International Journal of Neural Systems, Vol. 2, No. 3 (1991) 229–236

A FAST PROCEDURE FOR RETRAINING THE MULTILAYER PERCEPTRON

Chris Bishop
Neural Networks Group, AEA Technology, Harwell Laboratory, Oxfordshire OX11 0RA, UK

Received 22 May 1991
Revised 26 July 1991

In this paper we describe a fast procedure for retraining a feedforward network, previously trained by error backpropagation, following a small change in the training data. This technique would permit fine calibration of individual neural network based control systems in a mass-production environment. We also derive a generalised error backpropagation algorithm which allows an exact evaluation of all of the terms in the Hessian matrix. The fast retraining procedure is illustrated using a simple example.

1. Introduction

Multilayer perceptron neural networks are being exploited in an increasing number of applications as nonlinear control systems. The ability of such networks to generate a large class of nonlinear multivariate mappings,[1,2] as well as their speed in feedforward mode, makes possible a number of new approaches to the control of a nonlinear plant. However, in setting up controllers for mass produced items, there is often sufficient variation from one item to the next to make individual fine calibration of the control systems worthwhile. This represents a drawback for neural controllers since the training of multilayer perceptrons is a nonlinear optimisation problem, and is computationally very intensive. The training of a network for each system would therefore be very time consuming. In this paper we seek a fast procedure for training a network on a new dataset, given the corresponding weights and thresholds for a network of the same topology obtained for a dataset differing by a small amount from the new one. This retraining procedure allows the neural network to be set up quickly once a set of calibration measurements have been made on the plant.

In Sec. 2 we describe the fast one-step method for retraining the network. Some of the mathematical details of the learning algorithm are presented in the Appendix. In particular, we derive an extended backpropagation algorithm which allows the full Hessian matrix to be evaluated for a network of arbitrary feedforward topology. The retraining algorithm is illustrated in Sec. 3 using simulation results from a simple problem, and in Sec. 4 we contrast this retraining method with standard second-order training procedures. A brief summary is given in Sec. 5.

2. Fast Retraining

Consider a feedforward network with N inputs and M outputs, in which the activation of each unit is given by a nonlinear function of the weighted sum of its inputs:

$$z_i = f(a_i), \qquad a_i = \sum_j w_{ij} z_j + \theta_i \,, \tag{1}$$

where z_i is the activation of unit i, w_{ij} is the synaptic weight between units i and j, and θ_i is a bias associated with unit i. The bias terms can be considered as weights from an extra unit whose activation is fixed at $z_k = +1$. From now on, the bias terms will be absorbed into the weight matrix, without loss of generality, thereby simplifying the algebra. The activation of output unit m ($m = 1, \ldots, M$) can be written

$$y_m = y_m(\{x_n; w_{ij}\}) \,, \tag{2}$$

where x_n is the activation of input unit n ($n = 1, \ldots, N$). The functional form of Eq. (2) depends on the topology of the network and will be assumed to be fixed. Training data consists of a set of input vectors x_{np} ($p = 1, \ldots, P$), and corresponding target vectors t_{mp}, where t_m is the target activation for output unit m, and p labels the pattern. The standard backpropagation procedure provides an efficient method for evaluating the derivatives of an error function with respect to the weights $\{w_{ij}\}$. These derivatives are then used in optimisation algorithms based on gradient descent, conjugate gradients, quasi-

Newton methods, etc., to minimise the error. Note that we use the term 'backpropagation' to describe the evaluation of the derivatives of the error function, (since it is here that errors are propagated backwards through the network), rather than the particular optimisation strategy of gradient descent. The usual choice of error function is the sum-of-squares error, defined by

$$E = \sum_p E_p \,, \tag{3}$$

where the error E_p for pattern p is given by

$$E_p = \frac{1}{2} \sum_m [y_{mp} - t_{mp}]^2 \,, \tag{4}$$

and $y_{mp} \equiv y_m(\{x_{np}; w_{ij}\})$.

Suppose the network has been trained using this data set, to give a minimum of E, and we wish to find a set of weights $\{\tilde{w}_{ij}\}$ which minimise the error $\tilde{E}$ corresponding to a new data set given by

$$\tilde{x}_{np} = x_{np} + \Delta x_{np} \tag{5}$$

$$\tilde{t}_{mp} = t_{mp} + \Delta t_{mp} \,, \tag{6}$$

where Δx_{np} and Δt_{mp} are small shifts in the input and target vectors respectively for pattern p. We write the new set of weights in the form

$$\tilde{w}_{ij} = w_{ij} + \Delta w_{ij} \,, \tag{7}$$

and we now seek to calculate Δw_{ij}. The error function for the new data set can be written

$$\tilde{E} = \sum_p \tilde{E}_p = \frac{1}{2} \sum_p \sum_m [y_m(\tilde{x}_{np}; \tilde{w}_{ij}) - \tilde{t}_{mp}]^2 \,. \tag{8}$$

A schematic illustration of the error functions E and $\tilde{E}$ is shown in Fig. 1. Using Eqs. (5), (6) and (7) we can Taylor expand in the Δ's to give

$$\tilde{E}_p = E_p + \sum_{\{ij\}} \frac{\partial E_p}{\partial w_{ij}} \Delta w_{ij} + \sum_m \frac{\partial E_p}{\partial t_{mp}} \Delta t_{mp}$$

$$+ \sum_n \frac{\partial E_p}{\partial x_{np}} \Delta x_{np} + \frac{1}{2} \sum_{\{ij\}} \sum_{\{kl\}} \frac{\partial^2 E_p}{\partial w_{ij} \partial w_{kl}} \Delta w_{ij} \Delta w_{kl}$$

$$+ \frac{1}{2} \sum_m \sum_{m'} \frac{\partial^2 E_p}{\partial t_{mp} \partial t_{m'p}} \Delta t_{mp} \Delta t_{m'p}$$

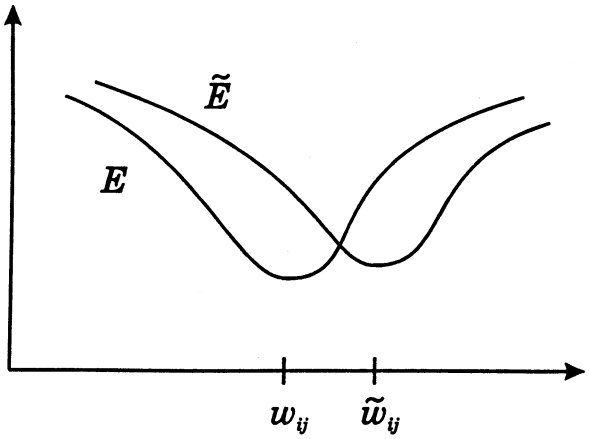

Fig. 1. Schematic illustration of the error function E defined with respect to the original data set, and the new error function $\tilde{E}$ defined with respect to the perturbed data set, as a function of the synaptic weights. The function $\tilde{E}$ has its minimum at $\tilde{w}_{ij}$, which differs by a small amount from the location w_{ij} of the minimum of E.

$$+ \frac{1}{2} \sum_n \sum_{n'} \frac{\partial^2 E_p}{\partial x_{np} \partial x_{n'p}} \Delta x_{np} \Delta x_{n'p}$$

$$+ \sum_n \sum_{\{ij\}} \frac{\partial^2 E_p}{\partial x_{np} \partial w_{ij}} \Delta x_{np} \Delta w_{ij}$$

$$+ \sum_{\{ij\}} \sum_m \frac{\partial^2 E_p}{\partial w_{ij} \partial t_{mp}} \Delta w_{ij} \Delta t_{mp}$$

$$+ \sum_n \sum_m \frac{\partial^2 E_p}{\partial x_{np} \partial t_{mp}} \Delta x_{np} \Delta t_{mp} + \mathcal{O}(\Delta^3) \,, \tag{9}$$

where $\{ij\}$ denotes the pair of indices corresponding to the units linked by the weight w_{ij}. We now demand that $\tilde{E}$ be minimised with respect to the w_{ij} so that

$$\frac{\partial \tilde{E}}{\partial \Delta w_{ij}} = 0 \,. \tag{10}$$

From Eqs. (3) and (9) we obtain, to lowest order in the Δ's,

$$\sum_p \frac{\partial E_p}{\partial w_{ij}} = 0 \,. \tag{11}$$

Equation (11) is identically satisfied since this is simply the condition that E be a minimum with respect to the $\{w_{ij}\}$. We therefore go to next order in the Δ's to give

$$\sum_p \sum_{\{kl\}} \frac{\partial^2 E_p}{\partial w_{ij} \partial w_{kl}} \Delta w_{kl} + \sum_p \sum_n \frac{\partial^2 E_p}{\partial x_{np} \partial w_{ij}} \Delta x_{np}$$

$$+ \sum_p \sum_m \frac{\partial^2 E_p}{\partial t_{mp} \partial w_{ij}} \Delta t_{mp} = 0 \,. \tag{12}$$

This can be rewritten in the form

$$\sum_{\{ij\}} A_{ij,kl} \Delta w_{ij} = -\Delta T_{kl} \,, \tag{13}$$

where we have defined

$$A_{ij,kl} \equiv \sum_p \frac{\partial^2 E_p}{\partial w_{ij} \partial w_{kl}} \,, \tag{14}$$

$$\Delta T_{kl} \equiv \sum_p \sum_n \frac{\partial^2 E_p}{\partial x_{np} \partial w_{kl}} \Delta x_{np} + \sum_p \sum_m \frac{\partial^2 E_p}{\partial w_{kl} \partial t_{mp}} \Delta t_{mp} \,. \tag{15}$$

Equation (13) represents a set of $\mathcal{N}$ inhomogeneous coupled linear equations (where $\mathcal{N}$ is the total number of weights and thresholds in the network) which determine the required Δw_{ij}. The various derivative terms appearing in Eqs. (14) and (15), including all elements of the Hessian matrix **A**, can be obtained using forward and backward propagation through the original network. Details are given in the Appendix. Inversion of the $\mathcal{N} \times \mathcal{N}$ matrix **A** then leads to the formal solution:

$$\Delta w_{ij} = -\sum_{\{kl\}} A^{-1}_{ij,kl} \Delta T_{kl} \,. \tag{16}$$

In practice the inversion would be performed using singular value decomposition to allow for possible ill-conditioning of the matrix **A**. Note that the derivatives of E_p depend only on the weights and the training data corresponding to the original network. Thus the evaluation of the derivatives, and the singular value decomposition of **A**, need only be done once. Given a set of values for Δx_{np} and Δt_{mp}, the setting up of each new network then simply involves evaluation of expressions (15) and (16), and is clearly a very fast procedure.

3. Simulation Results

In this section we present results from a software simulation of the fast retraining algorithm. For simplicity, we consider a network with a single input and a single output, so that the network transfer function can be visualised graphically. The network has a linear output unit, a single layer of hidden units, and no direct connection between the input and output units.

Consider the following function

$$y = x^{2+\lambda} \,. \tag{17}$$

We shall first of all train a network to learn this function for $\lambda = 0$, and then use fast retraining to generate a network whose transfer function corresponds to $\lambda \neq 0$. The function is chosen so that the effect of nonzero λ does not correspond to a linear rescaling of either x or y, and hence cannot be absorbed by a simple rescaling of the weights. Data is generated by sampling Eq. (17), with $\lambda = 0$, at 13 points equally spaced on the interval (0, 1), and is shown by the circles in Fig. 2. The initial training of the network was performed using the memoryless BFGS algorithm,[3] which has been found to be much faster and more robust than techniques based on gradient descent with momentum. A network with 13 hidden units was trained on this data, to give a residual error of 3.9×10^{-8}. The resulting network function is shown by the curve in Fig. 2. Note that, in this example, the number of hidden units is sufficient to allow the error to be made arbitrarily small. This is not an essential feature, and the method is equally applicable when the minimum of the error function occurs at some nonzero value.

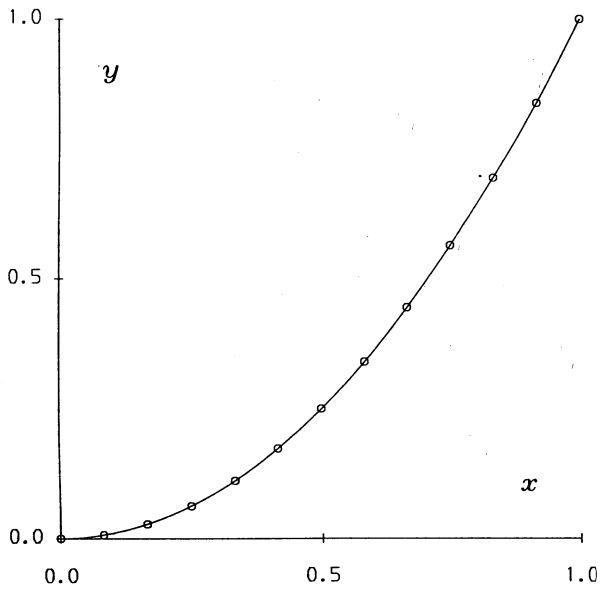

Fig. 2. Circles show the training data set obtained by sampling the function $y = x^{2+\lambda}$, for $\lambda = 0$, at equal intervals of x. The curve shows the corresponding network function after training, using the memoryless BFGS algorithm, to give an error of 3.9×10^{-8}.

The next step is the evaluation of the various derivatives of the error function and the inversion (singular value decomposition) of the Hessian. The parameter λ in Eq. (17) can now be varied and for each value of λ a new set of training data generated. The values of x chosen for the training data should be close to the original values, and we take them to equal the original values. This corresponds to a situation typical of a practical application of the method in which a fixed set of inputs is applied to the plant to be controlled and the corresponding outputs measured. For each new training set it is now a very fast procedure, using Eqs. (15) and (16), to calculate the corresponding set of network weights and thresholds which minimise the error function. The curve in Fig. 3 shows the behaviour of Eq. (17), as a function λ, for $x = 0.5$. The circles show the output of the retrained network, for the corresponding value of λ, with an input of $x = 0.5$. It is clear that the network mapping is changing so as to follow the modifications to the training set.

In Fig. 4 we show the behaviour of the error, defined by Eqs. (3) and (4), and evaluated using the new data set, as λ is varied. The triangles show the error between the new data set and the original network function, while the circles show the error between the new data set and the network function obtained from the corresponding retrained network. The substantial reduction in the error resulting from retraining is clear.

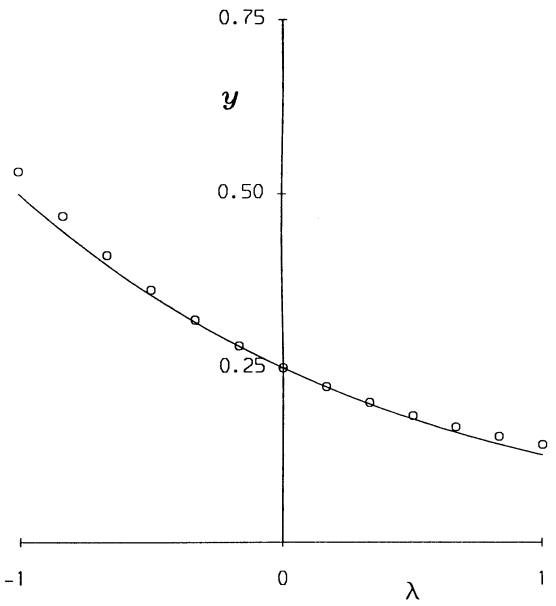

Fig. 3. The curve shows the function $y = x^{2+\lambda}$ as function of λ for $x = 0.5$. The circles show the output of the retrained network, at the corresponding value of λ, for an input of $x = 0.5$. This shows that retraining allows the network to generate a much improved prediction for the function, since without retraining the network output would be fixed at $y = 0.25$.

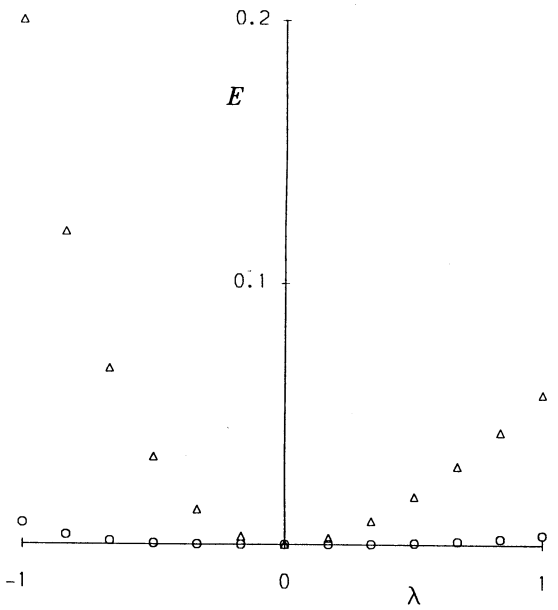

Fig. 4. Triangles denote the error function evaluated using the perturbed dataset together with the original network function, showing that the error gets rapidly worse as $|\lambda|$ is increased. Circles denote the error evaluated using the perturbed data together with the network function from the retrained network for the corresponding value of λ, demonstrating the substantial reduction in error resulting from retraining.

Although fast retraining is limited to small values of Δx and Δt, as a consequence of the expansion introduced in Eq. (9), it is clear from Figs. 3 and 4 that substantial variations in the training data can be accommodated. For instance, a value of $\lambda = 0.5$ produces a change in y (at $x = 0.5$) which is 7% of its full scale range.

As an indication of the speed of the fast retraining procedure, we compare the time taken to train the network for the new data corresponding to $\lambda = 0.1$, using a variety of methods. The original network weights (corresponding to $\lambda = 0.0$) gave an error of 9.0×10^{-4}, while the fast retraining procedure reduced this by some two orders of magnitude to 8.6×10^{-6}. For comparison we consider the time taken to reduce the error to the same level using BFGS starting from random initial weights, and starting from the original network weights. The results are summarised in Table I.

Note that one cycle of the BFGS algorithm involves $\mathcal{N}$ line minimisations, and for this problem takes about

0.7 sec. (running on an IBM RS6000 UNIX workstation). Starting from random initial weights the BFGS algorithm takes some 30 cycles (depending on the particular weight values) to reduce the error to the target value. When started with the original network weights only a single BFGS cycle is needed. The calculation and inversion of **A**, however, need only be done once, and the results then used to evaluate the weights for a number of different training datasets.

Table I.

Timing results for test problem

Method	Time (seconds)
BFGS starting from random weights	21.0
BFGS starting from original weights	0.7
Calculation and inversion of **A**	0.035
Evaluation of new weights	8×10^{-5}

We see from Table I that the fast retraining procedure can evaluate the weights for a new network some four orders of magnitude faster than the BFGS algorithm. (This ratio would be expected to be even larger for algorithms based on gradient descent with momentum, since such methods tend to have poor performance close to a minimum.)

An important consideration is the way in which these timing results can be expected to scale up from this simple example to large-scale applications. If we assume that the appropriate number of patterns for training a network is proportional to the number $\mathcal{N}$ of weights in the network, then both the inversion of the Hessian **A**, and a single cycle of the BFGS algorithm, will scale like $\mathcal{N}^3$. The evaluation of the weights in fast retraining (using Eqs. (15) and (16)) scales like $\mathcal{N}^2$. Thus the improvement in speed is likely to be even more marked for large networks.

4. Relation to Second-Order Training Algorithms

It is instructive to compare the algorithm presented in this paper with standard second-order optimisation methods for training feedforward networks.[4] We begin by expanding the error function $\tilde{E}$ up to second order in Δw to give

$$\tilde{E}(w + \Delta w) = \tilde{E}(w) + \sum_{\{ij\}} \frac{\partial \tilde{E}}{\partial w_{ij}} \Delta w_{ij} + \frac{1}{2} \sum_{\{ij\}} \sum_{\{kl\}} \frac{\partial^2 \tilde{E}}{\partial w_{ij} \partial w_{kl}} \cdot \Delta w_{ij} \Delta w_{kl} + \mathcal{O}(\Delta^3) \, . \tag{18}$$

At a minimum of $\tilde{E}$ we can apply Eq. (10) from which we obtain, to leading order,

$$\Delta w_{ij} = - \sum_{\{kl\}} A^{-1}_{ij,kl} \frac{\partial \tilde{E}}{\partial w_{kl}} \tag{19}$$

where we have used Eq. (14) together with the fact that $\tilde{E} = E + \mathcal{O}(\Delta)$. Equation (19) forms the basis for standard second-order training algorithms. Its relation to the formalism of Sec. 2 can be seen by using Eqs. (5), (6) and (11) to give

$$\frac{\partial \tilde{E}}{\partial w_{kl}} = \sum_p \sum_n \frac{\partial^2 E_p}{\partial x_{np} \partial w_{kl}} \Delta x_{np} + \sum_p \sum_m \frac{\partial^2 E_p}{\partial w_{kl} \partial t_{mp}} \Delta t_{mp} \, . \tag{20}$$

Substitution of Eq. (20) into Eq. (19) gives rise to Eq. (16), showing that the two approaches are formally equivalent. In practice, however, there exist several important differences. First, the evaluation and inversion of the full Hessian matrix is computationally intensive. In an iterative training scheme, therefore, it is in general not feasible to calculate the inverse Hessian matrix at each step. Instead, the Hessian is approximated, for instance, by retaining only the terms on the leading diagonal.[5,6,7] The fast retraining procedure, however, is a one-step process and it is therefore important to maintain accuracy by using the full Hessian matrix. Since the inverse is evaluated only once, the computational requirements are much less severe. Note, however, that use of the full Hessian requires $\mathcal{O}(\mathcal{N}^2)$ storage as compared with the $\mathcal{O}(\mathcal{N})$ storage required by many network training algorithms such as gradient descent and conjugate gradients. The second difference is that successive application of Eq. (19) only leads to convergence to the required solution if the initial starting point is a good approximation to the final solution. For the general problem of network training, such an estimate is not usually available. In the case of fast retraining, however, the initial weight matrix is known to be close to the desired solution. The third difference results from the fact that, in general, iteration of Eq. (19) can lead to convergence to a local maximum or saddle point as well as to a local minimum. Again, the problem does not arise in the case of fast retraining because the original error function was already at a (local) minimum. Thus, since

$$\frac{\partial^2 \tilde{E}_p}{\partial w_{ij} \partial w_{kl}} = \frac{\partial^2 E_p}{\partial w_{ij} \partial w_{kl}} + \mathcal{O}(\Delta) \, , \tag{21}$$

it follows that, for Δx and Δt sufficiently small, the new solution will also correspond to a (local) minimum. Finally, in a standard training algorithm, the first derivatives of the error function are evaluated using an expression of the form

$$\frac{\partial \tilde{E}}{\partial w_{ij}} = \sum_p \frac{\partial \tilde{E}_p}{\partial \tilde{a}_{ip}} \tilde{z}_{jp} , \tag{22}$$

which follows from Eq. (1). Here ˜ denotes quantities evaluated using $\{\tilde{x}_p, \tilde{t}_p\}$, and the derivatives $\partial \tilde{E}_p / \partial \tilde{a}_{ip}$ are evaluated using the standard error backpropagation algorithm. Each term in this sum, however, is $\mathcal{O}(1)$, while the total sum is $\mathcal{O}(\Delta)$. Numerical errors can therefore be expected to be better handled by using Eq. (20), in which all terms are $\mathcal{O}(\Delta)$, rather than Eq. (22).

5. Summary

We have described a fast procedure for training a feedforward neural network on a new dataset, given the corresponding weights and thresholds for a network of the same topology obtained for a dataset which is close to the new one. This procedure corresponds to the solution of a set of linear equations and is therefore fast. Most of the computation involves the calculation of the Hessian matrix **A** and the evaluation of its inverse. Since **A** depends only on the original network parameters, this matrix and its inverse need be computed only once.

We have also derived an extended backpropagation algorithm which allows all elements of the Hessian matrix, for a network of arbitrary feedforward topology, to be evaluated exactly.

Results from a software simulation applied to a simple problem show that the retraining procedure can generate the required network weights many orders of magnitude faster than conventional training algorithms, and this improvement in speed is likely to be even more substantial for large-scale applications.

Appendix. Calculation of Derivatives

In this appendix we discuss the calculation of the derivative terms required in Eqs. (14) and (15). We shall show that they can be evaluated using forward propagation through the network, followed by backward propagation. The resulting algorithm is related to a technique for training networks whose error functions contain derivative terms.[3] A procedure for evaluating the diagonal terms of the Hessian matrix has been given by Ricotta *et al.*[5] The algorithm presented here follows a slightly different approach, and allows all of the terms of the Hessian matrix to be evaluated without making approximations.[8]

We shall take the nonlinear activation function $f(x)$ of Eq. (1) to be the standard sigmoid:

$$f(x) \equiv \frac{1}{1 + \exp(-x)} , \tag{23}$$

which has the useful property

$$f'(x) = f(x)[1 - f(x)] , \tag{24}$$

where the prime denotes a derivative with respect to x. From Eq. (1) we have

$$\sum_m \frac{\partial^2 E_p}{\partial w_{ij} \partial t_{mp}} \Delta t_{mp} = \frac{\partial}{\partial w_{ij}} \left(\sum_m \frac{\partial E_p}{\partial t_{mp}} \Delta t_{mp} \right) = z_{jp} \pi_{ip} , \tag{25}$$

where we have defined

$$\pi_{ip} = \frac{\partial}{\partial a_{ip}} \left(\sum_m \frac{\partial E_p}{\partial t_{mp}} \Delta t_{mp} \right) . \tag{26}$$

Using the chain rule for partial derivatives, together with Eqs. (1) and (24), we obtain

$$\pi_{ip} = z_{ip}(1 - z_{ip}) \sum_k w_{ki} \pi_{kp} , \tag{27}$$

where the sum runs over all units k to which unit i sends connections. For the output units we have, from Eq. (4),

$$\pi_{mp} = -y_{mp}(1 - y_{mp}) \Delta t_{mp} . \tag{28}$$

The various terms $\{\pi_{ip}\}$ can now be calculated by backpropagation from the output units using Eqs. (27) and (28).

To evaluate the elements of the Hessian matrix,[8] we note that the units in a feedforward network can always be arranged in layers, with no intralayer connections and no feedback connections. Taking unit i to be in the same layer as unit k, or in a lower layer (i.e., one nearer the input), we can write

$$\frac{\partial^2 E_p}{\partial w_{ij}\partial w_{kl}} = z_{jp}z_{ip}(1 - z_{ip})\frac{\partial}{\partial z_{ip}}\left(\frac{\partial E_p}{\partial w_{kl}}\right). \quad (29)$$

Note that the remaining terms, in which unit i is above unit k, can be obtained from the symmetry of the second derivative without further calculation. We now write

$$\frac{\partial}{\partial z_{ip}}\left(\frac{\partial E_p}{\partial w_{kl}}\right) = \frac{\partial}{\partial z_{ip}}(\sigma_{kp}z_{lp}) = \beta_{ikp}z_{lp} + \gamma_{ilp}\sigma_{kp}\,, \quad (30)$$

where we have defined

$$\sigma_{kp} \equiv \frac{\partial E_p}{\partial a_{kp}}\,, \quad (31)$$

$$\beta_{ikp} \equiv \frac{\partial \sigma_{kp}}{\partial z_{ip}}\,, \quad (32)$$

$$\gamma_{ilp} \equiv \frac{\partial z_{lp}}{\partial z_{ip}}\,. \quad (33)$$

Using the chain rule for partial derivatives we have

$$\gamma_{ilp} = \sum_j \frac{\partial z_{jp}}{\partial z_{ip}}\frac{\partial z_{lp}}{\partial z_{jp}}\,, \quad (34)$$

from which we obtain the forward propagation equation

$$\gamma_{ilp} = z_{lp}(1 - z_{lp})\sum_j w_{lj}\gamma_{ijp}\,, \quad (35)$$

where the sum runs over all units j which send connections to unit l. The $\{\gamma_{ilp}\}$ can be calculated by forward propagation using Eq. (35), starting with the condition

$$\gamma_{ilp} = \delta_{il}\,, \quad (36)$$

(where δ_{il} is the Kronecker delta) when i and l refer to the same layer, and $\gamma_{ilp} = 0$ if l refers to a layer below the layer containing unit i.

Again using the chain rule, we can write

$$\sigma_{kp} = \sum_l \frac{\partial a_{lp}}{\partial a_{kp}}\frac{\partial E_p}{\partial a_{lp}}\,, \quad (37)$$

which gives the backpropagation equation

$$\sigma_{kp} = z_{kp}(1 - z_{kp})\sum_l w_{lk}\sigma_{lp}\,, \quad (38)$$

where the sum runs over all units l to which unit k sends connections. The $\{\sigma_{kp}\}$ can be calculated by backpropagation using Eq. (38), starting from the output units for which, using Eqs. (4), (24) and (31), we have

$$\sigma_{mp} = (y_{mp} - t_{mp})y_{mp}(1 - y_{mp})\,. \quad (39)$$

Similarly, using Eqs. (32), (33) and (38) we obtain a backpropagation equation for the $\{\beta_{ikp}\}$ in the form

$$\beta_{ikp} = (1 - 2z_{kp})\gamma_{ikp}\sum_l w_{lk}\sigma_{lp} + z_{kp}(1 - z_{kp})\sum_l w_{lk}\beta_{ilp}\,, \quad (40)$$

where again the sum runs over all units l to which unit k sends connections. Note that the first summation in Eq. (40) need not be performed explicitly when evaluating the $\{\beta_{ikp}\}$ as it has already been computed in evaluating the $\{\sigma_{kp}\}$ in Eq. (38). From Eqs. (32), (33) and (39), we have, for the output units,

$$\beta_{imp} = \gamma_{imp}\{(y_{mp} - t_{mp})(1 - 2y_{mp}) + y_{mp}(1 - y_{mp})\}\,. \quad (41)$$

Finally, derivatives of the form

$$\frac{\partial^2 E_p}{\partial x_{np}\partial w_{kl}} \quad (42)$$

are a special case of Eq. (30) and so are contained in the above formalism.

The calculation of the various derivatives therefore consists of three steps. First, inputs for pattern p are presented to the network and forward propagated using Eqs. (1), (35) and (36) to obtain the z's and γ's. Next, the π's, σ's and β's are found by backpropagation using Eqs. (27), (28) and (38)–(41). Finally, the required derivatives are evaluated using Eqs. (25), (29) and (30). The total number of distinct forward and backward propagations required (per training pattern) is roughly proportional to the number of units in the network. This algorithm can be extended to other error functions (provided they are functions of the output unit activations) by modifying Eqs. (28), (39) and (41) as appropriate.

In this appendix, we have derived a general algorithm for evaluation of the derivative terms required by fast retraining, for a network of arbitrary feedforward topology. Many of the expressions simplify in an obvious way if there is a single layer of hidden units,

or if the output units are taken to be linear. Note that, unlike the standard backpropagation algorithm, several of the quantities involved are nonlocal. Although locality may be an important consideration for hardware implementations of neural networks, it is not directly relevant if the network is to be simulated in software.

References

1. N. H. Funahashi, "On the approximate realisation of continuous mappings by neural networks," *Neural Networks* **2**, 183 (1989).
2. K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks* **2**, 359 (1989).
3. C. M. Bishop, "Curvature-driven smoothing in feedforward networks," *Proc. Int. Neural Network Conf.*, Paris, Vol. 2, p. 749 (1990), submitted to *Neural Networks*.
4. A. R. Webb, D. Lowe, and M. D. Bedworth, "A comparison of nonlinear optimisation strategies for feedforward adaptive layered networks," RSRE Memorandum 4157 (1988), R.S.R.E., St Andrews Road, Malvern, Worcs., WR14 3PS, U.K.
5. L. P. Ricotta, S. Ragazzini, and G. Martinelli, "Learning of word stress in a sub-optimal second order backpropagation neural network," *Proc. IEEE Int. Conf. Neural Networks*, Vol. 1, p. 355 (1988).
6. S. Becker and Y.le Cun, "Improving the convergence of back-propagation learning with second order methods," *Proc. Connectionist Models Summer School*, ed. Touretzky, Hinton and Sejnowski, Morgan Kaufmann, (1988) p. 29.
7. D. B. Parker, "Optimal algorithms for adaptive networks: Second order backpropagation, second order direct propagation, and second order Hebbian learning," *Proc. IEEE First Int. Conf. Neural Networks*, San Diego, CA (1987) Vol. II, p. 593.
8. C. M. Bishop, "Exact calculation of the Hessian matrix for the multilayer perceptron," submitted to *Neural Computation* (1991).