EXTENDED NETWORK FOR BACKPROPAGATION ALGORITHM

MIRCEA PETRINI [*]

ABSTACT: *This paper presents the backpropagation algorithm based on an extended network approach in which the algorithm reduces to a graph labeling problem. This method is not only more general than the usual analytical derivations, which handle only the case of special network topologies, but also much easier to follow. It also shows how the algorithm can be efficiently implemented in computing systems in which only local information can be transported through the network.*

KEYWORDS: *Artificial Neural Network (ANN); backpropagation; extended network; feed-forward computation; training pattern.*

JEL CLASSIFICATION: *C*

1. BACKPROPAGATION ALGORITHM

The backpropagation algorithm used in artificial neural network (ANN) looks for the minimum of the error function in weight space using the method of gradient descent. The combination of weights which minimizes the error function is considered to be a solution of the learning problem. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function.

Obviously we have to use a kind of activation function other than the step function used in perceptrons, because the composite function produced by interconnected perceptrons is discontinuous, and therefore the error function too. One of the more popular activation functions for backpropagation networks is the *sigmoid*.

[*] *Lecturer, University of Petroşani, Romania, petrini_mircea@yahoo.com*

2. EXTENDED NETWORK

We will consider a network with n input sites, k hidden, and m output units. The weight between input site i and hidden unit j will be called $w_{ij}^{(1)}$. The weight between hidden unit i and output unit j will be called $w_{ij}^{(2)}$. The bias of each unit is implemented as the weight of an additional edge. Input vectors are thus extended with a 1 component, and the same is done with the output vector from the hidden layer. Figure 1 shows how this is done. The weight between the constant 1 and the hidden unit j is called $w_{n+1,j}^{(1)}$ and the weight between the constant 1 and the output unit j is denoted by $w_{k+1,j}^{(2)}$.

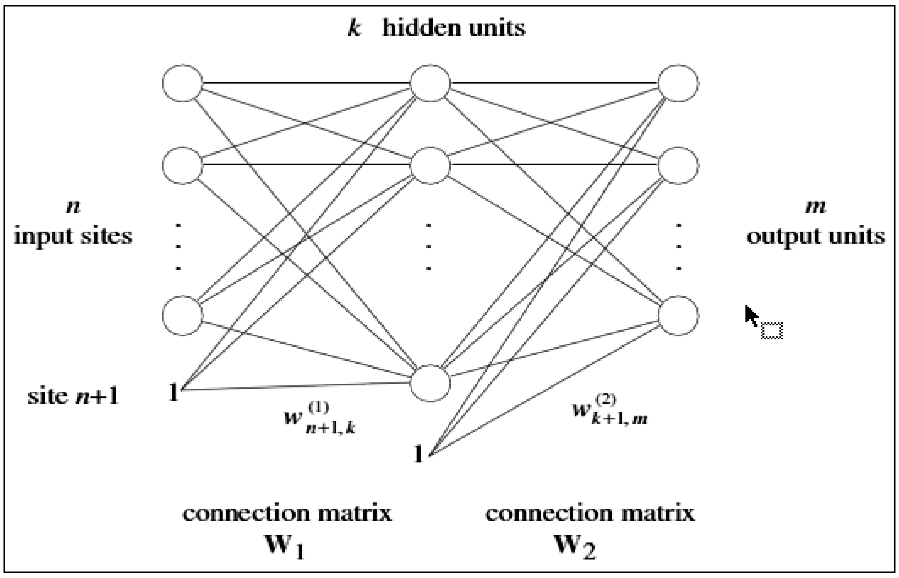


Figure 1. Notation for the three-layered network

There are $(n+1)\times k$ weights between input sites and hidden units and $(k+1)\times m$ between hidden and output units. Let $\overline{W}_1$ denote the $(n+1)\times k$ matrix with component $w_{ij}^{(1)}$ at the i row and the j column. Similarly let $\overline{W}_2$ denote the $(k+1)\times m$ matrix with components $w_{ij}^{(2)}$. We use an overlined notation to emphasize that the last row of both matrices corresponds to the biases of the computing units. The matrix of weights without this last row will be needed in the backpropagation step. The n-dimensional input vector $o=(o_1,\dots,o_n)$ is extended, transforming it to $\hat{o}=(o_1,\dots,o_n,1)$ the excitation net_j of the j hidden unit is given by $net_j = \sum_{i=1}^{n+1} w_{ij}^{(1)}\hat{o}_i$.

The activation function is a sigmoid and the output $o_j^{(1)}$ of this unit is thus $o_j^{(1)} = s\left(\sum_{i=1}^{n+1} w_{ij}^{(1)} \hat{o}_i\right)$.

The excitation of all units in the hidden layer can be computed with the vector-matrix multiplication $\hat{o}\overline{W}_1$. The vector $o^{(1)}$ whose components are the outputs of the hidden units is given by

$$\mathbf{o}^{(1)} = s(\hat{\mathbf{o}}\overline{\mathbf{W}}_1),$$

using the convention of applying the sigmoid to each component of the argument vector. The excitation of the units in the output layer is computed using the extended vector $\hat{o}^{(1)} = (o_1{}^{(1)}, \ldots, o_k{}^{(1)}, 1)$. The output of the network is the *m*-dimensional vector $o^{(2)}$, where $\mathbf{o}^{(2)} = s(\hat{\mathbf{o}}^{(1)}\overline{\mathbf{W}}_2)$.

These formulas can be generalized for any number of layers and allow direct computation of the flow of information in the network with simple matrix operations.

3. THE ALGORITHM

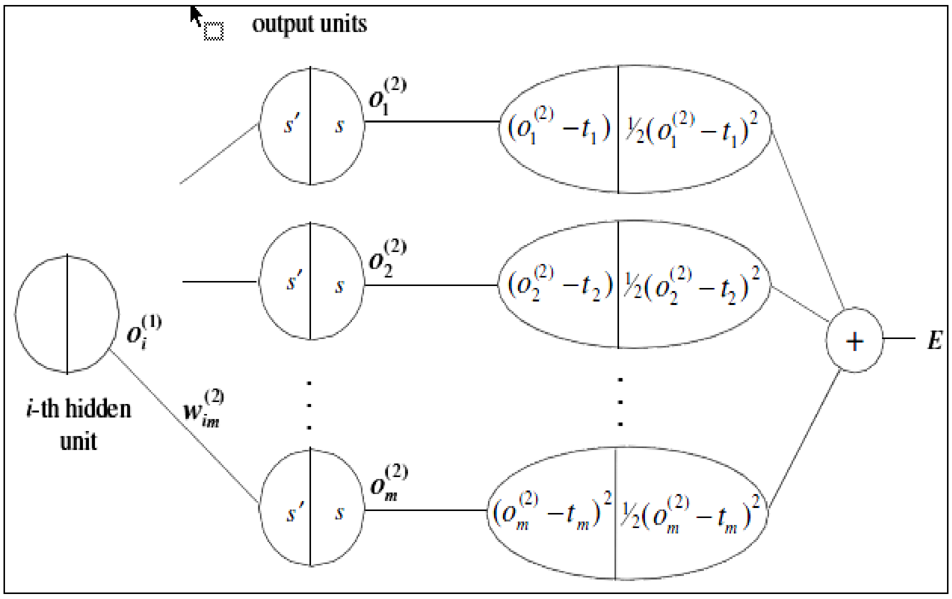
Figure 2 shows the extended network for computation of the error function. In order to simplify the discussion we deal with a single input-output pair (o, t) and generalize later to *p* training examples. The network has been extended with an additional layer of units. The right sides compute the quadratic deviation or the *i* component of the output vector and the left sides store $(o_i^{(2)} - t_i)$. Each output unit *i* in the original network computes the sigmoid s and produces the output $o_i^{(2)}$. Addition of the quadratic deviations gives the error E. The error function for p input-output examples can be computed by creating p networks like the one shown, one for each training pair, and adding the outputs of all of them to produce the total error of the training set.

After choosing the weights of the network randomly, the backpropagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps: i) Feed-forward computation; ii) Backpropagation to the output layer; iii) Backpropagation to the hidden layer; iv) Weight updates.

The algorithm is stopped when the value of the error function has become sufficiently small.

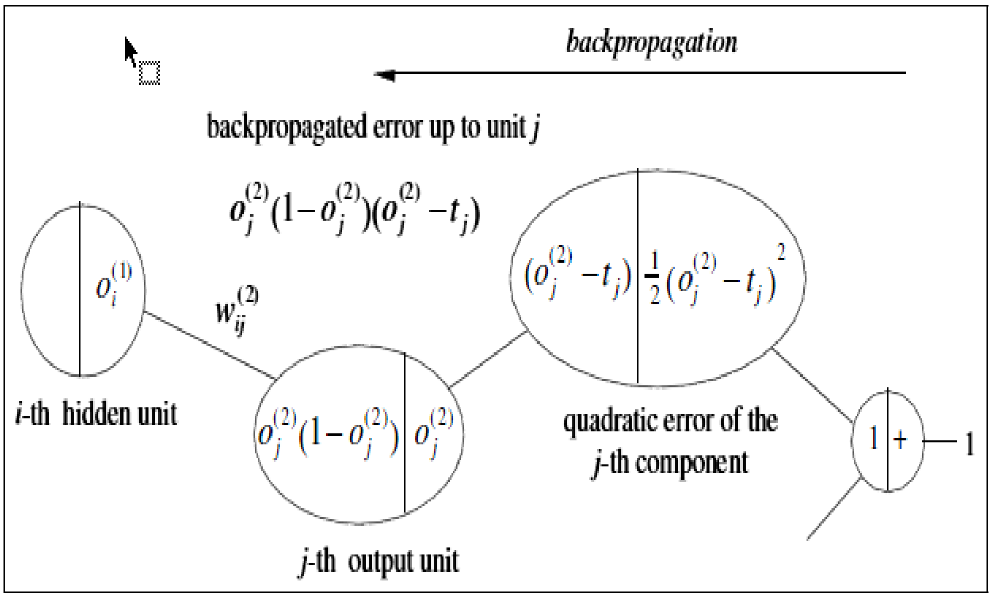
i) First step: feed-forward computation. The vector o is presented to the network. The vectors $o^{(1)}$ and $o^{(2)}$ are computed and stored. The evaluated derivatives of the activation functions are also stored at each unit.

ii) Second step: backpropagation to the output layer. We are looking for the first set of partial derivatives $\frac{\partial E}{\partial w_{ij}^{(2)}}$. The backpropagation path from the output of the network up to the output unit j is shown in the B-diagram of figure 3.



output units
$o_1^{(2)}$
s' s
$(o_1^{(2)}-t_1)$ $½(o_1^{(2)}-t_1)^2$
$o_2^{(2)}$
$(o_2^{(2)}-t_2)$ $½(o_2^{(2)}-t_2)^2$
$o_i^{(1)}$
$+$
E
i-th hidden unit
$w_{im}^{(2)}$
$o_m^{(2)}$
$(o_m^{(2)}-t_m)^2$ $½(o_m^{(2)}-t_m)^2$

Figure 2. Extended multilayer network for the computation of E



backpropagation
backpropagated error up to unit j
$o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j)$
$(o_j^{(2)}-t_j)$ $\frac{1}{2}(o_j^{(2)}-t_j)^2$
$o_i^{(1)}$
$w_{ij}^{(2)}$
i-th hidden unit
$o_j^{(2)}(1-o_j^{(2)})$ $o_j^{(2)}$
quadratic error of the j-th component
1 + 1
j-th output unit

Figure 3. Backpropagation path up to output unit *j*

From this path we can collect by simple inspection all the multiplicative terms which define the backpropagated error $\delta_j^{(2)}$. Therefore $\delta_j^{(2)} = o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j)$, and the partial derivative we are looking for is

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = [o_j^{(2)}(1 - o_j^{(2)})(o_j^{(2)} - t_j)]o_i^{(1)} = \delta_j^{(2)} o_i^{(1)}.$$

Remember that for this last step we consider the weight $w_{ij}^{(2)}$ to be a variable and its input $o_i^{(1)}$ a constant.

iii) Third step: backpropagation to the hidden layer. Now we want to compute the partial derivatives $\frac{\partial E}{\partial w_{ij}^{(1)}}$. Each unit *j* in the hidden layer is connected to each unit *q* in the output layer with an edge of weight $w_{ij}^{(2)}$, for q = 1,...,m. The backpropagated error up to unit *j* in the hidden layer must be computed taking into account all possible backward paths, as shown in figure 4.

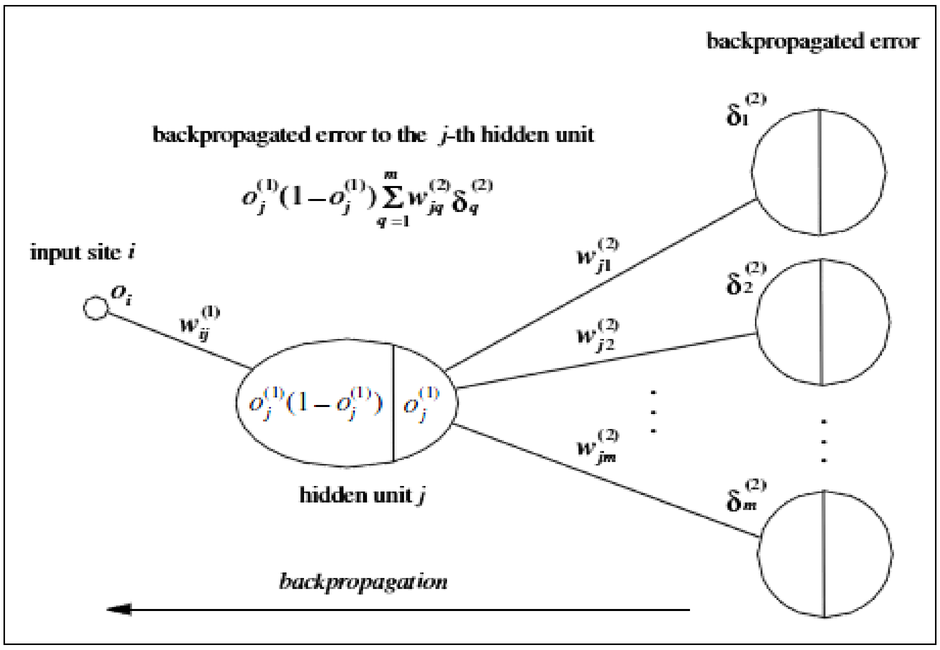


Figure 4. All paths up to input site *i*

The backpropagated error is then

$$\delta_j^{(1)} = o_j^{(1)}(1 - o_j^{(1)}) \sum_{q=1}^{m} w_{jq}^{(2)} \delta_q^{(2)}.$$

Therefore the partial derivative we are looking for is

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_i.$$

The backpropagated error can be computed in the same way for any number of hidden layers and the expression for the partial derivatives of *E* keeps the same analytic form.

iv) Fourth step: weight updates. After computing all partial derivatives the network weights are updated in the negative gradient direction. A learning constant defines the step length of the correction. The corrections for the weights are given by

$$\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)} \delta_j^{(2)}, \quad \text{for } i = 1, \ldots, k+1; j = 1, \ldots, m,$$

and

$$\Delta w_{ij}^{(1)} = -\gamma o_i \delta_j^{(1)}, \quad \text{for } i = 1, \ldots, n+1; j = 1, \ldots, k,$$

where we use the convention that $o_{n+1} = o_{k+1}^{(1)} = 1$. It is very important to make the corrections to the weights only after the backpropagated error has been computed for all units in the network. Otherwise the corrections become intertwined with the backpropagation of the error and the computed corrections do not correspond any more to the negative gradient direction.

4. TRAINING PATTERN

In the case of p>1 input-output patterns, an extended network is used to compute the error function for each of them separately. The weight corrections are computed for each pattern and so we get, for example, for weight $w_{ij}^{(1)}$ the corrections

$$\Delta_1 w_{ij}^{(1)}, \Delta_2 w_{ij}^{(1)}, \ldots, \Delta_p w_{ij}^{(1)}.$$

The necessary update in the gradient direction is then

$$\Delta w_{ij}^{(1)} = \Delta_1 w_{ij}^{(1)} + \Delta_2 w_{ij}^{(1)} + \cdots + \Delta_p w_{ij}^{(1)}.$$

We speak of batch or off-line updates when the weight corrections are made in this way. Often, however, the weight updates are made sequentially after each pattern presentation (this is called on-line training).

In this case the corrections do not exactly follow the negative gradient direction, but if the training patterns are selected randomly the search direction oscillates around the exact gradient direction and, on average, the algorithm implements a form of descent in the error function.

The rationale for using on-line training is that adding some noise to the gradient direction can help to avoid falling into shallow local minima of the error function. Also, when the training set consists of thousands of training patterns, it is very expensive to compute the exact gradient direction since each epoch (one round of presentation of all patterns to the network) consists of many feed-forward passes and on-line training becomes more efficient.

5. ERROR DURING TRAINING

Figure 5 shows the evolution of the total error during training of a network of three computing units.

After 600 iterations the algorithm found a solution to the learning problem. In the figure the error falls fast at the beginning and end of training. Between these two zones lies a region in which the error function seems to be almost flat and where progress is slow. This corresponds to a region which would be totally flat if step functions were used as activation functions of the units. Now, using the sigmoid, this region presents a small slope in the direction of the global minimum.

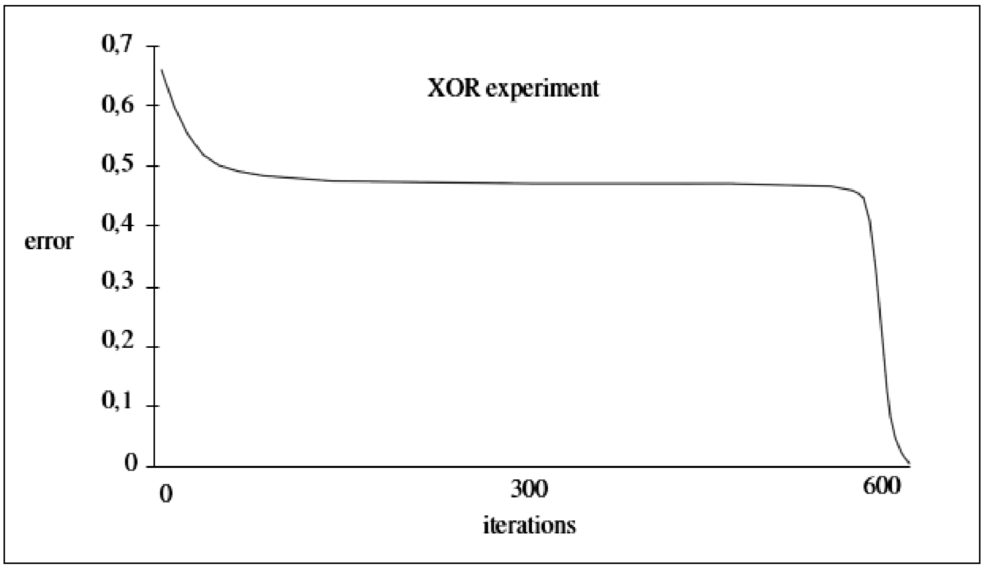


Figure 5. Error function for 600 iterations of backpropagation

6. CONCLUSIONS

The Back-propagation Neural Network (BPNN) is a supervised learning neural network model highly applied in different engineering fields around the globe. Although it is widely implemented in the most practical ANN applications and performed relatively well, it is suffering from a problem of slow convergence and convergence to local minima. This makes Artificial Neural Network's application very challenging when dealing with large problems.

One of the lessons learned over the past years is that significant improvements in the approximation capabilities of neural networks will only be obtained through the use of modularized networks. In the future, more complex learning algorithms will deal not only with the problem of determining the network parameters, but also with the problem of adapting the network topology.

REFERENCES:

[1]. Albrecht, R.F.; Reeves, C.R.; Steele, N.C. (1993) *Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag, Vienna

[2]. Kosbatwar, S.P.; Pathan, **S.K.** (2012) *Pattern Association for character recognition by Back-Propagation algorithm using Neural Network approach*, (IJCSES) Vol.3, No.1, February

[3]. Rajapandian, J.V.V.; Gunaseeli, N. (2007) Modified Standard Backpropagation Algorithm With Optimum Initialization For Feedforward Neural Networks" JISE, GA, USA, ISSN:1934-9955, vol.1, no.3

[4]. Rehman, M.Z.; Nawi, N.M.; Ghazali, M.I. (2011) *Noise-Induced Hearing Loss (NIHL) Prediction in Humans Using a Modified Back Propagation Neural Network*, in: 2nd International Conference on Science Engineering and Technology, pp. 185--189

[5]. Schiffmann, W.; Joost, M.; Werner, R. (1933) *Comparison of Optimized Backpropagation Algorithms*, in: M. Verleysen (ed.), European Symposium on Artificial Neural Networks, Brussels, pp. 97–104

[6]. Zweiri, Y.H.; Seneviratne, L.D.; Althoefer, K. (2005) *Stability Analysis of a Three-term Back-propagation Algorithm*, J. Neural Networks. 18, 1341-1347