JCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Issue-7 E-ISSN: 2347-2693

Class Label Prediction using Back Propagation Algorithm: A comparative study with and without Thresholds (Bias)

N.V. Saiteja Reddy[1*] and T. Srikanth[2]

[1*,2]*Department of CSE, GITAM University, Visakhapatnam, Andhra Pradesh, India*

www.ijcseonline.org

Received: Jun/09/2015 Revised: Jun/28/2015 Accepted: July/18/2015 Published: July/30/ 2015

ABSTRACT: The Back propagation Algorithm is a multilayered, feed forward neural network and is one of the most popular and efficient techniques used. This can be used for dataset classification with suitable combination of training, learning and transfer functions. However, there are some problems associated with this Algorithm like Step-size Problem and Local Minima. In this paper we will discuss about the working of the algorithm and efficient ways to perform learning by overcoming the problems in it. We use three common classification problems to illustrate the ways of efficient learning. All the methods and algorithms were implemented using the features of Java.

Keywords: Back Propagation Algorithm, Neural Network, Programming Neural Networks

1. INTRODUCTION

The Back propagation algorithm [6] performs learning on a multilayer feed-forward neural network [7]. This has two phases, training and testing (in some cases validation). In the former phase, we make it learn by providing set of training examples along with their respective class label or outputs. In the latter phase, we will try to test its performance based on its correctness in classification.

The Feedforward Neural Network is made up of set of neurons arranged in numerous layers. They are of three types viz, an input layer, hidden layer and an output layer. Learning in Back propagation works by approximating the non-linear relationship between the input and output neurons by adjusting the weights. We can perform the training phase with or without the Thresholds (bias) [4]. We will see the difference in the learning in both the cases i.e. with and without Thresholds. We will also see the efficient ways of selecting the number of Hidden Neurons, Learning rate and Momentum [6]. We will also determine how efficiently the algorithm performs on changing the ratio of the Training Set and the Test Set.

The Feedforward Neural Network is a type of Neural Network architecture where the connections are '*Fed-Forward*' i.e. do not form cycles. The term Feedforward is also used when you input something at the input layer and it travels from input to hidden and from hidden to output layer. The Values are fed-forward.

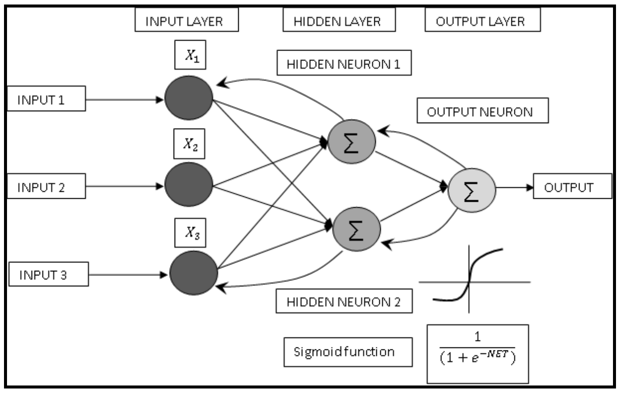


Figure 1: FeedForward Network and BackPropagation

Back propagation performs two steps:

1. Feed forward the values
2. Calculate the error and propagate it back to the earlier layers.
   So, forward propagation is a part of the Back propagation Algorithm.

And the process of training has two different ways:

1. Update weights after all errors for one input vector are calculated.
2. Update weights after all errors for all the input vectors are calculated.

We perform the training by using the second way, as our process of learning is a steepest descent method [6]. The first way is acceptable when your weights update do not interfere with your error computation. The same applies to the bias or Thresholds.

2. Back Propagation Learning Algorithm

Different datasets can be used for performing the back propagation algorithm. In this paper, we use Iris Dataset [2], Wine Dataset [8] and Breast Cancer Dataset [1] for understanding the algorithm and analyzing different ways to improve its efficiency in Learning. We also make use of some of the previous works done on this Algorithm and try to analyze the performance.

We use *Sigmoid* Function [4] as the transfer function. Sigmoid activation allows for a smooth curve of real numbers from 0 to 1. We normalize the dataset before we input the Training Set to the algorithm. There are many Normalization Techniques, here we do column normalization. If the output is an integer value, we convert it into binary form so that it can be represented in 0's and 1's.

The number of input neurons depends on the number of Attributes present in the dataset. Similarly the number of output neurons depends on the size of the binary number of the output. (For Example, for class labels 1, 2, and 3, they can be represented as 0, 0, 1 and 0, 1, 0 and 0, 1, 1. So, we have 3 Output Neurons). We will determine the Number of Hidden Neurons to be taken for a particular Dataset by using some previous works.

We use a parameter called *Learning Rate* which is Training Parameter that controls the size of weights and bias changes in Learning.

The Back propagation Algorithm technique is based on the gradient descent method that tries to minimize the error by moving down the gradient of the error curve. The Weights of the networks are updated by the algorithm, thus reducing the error. But in Back propagation there are some limitations like slow learning or getting trapped at Local Minima. These problems are completely depended on the type of network architecture we construct.

In *Gradient Descent* we start at some point on the error function and attempt to move to the global minimum of the function. In case of simple functions (fig-2), this works.

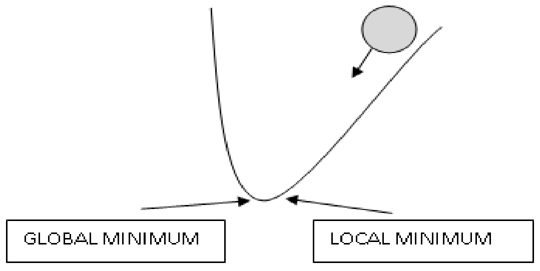


Figure 2: Gradient Descent General Cases

But if we consider real problems (fig-3), error surfaces are typically complex as shown in the above figure. Here we will find many Local Minima, and thus we can't make the progress. Thus, in order to escape this Local Minima we will make use of *Momentum* [0, 1].

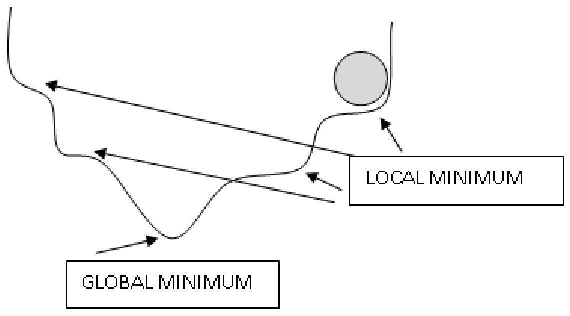


Figure 3: Gradient Descent Complex Cases

We will also determine how to construct a Network Architecture which is more efficient by changing the values of the parameters. The Steps we perform in Back Propagation Algorithms are as follows;

a. Initialization

The number of input neurons depends on the number of attributes.The number of hidden neurons can be determined by some rule-of-thumb methods based on the previous works [5].The number of output neurons can be determined by the size of the binary number of the Class Label.

The weights and thresholds (bias) values are assigned randomly between

(-1, 1) or (-0.5, +0.5). Here, we use Random function [4] and use seed to have same weights and thresholds every time we run the algorithm.

Random rand = new Random (int seed);seed is any integer value

Learning rate is taken in the range from 0.01 to 1.0. But too low a learning rate makes the network learn very slowly. Too high a learning rate makes the weights and objective function diverge, so there is no learning at all. We will determine which value to be taken depending on the observations.

Momentum is taken in the range from 0.1 to 0.9. We will also study the observations for various momentum values.

b. Feed-Forward

The input values from the training set are sent to the Input Layer Neurons, and the values are fed- forward for obtaining the output.

$$O_i(P) = \text{sigmoid}[\sum_{j=1}^{n} \boldsymbol{X_{ij}(P) * W_{ij}(P) + \theta_i}]$$

Where,**P** is the P^th training record in the training set, **j** is the j^th Neuron, $\mathbf{X_j}$ is an input into a neuron, and $\mathbf{O_i}$ is the resulting output. W_{ij} is the weight and $\boldsymbol{\theta_I}$ is the bias (Threshold) value.

$$O_{sigmoid} = \frac{1}{1+e^{-x}}$$

The sigmoid represents the Sigmoid Transfer Function.

c. Finding Errors and Updating Weights
In this step, the error is calculated depending on the Original Output and the Target Output.

$$\partial_i(P) = O_i(P) * [1 - O_i(P)] * e_i(P)$$

$$e_i(P) = T_i(P) - O_i(P)$$

Global error +=$[T_i(P) - O_i(P)]^2$

Where, $\partial_j$is the error gradient for the ith Output Neuron, **T** is the target output for record Pwhere, **P** is the Pth Training record. **Global Error** is the error for the particular training Record.

$$error_P = \sqrt{\frac{Global\ error}{(no.of\ training\ records) * (no.of\ output\ neurons)}}$$

We calculate the error by taking average error across all the training set elements. This is also known as Root Mean Square error (RMS) [4]. After the RMS error has been calculated, Global error is again made *Zero*, in order to calculate the new global error for the next iteration.

Now we need to update the weights for the Output Neurons.

$$W_{ji}(P+1) = W_{ji}(P) + \Delta W_{ji}(P)$$

$$\Delta W_{ji}(P) = \alpha * O_j(P) * \delta_i(P) + \mu * \Delta W_{ji}(n-1)$$

Where **α** is the Learning Rate, **µ** is the Momentum, $\Delta W_{ji}(n-1)$ is the accumulated weight of the previous iteration.
Now, we calculate errors for the hidden layer neurons.

$$\partial_j(P) = O_j(P) * [1 - O_j(P)] * \sum_{i=1}^{n} \partial_i(P).W_{ji}(P)$$

Now we need to update the weights for the Hidden Neurons.

$$W_{ji}(P+1) = W_{ji}(P) + \Delta W_{ji}(P)$$

$$\Delta W_{ji}(P) = \alpha * X_j(P) * \delta_i(P) + \mu * \Delta W_{ji}(n-1)$$

Similarly, bias (Thresholds) for every neuron are updated same as the weights.

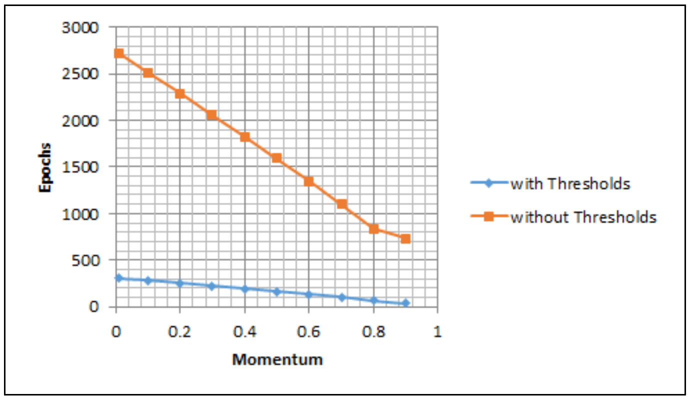
d. Iteration
The steps **b**, **c** repeats until certain condition is met.
Our termination condition [3] is met when the error isless than**0.1**.
The Results obtained in our paper are, Total Number of Errors, Epochs (Number of Iterations), Execution Time.

3. Analyzing the Results and Identifying the Network Architecture and Learning Parameters

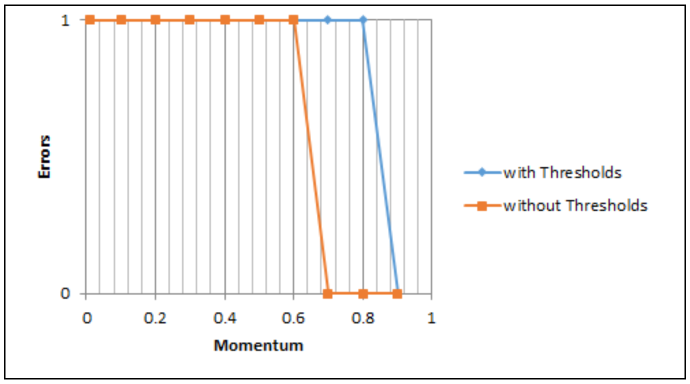
We train the algorithm in two different ways, one is with Thresholds (bias) and the other without Thresholds (bias). We will analyze the difference in both the cases. We also try to analyze what values of parameters are required, so that the Network Architecture is fast and efficient in Learning.

a. IRIS DATASET
We divide the records such that there are 120 (80%) Training Records and 30 (20%) Test Records.

Determining the Momentum:
With values ranging from 0.01 to 0.9 making others constant, we will determine the value for the momentum for which the Learning is efficient and simple. The number of Input Neurons are **4**, number of Hidden neurons are **4**, number of Output Neurons are **3**, and Learning Rate is **0.1**.

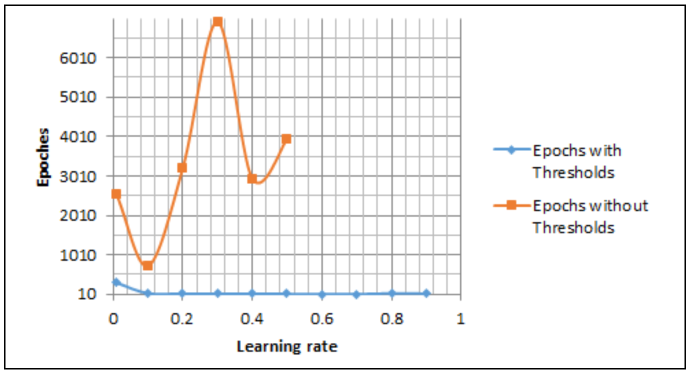


By observing the results, we can easily say that, learning the algorithm with Thresholds (bias) is much faster than the algorithm without Thresholds (bias) depending on the Epochs both have taken to Learn.



In both cases, Learning is fast and efficient when the Momentum is **0.9**.

Determining the Learning Rate:
With values ranging from 0.1 to 0.9 making others constant, we will determine the value for the learning rate for which the Learning is efficient and simple. The number of Input Neurons are **4**, number of Hidden neurons are **4**, number of Output Neurons are **3**, and Momentum is **0.9**.

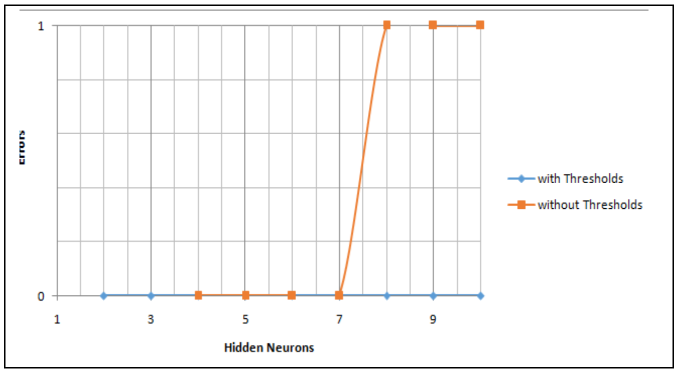


By observing the results, the performance of the algorithm is efficient when the Learning rate is 0.01 or 0.1, but learning rate with 0.1 is more efficient as it takes less time for Learning. The value of the Learning rate also depends on the number of records in the dataset. As said before, too much learning rate is also not advised. By observing results, the algorithm without bias is unable to perform learning when learning rate is greater than **0.5.**

Determining the Hidden Neurons:
By using the rule-of-thumb methods [5] we can estimate valid values for the number of Hidden Neurons and then perform analysis.
The number of Input Neurons are **4**, number of Output Neurons are **3**, Momentum is **0.9**, and Learning rate is **0.1**.
We will use the Rule for estimating,
By using Rule,

$\frac{90}{100} * 4$ To $\left[\frac{90}{100} * 4\right] + 3$ = 4 to 7 Hidden Neurons

Now, the obtained results are,



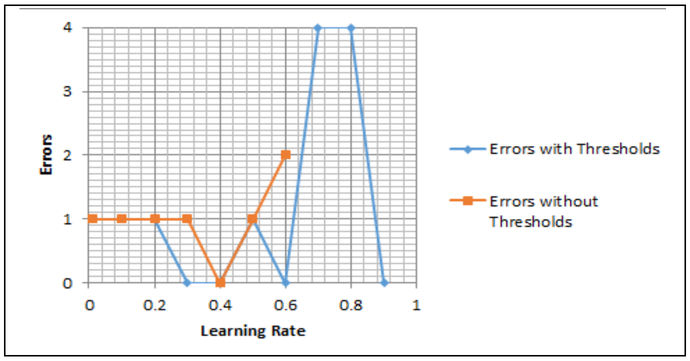
From the above results, we can see that the algorithm can perform efficiently when the Number of Hidden Neurons in the Network Architecture is 4, 5, 6, and 7. If we consider only 1 Hidden Neuron, then there is no learning at all.
Thus a Network Architecture with Learning Rate 0.1, Momentum 0.9 and Number of Hidden Neurons as [4, 5, 6, and 7] can perform efficiently.

b. WINE DATASET

We divide the records such that 80% of the records are in Training Set and 20% are in Test Set, such that there are 142 Training Records and 36 Test Records.
Now we perform learning by calculating the number of Hidden Neurons required to classify this dataset efficiently.
Determining the Hidden Neurons:
Again by using the rule-of-thumb methods, we estimate valid values for the number of Hidden Neurons and then perform analysis.
The number of Input Neurons are **13**, number of Output Neurons are **3**, Momentum is **0.9**, and Learning rate is **0.1**.
We will use the First Rule for estimating,

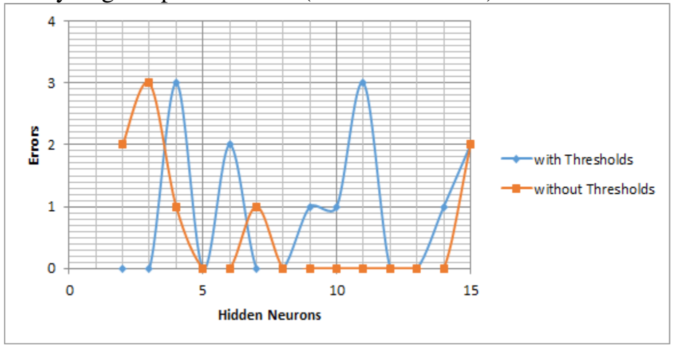
$\frac{90}{100} * 13$ **To** $\left[\frac{90}{100} * 13\right] + 3$ = 12 to 15 Hidden Neurons

But, by performing learning with these values the observed results are not satisfying. The learning process is slow and the number of errors are not changing for any number of Hidden Neurons. So in order to speed up the process of learning we need to update the value of Learning Rate.
Determining the Learning Rate:
The number of Input Neurons are **13**, number of Hidden neurons are **13**, number of Output Neurons are **3**, and Momentum is **0.9**.



By observing the above results, with Learning Rate as 0.4 the performance of the Algorithm and Learning is increased. Now, we will analyze how the algorithm performs when learning rate is **0.4.**

Analyzing the performance (Hidden Neurons):

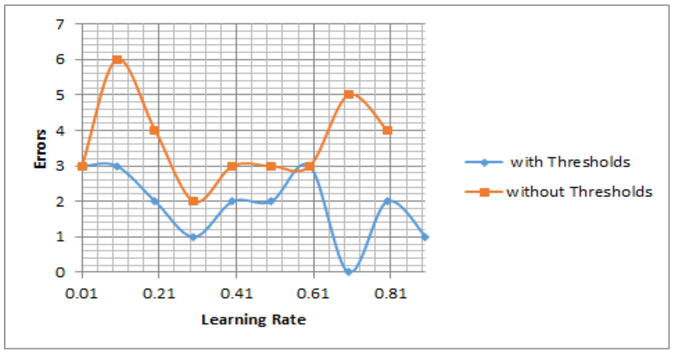


From the above results, we can see that the performance and Learning is increased if we vary the value of learning rate to some extent.

c. BREAST CANCER DATASET

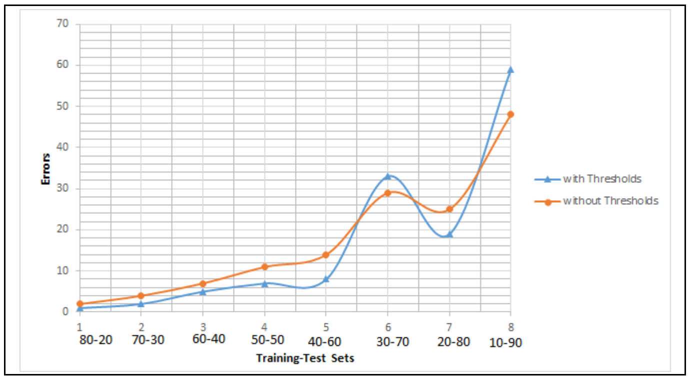
We divide the records such that 80% of the records are in Training Set and 20% are in Test Set. Such that there are 547 Training Records and 136 Test Records.

Determining the Learning Rate:
The number of Input Neurons are **9**, number of Hidden neurons are **9**, number of Output Neurons are **3**, and Momentum is **0.9**.



Being a large Dataset, very low learning rate can't perform well. As seen in the previous datasets learning rate values with 0.01 and 0.1 performs well but in this case it varies. Learning rate values with greater than 0.3 and less than 0.7 can be used for better performance. Learning rate with greater than 0.8 causes over-fitting and it doesn't learn at all. Now we will vary the sizes of Testing and Training Sets and analyze the changes that take place in the Learning.

Analyzing the Results:
The number of Input Neurons are **9**, number of Hidden neurons are **9**, number of Output Neurons are **3**, Learning rate is 0.3 and Momentum is **0.9**.



From the above results we can conclude that with the decrease in the number of Training Sets and increase in the Testing Sets, there will result in more errors i.e. the algorithm requires more training examples to classify the given set of records.

4. CONCLUSION

In this study, we have seen how the Back Propagation Algorithm performs with and without Thresholds (bias). In every case we came across, the algorithm with Thresholds performed more efficiently when compared to algorithm with no Thresholds. We also analyzed in many cases how to improve the network architecture by changing the values of the parameters. We observed how the algorithm performs when we vary sizes of Training and Testing Sets.

Thus, it is practically very difficult to determine a good network topology just from the number of inputs and outputs. It also depends on the number of training examples. The requirement of the hidden neurons also depends on the type of the dataset samples. Unnecessary increase in the number of hidden neurons causes over-fitting problem. So, performing some observations on the training data is essential for the design of an efficient Neural Architecture.

REFERENCES

[1]. Breast Cancer Wisconsin (Original) Dataset - https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

[2]. Iris Data Set:https://archive.ics.uci.edu/ml/datasets/Iris

[3]. T. Lalis, B. D. Gerardo and Y. Byun (**2014**). "An Adaptive Stopping Criterion for Backpropagation Learning in Feedforward Neural Network". International Journal of Multimedia and Ubiquitous Engineering Vol.9, No. 8, pp. 149-156

[4]. JeffHeaton (**2005**). "Programming Neural Networks in Java". Heaton Research

[5]. Saurabh Karsoliya (**2012**). "Approximating Number of Hidden layer neurons in MultipleHidden Layer BPNN Architecture". International Journal of Engineering Trends and Technology-Volume3 Issue6

[6]. Tom M. Mitchell (**1997**). "Machine Learning". McGraw Hill

[7]. Wouter F. Schmidt, Martin A. Kraaijveld and Robert P.W. Duin (**1992**). "Feed Forward Neural Networks With Random Weights"- Proceedings. 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems

[8]. Wine Data Set - https://archive.ics.uci.edu/ml/datasets/Wine

AUTHORS

N.V.Saiteja Reddy is currently pursuing his B.Tech in the department of Computer Science and Engineering, Institute of Technology, GITAM University, Visakhapatnam, Andhra Pradesh. His area of interest includes Data Structures, Design and Analysis of Algorithms and has passion to work with Machine Learning Techniques and Neural Networks.

T.Srikanth is presently working as Associate Professor in the department of Computer Science and Engineering, Institute of Technology, GITAM University, Visakhapatnam, Andhra Pradesh, India. His areas of interest include Machine learning, Artificial Intelligence, Data Mining, and Softcomputing.