Comparison of Back Propagation and Resilient Propagation Algorithm for Spam Classification

Navneel Prasad, Rajeshni Singh
UXC Eclipse Ltd.,
Suva, Fiji Islands
e-mail: NPrasad@uxceclipse.com
RSingh-Prasad@uxceclipse.com

Sunil Pranit Lal
School of Computing, Information, and Mathematical Sciences
University of the South Pacific
Suva, Fiji Islands
e-mail: lal.sunil@ieee.org

***Abstract*— In this paper we compare the performance of back propagation and resilient propagation algorithms in training neural networks for spam classification. Back propagation algorithm is known to have issues such as slow convergence, and stagnation of neural network weights around local optima. Researchers have proposed resilient propagation as an alternative. Resilient propagation and back propagation are very much similar except for the weight update routine. Resilient propagation does not take into account the value of the partial derivative (error gradient), but rather considers only the sign of the error gradient to indicate the direction of the weight update. We show that resilient propagation yields faster convergence and higher accuracy on the UCI Spambase dataset.**

Keywords- Neural Networks; Back Propagation; Resilient Propagation; Spam Classification

I. INTRODUCTION

In a relatively short timeframe, Internet has become irrevocably and deeply entrenched in our modern society primarily due to the power of its communication substrate linking people and organizations around the globe. Email has become one of the most reliable and economical forms of communication as the number of Internet users has increased, and individuals and organizations rely more and more on the emails to communicate and share information and knowledge. The number of emails has been increasing all the time; however, this explosive growth comes with a variety of problems. The number of unsolicited commercial emails or spam emails has been increasing dramatically over the last few years.

To overcome this issue, spam filters are introduced. One of the methods of filtering spam is using neural networks to intelligently classify an email as a spam or a ham. In order to use a Neural Network it has to be trained first to get the optimal weights.

Previous researches have shown that neural network can achieve very accurate results [1]. On the other hand, there are disadvantages to the method also. The main disadvantage of neural network is that it requires considerable time for parameter selection and network training. If a poor choice is made for the learning rate, training momentum or delta values then training will not be as successful.

A hybrid method that combines neural network and genetic algorithms for feature selection is presented for robust detection of spam [2]. Cui et al. proposed a model based on the neural network to classify personal emails, and the use of principal component analysis (PCA) as a pre-processor of neural network to reduce the data in terms of both dimensionality and size [3]. These studies show that neural network can be successfully used for automated email classification and spam filtering. Back propagation (BP) neural network is the most popular among all the neural network applications. It has the advantages of yielding high classification accuracy. However, practical applications are difficult to be satisfied because of the problems of slow learning and the likelihood of being trapped into a local minimum especially when the size of the network is large. These problems are due to the fact that the learning of BP neural network is mechanical and elementary. Many researchers have worked to overcome these problems, especially the local convergence [4].

Multilayer networks typically use sigmoid transfer functions in the hidden layers. These functions are often called "squashing" functions, because they compress an infinite input range into a finite output range. Sigmoid functions are characterized by the fact that their slopes approach zero, as the input gets large. This causes a problem when you use steepest descent (gradient decent/ back propagation) to train a multilayer network with sigmoid functions, because the gradient can have a very small magnitude and, therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

In this paper we use resilient propagation to overcome the drawbacks of back propagation learning. Back propagation as mentioned previously is slow at converging due to the gradients having a very small magnitude, which causes small changes in weights. The purpose of the resilient propagation (RPROP) training algorithm is to eliminate the

harmful effects of these magnitudes of the partial derivatives. Only the sign of the derivative can determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. Another most difficult aspect of the back propagation learning was picking the correct training parameters. Resilient propagation does have training parameters, but it is extremely rare that they need to be changed from their default values. This makes resilient propagation a very easy way to use a training algorithm. It also has the nice property that it requires only a modest increase in memory requirements. Additionally, resilient propagation is considerably more efficient than back propagation.

II. Resilient Propagation

A. Description

Resilient propagation, in short, RPROP is one of the fastest training algorithms available. The RPROP algorithm just refers to the direction of the gradient. It is a supervised learning method. It works similarly to back propagation, except that the weight updates is done in a different manner.

In back propagation the change in weight is calculated with the magnitude of the partial derivative:

$$\Delta w_{ij}(t) = \alpha \times x_i(t) \times \delta_j(t) \tag{1}$$

where α is the learning rate, $x_i(t)$ represents the inputs propagating back to the i^{th} neuron at time step t, and δ is the corresponding error gradient.

Resilient propagation, on the other hand, calculates an individual delta Δ_{ij}, for each connection, which determines the size of the weight update. The following learning rule is applied to calculate delta:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \times \Delta_{ij}^{(t-1)} & , \quad if \quad \frac{\partial E}{\partial w_{ij}}^{(t-1)} \times \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ \eta^- \times \Delta_{ij}^{(t-1)} & , \quad if \quad \frac{\partial E}{\partial w_{ij}}^{(t-1)} \times \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)} & , \quad else \end{cases} \tag{2}$$

$$where \quad 0 < \eta^- < 1 < \eta^+$$

The update-value Δ_{ij} evolves during the learning process based on the sign of the error gradient of the previous iteration, $\frac{\partial E}{\partial w_{ij}}^{(t-1)}$ and the error gradient of the current iteration, $\frac{\partial E^{(t)}}{\partial w_{ij}}$. Every time the partial derivative (error gradient) of the corresponding weight w_{ij} changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update-value Δ_{ij} is decreased by the factor η^-, which is a constant usually with a value of 0.5. If the derivative retains its sign, the update value is slightly increased by the factor η^+ in order to accelerate convergence in shallow regions. η^+, is a constant usually with a value of 1.2. If the derivative is 0 then we do not change the update-value.

Once the update-value is calculated for each weight, the weight-update is then calculated. There are two rules to follow to calculate the weight-update.

The first rule is that if the current derivative and the previous derivative retain their signs then Equation 3 is used to calculate the weight-update.

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \quad if \quad \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)} & , \quad if \quad \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0 & , \quad else \end{cases} \tag{3}$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

If the current derivative is a positive value meaning the previous value was also a positive value (increasing error), then the weight is decreased by the update value. If the current derivative is negative value meaning the previous value was also a negative value (decreasing error) then the weight is increased by the update value.

The second rule is that if the current derivative and the previous derivative have changed their signs i.e. there was a big step taken then chances are that a minimum was missed. To avoid such big jumps, the weights need to be reverted to the previous state.

$$\Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t-1)}, \quad if \quad \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \tag{4}$$

If the weight was reverted then the previous derivative needs to also be changed, otherwise when the weight is updated again then it will reapply the same changes, repeating this scenario. Therefore, the previous derivative $\frac{\partial E}{\partial w_{ij}}^{(t-1)}$ is set to 0.

B. Parameters

Resilient propagation uses the following parameters, Δ_0, Δ_{max}, Δ_{min}, η^+ and η^-.

Δ_0 is the initial value of the delta update-value Δ_{ij}. This value is set to 0.1. Martin Riedmiller, has proved that the choice of this parameter is not critical at all. Even for much larger or smaller of this value, fast convergence is achieved.

Δ_{max}, is the maximum value a delta update, Δ_{ij}, can have. This value is set to 50. Δ_{min}, is the minimum value a delta

update, Δ_{ij}, can have. This is set to a very low positive value, $1e^{-6}$.

The η^- was given a value of 0.5. η^- value is used as a reducing factor when the derivative has changed sign. This is usually a big jump, probably missing the minimum. Since it is not known by how much the minimum was missed, it is a good guess to halve the update-value by using $\eta^- = 0.5$. On the other hand, η^+ has to be large enough for fast growth. However, if it is too large a value then learning process can be disturbed. η^+ was chosen as 1.2. It is stated in [5] that experiments were done to alter this value to see performance but changing the value did not make any difference to the convergence time, therefore it is advised to keep these constants with their default values.

C. Algorithm

Putting together the concepts discussed in the previous section, results in the following algorithm.

for all weights and biases

{

if $\left(\frac{\partial E}{\partial w_{ij}}(t-1) \times \frac{\partial E}{\partial w_{ij}}(t) > 0 \right)$ *then*

{

$$\Delta_{ij}(t) = \min(\eta^+ \times \Delta_{ij}(t-1), \Delta_{\max})$$

$$\Delta w_{ij}(t) = - sign\left(\frac{\partial E}{\partial w_{ij}}(t) \right) \times \Delta_{ij}(t)$$

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

}

elseif $\left(\frac{\partial E}{\partial w_{ij}}(t-1) \times \frac{\partial E}{\partial w_{ij}}(t) < 0 \right)$ *then*

{

$$\Delta_{ij}(t) = \max(\eta^- \times \Delta_{ij}(t-1), \Delta_{\min})$$

$$\Delta w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t-1)$$

$$\frac{\partial E}{\partial w_{ij}}(t) = 0$$

}

elseif $\left(\frac{\partial E}{\partial w_{ij}}(t-1) \times \frac{\partial E}{\partial w_{ij}}(t) = 0 \right)$ *then*

{

$$\Delta w_{ij}(t) = - sign\left(\frac{\partial E}{\partial w_{ij}}(t) \right) \times \Delta_{ij}(t)$$

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

}

}

D. Discussion

Both back propagation and resilient propagation technique work in similar manner. There are three distinct steps:

1. Perform a regular feed forward pass
2. Process the levels backwards and determine the error gradients at each level
3. Apply the changes to the weights

First, a regular feed forward pass is performed. The output from each level is kept so that the error for each level can be evaluated independently. Second, the errors are calculated at each level, and the derivatives of each of the activation functions are used to calculate gradient descents. These gradients will be used in the third step.

The third step is where the two algorithms vary. Back propagation simply takes the gradient descents and scales them by a learning rate. The scaled gradient descents are then directly applied to the weights and thresholds. RPROP keeps an individual delta value for every weight and only uses the sign of the gradient descent to increase or decrease the delta amounts. The delta amounts are then applied to the weights.

III. Results

A. Data Preparation

The dataset used in this experiment is from the UCI Machine Learning Repository [1] and consists of 4601 instances. The class distribution comprised of 1813 Spam messages (39.4%) and 2788 Ham messages (60.6%). The attribute information of the dataset is provided in Table I.

TABLE I

ATTRIBUTES OF SPAMBASE DATASET

Num of Attributes	Data type	Range	Description
48	Real	[0,100]	Word frequency expressed as a percentage
6	Real	[0,100]	Char frequency expressed as a percentage
1	Real	[1,…]	Average length of uninterrupted sequences of capital letters
1	Integer	[1,…]	Average length of uninterrupted sequences of capital letters
1	Integer	[1,…]	Total number of capital letters in the e-mail
1	Nominal	{0,1}	Class attribute {0=Ham, 1= Spam}
58 Total Attributes			

[1] *http://archive.ics.uci.edu/ml/datasets/Spambase*

The data was normalized in the range [0, 1]. Two dataset were prepared from this single set, training set (80%) and testing set (20%). Each of these sets had a distribution of 60% ham and 40% spam. From the training and testing set, five sets were created by randomly selecting email data from them for training and testing, maintaining the ratio of hams to spams stated above.

B. Testing Methodology

For our study we implemented two learning procedures: back propagation learning algorithm and the resilient propagation algorithm.

The initial network structure contained 57 input neurons at the input layer, 29 neurons in the hidden layer and 1 neuron in the output layer.

To allow a fair comparison between the two algorithms they were tested for 5 different threshold values for a fixed number of epochs (500). The threshold applied to the output neuron, marks the boundary between spam and ham. The accuracy and false positive were obtained for each of the threshold values as an average of five runs per threshold on five different training sets. These values were then compared for the two algorithms.

Furthermore, the better of the two algorithms was chosen and further tested by varying the hidden layers, and the neuron count in the layers.

C. Accuracy and False Positive Comparison

Both, back propagation and resilient propagation algorithm were tested on 5 different thresholds with 5 runs for each threshold. Once this was done the average of the accuracy and false positive was tabulated for comparison (Table II).

As seen in Fig. 1, the resilient propagation was able to converge faster towards better accuracy and false positive rates.

TABLE II
ACCURACY AND FALSE POSITIVE FOR DIFFERENT THRESHOLDS

Threshold	Back propagation		Resilient propagation	
	FP %	Acc %	FP %	Acc %
0.3	40.65	69.14	24.62	78.17
0.4	35.45	71.56	21.97	77.76
0.5	41.29	68.01	26.49	75.32
0.6	39.86	69.73	23.91	75.87
0.7	42.87	67.88	22.87	77.32

Also from the results the threshold that produced an accepted value for the accuracy and false positive was 0.4 in both the cases. Since resilient propagation produced better results it was chosen as our learning algorithm for the neural network.

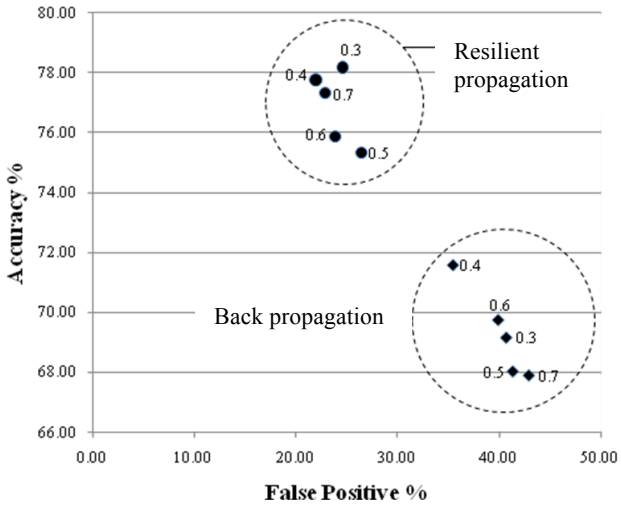

Fig. 1. Distribution of accuracy and false positive for different threshold values {0.3, 0.4, 0.5, 0.6, 0.7}

D. Convergence Comparison

The key design change in the weight update routine of the resilient propagation algorithm has been attributed to better convergence characteristics compared to back propagation algorithm. In order to verify this, both the networks were run multiple times for 500 epochs at the optimal threshold (0.4). Fig. 2 shows the convergence rate of both the propagation algorithms, and confirms that resilient propagation converges much faster than the normal back propagation algorithm.

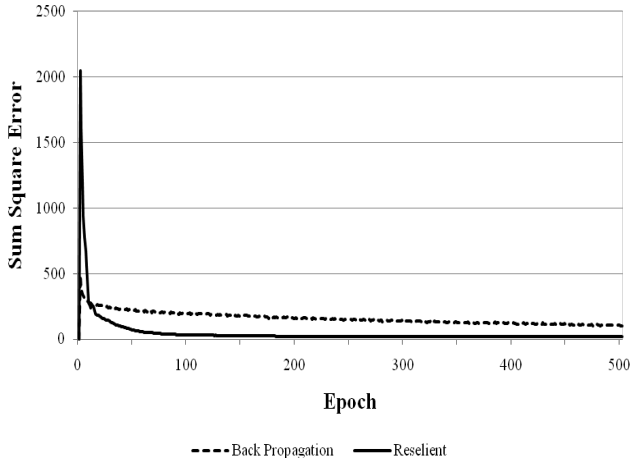

Fig. 2. Sum square error plotted for for back propagation and resilient propagation algorithm.

E. Efficiency Comparison

As resilient propagation only considers the sign of the partial derivative of the delta to update its weights, it is therefore less computationally intensive, which significantly

reduces the convergence time. Fig. 3 shows the comparison of the time it takes to train a neural network for 500 epochs using the two algorithms with the threshold value set to 0.4.

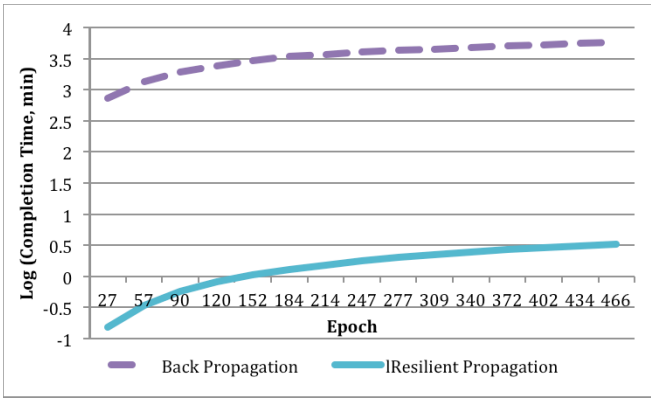

Fig. 3. Time taken to complete training with 500 epochs

F. *Effect of Increasing Hidden Layers on Resilient Propagation Network*

We selected resilient propagation algorithm for further investigation as it yielded better results compared back propagation algorithm. To investigate the effect of the number of hidden layers, the network was tested with 1, 2, 3, and 4 hidden layers. Each hidden layer contained half of the neurons in the previous layer. The network was run for 500 epochs for all the thresholds to obtain the threshold, which provided the highest accuracy and the lowest false positives. The results in Table III, show that still the threshold of 0.4 provided the best results with one hidden layer. A network with more than two hidden layers can start generating arbitrary complex regions in the state space [6] and end up over fitting the parameters.

TABLE III
EFFECT OF CHANGING HIDDEN LAYERS

Thres	# of Hidden Layers							
	1		2		3		4	
	FP %	Acc %	FP %	Acc %	FP %	Acc %	FP %	Acc %
0.4	**20.9**	**77.8**	26.8	75.8	27.3	74.9	23.8	78.7
0.5	25.1	75.3	25.8	75.6	23.2	78.4	24.4	78.1
0.6	22.7	75.9	27.7	74.5	26.7	75.0	23.4	78.8
0.7	21.7	77.3	26.0	75.8	26.5	74.8	23.2	78.9

G. *Effect of Changing Neuron Count*

The hidden layer initially consisted of 29 neurons. To find out the effect of the neuron count on the network, the number of neurons was altered in the hidden layer. The neuron counts tested were 15, 20, 25, 30, 35, 40 and 45. The threshold value of 0.4 was used on a three-layer network, as a single hidden layer gave better results.

Fig 4 and 5 show the effect of changing the neuron count at the hidden layer. From the results it is evident that 40 neurons at the hidden layer yielded the best solution.

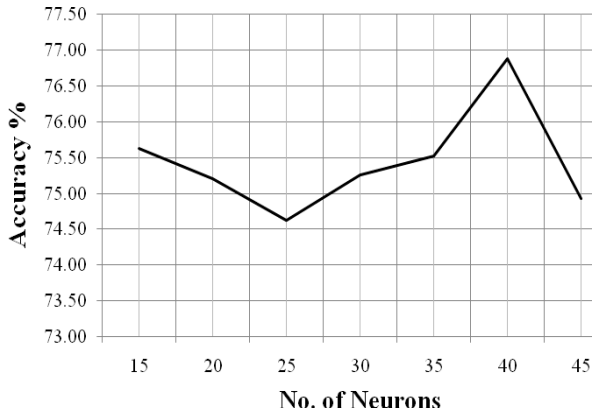

Fig. 4. Accuracy rate against the number of neurons at the hidden layer for a network with single hidden layer trained with resilient propagation.

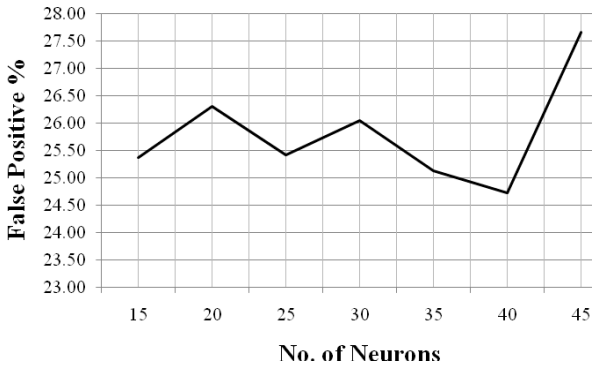

Fig. 5. False positive rate against the number of neurons at the hidden layer for a network with single hidden layer trained with resilient propagation

IV. CONCLUSION

In certain online learning environments, it is sometimes preferably to get good results in a short period of time than to wait a long duration for the best result. Resilient propagation algorithm is similar to back propagation algorithm for neural networks, except for a key difference, which relates to how the weights are updated depending on the sign of the error gradient. Through empirical means, we

have evaluated the effectiveness of this key difference in producing faster convergence and higher accuracy on the Spambase dataset. This makes resilient propagation a promising choice for training neural networks for time-sensitive machine learning applications.

REFERENCES

[1] J. Clark, I. Koprinska, and J. Poon, "A neural network based approach to automated email classification," In Proceedings of IEEE/WIC international conference on web intelligence, Halifax, Canada, 2003, pp. 702–705.

[2] D. Gavrilis, I. G. Tsoulos, and E. Dermatas, "Neural recognition and genetic features selection for robust detection of e-mail spam," In Proceedings of the 4th Helenic conference on AI, Heraklion, Crete, Greece. Lecture notes in computer science, 2006, Vol. 3955, pp. 498–501.

[3] B. Cui, A. Mondal, J. Shen, G. Cong, and K. L. Tan, "On effective e-mail classification via neural networks," In Proceedings of the 16th international conference on database and expert systems applications (DEXA05), Copenhagen, Denmark, 2005, pp. 85–94.

[4] S. Ma, and C. Y. Ji, A unified approach on fast training of feedforward and recurrent networks using EM algorithm. IEEE Transaction on Signal Processing, 1998, Vol. 46(8), pp. 2270–2274.

[5] M. Riedmiller, and H. Braun. "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," IEEE International Conference on Neural Networks, 1993, pp. 586-591.

[6] S. Haykin, Neural networks: A comprehensive foundation, 2nd edn. 1998, Prentice Hall, New York