International Conference Mathematical and Computational Biology 2011
International Journal of Modern Physics: Conference Series
Vol. 9 (2012) 448–455
© World Scientific Publishing Company
DOI: 10.1142/S2010194512005533

SOLVING LOCAL MINIMA PROBLEM IN BACK PROPAGATION ALGORITHM USING ADAPTIVE GAIN, ADAPTIVE MOMENTUM AND ADAPTIVE LEARNING RATE ON CLASSIFICATION PROBLEMS

NORHAMREEZA ABDUL HAMID

Post Graduate Centre, Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, 86400 Parit Raja, Batu Pahat, Johor, Malaysia
gi090007@siswa.uthm.edu.my

NAZRI MOHD NAWI

Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, 86400 Parit Raja, Batu Pahat, Johor, Malaysia
nazri@uthm.edu.my

ROZAIDA GHAZALI

Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, 86400 Parit Raja, Batu Pahat, Johor, Malaysia
rozaida@uthm.edu.my

MOHD NAJIB MOHD SALLEH

Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, 86400 Parit Raja, Batu Pahat, Johor, Malaysia
najib@uthm.edu.my

This paper presents a new method to improve back propagation algorithm from getting stuck with local minima problem and slow convergence speeds which caused by neuron saturation in the hidden layer. In this proposed algorithm, each training pattern has its own activation functions of neurons in the hidden layer that are adjusted by the adaptation of gain parameters together with adaptive momentum and learning rate value during the learning process. The efficiency of the proposed algorithm is compared with the conventional back propagation gradient descent and the current working back propagation gradient descent with adaptive gain by means of simulation on three benchmark problems namely iris, glass and thyroid.

Keywords: back propagation; gain; momentum; learning rate.

1. Introduction

Artificial Neural Network (ANN) is computational models whose architecture and operation is inspired by human knowledge of biological nervous system. Due to its ability to solve some problems with relative ease of use, robustness to noisily input data, execution speed and can analyse complicated systems without accurate modeling in advance, ANN has successfully been implemented across an extraordinary range of problem domains[1-4]. ANN consists of input layer, hidden layer and output layer with every node in a layer is connected to every node in the adjacent forward layer.

The BP algorithm is the most popular, effective and easy to produce model for complex ANN. BP algorithm is a supervised learning method that uses a gradient descent (GD) algorithm which attempts to minimise the error of the network by moving down the gradient of the error curve. Since BP uses GD algorithm, the problems include a slow learning convergence and easily get trapped at local minima[5]. We have noted that many local minima difficulties are closely related to the neuron saturation in the hidden layer. Once such saturation occurs, neurons in the hidden layer will lose their sensitivity to input signals, and the propagation of information is blocked severely. In some cases, the network can no longer learn. Moreover, the convergence behaviour also depends on the selection of network topology, initial weights and biases, learning rate (LR), momentum coefficient (MC), activation function (AF) and gain value in the AF.

Over the years, a significant number of methods have been produced to improve the generalisation ability of ANN. These implicated the development of heuristic techniques, based on properties studies of the conventional BP algorithm.[6] clarified the used of Levenberg-Marquardt algorithm for training ANN. Though, the training times required strongly depend on neighborhood size. While[5] designed the optical BP (OBP) algorithm which is used for training process that depends on ANN with a very small LR, especially when using a large size training set. On the contrary, it does not guarantee to converge at local minima since if the error closes to maximum, the OBP error grows increasingly.

Nazri, *et al.*[7] demonstrated that changing the 'gain' value adaptively for each node can significantly reduce the training time without changing the network topology. For the j^{th} node, a logistic sigmoid AF which has a range of $[0,1]$, viz:

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

where,

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

Where o_j represents the output of the j^{th} unit, while o_i denotes the output of the i^{th} unit, w_{ij} indicates the weight of the link from unit i to unit j, $a_{net,j}$ construes the net of input AF for the j^{th} unit, θ_j represent the bias for the j^{th} unit and c_j denotes the gain of the AF.

Nazri *et al.*[7] demonstrated that the value of the gain c_j directly influences the slope of the AF. For $(c \geq 1)$, the AF approaches a 'step function' whereas for small gain values $(0 < c \leq 1)$, 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.

The LR is one of the most effective means to accelerate the convergence of BP learning. It is a crucial factor to control the variable of the neuron weight adjustments at each iteration during the training process and therefore affects the convergence rate. The algorithm will take longer time to converge or may never converge if the LR is too small. On the contrary, the network will accelerate the convergence rate significantly and still possibly will cause the instability whereas the algorithm may oscillate on the ideal path if the LR value is too high. The value of LR usually set to be constant. Yet, Ye[8] stated that the constant LR of the BP fails to optimise the search for the optimal weight combination. Another effective approach to hasten up the convergence and stabilise the training procedure is by adding some MC to the network. Moreover, with MC, the network can slide through shallow local minima. Formerly, the MC is typically preferred to be constant in the interval $[0,1]$. In spite of that, it is discovered from simulations that the fixed MC value seems to hasten up learning only when the recent downhill gradient of the error function and the last change in weight have a parallel direction. When the recent negative gradient is in a crossing direction to the previous update, the MC may cause the weight to be altered up the slope of the error surface as opposed to down the slope as preferred. This leads to the emergence of diverse schemes for adjusting the MC value adaptively instead of being kept constant throughout the training process.

Results in[9] demonstrated that the LR, MC and gain of the AF have a significant impact on training speed. Thimm *et al.*[10] also proved the relationship between the gain value, a set of initial weight and LR value exists. Norhamreeza *et al.*[11] demonstrated that the adaptive MC and adaptive gain of the AF significantly improved the training time.

This study proposed a further improvement on[7] where each training pattern has its own AF of neurons in the hidden layer. The AFs are adjusted by the adaptation of gain parameters together with MCs and LRs during the learning process. The proposed algorithm, back propagation gradient descent with adaptive gain, adaptive momentum and adaptive learning rate (BPGD-AGAMAL) significantly can prevent the network from trapping into local minima. The performance of the proposed algorithm will be compare with the conventional back propagation gradient descent (BPGD) and back propagation gradient descent with adaptive gain (BPGD-AG)[7]. Some simulation experiments were performed on three classification problems including iris[12], glass[13] and thyroid[14] problems and the validity of our algorithm is substantiated.

The remaining of the paper is organised as follows. In Section 2 presents the proposed algorithm. The performance of the proposed algorithm is simulated on benchmark dataset problems in Section 3. This paper is concluded in the final section.

2. The Proposed Algorithm

In this section, a further improvement on the BPGD-AG[7] for improving the learning efficiency of BP is proposed. The proposed algorithm modifies the initial search direction by changing gain value, MC and LR adaptively for each node. 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 following iterative algorithm is proposed for the batch mode of training. The weights, biases, gains, LRs and MCs are calculated and updated for the entire training set which is being presented to the network.

```
For a given epoch,
    For each input vector,
        Step 1.
        Calculate the weight and bias values using the previously converged gain, MC and LR value.
        Step 2.
        Use the weight and bias value calculated in Step (1) to calculate the new gain, MC and LR value.
    Repeat Steps (1) and (2) for each input vector and sum all the weights, biases, LR, MC and gain
    updating terms
Update the weights, biases, gains, MCs and LRs using the summed updating terms and repeat this
procedure on epoch-by-epoch basis until the error on the entire training data set reduces to a
predefined value.
```

The gain update expression for a GD algorithm is calculated by differentiating the following error term *E* with respect to the corresponding gain parameter. The network error *E* is defined as follows:

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

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

$$\Delta\, c_k = \eta\,(-\frac{\partial E}{\partial c_k}) \tag{4}$$

$$\Delta\, c_j = \eta\,(-\frac{\partial E}{\partial c_j}) \tag{5}$$

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

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

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

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

and the gain update expression for the links connecting hidden nodes is:

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

Similarly, the weight and bias expressions are calculated as follows:
The weight updates expression for the links connecting to output nodes is:

$$\Delta w_{jk}(n+1)=\eta\,(t_k-o_k)\,o_k(1-o_k)\,c_k o_j+\alpha\Delta w_{jk}(n) \tag{10}$$

Where the LR, η and MC, α are randomly generated.
Similarly, the bias update expressions for the output nodes would be:

$$\Delta\theta_k(n+1)=\eta\,(t_k-o_k)\,o_k(1-o_k)c_k+\alpha\Delta w_{jk}(n) \tag{11}$$

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

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

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

$$\Delta\theta_j(n+1)=\eta\left[\sum_k c_k w_{jk} o_k(1-o_k)\,(t_k-o_k)\right]c_j o_j(1-o_j)+\alpha\Delta w_{ik}(n) \tag{13}$$

3. Results and Discussion

The performance criterion used in this research focuses on the speed of convergence, measured in number of iterations and CPU time. Three classification problems have been tested and verified namely iris[12], glass[13] and thyroid[14] problems. The simulations have been carried out on a Pentium IV with 2 GHz HP Workstation, 3.25 GB RAM and using MATLAB version 7.10.0 (R2010a). On each problem, BPGD, BPGD-AG[7] and BPGD-AGAMAL were analysed and simulated. In order to compare those algorithms, the network parameters were kept the same. For all problems, the neural network had one hidden layer with five hidden nodes and sigmoid AF was used for all nodes. All algorithms were tested using the same initial weights which were initialised randomly from range $[0,1]$ and received the input patterns for training in the same sequence.

For all training algorithms, as the gain, MC and LR value was modified, the weights and biases were updated using the new value of gain, MC and LR. To avoid oscillations during training and to achieve convergence, an upper limit of 1 is set for the gain value. The initial value used for the gain parameter is set to 1. The MC and LR are generated in interval $[0,1]$ by using trial and error method. The best MC and LR value were selected. The number of iterations until the network converged is accumulated for each algorithm from which the mean, the standard deviation (SD) and the number of failures are calculated. For each problem, 50 different trials were run. A failure occurs when the network exceeds the iteration limit; each experiment is run to 10000 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. *Iris classification problem*

This dataset involved classifying the data of petal width, petal length, sepal width and sepal length into three classes of species which are Iris Sentosa, Iris Versicolor and Iris Verginica. The selected architecture of the network is 4-5-3 with target error was set to 0.001. The best MC and LR for BPGD and BPGD-AG are 0.4 and 0.6 respectively while BPGD-AGAMAL is initialised randomly in range $[0.1, 0.4]$ for MC and $[0.4, 0.8]$ for LR.

Table 1. Algorithm performance for Iris Classification Problem[12]

	BPGD	BPGD-AG	**BPGD-AGAMAL**
Average of Epoch	1081	721	533
Total CPU time (s) of converge	1.23×10^1	5.89	4.42
CPU time (s) / Epoch	1.14×10^{-2}	8.17×10^{-3}	8.29×10^{-3}
SD	1.4×10^2	4.09×10^2	2.45×10^2
Accuracy (%)	91.9	90.3	93.1
Failures (%)	2	0	0

Table 1 clearly show that the BPGD-AGAMAL outperformed BPGD with an improvement ratio, nearly 2.8 seconds while BPGD-AG, the BPGD-AGAMAL outperformed 1.33 seconds for the total time of converge. Furthermore, the accuracy of BPGD-AGAMAL is much better than BPGD and BPGD-AG algorithm.

3.2. *Glass classification problem*

This dataset is used for separating glass splinters into float/non-float processed building windows, vehicle windows, containers, tableware, or head lamps[13]. The architecture of the network is 9-5-6 with target error 0.001. The best MC and LR for BPGD and BPGD-AG are 0.1 and 0.1 respectively while BPGD-AGAMAL is initialised randomly in range $[0.1, 0.3]$ for MC and $[0.1, 0.2]$ for LR.

Table 2. Algorithm performance for Glass Classification Problem[13]

	BPGD	BPGD-AG	**BPGD-AGAMAL**
Average of Epoch	8613	2057	2052
Total CPU time (s) of converge	572.54	59.57	56.16
CPU time (s) / Epoch	6.65×10^{-2}	2.9×10^{-2}	2.74×10^{-2}
SD	2.15×10^3	2.45 x 10	3.12 x 10
Accuracy (%)	79.42	79.98	82.24
Failures (%)	70	0	0

Table 2 proved that the BPGD-AGAMAL still outperformed other algorithms for all performance metrics. The BPGD-AGAMAL required 2052 epochs in 56.16 seconds with 82.24% accuracy. Whereas BPGD-AG required 2057 epochs in 59.57 seconds with 79.98% accuracy. While BPGD needs 8613 epochs in 572.54 seconds and 79.42% accuracy. The results show that the average number of learning iterations for the

BPGD-AGAMAL was reduced up to 4.97 and 1.002 times faster as compared to BPGD and BPGD-AG respectively.

3.3. *Thyroid classification problem*

The dataset has 21 attributes and can be assigned to hyper-, hypo- and normal function of thyroid gland. The selected architecture of the network is 21-5-3 with target error 0.001. The best MC for conventional BPGD and BPGD-AG is 0.2 and LR is 0.5 whilst BPGD-AGAMAL is randomly initialised interval $[0.4,0.6]$ for MC and $[0.1,0.3]$ for LR value.

Table 3. Algorithm performance for Thyroid Classification Problem[13]

	BPGD	BPGD-AG	**BPGD-AGAMAL**
Average of Epoch	10000	1115	935
Total CPU time (s) of converge	1427.1	134.86	10.51
CPU time (s) / Epoch	1.43×10^{-2}	1.21×10^{-2}	1.11×10^{-2}
SD	1×10^{4}	9.12×10^{2}	9.1×10^{2}
Accuracy (%)	95	89	95.72
Failures (%)	100	0	0

From Table 3, it is worth noticing that the performance of the BPGD-AGAMAL is 90.65% faster than BPGD and almost 16.2% faster than BPGD-AG. Still, the BPGD-AGAMAL surpasses the BPGD and BPGD-AG algorithm to learn the pattern. As we can see in the Table 3, the number of success rate for the BPGD-AGAMAL and BPGD were 100% in learning the patterns. However, the BPGD did not perform well in this dataset since 100% of the simulation results failed in learning the patterns.

The results show that the BPGD-AGAMAL perform considerably better as compared to the conventional BPGD and BPGD-AG. Moreover, when comparing those algorithms, it has been empirically demonstrated that the BPGD-AGAMAL performed highest accuracy than BPGD and BPGD-AG algorithm. This conclusion enforces the usage of the proposed algorithm as an alternative training algorithm of BP algorithm.

4. Conclusion

We have proposed a further improvement on the algorithm proposed by Nazri, *et al.*[7]. In this algorithm, each training pattern has its own AF of neurons in the hidden layer. The AFs are adjusted by the adaptation of gain parameters during the learning process. These adjustments are made in order to prevent the network from trapping into a local minima caused by the neuron saturation in the hidden layer. The proposed algorithm adaptively changes the gain parameter of the AF together with MC and LR value to improve the learning efficiency. Thus, the proposed algorithm does not change the network topology and does not require additional computation. The effectiveness of the proposed algorithm has been compared with the conventional BPGD and BPGD-AG[7], verified by means of simulation on three classification problems including iris[12], glass[13] and thyroid[14] using batch mode training. Finally, the result shows that the BPGD-AGAMAL has been indicated to be very effective in avoiding the local minima with a better convergence rate and learning efficiency as compared to the conventional BPGD and BPGD-AG[7].

Acknowledgments

The authors would like to thank Universiti Tun Hussein Onn Malaysia (UTHM) for supporting this research under the Postgraduate Incentive Research Grant.

References

1. N. Mohd Nawi, R.S. Ransing, M.N. Mohd Salleh, R. Ghazali and N. Abdul Hamid, An Improved Back Propagation Neural Network Algorithm on Classification Problems, in Database Theory and Application, Bio-Science and Bio-Technology, ed. 2010), p. 177-188.
2. N. Mohd Nawi, M.N. Mohd Salleh and R. Ghazali, The Development of Improved Back-Propagation Neural Networks Algorithm for Predicting Patients with Heart Disease, in Information Computing and Applications, ed. 2010), p. 317-324.
3. Steganalysis and payload estimation of embedding in pixel differences using neural networks. (V. Sabeti, S. Samavi, M. Mahdavi and S. Shirani), Pattern Recogn. 43, 1 (2010).
4. Soft computing methods applied to the control of a flexible robot manipulator. (B. Subudhi and A.S. Morris), Applied Soft Computing. 9, 1 (2009).
5. M.A. Otair and W.A. Salameh. Speeding Up Back-Propagation Neural Networks, in Proceeding of the 2005 Informing Science and IT Education Joint Conference, eds. (Flagstaff, Arizona, USA, 2005), p. 167-173.
6. Neighborhood based Levenberg-Marquardt algorithm for neural network training. (G. Lera and M. Pinzolas), IEEE Transaction on Neural Networks. 13, 5 (2002).
7. An Improved Conjugate Gradient Based Learning Algorithm for Back Propagation Neural Networks. (N. Mohd Nawi, R.S. Ransing and M.S. Ransing), International Journal of Information and Mathematical Sciences. 4, 1 (2008).
8. Y.C. Ye, Application and Practice of the Neural Networks, edn,(Scholars Publication,Taiwan,2001).
9. The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study. (H.R. Maier and G.C. Dandy), Environmental Modelling and Software. 13, 2 (1998).
10. The interchangeability of learning rate and gain in backpropagation neural networks. (G. Thimm, P. Moerland and E. Fiesler), Neural Comput. 8, 2 (1996).
11. N. Abdul Hamid, N. Mohd Nawi and R. Ghazali. The Effect of Adaptive Gain and Adaptive Momentum in Improving Training Time of Gradient Descent Back Propagation Algorithm on Classification problems, in Proceeding of the International Conference on Advanced Science, Engineering and Information Technology 2011, eds. (Hotel Equatorial Bangi-Putrajaya, Malaysia, 2011), p. 178-184.
12. The use of multiple measurements in taxonomic problems. (R.A. Fisher), Annals of Eugenics. 7, 2 (1936).
13. Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. (R.S. Michalski and R.L. Chilausky), International Journal of Policy Analysis and Information Systems. 4:2, (1980).
14. Comparison of Multivariate Discrimination Techniques for Clinical Data - Application to The Thyroid Functional State. (D. Coomans, I. Broeckaert, M. Jonckheer and D.L. Massart), Methods of Information Medicine. 22, 2 (1983).