

A Review of Standard Back-Propagation

Emre Uğur

March 7, 2003

Abstract

The aim of this work is to re-study the basic structure of the back-propagation learning algorithm. The other objective is to provide a complete description of this training algorithm with all the technical details. The algorithm is applied to two different benchmark problems and the network is analysed in various aspects. The experiments are mainly conducted to optimize the parameters of the system and to understand the relation between the initial parameter set and nature of the problem.

1 Introduction

Artificial Neural Networks (ANNs) has been studied thoroughly in numerous applications and in various problems since mid-80's after the publication of *Learning internal representations by back-propagating errors* [1]. Although very important modifications and optimizations have been done in the architecture and in the training algorithms since the old back-propagation (in short back-prop) days, it is worth to re-analyze to understand the basics before going into detailed analysis of complex ones.

Thus, the aim of this work is to re-study the basic structure of the back-prop algorithm. The other objective of this study is to provide a complete description of this training algorithm with all the technical details. The algorithm is applied to two different benchmark problems and the network is analysed in various aspects. The experiments are mainly conducted to optimize the parameters of the system and to understand the relation between the initial parameter set and nature of the problem.

A standard full-interconnected Feed-Forward Neural Network (FFANN) is constructed with one input, one hidden and one output layers. The details of the network will be given in Section 2. First, a famous benchmark problem, a constraint form of the parity checking, XOR problem is used to train network (Section 3). The nature of error reduction during the learning process is examined and parameters are adjusted to obtain a faster convergence rate in learning. Then, the same network is employed in a pattern recognition problem with different hidden unit numbers. In the problem, network is trained to differentiate *digits* which are encoded in a 3x5 image grid. Although in pattern recognition problems pre-processing has a crucial importance to obtain good results, since problem is a very basic one and discrete in its nature, no special attention is given. You can find the details of work in Section 4.

Back-propagation learning algorithm mainly consists of two phases, one information flow in forward direction and one flow in backward direction. In first phase, activation of hidden units are propagated to output units. Then, using activation values of output units and desired values, the error is calculated. In the second phase, this error is propagated in backward direction, to modify weights connecting different levels of units(neurons). In both problems, a set of input-output patterns are given continuously, in order to minimize the difference between target and network output values by the means of modifying connection weights. Input and output values can get only values -1 or 1. The learning takes place incrementally, which means weights are updated just after each pattern's presentation. You can find detailed discussion on the algorithm in Section 2.

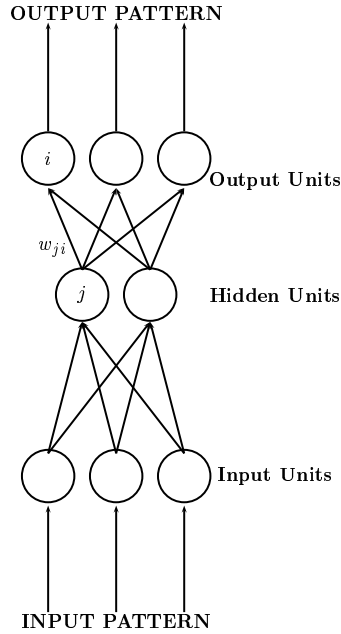


Figure 1: A sample full-interconnected feed-forward artificial neural network. In this network 3 input units are connected to 2 hidden units, and these hidden units are connected to 3 output units. Input pattern is given from input units and network gives the result by the means of its output units.

2 Back Propagation Algorithm

In the back-prop algorithm, a set of input-output patterns are given to the network (Figure 1). If there is a difference between actual and desired outputs, weights are updated in order to reduce error. The generalized delta rule is used to obtain such a behavior.

An activation value is computed for each hidden unit. Activation value could be simply the sum of weighted input values, but in order to scale the sum within a range, different functions can be used. As a squashing function, sigmoid is one alternative to obtain a range $[0, 1]$. However, because input-output patterns are bi-polar, hyperbolic tangent function

will be used to obtain a feasible range $([-1, 1])$. Thus, activation function ¹ is

$$o_{pj} = \tanh(net_{pj}) \quad (1)$$

where p denotes the corresponding pattern and j is the neuron whose output will be computed.

$$net_{pj} = \sum_i w_{ji} net_{pi} \quad (2)$$

Synaptic weight from unit i to unit j , w_{ji} (Figure 1), is multiplied by the output(activation) of each low level connected units i to obtain a net_{pj} . Both hidden and output unit activations are computed in the same manner. The meaning of such a calculation is that each low level activated neuron affects a higher level neuron proportional to the synaptic weight, both in inhibitory and excitatory sense.

Next step is to compute error using the difference between target and actual output:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \quad (3)$$

In order to reduce error, we must follow the steepest path on the error-weight surface. To reach a global minimum, or at least local minimum error value, weight updates should be implemented such that derivative of error with respect to each weight is proportional to the weight change. If a constant, μ , is defined as the learning rate, Δw is found using the formula:

$$\Delta_p w_{ji} = \mu \frac{\partial E_p}{\partial w_{ji}} \quad (4)$$

Chain rule is applied to find the derivative of error:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} \quad (5)$$

Second part of the above equation is derived using Equation 2 such that,

$$\frac{\partial net_{pj}}{\partial w_{ji}} = o_{pi} \quad (6)$$

¹ *Activation* and *output* will be used in exactly same meaning throughout the text.

From now on, we will call δ_{pj} as error signal:

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} \quad (7)$$

Above calculations show us that, weight change is directly affected by the error signal and activation of the connected unit.

First part of Equation 5 is solved using chain rule:

$$\frac{\partial E_p}{\partial net_{pj}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}} \quad (8)$$

For output units, using Equation 3

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) \quad (9)$$

For hidden units,

$$\frac{\partial E_p}{\partial o_{pj}} = -\sum_k \delta_{pk} w_{kj} \quad (10)$$

where k 's are the output units which gives their error signals to hidden units. This equation shows us how the lack of information in hidden units is solved. Because there is no target value for hidden activations, error is propagated from output units proportional to the synaptic weights.

Recall Equation 1, second term in the Equation 8 directly will be

$$\frac{\partial o_{pj}}{\partial w_{ji}} = 1 - o_{pi}^2 \quad (11)$$

where o_{pi} is the i^{th} input value of the pattern if j^{th} unit is a hidden unit, or o_{pi} is the activation of i^{th} hidden unit if j^{th} unit is an output unit. It is the result of the derivation property of hyperbolic tangential function, $\frac{\partial tanh(x)}{\partial x} = 1 - tanh(x)^2$.

In summary, error signal in output units is computed using the formula,

$$\delta_{pj} = (t_{pj} - o_{pj})(1 - o_{pi}^2) \quad (12)$$

and for hidden units,

$$\delta_{pj} = (1 - o_{pi}^2) \sum_k \delta_{pk} w_{kj} \quad (13)$$

-1	-1	→	-1
+1	+1	→	-1
-1	+1	→	+1
+1	-1	→	+1

Table 1: XOR Problem, input-output pattern set

To remind, lets write Equation 4 again:

$$\Delta_p w_{ji} = \mu \delta_{pj} o_{pi} \quad (14)$$

The only missing point up to now is the bias weights. Bias term for each neuron can be thought as a connection from another neuron which is always ON/1. As a result, there will be no difference from other weights in the means of modification of self, except there will be no error backpropagation through this connection.

In the next sections, *XOR* problem and *digit recognition* problem will be used as test cases for the described network.

3 XOR Problem

Although XOR or in general parity problems are one of the most famous benchmark problems in the history of neural networks, Fahlman proposed these type of test cases were not suitable for "*real-world pattern-classification tasks*". He reached such a result because in pattern classification, similar data tend to be grouped in the same output while in parity problems most similar data inputs give different outputs. Since the non-linear nature of the exclusive-or problem requires a hidden layer in the network and number of units is low, it is thought to be a good starting point. Table 1 shows the input/output pattern set, which is given to the network.

XOR problem deserves its fame to its well-known non-linearity characteristic. Since a decision space which is separated linearly into two regions would be insufficient to solve the problem, hidden layers are used to obtain non-linearity. But how could the classification boundary compose of several regions? To visualize such a surface, a network is trained using

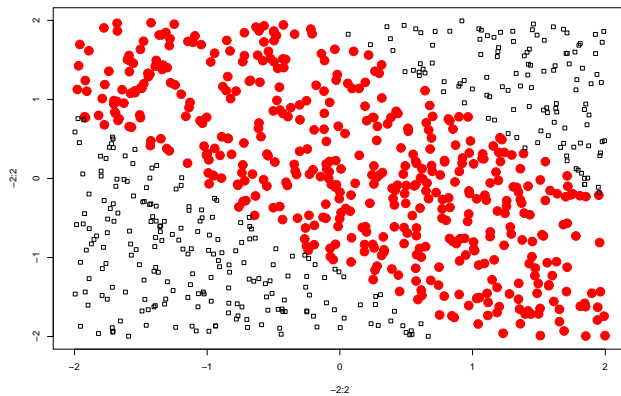


Figure 2: Non-linear decision surface created by two hidden neurons in the XOR problem.

back-prop algorithm in order to solve XOR problem. Then, 1000 real valued input vectors are randomly chosen in the range $[-2, +2]$ and activation values are found. Each input point is plotted onto a surface according to its sign, Figure 2 shows the decision surface for this particular problem.

As one of the main objectives of this work (Section 1) is to analyse parameters in the network and to optimize speed and accuracy of the back-prop algorithm by means of modifying these parameters, learning rate should be the first to be looked for. In three different values, learning rate is examined and an optimum value is found for this problem. The value of learning rate highly depends on the initial range of synaptic weights. I will restrict myself to adjust one range for weights and postpone the study of weight optimization to another time. Weights are randomly initiated at the beginning of Neural Net construction to the range $[0, 1]$. Now let's plot the error graph with respect to epoch number (Figure 3) to examine characteristics of curves for each learning rate. Here epoch number denotes the number of pattern sets, which are presented to the network. The largest one (0.25) drops the error instantly to 2.5, then error starts to oscillate in a rather big range. μ is so

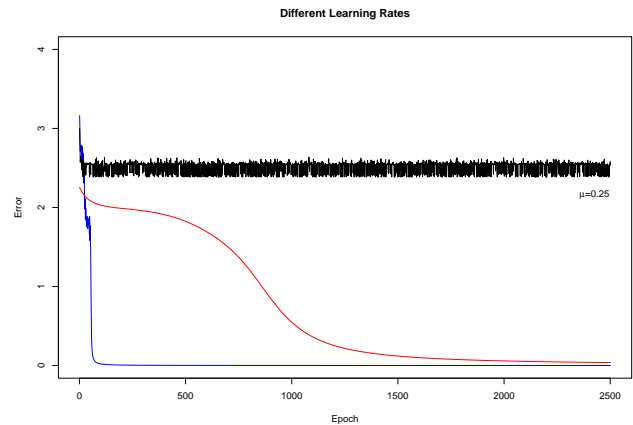


Figure 3: Error back-propagation learning algorithm is applied to learn XOR problem. Error signal and error is implemented as real values to clearly show the effect of different learning rates in the behavior of error change. Learning rates are from above to below 0.25, 0.005, 0.05

large to be able to move in small steps in the gradient descent to find a global minimum. The lowest learning rate (0.005) could reach a global minimum after a very long period. It is a sample proof of the famous theory, "if there is a global minimum, you can certainly reach to that point by moving downwards with infinitely small steps". The optimal value for the learning rate seems to be 0.05 since it is very fast and effective.

Above discussions say little about the accuracy of the experiments since it deals with a continuous activation and output. However in the problem context, outputs are bi-polar so we should find a way to map real results of activation functions to corresponding discrete values. In order to measure the success of the algorithm, a new error calculation should be performed which will be only used for that purpose. To change the error function would be a disastrous attempt since all calculations takes their roots from the original error function. The only alternative is to

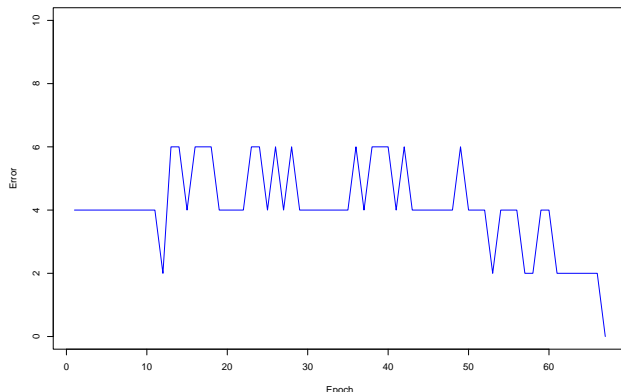


Figure 4: $\mu = .10$, initial connection weight values are in the range $[0, 1]$

map real values directly into -1 or 1:

$$(o) = \begin{cases} -1 & \text{if } o < 0 \\ +1 & \text{if } o > 0 \end{cases}$$

After adjusting the calculation of the error, tens of experiments are done. In general, XOR problem converges in 60-130 iterations, Figure 4 shows a typical step graph of the *error* wrt. *number of epochs*. The large oscillations in the discrete surface is not absurd since it is not directly resulted from a derivative calculation but it is the sum of something we define as error.

4 Pattern Recognition Problem

Pattern recognition problems deserve more than to be mentioned as test cases of neural networks. Since the beginning, tens of leading researchers of Neural Networks, were attracted and trapped in this field while they were studying in image processing. Current problem is a very restricted one, which looks for if the error back-propagation learning algorithm has the capability to train a feed-forward neural network

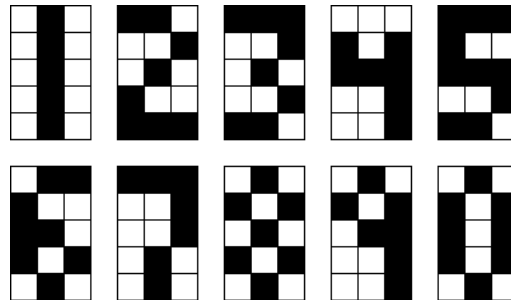


Figure 5: Input Pattern Set

in the task of digit recognition. Inputs will be given in the form of 3×5 binary grid and output is the binary representation of digits. Figure 5 shows the binary retina, which will be serialized as a vector to be given into the network.

First experiment set addresses the usage of multi-layer network, which consists of 15 input and 4 output neurons. Except the hidden unit calculations, all computations can be applied in the same manner to input and output units and the synaptic weights between them. There are two phases again, in first phase, using input pattern activation of output units are directly computed (Equation 1). Then the error signal is extracted using Equation 12, and weights are updated according to Equation 14. 100 iterations in average was sufficient to train the network with a learning rate of 0.2. Figure 6 shows a nice error decreasing function which takes approximately 80 epochs to reach zero.

The next experiment set is performed using different number of hidden units. Using 2 hidden units, back-propagation algorithm cannot succeed in recognizing digits. First, different static learning rates were given to the network, then rate is changed throughout the training process. Additionally, I played with the initial weight ranges, but no modification in the parameters changed the result. In the first 60 iterations, error decreases 66%, then an oscillating movement starts in the graph and goes on.

With 5 or 10 hidden units, to my surprise, network is trained in very few iterations. In average, respectively 20 and 40 iterations were enough to reach a

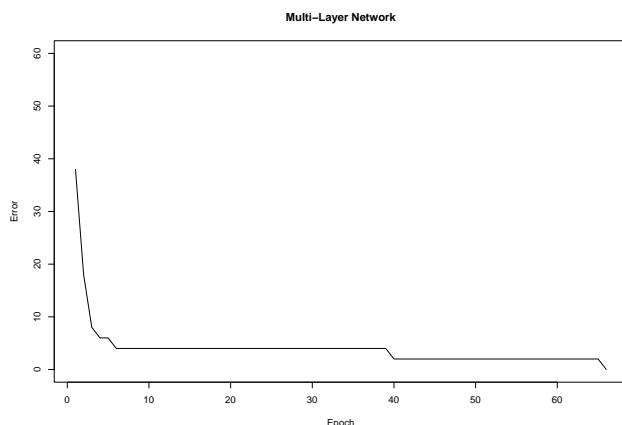


Figure 6: Error vs Epoch Number. This graph is taken from the training process of a multi-layer network for digit recognition task

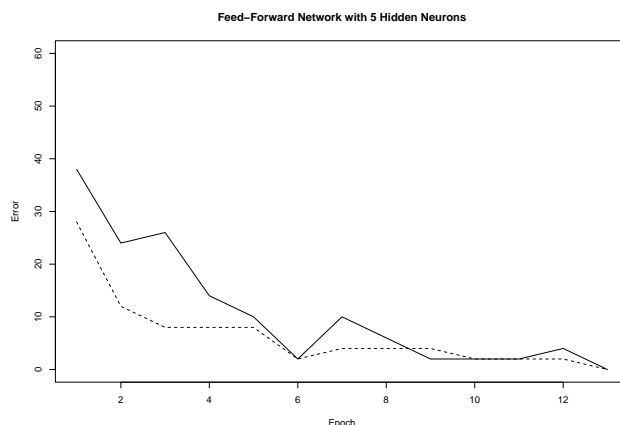


Figure 7: Error vs Epoch Number. Neural Network is trained by an input-output pattern set whose error line is on the above. Dotted line on the below may be viewed as test data, whose error is decreased indirectly by a similar pattern set

result . I tried to decrease hidden neuron number to understand the limits of the internal representation capacity. Using 3 hidden units, approximately 200 iterations was sufficient to obtain good results. However independent from the iteration number, I concluded that it is impossible to solve the pattern recognition problem using 2 hidden neurons and standard back-propagation learning algorithm. When the hidden unit number is increased the the epoch number for the network to become stable will increase too. It is worth to mention that the speed of the training process not only depends on the iteration number. The time spent to train network with only one input-output pattern is another important criterion which should be taken account. In the actual problem, while the number of hidden units are increasing at the same time, speed of the one training epoch decreases non-linearly.

In real-world pattern recognition problems, noise factor should be inserted into the problem like in all other realistic tasks. We can discuss noise in this problem context as bit shifts or bit flips. When only one bit of the input is changed for each pattern, the success rate of the trained network is $\frac{149}{150}$ in bit level

and if two bits changed for each input vector, the gap between actual and desired largens to $\frac{10}{150}$ for a network with 5 hidden neurons. Figure 7 illustrates the error change in a noisy data vector, while it is trained by another input vector.

References

- [1] D.E.Rumhart,D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol.1, Cambridge, MA:MIT Press, pp.318-362, 1986.
- [2] Scott E. Fahlman. *An Empirical Study of Learning Speed in Back-Propagation Networks*, CMU-CS-88-162, September 1988.