Bayesian Regularisation and Pruning using a Laplace Prior

Peter M Williams

Cognitive Science Research Paper CSRP- 12
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

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

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)

$$E_W = \frac{1}{}$$

that constraining the mean of the signed weights to be zero is not an adequate expression of the intrinsic

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

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

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

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

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, $E_D^i =$ [1]

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.

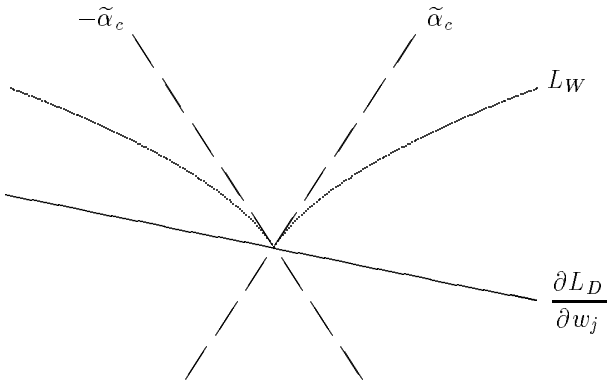


Figure 1: Space-like data gradient at w_j

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

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

$\partial L/\partial$
w $b$when

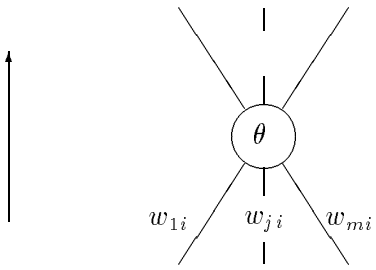
where the two conditions are to be evaluated in order so that the last three apply in the sost12.960213.4400Td(v

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 beR20r


θ
w_{1i}
$w_{j\,i}$
w_{mi}

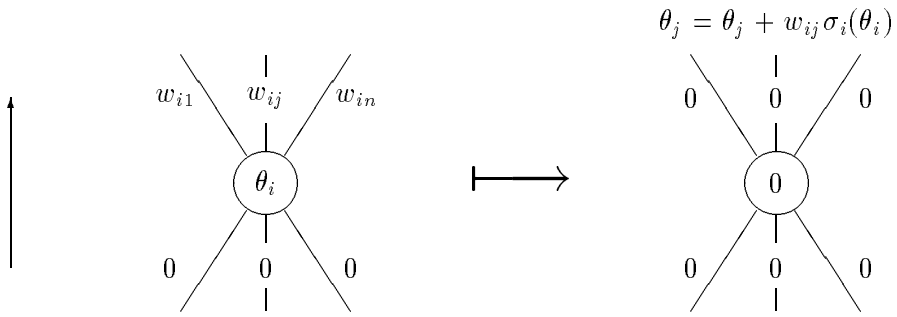


Figure 4: Replacing the output weights of input-dead hidden units by zeros by means of a forward pass, with compensating adjustments in the biases of destination units.

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

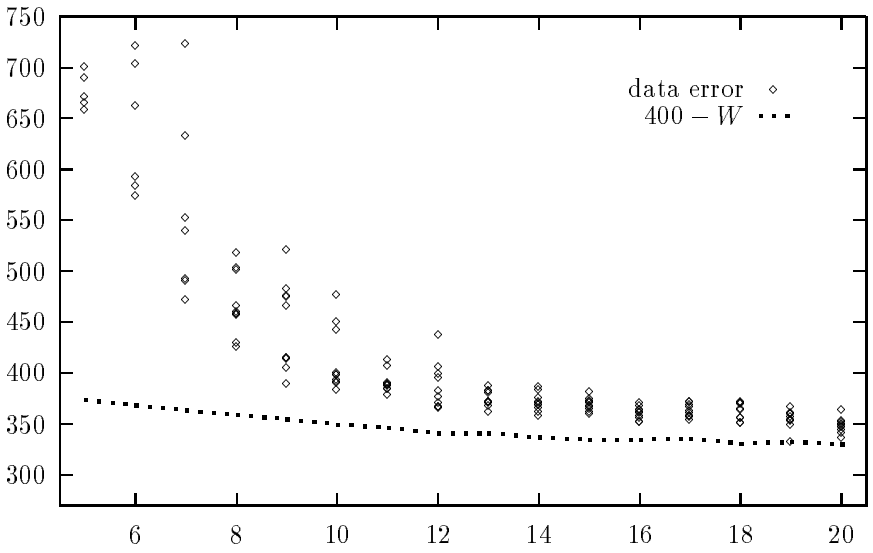

750
700
650
600
550
500
450
400
350
300
6
8
10
12
14
16
18
20
data error
400 − W

Figure 5:

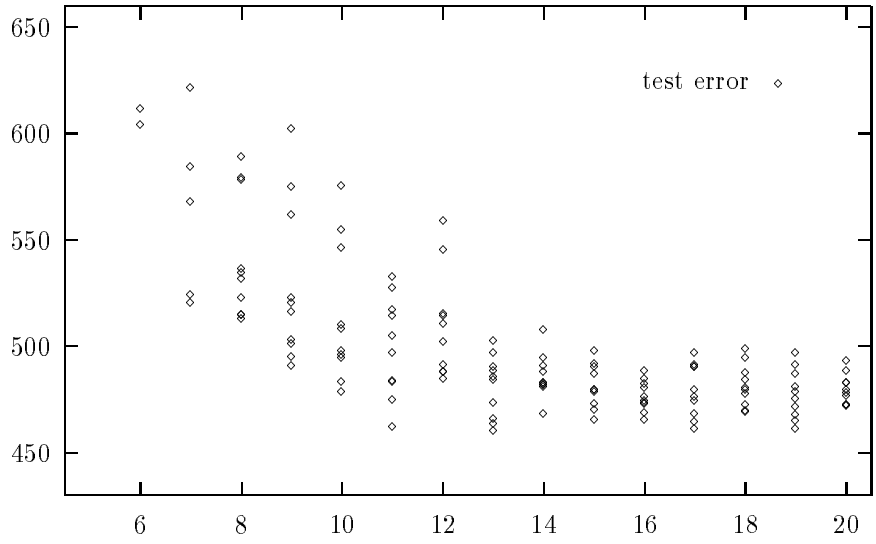


Figure 6: Test error versus number of hidden units.

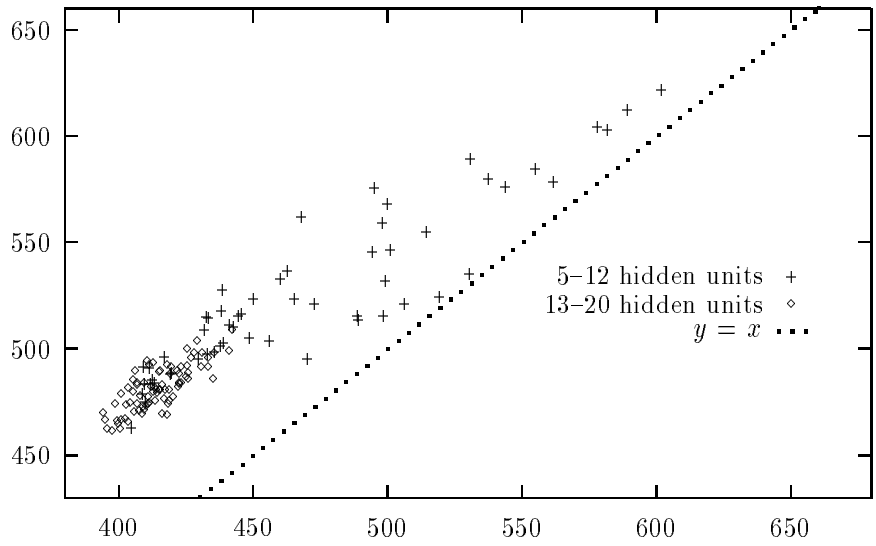
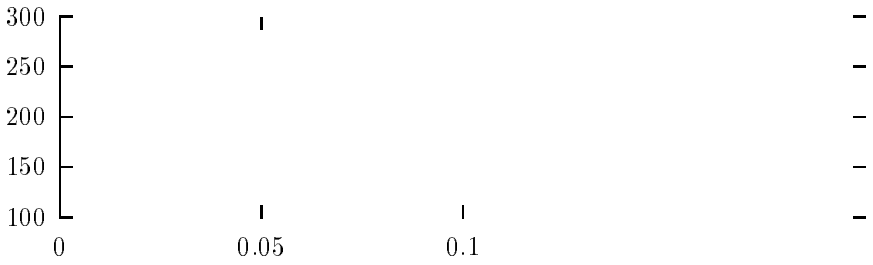


Figure 7: Errors on two test sets.


300
250
200
150
100
0
0.05
0.1

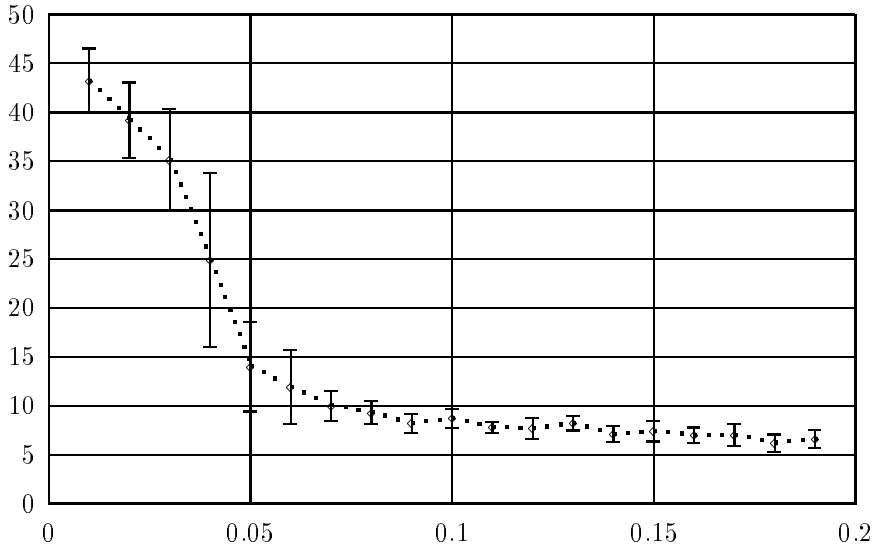


Figure 9: Live hidden units versus noise level for an initial 50 hidden units.

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.

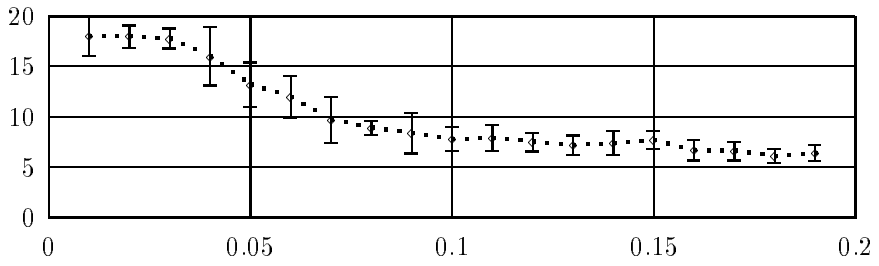
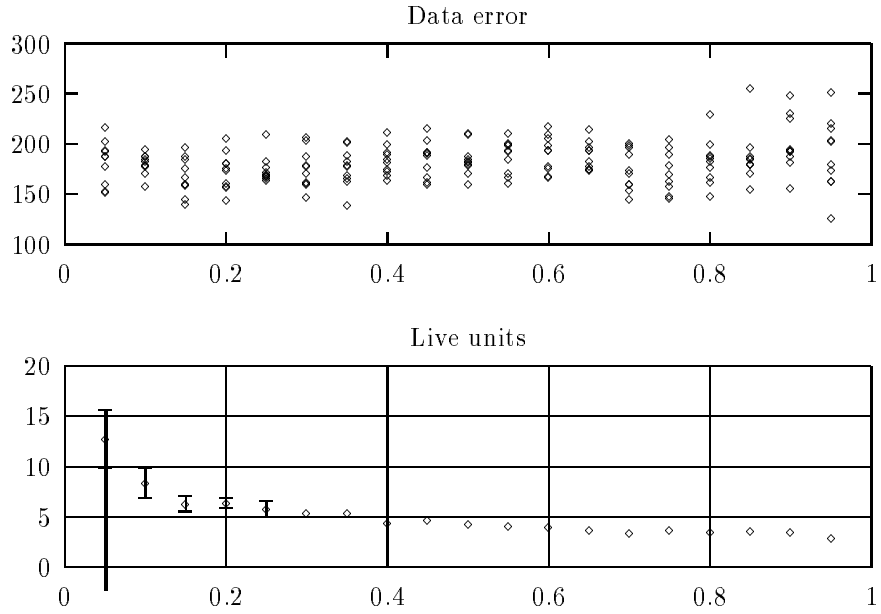


Figure 10: Live hidden units versus noise level for an initial 20 hidden units.



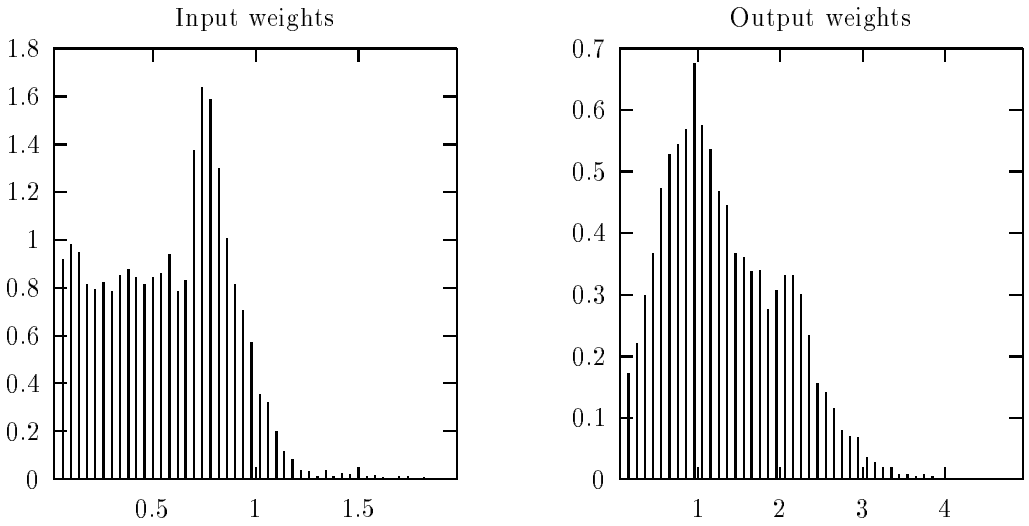


Figure 12: Empirical posterior distributions of the size of non-zero input and output weights for 500 trained networks each using 20 hidden units. Mean values are 0.55 for input weights and 1.31 for output weights. The natural hyperbolic tangent was used as transfer function for hidden units.

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

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

$$M(\mathbf{w}) \;=\; M(\mathbf{w}_{\mathrm{MP}}) + \tfrac{1}{}$$

difference between the factors

[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.