Regularization and Complexity Control in Feed-forward Networks

Christopher M. Bishop

Neural Computing Research Group
Dept. of Computer Science and Applied Mathematics
Aston University, Birmingham, UK.
c.m.bishop@aston.ac.uk

Technical Report: NCRG/95/022
Available from: http://neural-server.aston.ac.uk/

Abstract

In this paper we consider four alternative approaches to complexity control in feed-forward networks based respectively on architecture selection, regularization, early stopping, and training with noise. We show that there are close similarities between these approaches and we argue that, for most practical applications, the technique of regularization should be the method of choice.

1 Introduction

A central issue in the application of feed-forward networks is the determination of the appropriate level of complexity for the network. The complexity determines the generalization properties of the model, since a network which is either too simple or too complex will have poor generalization[1]. This can be easily understood by analogy to the problem of curve fitting using polynomials. Consider a data set generated from a smooth underlying function with additive noise on the target variables. A polynomial with too few coefficients will be unable to capture the underlying function from which the data was generated, while a polynomial with too many coefficients will start to model the noise on the data (the phenomenon of over-fitting) and hence will again result in a poor representation of the underlying smooth function. The order of the polynomial controls the number of degrees of freedom (i.e. the number of independently adjustable parameters) in the model, and there will be some optimum number of coefficients for which the fitted polynomial will give the best representation of the function. It is this polynomial which will, on average, give the best predictions for new data. A similar situation arises in the application of neural networks, where it is again necessary to match the complexity of the model to the problem being solved.

[1] Here we are adopting a frequentist viewpoint. From a Bayesian perspective there may be, in principle at least, no need to limit model complexity [6].

2 Complexity Control

In this paper we consider four common approaches to complexity control based respectively on network architecture selection, regularization, early stopping, and training with noise. We show that there are relationships and similarities between these techniques, and we shall argue that, for the majority of applications, the technique of regularization should be the method of choice. For each of these techniques there is some parameter (number of hidden units, regularization coefficient, number of training steps, or noise variance) which controls the model complexity. Its value is chosen to maximize the generalization performance of the model, using techniques such as cross-validation [2].

2.1 Architecture Selection

The complexity of a neural network model is governed by the number of degrees of freedom, which in turn is controlled by the number of adaptive parameters (weights and biases) in the network. Changes to the network architecture which alter the number of parameters can therefore be used to control complexity. One of the simplest approaches involves the use of networks with a single hidden layer, in which the the number of parameters is controlled by adjusting the number of hidden units. Other approaches involve growing or pruning the network structure as part of the training process itself.

2.2 Regularization

Network training corresponds to the minimization of an error function E. The technique of regularization encourages smoother network mappings by adding a penalty Ω to the error function to give

$$\widetilde{E} = E + \nu\Omega \tag{1}$$

where ν is a parameter called a regularization coefficient which controls the model complexity. In a Bayesian context, the regularization function corresponds to the negative logarithm of the prior distribution of network weights.

One of the simplest forms of regularizer is called *weight decay* and consists of the sum of the squares of the adaptive parameters

$$\Omega = \frac{1}{2}\|\mathbf{w}\|^2 \tag{2}$$

where $\mathbf{w}$ denotes the vector of all weights and biases in the network. Regularizers of this form encourage the network function to be smooth. In conventional curve fitting, the use of this form of regularizer is called *ridge regression*. The gradient of the regularized error function is given from (1) and (2) by

$$\nabla\widetilde{E} = \nabla E + \nu\mathbf{w} \tag{3}$$

We can gain insight into the behaviour of the weight decay regularizer by considering the particular case of a quadratic error function. A general quadratic error can be written in the form

$$E = E_0 + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{4}$$

where the Hessian matrix $\mathbf{H}$ is positive definite and constant, and $\mathbf{w}^*$ corresponds to the minimum of the error function. In the presence of the regularization term, the minimum moves to a point $\widetilde{\mathbf{w}}$ which, from (3) and (4), satisfies

$$\mathbf{H}(\widetilde{\mathbf{w}} - \mathbf{w}^*) + \nu\widetilde{\mathbf{w}} = 0 \tag{5}$$

We can better interpret the effect of the weight decay term if we rotate the axes in weight space so as to diagonalize the Hessian. This is done by considering the eigenvector equation for the Hessian given by

$$\mathbf{H}\mathbf{u}_j = \lambda_j \mathbf{u}_j \tag{6}$$

We can now expand $\mathbf{w}^*$ and $\widetilde{\mathbf{w}}$ in terms of the eigenvectors to give

$$\mathbf{w}^* = \sum_j w_j^* \mathbf{u}_j \qquad \widetilde{\mathbf{w}} = \sum_j \widetilde{w}_j \mathbf{u}_j \tag{7}$$

Combining (5) and (7), and using the orthonormality of the $\{\mathbf{u}_j\}$, we obtain the following relation between the minima of the original and the regularized error functions

$$\widetilde{w}_j = \frac{\lambda_j}{\lambda_j + \nu} w_j^* \tag{8}$$

The eigenvectors $\mathbf{u}_j$ represent the principal directions of the quadratic error surface. Along those directions for which the corresponding eigenvalues are relatively large, so that $\lambda_j \gg \nu$, (8) shows that $\widetilde{w}_j \simeq w_j^*$, and so the minimum of the error function is shifted very little. Conversely, along directions for which the eigenvalues are relatively small, so that $\lambda_j \ll \nu$, (8) shows that $|\widetilde{w}_j| \ll |w_j^*|$, and so the corresponding components of the minimum weight vector are suppressed. This effect is illustrated in Figure 1. Thus only those directions for which $\lambda_j \gg \nu$ contribute significantly to fitting the data, with the weight vector components in remaining directions being set to small values by the regularizer. The quantity

$$\gamma = \sum_j \frac{\lambda_j}{\lambda_j + \nu} \tag{9}$$

defines the *effective* number of parameters in the model [5]. Clearly as ν is increased so the effective number of parameters is reduced.

The simple weight decay regularizer given by (2) suffers from the problem of being inconsistent with known scaling properties of the network function [2]. For example, if a linear rescaling is applied to the input data, this can be absorbed by a linear rescaling of the first-layer weights (and biases) such that the network outputs are unchanged. Similarly, a linear transformation of the output variables of the network can be achieved by linearly rescaling the final-layer weights (for a network with linear output units). The simple weight decay regularizer, however, would arbitrarily favour configurations with smaller weight values. We should instead divide the weight and bias parameters into groups and assign a separate regularizer (with its own regularization coefficient) to each such group [2]. Appropriate groups could, for instance, consist of the first-layer weights, the first-layer biases and the second-layer weights.

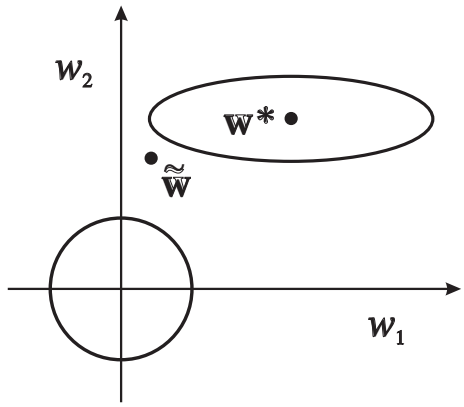

Figure 1: Illustration of the effect of a simple weight decay regularizer on a quadratic error function. The circle represents a contour along which the weight decay term is constant, and the ellipse represents a contour of constant un-regularized error. The effect of the regularizer is to shift the minimum of the error function from $\mathbf{w}^*$ to $\widetilde{\mathbf{w}}$. This reduces the value of w_1 at the minimum significantly since this corresponds to a small eigenvalue, while the value of w_2, which corresponds to a large eigenvalue, is hardly affected [5].

2.3 Early Stopping

Another well-known approach to controlling model complexity is called *early stopping*. During a typical training session, the error measured with respect to the training data set decreases as a function of the number of iterations. However, the error measured with respect to independent data often shows a decrease at first, followed by an increase as the network starts to over-fit. Early stopping aims to achieve the best generalization by setting the weight vector to the value which gives the smallest error on new data.

We can demonstrate a relationship between early stopping and regularization for the case of a quadratic error function of the form (4). Suppose the initial weight vector $\mathbf{w}^{(0)}$ is chosen to be at the origin, and is updated using simple gradient descent

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \nabla E \tag{10}$$

where τ denotes the step number, and η is the learning rate (which is assumed to be small). Substituting (4) into (10) and making use of (7) we then obtain

$$w_j^{(\tau)} - w_j^* = (1 - \eta\lambda_j)(w_j^{(\tau-1)} - w_j^*) \tag{11}$$

After τ steps of gradient descent we then have

$$w_j^{(\tau)} = \{1 - (1 - \eta\lambda_j)^\tau\}\, w_j^* \tag{12}$$

Note that, as $\tau \rightarrow \infty$ this gives $\mathbf{w}^{(\tau)} \rightarrow \mathbf{w}^*$ as expected, provided $|1 - \eta\lambda_j| < 1$. Now suppose that training is halted after a finite number τ of steps. Then it

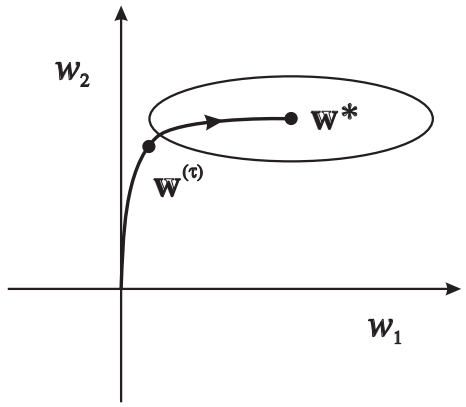

Figure 2: A schematic illustration of why early stopping can give similar results to weight decay in the case of a quadratic error function.

is easily seen that

$$w_j^{(\tau)} \simeq w_j^* \quad \text{when} \quad \lambda_j \gg (\eta\tau)^{-1} \tag{13}$$

$$|w_j^{(\tau)}| \ll |w_j^*| \quad \text{when} \quad \lambda_j \ll (\eta\tau)^{-1} \tag{14}$$

Comparison with the corresponding result (8) obtained using weight decay regularization shows that early stopping and regularization lead to similar solutions, and that the quantity $(\eta\tau)^{-1}$ is analogous to the regularization parameter ν. This result also show that the effective number of parameters in the network, as defined by (9) with ν replaced by $(\eta\tau)^{-1}$, grows as the training progresses [9].

The relationship between early stopping and regularization is illustrated in Figure 2. A weight vector which starts at the origin and moves according to the local error gradient will follow a trajectory shown by the curve. By stopping training early, a weight vector $\mathbf{w}^{(\tau)}$ is found which is qualitatively similar to that obtained with a simple weight decay regularizer as can be seen by comparing with Figure 1.

2.4 Training With Noise

The final technique which we consider for controlling the (effective) complexity of a feed-forward network is based on the addition of noise to the input vectors during training. A theoretical analysis of this technique [3] shows that, for small values of the noise variance, training with noise is equivalent to the use of regularization (and no added noise) where the regularizer depends on the derivatives of the network function. Here we summarize the analysis for the case of a sum-of-squares error.

For a network function $y(\mathbf{x})$ with input vector $\mathbf{x}$ and a single output y the sum-of-squares error can be written in the form

$$E = \langle \{y(\mathbf{x}) - t\}^2 \rangle_{\mathbf{x},t} \tag{15}$$

where t denotes the target variable and $\langle \cdot \rangle$ denotes the expectation. Now suppose that each time an input vector $\mathbf{x}$ is presented to the network a random vector $\boldsymbol{\epsilon}$ is added first. The error function is then given by an average over both the distribution of data and the distribution of noise in the form

$$\widetilde{E} = \langle \{y(\mathbf{x} + \boldsymbol{\epsilon}) - t\}^2 \rangle_{\mathbf{x},t,\boldsymbol{\epsilon}} \tag{16}$$

For small values of $\boldsymbol{\epsilon}$ we can make use of a Taylor expansion of the form

$$y(\mathbf{x} + \boldsymbol{\epsilon}) = y(\mathbf{x}) + \boldsymbol{\epsilon}^{\mathrm{T}} \nabla y(\mathbf{x}) + \frac{1}{2} \boldsymbol{\epsilon}^{\mathrm{T}} \nabla \nabla y(\mathbf{x}) \boldsymbol{\epsilon} + \mathcal{O}(\boldsymbol{\epsilon}^3) \tag{17}$$

We now substitute (17) into (16) and assume that the noise distribution has zero mean, and a covariance matrix which is proportional to the unit matrix $\mathbf{I}$ so that

$$\langle \boldsymbol{\epsilon} \rangle = 0 \tag{18}$$

$$\langle \boldsymbol{\epsilon} \boldsymbol{\epsilon}^{\mathrm{T}} \rangle = \nu \mathbf{I} \tag{19}$$

We then obtain the effective error function in the form[2] [10]

$$\widetilde{E} = E + \nu \langle \|\nabla y\|^2 \rangle_{\mathbf{x}} + \nu \langle (y - t) \nabla^2 y \rangle_{\mathbf{x},t} \tag{20}$$

We are interested in finding the network function which minimizes this error. By functional differentiation of (20), and making use of (15), we see that this function takes the form

$$y(\mathbf{x}) = \langle t | \mathbf{x} \rangle_t + \mathcal{O}(\nu) \tag{21}$$

where $\langle t | \mathbf{x} \rangle_t$ denotes the conditional average of the target data, also known as the *regression* of the target variable. If we now back-substitute (21) into (20) we see that the term involving the second derivatives of the network function vanishes to $\mathcal{O}(\nu)$. For a data set of input vectors $\mathbf{x}^n$ and corresponding target values t^n (where $n = 1, \ldots, N$) the effective error function can then be written in the form [3]

$$\widetilde{E} = \sum_{n=1}^{N} \{y(\mathbf{x}^n) - t^n\}^2 + \nu \sum_{n=1}^{N} \|\nabla y(\mathbf{x}^n)\|^2 \tag{22}$$

We therefore see that, for sufficiently small values of the noise variance ν, the minimization of a sum-of-squares error with noise added to the input data is equivalent to the minimization of the regularized error function (22) without the addition of noise. For linear network models, the regularizer in (22) reduces to the weight-decay regularizer of (2) but with the bias parameters omitted.

In Figure 3 we present an empirical comparison between training with noise (using ten copies of each data point and a Gaussian noise distribution) and the use of the regularized error function (22). The required gradients of the regularized error were evaluated efficiently by using an extended form of back-propagation [1]. These gradients were then used in a standard BFGS quasi-Newton non-linear optimization algorithm.

[2] A similar analysis was considered by Reed *et al.* [7] who arbitrarily discard the second derivative terms.

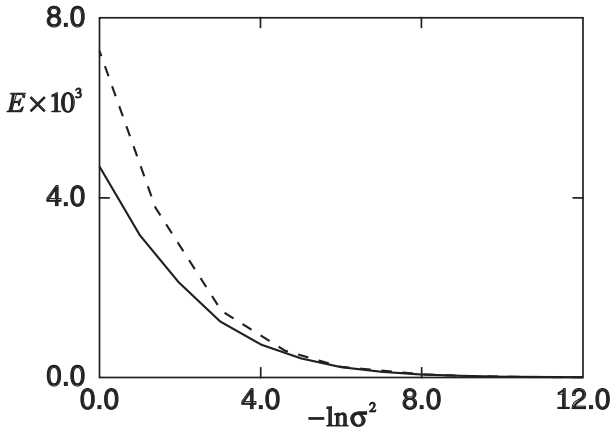

Figure 3: A plot of training error versus $-\ln \sigma^2$ (dashed curve) for training with noise having input variance σ^2. Also shown on the same scale is the corresponding plot of training error versus $-\ln \nu$ (solid curve) for the regularized error function given by given by (22) and noise-free data. The data set was taken from a problem concerned with the monitoring of oil flows along multi-phase pipelines [4].

3 Discussion

In this paper we have considered four approaches to the problem of complexity control in feed-forward networks. Network architecture selection changes the actual number of adaptive parameters in the network, while regularization controls the effective number of parameters. We have seen that early stopping also limits the effective number of parameters, while training with noise is equivalent (at small noise variance) to a derivative-based regularizer. It is natural to ask which of these four methods should be used in practice. Here we argue that, for most applications, techniques based on regularization are to be preferred Reasons for this include the following

1. One way to view complexity control in feed-forward networks is in terms of the trade-off between the bias and the variance of the network function. Regularizers which exploit prior knowledge can lead to significant reductions in variance while producing a relatively small increase in bias, thereby leading to improved generalization. For example, regularizers can be constructed which encourage invariance to translations and rotations in character recognition applications [8].

2. When regularization is used the effective complexity is controlled by one or more continuous parameters, which are generally much easier to optimize than a discrete quantity such as the number of hidden units. For instance, the evidence approximation to the Bayesian approach [5] allows the values of regularization parameters to be optimized (using simple re-estimation procedures) as part of the training algorithm, without the need for separate validation data.

3. The technique of early stopping is largely ad-hoc, and the solution found for the weights will depend on the path through weight space which will in turn depend on the initial randomly chosen weight vector as well as the parameters of the learning algorithm.

4. Training with noise, if used with batch optimization algorithms, requires that the data set be replicated many times, and this leads to a substantial increase in the computational cost of training the network. By contrast, direct minimization of a regularized error function leads to a relatively small increase in computational cost.

5. As we discussed in Section 2.2, a practical application will typically require the use of regularizers which are more sophisticated than simple weight decay and which are controlled by several independent regularization coefficients. The effects of such regularizers cannot easily be replicated using the other methods described in this paper.

For most practical applications the technique of regularization will be the preferred choice since it provides the most flexible form of model complexity control, it is computationally efficient, and it allows prior knowledge to be incorporated directly into the network training procedure.

References

[1] C. M. Bishop. Curvature-driven smoothing: a learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 4(5):882–884, 1993.

[2] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[3] C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

[4] C. M. Bishop and G. D. James. Analysis of multiphase flows using dual-energy gamma densitometry and neural networks. *Nuclear Instruments and Methods in Physics Research*, A327:580–593, 1993.

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

[6] R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, Canada, 1994.

[7] R. Reed, R. J. Marks II, and S. Oh. Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter. *IEEE Transactions on Neural Networks*, 6(3):529–538, 1995.

[8] P. Simard, B. Victorri, Y. Le Cun, and J. Denker. Tangent prop – a formalism for specifying selected invariances in an adaptive network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

[9] C. Wang, S. S. Venkatesh, and J. S. Judd. Optimal stopping and effective machine complexity in learning. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 303–310. Morgan Kaufmann, 1995.

[10] A. R. Webb. Functional approximation by feed-forward networks: a least-squares approach to generalisation. *IEEE Transactions on Neural Networks*, 5(3):363–371, 1994.