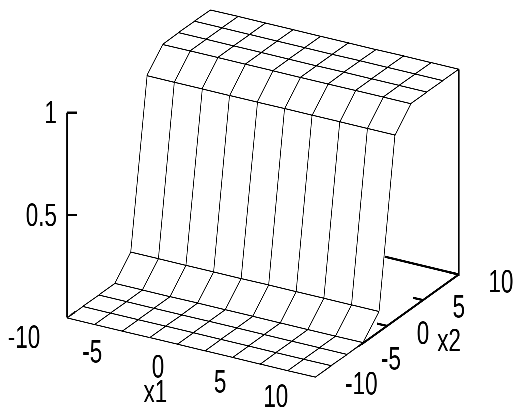
16:00, Tuesday 21 March 1995

Neural networks are parameterized non-linear models used for empirical regression and classification modelling. Their flexibility makes them able to discover more complex relationships in data than traditional statistical models.

Bayesian probability theory provides a unifying framework for data modeling which offers several benefits. First, the **overfitting problem** can be solved by using Bayesian methods to control model complexity. Second, probabilistic modelling handles uncertainty in a natural manner. There is a unique prescription, **marginalization**, for incorporating uncertainty about parameters into predictions. Third, we can define more sophisticated probabilistic models which are able to extract more information from data.

I will discuss the practical application of these methods to time series prediction of a building's energy consumption, and prediction of weld toughness.

Contents



$\mathbf{w} = (0, 2)$

Figure 1: Output of simple neural network as a function of its input

1 Introduction to neural networks

A neural network implements a function $y(\mathbf{x}; \mathbf{w})$; the 'output' of the network, y, is a non-linear function of the 'input' $\mathbf{x}$; this function is parameterized by 'weights' $\mathbf{w}$.

A simple network

A very simple neural network, a neuron with two inputs $\mathbf{x} = (x_1, x_2)$ and two weights $\mathbf{w} = (w_1, w_2)$, produces an output between 0 and 1 as the following function of $\mathbf{x}$:

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}} \tag{1}$$

This example has two important features: (a) a linear operation, $a = (w_1 x_1 + w_2 x_2)$; followed by (b) a non-linear function, $f(a) = 1/(1 + e^{-a})$. Most neural network models share these features, having multiple linear operations alternating with non-linear functions. Figure 1 shows the output of the neuron as a function of the input vector, for $\mathbf{w} = (0, 2)$. The two horizontal axes of this figure are the inputs x_1 and x_2, with the output y on the vertical axis. Notice that on any line perpendicular to $\mathbf{w}$, the output is constant; and along a line in the direction of $\mathbf{w}$, the output is a 'sigmoid' function.

We now introduce the important concept of the **weight space**, that is, the parameter space of the network. In this case, there are two parameters w_1 and w_2, so the weight space is two dimensional. This weight space is shown in figure 2. For a selection of different values of the parameter vector $\mathbf{w}$, smaller inset figures show the function of $\mathbf{x}$ performed by the network when $\mathbf{w}$ is set to those given values. Each of these smaller figures is equivalent to figure 1. Thus each *point* in $\mathbf{w}$ space corresponds to a *function* of $\mathbf{x}$. Notice that the gain of the sigmoid function (the gradient of the ramp) increases as the magnitude of $\mathbf{w}$ increases.

Now, the central idea of the neural network method is this. Given **examples** of a relationship between an input vector $\mathbf{x}$, and a 'target' t, we hope to make the neural network 'learn' a model of the relationship between $\mathbf{x}$ and t. A successfully 'trained' network will, for any given $\mathbf{x}$, give

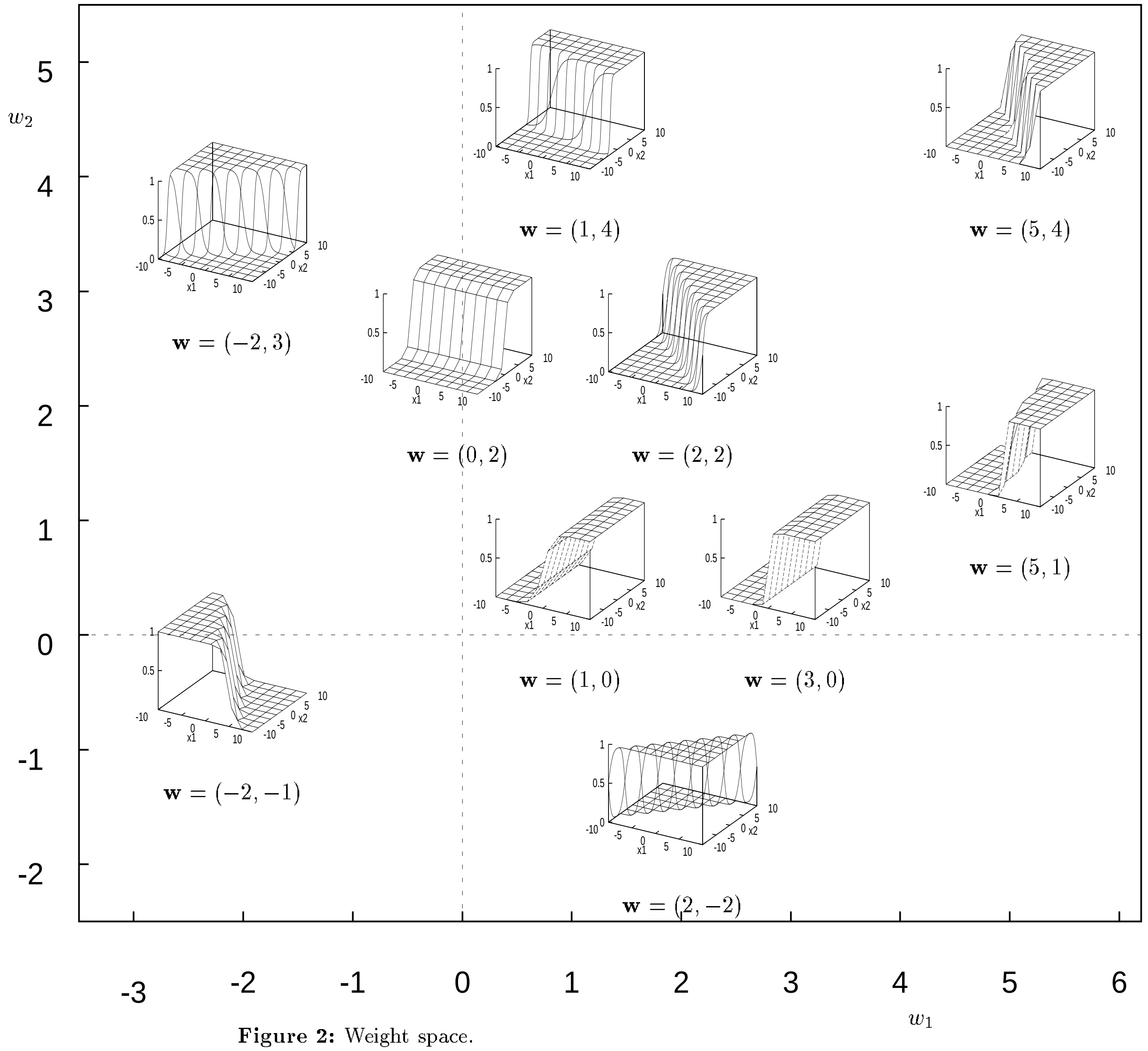


Figure 2: Weight space.

an output y that is close (in some sense) to the target value t. '**Training**' the network involves searching in the weight space of the network for a value of $\mathbf{w}$ that produces a function that fits the provided training data well.

As an example, if we wished to predict the toughness of a weld with a given chemical composition, then we might define the inputs $\mathbf{x}$ to be the chemical composition, *e.g.*, percentage weight of elements, and the output y to be the predicted toughness. The training data would be a set $\{\mathbf{x}, t\}$ of measured weld compositions $\mathbf{x}$ and measured toughnesses t. The hope would be that a trained network would extract a non-linear relationship between the toughness and the input variables. But we would expect the toughness of a weld, as a function of composition, to be a more complex function than any of the functions shown in the weight space of figure (2). We therefore turn to more complex networks.

More complex neural networks

I now describe a popular form of network that can be trained to perform regression tasks. Similar nets can be used for classification problems also and the Bayesian methods described here produce significant benefits for them too (see MacKay (1992b) and MacKay (1995b)); but these notes will concentrate exclusively on regression nets.

The mapping for a *two layer network* has the form:

$$\text{Hidden layer:} \quad a_j^{(1)} = \sum_l w_{jl}^{(1)} x_l + \theta_j^{(1)}; \quad h_j = f^{(1)}(a_j^{(1)}) \tag{2}$$

$$\text{Output layer:} \quad a_i^{(2)} = \sum_j w_{ij}^{(2)} h_j + \theta_i^{(2)}; \quad y_i = f^{(2)}(a_i^{(2)}) \tag{3}$$

where, for example, $f^{(1)}(a) = \tanh(a)$, and $f^{(2)}(a) = a$. Here l runs over the inputs $x_1 \ldots x_L$, j runs over the hidden units, and i runs over the outputs. The 'weights' w and 'biases' θ together make up the parameter vector $\mathbf{w}$. The non-linear 'sigmoid' function $f^{(1)}$ at the hidden layer gives the neural network greater computational flexibility than a standard linear regression model. Graphically, we can represent the neural network as a set of layers of connected neurons. One neuron is shown in figure 3. The entire network is shown in figure 3. This is called a two layer network because it has two layers of weights. In terms of layers of neurons, it has one input layer, one hidden layer and one output layer.

What sorts of functions can these networks implement?

Just as for the simple network we made an exploration of its weight space, examining the functions it could produce, let us make a small exploration of the weight space of a regression network. In figure 4a I take a network with one input and one output and a large number H of hidden units, set the biases and weights $\theta_j^{(1)}$, $w_{jl}^{(1)}$, $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values, and plot the resulting function $y(x)$. I set the hidden unit biases $\theta_j^{(1)}$ to random values from a Gaussian with zero mean and standard deviation σ_{bias}; the input to hidden weights $w_{jl}^{(1)}$ to random values with standard deviation σ_{in}; and the bias and output weights $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values with standard deviation σ_{out}. Figure 4a shows one function for each of a sequence of values of σ_{bias}, σ_{in} and σ_{out}.

The sort of functions that we obtain depend on the values of σ_{bias}, σ_{in} and σ_{out} (Neal 1994). As the weights and biases are made bigger we obtain more complex functions with more features and a greater sensitivity to the input variable. Neal (1994) has also shown that in the limit as $H \to \infty$ the statistical properties of the functions generated by randomizing the weights are independent of the number of hidden units; so, interestingly, the complexity of the functions is independent of the number of parameters in the model. What determines the complexity of the typical functions is the characteristic magnitude of the weights. Thus we anticipate that when we fit these models to real data, an important way of controlling the complexity of the fitted function will be by controlling the characteristic magnitude of the weights.

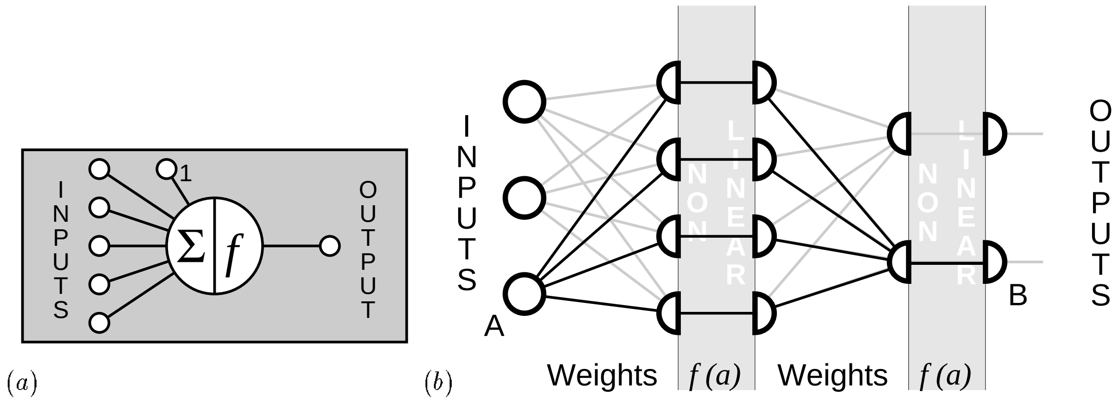


Figure 3: Graphical representation of a neural network.

(*a*) A single neuron. (*b*) A two layer network with 3 inputs, four hidden units and 2 outputs. The weights from input A to the hidden layer have been highlighted, and the weights from the hidden layer to output B.

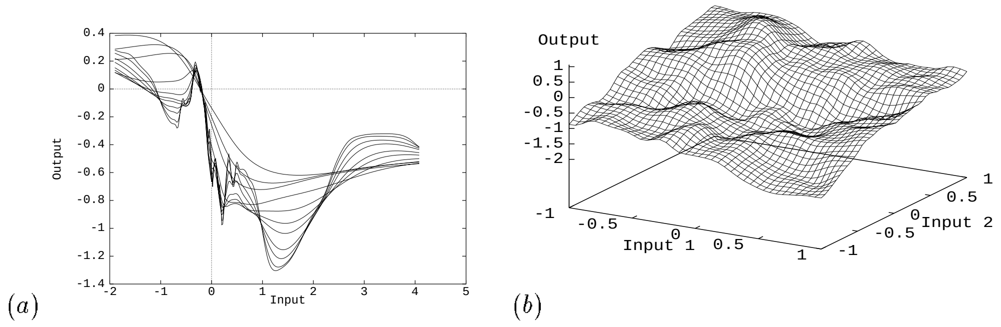


Figure 4: Samples from the prior of (*a*) a one input network; and (*b*) a two input network.

(*a*) Varying $\sigma^w_{\rm bias}$ and $\sigma^w_{\rm in}$ in a one input network: In this figure, $H = 400$, $\sigma^w_{\rm out} = 0.05$. For each graph the parameter $\sigma^w_{\rm bias}$ takes a different value in the sequence: 8, 6, 4, 3, 2, 1.6, 1.2, 0.8, 0.4, 0.3, 0.2. The parameter $\sigma^w_{\rm in}$ was also varied such that $\sigma^w_{\rm in}/\sigma^w_{\rm bias} = 5.0$. The larger values of $\sigma^w_{\rm in}$ and $\sigma^w_{\rm bias}$ produce the more complex functions with more fluctuations.

(*b*) A typical function produced by a two input network with $\{H, \sigma^w_{\rm in}, \sigma^w_{\rm bias}, \sigma^w_{\rm out}\} = \{400, 8.0, 8.0, 0.05\}$.

Figure 4b shows one typical function produced by a network with two inputs and one output. This should be contrasted with the function produced by a traditional linear regression model, which is a flat plane. Clearly neural networks can create functions with much more structure than a linear regression!

How a regression network is traditionally trained

A network as defined in equations (2) and (3) is trained using a data set $D = \{\mathbf{x}^{(m)}, \mathbf{t}^{(m)}\}$ by adjusting $\mathbf{w}$ so as to minimize an error function, *e.g.*,

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_m \sum_i \left(t_i^{(m)} - y_i(\mathbf{x}^{(m)}; \mathbf{w}) \right)^2 . \tag{4}$$

This objective function is a sum of terms, one for each input/target pair $\{\mathbf{x}, t\}$, measuring how close the output $\mathbf{y}(\mathbf{x}; \mathbf{w})$ is to the target t.

This minimization is based on repeated evaluation of the gradient of E_D using 'backpropagation' (Rumelhart, Hinton and Williams 1986). The backpropagation algorithm computes for each input-output pair m the gradient of $\frac{1}{2}\left(t_i^{(m)} - y_i(\mathbf{x}^{(m)}; \mathbf{w})\right)^2$ by following the 'forward pass' of equations (2–3) by a 'backward pass', in which information about the errors $\left(t_i^{(m)} - y_i(\mathbf{x}^{(m)}; \mathbf{w})\right)$ propagates back through the network by the chain rule.

Often, regularization is included, modifying the objective function to:

$$M(\mathbf{w}) = \beta E_D + \alpha E_W \tag{5}$$

where, for example, $E_W = \frac{1}{2} \sum_i w_i^2$. This additional term favours small values of $\mathbf{w}$ and thus encourages the model to find simpler solutions with less tendency to 'overfit' noise in the training data. Regularization is also known as 'weight decay' because the derivative of αE_W with respect to $\mathbf{w}$ is $\alpha \mathbf{w}$. This means that in a simple steepest descent optimizer the effect of the extra term is simply to encourage weights to decay to zero.

2 Neural networks as probabilistic models

A review of probability theory is given in appendix A.

Neural network learning as inference

The neural network learning process above can be given the following probabilistic interpretation. The error function is interpreted as defining the probability distribution of a noise model:

$$P(D|\mathbf{w}, \beta, \mathcal{H}) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D). \tag{6}$$

Thus, the use of the sum-squared error E_D (4) corresponds to an assumption of Gaussian noise on the target variables, and the parameter β defines a noise level $\sigma_\nu^2 = 1/\beta$.

Similarly the regularizer is interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w}|\alpha, \mathcal{H}) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{7}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$. Recall that a moment ago, when we were creating functions with random networks, we set the weights to random Gaussian values. We are thus already familiar with the implicit probabilistic model defined by the quadratic regularizer.

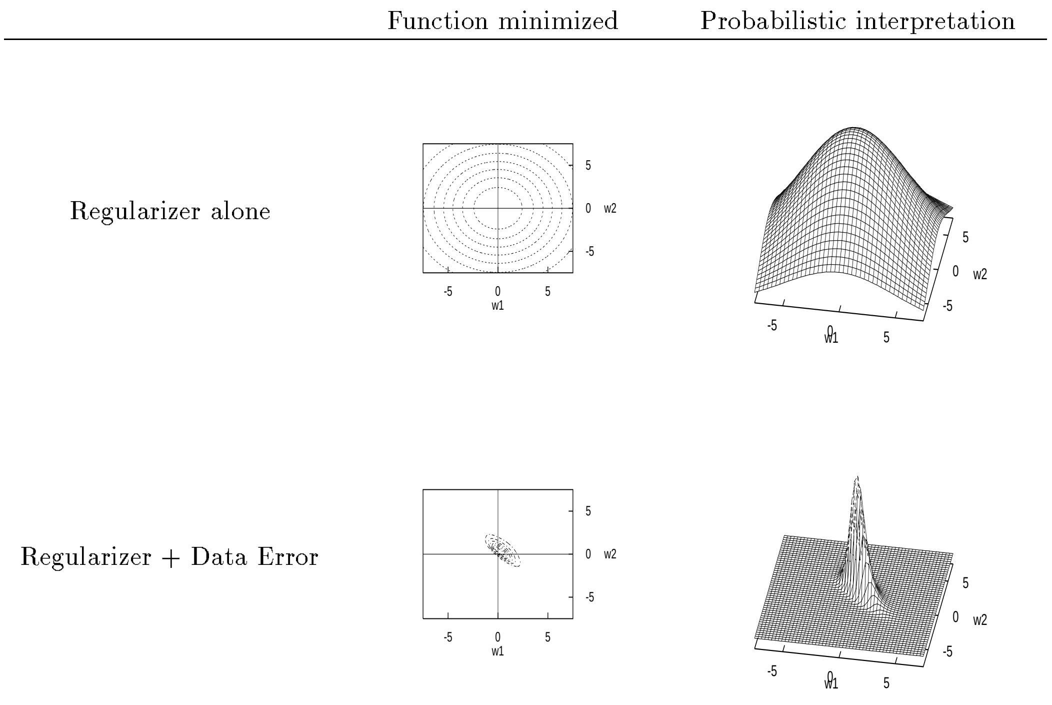


Figure 5: Interpretation of objective functions as probability distributions.

The probabilistic model $\mathcal{H}$ specifies the functional form of the network, the likelihood (6), and the prior (7). The objective function $M(\mathbf{w})$ then corresponds to the **inference** of the parameters $\mathbf{w}$, given the data. By Bayes' theorem:

$$P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}) = \frac{P(D|\mathbf{w},\beta,\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})}{P(D|\alpha,\beta,\mathcal{H})} \tag{8}$$

$$= \frac{1}{Z_M}\exp(-M(\mathbf{w})). \tag{9}$$

The $\mathbf{w}$ found by (locally) minimizing $M(\mathbf{w})$ is then interpreted as the (locally) most probable parameter vector, $\mathbf{w}_{\mathrm{MP}}$.

Why is it natural to interpret the error functions as *log* probabilities? Error functions are usually additive. For example, E_D is a *sum* of squared errors. Probabilities, on the other hand, are multiplicative: for independent events A and B, the joint probability is $P(A,B) = P(A)P(B)$. The logarithmic mapping maintains this correspondence.

To summarise this section: 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 $\mathbf{w}^*$ that minimizes $M(\mathbf{w})$. In contrast, given the Bayesian interpretation of $M(\mathbf{w})$ as a log probability, the product of learning is an *ensemble* of plausible parameter values (figure 5). What can we now do with this view of neural network learning? There are a number of useful consequences, relating to the automatic control of neural network complexity, and to the quantification of the uncertainty of a model's predictions. These benefits will now be described, using a set of computational short cuts based on Gaussian approximations. An alternative class of computational techniques for implementing Bayesian neural nets are Monte Carlo methods which have been thoroughly developed by Neal (1993).

The Gaussian approximation

The posterior probability distribution $P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}) \propto \exp(-M(\mathbf{w}))$, which is maximized in the traditional approach to neural net learning, is typically a complex multimodal distribution in weight space. Each local maximum corresponds to a different interpretation of the data. One way to approximate this distribution, which relates in a straightforward way to traditional algorithms that optimize $M(\mathbf{w})$, is to find as many optima $\mathbf{w}_{\rm MP}$ as possible and to make a Gaussian approximation,

$$P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}) \simeq \frac{1}{Z'_M}\exp\left(-M(\mathbf{w}_{\rm MP}) - \frac{1}{2}(\mathbf{w}-\mathbf{w}_{\rm MP})^{\rm T}\mathbf{A}(\mathbf{w}-\mathbf{w}_{\rm MP})\right), \tag{10}$$

at each maximum. The matrix $\mathbf{A}$ is matched to the local curvature of $M(\mathbf{w})$ by setting $\mathbf{A} = -\nabla\nabla \log P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}) = \nabla\nabla M(\mathbf{w})$. [Here ∇ denotes differentiation with respect to $\mathbf{w}$, which is what the backpropagation algorithm is all about. Details of how to compute $\mathbf{A}$ are given elsewhere, *e.g.*, (MacKay 1992c, Bishop 1992).] The inverse $\mathbf{A}^{-1}$ of this matrix defines the covariance matrix of the Gaussian approximation. It defines error bars on the parameters $\mathbf{w}$, which are important to the methods that follow.

3 Complexity control

The overfitting problem

Consider a control parameter which influences the complexity of a model (for example a regularization constant α). As the control parameter is varied to increase the complexity of the model (descending from figure 6*a*-*c* and going from left to right across figure 6*d*), the best fit to the **training** data that the model can achieve becomes increasingly good. However, the empirical performance of the model, the **test error**, has a minimum as a function of the control parameters. *An over-complex model overfits the data and generalizes poorly.* This problem may also complicate the choice of architecture in a multilayer perceptron, the radius of the basis functions in a radial basis function network, and the choice of the input variables themselves in any regression problem. Finding values for model control parameters that are appropriate for the data is therefore an important and non-trivial problem.

A central message of this paper is illustrated in figure 6*e*. If we give a probabilistic interpretation to the model, then we can evaluate the 'evidence' for alternative values of the control parameters. Over-complex models turn out to be less probable, and the quantity $P(\text{Data}|\text{Control Parameters})$ can be used as an objective function for optimization of model control parameters. The setting of α that maximizes this quantity is displayed in figure 6*b*.

Bayesian optimization of model control parameters has four important advantages. (1) No 'test set' or 'validation set' is involved, so all available training data can be devoted to both model fitting and model comparison. (2) Regularization constants can be optimized on-line, *i.e.*, simultaneously with the optimization of ordinary model parameters. (3) The Bayesian objective function is not noisy, in contrast to a cross-validation measure. (4) The gradient of the evidence with respect to the control parameters can be evaluated, making it possible to simultaneously optimize a large number of control parameters.

Setting regularization constants α and β

The control parameters α and β in equation (5) determine the complexity of the model. The term model here refers to a triple: the network architecture; the form of the prior on the parameters; and the form of the noise model. Different values for the hyperparameters α and β define different sub-models. To infer α and β given the data, we simply apply Bayes' theorem:

$$P(\alpha,\beta|D,\mathcal{H}) = \frac{P(D|\alpha,\beta,\mathcal{H})P(\alpha,\beta|\mathcal{H})}{P(D|\mathcal{H})}. \tag{11}$$

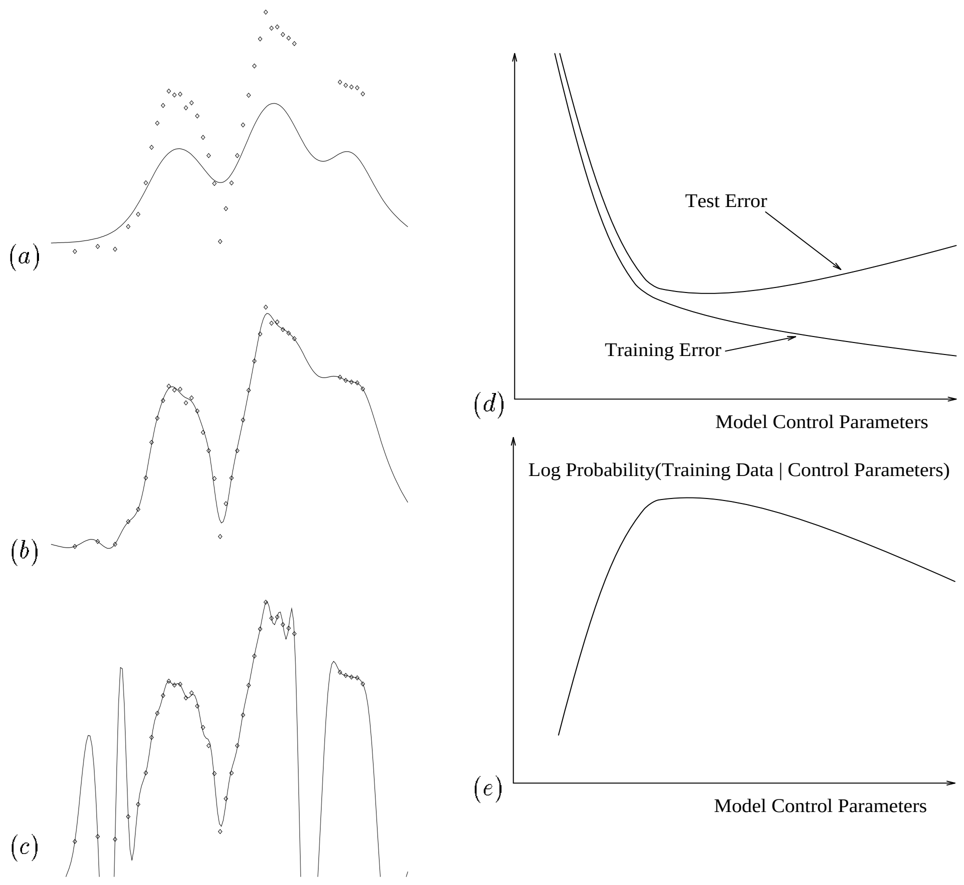


Figure 6: Optimization of model complexity. Figures *a*-*c* show a radial basis function model interpolating a simple data set with one input variable and one output variable. As the regularization constant is varied to increase the complexity of the model (from *a* to *c*), the interpolant is able to fit the training data increasingly well, but beyond a certain point the generalization ability (test error) of the model deteriorates. Probability theory allows us to optimize the control parameters without needing a test set.

The data-dependent factor $P(D|\alpha,\beta,\mathcal{H})$ is the normalizing constant from our previous inference (8); we call this factor the 'evidence' for α and β.

Assuming we have only weak prior knowledge about the noise level and the smoothness of the interpolant, the evidence framework optimizes the constants α and β by finding the maximum of the evidence for α and β. If we can approximate the posterior probability distribution in equation (9) by a single Gaussian as in equation (10), then the evidence for α and β can be written down:

$$\begin{aligned}\log P(D|\alpha,\beta,\mathcal{H}) &= \log \frac{Z'_M}{Z_W(\alpha)Z_D(\beta)} && (12)\\ &= -M(\mathbf{w}_{\rm MP}) - \frac{1}{2}\log\det(\mathbf{A}/2\pi) - \log Z_W(\alpha) - \log Z_D(\beta). && (13)\end{aligned}$$

The maximum of the evidence has some elegant properties which allow it to be located efficiently by on-line re-estimation techniques. As shown in (Gull 1989, MacKay 1992a), the maximum evidence $\alpha = \alpha_{\rm MP}$ satisfies the following implicit equation:

$$1/\alpha_{\text{MP}} = \sum_i w_i^{\text{MP}2}/\gamma \tag{14}$$

where $\mathbf{w}^{\text{MP}}$ is the parameter vector which minimizes the objective function $M = \beta E_D + \alpha E_W$, and γ is the 'number of well-determined parameters', given by

$$\gamma = k - \alpha \text{Trace}(\mathbf{A}^{-1}). \tag{15}$$

Here k is the total number of parameters, and the matrix $\mathbf{A}^{-1}$ describes the error bars on the parameters $\mathbf{w}$ (equation (10)). Thus $\gamma \rightarrow k$ when the parameters are all well-determined in relation to their prior range, which is defined by α. Similarly, in a regression problem with a Gaussian noise model, the maximum evidence value of β satisfies:

$$1/\beta_{\text{MP}} = 2E_D/(N - \gamma). \tag{16}$$

Equations (14) and (16) can be used as re-estimation formulae for α and β. The computational overhead for these Bayesian calculations is not severe: it is only necessary to evaluate properties of the error bar matrix, $\mathbf{A}^{-1}$. One might argue that this matrix should be evaluated anyway, in order to compute the uncertainty in a model's predictions. This matrix may be evaluated explicitly (MacKay 1992c, Bishop 1992), which does not take significant time when the number of parameters is small (a few hundred). For large problems these calculations can be performed more efficiently using algorithms which evaluate products $\mathbf{Av}$ without explicitly evaluating $\mathbf{A}$ (Skilling 1993, Pearlmutter 1994).

Multiple regularization constants and 'automatic relevance determination'

For simplicity, it has so far been assumed that there is only a single class of weights, which are modelled as coming from a single Gaussian prior with $\sigma_W^2 = 1/\alpha$. However, in dimensional terms, weights usually fall into three or more distinct groups, which for consistency should not be modelled as coming from a single prior. It is therefore desirable to divide the parameters into several classes c, with independent scale parameters α_c. Furthermore, if we separate the weights associated with each *input* into a class of their own, then each hyperparameter α_c can be viewed as describing the relevance of each input. The weights of an input with a large value of α_c have more of a tendency to decay to zero, so such an input is not as significant in the regression. From where can we get values for all these hyperparameters? How can we determine which are the relevant input variables for our regression? The answer is, of course, by Bayesian inference. Just as in the previous section we optimized a single regularization constant α by maximizing the evidence, we can derive equivalent prescriptions for optimizing multiple regularization constants. The formulae are similar to equations (14) and (15), and are given for example in (MacKay 1995b). For simplicity, the following discussion will assume once more that there is only a single parameter α.

Model comparison

The evidence framework divides our inferences into distinct 'levels of inference', of which we have now completed the first two.

- **Level 1:** Infer the parameters $\mathbf{w}$ for given values of α, β:

$$P(\mathbf{w}|D, \alpha, \beta, \mathcal{H}) = \frac{P(D|\mathbf{w}, \alpha, \beta, \mathcal{H}) P(\mathbf{w}|\alpha, \beta, \mathcal{H})}{P(D|\alpha, \beta, \mathcal{H})}. \tag{17}$$

- **Level 2:** Infer α, β:

$$P(\alpha, \beta|D, \mathcal{H}) = \frac{P(D|\alpha, \beta, \mathcal{H}) P(\alpha, \beta|\mathcal{H})}{P(D|\mathcal{H})}. \tag{18}$$

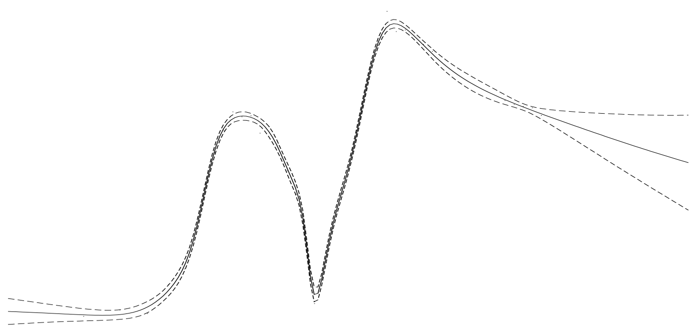


Figure 7: Error bars on the predictions of a trained regression network. The solid line gives the predictions of the best fit parameters of a multilayer perceptron trained on the data points (shown by squares). The error bars (dotted lines) are those produced by the uncertainty of the parameters **w**. Notice that the error bars become larger where the data are sparse.

- **Level 3:** Compare models:

$$P(\mathcal{H}|D) \propto P(D|\mathcal{H})P(\mathcal{H}). \tag{19}$$

There is a pattern in these three applications of Bayes' rule: at each of the higher levels 2 and 3, the data-dependent factor (*e.g.* in level 2, $P(D|\alpha, \beta, \mathcal{H})$) is precisely the normalizing constant (the 'evidence') from the preceding level of inference. This pattern of inference continues when we compare different models $\mathcal{H}$, which might use different architectures, preprocessings, regularizers or noise models. Alternative models are ranked by evaluating $P(D|\mathcal{H})$, the normalizing constant of inference (18). The Gaussian approximation can be used to calculate $P(D|\mathcal{H})$ and compare alternative solutions, but unfortunately this approximation of the evidence is often not sufficiently accurate, particularly with large networks. It is therefore wise to use other methods for comparing different solutions in parallel with Bayesian approximations—for example 'cross validation' evaluates the predictive performance of each solution on some set aside *validation data*.

4 Error bars and predictions

Having progressed up the three levels of modelling, the next inference task is to make predictions with our adapted model. It is common practice simply to use the most probable values of $\mathcal{H}, \mathbf{w}$, etc., when making predictions, but this is not optimal. Bayesian prediction of a new datum $\mathbf{t}^{(N+1)}$ involves *marginalizing* over all these levels of uncertainty:

$$P(\mathbf{t}^{(N+1)}|D) = \sum_{\mathcal{H}} \int d\alpha \, d\beta \int d^k\mathbf{w} \, P(\mathbf{t}^{(N+1)}|\mathbf{w}, \alpha, \beta, \mathcal{H})P(\mathbf{w}, \alpha, \beta, \mathcal{H}|D). \tag{20}$$

The evaluation of the distribution $P(\mathbf{t}^{(N+1)}|\mathbf{w}, \alpha, \beta, \mathcal{H})$ for specified model parameters $\mathbf{w}$ is generally straightforward, requiring a single forward pass through the network. Typically, marginalization over $\mathbf{w}$ and $\mathcal{H}$ affects the predictive distribution significantly, but integration over α and β has a lesser effect.

Error bars in regression

Integrating first over $\mathbf{w}$ for fixed α and β, the predictive distribution is:

$$P(\mathbf{t}^{(N+1)}|D,\alpha,\beta,\mathcal{H}) = \int d^k\mathbf{w}\, P(\mathbf{t}^{(N+1)}|\mathbf{w},\beta,\mathcal{H})P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}). \tag{21}$$

If a Gaussian approximation (10) is made for the posterior $P(\mathbf{w}|D,\alpha,\beta,\mathcal{H})$, with covariance matrix $\mathbf{A}^{-1}$; if the noise model is Gaussian; and if a local linearization of the output is made as a function of the parameters,

$$y(\mathbf{x}^{N+1},\mathbf{w}) \simeq y(\mathbf{x}^{N+1};\mathbf{w}_{\mathrm{MP}}) + \mathbf{g}\cdot(\mathbf{w}-\mathbf{w}_{\mathrm{MP}}), \tag{22}$$

with $\mathbf{g} = \partial y/\partial\mathbf{w}$; then the predictive distribution (21) is a straightforward Gaussian integral. This distribution has mean $y(\mathbf{x}^{N+1},\mathbf{w}_{\mathrm{MP}})$, and variance $\sigma^2_{t|\alpha,\beta} = \mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g} + \sigma^2_\nu$.

Joint error bars on multiple predictions.

Often one wishes simultaneously to predict several targets $\{t^{(u)}\}_{u=1}^{U}$, when a single network has multiple outputs, or when we have a sequence of different inputs and wish to predict all the corresponding targets, or both. Let the sensitivities of these outputs to the parameters be $\mathbf{g}_{(u)} \equiv \frac{\partial y^{(u)}}{\partial \mathbf{w}}$. Then the covariance matrix of the values $\{y^{(u)}\}$ is

$$\mathbf{Y} = \mathbf{G}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{G}, \tag{23}$$

where the matrix $\mathbf{G} = \left[\mathbf{g}_{(1)}\mathbf{g}_{(2)}\ldots\mathbf{g}_{(U)}\right]$. This matrix expresses, within the Gaussian approximation, the expected variances and covariances of the predicted variables. These correlated error ellipsoids have been demonstrated for a two output regression network in MacKay (1992c).

Integrating over models: committees

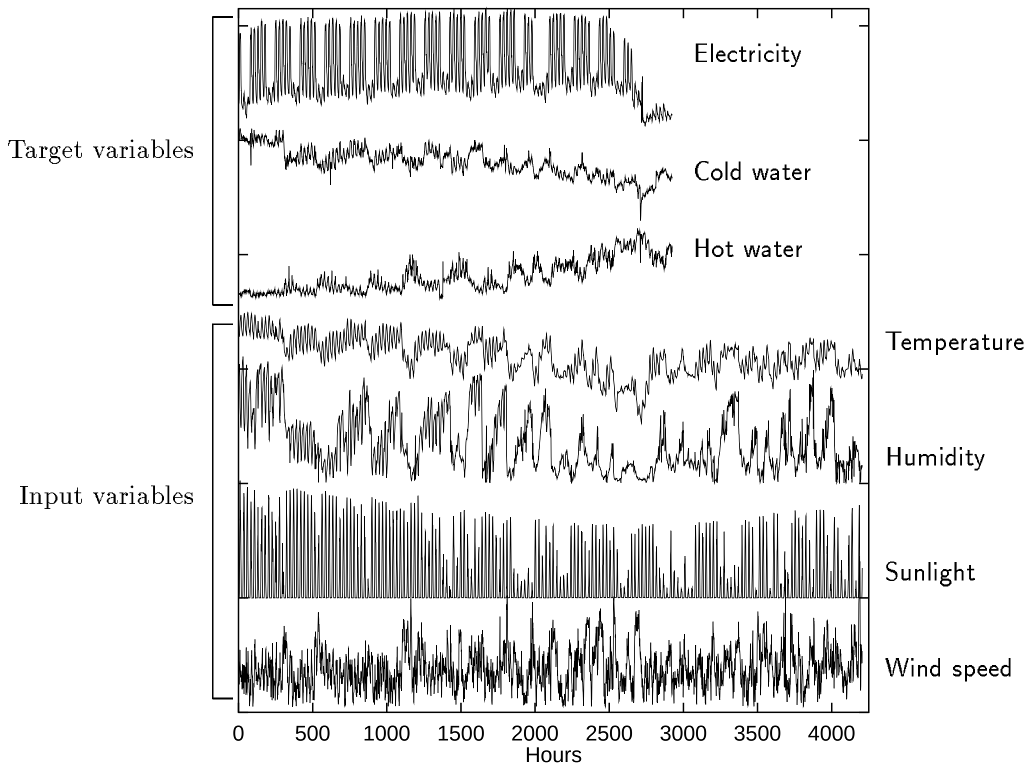
If we have multiple regression models $\mathcal{H}$, then our predictive distribution is obtained by summing together the predictive distribution of each model, weighted by its posterior probability. If a single prediction is required and the loss function is quadratic, the optimal prediction is a weighted mean of the models' predictions $y(\mathbf{x}^{N+1};\mathbf{w}_{\mathrm{MP}},\mathcal{H})$. The weighting coefficients are the posterior probabilities, which are obtained from the evidences $P(D|\mathcal{H})$. If we cannot evaluate these accurately then alternative pragmatic prescriptions for the weighting coefficients exist (Breiman 1992) (see appendix B for further details).

5 Prediction competition

The American Society of Heating, Refrigeration and Air Conditioning Engineers organised a prediction competition in 1993. The competition had two parts, both of which involved creating an empirical model based on training data (as distinct from a physical model), and making predictions for a test set. Part A, which I describe here, involved three target variables, and the test set came from a different time period from the training set, so that extrapolation was involved.

The task

The training set (figure 8) consisted of hourly measurements from September 1 1989 to December 31 1989 of four input variables (temperature, humidity, solar flux and wind), and three target variables (electricity, cooling water and heating water) — 2926 data points for each target. The testing set consisted of the input variables for the next 54 days — 1282 points. The organizers requested predictions for the test set; no error bars on these predictions were requested. The performance measures for predictions were the Coefficient of Variation ('CV', a sum squared error measure normalized by the data mean), and the mean bias error ('MBE', the average residual normalized by the data mean).



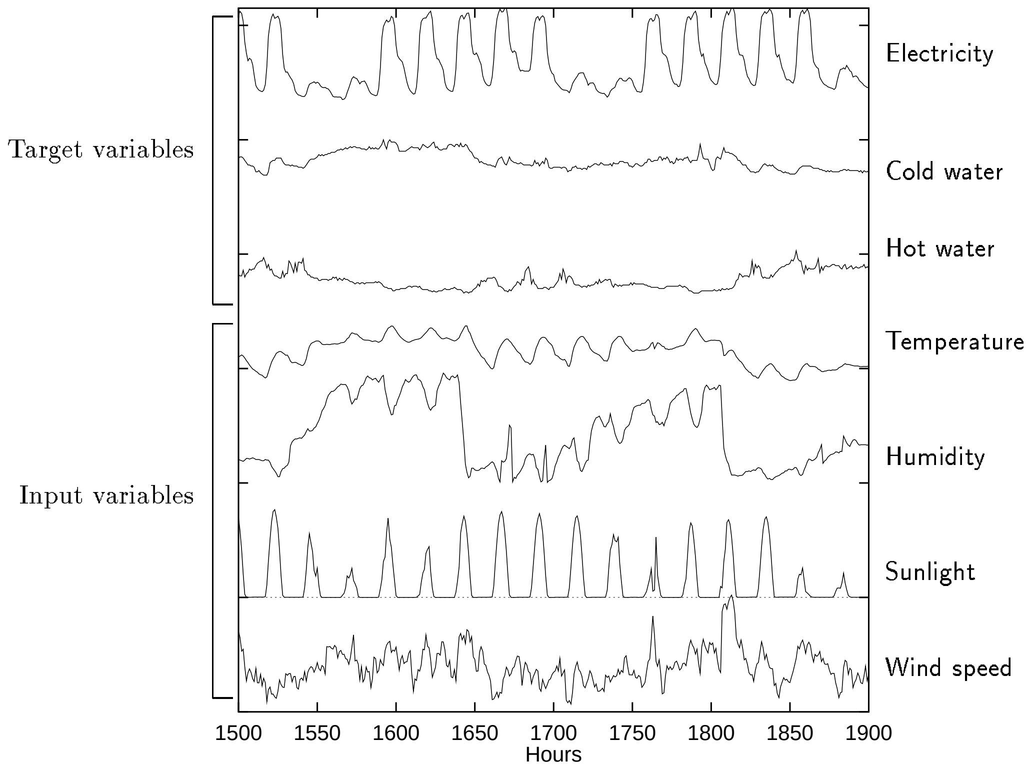


Figure 8: The Prediction Competition Data
a) The entire training set. b) Detail.

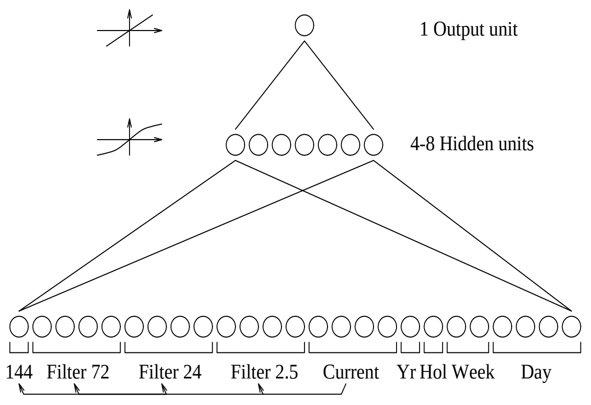


Figure 9: A typical network used for problem A.

The filters produced moving averages of the four environmental inputs on three time-scales: 2.5, 24 and 72 hours. The temperature variable was also given a 144 hour filter. Time was represented using the cos of the year angle, a holiday indicator, and the cos and sin of: the week angle, the day angle, and twice the day angle. All hidden and output units also had a connection to a bias unit (not shown).

Method

I trained a large number of neural nets using the ARD model, for each of the prediction problems. The data seemed to include some substantial glitches. Because I had not yet developed an automatic Bayesian noise model that anticipates outliers (though this could be done (Box and Tiao 1973)), I omitted by hand those data points which gave large residuals relative to the first models that were trained. These omitted periods are indicated on some of the graphs in these notes. 25% of the data was selected at random as training data, the remainder being left out (a) to speed the optimizations, and (b) for use as a validation set. All the networks had a single hidden layer of tanh units, and a single linear output (figure 9). It was found that models with between 4 and 8 hidden units were appropriate for these problems.

A large number of inputs were included: different temporal preprocessings of the environmental inputs, and different representations of time and holidays. All these inputs were controlled by the ARD model. ARD proved a moderately useful guide for decisions concerning preprocessing of the data, in particular, how much time history to include. Moving averages of the environmental variables were created using filters with a variety of exponential time constants. This was thought to be a more appropriate representation than time delays, because (a) filters suppress noise in the input variables, allowing one to use fewer filtered inputs with long time constant; (b) with exponentially filtered inputs it is easy to create (what I believe to be) a natural model, giving equal status to filters having timescales 1, 2, 4, 8, 16, etc..

The on–line optimization of regularization constants was very successful. For problem A, 28 such control constants were simultaneously optimized in every model. Most models did not show 'overtraining' as the optimization proceeded, so 'early stopping' was not generally used. The numerical evaluation of the 'evidence' for the models proved problematic, so validation errors were used to rank the models for prediction. For each task, a committee of models was assembled, and their predictions were averaged together; this procedure was intended to mimic the Bayesian predictions $P(\mathbf{t}|D) = \int P(\mathbf{t}|D,\mathcal{H})P(\mathcal{H}|D)\, d\mathcal{H}$. The size of the committee was chosen so as to minimize the validation error of the mean predictions. This method of selecting committee size has also been described under the name 'stacked generalization' (Breiman 1992). In all cases, a committee was found that performed significantly better on the validation set than any individual model.

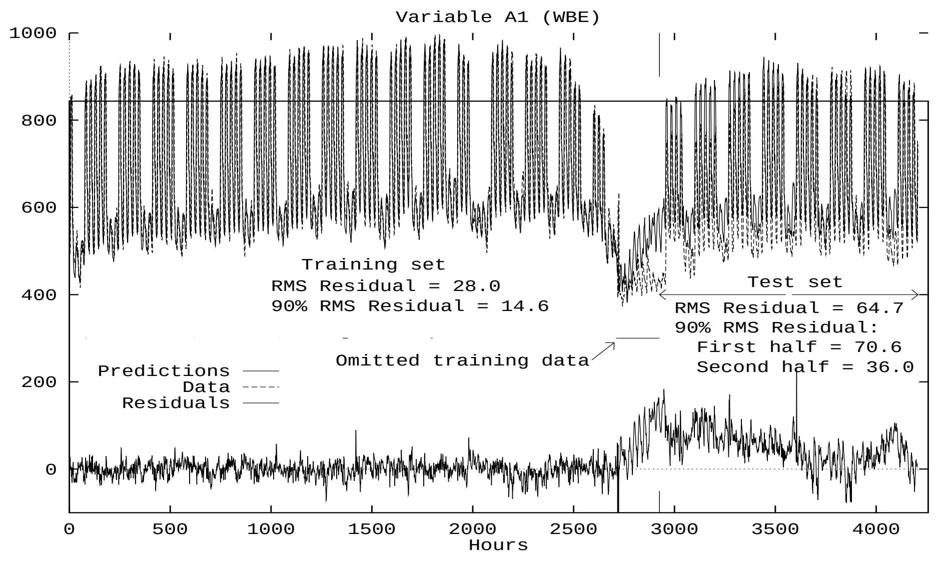


Figure 10: Target A1 — Electricity

Results

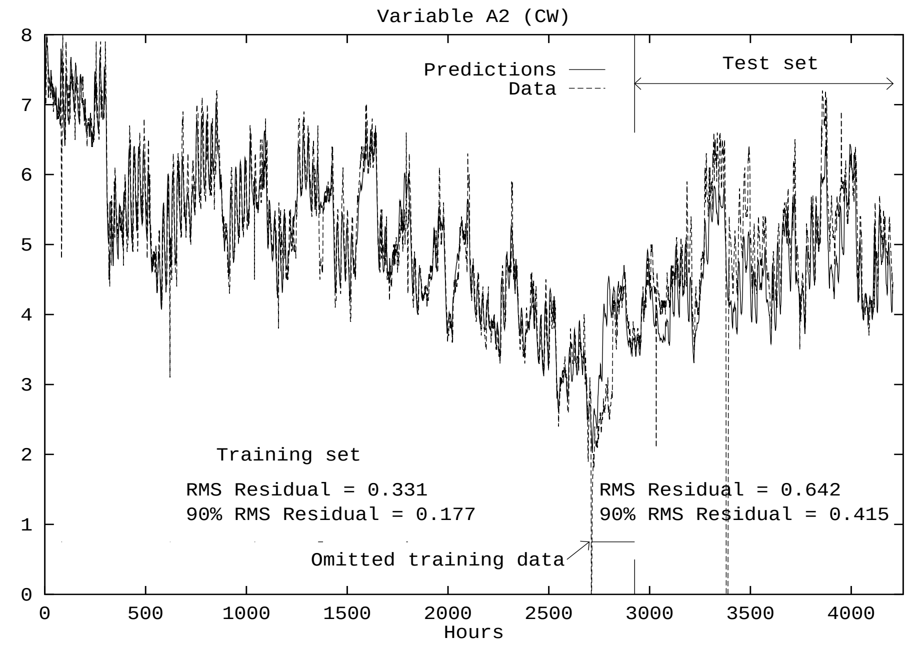
The three target variables are displayed in their entirity, along with my models' final predictions and residuals, in figures 10–12.

There are local trends in the testing data which the models were unable to predict. Such trends were presumably 'overfitted' in the training set. Clearly a model incorporating local correlations among residuals is called for. Such a model would not perform much better by the competition criteria, but its on-line predictive performance would be greatly enhanced.

In the competition rules, it was suggested that scatter plots of the model predictions versus temperature should be made. The scatter plot for problem A3 is particularly interesting. Target A3 showed a strong correlation with temperature in the training set (dots in figures 13b). When I examined my models' predictions for the testing set, I was surprised to find that, for target A3, a significantly offset correlation was predicted ('+'s in figure 13a). This change in correlation turned out to be correct ('+'s in figure 13b). This indicates that these non-linear models controlled with Bayesian methods discovered non-trivial underlying structure in the data. Most other entrants' predictions for target A3 showed a large bias; presumably none of their models extracted the same structure from the data.

In the models used for problem A3, I have examined the values of the parameters $\{\alpha_c, \gamma_c\}$, which give a qualitative indication of the inferred 'relevance' of the inputs. For prediction of the hot water consumption, the time of year and the current temperature were the most relevant variables. Also highly relevant were the holiday indicator, the time of day, the current solar and wind speed, and the moving average of the temperature over the last 144 hours. The current humidity was not relevant, but the moving average of the humidity over 72 hours was. The solar was relevant on a timescale of 24 hours. None of the 2.5 hour filtered inputs seemed especially relevant.

After the competition, it was revealed that the building in this study was a large university engineering center in Texas. Some of the glitches in the data were caused by the bursting of cold water pipes during a frost — a rare event apparently not anticipated by Texan architects!



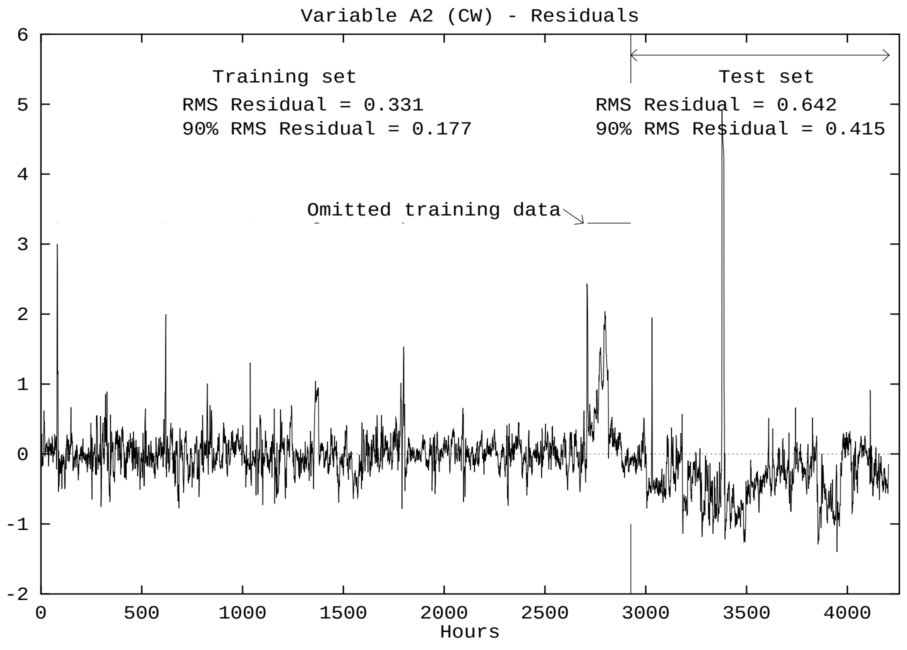
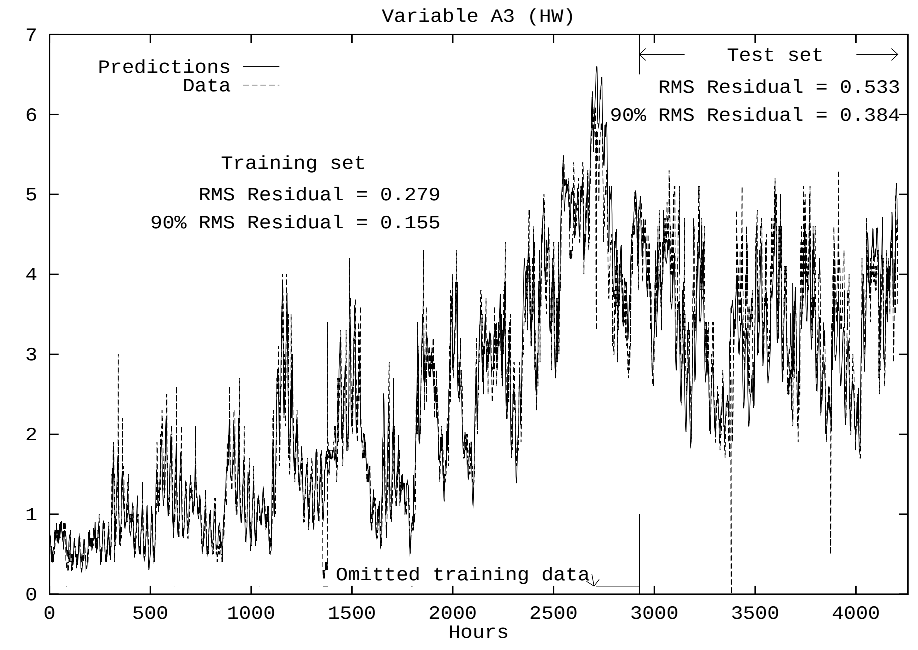


Figure 11: Target A2 — Cooling water



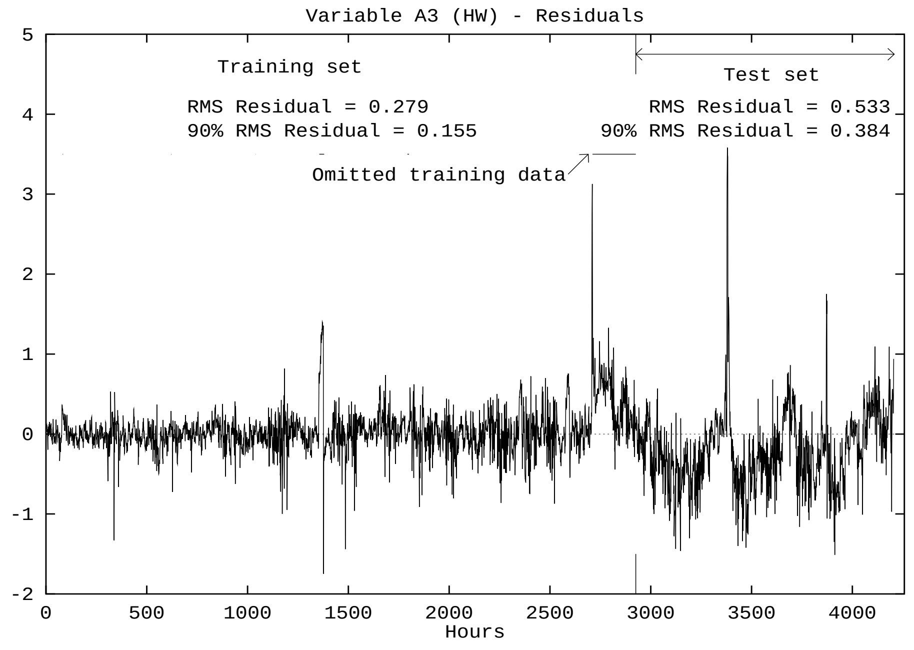


Figure 12: Target A3 — Heating water

Problem A1	RMS	Mean	**CV**	MBE	$\text{RMS}_{90\%}$	$\text{Mean}_{90\%}$	**RCV**
ARD	64.7	50.3	**10.3**	8.1	54.1	42.2	**11.1**
ARD off	71.2	56.2	**11.4**	9.0	59.3	47.3	**12.2**
Entrant 6			**11.8**	10.5			
Median			**16.9**	-10.4			
Problem A2	RMS	Mean	**CV**	MBE	$\text{RMS}_{90\%}$	$\text{Mean}_{90\%}$	**RCV**
ARD	.642	-.314	**13.0**	-6.4	.415	-.296	**11.2**
ARD off	.668	-.367	**13.5**	-7.4	.451	-.349	**12.2**
Entrant 6			**13.0**	-5.9			
Median			**14.8**	-7.6			
Problem A3	RMS	Mean	**CV**	MBE	$\text{RMS}_{90\%}$	$\text{Mean}_{90\%}$	**RCV**
ARD	.532	-.204	**15.2**	-5.8	.384	-.167	**9.15**
ARD off	.495	-.121	**14.2**	-3.5	.339	-.094	**8.08**
Entrant 6			**30.6**	-27.3			
Median			**31.0**	-27.0			

Key:

My models:

ARD	The predictions entered in the competition using the ARD model.
ARD off	Predictions obtained using derived models with the standard regularizer.

Other entries:

Entrant 6	The entry which came 2nd by the competition's average CV score.
Median	Median (by magnitude) of scores of all entries in competition.

Raw Performance measures:

RMS	Root mean square residual.
Mean	Mean residual.
CV	Coefficient of variation (percentage). The competition performance measure.
MBE	Mean Bias Error (percentage).

Robust Performance measures:

$\text{RMS}_{90\%}$	Root mean square of the smallest 90% of the residuals.
$\text{Mean}_{90\%}$	Mean of those residuals.
RCV	$\text{RMS}_{90\%}$/(90% data range).

Normalizing constants:

Problem	Mean of test data	90% data range
A1	624.77	486.79
A2	4.933	3.7
A3	3.495	4.2

Table 1: Performances of different methods on test sets

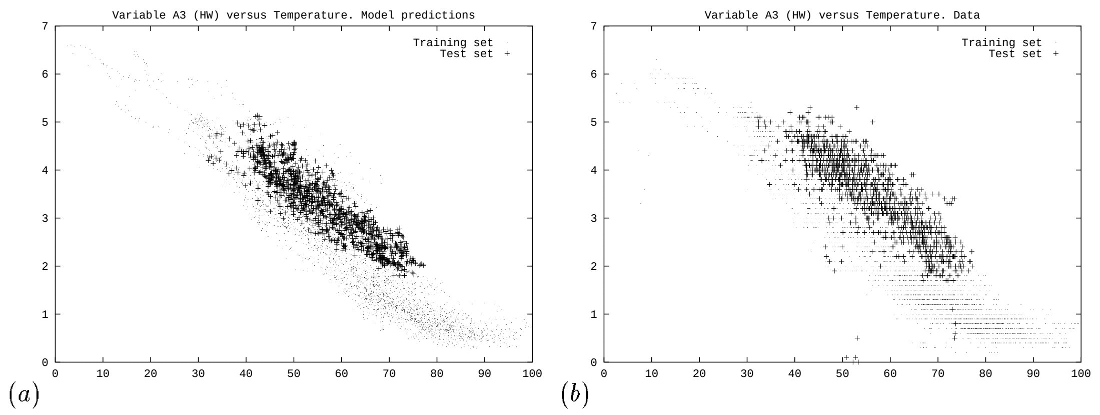


Figure 13: Predictions for target A3 (HW) versus temperature

(*a*) Model predictions. This graph shows that my model predicted a substantially different correlation between target A3 and temperature (+) from that shown in the training set (·). (*b*) Data. This predicted offset was correct.

Discussion

The ARD prior was a success because it made it possible to include a large number of inputs without fear of overfitting. Further work could be well spent on improving the noise model, which assumes the residuals are Gaussian and uncorrelated from frame to frame. A better predictive model for the residuals shown in figures 10–12 might represent the data as the sum of the neural net prediction and an unpredictable, but auto-correlated, additional disturbance. Also, a robust Bayesian noise model is needed which captures the concept of outliers.

It should be emphasised that this was not a fully automated modelling method. Human interaction was essential, and common sense was needed to check whether the assumptions were appropriate and intervene when not (see appendices B and C). Having said this, the Bayesian framework allowed for far more automation than is possible in a traditional neural network approach. Specifically, because appropriate priors (regularizers / weight decay) are used, there is no need to worry about 'early stopping'. And the problem of variable selection can be solved by using the ARD prior.

In conclusion, the winning entry in this competition was created using the following data modelling philosophy: use huge flexible models, including all possibilities that you can imagine might be appropriate; control the flexibility of these models using sophisticated priors; and use Bayes as a helmsman to guide the search in this model space.

6 Prediction of weld toughness

The toughness of a steel depends on many variables, and that of a weld on many more, because of the complexity of the welding process. The Charpy test can be used to characterize the toughness of a weld. We (Bhadeshia, MacKay and Svensson (1995)) have developed an empirical non-linear model of the Charpy toughness as a function of the composition (percentage by weight of C, Si, Mn, P, S, Al, N, O), the yield strength, the temperature at which the weld was formed, and the percentages of primary microstructure, secondary microstructure, allotriomorphic ferrite, acicular ferrite and Widmanstätten ferrite.

The same probabilistic model ('ARD') was used as in the prediction competition work. 181 data points were divided into 100 training examples and 81 test examples. Figure 14 shows, for

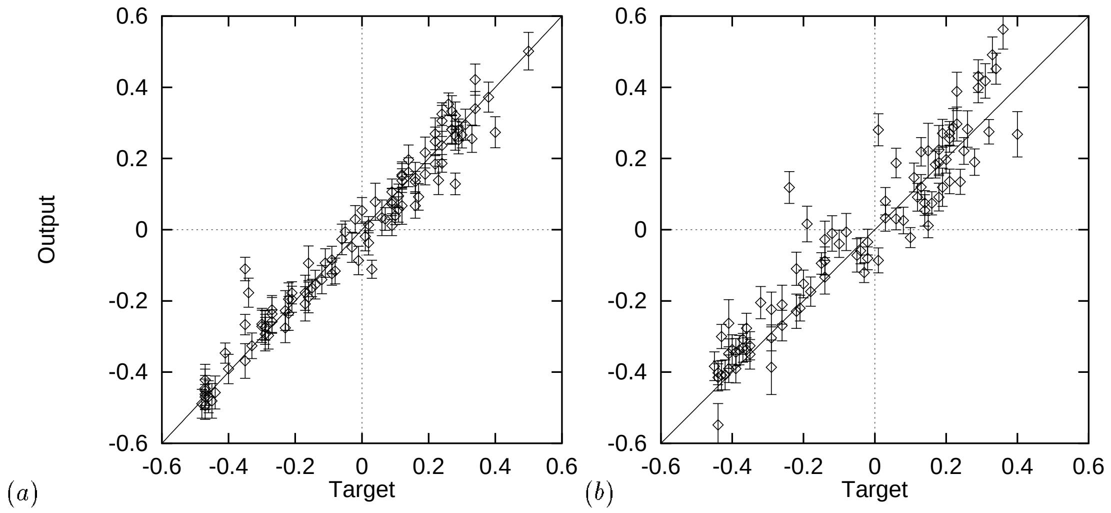


Figure 14: Performance on weld data. Scatter plot of predicted versus actual toughness. (a) training set. (b) test set.

a single selected model having four hidden units,[1] a scatter plot of the predicted versus actual toughness on the training data (left) and on the test data (right). The error bars are plus or minus one standard deviation predictive error bars. Although there are a few test cases where the model's prediction is badly wrong, in most test cases the predictions are within 15% of the actual value, and the errors are compatible with the predictive error bars.

For further discussion of interesting non-linear effects discovered by this model, see Bhadeshia et al. (1995).

In this demonstration, we have cheated slightly in that we used the error on the test set to identify this model as the best model, and we then displayed its performance on the same test data. Ideally we would have had more data and ranked the models on a *validation* set before finally evaluating the performance on a separate test set (see appendix B).

Appendices

A Probability theory

Bayesian probability theory provides a unifying framework for data modelling. A Bayesian data-modeller's aim is to develop probabilistic models that are well-matched to the data, and make optimal predictions using those models. The Bayesian framework has several advantages.

Probability theory forces us to make explicit all our modelling assumptions, whereupon our inferences are mechanistic. Once a model space has been defined, then, whatever question we wish to pose, the rules of probability theory give a unique answer which consistently takes into account all the given information. This is in contrast to orthodox (also known as 'frequentist' or 'sampling theoretical') statistics, in which one must invent 'estimators' of quantities of interest and then choose between those estimators using some criterion measuring their sampling properties; there is no clear principle for deciding which criterion to use to measure the performance of an estimator; nor, for most criteria, is there any systematic procedure for the construction of optimal estimators.

Bayesian inference satisfies the likelihood principle (Berger 1985): our inferences depend only on the probabilities assigned to the data that were received, not on properties of other data sets

[1] One would expect better predictions to be obtained by combining the predictions of several good models.

which might have occurred but did not.

Probabilistic modelling handles uncertainty in a natural manner. There is a unique prescription (marginalization) for incorporating uncertainty about parameters into our predictions of other variables.

Finally, Bayesian model comparison embodies **Occam's razor**, the principle that states a preference for simple models. This point is expanded on in MacKay (1992a) and other references.

Joint, conditional and marginal probabilities

The language of coherent inference is probability theory (Cox 1946). All coherent beliefs and predictions can be mapped onto probabilities. I will use the following notation for *conditional* probabilities: $P(A|B,\mathcal{H})$ is pronounced 'the probability of A, given B and $\mathcal{H}$'. The statements B and $\mathcal{H}$ list the conditional assumptions on which this measure of plausibility is based. For example, imagine that Joe has a test for a nasty disease; if A is the proposition "the test result is positive", and B is "Joe has the disease", then the quantity $P(A|B,\mathcal{H})$ is a number between 0 and 1 which expresses how likely we think the test would be to give the right answer, assuming that Joe did have the disease, and given the overall assumptions $\mathcal{H}$ about the reliability of the test.

There are two rules of probability. The *product rule* relates the *joint* probability of A *and* B, $P(A,B|\mathcal{H})$ to the conditional probability above:

$$P(A,B|H) = P(A|B,\mathcal{H})P(B|\mathcal{H}). \tag{24}$$

Thus the probability that Joe has the disease (B) *and* the test is positive (A) is the product of the probability that Joe has the disease, and the probability that the test detects the disease, given that he has got it.

The *sum rule* relates the *marginal* probability distribution of A, $P(A|\mathcal{H})$, to the joint and conditional distributions:

$$P(A|\mathcal{H}) = \sum_B P(A,B|\mathcal{H}) = \sum_B P(A|B,\mathcal{H})P(B|\mathcal{H}).$$

Here we sum over (or 'marginalize over') the alternative values of B. For example, if Joe either has the disease (B) or does not ($\bar{B}$), then the sum rule states $P(A|\mathcal{H}) = P(A,B|\mathcal{H})+P(A,\bar{B}|\mathcal{H})$, *i.e.*, the probability of obtaining a positive result is the sum of the probabilities of the two alternative explanations: the probability that the result is positive and Joe has the disease, plus the probability that the result is positive and Joe does not in fact have the disease. If B is a continuous variable then the sum is replaced by an integral, and the probability $P(B|\mathcal{H})$ is a probability density.

Having specified the joint probability of all variables as in equation (24), we can then use the rules of probability to evaluate how our beliefs and predictions should change when we gain new information, *i.e.*, as we change the conditioning statements to the right of the "|" symbol. For example, given that Joe's test result is positive, we might wish to know how plausible it is that Joe has the disease; this is measured by the probability $P(B|A,\mathcal{H})$, which can be obtained by Bayes' theorem,

$$P(B|A,\mathcal{H}) = \frac{P(A|B,\mathcal{H})P(B|\mathcal{H})}{P(A|\mathcal{H})}.$$

Here, our overall model of the situation, $\mathcal{H}$, is a conditioning statement on the right hand side of all the probabilities. In this sense, Bayesian inferences are 'subjective', in that it is not possible to reason about data without making assumptions. At the same time, Bayesian inferences are objective, in that anyone who shares the same assumptions $\mathcal{H}$ will draw identical inferences; there is only one answer to a well-posed problem. Bayesian methods thus force us to make all tacit assumptions explicit, and then provide rules for reasoning consistently given those assumptions. Note that the conditioning notation does not imply causation. $P(B|A)$ does not mean 'the probability that Joe's illness is caused by his positive test result'. Rather, it measures the plausibility of proposition B, assuming that the information in proposition A is true.

Posterior, likelihood and prior

When comparing alternative hypotheses $\mathbf{w}$ in the light of data D, Bayes' theorem reads:

$$P(\mathbf{w}|D,\mathcal{H}) = \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})}{P(D|\mathcal{H})}. \tag{25}$$

Each of these terms is given a name:

$P(\mathbf{w}\|D,\mathcal{H})$	*Posterior probability* of $\mathbf{w}$.
$P(D\|\mathbf{w},\mathcal{H})$	*Likelihood* of $\mathbf{w}$.
$P(\mathbf{w}\|\mathcal{H})$	*Prior probability* of $\mathbf{w}$.
$P(D\|\mathcal{H})$	*Evidence* for $\mathcal{H}$.

Note that whereas colloquially the terms likelihood and probability are used interchangeably, here the word likelihood has a distinct meaning.

The evidence is an unimportant normalizing constant in equation (25) but it assumes an important role if we compare alternative models:

$$\begin{aligned} P(\mathcal{H}_1|D) &= \frac{P(D|\mathcal{H}_1)P(\mathcal{H}_1)}{P(D)} \\ P(\mathcal{H}_2|D) &= \frac{P(D|\mathcal{H}_2)P(\mathcal{H}_2)}{P(D)}. \end{aligned}$$

B Summary of neural network modelling suggestions

1. When data modelling is in the research stages, *always* divide the available data into three sets—a training set, a validation set, and a testing set—unless data are really too scarce.

   The training set is used to optimize the parameters of each model. The validation set is used to compare alternative models and to compare alternative modelling strategies. It is a surrogate for the testing set which is set aside and reserved as the final once-only 'real world test'.

   In a non-Bayesian approach, the validation set may be used to set hyperparameters such as weight decay rates. In the Bayesian approach, the training set alone can be used both to infer parameters and hyperparameters.

   The training, validation and test sets should be designed in a way that reflects the real world problem to be solved. This might in some cases mean randomly selecting inputs to put them in each set; or in other scenarios one might wish to systematically arrange that the training set, say, contains a uniform sampling of the input space. Or one might know that in the real world one would have to extrapolate, in which case it would be appropriate to construct a training set having a different input distribution from the other sets.

   Given scarce data it might be worth systematically repeating an experiment, say, ten times, with ten different divisions of the data into training and validation sets.

2. Eyeball the data to check for glitches and outliers.

3. Choose the measure of validation and test performance carefully. It is popular to use the test error (sum squared error) as the default performance measure, but in fact this may be a misleading criterion (MacKay and Oldfield 1995). In many applications there will be an opportunity not to simply make a scalar prediction, but rather to make a prediction with error bars, or maybe an even more complicated predictive distribution. It is then reasonable to compare models in terms of their predictive performance as measured by the log predictive probability of the test data or validation data. Under the log predictive error, as contrasted with the test error, the penalty for making a wild prediction is much less if that wild prediction is accompanied by appropriately large error bars. The log

predictive error (LPE), assuming that for each example m the model gives a prediction $\text{Normal}(y^{(m)}, {\sigma_y^{(m)}}^2)$ is:

$$\begin{aligned}\text{LPE} &= \sum_m - \log P(t^{(m)}|D, \mathcal{H}) \\ &= \sum_m \left[\frac{1}{2} \left(t^{(m)} - y^{(m)} \right)^2 / {\sigma_y^{(m)}}^2 + \log(\sqrt{2\pi}\sigma_y^{(m)}) \right]\end{aligned}$$

Also, going back to scalar predictions: consider using more robust error measures than the mean-square error. After all, is it usually the case that if an error of size 2.0 costs 3 pounds more than an error of size 1.0, then an error of size 3.0 costs 5 pounds more than the size 2.0 error, and an error of size 4.0 costs 7 pounds more than the size 3.0 error? Often a more robust measure such as the mean absolute deviation, or the mean squared error of the smallest 90% of the residuals, may be a more realistic loss function.

4. Optimize the model using a high quality optimizer. Don't use a plain steepest descent optimizer. Use conjugate gradients or other methods that keep track of higher order information about the optimized function (see Press, Flannery, Teukolsky and Vetterling (1988) and Jervis and Fitzgerald (1993), but don't use the code from Press et al. (1988), because it is suboptimal in many ways).

5. Use regularizers, also known as priors, also known as weight decay. If possible, control the regularization constants either using your prior knowledge of the problem, or using Bayesian hyperparameter optimization methods. If you have other ways of controlling the hyperparameters, try those too; and compare them with the Bayesian approach on appropriate validation or test data.

6. Run the optimizations multiple times using different seeds, different initial conditions, different optimization schedules, and examine the validation scores to check that you have got the optimizers running with the knobs set at their best values. Bear in mind that too much regularization early on in the learning may prevent a network from learning interesting structure. A possible rule of thumb is to hold the regularization at a weak value for a considerable fraction of the optimization, and only then allow the hyperparameter control to kick in.

7. During optimization of a model's parameters, keep an eye on the validation performance. If the performance reaches an optimum and then becomes significantly worse, then something is probably wrong with the model. A traditional response to the symptom of 'over-training' is 'early stopping'—stopping the training at the point where the validation performance is best. This seems to me to be a terrible response to a clear model sickness. When a child learns, it is not thought important to whisk them away from the teacher before they 'over-learn'! Over-learning is often an indication that a probabilistic assumption in the noise model or the regularizer is inappropriate.

   The following diagnostic procedures may help track down the problem, and should be followed even if there is not an overtraining problem.

   (a) Look at the residuals, *i.e.*, the differences between the target and the model's output. Are there some huge outliers? You may now be able to detect glitches in the data that were not obvious at the start. (If so, and if they are caused by severely non-Gaussian noise, then maybe they should be left out of the training data.)

   (b) Look at the weights. Are there some parameters that are far bigger than others? Are there some parameters that are all going to zero? Are some hidden units in the network only exploring their linear regime?

   (c) Look at the weights in each regularization class. Do they all have similar magnitudes to each other, or are some weights much bigger than others? If the latter, then this may be a hint that they should not be in the same class as each other.

8. Investigate alternative representations of the problem, *e.g.*, alternative input and output representations. If you think you have a good idea for a preprocessing of the inputs, try it out; but also try feeding a net the raw unprocessed inputs, so it can figure out its own preprocessing (maybe giving such a net an extra hidden layer). Consider constructing a net such that the human solution is a special case of the mappings the net can perform, and initialize the net to that special case. Then when the net is optimized, things can only get better. The worst possible outcome is that you end up with a net that is as good as the preexisting human solution!

9. When making predictions:

   (a) use parameter error bars to compute predictive error bars.

   (b) don't just use one best model. Use multiple good models and combine their predictions. This is called 'forming a committee'. The size of the committee can be controlled for example by looking at the validation error of the committee's predictions, by whatever error function is to be used at the end of the day. Typically there will be an optimum committee of size $L > 1$.

   The following formula (derived by computing the mean and variance of a mixture of Gaussian distributions) gives the error bars for the predictions of a committee consisting of L equally weighted members whose predictions are $y^{(l)} \pm \sigma_y^{(l)}$:

$$\begin{aligned} \text{Committee prediction } \bar{y} &= \frac{1}{L}\sum_k y^{(l)} \\ \text{Variance } \sigma^2 &= \frac{1}{L}\sum_l {\sigma_y^{(l)}}^2 + \frac{1}{L}\sum_l (y^{(l)} - \bar{y})^2. \end{aligned}$$

   Thus the variance of the committee's predictive distribution is the variance of the means plus the mean of the variances.

10. Try to ensure that prior knowledge of the problem is hard wired into the structure of the network.

    If the function is expected to have a certain continuity property as a function of the input variables, then use an input representation that respects this continuity. For example, if an input variable is an angle, do not represent it by a number between -180 and 180; rather, since 180 and -180 are in fact the same as each other, use a representation that makes this explicit. For example, represent the angle as its cosine and sine.

    If an output variable is known to be between 0 and 1, use an output function f that respects these bounds.

C Summary of assumptions

It is important to be aware of the assumptions implicit in the above modelling procedure. If any of these assumptions are expected to give trouble then (with additional effort) new probabilistic models can be designed which implement more appropriate or more general assumptions. The above approach implicitly assumes:

1. That the target is a noisy version of the true output.

2. That the noise is Gaussian with a variance that is the same for all examples.

3. That the underlying mapping from input to output is the same for all time.

4. That the input variables are noise-free.[2]

[2]This assumption could be removed if we made a more sophisticated output noise model.

5. That the true function has the character of typical samples from the distribution illustrated in figure 4. This means in general that:
   (a) The true function is a continuous function of the inputs.
   (b) The typical values of the input variables should be centred on zero. Rescaling of input variables so that they are in the interval [-0.5,0.5], is often a good idea, though not obligatory.
6. That there are no missing inputs. If there are in fact missing inputs then caution is needed. It is not valid to set missing inputs to zero. Probabilistic models that can handle missing inputs (MacKay 1995a) are hard work to implement if they are based on neural networks. An idea worth looking into might be to represent each input by two numbers defining a confidence interval; missing inputs would have the broades interval. An alternative approach is to turn to more tractable graphical probabilistic models (Spiegelhalter and Lauritzen 1990), which can handle missing inputs with no problems, but don't have the same non-linear capabilities as neural networks. The `BUGS` program developed by Spiegelhalter *et al* is an excellent environment for implementing graphical models.

References

Berger, J. (1985). *Statistical Decision theory and Bayesian Analysis*, Springer.

Bhadeshia, H. K. D. H., MacKay, D. J. C. and Svensson, L. E. (1995). Bayesian neural network modelling of weld toughness, *Materials Science and Technology.*

Bishop, C. M. (1992). Exact calculation of the Hessian matrix for the multilayer perceptron, *Neural Computation* **4**(4): 494–501.

Box, G. E. P. and Tiao, G. C. (1973). *Bayesian inference in statistical analysis*, Addison–Wesley.

Breiman, L. (1992). Stacked regressions, *Technical Report 367*, Dept. of Stat., Univ. of Cal. Berkeley.

Cox, R. (1946). Probability, frequency, and reasonable expectation, *Am. J. Physics* **14**: 1–13.

Gull, S. F. (1989). Developments in maximum entropy data analysis, *in* J. Skilling (ed.), *Maximum Entropy and Bayesian Methods, Cambridge 1988*, Kluwer, Dordrecht, pp. 53–71.

Jervis, T. T. and Fitzgerald, W. J. (1993). Optimization schemes for neural networks, *Technical Report CUED/F-INFENG/TR 144*, Cambridge University Engineering Department, Trumpington Street, Cambridge, England.

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

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

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

MacKay, D. J. C. (1995a). Bayesian neural networks and density networks, *Nuclear Instruments and Methods in Physics Research, Section A* **354**(1): 73–80.

MacKay, D. J. C. (1995b). Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks, *Network: Computation in Neural Systems.* to appear.

MacKay, D. J. C. and Oldfield, M. J. (1995). Generalization error and the number of hidden units in a multilayer perceptron, in preparation.

Neal, R. M. (1993). Bayesian learning via stochastic dynamics, *in* C. L. Giles, S. J. Hanson and J. D. Cowan (eds), *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, San Mateo, California, pp. 475–482.

Neal, R. M. (1994). Priors for infinite networks, *Technical Report in preparation*, Univ. of Toronto.

Pearlmutter, B. A. (1994). Fast exact multiplication by the Hessian, *Neural Computation* **6**(1): 147–160.

Press, W., Flannery, B., Teukolsky, S. A. and Vetterling, W. T. (1988). *Numerical Recipes in C*, Cambridge.

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

Skilling, J. (1993). Bayesian numerical analysis, *in* W. T. Grandy, Jr. and P. Milonni (eds), *Physics and Probability*, C.U.P., Cambridge.

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