Bayesian Regularisation and Pruning using a Laplace Prior

Peter M Williams

Cognitive Science Research Paper CSRP-312
School of Cognitive and Computing Sciences
University of Sussex
Falmer, Brighton BN1 9QH
email: peterw@cogs.susx.ac.uk

February 1, 1994

Abstract

Standard techniques for improved generalisation from neural networks include weight decay and pruning. Weight decay has a Bayesian interpretation with the decay function corresponding to a prior over weights. The method of transformation groups and maximum entropy indicates a Laplace rather than a Gaussian prior. After training, the weights then arrange themselves into two classes: (1) those with a common sensitivity to the data error (2) those failing to achieve this sensitivity and which therefore vanish. Since the critical value is determined adaptively during training, pruning—in the sense of setting weights to exact zeros—becomes a consequence of regularisation alone. The count of free parameters is also reduced automatically as weights are pruned. A comparison is made with results of MacKay using the evidence framework and a Gaussian regulariser.

1 Introduction

Neural networks designed for regression or classification need to be trained using some form of stabilisation or regularisation if they are to generalise well beyond the original training set. This means finding a balance between complexity of the network and information content of the data.

Denker et al [3] distinguish *formal* and *structural* stabilisation. Formal stabilisation involves adding an extra term to the cost function that penalises more complex models. In the neural network literature this often takes the form of **weight decay** [18] using the penalty function $\sum_j w_j^2$ where summation is over components of the weight vector. Structural stabilisation is exemplified in polynomial curve fitting by explicitly limiting the degree of the polynomial. Examples relating to neural networks are found in the **pruning** algorithms of le Cun et al [8] and Hassibi & Stork [6]. These use second-order information to determine which weight can be eliminated next at the cost of minimum increase in data misfit. They do not by themselves, however, give a criterion for when to stop pruning.

This paper advocates a type of formal regularisation in which the penalty term is proportional to the logarithm of the L_1 norm of the weight vector $\sum_j |w_j|$. This simultaneously provides **both** forms of stabilisation without the need for additional assumptions.

2 Probabilistic interpretation

Choice of regulariser corresponds to a preference for a particular type of model. From a Bayesian point of view the regulariser corresponds to a prior probability distribution over free parameters $\mathbf{w}$ of the model. Using the notation of MacKay [9, 10] the regularised cost function can be written as

$$M(\mathbf{w}) = \beta E_D(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{1}$$

where E_D measures the data misfit, E_W is the penalty term and $\alpha, \beta > 0$ are regularising parameters determining a balance between the two. (1) corresponds, by taking negative logarithms and ignoring constant terms, to the probabilistic relation

$$P(\mathbf{w}|D) \propto P(D|\mathbf{w})P(\mathbf{w})$$

where $P(\mathbf{w}|D)$ is the posterior density in weight space, $P(D|\mathbf{w})$ is the likelihood of the data D and $P(\mathbf{w})$ is the prior density over weights.[1] According to this correspondence

$$P(D|\mathbf{w}) = Z_D^{-1} \exp -\beta E_D \quad \text{and} \quad P(\mathbf{w}) = Z_W^{-1} \exp -\alpha E_W \tag{2}$$

where $Z_D = Z_D(\beta)$ and $Z_W = Z_W(\alpha)$ are normalising constants. It follows that the process of minimising

$$M(\mathbf{w}) = -\log P(\mathbf{w}|D) + \text{constant}$$

is equivalent to finding a maximum of the posterior density.

2.1 The likelihood function for regression networks

Suppose a training set of pairs $(\mathbf{x}_p, t_p)$, $p = 1, \ldots, N$, is to be fitted by a neural network model with adjustable weights $\mathbf{w}$. The $\mathbf{x}_p$ are input vectors and the t_p are target outputs. The network is assumed for simplicity to have a single output unit. Let $y_p = f(\mathbf{x}_p, \mathbf{w})$, $p = 1, \ldots, N$, be the corresponding network outputs when f is the network mapping, and assume that the measured values t_p differ from the predicted values y_p by an additive noise process

$$t_p = y_p + \nu_p \qquad p = 1, \ldots, N.$$

If the ν_p have independent normal distributions, each with zero mean and the same known standard deviation σ, the likelihood of the data is

$$\prod_{p=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{1}{2}\left(\frac{y_p - t_p}{\sigma}\right)^2$$

which implies that

$$E_D = \tfrac{1}{2}\sum_{p=1}^{N}(y_p - t_p)^2 \tag{3}$$

[1] The notation is somewhat schematic. See [2, 9, 15] for more explicit notations.

according to (2) with $\beta = 1/\sigma^2$ and $Z_D = (2\pi/\beta)^{N/2}$. As $\alpha \to 0$ we have the improper uniform prior over $\mathbf{w}$ so that $P(\mathbf{w}|D) \propto P(D|\mathbf{w})$ and M is proportional to E_D. This means that least squares fitting, which minimises E_D alone, is equivalent to simple maximum likelihood estimation of parameters assuming Gaussian noise [19, §14.1]. Other models of the noise process are possible but the Gaussian model is assumed throughout.[2]

2.2 Weight prior

A common choice of weight prior assumes that weights have identical independent normal distributions with zero mean. If $\{w_j | j = 1, \ldots, W\}$ are components of the weight vector, then according to (2)

$$E_W = \tfrac{1}{2} \sum_{j=1}^{W} w_j^2 \qquad \text{(Gauss)} \tag{4}$$

where $1/\alpha$ is the variance. Alternatively if the absolute values of the weights have exponential distributions then

$$E_W = \sum_{j=1}^{W} |w_j| \qquad \text{(Laplace)} \tag{5}$$

where $1/\alpha$ is the mean absolute value. Another possibility is the Cauchy distribution

$$E_W = (1/\alpha) \sum_{j=1}^{W} \log\left(1 + \alpha^2 w_j^2\right) \qquad \text{(Cauchy)} \tag{6}$$

where $1/\alpha$ is the median absolute value.[3]

Jaynes [7] offers two principles—transformation groups and maximum entropy—for setting up probability distributions in the absence of frequency data. These can be applied as follows. For any feed-forward network in which there are no direct connections between input and output units, there is a functionally equivalent network in which the weight on a given connection has the same size but opposite sign. This is also true if there are direct connections, except for the direct connections. This is evident if the transfer function σ is odd, such as the hyperbolic tangent. It is true, more generally, provided there are constants b, c such that $\sigma(x - b) + \sigma(b - x) \equiv c$. For example $b = 0$, $c = 1$ for the logistic function. Consistency then demands that the prior for a given weight w_j should be a function of $|w_j|$ alone and the maximum entropy distribution for a non-negative quantity constrained to have a given mean is the exponential distribution [22, Ch.5]. It follows that the signed weight w_j has the two-sided exponential or Laplace density $\frac{1}{2}\alpha e^{-\alpha|w_j|}$ where $1/\alpha$ is the mean absolute value. Under the assumption of independence for the joint distribution this leads to the Laplace expression (5) for the regularising term with $Z_W = (2/\alpha)^W$ as normalising constant.

The Gaussian prior would be obtained if constraints were placed on the first two moments of the distribution of the signed weights. The crux of the present argument is

[2] This paper concerns regression networks in which the target values are real numbers, but the same ideas can be applied to classification networks where the targets are exclusive class labels.

[3] A penalty function $w^2/(1 + w^2)$ similar to $\log(1 + w^2)$ is the basis of [23].

that constraining the mean of the signed weights to be zero is not an adequate expression of the intrinsic symmetry in the signs of the weights. A zero mean distribution need not be symmetric and a symmetric distribution need not have a mean. Note that the present argument uses a specific property of neural network models that does not apply to linear regression models generally.[4]

3 Comparison of sensitivities

It is revealing to compare the conditions for a minimum of the overall cost function in the cases of Gauss and Laplace weight priors. Recalling that $M = \beta E_D + \alpha E_W$ it follows that, at a minimum of M where $\partial M / \partial w_j = 0$,

$$\left| \frac{\partial E_D}{\partial w_j} \right| = \frac{\alpha}{\beta} |w_j| \tag{7}$$

assuming E_W is given by the Gaussian regulariser (4). Sensitivity of the data misfit to a given weight is therefore proportional to its size and will be unequal for different weights. Furthermore if w_j is to vanish at a minimum of M then $\partial E_D / \partial w_j = 0$. This is the same condition as for an unregularised network so that Gaussian weight decay contributes nothing towards network pruning in the strict sense.

Condition (7) should be contrasted with Laplacian weight decay (5) where sufficient conditions for a stationary point are, as we shall see, that

$$\left| \frac{\partial E_D}{\partial w_j} \right| = \frac{\alpha}{\beta} \quad \text{if } |w_j| > 0 \tag{8}$$

$$\left| \frac{\partial E_D}{\partial w_j} \right| < \frac{\alpha}{\beta} \quad \text{if } |w_j| = 0. \tag{9}$$

(8) means that, at a minimum, the non-zero weights must arrange themselves so that the sensitivity of the data misfit to each is the same. (9) means that there is a definite cut-off point for the contribution which each weight must make. Unless the data misfit is sufficiently sensitive to the weight on a given connection, that weight is set to zero and the connection can be pruned. At a minimum the weights therefore divide themselves into two classes (i) those with common sensitivity α/β and (ii) those that fail to achieve this sensitivity and which therefore vanish. The critical ratio α/β can be determined adaptively during training. Pruning is therefore automatic and performed entirely by the regulariser.

4 Elimination of α and β

The regularising parameters α and β are not generally known in advance. MacKay [9, 10] proposes the 'evidence framework' for determining these parameters. This paper uses the method of integrating over hyperparameters [2]. A comparison is made in the Appendix.

[4] A possible alternative would be to assume each $|w_j|$ has a log-normal distribution or a mixture of a log-normal and an exponential distribution, compare [16]. For an approach to formal stabilisation, more in the style of [21], see Bishop [1].

The weight prior in (2) depends on α and can be written

$$P(\mathbf{w}|\alpha) = Z_W(\alpha)^{-1} \exp -\alpha E_W \tag{10}$$

where α is now considered as a nuisance parameter. If a prior $P(\alpha)$ is assumed, α can be integrated out by means of

$$P(\mathbf{w}) = \int P(\mathbf{w}|\alpha) P(\alpha)\, d\alpha. \tag{11}$$

Since α is a scale parameter, it is reasonable to use the improper $1/\alpha$ ignorance prior.[5] Using (5) and (10) with $P(\alpha) = 1/\alpha$ it is straightforward to show that

$$-\log P(\mathbf{w}) = W \log E_W$$

to within an additive constant.

If the noise level $\beta = 1/\sigma^2$ is known, or assumed known, the objective function to be minimised in place of M is now

$$L = \beta E_D + W \log E_W\,. \tag{12}$$

In practice β is generally not known in advance and similar treatment can be given to β as was given to α. This leads to

$$-\log P(D|\mathbf{w}) = \tfrac{1}{2} N \log E_D$$

assuming the Gaussian noise model.[6] The negative log posterior $-\log P(\mathbf{w}|D)$ is now given by

$$L = \tfrac{1}{2} N \log E_D + W \log E_W \tag{13}$$

and this replaces (1) as the loss function to be minimised.

It is worth noting that if α and β are assumed known, differentiation of (1) yields $\nabla M = \beta \nabla E_D + \alpha \nabla E_W$ with $1/\beta$ as the variance of the noise process and $1/\alpha$ as the mean absolute value of the weights. Differentiation of (13) yields $\nabla L = \widetilde{\beta} \nabla E_D + \widetilde{\alpha} \nabla E_W$ where

$$1/\widetilde{\beta} = \frac{1}{N} \sum_{p=1}^{N} (y_p - t_p)^2 \tag{14}$$

is the sample variance of the noise and

$$1/\widetilde{\alpha} = \frac{1}{W} \sum_{j=1}^{W} |w_j| \tag{15}$$

is the sample mean of the size of the weights. This means that minimising L is effectively equivalent to minimising M assuming α and β are continuously adapted to the current sample values $\widetilde{\alpha}$ and $\widetilde{\beta}$.

[5] This means assuming that $\log \alpha$ is uniformly distributed or equivalently that $\log \kappa \alpha^{\nu}$ is uniformly distributed for any $\kappa > 0$ and $|\nu| > 0$. The same results can be obtained as the limit of a Gamma prior [14, 26].

[6] The $\frac{1}{2}$ comes from the fact that E_D is measured in squared units. Assuming Laplacian noise this term becomes $N \log E_D$ with $E_D = \sum_p |y_p - t_p|$.

5 Priors, regularisation classes and initialisation

For simplicity Section 2.2 assumed a single weight prior for all parameters. In fact different priors are suitable for the three types of parameter found in feedforward networks, distinguished by their different transformational properties.

Internal weights. These are weights on connections that either input from a hidden unit or output to a hidden unit. The argument of Section 2.2 indicates a Laplace prior. MacKay [10] points out, however, that there are advantages in dividing such weights into separate classes with each class c having its own adaptively determined scale. This leads by the arguments of Section 4 to the more general cost function

$$L \;=\; \tfrac{1}{2}\, N \log E_D + \sum_c W_c \,\log E_W^c \tag{16}$$

where summation is over regularisation classes, W_c is the number of weights in class c and $E_W^c = \sum_{j\in c} |w_j|$ is the sum of absolute values of weights in that class. A simple classification uses two classes consisting of (1) weights on connections with output units as destinations and (2) weights on connections with hidden units as destinations. More refined classifications might be preferred for specific applications.

Biases. Regularisation classes must be exclusive but need not be exhaustive. Parameters belonging to no regularisation class are unregularised. This corresponds to a uniform prior. This is appropriate for biases which transform as location parameters [26]. The prior suitable for a location parameter is one with constant density. Biases are therefore excluded from regularisation.

Direct connections. If direct connections are allowed between input and output units, the argument of Section 2.2 does not apply. There is no intrinsic symmetry in the signs of these weights. It is then reasonable to use a Gaussian prior contributing an extra term $\frac{1}{2}\, W_d \,\log E_W^d$ to the righthand side of (16) where d is the class of direct connections, W_d is the number of direct connections and $E_W^d = \frac{1}{2}\sum_{j\in d} w_j^2$ is half the sum of their squares.

Initialisation

It is natural to initialise the weights in the network in accordance with the assumed prior. For internal weights with the Laplace prior, this is done by setting each weight to $\pm\, a \log r$ where r is uniformly random in $(0, 1)$, the sign is chosen independently at random and $a > 0$ determines the scale. a is then the average initial size of the weights. Satisfactory results are obtained with $a = 1/\sqrt{2m}$ for input weights and $a = 1.6/\sqrt{2m}$ for remaining weights where m is the fan-in of the destination unit. The network function corresponding to the initial guess then has roughly unit variance outputs for unit variance inputs, assuming the natural hyperbolic tangent as transfer function. All biases are initially set to zero.

6 Multiple outputs and noise levels

Suppose the regression network has n output units. In general the noise levels will be different for each output. The data misfit term then becomes $\sum_i \beta_i E_D^i$ where summation is over output units and, assuming independent Gaussian noise, $E_D^i = \frac{1}{2}\sum_p (y_{pi} - t_{pi})^2$ is the error on the ith output, summed over training patterns.[7] If each $\beta_i = 1/\sigma_i^2$ is known, the objective function becomes

$$L = \sum_i \beta_i E_D^i + W \log E_W \tag{17}$$

in place of (12), assuming a single regularisation class. Otherwise integrating over each β_i with the $1/\beta_i$ prior gives

$$L = \tfrac{1}{2} N \sum_i \log E_D^i + \sum_c W_c \log E_W^c$$

in place of (16), assuming multiple regularisation classes.

Multiple noise levels. Even in the case of a single output regression network there may be reason to suspect that the noise level differs between different parts of the training set.[8] In that case the training set can be partitioned into two or more subsets and the term $\frac{1}{2} N \log E_D$ in (16) is replaced by $\frac{1}{2}\sum_s N_s \log E_D^s$ where N_s is the number of patterns in subset s, with $\sum N_s = N$, and $E_D^s = \frac{1}{2}\sum_{p\in s}(y_p - t_p)^2$ is the data error over that subset.

7 Non-smooth optimisation and pruning

The practical problem from here on is assumed to be unconstrained minimisation of (16). The objective function L is non-differentiable, however, on account of the discontinuous derivative of $|w_j|$ at each $w_j = 0$. This is a case of *non-smooth* optimisation [4, Ch. 14]. On the other hand, since L only has discontinuities in its first derivative and these are easily located, techniques applicable to smooth problems can still be effective [5, §4.2].

Most optimisation procedures applied to L as objective function are therefore likely to converge despite the discontinuities, though with a significant proportion of weights assuming negligibly small terminal values, at least for real noisy data. These are weights that an exact line search would have set to exact zeros. They are in fact no longer free parameters of the model and should not be included in the counts W_c of weights in the various regularisation classes. For consistency, these numbers should be reduced during

[7] In many applications it will be unwise to assume that the noise is independent across outputs. This is often a reason for not using multiple output regression models in practice, unless one is willing to include cross terms $(y_{pi} - t_{pi})(y_{pj} - t_{pj})$ in the data error and reestimate the inverse of the noise covariance matrix during training.

[8] Typically this arises when training items relate to domains with an intrinsic topology. For example, predictability of some quantity of interest may vary over different regions of space (mineral exploration) or periods of time (financial forecasting).

the course of training, otherwise the trained network will be over-regularised. The rest of the paper is devoted to this issue.[9]

The approach is as follows. It is assumed that the training process consists of iterating through a sequence of weight vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots$ to a minimum of L. If these are considered to be joined by straight lines, the current weight vector traces out a path in weight space. Occasionally this path crosses one of the hyperplanes $w_j = 0$ where w_j is one of the components of the weight vector. This means that w_j is changing sign. The question is whether w_j is on its way from being sizeably positive to being sizeably negative, or vice versa, or whether $|w_j|$ is executing a Brownian motion about $w_j = 0$. The proposal is to pause when the path crosses, or is about to cross, a hyperplane and decide which case applies. This is done by examining $\partial L/\partial w_j$. If $\partial L/\partial w_j$ has the same sign on both sides of $w_j = 0$, w_j is on its way elsewhere. If it has different signs—more specifically the same sign as w_j on either side—this is where w_j wishes to remain since L increases in either direction. In the second case the proposal is to freeze w_j permanently at zero and exclude it from the count of free parameters. From then on the search continues in a lower dimensional subspace.

With this in mind there are three problems to solve. The first concerns the behaviour of L at $w_j = 0$ and a convenient definition of $\partial L/\partial w_j$ in such a case. The second concerns the method of setting weights to exact zeros and the third concerns the implementation of pruning and the recount of free parameters.

7.1 Defining the derivative

For convenience we write the objective function (16) as $L = L_D + L_W$ where

$$\begin{aligned} L_D &= \tfrac{1}{2} N \log E_D \\ L_W &= \textstyle\sum_c W_c \log \sum_{j \in c} |w_j|. \end{aligned}$$

The problem in defining $\partial L/\partial w_j$ lies with the second term since $|w_j|$ is not differentiable at $w_j = 0$.

Suppose that w_j belongs to regularisation class c and consider variation of w_j about $w_j = 0$ keeping all other weights fixed. This gives the cusp-shaped graph for L_W shown in Figure 1 which has a discontinuous derivative at $w_j = 0$. Its one-sided values are $\pm\, \widetilde{\alpha}_c$ depending on the sign of w_j, where

$$1/\widetilde{\alpha}_c \;=\; \frac{1}{W_c} \sum_{j \in c} |w_j|$$

is the mean absolute value of weights in class c. The two corresponding tangents to the curve are shown as dashed lines.

Consider small perturbations in w_j around $w_j = 0$ keeping other weights fixed. So far as the regularising term L_W alone is concerned, w_j will be restored to zero, since a

[9] Typical features of Laplace regularisation can be sampled by applying some preferred optimisation algorithm directly to the objective functions given by (13) or (16). This corresponds to the 'quick and dirty' method of [10, §6.1].

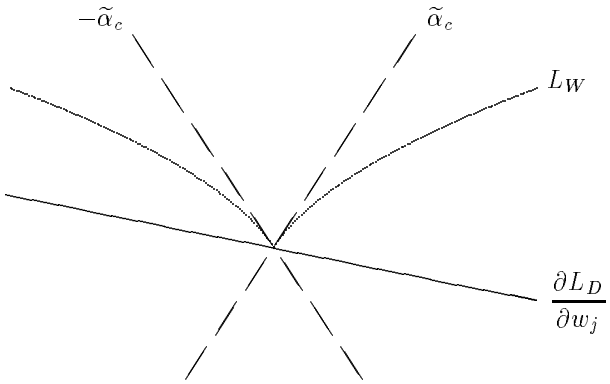

Figure 1: Space-like data gradient at $w_j = 0$.

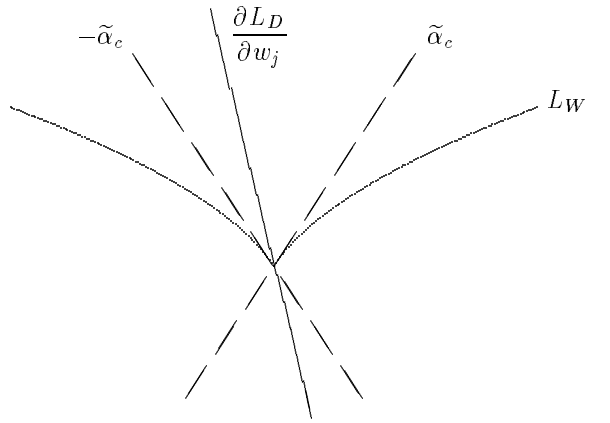

Figure 2: Time-like data gradient at $w_j = 0$.

change in either direction increases L_W. The full objective function, however, is $L = L_D + L_W$ so that behaviour under small perturbations is governed by the sum of the two terms $\partial L_D/\partial w_j$ and $\partial L_W/\partial w_j$. Figure 1 shows one possibility for the relationship between them. Here $\partial L_D/\partial w_j$ is 'space-like' with respect to $\pm\,\widetilde{\alpha}_c$. This is stable since $\partial L/\partial w_j$, which is the sum of the two, has the same sign as w_j in either direction. Small perturbations in w_j will be restored to zero.

Contrast this with Figure 2 where $\partial L_D/\partial w_j$ is now 'time-like' with respect to $\pm\,\widetilde{\alpha}_c$. Increasing w_j will escape the origin since $\partial L_D/\partial w_j$ is more negative than $\partial L_W/\partial w_j = \widetilde{\alpha}_c$ is positive. In short $\partial L/\partial w_j$ is negative for small positive w_j. It follows that the criterion for stability at $w_j = 0$ is that

$$\left|\frac{\partial L_D}{\partial w_j}\right| < \widetilde{\alpha}_c. \tag{18}$$

If L is given by (16) so that $L_D = \frac{1}{2}\,N \log E_D$, then $\partial L_D/\partial w_j = \widetilde{\beta}\,\partial E_D/\partial w_j$ with $\widetilde{\beta}$ given

by (14). The criterion for stability can then be written in terms of E_D as

$$\left|\frac{\partial E_D}{\partial w_j}\right| < \widetilde{\alpha}_c/\widetilde{\beta}$$

and a similar argument establishes (9) in the case of a single regularisation class when α and β are assumed known.

It is convenient to define the objective function partial derivative $\partial L/\partial w_j$ at $w_j = 0$ as follows. If w_j is bound to zero, i.e. the partial derivative $\partial L_D/\partial w_j$ is space-like, $\partial L/\partial w_j$ is defined to be zero. If it is time-like, it is defined to be the value of the downhill derivative. Explicitly using the abbreviations

$$b = \frac{\partial L_D}{\partial w_j} \quad \text{and} \quad a = \widetilde{\alpha}_c$$

then

$$\frac{\partial L}{\partial w_j} = \begin{cases} b + a & \text{if } w_j > 0 \\ b - a & \text{if } w_j < 0 \\ b + a & \text{if } b + a < 0 \\ b - a & \text{if } b - a > 0 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

where the conditions are to be evaluated in order so that the last three apply in the case $w_j = 0$. The last of all applies in the case of (18) when $|b| < a$. Note that if w_j belongs to no regularisation class, e.g. w_j is a bias, then $\partial L/\partial w_j = b$.

If a weight w_j has been set to zero, the value of $\partial L/\partial w_j$ indicates whether or not this is stable for w_j. If so, the partial derivative is zero, showing that no reduction can be made in L by changing w_j in either direction. If not, L can be reduced by increasing or decreasing w_j and $\partial L/\partial w_j$ as defined above now measures the immediate rate at which the reduction would be made.

7.2 Finding zeros

The next task is to ensure that the training algorithm has the possibility of finding exact zeros for weights. A common approach to the unconstrained minimisation problem assumes that at any given stage there is a current weight vector $\mathbf{w}$ and a search direction $\mathbf{s}$. No assumptions need be made about the precise way in which successive directions are determined. Once the search direction is established, we have to solve a one-dimensional minimisation problem. If the scalar function $f(\xi)$ is defined by

$$f(\xi) = L(\mathbf{w} + \xi\mathbf{s}) \tag{20}$$

the problem is to choose $\xi = \xi^* > 0$ to minimise f. Assuming that f is locally quadratic with $f'(0) < 0$ and $f''(0) > 0$, this can be solved by taking $\xi^* = -f'(0)/f''(0)$. Numerator and denominator can be calculated by $f'(0) = \mathbf{s} \cdot \nabla L$ and $f''(0) = \mathbf{s} \cdot \nabla\nabla L\, \mathbf{s}$ where $\nabla\nabla L$ is the Hessian of L.[10] The new weight vector is then $\mathbf{w} + \xi^*\mathbf{s}$. It is not required, however,

[10] The matrix-vector product $\nabla\nabla L\, \mathbf{s}$ can be calculated using [17] or [12]. Alternatively $f''(0)$ can be calculated by differencing first derivatives. Levenberg-Marquardt methods can be used in case $f''(0)$ is sometimes negative or if the quadratic assumption is poor [4, 24, 13].

that ξ^* is determined in this way. All that is required is an iterative procedure that moves at each step some distance along a search direction $\mathbf{s}$ from $\mathbf{w}$ to $\mathbf{w}+\xi\mathbf{s}$, together with some preferred way of determining $\xi = \xi^*$.[11] Unless it was specially designed for that purpose, however, it can be assumed that the preferred algorithm never accidentally alights on an exact zero for any weight.

To allow for this possibility, note that the line $\mathbf{w}+\xi\mathbf{s}$ intersects the hyperplane $w_j = 0$ at $\xi = \xi_j$ where

$$\xi_j = -\frac{w_j}{s_j} \tag{21}$$

provided $|s_j| > 0$, i.e. provided the line is not parallel to the hyperplane. Let ξ_k be the nearest to ξ^* of the $\{\xi_j\}$ defined by (21). In other words $\mathbf{w}+\xi_k\mathbf{s}$ is that point of intersection of the search direction with one of the hyperplanes $\{w_j = 0\}$ which is nearest to $\mathbf{w}+\xi^*\mathbf{s}$. If $\mathbf{w}+\xi_k\mathbf{s}$ is sufficiently close to where the predicted minimum occurs at $\mathbf{w}+\xi^*\mathbf{s}$, or equivalently if ξ_k is sufficiently close to ξ^*, replace ξ^* by ξ_k. In that case the next weight vector in the optimisation process is given by $\mathbf{w}+\xi_k\mathbf{s}$ rather than $\mathbf{w}+\xi^*\mathbf{s}$. More explicitly the criterion for ξ_k being sufficiently close to ξ^* is that

$$\left|1 - \frac{\xi_k}{\xi^*}\right| < \delta$$

for suitable $\delta > 0$. Since ξ_k is the closest to ξ^* of the $\{\xi_j\}$ we need only evaluate

$$\delta_j = \left|1 + \frac{w_j}{s_j\,\xi^*}\right|$$

for each index j for which $|w_j| > 0$ and $|s_j| > 0$, and choose k to be the index that minimises δ_j. Provided $\delta_k < \delta$, replace ξ^* by ξ_k. If not, or if there are no j with $|w_j| > 0$ and $|s_j| > 0$, leave the initial value ξ^* unchanged. Note that if $\delta < 1$, it is not necessary to make a separate check that $\xi_k > 0$ since it is assumed that $\xi^* > 0$.

The choice of δ is not critical. If $f(\xi)$ were quadratic with minimum at $\xi = \xi^*$ then

$$\left|\frac{f(\xi) - f(\xi^*)}{f(0) - f(\xi^*)}\right| < \delta^2$$

whenever $|1 - \xi/\xi^*| < \delta$. Taking $\delta = 0.1$ the reduction in L in moving from $\mathbf{w}$ to $\mathbf{w}+\xi_k\mathbf{s}$ would then be at least 99% of the reduction in moving to where the predicted minimum occurs at $\mathbf{w}+\xi^*\mathbf{s}$. But any value of δ in the range $(0,1)$ gives a reduction in L on the quadratic assumption. For quadratic descent methods $\delta = 1$ has been found satisfactory and is proposed as the default.

Two numerical issues are important. (1) If a nearby zero has been found at $\xi = \xi_k$, the kth component of $\mathbf{w}+\xi\mathbf{s}$ will be $w_k + \xi_k s_k = w_k - (w_k/s_k)\,s_k = 0$. But for later pruning purposes, the new w_k needs to be set to 0 explicitly. It would be unwise to rely on floating point arithmetic.[12] (2) Line search descent methods usually demand a strict

[11] It is not even required that this is uniformly a descent method. Nor does the optimisation algorithm need to make explicit use of a search direction. If it jumps directly from $\mathbf{w}$ to $\mathbf{w}'$, define $\mathbf{s} = \mathbf{w}' - \mathbf{w}$, take $\xi^* = 1$ and proceed as in the text.

[12] Effectively the floating point zero is being used as a natural and convenient boolean flag.

decrease in the value of the objective function on each iteration, or even more [4, §2.5]. But this is inappropriate when we choose a new $w_k = 0$ by using $\xi = \xi_k$. The reason is that $\|\xi_k \mathbf{s}\|$ may be very small, so that the hyperplane is crossed almost immediately on setting out from $\mathbf{w}$ and roundoff errors in computing L become dominant. In that case it is sufficient to require that

$$f(\xi_k) - f(0) \; < \; \epsilon \, |f(\xi_k) + f(0)|$$

where $\epsilon = 10^{-5}$ say for single precision.

In summary it is left to the reader to supply the algorithms for determining successive search directions and the initially preferred value of ξ^*. In this section it has been shown how ξ^* can be modified to find a nearby zero of one of the weights. The previous section dealt with the problem of defining ∇L when that occurs.

8 Pruning

The remaining question is whether a weight that has been set to zero should be allowed to recover or whether it should be permanently bound to zero and pruned from the network.

A weight w_j is said to be *bound* if it satisfies the condition

$$w_j = 0 \;\text{ and }\; \frac{\partial L}{\partial w_j} = 0. \tag{22}$$

It is proposed to prune a weight from the network as soon as it becomes bound. Thereafter it will be frozen at zero and no longer included in the count of free parameters. The connection to which it corresponds has effectively been removed from the network. In this way the current value for the number W_c of free parameters in a given regularisation class c can only decrease.[13]

In more detail the process is as follows. At every stage of the training process each component of the weight vector is classified as being either *frozen* or *not frozen*. Initially no weights are frozen. Only zero weights are ever frozen in the course of training and, once frozen, are never unfrozen. Frozen weights correspond to redundant directions in weight space which are never thereafter explored.[14]

After each change, the weight vector is examined for new zeros. According to the proposal of Section 7.2 at most one new component is set to zero on each iteration. This component, w_j say, is examined to see if it meets the second part of the condition (22). The value W_c to be used in (19) when computing $\partial L/\partial w_j$ via $\tilde{\alpha}_c$ is the number of currently free weights in the class c to which w_j belongs. If (22) is satisfied, w_j is frozen at zero and W_c is reduced. If not, w_j remains at zero but unfrozen, and W_c is unchanged. The process then continues.

[13] In the case of stochastic gradient descent it should be remembered that the definition of being bound for a zero weight depends on the value of the full partial derivative $\partial L_D/\partial w_j$ summed over training patterns.

[14] Alternatively an implementation may choose to actually remove the corresponding connections from the network data structure.

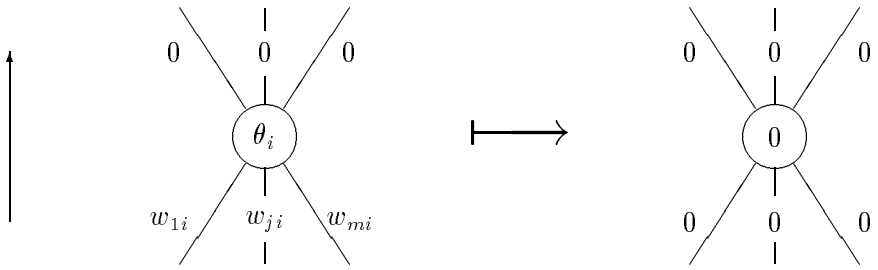

Figure 3: Replacing the input weights of output-dead hidden units by zeros by means of a backward pass.

8.1 Tidying up dead units

An important addition needs to be made to the process just described. It concerns dead units. These are hidden units all of whose inputs weights are frozen or all of whose output weights are frozen. In either case the unit is redundant and neither its input weights nor its output weights should count as free parameters of the model.

Output-dead units. Suppose that all *output* weights of a given hidden unit have been frozen at zero. These connections are effectively disconnected. The input to any other unit from this unit is constantly zero. Its input weights are therefore also redundant parameters and should not be counted as free parameters of the model. This situation is shown in the lefthand diagram of Figure 3 in which the direction of forward propagation is from bottom to top.[15]

A functionally equivalent network is obtained by replacing all the input weights and bias by zero, as shown in the righthand part of Figure 3. Since the data gradient of each of these weights is zero, condition (22) is satisfied, so that these weights should also be frozen and no longer included in the count of free parameters. The process indicated in Figure 3 should be performed using a backward pass through the network, since the newly frozen input weights of the unit indicated may be output weights for some other hidden unit.

Input-dead units. Figure 4 shows the dual situation in which all the *input* weights of a hidden unit have been frozen at zero. In this case the unit is not altogether redundant since it generally computes a non-zero constant function $y_i = \sigma_i(\theta_i)$ where σ_i is the transfer function for the ith unit and θ_i is its bias. Suppose that unit i outputs to unit j amongst others. Then the input contribution to unit j from unit i is the constant $w_{ij}\sigma_i(\theta_i)$. There is a degeneracy as things stand since the effective bias on unit j depends only on the sum $\theta_j + w_{ij}\sigma_i(\theta_i)$. The network will compute the same function if w_{ij} is set

[15] It is assumed, as the definition of a feedforward network, that the underlying directed graph is acyclic, so that the units can be linearly ordered as $u_1, \ldots, u_n$ with $i < j$ whenever u_i outputs to u_j. The terms forward and backward are to be understood in the sense of some such ordering.

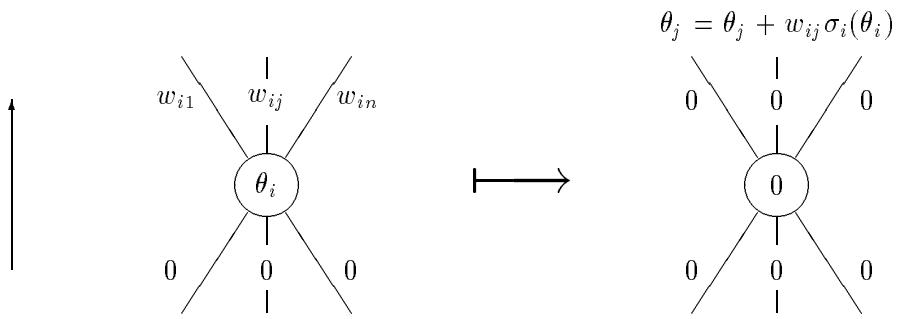

Figure 4: Replacing the output weights of input-dead hidden units by zeros by means of a forward pass, with compensating adjustments in the biases of destination units.

to zero and the bias θ_j on unit j is increased by $w_{ij}\sigma_i(\theta_i)$. The result of doing so is shown in the righthand part of Figure 4. This process should be performed using a forward pass through the network since the newly frozen output weights of the unit indicated may be input weights for some other hidden unit.

Let us call a network *tidy* if each hidden unit satisfies the condition that its bias and all input and output weights are zero whenever either all its input weights are zero or all its output weights are zero. It can be shown that every feedforward network is functionally equivalent to a tidy network and that a functionally equivalent tidy network can be obtained by a single forward and backward pass of the transformations indicated in Figures 3 and 4 performed in either order.

In fact we are concerned here only to tidy networks in which all input weights or all output weights are not merely zero but are *frozen* at zero. But the process is the same. Furthermore it is clear that if all the input and output weights of a hidden unit are zero, necessarily $\partial L_D/\partial w_j = 0$ for each of these weights and consequently $\partial L/\partial w_j = 0$ in virtue of (19). It follows that condition (22) is satisfied so that these weights are automatically frozen and no longer included in the count of free parameters.

8.2 The algorithm

The algorithm can be stated as follows. Consider all weights and biases in the network to form an array `w`. Suppose there is also a parallel Boolean array `frozen`, of the same length as `w`, initialised to `FALSE` for each component. Let `g` stand for the array corresponding to ∇L. Suppose in addition that there is a variable `W[c]` for each regularisation class counting the number of currently non-frozen weights in that class.

It is assumed that a sequence of weight vectors `w` arises from successive iterations of the optimisation algorithm and that the weight vector occasionally includes a new zero component `w[j]` using the procedure of Section 7.2. After each iteration on which the new weight vector contains a new zero, `w` and `frozen` must be processed as follows.

1. freeze zeros in accordance with (22)

```
frozen[j] := (frozen[j] OR (w[j] = 0 AND g[j] = 0))
```

for each component of the weight vector;

2. extend freezing, maybe, using the tidying algorithm of Section 8.1 and set

```
frozen[j] := TRUE
```

for each newly zeroed weight;

3. recount the number `W[c]` of non-frozen weights in each class.

Because of the `OR` in step 1, freezing is irreversible and after a weight is frozen at zero its value should never change. If `s` is the array corresponding to the search vector, this is best enforced whenever the search direction is changed by requiring that

```
IF frozen[j] THEN s[j] := 0
```

for each component of `s`. It is also wise to append to the definition of the gradient array `g` the stipulation

```
IF frozen[j] THEN g[j] := 0
```

for each component of `g`.

Each time a weight is frozen the objective function L defined by (16) changes because of a change in the relevant W_c. But since E_D is unchanged, this is a simple. It will also be necessary to recalculate the gradient vector ∇L. But this is equally simple since ∇L_D only changes if it was necessary to do some tidying in step 2 and this will only be for newly frozen weights which automatically have zero gradients, without the need for calculation.

Whenever one or more weights are frozen, the optimisation process restarts in a lower dimensional space with the projection of the current weight vector serving as the new initial guess. This means that the compound process enjoys whatever convergence and stability properties are enjoyed by the simple process in the absence of freezing. Assuming the simple process always converges, each period in which the objective function is unchanged either terminates with convergence or with a strict reduction in $\sum_c W_c$. Since each W_c is finite the compound process must terminate.

9 Examples

Examples of Laplace regularisation applied to problems in geophysics can be found in [25] and [26]. This section compares results obtained using the Laplace regulariser with those of MacKay [10] using the Gaussian regulariser and the evidence framework. The problem concerns a simple two joint robot arm. The mapping $(x_1, x_2) \mapsto (y_1, y_2)$ to be interpolated is defined by

$$\begin{aligned} y_1 &= r_1 \cos(x_1) + r_2 \cos(x_1 + x_2) \\ y_2 &= r_1 \sin(x_1) + r_2 \sin(x_1 + x_2) \end{aligned}$$

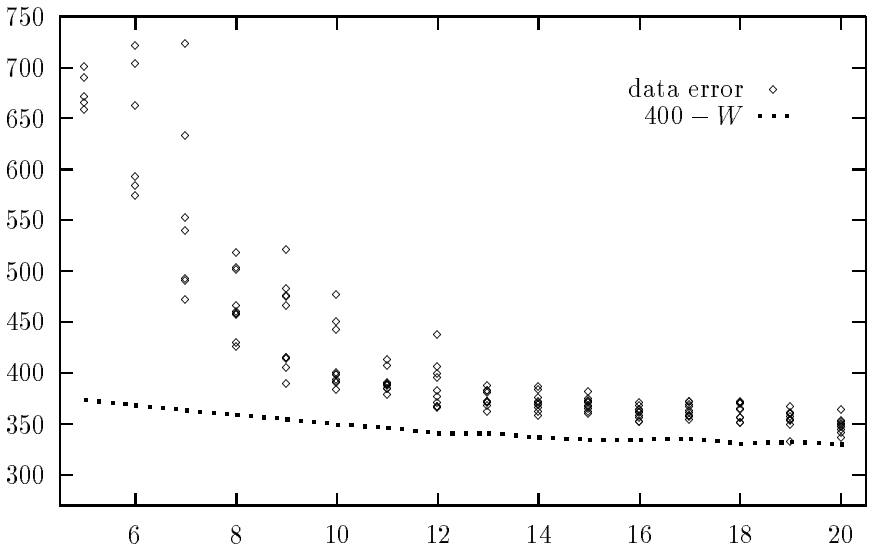

Figure 5: Plot showing the data error of 148 trained networks. 10 networks were trained for each of 16 network architectures with hidden units ranging from 5 to 20. 12 outliers relating to small numbers of hidden units have been excluded. The dotted line is $400 - W$ where W is the empirically determined number of free parameters remaining after Laplace regularised training, averaged over each group of 10 trials.

where $r_1 = 2.0$ and $r_2 = 1.3$. As training set, 200 random samples were drawn from a restricted range of (x_1, x_2) and Gaussian noise of standard deviation 0.05 was added to the calculated values of (y_1, y_2) as target values.[16] Simple three layer networks were used with 2 input, 2 output and from 5 to 20 hidden units. Results are shown in Figure 5.

For comparability with MacKay's results a single regularisation class was used and it was assumed that the noise level $\sigma = 0.05$ was known in advance. The objective function to be minimised is therefore (17) with $\beta_1 = \beta_2 = 1/\sigma^2$. The ordinate in Figure 5 is twice the final value of the first term on the righthand side of (17). This is a dimensionless χ^2 quantity whose expectation is 400 ± 20 relative to the actual noise process used in constructing the training set. Results on a test set also of size 200 and drawn from the same distribution as the training set are shown in Figure 6 using the same error units. Comparison with results on a further test set, of the same size and drawn from the same distribution, is shown in Figure 7. This confirms MacKay's observation that generalisation error on a test set is a noisy quantity, so that many data would have to be devoted to a test set for test error to be a reliable way of setting regularisation parameters.

Performance on both training and test sets settles down after around 13 hidden units. Little change is observed when further hidden units are added since the extra connections are pruned by the regulariser as shown by the dotted line in Figure 5. This contrasts

[16] Training and test sets used here are the same as those in [10] by courtesy of David MacKay.

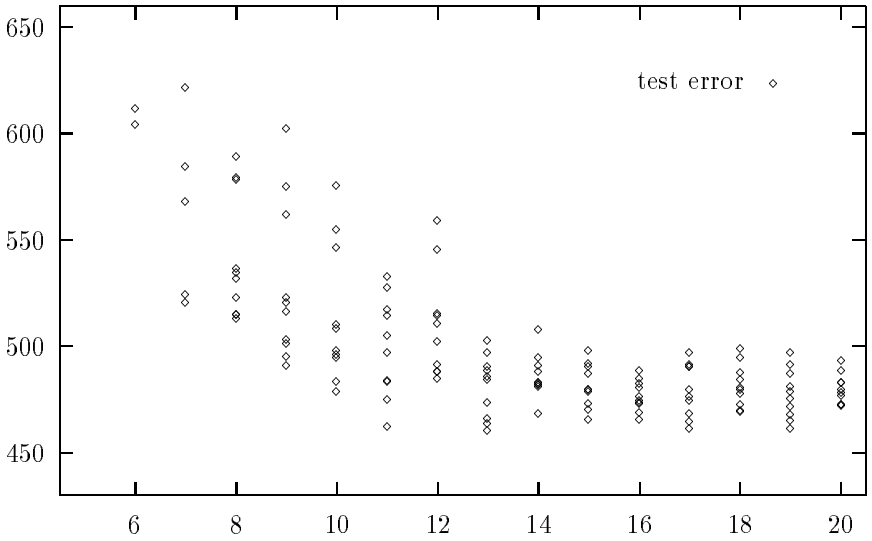

Figure 6: Test error versus number of hidden units.

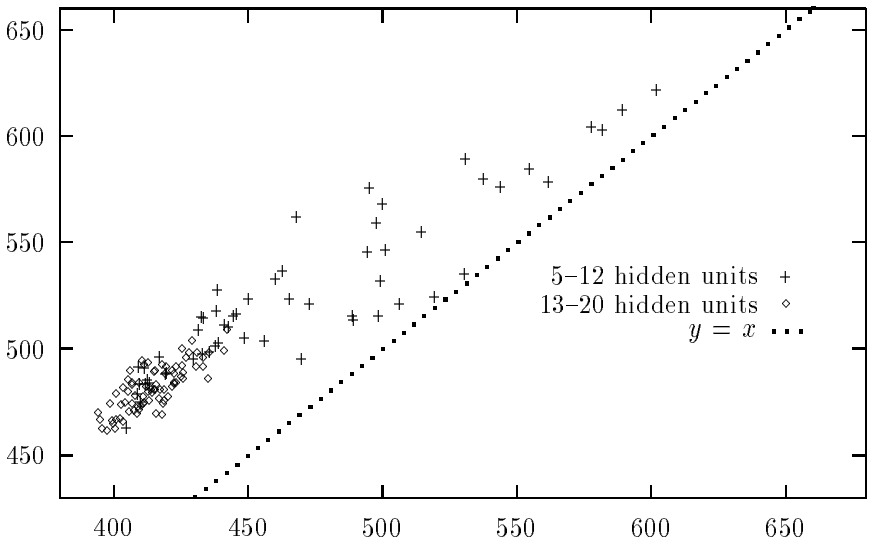

Figure 7: Errors on two test sets.

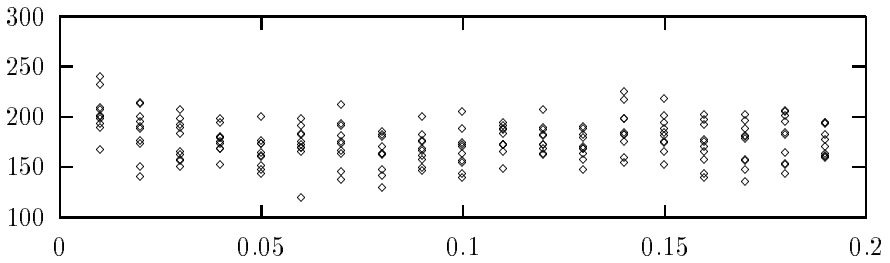

Figure 8: Data error versus noise level for an initial 50 hidden units.

with MacKay's results using the sum of squares regulariser for which the training error continues to decrease as more hidden units are added and where the training error for approaching 20 hidden units differs very little from the best possible unregularised fit. MacKay's approach is to evaluate the 'evidence' for each solution and to choose a number of hidden units that maximises this quantity, which in this case is approximately 11 or 12. The present heuristic is to supply the network with ample hidden units and to allow the regulariser to prune these to a suitable number. Provided the initial number of hidden units is sufficient, the results are largely independent of the number of units initially supplied.

9.1 Varying the noise

For a further demonstration of Laplace pruning, the problem is changed to one in which the network has a single output. Multiple output regression networks are unusual in practice, especially ones satisfying a relation such as $y_1(x_1, x_2) \equiv y_2(x_1 + \pi/2,\, x_2)$. There is also the possibility that the hidden units divide themselves into two groups, each serving one of the two outputs exclusively, which can make it difficult to interpret results. We therefore consider interpolation of just one of the outputs considered above, specifically the cosine expression y_1. The same 200 input pairs (x_1, x_2) were used as for MacKay's training set, but varying amounts of Gaussian noise were added to the target outputs.

Results using a network with 50 hidden units and with noise varying from 0.01 to 0.19 in increments of 0.01 are shown in Figure 8. In this case the noise was resampled on each trial so that each of the 190 different networks was trained on a different training set. Two regularisation classes were used and it was no longer assumed that the noise level was known in advance. The objective function is therefore given by equation (16) with input and output weights forming the two classes. The data error in Figure 8 is again shown in χ^2 units whose expected value is now 200 relative to the actual noise process since there is only one output unit. Specifically the ordinate in Figure 8 measures $\sum_p [(y_p - t_p)/\sigma]^2$ where σ is the abscissa and p ranges over the 200 training items. The actual data error increases proportionately with the noise so that the normalised quantity is effectively constant.

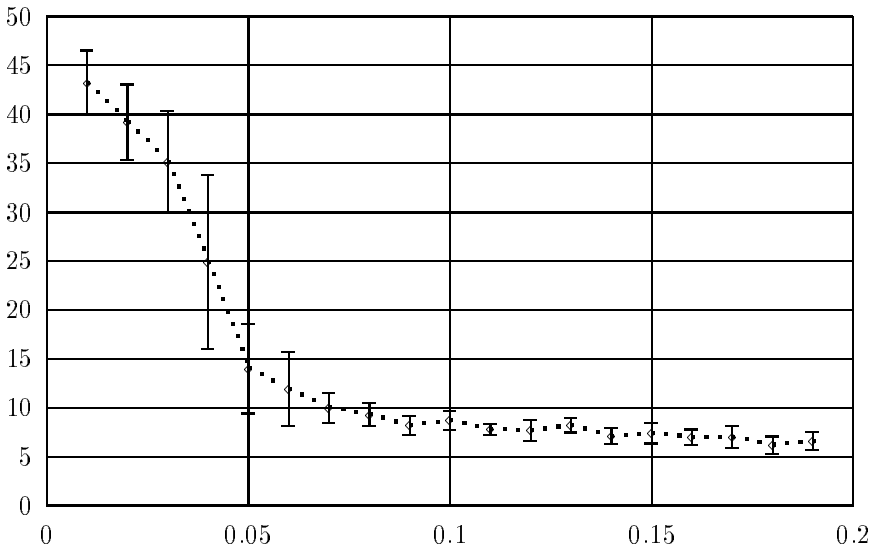

Figure 9: Live hidden units versus noise level for an initial 50 hidden units.

Figure 9 shows mean numbers of live hidden units, with one standard deviation error bars, in networks corresponding to each of the 19 noise levels. This is the number of hidden units remaining in the trained network after the pruning implicit in Laplace regularisation. Note that the number of initially free parameters in a 50 hidden unit network with 2 inputs and 1 output is 201 so that with 200 data points the initial ratio of data points to free parameters is approximately 1. This should be contrasted with the statement in [10] that the numerical approximation needed by the evidence framework, when used with Gaussian regularisation, seems to break down significantly when this ratio is less than 3 ± 1.

Figure 9 indicates that there ought to be little purpose in using networks with more than 20 hidden units for noise levels higher than 0.05, if it is to be correct to claim that results are effectively independent of the number of hidden units used, provided there are enough of them. To verify this a further 190 networks were trained using an initial architecture of 20 hidden units. Results for the final numbers of hidden units are shown in Figure 10. Comparison with Figure 9 shows that if more than 20 hidden units are available for noise levels below 0.05 the network will use them. But for higher noise levels, there is no significant difference in the number of hidden units finally used, whether 20 or 50 are initially supplied. The algorithm also works for higher noise levels. Figure 11 shows corresponding results for noise levels from 0.05 to 0.95 in increments of 0.05. Note that in all these demonstrations with varying noise, the level is automatically detected by the regulariser and the number of hidden units, or more generally the number of parameters, is accommodated to suit the level of noise detected.

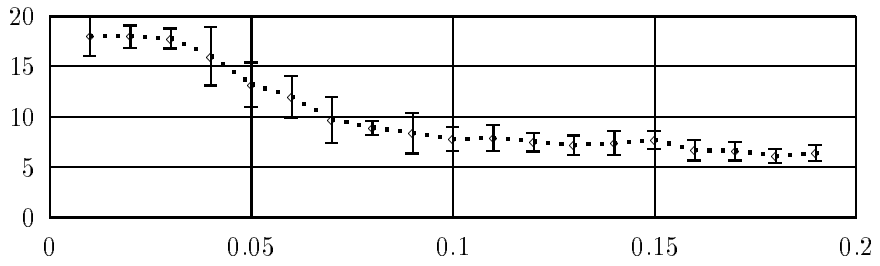

Figure 10: Live hidden units versus noise level for an initial 20 hidden units.

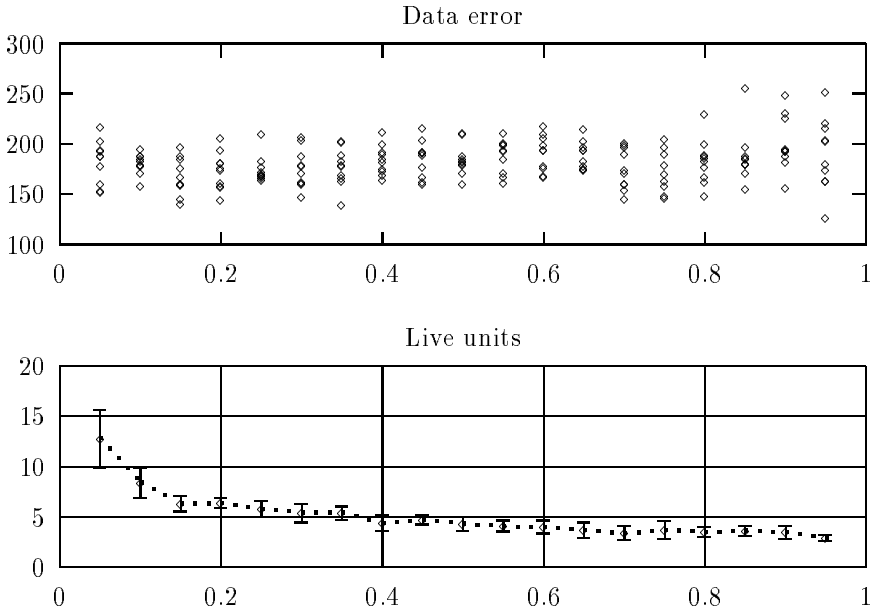

Figure 11: Data error and live hidden units versus larger noise levels for 20 hidden units.

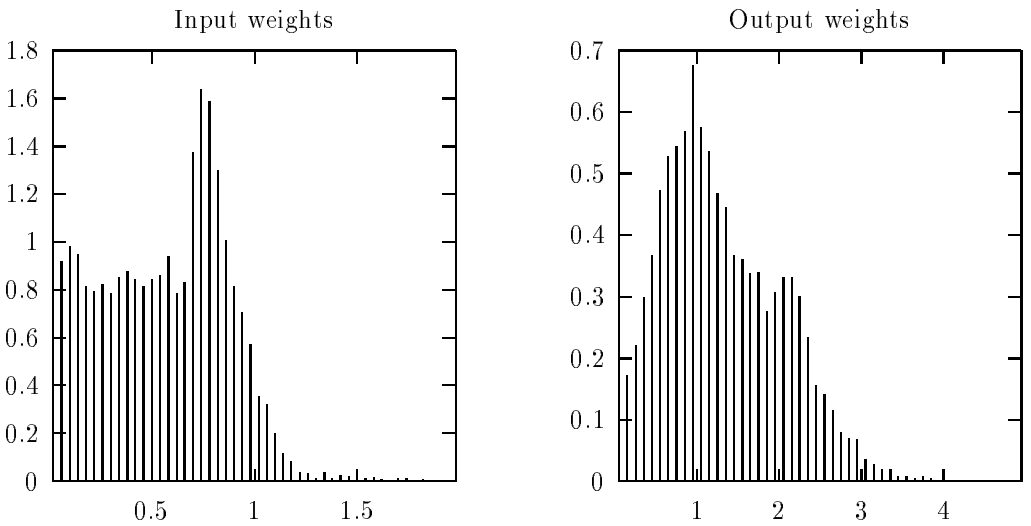

Figure 12: Empirical posterior distributions of the size of non-zero input and output weights for 500 trained networks each using 20 hidden units. Mean values are 0.55 for input weights and 1.31 for output weights. The natural hyperbolic tangent was used as transfer function for hidden units.

9.2 Posterior weight distribution

It was noted in Section 3 that the weights arrange themselves at a minimum so that the sensitivity of the data error to each of the non-zero weights in a given regularisation class is the same, assuming Laplace regularisation is used. For the weights themselves, the posterior conditional distributions in a given class are roughly uniform over an interval. Figure 12 shows the empirical distributions for a sample of 500 trained networks. These plots answer the question "what is the probability that the size of a randomly chosen input (output) weight of a trained network lies between x and $x + \delta x$ conditional on its being non-zero?" The unconditional distributions have discrete components at the origin. The probability of an output weight being zero was 0.38 and the probability of an input weight being zero was 0.47. These networks were trained on the cosine output of the robot arm problem using MacKay's sampling of the noise at the 0.05 level.

10 Summary and conclusions

This paper has argued that the $\sum |w|$ regulariser is more appropriate for the hidden connections of feed-forward networks than the $\sum w^2$ regulariser. It has shown how to deal with discontinuities in the gradient of $|w|$ and how to recount the free parameters of the network as they are pruned by the regulariser. No numerical approximations need be made and the method can be applied exactly to small noisy data sets where the ratio of free parameters to data points may approach unity.

Appendix

The evidence framework [9, 10, 20] proposes to set the regularising parameters α and β by maximising

$$P(D) = \int P(D|\mathbf{w})P(\mathbf{w})\,d\mathbf{w} \tag{23}$$

considered as a function of α and β. This quantity is interpreted as the *evidence* for the overall model including both the underlying architecture and the regularising parameters.

From equations (1) and (2) it follows that

$$P(D) \;=\; (Z_W Z_D)^{-1} \int e^{-M}\, d\mathbf{w}.$$

To evaluate the integral analytically, M is usually approximated by a quadratic in the neighbourhood of a maximum of the posterior density at $\mathbf{w} = \mathbf{w}_{\mathrm{MP}}$ where ∇M vanishes. The approximation is then

$$M(\mathbf{w}) \;=\; M(\mathbf{w}_{\mathrm{MP}}) + \tfrac{1}{2}(\mathbf{w} - \mathbf{w}_{\mathrm{MP}})^{\mathrm{T}}\mathbf{A}\,(\mathbf{w} - \mathbf{w}_{\mathrm{MP}}) \tag{24}$$

where $\mathbf{A} = \nabla\nabla M$ is the Hessian of M evaluated at $\mathbf{w}_{\mathrm{MP}}$. It follows that

$$-\log P(D) \;=\; \alpha E_W + \beta E_D + \tfrac{1}{2}\log\det\mathbf{A} + \log Z_W + \log Z_D + \text{constant}$$

where the constant, which also takes account of the order of the network symmetry group, does not depend explicitly on α or β.

Now the Laplace regulariser E_W is locally a hyperplane. This means that $\nabla\nabla E_W$ vanishes identically so that $\mathbf{A} = \beta\mathbf{H}$ where $\mathbf{H} = \nabla\nabla E_D$ is the Hessian of the data error alone. Assuming the Laplace regulariser and Gaussian noise, $Z_W = (2/\alpha)^W$ and $Z_D = (2\pi/\beta)^{N/2}$ so that

$$-\log P(D) \;=\; \alpha E_W + \beta E_D + \tfrac{k}{2}\log\beta - W\log\alpha - \tfrac{N}{2}\log\beta + \text{constant}$$

where k is the full dimension of the weight vector. Setting partial derivatives with respect to α and β to zero yields $\alpha = W/E_W$ and $\beta = (N-k)/2E_D$ so that

$$1/\beta \;=\; \frac{1}{N-k}\sum_{p=1}^{N}(y_p - t_p)^2 \tag{25}$$

and

$$1/\alpha \;=\; \frac{1}{W}\sum_{j=1}^{W}|w_j|. \tag{26}$$

These should be compared with (14) and (15). If (25) and (26) are used as reestimation formulae during training, the difference between the evidence framework and the method of integrating over hyperparameters, in the case of Laplace regularisation, reduces to the

difference between the factors $N-k$ and N when reestimating β.[17] In many applications the differences in results, when using these two factors with Laplace regularisation, are not sufficiently clear to decide the matter empirically and it needs to be settled on grounds of principle [11, 27]. In the present context, this paper prefers the method of integrating over hyperparameters for reasons of simplicity. Its main purpose, however, is to advocate the Laplace over the Gaussian regulariser, in which case the difference between these two methods of setting regularising parameters appears less significant.

Acknowledgements

I am grateful to Dr Perry Eaton, Dr Colin Barnett and other members of the Geophysical Department of Newmont Exploration Limited for stimulating discussions on the subject of this paper and related topics over the last few years.

References

[1] C. M. Bishop. Curvature-driven smoothing in backpropagation neural networks. In *Proceedings International Neural Network Conference, Vol.2*, pages 749–752, Paris, 1990.

[2] Wray L. Buntine and Andreas S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

[3] John Denker, Daniel Schwartz, Ben Wittner, Sara Solla, Richard Howard, Lawrence Jackel, and John Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

[4] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, second edition, 1987.

[5] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, 1981.

[6] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan Kaufmann, 1993.

[7] E. T. Jaynes. Prior probabilities. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(3):227–241, 1968.

[17] If β is assumed known in advance the methods are apparently equivalent. For multiple regularisation classes the same argument leads, on either approach, to the reestimation formula $\alpha_c = W_c/E_W^c$ for each regularisation class c. For the multiple noise levels envisaged in Section 6, however, results will generally differ unless the levels are known in advance. Note that, in saying that the Laplace regulariser is locally a hyperplane, it is assumed that none of the regularised weights vanishes, otherwise the Hessian $\mathbf{A}$ is not defined and the quadratic assumption (24) is no longer meaningful. It is therefore assumed that zero weights are also pruned for the Laplace regulariser when using the evidence framework [20].

[8] Yann le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, 1990.

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

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

[11] David J. C. MacKay. Hyperparameters: Optimise, or integrate out? In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Barbara 1993*, Dordrecht, 1994. Kluwer.

[12] Martin F. Møller. Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in $O(n)$ time. Report DAIMI PB-432, Computer Science Department, Aarhus University, Denmark, 1993.

[13] Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

[14] Radford M. Neal. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Technical Report CRG-TR-92-1, Department of Computer Science, University of Toronto, April 1992.

[15] Radford M. Neal. Bayesian learning via stochastic dynamics. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 475–482. Morgan Kaufmann, 1993.

[16] Steven J. Nowlan and Geoffrey E. Hinton. Adaptive soft weight tying using gaussian mixtures. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 993–1000. Morgan Kaufmann, 1992.

[17] Barak A Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1), 1994.

[18] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back-propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh PA 15213, 1986.

[19] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[20] Hans Henrik Thodberg. Ace of Bayes: Application of neural networks with pruning. Manuscript 1132E, The Danish Meat Research Institute, May 1993.

[21] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. John Wiley & Sons, 1977.

[22] Myron Tribus. *Rational Descriptions, Decisions and Designs*. Pergamon Press, 1969.

[23] Andreas S. Weigend, David E. Rumelhart, and Bernado A. Huberman. Generalization by weight-elimination with application to forecasting. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. Morgan Kaufmann, 1991.

[24] P. M. Williams. A Marquardt algorithm for choosing the step-size in backpropagation learning with conjugate gradients. Cognitive Science Research Paper CSRP 229, University of Sussex, February 1991.

[25] P. M. Williams. Aeromagnetic compensation using neural networks. *Neural Computing & Applications*, 1:207–214, 1993.

[26] P. M. Williams. Improved generalization and network pruning using adaptive Laplace regularization. In *Proceedings of 3rd IEE International Conference on Artificial Neural Networks*, pages 76–80, Institution of Electrical Engineers, London, 1993.

[27] David H. Wolpert. On the use of evidence in neural networks. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 539–546. Morgan Kaufmann, 1993.