Appl. Num. Anal. Comp. Math. **1**, No. 2, 535 – 539 (2004) / **DOI** 10.1002/anac.200410016

A Local Supervised Learning Algorithm For Multi-Layer Perceptrons

D.S. Vlachos*

Hellenic Center for Marine Research, PO BOX 712, 19013, Anavyssos, Greece

Received 5 October 2004, revised 30 November 2004, accepted 2 December 2004
Published online 20 December 2004

The back propagation of error in multi-layer perceptrons when used for supervised training is a non-local algorithm in space, that is it needs the knowledge of the network topology. On the other hand, learning rules in biological systems with many hidden units, seem to be local in both space and time. In this work, a local learning algorithm is proposed which makes no distinction between input, hidden and output layers. Simulation results are presented and compared with other well known training algorithms.

1 Introduction

One of the major problems with feed-forward network learning remains the accuracy and speed of the learning algorithms. Since the learning problem is a complex and highly nonlinear one [1],[2], iterative learning procedures must be used to solve the optimization problem [3], [4]. A continuous desire to improve the behavior of the learning algorithm has lead to many excellent optimization algorithms which are special tailored for feed-forward network learning. On the other hand, although approximation capabilities of multi-layer perceptrons have been established, the optimal network structure which reaches a predefined accuracy for the approximation, remains still an open question. Moreover, certain characteristics of the network behavior (like stability, local minima, learning speed etc) are very close related to network topology.

An interesting approach to overcome the aforementioned problems, is that of dynamic network topologies and functional networks. Dynamic networks change the structure of the network in order to meet certain constrains during the learning stage. The ill-conditioning of the network learning for example, can be handled by changing the structure of the network [5]. Functional networks on the other hand, face the structure problem using building blocks and a modular architecture, splitting thus the complexity of the problem under consideration. Both approaches require a learning algorithm which can be easily adapted to varying network topologies.

Various algorithms for supervised learning have been proposed, in which apart from the fact that they require a set of teaching input-output pairs, there is a second reason that makes them biological inplausible: they depend on global computations. This non-local in space character of the learning algorithm produces many difficulties as the composition of the activation functions in multi-layer networks and the poor capability of modular construction of different network topologies.

The difference between local and global computations in the context of neural networks has to do with both space and time. Local in space is meant to say that changes of a unit's weight vector should depend solely on activation information from the unit itself and from connected units. The update complexity for a unit's weight vector at a given time should be only proportional to the dimensionality of the weight vector. Local in time is meant to say that weight changes should take place continually, and that changes should depend only on information about units and weights from a fixed recent time interval. The expression 'local in time' corresponds to the notion of 'on-line' learning. As far as we can judge today, biological systems use completely local computations to accomplish complex spatio-temporal credit assignments tasks.

In this work, a local in space supervised learning algorithm is proposed for use in multi-layer perceptrons. The algorithm is based on back propagating not only the error but a desired pattern for the output of the previous layer as well. In this way, the network can be sliced in autonomous layers each of which can be trained independently.

* Corresponding author: e-mail: dvlachos@ncmr.gr, Phone: +30 22910 76410, Fax: +30 22910 76323

2 The learning algorithm

Figure 1 shows the neuron model used. Each neuron is activated by a threshold functions, the input of which is the weighted sum of all neuron's inputs. The neuron topology that is considered here is a layered one, that is, the neurons are organized in layers and the inputs of each neuron are the outputs of the neurons of the preceding layer. The output of layer m is represented by the vector $x_m, m = 0(1)M$, x_0 is the input and x_M the output of the network. The weights of the network can be organized in two dimensional arrays $w_m, m = 1(1)M$ which contains the weights connecting layer-$(m-1)$ with layer-(m). All neurons are considered to have the same threshold function g, which is a smooth function of one variable. Under these assumptions, a recurrent relation can be written:

$$x_m^j = g(w_{m,i}^j \cdot x_{m-1}^i) \tag{1}$$

where m runs from 1 to M.

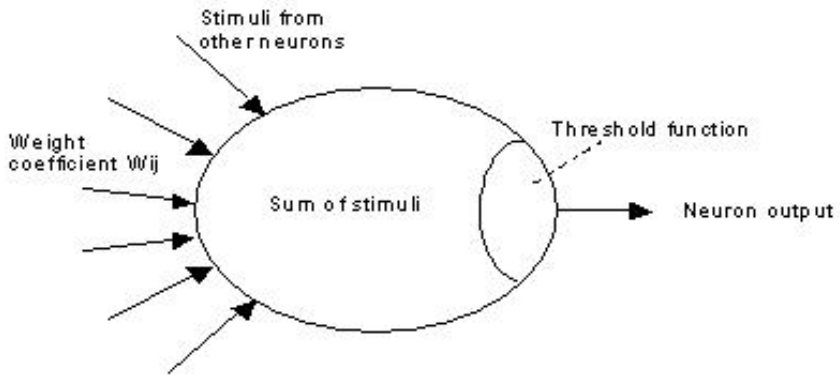

Fig. 1 Neuron model.

Supervised learning means that we have a set of couples $(x_0, \hat{x}_M)$ and and an algorithm to change the weights of the network, such that at the end the network will produce output $\hat{x}_M$ when it is feeded with input x_0. When the input of the network is x_0 and the output is x_M we can define the local error

$$e_M^i = (x_M^i - \hat{x}_M^i)^2 \tag{2}$$

and the global error

$$e_M = \sum_i e^i \tag{3}$$

An efficient way to obtain supervised learning of the network is to change the weights of the network in this direction that reduces the global error. Thus, in order to change the weights of the layer-(m) we need the information about the network topology from layer-(m) and above. This learning algorithm is local only for the last (output) layer.

Consider now the following case: Suppose that the network is feeded with input x_0 and produces the output x_M. The global error e_M represents the distance between x_M and the desired output $\hat{x}_M$. This error can be reduced by two ways; by changing the weights of the output layer in a certain direction (a local procedure) and by forcing the network to produce a different vector $\hat{x}_{M-1}$ which can be considered as the desired output of the previous layer. The next step is now obvious. We can use in the same way the vector $\hat{x}_{M-1}$ to change the weights of the layer-$(M-1)$ which is again a local procedure, and to produce the vector $\hat{x}_{M-2}$ which can be considered as the desired output of the layer-$(M-2)$. In this way we can change all the weights of the network by a local algorithm.

Let x_m be the output of the layer-(m). We can define the desired output $\hat{x}_m$ of the same layer as:

$$\hat{x}_m^i = x_m^i + \delta_{x,m}^i \tag{4}$$

where $\delta^i_{x,m}$ is a small change in x^i_m that would cause a decrease of the global error of the layer-$(m+1)$. For the weights now of the same layer, we can change their values by an amount $\delta^i_{w,m,j}$ which will cause a decrease of the global error of layer-(m). In order to determine these δ-values, we take the Taylor expansion of the global error of layer-(m):

$$e_m(x^i_{m-1}+\delta^i_{x,m-1}, w^k_{m,l}+\delta^k_{w,m,l}) = e_m(x^i_{m-1}, w^k_{m,l}) + W^l_{m,k}\cdot\delta^k_{w,m,l} + X_{m,i}\cdot\delta^i_{x,m-1} + P^{ql}_{m,pk}\cdot\delta^p_{w,m,q}\cdot\delta^k_{w,m,l} + Q^l_{m,ik}\cdot\delta^k_{w,m,l}\cdot\delta^i_{x,m-1} + R_{m,ij}\cdot\delta^i_{x,m-1}\cdot\delta^j_{x,m-1} + \ldots \quad (5)$$

where

$$W^l_{m,k} = \frac{\vartheta e_m}{\vartheta w^k_{m,l}} \quad (6)$$

$$X_{m,i} = \frac{\vartheta e_m}{\vartheta x^i_{m-1}} \quad (7)$$

$$P^{ql}_{m,pk} = \frac{1}{2}\frac{\vartheta^2 e_m}{\vartheta w^k_{m,l}\vartheta w^p_{m,q}} \quad (8)$$

$$Q^l_{m,ik} = \frac{\vartheta^2 e_m}{\vartheta w^k_{m,l}\vartheta x^i_{m-1}} \quad (9)$$

$$R_{m,ij} = \frac{1}{2}\frac{\vartheta^2 e_m}{\vartheta x^i_{m-1}\vartheta x^j_{m-1}} \quad (10)$$

Under these considerations, one can split the error of a certain layer in two sources: the error from previous layers and the error from the current layer. Let us now take only the first terms of equation (5) (the linear case). Then, by taking

$$\delta^i_{x,m-1} = -X_{m,i}$$
$$\delta^k_{w,m,l} = -W^l_{m,k}$$

the global error is decreased. By writing $G^i_m = 2\cdot(x^i_m - \hat{x}^i_m)\cdot g'(x^j_{m-1}\cdot w^i_{m,j})$ we have:

$$\delta^i_{x,m-1} = \sum_j \delta^i_{x,m,j} \quad (11)$$

$$\delta^k_{w,m,l} = -G^k_m \cdot x^l_{m-1} \quad (12)$$

where

$$\delta^i_{x,m,j} = -G^i_m \cdot w^i_{m,j} \quad (13)$$

An other approach is to split the error in two halves, one half coming from the previous layer and the other half caused by the current layer (the 1/2 rule). Then, the desired output of the m-layer is:

$$\hat{x}^i_m = x^i_m + \frac{1}{2}\delta^i_{x,m} \quad (14)$$

which can be used for the calculation of $\delta^k_{w,m,l}$. The other half of the error, is used for the calculation of δ^i_{m-1} of the previous layer:

$$\delta^i_{x,m-1} = \frac{1}{2}\sum_j \delta^i_{x,m,j} \quad (15)$$

Other possible approaches can be used for the splitting of the error, which can take into account higher terms of the error expansion given at equation (5). In all of them, the tuning of the weights in a certain layer does not depend on the general network topology but only on the current layer.

The above equations lead us to the construction of a modified neuron model as it is shown in figure 2. In this figure, the j-th neuron of layer-(m) is depicted. Both weights and desired layer outputs can be calculated continuously, where the only inputs of the network are the input pattern x_0 and the desired output $\hat{x}_M$, when the network is in the training mode. When the network has been trained, the output of the network x_M can be wired to the desired output $\hat{x}_M$.

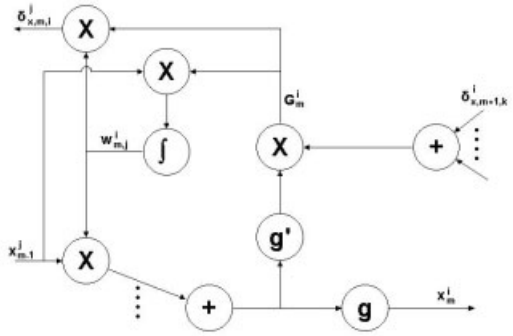

Fig. 2 Modified neuron model.

3 Simulation results

In order to test the efficiency of the new learning algorithm, we have performed two simulation test: the first concerns the problem of finding the maximum number between three ones (all three numbers are between zero and one) and the second is the calculation of the XOR of two binary inputs. Figure 3 shows the output error function during the learning phase. Three networks have been tested: the first one contains only one hidden layer with six nodes, and the back propagation learning algorithm is applied. The other two networks have two and three hidden layers respectively, and the local learning algorithm is applied (the linear case). The interesting part from this comparison is that, although the complexity of the network is increased by introducing more hidden layers, the local learning algorithm seems to behave in a stable and efficient way. This result is very promising for use of this algorithm in dynamic neural networks.

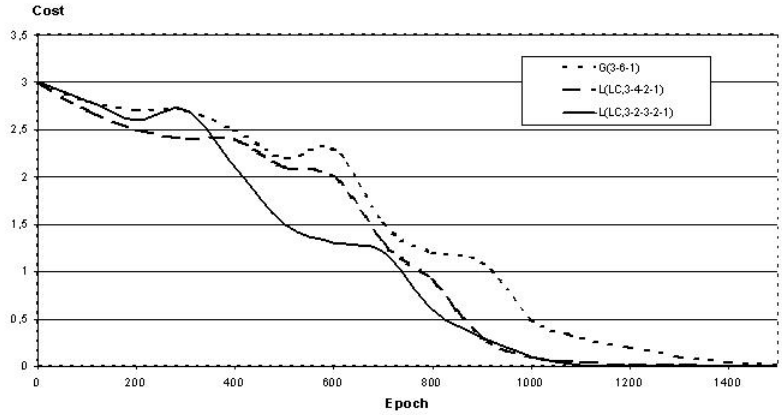

Fig. 3 Network error during learning phase for the calculation of the maximum between three numbers. G(3,6,1) is a network with one hidden layer with 6 nodes which uses global computations (error back propagation) for the calculation of weights. L(LC,3,4,2,1) is a network with 2 hidden layers which uses local computations (linear case) for the calculation of weights. L(LC,3,2,3,,2,1) is similar to the previous but with 3 hidden layers.

The second test is performed again in three network topologies. The first one has one hidden layer with four nodes and the back propagation learning algorithm is applied. The second and third networks are the same having two hidden layers but in the first one the 1/2 rule is applied while in the second, the linear case is considered for the calculation of the weights. Again the local learning algorithm behaves in an efficient way, even when the 1/2 rule is applied.

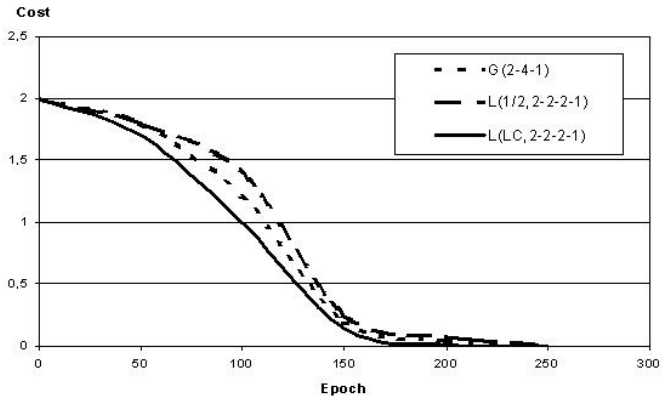

Fig. 4 Network error during learning phase for the calculation of the XOR between two binary inputs. G(2,4,1) is a network with one hidden layer with 4 nodes which uses global computations (error back propagation) for the calculation of weights. L(1/2,2,2,2,1) is a network with 2 hidden layers which uses local computations (the 1/2 rule) for the calculation of weights. L(LC,2,2,2,1) is similar to the previous but the linear case is considered for the calculation of the weights.

4 Conclusions

By considering that the output error of a specific layer (the distance of the actual output from the desired one) is a sum of the error of the output of the previous layer and the error introduced in the current layer, the weights of the current layer are affected only by the part of error caused by the current layer. Each layer notifies its previous layer about the reminding error, which will be used again for the tuning of the weights of the previous layer. The local character of this learning algorithm assures that the update complexity for a unit's weight vector at a given time should be only proportional to the dimensionality of the weight vector.

References

[1] E. DL. Sontag and H.J. Sussmann, Complex Systems **3**(1), 91–106 (1989).
[2] Bhaskar DasGupta, Hava T. Siegelmann and E.D. Sontag, IEEE Transactions on Neural Networks, **6**(6), 1490–1504 (1995).
[3] R. Battiti, Neural Computation **4**, 141–166 (1992).
[4] P. van der Smagt, Visual Robot Arm Guidance using Neural Networks, PhD thesis, Dept. of Computer Systems, University of Amsterdam, March 1995.
[5] P. van der Smagt and G. Hirzinger, in: Neural Networks: Tricks of the Trade, G. Orr and K.R. Muller (eds), Lecture Notes in Computer Science **1524**, Springer-Verlag, 193–206 (1998).