Design of an Adaptive Learning Rate for Neural Network Algorithms

Luis Quiles
Florida Institute of Technology
Melbourne, FL
lquiles@fit.edu

1. INTRODUCTION

Backpropagation networks are often used in classification, diagnosis, and decision problems. These networks make decisions based on internal weight links between input and output units, and these weights are updated based on the error of the network's final output values. A weight update is generally calculated from a user's predetermined learning rate and a measure of the error produced by the network's decision. Each weight is uniquely handled separate from the values of other weights. The same learning rate is used to update every weight of the net, despite the fact that some weights are further away from their best values than other weights in the net.

The algorithms developed during this project immediately do away with this weakness. With a single, unchanging learning rate, the change in the network's response is fixed. Even though some weights may be grossly incorrect and others may be very close to their desired value, the learning rate fixes the changes to these weights to a specific and unchanging rate. By allowing each weight to maintain a unique learning rate, and by allowing the learning rate to change, we allow the network to make better responses after each weight update. Ideally, weights with very high error will have a higher learning rate so that larger changes are made to the weight, while weights with very low error have smaller learning rates so that they do not become stuck oscillating around a certain value. From this change, the weights of the net should converge to produce output values with less error than the basic backpropagation neural network.

This paper will discuss the results and performance of two algorithms that calculate multiple unique learning rates. The first algorithm, Learning Rate by Relative Error, calculates the learning rate from the error at the output units connected to a particular hidden unit. The second algorithm, Learning Rate by Gradient, performs a similar calculation with the gradient term propagated backward during the training phase.

2. RELATED WORK

To understand the algorithms developed and the results of the project, it is important to provide a definition of the basic backpropagation neural network that is used in the experiment. The basic backpropagation neural net consists of three layers of units; the input, hidden, and output layers, respectively.

The consecutive layers of units are interconnected by weight values, initialized to values between [+.05, -.05]. The input layer receives its values from records in the data, while the values of the hidden and output layer units are calculated from the previous layer of linked units and the weight links connecting the layers.

The backpropagation net algorithm has two phases, output generation and backpropagation of error. Output generation is the process of calculating values for the output units for a given record of data. Backpropagation of error is the process of narrowing the error between the network's output and the expected output for the given training record.

The algorithm that inspired this project's experiment is known as SuperSAB (self-adaptive backpropagation). SuperSAB is an algorithm that introduced the concept of each weight having a unique learning rate [1]. Prior to this, a network would use a single learning rate for every weight in the network. This change was a novel approach, and combined with the algorithm for determining the new learning rate at each step, SuperSAB had considerable success as a competitor to the basic backpropagation neural network [1, 3].

3. APPROACH

3.1. Basic Backpropagation Network

The following are definitions for variables used by the backpropagation network algorithms:

- x_i – Input unit (where x_i is the i-th attribute of a record)
- h_j – Hidden unit
- o_k – Output unit
- t_k – Target value for an output unit
- n – Total number of input units
- m – Total number of hidden units
- p – Total number of output units
- $v_{i,j}$ – Weights linking input and hidden layers (initially random within [-.05,.05])
- $w_{j,k}$ – Weights linking hidden and output layers (initially random within [-.05,.05])
- $L_{j,k}$ – Learning rate for $w_{j,k}$ weights
- L_{base} – Learning rate defined by the user
- δ_k – Gradient term in backpropagation
- $\Delta w_{j,k}$ – Calculated change in a weight
- Sigmoid(x) = $1 / (1 + e^{-x}) = f(x)$

The variables m and L_{base} are user-defined parameters, while all other variables are either calculated or taken from the data.

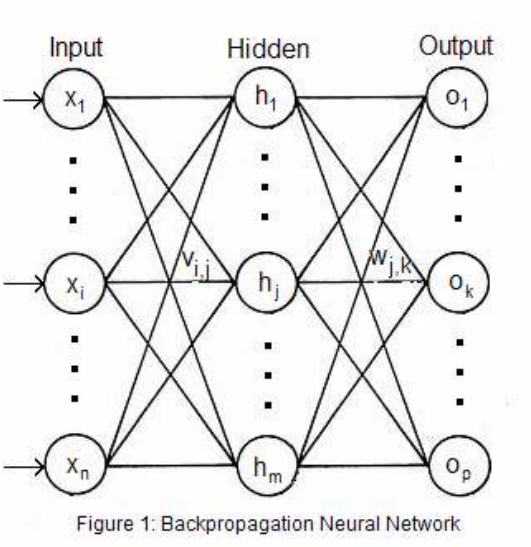

Figure 1: Backpropagation Neural Network

Figure 1 shows how these variables interact in the training phase of the neural network.

The output generation phase is common to all three algorithms being compared. The algorithm is as follows:

F1: $\boldsymbol{h_j = \text{Sigmoid}(\Sigma(x_i * v_{i,j}))}$

F2: $\boldsymbol{o_k = \text{Sigmoid}(\Sigma(h_j * w_{j,k}))}$

For the basic backpropagation network, the backpropagation algorithm used is defined as follows:

B1: $\boldsymbol{\delta_k = o_k(1 - o_k)(t_k - o_k)}$

B2: $\boldsymbol{\Delta w_{j,k} = L_{j,k} * w_{j,k} * \delta_k}$

B3: $\boldsymbol{w_{j,k} = w_{j,k} + \Delta w_{j,k}}$

In the basic backpropagation step, every learning rate ($L_{j,k}$) is initialized to the user-defined parameter for the learning rate, L_{base}.

3.2 Learning Rate Algorithms

Two algorithms were developed to generate the values of the learning rates. Learning Rate by Relative Error calculates a learning rate by statistical means based on the error at the output units attached to a particular hidden unit. Learning Rate by Gradient, on the other hand, uses the gradients calculated during backpropagation.

There is justification for both methods of calculating the learning rate. In the case of Learning Rate by Relative Error, it is often the case where only some of the weights need greater adjustment, while other weights hardly need any correction at all. The use of the standard deviation of the error of all the output nodes connected to a particular input node shows which weights were connected to nodes that needed the greatest correction. This knowledge allows the network to learn quicker by increasing the learning rate for that specific link.

For Learning Rate by Relative Error, several new variables need to be defined:

- $e_{j,k}$ - weighted error value from the absolute error at output unit k in connection to hidden unit j
- avg_j – Average value for $e_{j,k}$
- σ_j - Standard deviation of the weighted error values connected to hidden unit j

The algorithm is shown as follows, and is calculated after the first step of backpropagation (step B1).

B1: $\delta_k = o_k(1 - o_k)(t_k - o_k)$

A1e: $\boldsymbol{e_{j,k} = | w_{j,k} * (t_k - o_k) |}$

A2: $\boldsymbol{avg_j = \Sigma(e_{j,k}) / p}$

A3: $\boldsymbol{\sigma_j = \text{Sqrt}(\Sigma(e_{j,k} - avg_j) / p)}$

A4: $\boldsymbol{L_{j,k} = 1.1 * (e_{j,k} - avg_j) / \sigma_j + L_{base}}$

B2: $\Delta w_{j,k} = L_{j,k} * w_{j,k} * \delta_k$

B3: $w_{j,k} = w_{j,k} + \Delta w_{j,k}$

The Learning Rate by Relative Error algorithm is does not depend on the backpropagation algorithm, so it may also be calculated prior to step B1. Learning Rate by Gradient, on the other hand, is dependent on the first step, B1, of the backpropagation algorithm, so it must be calculated after B1. Since the output of step A4 is the learning rate $L_{j,k}$, the algorithm must be inserted between B1 and B2. The term ej,k is defined and calculated differently for this algorithm:

- $e_{j,k}$ – weighted gradient calculated during backpropagation for output unit k in connection to hidden unit j
- $e_{j,k} = | wj,k * \delta_k |$

The remainder of the algorithm remains the same. The integration into the backpropagation phase changes the whole phase to the following steps:

B1: $\delta_k = o_k(1 - o_k)(t_k - o_k)$

A1g: $\boldsymbol{e_{j,k} = | w_{j,k} * \delta_k |}$

A2: $\boldsymbol{avg_j = \Sigma(e_{j,k}) / p}$

A3: $\boldsymbol{\sigma_j = \text{Sqrt}(\Sigma(e_{j,k} - avg_j) / p)}$

A4: $\boldsymbol{L_{j,k} = 1.1 * (e_{j,k} - avg_j) / \sigma_j + L_{base}}$

B2: $\Delta w_{j,k} = L_{j,k} * w_{j,k} * \delta_k$

B3: $w_{j,k} = w_{j,k} + \Delta w_{j,k}$

The total error is the absolute indicator that determines which output units were grossly incorrect. The absolute error at the output unit does not provide enough information, unfortunately. In order to calculate individual and unique learning rates for the weight changing step, we need a value that indicates what portion of the error was caused by that particular weight. The term $e_{j,k}$ indicates what portion of the output error belongs to that particular weight link.

The average calculated in the second step of Learning Rate by Relative Error, and the standard deviation calculated in

the third step, are calculated from the total weighted error leaving a particular hidden unit. The algorithm tends to favor the weight connected to the output unit with the largest error, so if a particular output unit had a significantly large error while the error from other output unit are closer to their expected values, the weights connected to this unit will have the largest learning rate while the closer units will have much smaller learning rates.

This behavior is exactly what is desired. Whether the output unit is producing a large positive error or a large negative error, if the error is large, the learning rate should also be large to allow the weights to accelerate toward a good value. When the error is smaller, the learning rate should also be smaller so the weights may decelerate as they approach a good value. The error term ej,k is always positive, so the calculation of the learning rates only depends on the magnitude of the weighted error, and indicates both the sign of the error at the output units and the sign of the weights.

The calculated value of the learning rates depends on the user-defined parameter L_{base}, which is the base learning rate. This parameter is necessary, and is the mean of the learning rates for weight links connected to any single hidden unit.

Learning Rate by Gradient uses the gradient term calculated in the first step of the backpropagation phase. Because of this, the learning rate algorithm cannot be separated from the backpropagation phase, so their steps are integrated. In this algorithm, the learning rate for an individual weight link update is calculated based on which weight link has the largest gradient value. If the gradient term δ_k is very large, the learning rate should also be large so the weight may change rapidly and approach a better value more quickly. If the gradient term is small, the learning rate should also be small so the weight change does not cause the weight to overshoot its target value.

4. EVALUATION

4.1 Criteria

The two algorithms above were developed with one specific goal: to improve the final weight values calculated by the backpropagation neural network so that the total error of the decisions made by the neural network is reduced. Total error is the sum of the error of every output unit for every record in the dataset. The error of an output unit is the difference between its expected output (t_k) and its actual output (o_k).

To compare the weights generated by the basic backpropagation network and the networks using Learning Rate by Relative Error and Learning Rate by Gradient, the networks were tested on data segmented into training, validation, and testing datasets. The networks were trained on the training sets and tested on the validation and testing sets. The goal for the two learning rate algorithms was to produce a smaller total error on the test set than the basic neural network.

4.2 Data

The basic backpropagation algorithm was compared against both the Learning Rate by Relative Error and Learning Rate by Gradient algorithms, across three permutations of four datasets. The datasets were taken from the Proben database, mentioned in [5]. The only difference between this and the datasets found at www.ics.uci.edu/~mlearn is that they have been pre-filtered specifically to be used with neural networks.

The datasets that were used and their parameters are:

1. Heart disease (http://www.ics.uci.edu/~mlearn/databases/heart/)
   Given the personal details and symptoms of a patient, the goal is to determine if the patient has heart disease.
   460 training records
   230 validation records
   230 test records
   35 input attributes
   2 target output attributes
   5 hidden units
   Base learning rate: 0.3
2. Glass Identification (http://www.ics.uci.edu/~mlearn/databases/glass/)
   Given known attributes of a particular piece of glass, the goal is to identify the type of glass it is.
   107 training records
   54 validation records
   54 test records
   9 input attributes
   6 target output attributes
   10 hidden units
   Base learning rate: 0.4
3. Diabetes (http://www.ics.uci.edu/~mlearn/databases/diabetes/)
   Given the personal details and symptoms of a patient, the goal is to determine if the patient has diabetes.
   384 training records
   192 validation records
   192 test records
   8 input attributes
   2 target output attributes
   4 hidden units
   Base learning rate: 0.4
4. Breast Cancer (http://www.ics.uci.edu/~mlearn/databases/breast-cancer/)
   Given the personal details and symptoms of a patient, the goal is to determine if the patient has breast cancer.
   350 training records
   170 validation records
   170 test records
   9 input attributes
   2 target output attributes
   12 hidden units
   Base learning rate: 0.25

4.3 Procedure

The data can all be found in the archive referenced at [5]. For each problem, the archive contains three completely random permutations of the data. The first seven lines of the dataset contain the problem's parameters, with the exception of the learning rate and number of hidden units. These parameters are listed above.

The number of training records is taken to be half of the total dataset size, and is always the first half of the dataset. The validation set immediately follows, and it is one-fourth of the total dataset size. The final fourth of the dataset is the test set.

All three neural networks were trained over a minimum of 500 iterations with the provided training data. After every iteration of training, each network is tested on the training, validation, and test data, and the total error for each dataset is recorded. When 20 generations have passed, each neural network is allowed to "stop" when the recorded total error goes above the network's current error threshold. If the network stops before 500 iterations have passed, it is still trained until 500 iterations occur so that a trend in the training may be seen, but the best recorded error value does not change if better weights later cause a smaller error value.

The method to determine when to stop is very simple. The error from the network's output is assumed to have a standard distribution. Most of the values for the total error of the network should occur very close together, with some outlying values above and below the most common range of values. We prefer having values that are below the average total error, so when the validation set has its error at its minimum, the validation and test sets' total error is recorded, along with the current weights contained within the network.

The network will stop when the total output error is above a certain threshold. To calculate this value, we first find the average error of the validation set, then find what the error is at half a standard deviation above the average. Since new error values are recorded after every iteration of the network, and since the network could potentially go on for a very large number of iterations, it was impractical to keep track of the error at each step, so an online version of the standard deviation function was derived and used. The network stops training when the total error in the validation set is more than half a standard deviation above the average total error in the validation set. Below, n is the number of iterations that have occurred.

Online version of Standard Deviation:

- $A_0 = 0,\ A_n = A_{n-1} + x_n^2$
- $B_0 = 0, B_n = B_{n-1} + x_n$
- $\sigma_n = \text{sqrt}((A_n - 2*average* B_n + n*average^2) / n)$

The error produced by the validation and test datasets were recorded after every iteration of the networks' training, and every 25th data point was charted.

4.4 Results

4.4.1 Heart Disease

Table 1: Final Output Error for Heart Disease

Heart Disease	Gradient	Error	Basic Net
Permutation 1	*59.20*	**59.15**	59.25
Permutation 2	*59.20*	**59.19**	62.26
Permutation 3	**77.40**	*79.46*	79.70
Average	**65.27**	*65.93*	67.07

The heart disease problem contains 35 attributes per record which describe a patient and his symptoms. From these attributes, the goal of the network is to predict as accurately as possible whether the patient should be diagnosed with heart disease.

In testing against the first dataset, heart disease, the values received are encouraging, if perhaps a bit misleading. As evidenced in figures 2 and 3 below (permutation 3 of the heard disease dataset), the performance of the validation and test datasets are very similar (all graphs begin with a dip, then steadily rise), which is what we expect. However, for the basic net, the best error in the validation set occurs fairly "late" around the 85th iteration, compared to the two other algorithms which have minimums much earlier. Due to this, in the test set, the best point for the basic network, as chosen by the stopping criterion, is actually on the rising slope of the test set's curve. For the Learning Rate by Relative Error and Learning Rate by Gradient, the network finds a minimum earlier, and in both cases a better test set total error is found because of it.

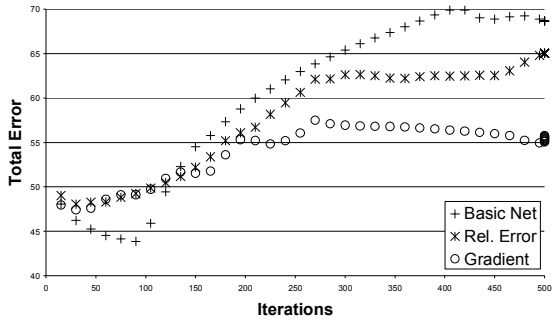

Figure 2: Heart Disease (Set 3, Validation)

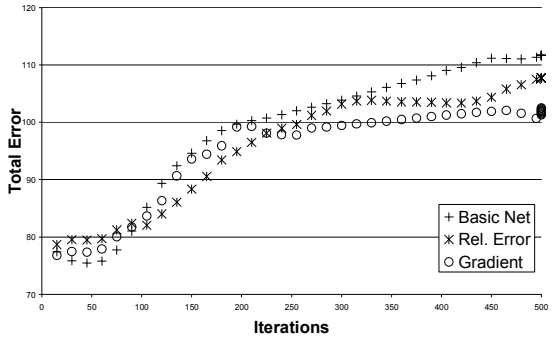

Figure 3: Heart Disease (Set 3, Testing)

It is interesting to note that, for this dataset, if the stopping criterion was simply measured in the number of iterations performed (ignoring the fact that we'd like to find weights faster), the result is that both Learning Rate by Gradient and Learning Rate by Relative Error perform better than the Basic Backpropagation net.

4.4.2 Glass Identification

Table 2: Final Output Error for Glass Identification

Glass	Gradient	Error	Basic Net
Permutation 1	28.90	**22.98**	26.31
Permutation 2	*33.11*	**29.65**	33.184
Permutation 3	*33.93*	**30.61**	34.40
Average	31.98	**27.75**	31.30

The glass identification problem contains 9 attributes per record which describe the content of certain elements for various pieces of glass. From these attributes, the goal of the network is to determine what type of glass is being presented.

The results of the Glass Identification dataset are also favorable toward the Learning Rate by Relative Error, though the Learning Rate by Gradient does not fare as well this time. Out of the three permutations, the Relative Error outperforms both other algorithms by a small margin in all three permutations of the data. The Gradient performs well in two of the tests, but the third test is bad enough that, when taking the average total error of the tests, the basic neural network still performs better.

Again, the performance between the validation and testing sets are similar in nature. However, in the case of the graph of the total error for the Gradient, it is evident that there is some divergence in performance. During validation, the nature of the error was to drop, then steadily approach some value before stopping. In the testing phase, however, the error in the gradient suddenly rose, which is why a high value ended up chosen for the gradient, when compared to the lower values for both the Basic net and the Relative Error net.

Simply regarding the Basic Backpropagation network against the Learning Rate by Relative Error network shows that the relative error network clearly has an advantage over the basic network. Initially the basic net performs better, but quickly loses its advantage long before either net's error curve reaches its minimum. The same pattern occurs in the testing set, though a bit more exaggerated. Thus, the relative error network is shown to work very well as a competitive neural network algorithm.

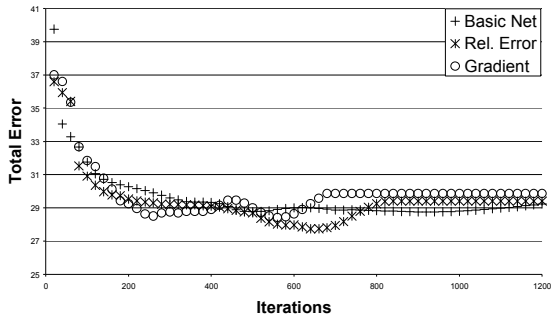

Figure 4: Glass Identification (Set 1, Validation)

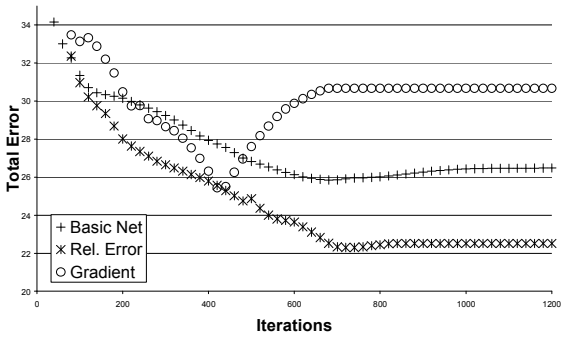

Figure 5: Glass Identification (Set 1, Testing)

4.4.3 Diabetes

Table 3: Final Output Error for Diabetes

Diabetes	Gradient	Error	Basic Net
Permutation 1	*62.52*	**62.36**	63.01
Permutation 2	71.25	**70.62**	70.66
Permutation 3	**62.17**	*62.26*	65.04
Average	*65.31*	**65.08**	66.24

The diabetes problem contains 8 attributes per record which describe a patient and his symptoms. From these attributes, the goal of the network is to predict as accurately as possible whether the patient should be diagnosed with diabetes.

The diabetes dataset is another set where the new learning rate networks outperform the basic network, though like the heart disease dataset, the results are very close. Once again, the Learning Rate by Relative Error network performs better than the Basic Backpropagation network in all three permutations of the data, while the Learning Rate by Gradient network loses in one case. Unlike the previous dataset, however, the gradient network performs better than the basic network overall, and performs nearly as well as the relative error network.

In the diabetes dataset the three networks perform remarkably similar to each other. Like the first dataset, heart disease, the winning performance from the relative error and gradient networks comes nearly as a side effect of the stopping criterion. Unlike the previous two datasets,

however, in the diabetes dataset, the basic network outperforms both of the new algorithms in the long run. This trend seems to imply that the basic network is the best c hoice; however, from the previous two examples, the basic network could not be shown to be the best choice every time.

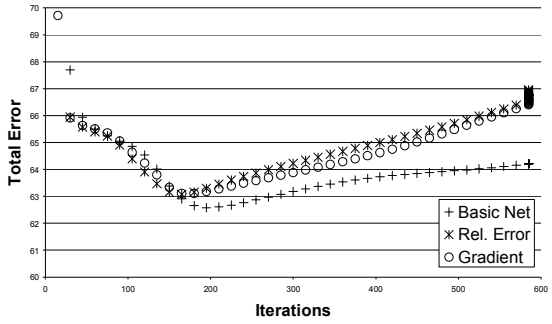

Figure 6: Diabetes (Set 2, Validation)

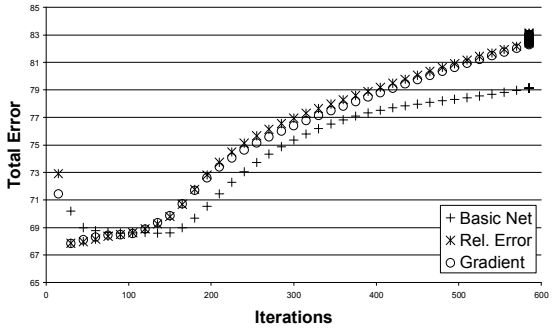

Figure 7: Diabetes (Set 2, Testing)

4.4.4 Breast Cancer

Table 1: Final Output Error for Breast Cancer

Breast Cancer	Gradient	Error	Basic Net
Permutation 1	3.42	3.40	**3.37**
Permutation 2	10.22	10.06	**9.83**
Permutation 3	10.71	10.73	**9.63**
Average	8.11	8.07	**7.61**

The breast cancer problem contains 9 attributes per record which describe a patient and his symptoms. From these attributes, the goal of the network is to predict as accurately as possible whether the patient should be diagnosed with breast cancer.

The breast cancer dataset is different from the previous datasets for one very important reason: the Basic Backpropagation network outperforms both the Learning Rate by Relative Error network and the Learning Rate by Gradient network in all three permutations of the data. Although all three networks have similar behavior for each permutation of the data, in every situation the basic network consistently performed better than either of the other two networks.

As seen below, the performance of the networks on the validation set coincides nearly exactly with the performance of the networks on the test set. This makes sense, even if the performance seen is not desirable. For each of the three networks, the error begins low, reaches a quick minimum value, and then drastically rises before leveling off. The curve itself basically shows that the neural networks are overtraining nearly from the first iteration of the network. Like the diabetes dataset, the basic network consistently performs better than the other two networks throughout the entire graph, demonstrating that the Learning Rate by Relative Error or by Gradient is not a very good solution for this particular problem.

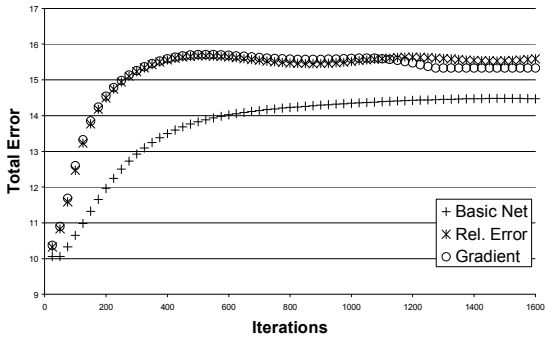

Figure 8: Breast Cancer (Set 3, Validation)

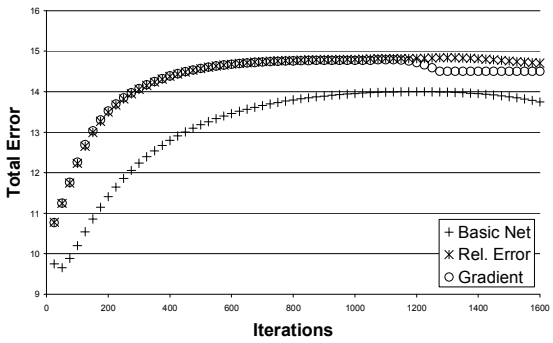

Figure 9: Breast Cancer (Set 3, Testing)

5. CONCLUSION

The Learning Rate by Gradient and Learning Rate by Relative Error networks appear to perform competitively well against the Basic Backpropagation neural network. Although neither of these two algorithms could perform better than the basic net in the Breast Cancer dataset, they both performed very well on the other three datasets. Overall, these two new algorithms are good alternatives that generally perform with equal or better performance when compared to the basic backpropagation algorithm. However, as the Breast Cancer database shows, there are problems that exist that have horrible performance with the Gradient and Relative Error networks.

Several ideas come to mind for possible future experiments with either or both algorithms. One of the most commonly known changes to the basic backpropagation algorithm to improve performance is the addition of a momentum term.

This leads to the question as to whether the relative error and gradient networks would improve with the addition of this term. Another interesting change would be the inclusion and handling of bias units (an extra input unit whose value is always 1) in each layer. A third possibility, and perhaps the most important of the three, is to see how the relative error and gradient networks would perform when there are multiple hidden layers, when compared to another network with multiple hidden layers.

6. REFERENCES:

1. Bishop, J. M., and J. M. Hannan. A Comparison of Fast Training Algorithms Over Two Real Problems. Artificial Neural Networks, Fifth International Conference on (Conf. Publ. No. 440). 1997.

2. Leustean, L. Liquid flow time series prediction using feed-forward neural networks and SuperSAB learning algorithm. Conti'2002. 5th International Conference on Technical Informatics, 18-19 October 2002, Timisoara, Romania, Buletinul stiintific al Universitatii "Politehnica" din Timisoara, seria Automatica si Calculatoare. Tomul 47(61), No. 1, 2002, pp. 77-82.

3. Sarkar, D. Methods to speed up error back-propagation learning algorithm. ACM Computer Surveys, 27(4), 519-542.

4. Schiffmann, W., M. Joost, and R. Werner. Comparison of Optimized Backpropagation Algorithms. Proc. of the European Symposium on Artificial Neural Networks. ESANN '93. Brussels. page 97-104. 1993

5. Proben1 Database (pre-filtered for use with neural networks): ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz