An Adaptive Neural Network for Email Spam Classification

Jitendra Kumar
Centre for Develoment Of Advanced Computing, NIT Trichy
Bengaluru, India
jitendra@cdac.in

A. Santhanavijayan
National Institute of Technology
Trichy, India
vijayana@nitt.edu

Balaji Rajendran
Centre for Develoment Of Advanced Computing
Bengaluru, India
balaji@cdac.in

B. S. Bindhumadhava
Centre for Develoment Of Advanced Computing
Bengaluru, India
bindhu@cdac.in

***Abstract*— In this paper, we propose an adaptive learning rate algorithm for gradient descent method used in Artificial Neural Network (ANN). The proposed adaptive learning rate neural network algorithm makes use of an initial assigned learning rate and after a set of backpropagation rounds the learning rate gets revised to a learning rate which is equal to minimum learning rate plus average of minimum learning rate and maximum learning rate (these parameters are updated in each recursive calls). During the next recursive call of the algorithm, the revised learning rate is used, if the average total error decreases, or the previous learning rate is continued. This is done until the desired learning rate is achieved which results in optimal weight parameters with high classification accuracy. We have compared the proposed algorithms with existing heuristics approaches and found that the convergence rate is to be faster and the proposed algorithm returns the optimized weights with higher classification accuracy. We have also tested the effectiveness of the algorithm on spam email classification using multilayer neural networks and found that it performs better than the existing approach and less prone to overfitting problems. The proposed algorithm results in 99.12% accuracy for the email classification.**

Keywords*— *Email Spam Classification, Gradient Descent Algorithm, Learning Rate, Machine Learning, MLP (Multi-layer perceptron), Neural Network

I. INTRODUCTION

The Backpropagation (BP) has been one of the most widely used algorithm for multi-layer feedforward artificial neural networks. It has gained popularity mostly of its successful application in many real-world problems and its simplicity. The two most widely cited shortcomings of BP are (1) its slow convergence or no convergence (2) and the possibility of getting stuck at local minimum solutions. Multi-Layer Perceptron (MLP) is a very popular and widely used technique in machine learning. The learning is a very attractive and useful ability of ANN which makes it very useful in solving problems which emulated human brain ability. The backpropagation is the techniques used in ANN for arriving suitable weights (theta parameters), the backpropagation is done repetitively after updating the theta parameter each time using the gradient descent algorithm. The gradient descent algorithm is as follows:

$$\begin{aligned} &Repeat\ \{ \\ &\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &\} \end{aligned}$$

Fig. 1. Gradient descent algorithm

The parameter α (learning rate parameter), $J(\theta)$ is the total cost of the MLP(multi layer perceptron) and $\frac{\partial}{\partial \theta_j}$, the gradient of the error that can be calculated by backpropagation(BP) method. A contour plot is a graph that contains many contour lines. A contour line is a line along which the cost value would be same. An example of a contour graph with two parameters θ_0 and θ_1 is right below.

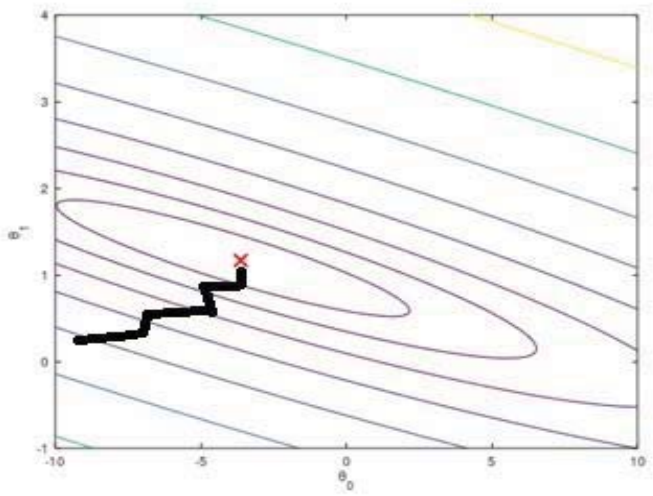

Fig. 2. Contour Graph of the cost function

The gradient descent algorithm iteratively tries to minimize the error $J(\theta)$ by updating its parameters and reaches to the optimum point as marked with a cross in red in Fig. 2. It takes passes through a zigzag path and eventually reaches the optimal value of the parameters where the cost function is minimum. The process is time-consuming as it takes a lot of computing power. Although its performance can be improved by doing feature scaling still it takes a lot of iteration before reaching to the optimal point.

978-1-7281-3807-7/19/$31.00 ©2019 IEEE

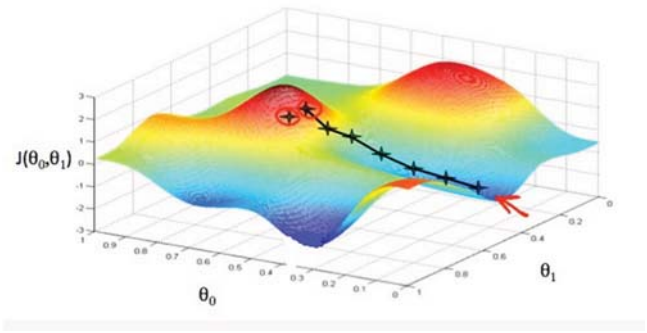

Fig. 3. Cost function graph based on θ_0 and θ_1

Imagine in Fig.3 that hypothesis function of MLP only depends on θ_0 and θ_1, hence we would get cost function as a function of parameters θ_0 and θ_1, where θ_0 is put on x axis, θ_1 is put along the y axis and the cost function on vertical z axis. The points on the graph would represent the cost function of hypothesis with the parameters θ_0 and θ_1 .The objective of the gradient descent graph would be achieved when our cost function is very bottom of the pits of the graph, that is its value is minimum. The red arrow in the graph shows one such point. There can be many local minima and one global minimum. In gradient descent algorithm we take the derivative of our cost function (a tangential line on the graph at a given point). The slope of the graph at that point is the derivative at that point and it will give the direction to move towards also. The main strategy in the gradient descent algorithm is to step down the cost function in the tangential direction towards steepest descent. The amount of descent is determined by a parameter α which is called the learning parameter. The distance between the each star in Fig. 3 is determined by α parameter, smaller the value smaller will the descent step and direction will be determined by the partial derivative of $J(\theta_1, \theta_2)$ .The problem with gradient descent error minimization is that its momentum depends on parameter α and value of this parameter should be arrived after the trial and error method. There are various methods devised to predict the initial value of the learning rate some of which includes, finding the accuracy of the model by varying learning rate over a range and then train a small neural network to get the value of alpha which has given the highest accuracy. This is a bit time consuming and sometimes suffers with the problems like if we chose very small value of learning rate then it may take more time and iterations to reach to the local optimum value, on the other hand, if we take very large value then our algorithm might overshoot the local optimum and never finds the minimum value. So, all the algorithms fix the learning rate then train their values. In this paper, we have designed an algorithm to find a better α (learning rate) and weights (theta parameters) that might correspond to global minimum, so that the cost function would be minimized and the learning algorithm would return better accuracy.

II. Related Work

One of the major drawbacks of backpropagation algorithm is its possibility of getting stuck in local minimum solution. Authors of the paper [1] have suggested using a small change in output e.g. ΔO using Taylor series expansion of output function O instead

of $\alpha * \frac{\partial}{\partial \theta_j}$ in standard backpropagation and claimed that this approach leads to faster convergence and leads to the global optimal solution. A gradient reuse algorithm [2], which suggests use of the gradient in the backpropagation several times until the resulting weight updates no longer lead to a reduction in error. Negnevitsky et. al., [3] describe the usage of fuzzy control of backpropagation learning, it suggests the network outputs and errors are calculated using the initial learning rate parameter. If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated. However, if the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05). Behera et. al., [4] propose learning algorithm based on Lyapunov stability theory and fixed learning rate in BP algorithm is replaced by an adaptive learning rate to avoid the problem of getting stuck at local minima. They also showed that by controlling the rate of weight update, it is possible to drive out of a local minimum by opposing the change in the direction of the weight update. Daiki Ito et. al., in their work [5] propose a new learning rate adaptation mechanism based on sign variation of the gradient to a mini-batch type learning algorithm with the gradient normalization. The authors further suggested that adjustment of learning rate based on the moving average of the variation of sign and the normalization of each component of the gradient by factor convoluting recent magnitude. Ventresca et.al., [6] in their work, propose a method based on the use of opposite networks (via opposite transfer functions) to improve learning time and accuracy for very large-scale gradient-based training of multi-layer networks. The underlying concept behind opposite transfer function is to provide means for dynamically altering the network structure such that knowledge stored in connection weights is retained but the input/output mapping differs. Wangpeng An et. al., proposed to use the learning rate that would vary in a sine wave fashion and in each epoch the learning rate is scheduled to scan a range of learning rates, while the maximum value of sine wave would decay exponentially along with training epochs and their experiments revealed that their algorithms produced a better results as it requires 2 to 5 times less epoch number than the then-used learning rate schedule schemes. Li et. al., have used Taylor formula and ignored the infinitesimal items to arrive the formula for change in error. They have further proposed the learning rate that can adapt the changes of the total quadratic error E and the gradient of the error curve surface. Yai Zhang [9] proposed an innovative

approach for improving the convergence rate of backpropagation network and suggested updating the learning rate parameter for each individual weight before each weight is updated. Bhattacharya et. al., [11] have elucidated self-adaption learning rate for the backpropagation algorithm. The self-adaptive learning rates are based on a combination of two updating rules - one for updating the connection weights and the other for updating the learning rate. They further suggested modification in the work of R. A. Jacobs [10] and proposed four heuristics to achieve faster rates of convergence in backpropagation learning algorithm. They proposed to confirm to Jacob [10] heuristics with additional clauses like modification of the learning rates at any time should be entirely based on the shape of the error surface at the present position, overall convergence performance of the algorithm should not necessarily depend on the choice of those parameter values, the values of the learning rates should not be allowed to increase indefinitely so that the weight values might explode.

III. Neural Network For Email Classification

Mostly spam is detected based on the words, hyperlinks, sender header, subject, images that are present in the emails. We have created a neural network for testing our algorithm and we trained the neural network with the dataset which contains equal number of spam and non-spam emails. The backpropagation is a method in the Neural Network in which gradient is calculated and is used in the calculation of the weights of the neural network. It is used to train the neural network; the term network means that which has more than one hidden layer in it.

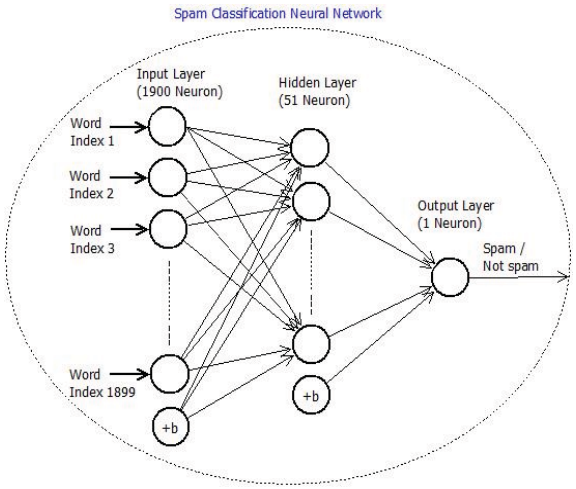

Fig. 4. Neural Network for Spam Classification

The backpropagation algorithm is used in the artificial neural network and it feeds already classified data or labeled data in a feed-forward multilayer artificial neural network. Further, when each sample label data is fed to the network the out response is evaluated and examined. The output result is then compared with the already known desired output result, the difference between the feed-forward output and known desired out is called the error value. This error would be backpropagated after adjusting the connection weights. This process is repeated a no. of times till the error value is minimized. The backpropagation algorithm is based on the Windrow-Hoff delta learning rule in which the weight adjustment is done through the mean square error of the output response to the sample input [2]. We have a dictionary of all the English words. Although if all the English grammar words were to be taken the text size would grow to a very large number so we use the Porter Stemmer to trim most of the words which have the same meaning. So, we need not consider all the words in English grammar but just those present in the dictionary created by porter stemmer. The words are converted to vector so that we can put them to train in the neural network. Then training is done according to the backpropagation algorithm. The backpropagation algorithm returns the value of errors of each layer weights. This error is further passed to the gradient descent algorithm to find the optimum value of weights.

IV. Algorithms for Email Classification

The backpropagation learning algorithm was introduced for feed-forward neural network training and in the backpropagation, the neurons or processing units are arranged in the layered architecture. The input layer simply passes feature vector X and hidden layers and output layer neurons are known as computation units. A set of input vectors are given as inputs to the neural network and their output are computed. The error is calculated as the mean of squared difference of actual output and the computed output of the ANN. The error computed from the output layer then backpropagated to the hidden layers of the networks and each the weights are modified based on their contribution to the actual error.

A. Data Preprocessing

Emails might contain URLs, an email addresses, numbers, and dollar amounts etc. While many emails would contain similar types of entities (e.g., numbers, other URLs, or other email addresses), the specific entities (e.g., the specific URL or specific dollar amount) will be different in almost every email. Therefore, stemming is often employed in processing emails is to "normalize" these values, so that all URLs are treated the same, all numbers are treated the same, etc. For example, we could replace each URL in the email with the unique string "httpaddr" to indicate that a URL was present. This has the effect of letting the spam classifier make a classification decision based on whether any URL was present, rather than whether a specific URL was present. This typically improves the performance of a spam classifier, since spammers often randomize the URLs, and thus the odds of seeing any particular URL again in a new piece of spam is very small. Email preprocessing and normalization steps:

- Lower-casing: The entire email is converted into lower case, so that capitalization is ignored (e.g., IndIcaTE is treated the same as Indicate)
- Stripping HTML: All HTML tags are removed from the emails. Many emails often come with HTML formatting; we remove all the
- HTML tags, so that only the content remains

- Normalizing URLs: All URLs are replaced with the text "httpaddr".
- Normalizing Email Addresses: All email addresses are replaced with the text "emailaddr".
- Normalizing Numbers: All numbers are replaced with the text "number".
- Normalizing Dollars: All dollar signs ($) are replaced with the text "dollar"
- Word Stemming: Words are reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Sometimes, the Stemmer actually strips off additional characters from the end, so "include", "includes", "included", and "including" are all replaced with "includ".
- Removal of non-words: Non-words and punctuation have been removed. All white spaces (tabs, newlines, spaces) have all been trimmed to a single space character.

B. Feature Selection and Extraction

After the preprocessing steps, we would get a list of words for each email. The next step is to select the features for the classifier, i.e. the classifier would use which set of words and which set of words it would leave out for the training. We have chosen a set of most frequent words from our set of training emails set as features and this we consider as a vocabulary list. We have left out those words which are occurring in very few emails of our training set, otherwise, this might cause the model to overfit for the given training set. The vocabulary list of words should be selected by choosing those would which occurs at least 100 in the spam corpus. The vocabulary list was selected in a list of 1899 words but in practice, a vocabulary list size should be 10,000 to 50,000 for better accuracy. These 1899 words of the vocabulary list are the features for our classifier.

C. Backpropagation Neural Network(BPNN) Algorithm

We created a three-layer neural network where the input layer consists of 1899 features plus one bias and the hidden layer consists of 49 plus one bias. The out layer consists of one neuron.

m1(length of hidden layer) =50
n1(length of input layer) =1900

For training, the dataset is based on a subset of the SpamAssassin Public Corpus.3 [12]. The file 'spam_features.mat' contains email data set for the training and is based on a subset of the SpamAssassin Public Corpus.3 [12]. The 'spam_features.mat' [13] is a matrix of 4000 spam emails data with 1899 features. Further, we tokenized the content of emails and normalized it using python port stemming and compared it with the list of the word with respect to the keywords present in the keyword file, which maps every word of the dictionary to one keyword. Then, this is used as an input for prediction. The algorithm for splitting dataset is as follows:

```
1. mat =loadmat('spam_features.mat',)
2. X = mat['X'] # Features Matrix of spam data
3. y = mat['y'] # Class label matrix of spam data
4. /* “X_train” is a dataset for training and “y_train” is
   its corresponding class labels and similarly “X_test”
   is a dataset for testing and  “y_test” its
   corresponding class labels
   */
5. X_train, X_test, y_train, y_test = train_test_split(X,
   y, test_size=0.2)
```

The nnCostFunction() function used below, returns total error and gradient for the neural network. The BPNN algorithm is as follows:

```
/* “grad” is gradient matrix, “input_layer_size”
size of input layer, “hidden_layer_size” is size
of “hidden layer”, “num_label” is label of output,
“theta1” and “theta2” are the weight matrix
between input layer to hidden layer and hidden
layer to output layer respectively, “lamda” is a
regularization parameter.
*/
1. function BPNN (theta1, theta2, alpha,
   iterations)
2. {
3. for it in range (iterations)
4. {
5. grad, error =
   nnCostFunction(theta1,theta2,input_layer_size,
   hidden_layer_size,num_labels,X_train,y_train,l
   amda)
6. /* “bpnn_error” will hold total error of BPPN
   algorithm for the given iterations*/
7. bpnn_error += error
8. Theta1_gradient =grad[0][0]
9. Theta2_gradient = grad[1][0]
10. theta1 = theta1 - alpha*Theta1_gradient
11. theta2 = theta2 - alpha*Theta2_gradient
12. }
13. THETA = [[theta1],[theta2]]
14. avg_error = bpnn_error / iterations
15. return THETA , avg_error
16. }
```

D. Adaptive Neural Network Algorithm

There are quite a few techniques to arrive at the initial value of the learning parameter, some of them included train a neural network with small data sets on randomly chosen learning rates and try to find out the learning rate which has given the best accuracy. This is a bit time consuming and sometimes suffers from bottlenecks like selection of very small value of learning rate and it might result in more time and iterations to reach to the local optimum value, on the other hand, if very large value has been selected then the algorithm might overshoot the local optimum and never finds the minimum value. So, all the algorithms fix the learning rate then train their values. What we research in this is to start with a genuine learning rate and revise it after some iteration, what this will do is help us to reach the local optimum faster as compared to the fixed learning rate. If the error is increased by increasing the value of the learning rate, then we revert to the previous value and continue with it till completion. Although this might help in the optimization of time and this method also helps in reaching the best results, which may not be achieved by fixing the learning rate, varying the learning rate during training gives better results than fixing the value to the same learning rate. The main concern which is the time taken by this algorithm might increase as we are varying the learning rate. Since the total time to train a gradient descent algorithm also includes finding the optimum value of alpha which in fact takes most of the time of designer, our algorithm will initialize the alpha, train it then save the progress, then again train it by changing the alpha to reach the optimum, this method if done automatically through following algorithm will give best result and help training the neural network better, to give high accuracy and precision. One another benefit of using this algorithm is that if our algorithm got stuck in the local optimum then this will help it reach the optimum that might correspond to a global optimum.

E. Pseudo code for Adaptive Neural Network

```
1. function Adaptive_Neural_Network(THETA,
   min, max, initial_error, alpha, total_iterations)
2. {
3. # accuracy_test() returns accuracy score
4. ac_score = accuracy_test(THETA)
5. if (ac_score > 99.5)
6. {
7. return THETA, alpha
8. }
9. n_min=min+(max-min)/2
10. n_max = max *2
11. alpha_new=n_min+(n_max-n_min)/2
12. THETA, ErrorTotal=BPNN(THETA, alpha_new
    ,test_iterations)
13. remaining_iterations=total_iterations-
    test_iterations
14. if(ErrorTotal-initial_error<0)
15. {
16. Adaptive_Neural_Network(THETA, n_min,
    n_max, ErrorTotal, alpha_new,
    remaining_iterations )
17. }
18. else
19. {
20. THETA,ErrorTotal=BPNN(THETA, alpha
    ,remaining_iterations)
21. return THETA, alpha
22. }
23. }
```

Following are initialization for test run the algorithm for which output is given in Section [4D].

```
1. /*Initialize the alpha with default value. Also
   initialize min, max and initial_error value. */
2. input_layer_size = 1899 #input layer size
3. hidden_layer_size = 50 #hidden layer size
4. num_labels = 1    #no of output label
5. lamda=.01 #Regularization parameter
6. alpha = 2 # learning rate
7. min=0
8. max=2
9. total_iteration=200
10. test_iterations=20
```

F. Results Analysis

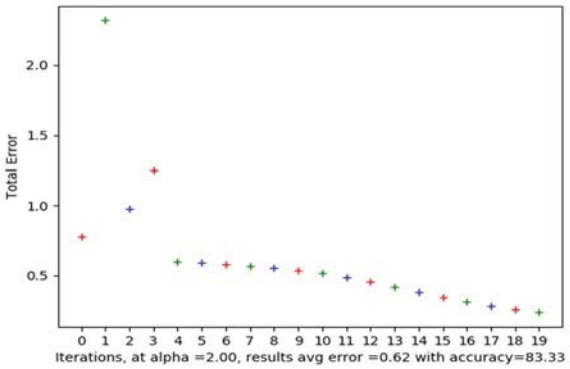

Fig. 5. Error vs. Iteration graph for first call of BPNN

The Fig.5 shows the graph for error vs. no. of iterations for alpha=2 for BPNN initially.

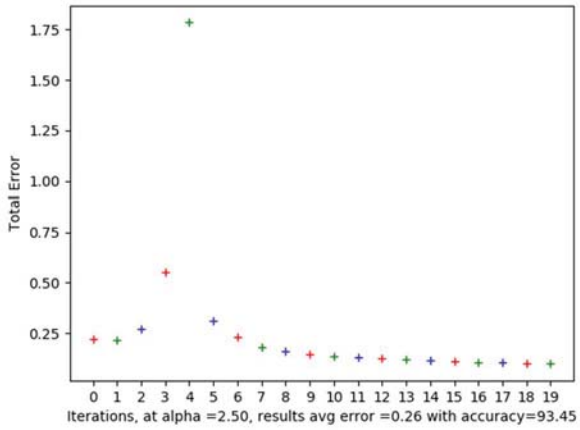

Fig. 6. Second call of BPNN and its Error vs. Iteration graph

The Fig.6 shows the graph for error vs. no. of iterations graph of BPNN second call. We can observe that average error has decreased to 0.26 and classification accuracy has increased to 93.45 compared to its previous call.

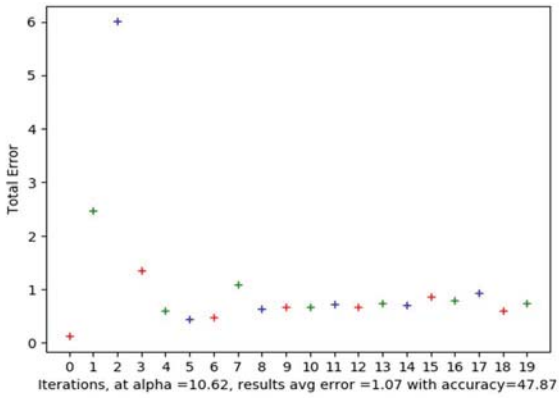

Fig. 7. Fourth call of BPNN and its Error vs. Iteration graph

The Fig.7 shows the graph for error vs. no. of iterations graph of fourth call of BPNN algorithm. We can observe that average error has increased to 1.07 and classification accuracy decreased to 47.87.

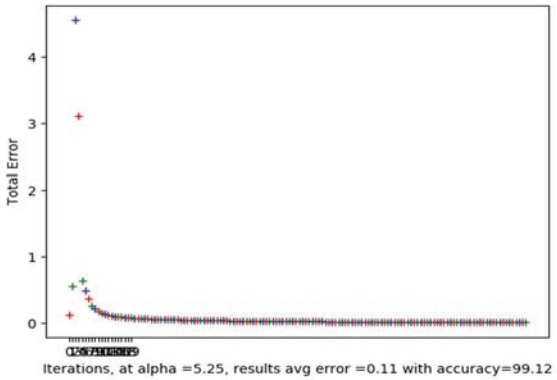

Fig. 8. Fifth call of BPNN and its error vs. iteration graph

Fig. 8 shows the graph for error vs. no. of iterations graph of call of BPNN for the fifth time. As in the previous call of BPNN average error increased and the proposed algorithm continues the next call of BPNN with old learning parameter i.e. 5.25 till completion remaining 120 iterations. We can also observe that average error has decreased to 0.11 and classification accuracy has increased to 99.12. The figures shown above depict the behavior of change in gradient descent with change in the learning rate. Initially, during training, we chose the smaller alpha. If we take the value of the learning rate very high, then the algorithm might overshoot the optimal value and the neural network might not result in better classification accuracy. But as we continue with this lower rate the graph tends to become steeper and doesn't change much, it tends to become a straight line. At this point, our algorithm changes the value of alpha with a binary search method.

alpha = min + (max-min) / 2

The above-mentioned execution of algorithm starts with alpha = 2 then after some iterations, the algorithm tries to change the alpha with value 2.5 and then changes to 5.25 as error decreased at alpha=10.62 the error increased concerning the previous call of BPNN and the algorithm continues with alpha=5.25 for the rest of the iterations. At last the proposed algorithm produced 99.12 % classification accuracy. In short, we were not able to pick large values of alpha initially for the gradient descent algorithm but after some iteration, we were able to pick large values of the alpha (learning rate) because choosing small values, or same value of alpha over the full period of training is not producing fruitful results. Varying the learning ensures us to reach an optimal solution and help our algorithm not to stuck in local optimum. This algorithm might help us to reach the global optimum by faster means and decrease the error to a very great assent. We compared different models with our dataset and calculated the model accuracy, which is as follows. The above comparison is done with scikit-learn Machine Learning Library in Python.

Model	Accuracy
Adaptive Neural Network (proposed in the paper, with 200 iterations and initial learning rate=2)	99.12 %
Logistic Regression	98.25%
SVM	96 %
MLP Classifier(with 200 iterations and learning rate=2)	98 %

A better result was produced with the Adaptive Neural Network designed in this paper with the total number of iterations 200 and initial learning rate 2 which later got increased to 5.25 for the last 120 iterations. We also observed in our experiment that the adaptive neural network has better convergence and less prone to overfit and requires less no. of hidden layers.

V. Conclusion

In this paper, we have proposed a new algorithm for deriving the optimal learning rates for neural network and tested the proposed algorithm by creating a three-layer neural network, which we further trained with email data for email spam classification. We have observed that the proposed algorithm has a faster convergence to the optimal classification with high accuracy and low error. The normal heuristic approach is to try with different values of learning rates and find out for which learning rate the classification accuracy is higher, in such an approach the user must manually try many learning rates which time is consuming. The proposed algorithm adaptively changes the best-suited learning rate during its recursive calls and returns the neural network optimized weights which results in high accuracy. We also observed that the proposed Adaptive Neural Network based on adaptive learning rate performs better and have fast convergence and less prone to overfitting problem. In the future, we will extend this algorithm to find the global minima of the cost function such that we could be able to find the best-optimized weights for the neural network.

References

[1] Zaiyong Tang,Gary J Koehleri, “A Convergent Neural Network Learning Algorithm," in proceedings of IJCNN International Joint Conference on Neural Networks, June 1992.

[2] D. R. Hush , J.M. Salas," Improving the learning rate of back-propagation with the gradient reuse algorithm ," IEEE 1988 International Conference on Neural Networks during 24-27 July 1988, San Diego, CA, USA

[3] M. Negnevitsky, M. Ringrose, "Accelerated learning in multi-layer neural networks," ICONIP'99. ANZIIS'99 & ANNES'99 & ACNN'99. 6th International Conference on Neural Information Processing. Proceedings (Cat. No.99EX378)

[4] L. Behera, S. Kumar, A. Patnaik," On Adaptive Learning Rate That Guarantees Convergence in Feedforward Networks" IEEE Transactions on Neural Networks (Volume: 17 , Issue: 5 , Sept. 2006)

[5] Daiki Ito,Takashi Okamoto,Seiichi Koakutsu,"A learning algorithm with a gradient normalization and a learning rate adaptation for the mini-batch type learning" 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE),19-22 Sept. 2017

[6] M. Ventresca,H. R. Tizhoosh,"Improving Gradient-Based Learning Algorithms for Large Scale Feedforward Networks," International Joint Conference on Neural Networks,14-19 June 2009

[7] Wangpeng An,Haoqian Wang,Yulun Zhang,Qionghai Dai,"Exponential Decay Sine Wave Learning Rate for Fast Deep Neural Network Training," IEEE Visual Communications and Image Processing (VCIP),10-13 Dec. 2017

[8] Yong Li,Yang Fu,Hui Li,Si-Wen Zhang,"The Improved Training Algorithm of Back Propagation Neural Network with Selfadaptive Learning Rate," International Conference on Computational Intelligence and Natural Computing, 6-7 June 2009

[9] Yai Zhang, "Updating learning rates for backpropagation network," Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan),25-29 Oct. 1993

[10] R. A. Jacobs, “Increased rates of convergence through learning rate adaptation,” Neural Networks,vol. 1, pp. 295-307, 1988.

[11] U. Bhattacharya, S.K. Parui, "Self-adaptive learning rates in backpropagation algorithm improve its function approximation performance," Proceedings of ICNN'95 - International Conference on Neural Networks, 27 Nov.-1 Dec. 1995

[12] http://spamassassin.apache.org/old/publiccorpus, [Online]. Available: http://spamassassin.apache.org/old/publiccorpus [Accessed: 26- April- 2019].

[13] Andrew Ng, 'Machine Learning Course, Stanford University, SVM chapter' [Online]. Available: https://www.coursera.org [Accessed: 26- April- 2019].