European Journal of Economic and Social Systems **14** N° 1 (2000) 1-16

Pruning neural networks by minimization of the estimated variance

PETER MORGAN[1], BRUCE CURRY[1] AND MALCOM BEYNON[1]

Abstract. – This paper presents a series of results on a method of pruning neural networks. An approximation to the estimated variance of errors, *V*, is constructed containing a supplementary parameter, α — the estimated variance itself being the limit of the function, *V*, as α tends to zero. The network weights are fitted using a minimization algorithm with *V* as objective function. The parameter, α, is reduced successively in the course of fitting. Results are presented using synthetic functions and the well-known airline passenger data. We find, for example, that the network can discover, in the course of being pruned, evidence of redundancy in the variables.

Classification Codes: C45, C63.

1. Introduction

Neural networks (NNs) are widely used for analysing and modelling data in a variety of business and economic applications and a number of surveys of such applications have been made, *e.g.* Wong *et al.* (1995). NNs can be thought of as a number of interconnecting units each capable of a rudimentary computation of its output based on its inputs. Modelled on the animal brain, they "learn" or are "taught" by feeding data into them. Their subsequent "learning" or adaptation to the data consists primarily in altering the strengths of network interconnections as represented by weights. Typically they are used in situations where the model underlying the data is not well understood and have gained a reputation as "black boxes" into which data can be fed with few prior assumptions and, once the data has been "learned", useful information such as predictions may be extracted - the essential information in the data having been reduced to the set of weights (adjustable parameters) inside the network. One then requires that the network is able to generalize from the data. There is a great diversity of architectures available to the user as well as many variants of training method. We shall be concerned in this paper with the feed-forward neural network with logistic hidden nodes (an example of which is shown later in

[1] Cardiff Business School, Aberconway Building, Colum Drive, Cardiff, CF1 3EU, Wales, UK.
Keywords: Neural network, pruning, generalization, penalty function, estimated variance.

Fig. 2). This type of network consists of a layer of input nodes which receive values of the independent variables and serve only to transmit these values to the network. These are succeeded by a layer or layers of nodes which sum the values coming from the inputs along any connecting links and output to the next layer the logistic function,

$$output = \frac{1}{1 + e^{-input}},$$

of this sum. These processed values then proceed, possibly through other layers of logistic nodes to the output layer nodes which, again, serve only to sum their inputs and transmit them to the outside world for comparison with corresponding values of the dependent variable. This network is then a graphical representation of a complicated function of logistic functions. The intermediate layers of logistic nodes are said to be the "hidden" layer or to contain "hidden" nodes. Since the logistic function is a threshold function (being somewhat like a smoothed Heaviside step function) and following the biological neuronal analogy, each hidden node can be thought of as "firing" (producing an output) when its summed input exceeds a critical value. The networks also employ bias nodes (shown as square nodes in Fig. 2) which serve only to provide a constant unit value and raise or lower this threshold according to the value of the weight on the link joining it to the hidden node in question. Because of this thresholding property, we can think of these networks as implementing "soft" or "fuzzy" switching or as "soft" logic devices.

The "black box" reputation of NNs has also made some workers rightly sceptical for at least two reasons. Firstly, even for a modest number of inputs (independent variables) and neural units, the number of interconnections and, therefore, adjustable weights becomes quite large. Secondly, in such a highly interconnected situation, even looking inside the "black box", it becomes very hard to put any interpretation on the values of those weights. Also, thirdly, excessive numbers of adjustable parameters bring with them the dangers of overfitting the data - the network no longer generalizing the data but fitting also its noisy idiosyncrasies. Although the potential of Neural Networks (NNs) as a data analysis tool is now beyond dispute, research is still required on a range of technical and practical issues related to this problem.

For the practitioner, the most important question to arise is the choice of network architecture. It would be more than useful to have systematic rules or guidelines to assist in choosing the number of nodes and weights, but in practice the position is less healthy. A survey article by Zhang *et al.* (1998) simply states that 'there is no clear-cut method for determining these parameters'. The principle of parsimony and associated model selection criteria are often invoked as an aid to model choice, *e.g.* Keuzenkamp and McAleer (1995). In the context of neural networks, this amounts to producing a network with as small a number of weights as possible which neither under- nor overfits the data.

Research continues on so-called 'weight elimination' or 'pruning' methods: see for example Cottrell *et al.* (1995). With NNs, the choice of model architecture is more difficult, given that the number of parameters is subject to 'combinatorial explosion'. It is in fact quite commonplace to see research where the number of weights clearly exceeds the number of data points being fitted. This paper suggests a new method of pruning a standard feedforward logistic network, whereby the algorithm is specifically rewarded for removing weights. The method involves a variation of the penalty function approach

for fitting constrained models to data in such a way that the pruning is carried out as a natural consequence of the fitting/learning process. In what follows, we explain our pruning method in both statistical and practical terms and proceed to test it on various data sets, including the airline data (Hyndman, 1998) commonly employed as a benchmark for forecasting techniques. Results are encouraging and suggest that further work would be desirable.

We hope, by means of these more parsimonious networks, to produce fits to data which will allow better out-of-sample prediction and perhaps to obtain sparsely connected networks which, by the logical switching analogy described above, can be interpreted in terms of simple logical rules.

2. A suggested pruning method

It is not proposed to review the literature relating to pruning/network restructuring methods here since this is covered admirably elsewhere (*e.g.* Reed, 1993). However, the problem amounts to applying one of these following methods.

1) Taking an existing network which satisfactorily fits or, perhaps, overfits the data and removing connections (and possibly whole nodes with their pendant connections) whilst not over-sacrificing goodness of fit.

2) Taking networks which are too small to fit the data (and therefore prone to specification error) and expanding them adding connections and/or nodes to improve the fit without overfitting the data.

3) Combining methods (1) and (2) above so that weights may be pruned, re-introduced or new ones introduced at various stages of the data fitting/learning process.

The method we have chosen to study is predominately of the Method (1) above. We take a large network and fit it to the data using a penalty function which biases the learning process towards smaller weights. We find that weights pruned in the earlier stages of the fitting process may, however, be re-introduced later. The method will not introduce new weights which were not in the starting model. We define the 'effective model size' in terms of the number of parameters to be fitted and use a testing function, in our case a Cauchy-type function, which "counts" the number of parameters which are equal to zero. The reasons for the specific form of this testing function are outlined below. Specifically, our method is based upon the minimization of an objective function, V, of the general form

$$V = \frac{\sum_{i=1}^{i=n} (y_i - f(\mathbf{x}_i, [w_1, w_2, w_3, \ldots w_m]))^2}{n - m + p} \quad \text{where} \quad p = \sum_{k=1}^{k=m} \left(\frac{\alpha^2}{\alpha^2 + w_k^2} \right), \qquad (2.1)$$

where

n = the number of data points,

m = the maximum number of model parameters,

$\mathbf{x}_i$ = the data input vectors,

w_k = the network weights,

α is a parameter of the algorithm,

$f(\ )$ is the output function of the network,

y_i are the dependent variable data values.

The function can be seen as a standard sum of squares error metric modified to take account of the number of parameters. In other words, a parsimony constraint is applied, but the constraint is expressed in quite general terms. When α tends to zero, V is, in fact, the best estimate of the variance of a Gaussian additive error term in the asymptotic case when n becomes infinite. The quantity p is, in the limit when α tends to zero, the number of weights which have been pruned out of the completely connected network. Since p has the form of a sum of Cauchy distribution functions, α can be interpreted as half the peak width of the distribution at half height and/or a tolerance on the weights, *i.e.* the absolute value of a weight which we would regard as being effectively zero. It is clear that one might have chosen another peaked function such as Gaussian functional form to define p and, in the limit of α tends to zero, we would obtain the same result for the number of pruned weights.

The use of penalty functions which simultaneously maximize the goodness of fit and the parsimony of the model is of course commonplace. The most popular functions such as AIC and BIC have a formal theoretical basis. In more practical terms they can be seen as simple modifications of a standard least squares error criterion, allowing in a simple way for the number of adjustable parameters. The downside, however, is that the use of any such criteria remains judgmental and they may give conflicting results, *e.g.* Chatfield (1998).

By way of an introductory non-neural network example, we shall examine the case of the function

$$y_i = a(1 + x_i) + be^{x_i} + \sigma E_G, \tag{2.2}$$

where a and b are constants, E_G is a standard normal random variable (a Gaussian noise term) and the other symbols are as in equation (2.1) above. This function is chosen such that $(1 + x_i)$ and e^{x_i} are "close" to each other in for small values of x_i and therefore we can expect substantial "trade off" between parameters when the same function with adjustable parameters in place of a and b is used to fit the data.

Ten values of this function for $x_i = -0.4 \ldots 0.5$ and $a = 0.5$, $b = 0.5$ were calculated and Gaussian pseudorandom "noise" with standard deviation, σ, added to them. The upper part of Figure 1 shows contour plots of the natural logarithm of the error function V defined in equation 2.1 using a value of α of 0.05 and a fitting function $y_i = par_1 (1 + x_i) + par_2 e^{x_i}$. The natural logarithm, $\ln(V)$, rather than V itself has been plotted solely in order to render the minima better in the contour plots. The plot indicates the positions of minima and the values of V at those minima. We can see that, for a small value of the added noise, the only minimum is that where $par_1 = 0.349$ and $par_2 = 0.639$. For smaller values of α and σ the minimum will, of course, approach the point (0.5, 0.5) representing the values originally used in the function in equation (2.1). As the noise level is increase, we see that this minimum disappears and two new minima appear very close to the axes where one or other of the fitting parameters, par_1 and par_2 are zero. As α tends to zero, these points would tend to lie exactly on axis and we could say that an adjustable parameter has been pruned exactly. Where there is more than one minimum, the

minimum with the lowest value of V (the global minimum) would give the best fit to the data in the sense that it minimizes the estimated variance of the error term (when α tends to zero). For the higher noise levels, the difference between the two component parts of the function in equation (2.2) has been swamped by the added random noise.

It is interesting to compare the square roots of these minimum values of V with that of σ the standard deviation of the noise added to the data set. We find that these are quite close to each other as is shown in Figure 1. We can interpret this as indicating that the method is neither drastically underfitting or overfitting the data. We can also see in the figure the broadening effect of noise on minima of sum-of-squares based error surfaces – the minimum in the first plot is far more steep-sided than the succeeding ones. Another feature of the plots is the shallow "valleys" which lie along the axes where the denominator of V is becoming larger as one of the fitting parameters approaches zero.

In the lower part of Figure 1, we plot ln(V) again – this time for the same fitting function but with one of the constants, a, in the data generating function (in Eq. (2.2)) set to zero. We are thus testing to see whether the method can "discover", despite the noise, that there is only a single component present. For, a fairly noisy data set, we find that two minima appear both on the axes. The minimum with the lowest value of V corresponds to

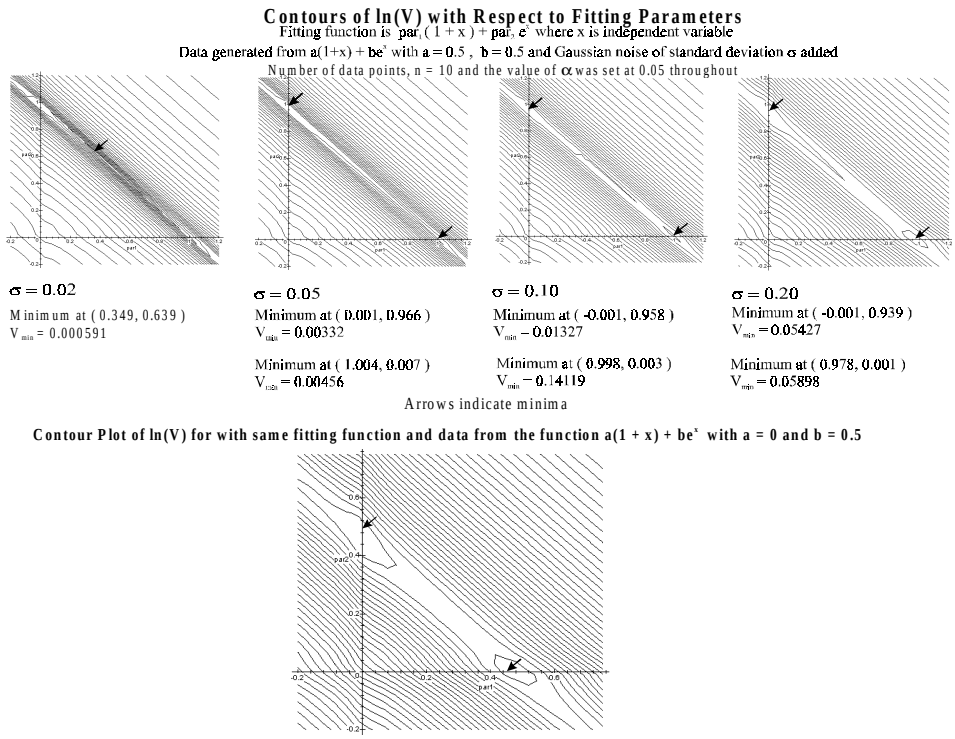


Fig. 1. *Contour diagrams for error function, V, for a simple case.*

par_1 = 0.001 and par_2 = 0.461 – a good approximation to the original value of b = 0.5 used to generate the data. In contrast, the values for the parameters at the minimum of the sum-of-squares are totally different with a higher value of *V*.

In the next section we examine some experiments on simulated data sets using feed-forward neural networks with logistic hidden nodes as fitting functions.

3. Implementation

Optimization was carried out using a variant of the Nelder-Mead or 'Polytope' algorithm, which has been employed in our previous work in preference to backpropagation (Curry and Morgan, 1997). The approach adopted is similar that used in the SUMT (Sequential Unconstrained Minimization Technique) for constrained optimisation described, for example, in Walsh (1975). Specifically,

A) α is initially set at a large value (say 10) and lower boundary set for its value.

B) A polytope optimization of the quantity *V* over the weights is carried out until an apparent optimum is reached.

C) The value of α is then reduced by a fixed proportion (by 50% in this work).

D) The value of α is checked against its lower limit. If it is not yet reached, we guard against the method becoming trapped in some subspace of the weight space by rotating the polytope randomly about the vertex with lowest objective function and return to Step B– otherwise the final values are output.

In the above, we use the terminology "apparent optimum" since local minima are quite commonly encountered in these situations of non-linear optimisation of sums-of-squares type functions, but we are also likely to encounter regions of almost zero gradient of the objective function. These are what White (1989) has called "flats", and are extremely common in neural networks with logistic activation functions, where trade-off between weights is very likely. In the idealized case where the trade-off between weights is perfect and the best fit value of the error function is attained, the global minimum will be a 'flat' and we will have some redundant weight or weights. The most parsimonious network would then lie at the intersection of one or more vertical planes in the error-weight axes and the flat region of the error surface. In practice, of course, the trade-off will be imperfect and the flats will not be truly flat. It would be left to some statistical test of significance between the constrained and unconstrained models to decide whether over-pruning is leading to specification error (assuming of, course, the validity of the test in the non-linear case where there may be degrees of freedom left when we have fitted our model). The point of pruning is, of course, to "solve" the problem of these flats by introducing a parsimony constraint. Another approach to the fitting process is to allow the optimization to choose its own value of α or α^2 and to add a term in α or α^2 to *V* to ensure that there is always a tendency for α to be reduced. Successive refits would then be made to ensure that some optimum had been reached. This was tried but problems were encountered in choosing a suitable weighting factor for α relative to *V* - the trade-off between sum of squares reduction and reduction of α becomes an issue. However, this method does have the advantage that the technique need not get trapped in some local minimum by the progressive reduction of α, since the optimisation can obviously choose to increase α. Overall, the method provided some good results.

Another problem with these optimizations arises in situations, commonly seen in NN applications, where the number of weights exceeds the number of data points and where it is quite feasible (and very likely, notwithstanding the non-linearity of the fitting function) that the value of the sum of squared deviations will drop to zero within computational accuracy. The mechanism whereby V is reduced through an increase in p (more weights being set to zero) is thus thwarted since the change in p will have comparatively little impact when the sum of squares in the numerator is already reduced to near round-off level. A combination of these approaches, where α is set equal to some multiple of the unadjusted sum of squares KS^2 (where S^2 is the unadjusted sum of squares and K is a constant - say 10), was then used and found to be satisfactory. During the course of the optimisation α is thus naturally reduced with the sum of squares. Again, in the spirit of SUMT, K is progressively reduced - say by 50% at each successive refit following an initial convergence of the polytope method. Ideally, and as hinted at above, a further development might be to choose the initial value of K through some knowledge of the geometry of the sum of squares surface close to the optimum. In the situation described above, where the maximum number of weights exceeds the number of data points (*i.e.* $n < m$) we might expect the possibility of the effective number of degrees of freedom, the quantity $(n - m + p)$, being negative! A check is included to ensure that this can never happen. This quantity is monitored before the estimated variance is calculated and, if less than a specified small positive number (10^{-4} for the results of this paper), it is set to that preset value. Experiments were carried out to see the effect of the choice of this quantity and it was found that, when this "floor" for the effective number of degrees of freedom was set at 1 the estimated variance was often rapidly reduced to zero whereas, with the choice of 10^{-4} which we have adopted, the network is pruned sufficiently to produce a non-zero estimated variance.

If values of

$$\frac{\partial^2(S^2)}{\partial w_k^2},$$

the Hessian matrix elements with respect to the sum of squares function, were available we might choose to invert this, define individual values of α for each weight and relate these to the standard errors of the fitted and the diagonal elements of the inverted Hessian. To do this, aside from the calculation of the Hessian, would require a doubling of the number of fitting parameters with a consequently massive increase in computation time and, for the algorithm presented here, we have chosen not to attempt this and it is, perhaps, an issue for further work.

In tests on functions with varying added noise we found that the final estimated variance was often quite close to the level of noise added to the pure function. This of course may well be entirely fortuitous but opens up interesting issues as to how these penalty function pruning methods strike a balance between over- and underfitting.

4. Results

4.1. *The effect of added noise and sample size*

Table 1 shows results for this latter method using random samples of various sizes from the test function,

$$F1(x_1,x_2) = 0.1\ (x_1 + 2\ x_2)_2 + (x_1 + 2\ x_2) + 1 + \sigma_G\ E_G, \tag{4.1}$$

taken over the interval $(-1, +1) \times (-1, +1)$ in x_1 and x_2 from various randomly chosen starting points and different amounts of noise where E_G is a standard Gaussian pseudo-random varable with σ_G being the standard deviation of the additive noise. This test function was chosen to see whether the method could "discover" the fact that, in the noise-free case where $\sigma_G = 0$, the function is degenerate with respect to the input variables, x_1, x_2 . The neural net being fitted in this case is of a 2:2:2:1 configuration with 2 input nodes, 1 output node and two hidden layers of 2 nodes each - having 14 adjustable weights (including biases) in total.

TABLE 1. Dependence of fitted estimated standard deviation and number of weights pruned on sample size and level of added pseudorandom noise.

Standard Deviation of Noise (σ_G) / **Number of points**	**0**	**10^{-3}**	**3×10^{-3}**	**10^{-2}**	**3×10^{-2}**	**10^{-1}**
10	7	6	7	7	7	5
	1.730×10^{-3}	1.344×10^{-3}	3.286×10^{-3}	1.920×10^{-2}	2.294×10^{-2}	1.253×10^{-1}
20	5	1	2	2	5	2
	7.874×10^{-5}	7.431×10^{-4}	1.786×10^{-3}	5.894×10^{-3}	2.196×10^{-2}	2.292×10^{-2}
50	0	27	4	1	5	6
	1.300×10^{-5}	1.001×10^{-3}	3.125×10^{-3}	8.997×10^{-3}	3.105×10^{-2}	1.058×10^{-1}
100	0	0	2	3	5	2
	4.914×10^{-5}	9.629×10^{-4}	2.750×10^{-3}	9.449×10^{-3}	2.810×10^{-2}	9.017×10^{-2}
200	3	3	2	1	3	3.03*
	4.852×10^{-4}	9.920×10^{-4}	2.966×10^{-3}	9.812×10^{-3}	2.876×10^{-2}	9.672×10^{-2}

* The non-integral value of this entry reflects the fact that a weight is close, but not equal to, to zero.

In Table 1, the first value in each cell of the table corresponds to the number of weights which have been effectively set to zero or pruned by the procedure. The second value is the estimated standard deviation of the noise which is

$$\sqrt{\frac{S^2}{n - m + p}}$$

for the pruned model. The figure quoted is the lowest value of the objective function out of five optimizations started from different random initial points in weight space. These random starts are also different for each of the cases (number of data values and noise level) studied. In each case, the learning sets of points are randomly drawn from *F1* in equation (4.1) over the unit square $(-1, +1)$, $(-1, +1)$ in each of the two input variables. A value drawn from a Gaussian pseudorandom distribution is then added to each point to give the training set which was used without any prior scaling. For the 50, 100 and 200 point cases it is clear that the values of the estimated standard deviations are very close to

those of the noise indicating that little overfitting has occurred. Also, for the 10 point and 20 point cases, some overfitting and underfitting seemed to occur.

There seemed to be little relationship between the noise level and the number of weights effectively set to zero. The probable reason for this was that, in the presence of local minima, the influence of the starting point in weight space was most likely dominant and masked any relationship between the number of weights pruned, the noise level and the number of data points. In view of this, a further numerical experiment was conducted where the weights were initialized at the same values at each run. The question was posed at the beginning of this section as to whether the algorithm could reflect, within its pruned weight structure, the fact that the noise-free function was degenerate with respect to x_1 and x_2, *i.e.* that effectively the function can be expressed in terms of a single variable $(x_1 + 2x_2)$. Figure 2 indicates the weights in terms of the thickness of the weight branch for a fit of the 2:2:2:1 network to 100 random points from test function *F1*.

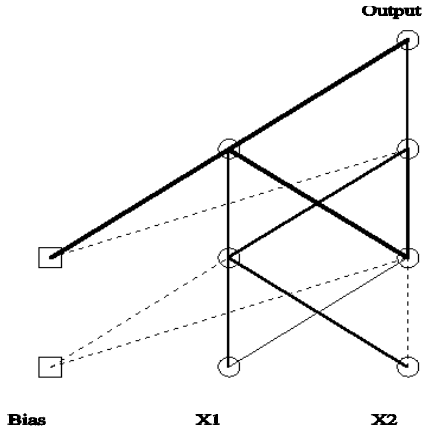


Fig. 2*. Network weight diagram for fit to test function* F1 *in equation (4.1).*

In Figure 2, weights less than 10^{-5} in magnitude are shown as dashed lines and we can see that the network makes up a "new" variable at the hidden layer level as a 1:2 combination of x_1 and x_2. In fact, we found that the ratio of these input weights were indeed very close to 1:2 as can be seen in Table 2 by comparing weights 2 and 3.

Another useful comparison which can be made lies in fitting the unpruned model, which has m weights and therefore, nominally, $(n - m)$ degrees of freedom. In Table 3, we can see the results of this where the unpruned model has been run from a starting point which is the finishing point for the pruned model.

The values in the third row of Table 3 correspond to the estimated standard deviation

$$\sqrt{\frac{S^2}{n-m}}$$

for the full model where no pruning has taken place and p is therefore set to zero. The initial weights for the full model are set at values close to those for the final refit of the

TABLE 2. Weights corresponding to Figure 2.

Successive layers involved	Weight joining successive layers	Weight value to 4 sig. figs.
Input to first hidden layer	1 Bias to first hidden unit	3.661×10^{-9}
	2 Input X1 to first hidden unit	0.2775
	3 Input X2 to first hidden unit	0.5564
	4 Bias to second hidden unit	3.429×10^{-8}
	5 Input X1 to second hidden unit	1.903×10^{-5}
	6 Input X2 to second hidden unit	-8.865×10^{-6}
First hidden layer to second hidden layer	7 Bias to first hidden unit	-33.95
	8 First hidden unit to first hidden unit	3.498
	9 Second hidden unit to first hidden unit	58.66
	10 Bias to second hidden unit	3.210×10^{-10}
	11 First hidden unit to second hidden unit	-9.439
	12 Second hidden unit to second hidden unit	60.50
Second hidden layer to output weights	13 First hidden unit to output	80.99
	14 Second hidden unit to output	-3.357

TABLE 3. Estimated standard deviations for pruned and unpruned models for the 50 point case.

Standard Deviation of noise	0	10^{-3}	3×10^{-3}	10^{-2}	3×10^{-2}	10^{-1}
Estimated standard deviation of error for pruned model	1.300×10^{-5}	1.001×10^{-3}	3.125×10^{-3}	8.997×10^{-3}	3.105×10^{-2}	1.058×10^{-1}
Estimated standard deviation of error for unpruned model	1.291×10^{-5}	1.074×10^{-3}	3.200×10^{-3}	9.065×10^{-3}	2.755×10^{-2}	1.090×10^{-1}

pruned model. It is apparent that the values are very similar. The values for the case of zero added noise represent a situation where the model is slightly underspecified whereas, in the other results, any residual "signal" has been completely swamped by the added noise.

4.2. *The performance of the algorithm on a weak quadratic*

In this second experiment, a function of two variables was used having a small quadratic component superimposed on a linear background with constant. This function was chosen as it is representative of commonly occurring situations of near linearity. The actual test function was

$$F2(x_1, x_2) = 0.1\, x_1^2 + 0.1\, x_1\, x_2 + 2\, x_1 + 3.2\, x_2 + 1 + \sigma_G\, E_G. \tag{4.2}$$

The algorithm was run on this target function for the 100 point case with $\sigma_G = 0.01$ for the added noise. Other runs were then made, each starting from the set of weights to

which this run converged, for various numbers of data points and levels of added noise as in Table 1. The results are shown in Table 4 below.

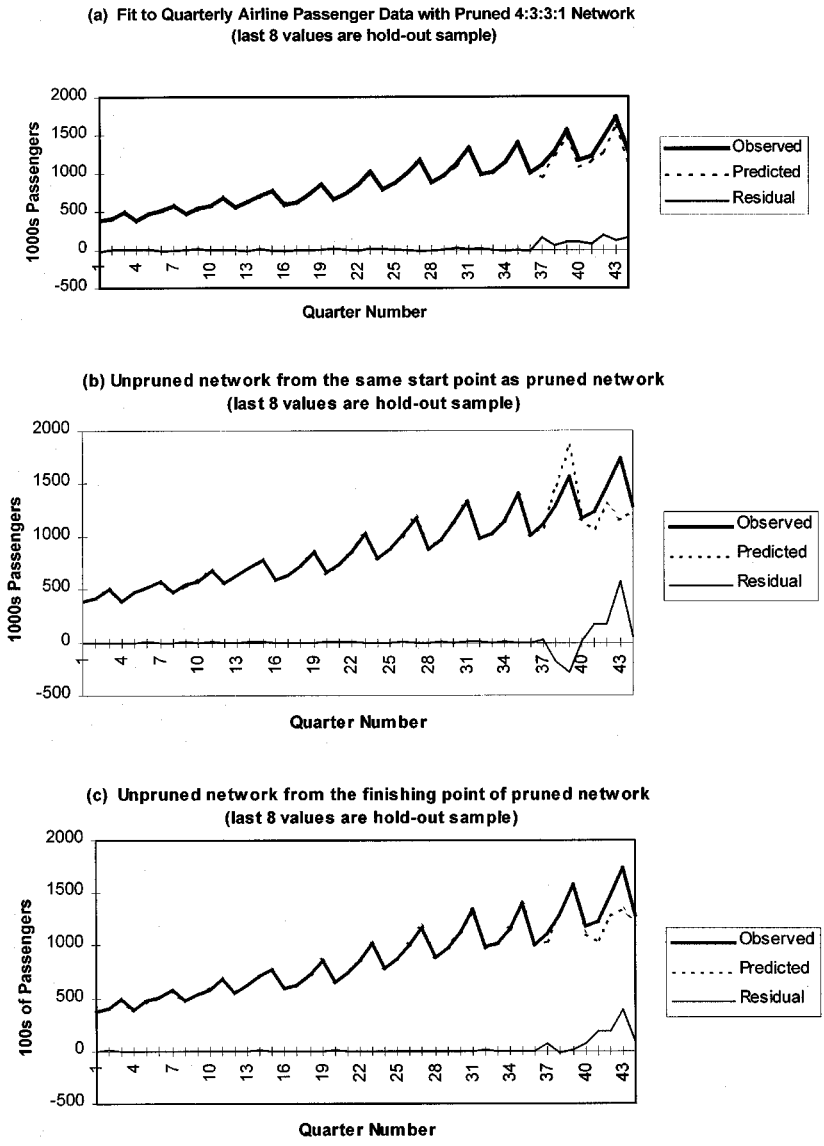
TABLE 4. Numbers of weights pruned and estimated standard deviations of error for a 2:5:1 network model.

Number of points \ Standard Deviation of Noise (σ_G)	**0**	**10^{-3}**	**3×10^{-3}**	**10^{-2}**	**3×10^{-2}**	**10^{-1}**
20	5 9.49×10^{-5}	4 9.71×10^{-4}	5 1.97×10^{-3}	6 5.95×10^{-3}	5 2.10×10^{-2}	7 6.24×10^{-2}
50	3 4.61×10^{-5}	3 1.19×10^{-3}	7 3.35×10^{-3}	6 1.07×10^{-2}	8 3.27×10^{-2}	7 9.76×10^{-2}
100	2 4.77×10^{-5}	2 9.45×10^{-4}	3 2.69×10^{-3}	3 8.92×10^{-3}	3 2.66×10^{-2}	4 8.77×10^{-2}
200	1 5.46×10^{-5}	2 9.93×10^{-4}	1 2.99×10^{-3}	3 9.93×10^{-3}	6 2.96×10^{-2}	9 9.74×10^{-2}

Each entry in Table 4 was obtained from single optimisations from the same starting point – this point being the finishing point of the algorithm for the case with 100 data points and 0.01 added noise. The preparation of the training data set was as above. We can see that, again (except in the case of zero added noise as above in Tab. 3), the noise levels are very similar to the estimated standard deviation of the errors showing that the method is not over- or under-fitting to any great degree.

4.3. *A time series example*

The autoregressive modelling of time series with neural networks has been the subject of a number of studies, *e.g.* Chatfield (1998) and a recent comprehensive review by Zhang *et al.* (1998). To see how the method herein might perform in this context, the well-known airline passenger data, which is kindly made available by Hyndman (1998), was converted to quarterly from monthly data to provide a small data set of 44 values. This is of interest since it has been remarked that neural networks are not very useful in the case of short time series. Four previous lags, X_{t-3}, X_{t-2}, X_{t-1}, and X_t were used as inputs to predict X_{t+1}. The eight last values were held out and hence 32 usable points were available for fitting. A 4:3:3:1 network was fitted using the algorithm above after scaling by dividing each value by 300. A series of 10 fits from different random start points was made and the fit with the lowest value of the estimated variance is shown in Figure 3 with the weight diagram for the fitted network being shown in Figure 4. It is clear that the method has pruned out 13 superfluous weights. Another feature of the plot is the lack of bias weights to the first hidden layer nodes. This means that the model is taking simple linear combinations of the lagged values without a constant term. The model also prunes out



Figs 3. *Networks fitted to quaterly airline passenger data.*

one of the three second hidden layer nodes completely. A more strict test of the model is of course its prediction of the hold-out sample values.

Figures 3(a), (b) and (c) compare the observed, predicted and residual values for (a) this pruned model with those for, (b) a completely unpruned model which was started from the same starting point as the pruned model and (c) an unpruned model (with the full 30 weights) refitted from a starting point which was the finishing point for the

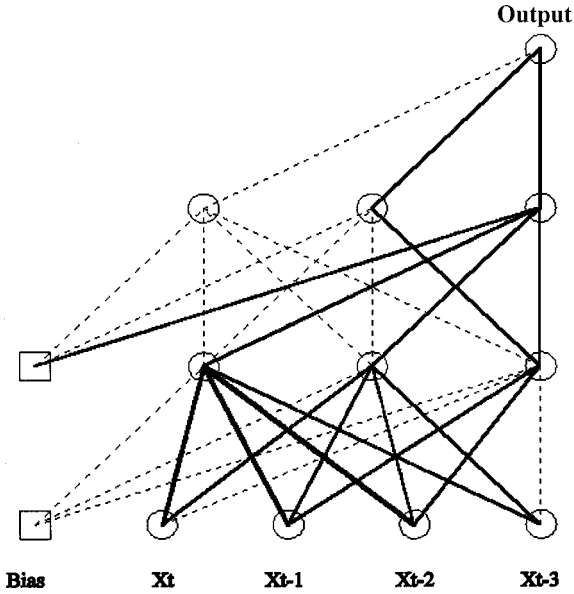


Fig. 4. *Neural network fit to quarterly airline passenger data.*

pruned model. The scaled estimated standard deviations of the error were 5.15×10^{-2}, 3.74×10^{-2} and 2.43×10^{-2} and respectively for (a), (b) and (c) above. The unpruned models are a seemingly better fit to the training set as the residuals in the diagram also show. However, as can be clearly seen, the prediction out-of-sample is much worse. Overfitting is often associated with poorer predictive capacity and pruning may serve to alleviate the problem. The estimated variance for the pruned model is higher than we might expect in comparison with the unpruned ones. As mentioned above, one of the problems with neural network fitting lies in the occurence of local minima and pathologically flat sections in the error surface. The use of a fitting method less susceptible to local minima (*e.g.* simulated annealing) should help to explore this situation further.

5. Conclusions, discussion and further work

5.1. *General remarks*

Neural networks are often treated as “black boxes” which “learn” the values of their weights (parameters). This opacity is, at once, a good and bad feature - good because little work seems to be required on the part of the operator and bad in the sense that the operator may never quite be certain whether overfitting has occurred and/or how good the trained network will be at predicting outside (or inside) the region of input space

within which the data is defined. This is very common where there are a large number of input nodes (variables) since even a modest number of hidden nodes in the first hidden layer will require a large number of connecting weights. In this last situation, when using backpropagation to train the network, there may well be unintentional use of the "early stopping" method (*e.g.* see Finnoff *et al.* (1993) where the network training is deliberately truncated to prevent overfitting). Our suggested pruning mechanism has some potential in this situation.

Another argument against neural networks is that their internal rules are seemingly impossible to fathom, *i.e.* it would not be possible to understand the mechanism inside the black box even if one could open it. For fully-connected networks, this is clearly a problem, but, if one were prepared to prune such a network whilst retaining a good fit to data, this objection might well be overcome and a much more illuminating and meaningful model should result. Pruning is therefore a crucial issue as far as practical applications of NNs are concerned.

We have seen that a NN fitting algorithm based on a minimization of the estimated variance can provide a method of pruning neural networks from a penalty function standpoint. For the functions *F1* and *F2* above we can see that the method avoids under- and over-fitting - at least in the case where $n > m$. The pruned networks, being sparser, are somewhat easier to interpret as the capacity of the 2:2:2:1 network to "discover" the functional degeneracy of *F1* shows. The results in the time series case also offer some encouragement in these respects. However, we should be wary of jumping to the conclusion that the internal structure of the network in the time series case necessarily corresponds to the underlying mechanism behind the data. The case where $m > n$ or even $m >> n$ need much further investigation since sparse networks offer hope for interpreting the network in terms of "simple" rules. However, in order to produce a such a sparse network, in the context of pruning complete networks, we will most often need to start from a completely connected network which is sufficiently large to accommodate all the rules we might want it to discover. This will inevitably mean using a very large number of nodes and hence weights. As mentioned above, another key issue is that of local minima and, in response to this, we are currently investigating the possibility of using optimisation methodologies such as genetic algorithms and simulated annealing which are better candidate methods for global minimization.

5.2. *The relationship with other fitting criteria*

The Akaike Information Criterion (AIC) is one of a number of measures of fit where the number of adjustable model parameters militates against profligate models and has been studied by a number of workers in the neural network context, *e.g.* Chatfield (1998). It can be stated in the form

$$\mathrm{AIC} = n \ln (S^2/n) + 2\, m_{par}\,, \tag{5.1}$$

where S^2 and n are the sum of squares and the sample size as before and m_{par} is the number of adjustable parameters in the model (model size).

Taking logarithms of the estimated variance, we get

$$\ln (V) = \ln \left(\frac{S^2}{n - m_{par}} \right) \ln(S^2/n) - \ln \left(1 - \frac{m_{par}}{n} \right)$$

where we have written $m_{par} = m - p$ using the definitions of m and p as previously.

If $n >> m_{par}$, then we can employ the series expansion of the last logarithmic term to first order to obtain

$$\ln (V) = \ln (S^2/n) + \frac{m_{par}}{n}$$

or, multiplying both sides by n to facilitate comparison with AIC in equation above

$$n \ln (V) = n \ln (S^2/n) + m_{par} \tag{5.2}$$

This we can see that, for this special case where $n >> m_{par}$, the minimization of V for a given data set with size n should provide a more parsimonious model which accords with the findings of this paper. However, the AIC penalizes excess model size twice as severely. All terms of the expansion of

$$\left(1 - \frac{m_{par}}{n} \right)$$

are of the same sign and as we decrease the value of n towards m_{par} we may expect to have to take more and more terms of this series into account. It would seem that the weighting against superfluous parameters will thus become heavier for smaller sample sizes at a given model size.

The method of this paper may be easily adapted to work with any model selection criterion which explicitly involves m_{par}. All that is required is the incorporation of the estimate of number of weights pruned,

$$p = \sum_{k=1}^{k=m} \left(\frac{\alpha^2}{\alpha^2 + w_k^2} \right),$$

into the expression for that criterion. Work has begun on the use of the Baysian Information Criterion (BIC) in this context and initial results are quite encouraging.

An obvious extension to this work will involve a comprehensive comparison between the work of this paper and the use of other model selection methodologies. To ensure a fair comparison, use should be made of synthetic data generated from a known model (*e.g.* ARIMA etc.) with added noise at various levels to assess both the relative performance of the various model selection criteria and the capacity of the network to uncover the structure in the model, e.g. which lags have been used to generate the data.

In particular, there is also scope for comparison with weight elimination methods such as the SSM method proposed by Cottrell *et al.* (1995). The latter eliminates weights on an individual basis using a pseudo *t*-statistic.

5.3. *Other activation functions*

Additionally, since nodes with the logistic activation function can mimic a constant bias source node (*e.g.* by having zero (pruned) input weights), the use of bi-polar functions such as the hyperbolic tangent, is an obvious area for further work in this pruning context.

Acknowledgments. The authors wish to thank their colleagues at Cardiff Business School for helpful suggestions and comments.

References

Chatfield C. (1998) Time Series forecasting with Neural Networks: a Comparative Study Using the Airline Data, *Applied Statistics* **47**, part 2., pp. 231-250.

Cottrell M., Girard B., Girard Y., Morgan, M., Muller C. (1995) Neural Modelling for Time Series: A Statistical Stepwise Method for Weight Elimination, *IEEE Transactions on Neural Networks* **6**, no. 6., pp. 1355-1364.

Curry B., Morgan P. (1997) Neural Networks: a Need for Caution, *Omega, International Journal of Management Science* **25**, no. 1, pp. 123-133.

Finnoff W., Hergert F., Zimmerman H.G. (1993) Improving Neural Networks by Non-convergent Methods, *Neural Networks* **6**, pp. 771-783.

Hyndman R. (1998) http//www.monash.edu.au/~hyndman/tseries.

Keuzenkamp H.A., McAleer M. (1995) Simplicity, Scientific Inference and Econometric Modelling, *The Economic Journal* **105**, no. 428, pp. 1-21.

Reed R. (1993) Pruning Algorithms - A Survey, *I.E.E.E Transactions on Neural Networks* **4**, no. 5, pp. 740-747.

Walsh G.R. (1975) *Methods of Optimization*, John Wiley, London.

White H. (1989) Learning in Artificial Neural Networks: A Statistical Perspective, *Neural Computation* **1**, pp. 425-464.

Wong B.K., Bodnovich T.A.E., Selvi Y. (1995) A Bibliography of Neural Network Business Applications Research: 1988-1994, *Expert Systems* **12**, no. 3, pp. 253-261.

Zhang G., Patuwo B., Hu M. (1998) Forecasting with Artificial Neural Networks: The State of The Art, *International Journal of Forecasting* **14**. no. 1, pp. 35-62.