PERFORMANCE IMPROVEMENT OF BACK-PROPAGATION NEURAL NETWORK LEARNING ALGORITHMS BY INTRODUCING GAIN VARIATION OF ACTIVATION FUNCTION

Nazri Mohd Nawi, Rozaida Ghazali, Mohd Najib Mohd Salleh[1]

1) Faculty of Information Technology and Multimedia, Universiti Tun Hussein Onn Malaysia (UTHM), Batu Pahat, Johor, MALAYSIA

http://www.uthm.edu.my

nazri@uthm.edu.my, rozaida@uthm.edu.my, najib@uthm.edu.my

Abstract *–We propose a method for improving the performance of the back propagation algorithm by introducing gain variation of the activation function. In a 'feed forward' algorithm, the slope of the activation function is directly influenced by a parameter referred to as 'gain'. In this paper, the influence of the variation of 'gain' on the learning ability of a neural network is analysed. Multi layer feed forward neural networks have been assessed. Physical interpretation of the relationship between the gain value and the learning rate and weight values is given. Instead of a constant 'gain' value, we propose an algorithm to change the gain value adaptively for each node. The efficacy of the proposed method is verified by means of simulation on a parity problem and classification problem. The results show that the proposed method considerably improves the learning speed of the general back-propagation algorithm.*

Keywords: *Back-propagation Neural Networks, Gain, Activation function, Learning rate, Training Efficiency*

1. Introduction

We consider standard multi-layer feed forward neural networks that have an input layer of neurons, a hidden layer of neurons and an output layer of neurons, and in that every node in a layer is connected to every other node in the adjacent forward layer. The back-propagation algorithm has been the most popular and most widely implemented for training these types of neural network. When using the back-propagation algorithm to train a multilayer neural network, the designer is required to arbitrarily select parameter such as the network topology, initial weights and biases, a learning rate value, the activation function, and a value for the gain in the activation function. An improper choice of any of these parameters can result in slow convergence or even network paralysis where the training process comes to a virtual standstill. Another problem is the tendency of the steepest descent technique, which is used in the training process, to get stuck at local minima.

In recent years, a number of research studies have attempted to overcome these problems. Theses involved the development of heuristic techniques, based on studies of properties of the general back-propagation algorithm. These techniques include such idea as varying the learning rate, using momentum, gain tuning of activation function. Perantonis et. al. [1] proposed an algorithm for efficient learning in feed forward neural networks using momentum acceleration. Kamarthi et. al. [2] presents a universal acceleration technique for the back-propagation algorithm based on extrapolation of each individual interconnection weight. Research has also focused on the use of standard numerical optimisation techniques. Moller [3] explained how conjugate gradient algorithm could be used to train multi-layer feed forward neural networks while Lera et. al. [4] described the use of Levenberg-Marquardt algorithm for training multi-layer feed forward neural networks. However, most of these methods are quite complex, require excessive memory and are computationally very expensive.

In order to improve the performance of the back-propagation algorithm an algorithm has been proposed in this paper to change the gain value adaptively. It is shown that changing the 'gain' value adaptively for each node can significantly reduce the training time. In order to verify the efficacy of the proposed method, and to compare it with the general back-propagation algorithm, we perform simulation experiments on a function approximation problem using the sequential as well as batch modes of training.

1.1 Effect of the gain parameter on the performance of a neural network

An activation function is used for limiting the amplitude of the output of a neuron. It generates an output value for a node in a predefined range as the closed unit interval [0, 1] or alternatively [-1, +1]. This value is a function of the weighted inputs of the corresponding node. The most commonly used activation function is the logistic sigmoid activation function. Alternative choices are the hyperbolic tangent, linear, step activation functions. For the j^{th} node, a logistic sigmoid activation function which has a range of [0, 1] is a function of the following variables, viz.

$$o_j = \frac{1}{1+e^{-c_j a_{net,j}}} \qquad (1)$$

Where,

$$a_{net,j} = \left(\sum_{i=1}^{l} w_{ij} o_i \right) + \theta_j \qquad (2)$$

Where,

o_j Output of the j^{th} unit.

w_{ij} weight of the link from unit i to unit j.

$a_{net,j}$ net input activation function for the j^{th} unit.

θ_j bias for the j^{th} unit.

c_j gain of the activation function.

The value of the gain parameter, c_j, directly influences the slope of the activation function. For large gain values ($c >> 1$), the activation function approaches a 'step function' whereas for small gain values ($0 < c << 1$), the output values change from zero to unity over a large range of the weighted sum of the input values and the sigmoid function approximates a linear function.

Most of the application oriented papers on neural networks tend to advocate that neural networks operate like a 'magic black box', which can simulate the "learning from example" ability of our brain with the help of network parameters such as weights, biases, gain, hidden nodes etc. There are very few publications, or textbooks, which give physical interpretation for various parameters used in the network. Also, a unit value for gain has generally been used for most of the research reported in the literature but a few authors have researched the relationship of the gain parameter with other parameters used in back-propagation algorithms. The recent results [5] show that learning rate, momentum constant and gain of the activation function have a significant impact on training speed. However, higher values of learning rate and/or gain cause instability [6]. Thimm et. al. [7] also proved that a relationship between the gain value, a set of initial weight values, and a learning rate value exists. Looney [8] suggested to adjust the gain value in small increments during the early iterations and to keep it fixed somewhere around halfway through the learning. Eom et. al. [9] proposed a method for automatic gain tuning using a fuzzy logic system.

However, the authors have not come across publications in the literature that have implemented adaptive gain variation as proposed in this research work.

2. The Proposed Method

In this section, a novel approach for improving the training efficiency of back propagation neural network algorithms is proposed. The proposed algorithm modifies the initial search direction by changing the gain value adaptively for each node. The following subsection describes the algorithm. The advantages of using an adaptive gain value have been explored. Gain update expressions as well as weight and bias update expressions for output and hidden nodes have also been proposed. These expressions have been derived using same principles as used in deriving weight updating expressions.

The sequential mode of training requires an immediate updating of weights, biases and gains after the presentation of training example whereas in the batch mode of training the weight, bias and gain updation terms are calculated and summed for all the training examples. In this paper, we only consider on the batch mode training as in the batch mode training the weights, biases and gains are updated after one complete presentation of the entire training set. An epoch is said to be complete after the presentation of the entire training set. A sum squared error value is calculated after the presentation of the training set and compared with the target error. Training is done on an epoch-by-epoch basis until the sum squared error falls below the desired target value. The following iterative algorithm is proposed by the authors for the batch mode of training. Weights, biases and gains are calculated and updated for the entire training set, which is being presented to the network.

The following iterative algorithm is proposed by the authors for the batch mode of training. Weight, biases and gains are calculated and updated for each training example, which is being presented to the network.

```
For a given epoch,
    For each example,
        Step 1 Calculate the weight and bias values using
               the previously converged gain value.
        Step 2 Use the weight and bias value calculated in
               step (1) to calculate the new gain value.
Repeat steps (1) and (2) for each example on an epoch-
by-epoch basis until the error on the entire training data
set reduces to a predefined value.
```

The gain update expression for a gradient descent method is calculated by differentiating the following error term E with respect to the corresponding gain parameter.

The network error E is defined as follows:

$$E = \frac{1}{2} \sum (t_k - o_k(o_j, c_k))^2 \qquad (3)$$

For output unit, $\frac{\partial E}{\partial c_k}$ needs to be calculated whereas for hidden units. $\frac{\partial E}{\partial c_j}$ is also required. The respective gain values would then be updated with the following equations.

$$\Delta c_k = \eta \left(-\frac{\partial E}{\partial c_k}\right) \qquad (4)$$

$$\Delta c_j = \eta \left(-\frac{\partial E}{\partial c_j}\right) \quad (5)$$

$$\frac{\partial E}{\partial c_k} = -(t_k - o_k)\, o_k (1 - o_k)\left(\sum w_{jk} o_j + \theta_k\right) \quad (6)$$

Therefore, the gain update expression for links connecting to output nodes is:

$$\Delta c_k(n+1) = \eta\, (t_k - o_k)\, o_k (1 - o_k)\left(\sum w_{jk} o_j + \theta_k\right) \quad (7)$$

$$\frac{\partial E}{\partial c_j} = \left[-\sum_k c_k w_{jk} o_k (1-o_k)(t_k - o_k)\right] o_j (1-o_j)\left(\left(\sum_j w_{ij} o_i\right) + \theta_j\right) \quad (8)$$

Therefore, the gain update expression for the links connecting hidden nodes is:

$$\Delta c_j(n+1) = \eta\left[-\sum_k c_k w_{jk} o_k (1-o_k)(t_k - o_k)\right] o_j (1-o_j)\left(\left(\sum_j w_{ij} o_i\right) + \theta_j\right) \quad (9)$$

Similarly, the weight and bias expressions are calculated as follows:

The weight update expression for the links connecting to output nodes with a bias is:

$$\Delta w_{jk} = \eta (t_k - o_k)\, o_k (1 - o_k)\, c_k o_j \quad (10)$$

Similarly, the bias update expressions for the output nodes would be:

$$\Delta \theta_k = \eta\, (t_k - o_k)\, o_k (1 - o_k) c_k$$
(11)

The weight update expression for the links connecting to hidden nodes is:

$$\Delta w_{ij} = \eta\left[\sum_k c_k w_{jk} o_k (1-o_k)(t_k - o_k)\right] c_j o_j (1-o_j) o_i$$
(12)

Similarly, the bias update expressions for the hidden nodes would be:

$$\Delta\theta_j = \eta\left[\sum_k c_k w_{jk} o_k (1-o_k)(t_k - o_k)\right] c_j o_j (1-o_j) \quad . (13)$$

3. Results and Discussions

The performance criterion used in this research focuses on the speed of convergence, measured in number of iterations and CPU time. The benchmark problems used to verify our algorithm are taken from the open literature. Four classification problems have been tested including parity 7 bit, Wisconsin breast cancer, Diabetes and IRIS classification problem. The simulations have been carried out on a Pentium IV with 3 GHz PC Dell, 1 GB RAM and using MATLAB version 6.5.0 (R13).

On each problem, the following three algorithms were analyzed and simulated.

1) The standard gradient descent with momentum (*traingdm*) from 'Matlab Neural Network Toolbox version 4.0.1'.
2) The standard Gradient descent with momentum (GDM)
3) The proposed Gradient descent with momentum and Adaptive Gain (GDM/AG)

To compare the performance of the proposed algorithm with respect to other standard optimization algorithms from the MATLAB neural network toolbox, network parameters such as network size and architecture (number of nodes, hidden layers etc), values for the initial weights and gain parameters were kept same. For all problems the neural network had one hidden layer with five hidden nodes and sigmoid activation function was used for all nodes. All algorithms were tested using the same initial weights, initialized randomly from range [0, 1] and received the input patterns for training in the same sequence.

For all training algorithms, the learning rate value was 0.3 and the momentum term value was 0.7. The initial value used for the gain parameter was one. For each run, the numerical data is stored in two files- the results file, and the summary file. The result file lists data about each network. The number of iterations until convergence is accumulated for each algorithm from which the mean, the standard deviation and the number of failures are calculated. The networks that fail to converge are obviously excluded from the calculations of the mean and standard deviation but are reported as failures.

For each problem, 100 different trials were run, each with different initial random set of weights. For each run, the number of iterations required for convergence is reported. For an experiment of 100 runs, the mean of the number of iterations (mean), the standard deviation (SD), and the number of failures are collected. A failure occurs when the network exceeds the maximum iteration limit; each experiment is run to ten thousand iterations; otherwise, it is halted and the run is reported as a failure. Convergence is achieved when the outputs of the network conform to the error criterion as compared to the desired outputs.

3.1 7 Bit Parity Problem

The parity problem is also one of the most popular initial testing tasks and very demanding classification for neural network to solve, because the target-output changes whenever a single bit in the input vector changes and this makes generalization difficult and learning does not always converge easily[10]. The selected architecture of the FNN is 7-5-1. The target error has been set to 0.05.

The results in table 1 show that the convergence success rate of GDM/AG is 96%, and that the average number learning iterations is reduced by about 60% compared to GDM, and by about 96% compared to traingdm. An outstanding characteristic of the proposed GDM/AG method is that the parity 7 bit problem can be solved with very few learning iterations.

	Parity 7 Bit Problem, Target Error = 0.05		
	traingdm	**GDM**	**GDM/AG**
Mean	14272	1347	537
CPU time(s)/Epoch	9.81×10^{-2}	3.80×10^{-2}	3.99×10^{-2}
Total CPU time(s) of converge	140.02	51.15	21.42
SD	2.72×10^{3}	5.09×10^{2}	1.83×10^{2}
Failures	12	7	4

Table 1. Algorithm Performance for Parity 7 bit problem

The *minimum* number of pages for your paper is 5. The *maximum* number of pages accepted is 8. Changing the formats described in the template may result in exclusion from the proceedings if repeated in your final accepted camera-ready paper. Rather, take some time to review the text again for improvement or consult the Conference Secretariat.

3.2 Breast Cancer Classification Problem

This dataset was created based on the 'breast cancer Wisconsin' problem dataset from UCI repository of machine learning databases from Dr. William H. Wolberg[11]. This problem tries to diagnosis of breast cancer by trying to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. The selected architecture of the Feed- forward Neural Network is 9-5-2. The target error is set as to 0.02.

Table 2 clearly shows that algorithms which implement our proposed method exhibit very good average performance in order to reach target error. Furthermore the number of failures for the proposed method is smaller as compared to others. GDM/AG needs only 405 epochs to converge as opposed to the standard GDM at about 1105 epochs and *traingdm* with 3419 epochs. The results clearly show that the proposed method GDM/AG outperform neural network toolbox algorithm with an improvement ratio, nearly 3.2, for the total time of converge.

	Breast Cancer Problem, Target Error = 0.02		
	traingdm	**GDM**	**GDM/AG**
Mean	3419	1105	405
CPU time(s)/Epoch	1.60×10^{-2}	4.71×10^{-2}	4.45×10^{-2}
Total CPU time(s) of converge	54.59	52.05	18.02
SD	1.22×10^{3}	1.16×10^{3}	6.64×10^{2}
Failures	14	4	3

Table 2. Algorithm Performance for Breast Cancer problem[14]

Type the title approximately 3.5 centimeters below the top border of the letter size paper sheet and use a 20 point size typefont. Center the title (horizontally) on the page. Leave approximately 1 cm (0.4 inches) between the title and the name and address of yourself and of your co-authors, if any. Type the name(s), address(es) and email in an 11 point font and center them (horizontally) on the page.

3.3 IRIS Classification problem

This is a classical classification dataset made famous by Fisher[12], who used it to illustrate principles of discriminant analysis. This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The selected architecture of the Feed- forward Neural Network is 4-5-3 with target error was set as 0.05.

	IRIS Problem, Target Error = 0.05		
	traingdm	**GDM**	**GDM/AG**
Mean	1609	754	581
CPU time(s)/Epoch	2.69×10^{-2}	3.89×10^{-2}	3.69×10^{-2}
Total CPU time(s) of converge	43.30	29.28	21.42
SD	6.58×10^{2}	2.94×10^{2}	2.43×10^{2}
Failures	15	4	3

Table 3. Algorithm Performance for IRIS problem[13]

In table 3 shows that algorithms which implement the proposed method still outperforms other algorithms in term of CPU time and number of epochs. As we can see that GDM/AG was 97% successful at learning the patterns and the average number of learning iterations was reduced greatly as it is three times faster as compared to neural network toolbox.

3.4 Diabetes Classification Problem

This dataset was created based on the 'Pima Indians diabetes' problem dataset from the UCI repository of machine learning database. From the dataset doctors try to diagnose diabetes of Pima Indians based on personal data (age, number of times pregnant) and the results of medical examinations (e.g. blood pressure, body mass index, result of glucose tolerance test, etc.) before decide whether a Pima Indian individual is diabetes positive or not. The selected architecture of the Feed-forward Neural Network is 8-5-2. The target error is set to 0.01.

In table 4, it is worth noticing that the performance of the proposed GDM/AG is almost twice faster as compared to neural network toolbox. GDM/AG took only 417 epochs to reach the target error compare to GDM at about 520 epochs and worst for *traingdm* that need about 965 epochs to converge. Still the proposed algorithm outperform other algorithms in term of the total time of converge.

	Diabetes Problem, Target Error = 0.01		
	traingdm	GDM	GDM/AG
Mean	965	520	417
CPU time(s)/Epoch	3.14×10^{-2}	5.00×10^{-2}	3.54×10^{-2}
Total CPU time(s) of converge	30.36	25.97	14.76
SD	1.45×10^{2}	1.14×10^{2}	1.02×10^{2}
Failures	13	5	4

Table 4. Algorithm Performance for Diabetes problem[14]

4. Advantages of Using Adapative Gain Variation

An algorithm has been proposed in this paper for the efficient calculation of the adaptive gain value in both sequential and batch modes of learning. We proposed that the total learning rate value can be split into two parts – a local (nodal) learning rate value and a global (same for all nodes in a network) learning rate value. The value of parameter gain is interpreted as the local learning rate of a node in the network. The network is trained using a fixed value of learning rate equal to 0.3 which is interpreted as the global learning rate of the network. However, as the gain value was modified, the weights and biases were updated using the new value of gain. This resulted in higher values of gain which caused instability [7]. To avoid oscillations during training and to achieve convergence, an upper limit of 2.0 is set for the gain value. This will be explained in detail in our next publication. The method has been illustrated for Gradient Descent training algorithm using the sequential and batch modes of training. An advantage of using the adaptive gain procedure is that it is easy to introduce into a back-propagation algorithm and it also accelerates the learning process without a need to invoke solution procedures other than the Gradient Descent method. The adaptive gain procedure has a positive effect in the learning process by modifying the magnitude, and not the direction, of the weight change vector. This greatly increases the learning speed by amplifying the directions in the weight space that are successfully chosen by the Gradient-Descent method. However, the method will also be advantageous when using other faster optimization algorithms such as Conjugate-Gradient method and Quasi-Newton method. These methods can only optimize an equivalent of the global learning rate (the step length). By introducing an additional local learning rate parameter, further increase in the learning speed can be achieved. Work is currently under progress to implement this algorithm using other optimization methods.

5. Conclusion

While the back-propagation algorithm is used in the majority of practical neural networks application and has been shown to perform relatively well, there still exist areas where improvement can be made. We proposed an algorithm to adaptively change the gain parameter of the activation function to improve the learning speed. It was observed that the influence of variation in the gain value is similar to the influence of variation in the learning rate value. An algorithm has been proposed in this paper to change the gain value adaptively for each node.

In order to verify the effectiveness of the proposed algorithm, some benchmark problems were simulated and analyzed using batch modes of training. The results showed that the proposed adaptive gain algorithm has a better convergence rate and learning efficiency as compared to the general back-propagation algorithm.

The network also demonstrated the principles of over fitting vs. generalization as the number of hidden nodes in the network was increased and the target error was reduced further. The choice of normally distributed random numbers in the range [-1, +1] for the initial values of weights and biases in the network greatly speeded the training process. The results strengthen our belief in the better working of the adaptive gain algorithm.

Acknowledgement

The authors would like to thank Universiti Tun Hussein Onn Malaysia for supporting this research under the Short Term Research Grant.

References

[1] Perantonis S. J. and Karras D. A., *An Efficient Constrained Learning Algorithm with Momentum Acceleration.* Neural Networks. 1995(8(2)): p. 237-249.

[2] Kamarthi S. V. and Pitner S., *Accelerating Neural Network Training using Weight Extrapolations.* Neural Networks, 1999. 12: p. 1285-1299.

[3] Moller M. F., *A Scaled Conjugate Gradient Algorithm for fast Supervised Learning.* Neural Networks, 1993. 6(4): p. 525-533.

[4] Lera G. and Pinzolas M., *Neighborhood based Levenberg- Marquardt Algorithm for Neural Network Training.* IEEE Transaction on Neural Networks, September 2002. 13(5): p. 1200-1203.

[5] Holger R. M. and Graeme C. D., *The Effect of Internal Parameters and Geometry on the Performance of Back-Propagation Neural Networks.* Environmental Modeling and Software, 1998. 13(1): p. 193-209.

[6] Hollis P. W., Harper J. S., and Paulos J. J., *The Effects of Precision Constraints in a Backpropagation Learning Network.* Neural Computation, 1990. 2(3): p. 363-373.

[7] Thimm G., Moerland F., and Fiesler E., *The Interchangeability of Learning Rate an Gain in Backpropagation Neural Networks*. Neural Computation, 1996. 8(2): p. 451-460.

[8] Looney C. G., *Stabilization and Speedup of Convergence in Training Feed Forward Neural Networks*. Neurocomputing, 1996. 10(1): p. 7-31.

[9] Eom K., Jung K., and Sirisena H., *Performance Improvement of Backpropagation algorithm by automatic activation function gain tuning using fuzzy logic*. Neurocomputing, 2003. 50: p. 439-460.

[10] Erik Hjelmas and P.W. Munro, *A comment on parity problem*. Technical Report, 1999: p. 1-7.

[11] Mangasarian O. L. and Wolberg W. H., *Cancer diagnosis via linear programming*. SIAM News, 1990. 23(5): p. 1-18.

[12] Fisher R.A., *The use of multiple measurements in taxonomic problems*. Annals of Eugenics, 1936. 7: p. 179-188.

[13]R.A. Fisher, *ftp://ftp.ics.uci.edu/pub/machine-learningdatabases/iris/iris.data.* 1988.

[14]Lutz Prechelt, *ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz*. 1994.