

Artificial Intelligence, An Introductory Course



The International Congress for global Science and Technology
www.icgst.com

Instructor: Ashraf Aboshosha, Dr. rer. nat.
Engineering Dept., Atomic Energy Authority,
8th Section, Nasr City, Cairo, P.O. Box. 29
[,aboshosha@icgst.com](mailto:aboshosha@icgst.com) E-mail:
Tel.: 012-1804952

Lecture (2): Artificial Neural Networks

Course Syllabus:

- An introduction to artificial intelligence and machine learning
- Artificial Neural Networks
- Intelligent search techniques
- Neural programming based on Matlab

[This is free educational material](#)

6.4. Learning of multilayer neural network

The adapted perceptron units are arranged in layers, and so the new model is naturally enough termed the multilayer perceptron. The basic details are shown in fig.(12). Our new model has three layers; an input layer, an output layer, and a layer in between, not connected directly to the input or the output, and so hidden layer. Each unit in the hidden layer and the output layer is like a perceptron unit. The units in the input layer serve to distribute the values they receive to the next layer, and so do not perform a weighted sum or threshold. Because we have modified the single-layer perceptron by changing the non-linearity form into sigmoid function, and added a hidden layer, we are forced to alter our learning rule as well. We now have a network that should be able to learn to recognize more complex things; let us examine the learning rule in more details.

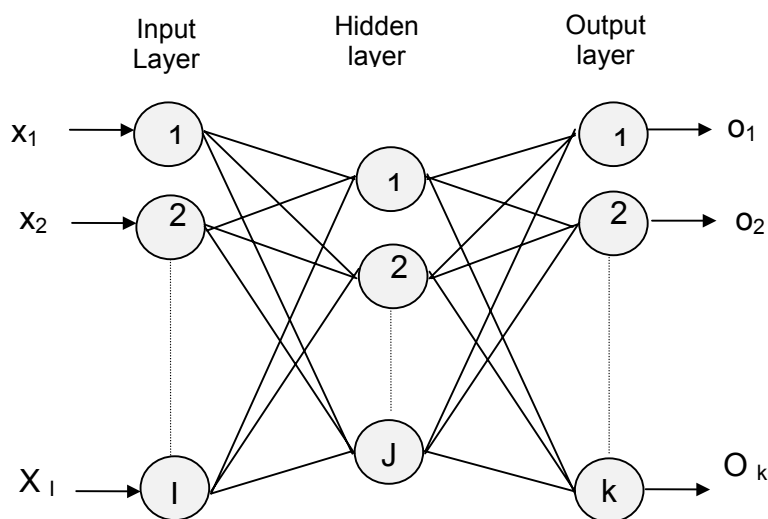


Fig.(12) Multi-Layer Neural Network

6.5. Backpropagation learning rule

The learning rule for multilayer network is called the “Generalized Delta rule”, or the “Backpropagation rule”, and was suggested in 1986 by Rumelhart, McClelland, and Williams. It signaled the renaissance of the whole subject. It was later found that Parker had published similar result in 1982, and then Werbos was shown to have done the work in 1974. such is the nature of science , however; groups working in diverse fields cannot keep up

with all the advances in other areas, and so there is often duplication of effort. However, Rumelhart and McClelland are credited with reviving the perceptron since they not only developed the rule independently to earlier claims, but used it to produce multilayer networks that they investigate and characterized. The operation of the network is similar to that of the single-layer perceptron, in that we show the net a pattern and calculate its response. Comparison with the desired response enables the weights to be altered so that the network can produce a more accurate output next time. The learning rule provides the method for adjusting the weights in the network, and, as we saw earlier in the chapter, the simple rule used in the single-layer perceptron will not work for multilayer networks. However, the use of the sigmoid function means that enough information about the output is available to units in earlier layers, so that these units can have their weights adjusted so as to decrease the error next time.

The learning rule is a little more complex than the previous one, however, and we can best understand it by considering how the net behaves as patterns are taught to it. When we show the untrained network an input pattern, it will produce any random output. We need to define an error function that represents the difference between the network's current output and the correct output that we want to produce it. because we need to know the "correct" pattern, this type of learning is known as supervised learning. In order to learn successfully we want to make the output of the net approach the desired output, that is, we want to continually reduce the value of this error function. This is achieved by adjusting the weights on the links between the units, and the generalized delta rule does this by calculating the value of the error function for that error from one layer to the previous one. Each unit in the net has its weights adjusted so that it reduces the value of the error function; for units actually on the output, their output and the desired output is known, so adjusting the weights is relatively simple, but for units in the middle layer, the adjustment is not so obvious. Intuitively, we might guess that the hidden units that are connected to outputs with a large error should have their weights adjusted a lot, while units that feed almost correct outputs should not be altered much. In fact, the mathematics shows that the weights for a

particular node should be adjusted in direct proportion to the error in the units to which it is connected; that is why back-propagation these error through the net allows the weights between all the layers to be correctly adjusted. In this way the error function is reduced and the network learns.

6.6. Error BACK-PROPAGATION

Fig.(13) illustrates the flowchart of the error back-propagation training algorithm for a basic two layer network as in fig.(12) the learning begins with the feedforward recall phase.

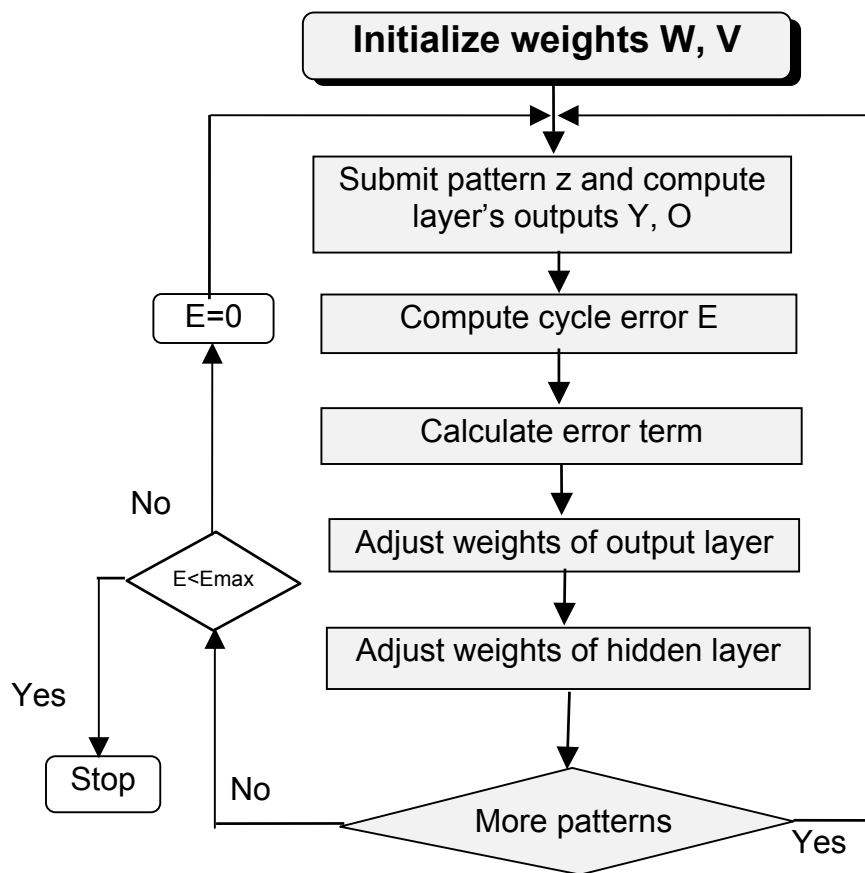


fig.(13) EBPT ALGORITHM

After a single pattern vector z is submitted at the input, the layers' responses y and o are computed in this phase. Then, the error signal computation phase follows. Note that the error signal vector must be determined in the output layer first, and then it is propagated toward the network input nodes. The $K \times J$ weights are subsequently adjusted within the matrix w in step

(5). Finally, $J \times I$ weights are adjusted within the matrix V in step (6). Note that cumulative cycle error of input to output mapping is computed in step 3 as a sum over all continuous output errors in the entire training set. The final error value for the entire training cycle is calculated after each completed pass through the training set $\{z_1, z_2, z_3, \dots, z_p\}$. the learning procedure stops when the final error value below the upper bound, E_{\max} is obtained as shown in step 8.

6.7. Error Back-Propagation Training Algorithm (EBPTA).

Given are P training pairs $\{z_1, d_1, z_2, d_2, \dots, z_p, d_p\}$ where z_i is $(i \times 1)$, d_i is $(K \times 1)$, and $i = 1, 2, 3, \dots, p$. note that the i 'th component of each z_i is of value -1 since input vectors have been augmented. Size $j-1$ of the hidden layer having output y is selected. note that the J 'th component of y is of the value -1, since hidden layer outputs have also been augmented, y is $(J \times 1)$ and o is $(K \times 1)$.

Step 1 : $c > 0$, E_{\max} chosen. Weights W and V are initialized at a small random values, W is $(K \times J)$ and V is $(J \times I)$.

Step 2 : Training step starts here. Input is presented and the layer's output computed.

$$y_j = f(v_j^t z), \text{ for } j = 1, 2, 3, \dots, J$$

$$o_k = f(w_k^t y), \text{ for } k = 1, 2, 3, \dots, K \quad \dots\dots\dots(33)$$

Step 3 : Error Value is computed;

$$E(k) = \frac{1}{2} (d_k - o_k)^2 + E(k-1) \quad \dots\dots\dots(34)$$

Step 4 : Error signal vectors δ_o and δ_y of both output and hidden layer are computed vector δ_o is $(K \times 1)$ and δ_y is $(J \times 1)$.

The error signal term of output layer are

$$\delta_{ok} = 0.5(d_k - o_k)(1 - o_k^2), \text{ for } k = 1, 2, \dots, K \quad \dots\dots\dots(35)$$

The error signal term of the hidden layer in this step are

$$\delta_{yj} = (1 - y_j) \sum_{k=1}^K \delta_{ok} W_{kj}, \text{ for } j = 1, 2, \dots, J \quad \dots\dots\dots(36)$$

Step 5: Output layer weights are adjusted:

$$w_{kj} \leftarrow w_{kj} + c \delta_{ok} y_j, \quad \text{for } k = 1, 2, \dots, K \text{ and } j = 1, 2, \dots, J \quad \dots(37)$$

Step 6: Hidden layer weights are adjusted:

$$v_{ji} \leftarrow v_{ji} + c \delta_{yj} z_i, \quad \text{for } j = 1, 2, \dots, J \text{ and } i = 1, 2, \dots, I \quad \dots(38)$$

Step 7: If $p < P$ then $p \leftarrow p+1$ and go to step 2;
otherwise, go to step 8.

Step 8: The training cycle is completed. For $E < E_{\max}$ terminate the training session. Output weights W, V , and E .

If $E < E_{\max}$, then $E \leftarrow 0$, $p \leftarrow 1$, and initiate the new training cycle by going to step 2.

6.8. Initializing neural network weights

The weights of the network to be trained are typically initialized at a small random values. The initialization strongly affects the ultimate solution. If all weights start out with equal weight values, and if the solution requires that unequal weights be developed, the network may not train properly. Unless the network is disturbed by random factors or the random character of input patterns during training, the initial representation may continuously result in symmetric weights. Also, the network may fail to learn the set of training examples with the error stabilizing or even increasing as the learning continues. In fact, many empirical studies of the algorithm point out that continuing training beyond a certain low-error results in the undesirable drift of weights. This causes the error to increase and the quality of mapping implemented by the network decreases. To counteract the drift problem, network learning should be restarted with other random weights. The choice of initial weights is, however, only one of several factors affecting the training of the network toward an acceptable error minimum.

6.9. Necessary number of hidden neurons

The size of a hidden layer is one of the most important considerations when solving actual problems using multilayer feedforward networks. The problem of the size choice is under intensive study with no conclusive answers available thus far for many tasks. The exact analysis of the issue is rather difficult because of the complexity of the network mapping and due to the nondeterministic nature of many successfully completed training procedures.

6.10. Momentum method

The purpose of the momentum method is to accelerate the convergence of the error back-propagation learning algorithm. The method involves supplementing the current weight adjustments with a fraction of the most recent weight adjustment. This is usually done according to the formula:

$$\Delta w(t) = -c \nabla E(t) + \alpha \Delta w(t-1) \dots\dots\dots(39)$$

Where the arguments t and $t-1$ are used to indicate the current and the most recent training step, respectively, and α is a used selected positive momentum constant. The second term, indicating a scaled most recent adjustment of weights, is called the momentum term. For the total of N steps using the momentum method, the current weight change can be expressed as:

$$\Delta w(t) = -c \sum_{n=0}^N \alpha^n \nabla E(t-n) \dots\dots\dots(40)$$

Typical value of α constant chosen less than unity [25].