Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Engineering
1(2), 2013, 120-124

ISSN 2347 – 7393

Review article

LEARNING IN FEEDFORWARD NEURAL NETWORKS

Naveen Malik, Naeem Akhtar, Hardeep Rohilla, Rahul, Pankaj Sharma

Dronacharya College of Engineering, Khentawas,
Farukhnagar, Gurgaon, India

Abstract
In this Research paper, learning in feedforward networks will be considered. A **feed forward neural network** is an artificial neural network where connections between the units do *not* form a directed cycle. This is different from recurrent neural networks. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves forwardly in only one direction, from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network.

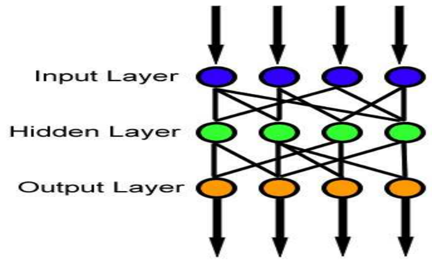


Fig1: Structure of feedforward netural

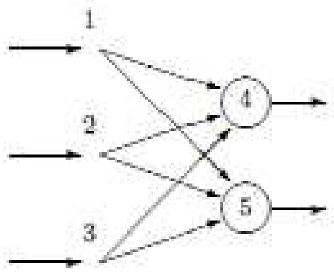
A feedforward neural network is a actually inspired classification algorithm. It comprise of a (possibly large) number of simple neuron-like processing units, organized in layers. Every unit in a layer is associated with all the units in the previous layer. These connections are not all equal, each connection may have a different strength or weight.

1. Introduction
Feedforward neural networks (FF networks) are the most popular and most broadly used Models in many practical applications. They are recognized by many different names, such as "multi-layer perceptrons."

For Correspondence:
malik2008@in.com
Received on: October 2013
Accepted after revision: December 2013
Downloaded from: www.johronline.com

Learning In Feedforward Network
The Feed Forward Network uses a *supervised* learning algorithm: not only the input pattern, but the neural net also needs to know to what category the pattern belongs. A collection of neurons connected together in a

network can be represented by a directed graph:



• Nodes represent the neurons, and arrows represent the links between them.

• Each node has its number, and a link connecting two nodes will have a pair of numbers (e.g. (1, 4) connecting nodes 1 and 4).

• Networks without cycles (feedback loops) are called a feed-forward networks (or perceptron).

Learning proceeds as follows: a pattern is given at the input layer . The pattern will be converted in its way through the layers of the network until it ranges the output layer. The units in the output layer all fit to a different category. The outputs of the network as they are now compared with the outputs as they preferably would have been if this pattern were correctly categorized: in the latter case the unit with the correct category would have had the largest output value and the output values of the other output units would have been very slightly. On the basis of this assessment all the connection weights are attuned a little bit to guarantee that, the next time this same pattern is presented at the inputs, the value of the output unit that agrees with the correct category is a little bit greater than it is now and that, at the same time, the output values of all the other incorrect outputs are a little bit lower than they are now. (The differences between the actual outputs and the venerated outputs are propagated back from the top layer to lower layers to be used at these layers to a mended connection weights. This is why the term *backpropagation network* is also often used to describe this type of neural network.

Time taking in learning :- This is hard to answer. It depends on the magnitude of the neural network, the number of patterns to be learned, the number of epochs, the forbearance of the minimizer and Computing Speed , how much computing time the learning phase takes.

Various learning methods are:-

1.1. Perceptron Convergence Procedure

Perceptron was introduced by Frank Rosenblatt in the late 1950's with a learning algorithm on it. Perceptron may have uninterrupted valued inputs. It works in the same way as the formal artificial neuron distinct before. Its activation is determined by equation:

$$a = \mathbf{w}T\mathbf{u} + \theta$$

The perceptron is a binary classifier which maps its input x (a real-valued vector) to an output value $f(x)$ (a single binary value):

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where w is a vector of real-valued weights, $w \cdot x$ is the dot product (which here computes a weighted sum), and b is the 'bias', a constant term that does not depend on any input value.

The Perceptron Convergence Procedure

Step 1. Initialize weights and thresholds.

• set the connection weights w_j and the threshold value θ to small random values.

Step 2. Present new input and desired output.

• present new continuous valued input $x_0, x_1, \ldots, x_{n-1}$ along with the desired output *d(t).*

Step 3. Calculate actual output calculated by:

$$y(t) = f_n\left(\Sigma_{J=0}^{n-1} w_j(t)\, x_j(t) - \theta \right)$$

Step 4. Adapt weights.

• when an error occurs the connection weights adapted by the neuron by the formula:

$$W_j(t+1) = w_j(t) + \eta\, [d(t) - y\{t)]\, x_j(t)$$

where η is a positive gain fraction that ranges from 0.0 to 1.0 and controls the adaption rate.

Step 5. Repeat by going to Step 2- 4 untill error .

1.2 The Least Mean Square Solution

It is a modification to the perceptron convergence procedure. It reduces the mean square error between the desired output of a perceptron-like net and the actual output. The algorithm is called the **Widrow-Hoff** or LMS algorithm. The LMS algorithm is indistinguishable to the perceptron convergence procedure except that the hard limiting nonlinearity is made linear or replaced by a threshold-logic nonlinearity. Weights are corrected on every test by an amount that depends on the difference between the desired and the actual output. A classifier could use desired outputs of 1 for class A and 0 for class B. During operation, the input would then be assigned to class A only if the output was above **0.5.**

LMS is a fast algorithm that decreases the MSE. The MSE is the average of the biased sum of the error for N training sample which defined as:

$$MSE = \frac{\sum_{j=1}^{N}(R - C_j)^2}{N}$$

where R is the output of the perceptron and Cj is the current test inputs.

In order to train the perceptron by using LMS, we can repeat the test set, taking a set of inputs, calculating the output and then using the error to regulate the weight. This process can be done either arbitrarily by the test set, or for each test of the set in series. The learning rule of LMS is given as:

$$w_{t+1} = w + p(R - C)E$$

The learning rule alters the weight based on the error (R-C or expected output minus actual output). Once the error is calculated, the weights are adjusted by a small amount , *p* in the direction of the input, E. This has the effect of regulating the weights to diminish the output error.

The implementation of LMS is very simple. Initially, the weights vector is initialized with small arbitrary weights. The main repetition then arbitrarily selects a test, calculates the output of the neuron, and then calculates the error. Using the error, the formula of learning rule is applied to every weight in the vector.

1.3 Gradient Descent Algorithm

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps relational to the *negative* of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps relational to the *positive* of the gradient, one approaches a local maximum of that function; the process is then known as **gradient ascent**. Gradient descent is also known as **steepest descent**, or the **method of steepest descent**. When recognized as the later, gradient descent should not be chaotic with the method of steepest descent for approximating integrals.

Enhanced approach would be to let the Adaline Linear Combiner to find the optimum weights by itself through a quest over the error surface. Instead of having a decently arbitrary search, some intelligence is added to the procedure such that the weight vector is changed by considering the gradient of e(**w**) iteratively [Widrow 60], according to formula known as *delta rule*:

$$\mathbf{w}(t+1)=\mathbf{w}(t)+\Delta\mathbf{w}(t)$$

where

$$\Delta\mathbf{w}(t)=-\eta\nabla e(\mathbf{w}(t))$$

In the above formula η is a small positive constant, determining the learning rate.

For the real valued scalar function *e* (**w**) on a vector space **w** ∈ *RN*, the gradient ∇*e*(**w**) provides the direction of the steepest upward slope, so the negative of the gradient is the direction of the steepest descent . This fact is demonstrated in Figure1 for a parabolic error surface on two dimensions.

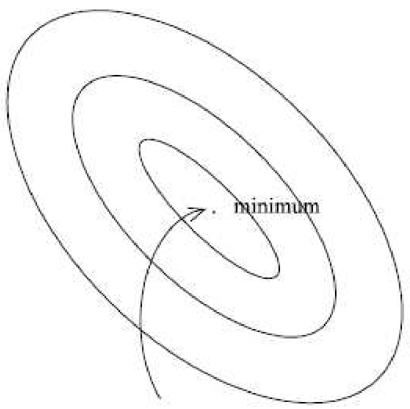


Fig2 : Direction of the steepest gradient descent on the the paraboliod error on two-dimensional weight space . only equpotential curves of the error surface is shown instead of the 3D-error surface .

Stepest descent algorithm

Step 0. Given x^0,set n := 0

Step 1. $d^n := -\nabla f(x^k)$. If $d^n = 0$, then stop.

Step 2. Solve minα $f(x^n + \alpha d^n)$ for the stepsize α^n, perhaps chosen by an exact or inexact linesearch.

Step 3. Set $x^{n+1} \leftarrow x^n + \alpha^n d^n$,$n \leftarrow n + 1$. Go to Step 1.

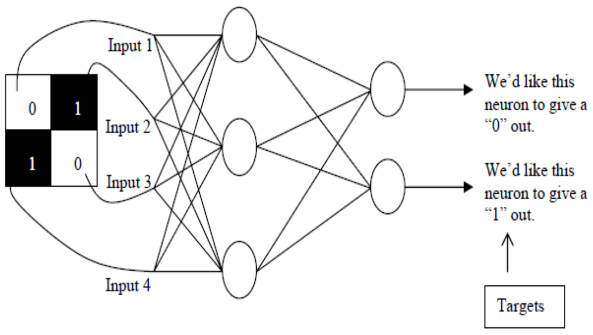
Note from Step 2 and the fact that $d^n = -\nabla f(x^n)$ is a descent direction, it follows that $f(x^{n+1}) < f(x^n)$.

1.4 Back propogation Method

Backpropagation, an abbreviation for "backward propagation of errors", is a common method of training artificial neural networks. From a desired output, the network learns from many inputs, similar to the way a child learns to identify a dog from examples of dogs.

It is a supervised learning method, and is a generalization of the delta rule. For making up the training set, it requires a dataset of the desired output for many inputs. It is most beneficial for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). Backpropagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable.

A Back Propagation network learns by example. You give the algorithm specimens of what you want the network to do and it changes the network’s weights so that, when training is finished, it will give you the vital output for a particular input. Back Propagation networks are perfect for simple Pattern Recognition and Mapping Tasks.


Input 1
0
1
1
0
Input 2
Input 3
Input 4
We'd like this neuron to give a "0" out.
We'd like this neuron to give a "1" out.
Targets

So, if we put in the first pattern to the network, we would like the output to be 0 1 as shown in figure 3.2 (a black pixel is represented by 1 and a white by 0 as in the previous examples). The input and its corresponding target are called a *Training Pair*.

Step of BackProgation algorithm

1. First apply the inputs to the network and work out the output – remember this original output could be anything, as the initial weights were arbitrary numbers.

2. Next work out the error for neuron B. The error is *What you want – What you actually get*, in other words:

ErrorB = OutputB (1-OutputB)(TargetB– OutputB)

The “*Output(1-Output)*” term is necessary in the equation because of the Sigmoid Function – if we were only using a threshold neuron it would just be *(Target – Output)*.

3. Change the weight. Let W+BC be the new (trained) weight and WAB be the initial weight.

W+BC = WBC + (ErrorB x OutputA)

Notice that it is the output of the connecting neuron (neuron A) we use (not B). We update all the weights in the output layer in this way.

4. Calculate the Errors for the hidden layer neurons. Unlike the output layer we can't compute these directly (because we don't have a Target), so we *Back Propagate* them from the output layer (hence the name of the algorithm). This is done by taking the Errors from the output neurons and running them back through the weights to get the errors present in the hidden layer . For example if neuron A is connected as shown to B and C then we take the errors from B and C to generate an error for A.

ErrorA = Output A (1 - Output A)(ErrorB WAB + ErrorC WAC)

Again, the factor *"Output (1 - Output)"* is present because of the sigmoid squashing function.

5. Having obtained the Error for the hidden layer neurons now proceed as in stage 3 to change the hidden layer weights. By repeating this method we can train a network of any number of layers.

2. Applications

Application of Feed Forward neural include:

• Function approximation (modelling)

• Pattern classification (analysis of time-series, customer databases, etc).

• Object recognition (e.g. character recognition)

• Data compression

• Security (credit card fraud)

References :-

1. http://www.fon.hum.uva.nl/praat/manual/Feedforward_neural_networks_1_1__The_learning_phase.html
2. http://en.wikipedia.org/wiki/Perceptron
3. http://urrg.eng.usm.my/index.php?option=com_content&view=article&id=165:learning-algorithms-of-neural-network-least-mean-squarelms-algorithm-&catid=31:articles&Itemid=70
4. The Steepest Descent Algorithm for Unconstrained Optimization and a Bisection Line-search Method Robert M. Freund February, 2004 Massachusetts Institute of Technology.
5. www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf
6. Comparitive Analysis Of Classification Algorithm In Multiple Categories Of Bioinformatics D.Chandra Varma (M.Tech), D.Dharmaiah M.Tech,(Ph.D) ,IJERT