Dynamic Programming Approach For Multilayer Neural Network Optimization

P.Saratchandran
School of Electrical and Electronics Engineering
Nanyang Technological Institute
Singapore 2263

Abstract: A new algorithm for weight adjustments in a multilayer neural network is derived using the principles of dynamic programming. The algorithm computes the optimal values for weights on a layer after layer basis starting from the output layer of the network. The algorithm can be used for continuous and discontinuous neuron activation functions.

1 Introduction

Dynamic programming [1] is a well-known approach used in operations research and control engineering for optimization of multistage decision processes. The key concept behind dynamic programming is the principle of optimality [1] according to which the cost to be minimized at every stage be split into two components viz i) the cost due to the current stage and ii) the minimum cost for the remaining stages obtained by using optimum decision sequences for the remainder of the process. A multilayer feed forward neural network can be thought of as a multistage decision process. Optimal selection of weights for each layer is akin to optimal choice of decisions at each stage since weights in a layer can affect only the outputs of subsequent layers as with decisions in a multistage decision process.

This paper presents a learning algorithm based on dynamic programming for multilayer networks. According to this algorithm at every layer the cost obtained using optimal values for weights for the remainder of the network is minimized using the weights in the current layer. Optimum weights are then computed recursively starting from the output layer. In section 2.0 mathematical formulation of the weight minimization problem is described within the

0-7803-0164-1/91/0000-0397$01.00©1991 IEEE

dynamic programming framework. In section 3.0 equations governing weight adjustments in each layer are derived when the activation functions for neurons are continuous eg. sigmoid functions.

2 Mathematical Formulation

The cost function to be minimized by adjusting the synaptic weights in the network is defined as

$$J(y(n)) = \sum_p \sum_i (t_{pi} - y_{pi}(n))^2 \tag{1}$$

where p is an index over patterns, i is an index over output neurons, $y_{pi}(n)$ is the actual output of neurons in the n^{th} layer and t_{pi} is the desired output at the final layer.

Define $I(y(k))$ as the minimum cost that can be achieved using optimal values for weights in all layers from k to n. Then for the n^{th} layer,

$$I(y(n)) = J(y(n)).$$

For the $(n-1)^{th}$ layer,

$$I(y(n-1)) = \min_{w(n-1)} I(y(n)) \tag{2}$$

where $w(n-1)$ represents all weights in the $(n-1)^{th}$ layer.

The outputs $y(n)$ of n^{th} layer is related to the outputs $y(n-1)$ as

$$y_{pi}(n) = f(net_{pi}(n)) \tag{3}$$

$$net_{pi}(n) = \sum_j w_{ij}(n-1) y_{pj}(n-1) \tag{4}$$

where net_{pi} is the total input at neuron i for pattern p and f is the activation function for the neurons.

Substituting for $y(n)$ in (2),

$$I(y(n-1)) = \min_{w(n-1)} I(f(y(n-1), w(n-1))).$$

Thus the minimum cost $I(y(k))$ for any layer $1 \leq k \leq n-1$ can be written as

$$I(y(k)) = \min_{w(k)} I(f(y(k), w(k))). \tag{5}$$

Equation (5) describes an iterative relationship for determining $I(y(k))$ for all $1 \leq k \leq n-1$ from the knowledge of $I(y(k+1))$. Solving it for successive values of k starting from $k = n-1$ will yield optimum values for weights recursively.

Minimization of (5) can in general be done only using numerical techniques. When the neuron activation function ie.f in equation (3) is not continuous minimization can be done by quantizing $y's$ and $w's$ and using such techniques as successive approximation [1,2].

When f is continuous equations for weight adjustments can be obtainined analytically using gradient descent search. These equations are derived in section 3.0.

3 Derivation of Equations When f is Continuous

Consider the network shown in figure 1. For the $(n-1)^{th}$ layer the minimum cost $I(y(n-1))$ is

$$I(y(n-1)) = \min_{w(n-1)} \sum_p \sum_i (t_{pi} - f(net_{pi}(n)))^2. \tag{6}$$

Using the simplest version of gradient search formula, the equation for updating weights w_{ij} in the $(n-1)^{th}$ layer is

$$w_{ij}^{new}(n-1) = w_{ij}^{old}(n-1) + \Delta w_{ij}(n-1) \tag{7}$$

where

$$\Delta w_{ij}(n-1) = -C \frac{\partial I(y(n))}{\partial w_{ij}(n-1)}, \quad C \textit{ being a constant.} \tag{8}$$

Since the summation in equation (6) is over all patterns,

$$\Delta w_{ij}(n-1) = \sum_p \Delta_p w_{ij}(n-1),$$

where

$$\Delta_p w_{ij}(n-1) = -C \frac{\partial (t_{pi} - f(net_{pi}(n)))^2}{\partial w_{ij}(n-1)}.$$

Upon performing the differentiation,

$$\Delta_p w_{ij}(n-1) = \epsilon\, \delta_{pi}(n)\, y_{pj}(n-1) \tag{9}$$

where

$$\begin{aligned} \epsilon &= 2C, \\ \delta_{pi}(n) &= [t_{pi} - f(net_{pi}(n))]\, f'(net_{pi}(n)),\ and \end{aligned}$$

$$f'(net_{pi}(n)) = \frac{\partial y_{pi}(n)}{\partial net_{pi}(n)}.$$

The minimized cost is

$$I(y(n-1)) = \sum_p \sum_i (t_{pi} - f(net^*_{pi}(n)))^2 \tag{10}$$

where

$$net^*_{pi}(n) = \sum_j w^*_{ij}(n-1) y_{pj}(n-1),$$

and $w^*_{ij}(n-1)$ is the optimum value of $w_{ij}(n-1)$ resulted from gradient search.

For the $(n-2)^{nd}$ layer ,

$$\Delta_p w_{jk}(n-2) = 2C \sum_i [t_{pi} - f(net^*_{pi}(n))]\, f'(net^*_{pi}(n))\, \frac{\partial net^*_{pi}(n)}{\partial w_{jk}(n-2)} \tag{11}$$

where

$$\frac{\partial net^*_{pi}(n)}{\partial w_{jk}(n-2)} = \sum_j w^*_{ij}(n-1) f'(net_{pj}(n-1))\, y_{pk}(n-2). \tag{12}$$

Substituting for $\frac{\partial net p^*_{pi}(n)}{\partial w_{jk}(n-2)}$ in equation (11)

$$\Delta_p w_{jk}(n-2) = \epsilon \sum_i \delta^*_p(n) \left(\sum_j \delta_p(n-1)\right) y_{pk}(n-2). \tag{13}$$

where

$$\begin{aligned} \delta^*_p(n) &= [t_{pi} - f(net^*_{pi}(n))] f'(net^*_{pi}(n)), \\ \delta_p(n-1) &= w^*_{ij}(n-1) f'(net_{pj}(n-1)). \end{aligned}$$

Generalizing equation (13), $w_{ij}(k)$ for any layer $1 \leq k \leq (n-1)$ will be

$$\Delta_p w_{ij}(k) = \epsilon \sum_n \delta_p^*(n)(\sum_{n-1} \delta_p^*(n-1)(\sum_{n-2} \delta_p^*(n-2) \cdots \cdots (\sum_{k+1} \delta_p(k+1)))) y_{pj}(k). \quad (14)$$

In equation (14) $\sum_n, \sum_{n-1}$ etc means over all neurons in layer n, n-1 etc. $\delta_p^*(k)$ for any layer $1 \leq k \leq n-1$ is given by

$$\delta_p^*(k) = w_{ij}^*(k) f'(net_{pj}^*(k)).$$

4 Conclusion

A learning algorithm based on dynamic programming has been derived for multilayer neural networks. The advantages of this algorithm over other well-known algorithms [4,5] are that it can be used when constraints need to be imposed on weights or when the activation functions for neurons are not continuous.

References

[1] Bellman,R.E., and Dreyfus,S.E. Applied Dynamic Programming, Princeton University Press, New Jersey, 1962.

[2] Cooper,L., and Cooper,M. Introduction to Dynamic Programming, Pergammon Press, 1981.

[3] Larson,R.E., and Casti,J.L. Principles of Dynamic Programming, Marcel Decker, NewYork, 1978.

[4] Rumelhart,D.E., et al. Learning Representations by Back Propagating Errors, Nature, 1986, 323, pp 533-536

[5] Widrow,B., and Winter, R. Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition, Computer, 1988, vol 21, No. 3, pp 25-39

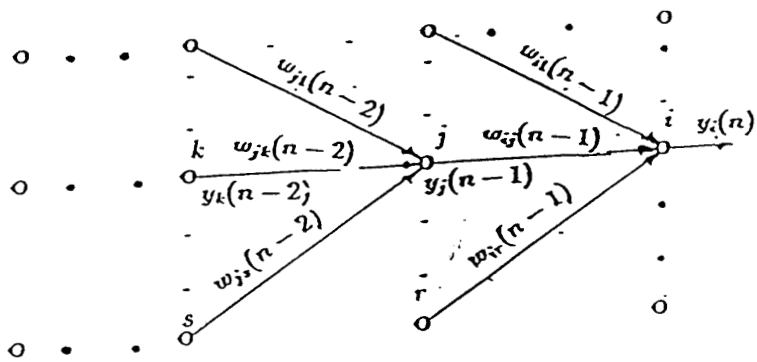

Figure 1. A Multilayer Neural Network. i, j, k are arbitary neurons in $n, n-1, n-2$ layer respectively.