Curvature-Driven Smoothing: A Learning Algorithm for Feedforward Networks

Chris M. Bishop

Abstract— **The performance of feedforward neural networks in real applications can often be improved significantly if use is made of *a priori* information. For interpolation problems this prior knowledge frequently includes smoothness requirements on the network mapping, and can be imposed by the addition to the error function of suitable regularization terms. The new error function, however, now depends on the derivatives of the network mapping, and so the standard backpropagation algorithm cannot be applied. In this letter, we derive a computationally efficient learning algorithm, for a feedforward network of arbitrary topology, which can be used to minimize such error functions. Networks having a single hidden layer, for which the learning algorithm simplifies, are treated as a special case.**

I. Introduction

The practical application of feedforward networks requires the provision of suitable sets of data for training and testing the networks. In addition, however, there is often some *a priori* information available concerning the expected form of the network mapping. Inclusion of this prior knowledge into the network training procedure can lead to significant improvements in the network performance, particularly when the amount of training data available is limited [1].

We consider here the use of multilayer perceptron networks for interpolation applications in which the network is used to generate a continuous mapping between multidimensional spaces. Training data for such networks is specified as a set of P input vectors $\boldsymbol{x}_p \in \mathbb{R}^L$ and corresponding target vectors $\boldsymbol{t}_p \in \mathbb{R}^N$ where $p = 1, \cdots, P$. Network training algorithms are often chosen to minimize a mean square error function of the form

$$E^S = \frac{1}{2P} \sum_{p=1}^{P} \|\boldsymbol{y}_p - \boldsymbol{t}_p\|^2 \tag{1}$$

where $\boldsymbol{y}_p$ is the value of the output vector $\boldsymbol{y}$ corresponding to an input vector $\boldsymbol{x}_p$, and $\|\cdots\|$ denotes the Euclidean norm. For many such applications, it is known *a priori* that the network transfer function should satisfy certain smoothness requirements. A widespread technique for imposing smoothness in least mean square algorithms is to add to the error function a regularizing term which penalizes mappings with large curvature [2]–[4]. This leads to a total error function of the form

$$E = E^S + \lambda E^C \tag{2}$$

where the curvature smoothing term is given by

$$E^C = \frac{1}{2P} \sum_{p=1}^{P} \sum_{l=1}^{L} \sum_{n=1}^{N} \left(\frac{\partial^2 y_{np}}{\partial x_{lp}^2} \right)^2 \tag{3}$$

where y_n and x_l denote the components of $\boldsymbol{y}$ and $\boldsymbol{x}$, respectively, and the parameter λ controls the degree of smoothness of the network mapping. The optimum value for λ will be problem dependent, and can be found by seeking the minimum error with respect to a cross-validation data set, or by a variety of techniques based on the statistical properties of the training data [2]. The formalism of (2) and (3) has also been used in the training of radial basis function neural networks [5], [6].

The standard mean square error E^S depends on the interconnection weights w_{ij} through the network function $\boldsymbol{y}(\boldsymbol{x})$, and, for a given training set $\{\boldsymbol{x}_p, \boldsymbol{t}_p\}$, the derivatives of the error E^S with respect to the weights and thresholds in the network can be calculated using the backpropagation algorithm. The curvature term E^C, however, depends on derivatives of $\boldsymbol{y}(\boldsymbol{x})$ and so the standard backpropagation procedure cannot be applied. The purpose of this letter is to derive a computationally efficient technique for calculating $\partial E^C / \partial w_{ij}$. In the next section we derive expressions for the derivatives of the curvature smoothing term with respect to the network weights and thresholds, for a feedforward network of arbitrary topology. The technique is related to a recent algorithm for the exact evaluation of the Hessian matrix for a multilayer perceptron [7].

II. Learning Algorithm

Consider a network which has a feedforward architecture in which each hidden unit generates a nonlinear function of the weighted sum of its inputs:

$$z_i = f(a_i) \qquad a_i = \sum_j w_{ij} z_j \tag{4}$$

where z_j is the activation of the jth unit, w_{ij} is the connection weight from unit j to unit i, and the function f is taken to be the sigmoid

$$f(a) \equiv \frac{1}{1 + e^{-a}} \tag{5}$$

which has the useful property

$$\frac{df}{da} = f(1 - f). \tag{6}$$

There also exist thresholds for each unit. Since, however, these are equivalent to weights from an extra unit whose output is permanently set to +1, they are contained implicitly in the formalism of (4).

The network has inputs $x_l, l = 1, \cdots, L$ and outputs $y_n, n = 1, \cdots, N$. Since we are interested in interpolation problems which require continuous outputs (rather than classification problems) we shall take the output units to be linear, so that

$$y_n = a_n = \sum_i w_{ni} z_i \tag{7}$$

where the sum runs over all units i which send connections to unit n. We next write the derivatives of the curvature smoothing term in the form

$$\frac{\partial E^C}{\partial w_{ij}} = \frac{1}{P} \sum_{p=1}^{P} \sum_{l=1}^{L} \frac{\partial E_{pl}^C}{\partial w_{ij}} \tag{8}$$

where, using (3), we have defined

$$E_{pl}^C = \frac{1}{2} \sum_{n=1}^{N} \left(\frac{\partial^2 y_{np}}{\partial x_{lp}^2} \right)^2. \tag{9}$$

For clarity we shall omit the explicit pattern suffix p from now on. Using (9) we can write

$$\frac{\partial E_l^C}{\partial w_{ij}} = \sum_n \frac{\partial^2 y_n}{\partial x_l^2} \frac{\partial}{\partial w_{ij}} \left(\frac{\partial^2 y_n}{\partial x_l^2} \right). \tag{10}$$

Manuscript received February 11, 1992.

The author is with the Neural Computing Research Group, Department of Computer Science, Aston University, Birmingham, B4 7ET, U.K.

IEEE Log Number 9208458.

Since w_{ij} and x_l are independent variables, we can interchange the order of the derivatives to give

$$\frac{\partial E_l^C}{\partial w_{ij}} = \sum_n \frac{\partial^2 y_n}{\partial x_l^2} \frac{\partial^2}{\partial x_l^2}\left(\frac{\partial y_n}{\partial w_{ij}}\right). \tag{11}$$

We now make use of (4) to write

$$\frac{\partial y_n}{\partial w_{ij}} = \frac{\partial y_n}{\partial a_i} z_j. \tag{12}$$

Substituting (12) into (11), we can write the required derivatives in the form

$$\frac{\partial E_l^C}{\partial w_{ij}} = \sigma_{li} \frac{\partial^2 z_j}{\partial x_l^2} + 2\hat{\sigma}_{li} \frac{\partial z_j}{\partial x_l} + \hat{\hat{\sigma}}_{li} z_j \tag{13}$$

where we have defined the following quantities:

$$\sigma_{li} \equiv \sum_n \frac{\partial^2 y_n}{\partial x_l^2}\left(\frac{\partial y_n}{\partial a_i}\right) \tag{14}$$

$$\hat{\sigma}_{li} \equiv \sum_n \frac{\partial^2 y_n}{\partial x_l^2} \frac{\partial}{\partial x_l}\left(\frac{\partial y_n}{\partial a_i}\right) \tag{15}$$

$$\hat{\hat{\sigma}}_{li} \equiv \sum_n \frac{\partial^2 y_n}{\partial x_l^2} \frac{\partial^2}{\partial x_l^2}\left(\frac{\partial y_n}{\partial a_i}\right). \tag{16}$$

From (4) and (6) we obtain the standard backpropagation equation

$$\frac{\partial y_n}{\partial a_i} = z_i(1 - z_i) \sum_k w_{ki} \frac{\partial y_n}{\partial a_k} \tag{17}$$

where the sum runs over all units k to which unit i sends connections. Substituting (17) into (14)–(16) gives

$$\sigma_{li} = z_i(1 - z_i) \sum_k w_{ki} \sigma_{lk} \tag{18}$$

$$\hat{\sigma}_{li} = z_i(1 - z_i) \sum_k w_{ki} \hat{\sigma}_{lk} + (1 - 2z_i) \frac{\partial z_i}{\partial x_l} \sum_k w_{ki} \sigma_{lk} \tag{19}$$

$$\hat{\hat{\sigma}}_{li} = z_i(1 - z_i) \sum_k w_{ki} \hat{\hat{\sigma}}_{lk} + 2(1 - 2z_i) \frac{\partial z_i}{\partial x_l} \sum_k w_{ki} \hat{\sigma}_{lk} + \left\{(1 - 2z_i) \frac{\partial^2 z_i}{\partial x_l^2} - 2\left(\frac{\partial z_i}{\partial x_l}\right)^2\right\} \cdot \sum_k w_{ki} \sigma_{lk}. \tag{20}$$

Equations (18)–(20) allow the $\sigma, \hat{\sigma}$, and $\hat{\hat{\sigma}}$ variables to be evaluated for all of the hidden units in the network by backpropagation from the output units. The initial conditions for this backpropagation follow from (7) which gives

$$\frac{\partial y_n}{\partial a_{n'}} = \delta_{nn'} \tag{21}$$

where $\delta_{nn'}$ is the Kronecker delta symbol, from which we obtain

$$\sigma_{ln} = \frac{\partial^2 y_n}{\partial x_l^2} \qquad \hat{\sigma}_{ln} = \hat{\hat{\sigma}}_{ln} = 0. \tag{22}$$

The derivatives of the hidden unit activations are calculated using forward propagation equations which follow from (4) and (6):

$$\frac{\partial z_i}{\partial x_l} = z_i(1 - z_i) \sum_j w_{ij} \frac{\partial z_j}{\partial x_l} \tag{23}$$

$$\frac{\partial^2 z_i}{\partial x_l^2} = z_i(1 - z_i) \sum_j w_{ij} \frac{\partial^2 z_j}{\partial x_l^2} + (1 - 2z_i) \frac{\partial z_i}{\partial x_l} \sum_j w_{ij} \frac{\partial z_j}{\partial x_l} \tag{24}$$

where the sums run over all units j which send connections to unit i. The initial conditions for this forward propagation are given by

$$\frac{\partial x_l}{\partial x_{l'}} = \delta_{ll'} \qquad \frac{\partial^2 x_l}{\partial x_{l'}^2} = 0. \tag{25}$$

For the output units, which are linear, we also evaluate

$$\frac{\partial^2 y_n}{\partial x_l^2} = \sum_j w_{nj} \frac{\partial^2 z_j}{\partial x_l^2}. \tag{26}$$

We can now summarize the evaluation of the derivatives of E^C as follows:

1) Apply inputs $\{x_l\}$ and forward propagate to generate, layer by layer, the unit activations z_j, y_n using (4) and (7), and the various derivatives $\partial z_j / \partial x_l$, etc., using (23), (24), and (26).

2) Compute $\sigma, \hat{\sigma}$, and $\hat{\hat{\sigma}}$ for the output units using (22) and backpropagate to obtain the $\sigma, \hat{\sigma}$, and $\hat{\hat{\sigma}}$ for the hidden units using (18)–(20).

3) Evaluate the derivatives of E_l^C using (13).

The complete derivative of E_p^C is then found by summing over all values of the index l. For gradient descent techniques, weight corrections, in the direction of the negative of this gradient, can be applied after the presentation of each pattern. With conjugate gradients and quasi-Newton methods, the derivatives are summed over all patterns to give a total error derivative.

III. Single Hidden Layer

In Section II we obtained a genral learning algorithm for a network of arbitrary feedforward topology. For more specific architectures the algorithm can sometimes be expressed in a simplified form, and in this section we consider the case of a network having a single layer of hidden units. The expressions given below follow directly from the results obtained in Section II.

For each input pattern, we first calculate the derivatives of the hidden unit activations by forward propagation using

$$\frac{\partial z_m}{\partial x_l} = z_m(1 - z_m) w_{ml} \tag{27}$$

$$\frac{\partial^2 z_m}{\partial x_l^2} = z_m(1 - z_m)(1 - 2z_m) w_{ml}^2 \tag{28}$$

where m labels the hidden units. Similarly, for the output units, we evaluate

$$\frac{\partial^2 y_n}{\partial x_l^2} = \sum_{m=1}^{M} w_{nl} \frac{\partial^2 z_m}{\partial x_l^2} \tag{29}$$

where M is the total number of units in the hidden layer. Introducing the definition

$$\Delta_{lm} = \sum_n w_{nm} \frac{\partial^2 y_n}{\partial x_l^2} \tag{30}$$

we have the following results:

A) For a weight w_{nm} between a hidden unit m and an output unit n,

$$\frac{\partial E_l^C}{\partial w_{nm}} = \frac{\partial^2 y_n}{\partial x_l^2}\frac{\partial^2 z_m}{\partial x_l^2}. \tag{31}$$

B) For a weight $w_{ml'}$ between an input unit l' and a unit m in the hidden layer,

$$\frac{\partial E_l^C}{\partial w_{ml'}} = \left\{ 2\delta_{ll'}(1-2z_m)\frac{\partial z_m}{\partial x_l} + x_{l'} \cdot \left[(1-2z_m)\frac{\partial^2 z_m}{\partial x_l^2} - 2\left(\frac{\partial z_m}{\partial x_l}\right)^2 \right] \right\} \Delta_{lm} \tag{32}$$

where $\delta_{nn'}$ is the Kronecker symbol.

C) For a weight between an input unit l' and an output unit n,

$$\frac{\partial E_l^C}{\partial w_{nl'}} = 0 \tag{33}$$

where (33) expresses the fact that, with linear output units, weights which directly connect input and output units contribute an additive linear transformation to the network mapping function, and this necessarily has zero curvature.

IV. Discussion

The learning algorithm derived in this letter is straightforward to implement, involving the forward and backward propagation of terms in an analogous way to the standard algorithm. Furthermore, it can easily be adapted to other error measures which are differentiable functions of the network mapping and its derivatives, and to other forms of the activation function $f(\,)$.

For each pattern, the curvature error function has $L \times N$ terms, where L is the number of input units and N is the number of outputs. The quantities which are backpropagated, however, contain implicit sums over the output unit index n (as is the case in standard backpropagation) and so, compared with the standard algorithm, the number of extra computational steps required to evaluate the curvature error derivatives scales like L. The algorithm is therefore most applicable to those applications in which the number of input units is relatively small.

Simulations using this algorithm, and applications to a number of mutlidimensional interpolation problems, will be described in a subsequent paper.

References

[1] W. H. Joerding and J. L. Meador, "Encoding *a priori* information in feedforward networks," *Neural Networks,* vol. 4, pp. 847–856, 1991.

[2] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill Posed Problems.* New York: Wiley, 1977.

[3] C. M. Bishop, "Curvature-driven smoothing in backpropagation neural networks," in *Proc. Int. Neural Network Conf.,* Paris, France, vol. 2, 1990, pp. 749–752.

[4] C. M. Bishop, "Curvature-driven smoothing in backpropagation neural networks," in *Theory and Applications of Neural Networks.* Springer-Verlag, 1992, pp. 139–148.

[5] C. M. Bishop, "Improving the generalization properties of radial basis function neural networks," *Neural Computation,* vol. 3, pp. 579–588, 1991.

[6] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE,* vol. 78, p. 1481, 1990.

[7] C. M. Bishop, "Exact calculation of the Hessian matrix for the multilayer perceptron," *Neural Computation,* vol. 4, pp. 494–501, 1992.

Single Layer Neural Networks for Linear System Identification Using Gradient Descent Technique

Satyendra Bhama and Harpreet Singh

***Abstract*—Recently, some researchers have focused on the applications of neural networks for the system identification problems. In this letter we describe how to use the gradient descent (GD) technique with single layer neural networks (SLNN's) to identify the parameters of a linear dynamical system whose states and derivatives of state are given. It is shown that the use of the GD technique for the purpose of system identification of a linear time invariant dynamical system is simpler and less expensive in implementation because it involves less hardware than the technique using the Hopfield network as discussed by Chu. The circuit is considered to be faster and is recommended for on-line computation because of the parallel nature of its architecture and the possibility of the use of analog circuit components. A mathematical formulation of the technique is presented and the simulation results of the network are included.**

I. Introduction

Some researchers have recently focused on the applications of neural networks for nonlinear system modeling and identification [1], [2]. Shoureshi *et al.* [3] have proposed Hopfield network for identifying a linear time invariant dynamical system, by measuring the inputs, states, and derivative of states. However, for some problems, a globally optimal solution is not guaranteed by the Hopfield network; the network computes locally optimum solutions [4], [5]. In such cases it might not be advisable to use this network, as it gives quite inaccurate results and is computationally complex [5], [6].

This letter proposes a technique that uses a gradient descent learning algorithm also known as instant back-propagation to train a single layer neural network (SLNN) for identifying the parameters of a linear system. An appropriately formulated least mean square error (LMSE) is used as a performance measure for the network proposed in [7], [8]. In recent papers, the present authors have suggested the use of such a network for dynamic image modeling [9], [10]. It is shown by mathematical arguments and simulation results that the neural network trained by this technique produces output results which are similar to that obtained by [3]. The proposed network is simpler in structure and, therefore, less expensive in implementation. The circuit is suitable for on-line computation because of the parallel nature of its architecture and possibility of the use of analog circuit elements. The work in this letter is organized as follows: Section II deals with the basics of system dynamics, error function formulation, and a brief mathematical analysis of the proposed technique. Section III addresses some design issues of the structure of the network. In Section IV, the algorithm for computer simulation of SLNN is given. Our simulation results, some recommendations for fine tuning the network, and a brief performance comparison with the traditional system identification approaches are summarized in Section V. Finally, in Section VI, we conclude by giving some comments on the application of the technique.

Manuscript received March 25, 1992; revised October 8, 1992.

S. Bhama was with the Computer Research Laboratory, Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202. He is now with the GM Research and Development Center (GMR&D), Warren, MI 48090.

H. Singh is with the Computer Research Laboratory, Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202.

IEEE Log Number 9209804.