Backpropagation

Introduction

The Backpropagation neural network is a multilayered, feedforward neural network and is by far the most extensively used[6]. It is also considered one of the simplest and most general methods used for supervised training of multilayered neural networks[6]. Backpropagation works by approximating the non-linear relationship between the input and the output by adjusting the weight values internally. It can further be generalized for the input that is not included in the training patterns (predictive abilities).

Generally, the Backpropagation network has two stages, training and testing. During the training phase, the network is "shown" sample inputs and the correct classifications. For example, the input might be an encoded picture of a face, and the output could be represented by a code that corresponds to the name of the person.

A further note on encoding information - a neural network, as most learning algorithms, needs to have the inputs and outputs encoded according to an arbitrary user defined scheme. The scheme will define the network architecture so that once a network is trained, the scheme cannot be changed without creating a totally new net. Similarly there are many forms of encoding the network response.

The following figure shows the topology of the Backpropagation neural network that includes and input layer, one hidden layer and an output layer. It should be noted that Backpropagation neural networks can have more than one hidden layer.

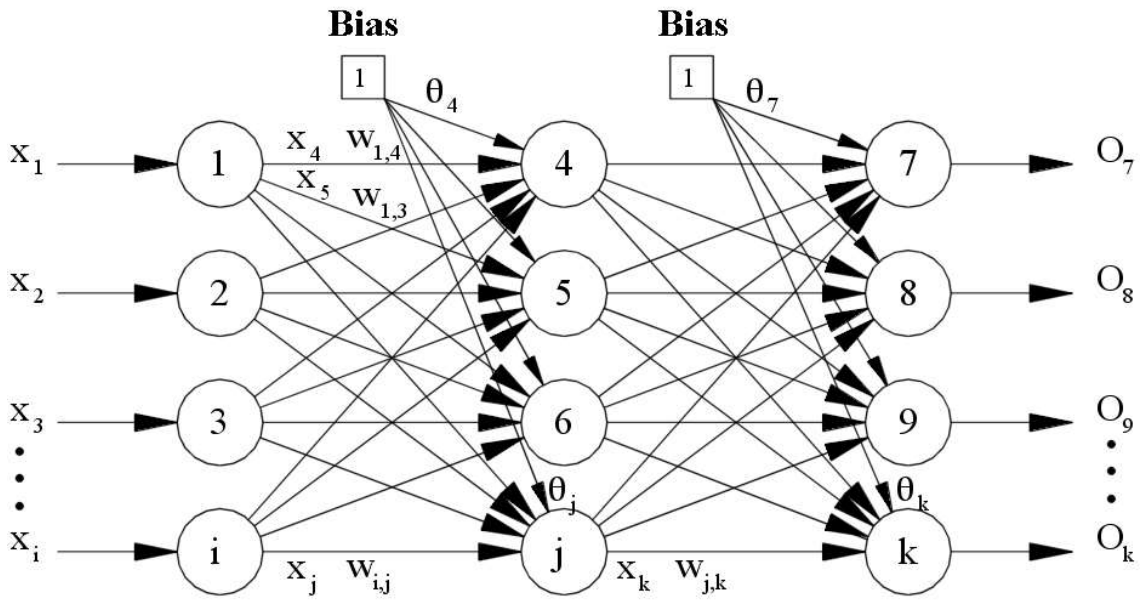


Figure 5 Backpropagation Neural Network with one hidden layer[6]

Theory

The operations of the Backpropagation neural networks can be divided into two steps: feedforward and Backpropagation. In the feedforward step, an input pattern is applied to the input layer and its effect propagates, layer by layer, through the network until an output is produced. The network's actual output value is then compared to the expected output, and an error signal is computed for each of the output nodes. Since all the hidden nodes have, to some degree, contributed to the errors evident in the output layer, the output error signals are transmitted backwards from the output layer to each node in the hidden layer that immediately contributed to

the output layer. This process is then repeated, layer by layer, until each node in the network has received an error signal that describes its relative contribution to the overall error.

Once the error signal for each node has been determined, the errors are then used by the nodes to update the values for each connection weights until the network converges to a state that allows all the training patterns to be encoded. The Backpropagation algorithm looks for the minimum value of the **error function** in weight space using a technique called the delta rule or **gradient descent**[2]. The weights that minimize the error function is then considered to be a solution to the learning problem.

The network behaviour is analogous to a human that is shown a set of data and is asked to classify them into predefined classes. Like a human, it will come up with "theories" about how the samples fit into the classes. These are then tested against the correct outputs to see how accurate the guesses of the network are. Radical changes in the latest theory are indicated by large changes in the weights, and small changes may be seen as minor adjustments to the theory.

There are also issues regarding generalizing a neural network. Issues to consider are problems associated with under-training and over-training data. Under-training can occur when the neural network is not complex enough to detect a pattern in a complicated data set. This is usually the result of networks with so few hidden nodes that it cannot accurately represent the solution, therefore under-fitting the data (Figure 6)[1].

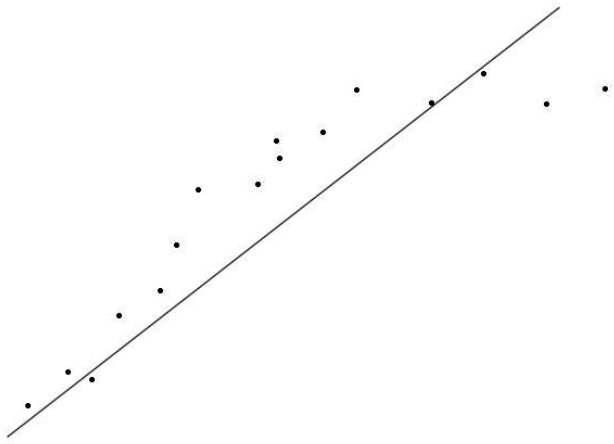


Figure 6 Under-fitting data

On the other hand, over-training can result in a network that is too complex, resulting in predictions that are far beyond the range of the training data. Networks with too many hidden nodes will tend to over-fit the solution (Figure 7)[1].

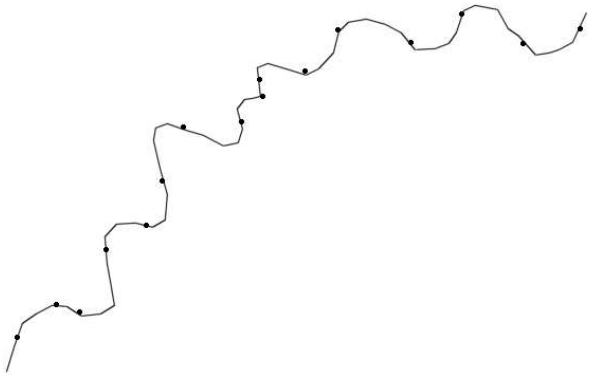


Figure 7 Over-fitting data

The aim is to create a neural network with the "right" number of hidden nodes that will lead to a good solution to the problem (Figure 8)[1].

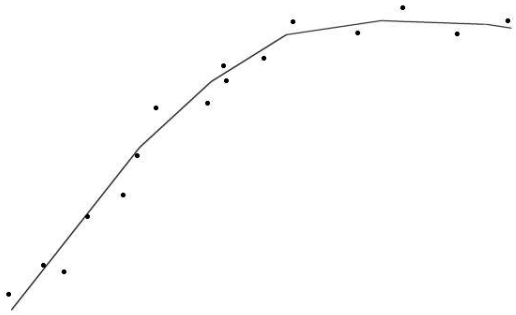


Figure 8 Good fit of the data

Algorithm

Using **Figure 3**, the following describes the learning algorithm and the equations used to train a neural network. For an extensive description on the derivation of the equations used, please refer to reference [10].

Feedforward[10]

When a specified training pattern is fed to the input layer, the weighted sum of the input to the j^{th} node in the hidden layer is given by

$$Net_j = \sum w_{i,j} x_j + \theta_j \quad \textbf{(1)}$$

Equation (1) is used to calculate the aggregate input to the neuron. The θ_j term is the weighted value from a **bias** node that always has an output value of 1. The bias node is considered a "pseudo input" to each neuron in the hidden layer and the output layer, and is used to overcome the problems associated with situations where the values of an input pattern are zero. If any input pattern has zero values, the neural network could not be trained without a bias node.

To decide whether a neuron should fire, the "Net" term, also known as the action potential, is passed onto an appropriate activation function. The resulting value from the activation function determines the neuron's output, and becomes the input value for the neurons in the next layer connected to it..

Since one of the requirements for the Backpropagation algorithm is that the activation function is differentiable, a typical activation function used is the Sigmoid equation (refer to Figure 4):

$$O_j = x_k = \frac{1}{1 + e^{-Net_j}} \quad \textbf{(2)}$$

It should be noted that many other types of functions can, and are, used:- hyperbolic tan being another popular choice.

Similarly, equations (1) and (2) are used to determine the output value for node k in the output layer.

Error Calculations and Weight Adjustments - Backpropagation [10]

Output Layer

If the actual activation value of the output node, k, is O_k, and the expected target output for node k is t_k, the difference between the actual output and the expected output is given by:

$$\Delta_k = t_k - O_k \quad \textbf{(3)}$$

The error signal for node k in the output layer can be calculated as

$$\delta_k = \Delta_k O_k (1 - O_k)$$

or

$$\delta_k = (t_k - O_k) O_k (1 - O_k) \quad \textbf{(4)}$$

where the $O_k(1-O_k)$ term is the derivative of the Sigmoid function.

With the delta rule, the change in the weight connecting input node j and output node k is proportional to the error at node k multiplied by the activation of node j.

The formulas used to modify the weight, $w_{j,k}$, between the output node, k, and the node, j is:

$$\Delta w_{j,k} = l_r \delta_k x_k \quad \textbf{(5)}$$

$$w_{j,k} = w_{j,k} + \Delta w_{j,k} \quad \textbf{(6)}$$

where $\Delta w_{j,k}$ is the change in the weight between nodes j and k, l_r is the **learning rate**. The learning rate is a relatively small constant that indicates the relative change in weights. If the learning rate is too low, the network will learn very slowly, and if the learning rate is too high, the network may oscillate around minimum point (refer to Figure 6), overshooting the lowest point with each weight adjustment, but never actually reaching it. Usually the learning rate is very small, with 0.01 not an uncommon number. Some modifications to the

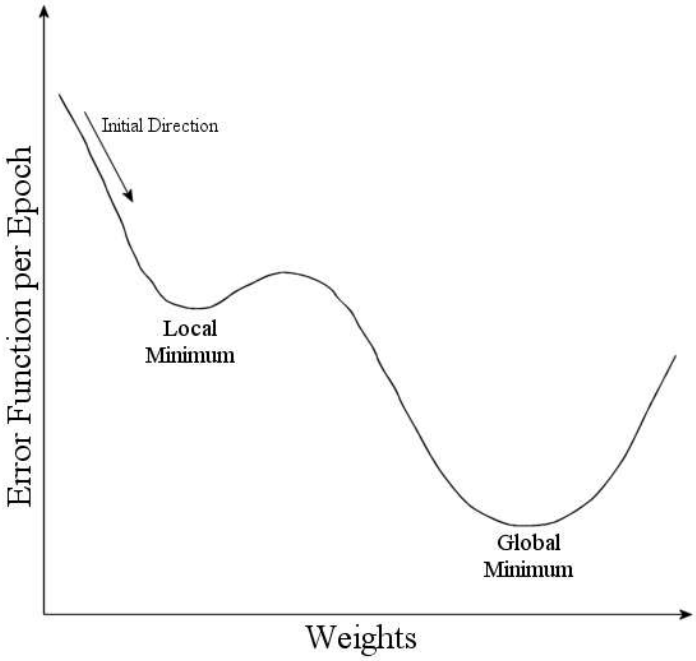
Backpropagation algorithm allows the learning rate to decrease from a large value during the learning process. This has many advantages. Since it is assumed that the network initiates at a state that is distant from the optimal set of weights, training will initially be rapid. As learning progresses, the learning rate decreases as it approaches the optimal point in the minima. Slowing the learning process near the optimal point encourages the network to converge to a solution while reducing the possibility of overshooting. If, however, the learning process initiates close to the optimal point, the system may initially oscillate, but this effect is reduced with time as the learning rate decreases.

It should also be noted that, in equation (5), the x_k variable is the input value to the node k, and is the same value as the output from node j.

To improve the process of updating the weights, a modification to equation (5) is made:

$$\Delta w_{j,k}^{n} = l_r \delta_k x_k + \Delta w_{j,k}^{(n-1)} \mu \quad \textbf{(7)}$$

Here the weight update during the nth iteration is determined by including a **momentum** term (μ), which is multiplied to the (n-1)th iteration of the $\Delta w_{j,k}$. The introduction of the momentum term is used to accelerate the learning process by "encouraging" the weight changes to continue in the same direction with larger steps. Furthermore, the momentum term prevents the learning process from settling in a local minimum. by "over stepping" the small "hill". Typically, the momentum term has a value between 0 and 1.



It should be noted that no matter what modifications are made to the Backpropagation algorithm, such as including the momentum term, there are no guarantees that the network will not settle in a local minimum.

Figure 9 Global and Local Minima of Error Function[3]

Hidden Layer

The error signal for node j in the hidden layer can be calculated as

$$\delta_k = (t_k - O_k) O_k \sum(w_{j,k} \delta_k) \quad \textbf{(8)}$$

where the Sum term adds the weighted error signal for all nodes, k, in the output layer.

As before, the formula to adjust the weight, $w_{i,j}$, between the input node, i, and the node, j is:

$$\Delta w_{i,j}^{n} = l_r \delta_j x_j + \Delta w_{i,j}^{(n-1)} \mu \quad \textbf{(9)}$$

$$w_{i,j} = w_{i,j} + \Delta w_{i,j} \quad \textbf{(10)}$$

Global Error

Finally, Backpropagation is derived by assuming that it is desirable to minimize the error on the output nodes over all the patterns presented to the neural network. The following equation is used to calculate the **error function**, E, for all patterns

$$E = \frac{1}{2} \sum (\sum (t_k - O_k)^2) \quad \textbf{(11)}$$

Ideally, the error function should have a value of zero when the neural network has been correctly trained. This, however, is numerically unrealistic.

Additional Information

Pseudo Coding

The following describes the Backpropagation algorithm[9],[10]

```
Assign all network inputs and output

Initialize all weights with small random numbers, typically between -1 and
1

repeat

   for every pattern in the training set

      Present the pattern to the network

//       Propagated the input forward through the network:
         for each layer in the network
            for every node in the layer
               1. Calculate the weight sum of the inputs to the node
               2. Add the threshold to the sum
               3. Calculate the activation for the node
            end
         end

//       Propagate the errors backward through the network
          for every node in the output layer
            calculate the error signal
         end

         for all hidden layers
            for every node in the layer
               1. Calculate the node's signal error
               2. Update each node's weight in the network
            end
         end

//       Calculate Global Error
         Calculate the Error Function
```

```
    end

while ((maximum  number of iterations < than specified) AND
        (Error Function is > than specified))
```