Accelerating Learning Performance of Back Propagation Algorithm by Using Adaptive Gain Together with Adaptive Momentum and Adaptive Learning Rate on Classification Problems

Norhamreeza Abdul Hamid, Nazri Mohd Nawi,
Rozaida Ghazali, and Mohd Najib Mohd Salleh

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,rozaida,najib}@uthm.edu.my

Abstract. The back propagation (BP) algorithm is a very popular learning approach in feedforward multilayer perceptron networks. However, the most serious problem associated with the BP is local minima problem and slow convergence speeds. Over the years, many improvements and modifications of the back propagation learning algorithm have been reported. In this research, we propose a new modified back propagation learning algorithm by introducing adaptive gain together with adaptive momentum and adaptive learning rate into weight update process. By computer simulations, we demonstrate that the proposed algorithm can give a better convergence rate and can find a good solution in early time compare to the conventional back propagation. We use two common benchmark classification problems to illustrate the improvement in convergence time.

Keywords: back propagation, convergence speed, adaptive gain, adaptive momentum, adaptive learning rate.

1 Introduction

Artificial Neural Network (ANN) is a computational paradigm which employs simplified models of the biological neurons system, though loosely based on diverse characteristics of the human brain functionality. ANN is a powerful data modelling tool which capable to capture and represent complex input-output relationships. Over the years, the application of ANN has been growing in acceptance level since it is proficient of capturing process information in a black box mode. Due to its ability to solve some problems with relative ease of use, robustness to noisily input data, execution speed and analysing complicated systems without accurate modelling in advance, ANN has successfully been implemented across an extraordinary range of problem domains that involves prediction and a wide ranging usage area in the classification problems [1-8].

Among all the diverse type of ANN architectures, Multilayer Perceptron (MLP) is the well known and the most frequently used [9]. Moreover, it is suitable for a large variety of applications [10]. In general, MLP is a Feedforward Neural Network (FNN) which is made up of sets of neurons (nodes) arranged in numerous layers. There are three distinct types of layers viz, an input layer, one or more hidden layers and an output layer. The connections between the nodes of adjacent layers relay the output signals from one layer to the subsequent layer.

ANN using the Back Propagation (BP) algorithm to perform parallel training for improving the efficiency of MLP network. It is the most popular, effective, and easy to learn model for complex, multilayered networks. A BP is a supervised learning technique which is based on the gradient descent (GD) optimisation technique that attempts to minimise the error of the network by moving down the gradient of the error curve [11]. The weights of the network are adjusted by the algorithm, thus the error is reduced along a descent direction. Unfortunately, despite the common success of BP in learning MLP network, several major drawbacks are still required to be solved. Since BP algorithm uses GD optimisation technique, the limitations comprise a slow learning convergence and easily get trapped at local minima [12-13]. We have noted that many local minima problems are closely associated with the neuron saturation in the hidden layer. When such saturation occurs, neuron in the hidden layer will lose their sensitivity to the input signals and propagation chain is blocked severely in some cases, the network can no longer learn. Additionally, the convergence behaviour of the BP algorithm also depends on the selection of network architecture, initial weights and biases, learning rate, momentum coefficient, activation function and value of the gain in the activation function.

In the last decades, a significant number of different learning algorithms have been introduced to improve the performance of BP algorithm. These involved the development of heuristic techniques, based on studies of properties of the conventional BP algorithm. These techniques include such idea as varying the learning rate, using momentum and gain tuning of activation function. Wang *et al.* [14] proposed an improved BP algorithm caused by neuron saturation in the hidden layer. Each training pattern has its own activation function of hidden nodes in order to prevent neuron saturation when the network output has not acquired the desired signals. The activation functions are adjusted by the adaptation of gain parameters during the learning process. However, this approach not performed well on the large problems and practical applications. Meanwhile, Ng *et al.* [15] localised generalisation error model for single layer perceptron neural network (SLPNN). This is an extensibility of the localised generalisation error model for supervised learning with mean squared error minimisation. On the other hand, this approach serves as the first step of considering localised generalisation error models of MLP. Ji *et al.* [16] proposed a BP algorithm that improved conjugate gradient (CG) based. In the CG algorithms, a search is performed along conjugate directions which usually lead to faster convergence compared to GD directions. Nevertheless, if it reaches a local minimum, it remains forever, as there is no mechanism for this algorithm to escape.

The learning rate value is one of the most effective means to accelerate the convergence of BP learning. Nonetheless, the size of weight adjustments needs to be done appropriately during the training process. If the chosen value of learning rate is too large for the error surface in order to speed up the training process, the network may oscillates and converges comparatively slower than a direct descent. Besides, it may cause instability to the network. Conversely, algorithm will take longer time to converge or may never converge if the learning rate value is too small. The descent takes in a very small steps, extensively increases the total of time to converge. Another effective approach to improve the convergence behaviour is by adding some momentum coefficient to the network. It can help to smooth out the descent path by preventing extreme changes in the gradient due to local anomalies [17]. Consequently, it is liable to suppress any oscillation that result from changes in the slope of the error surface. The momentum coefficient is typically chosen to be constant in the conventional back propagation algorithm with momentum. However, such a momentum with a fixed coefficient seems to speed up learning only when the current downhill gradient of the error function and the last change in weight have a similar direction. While the current negative gradient is in an opposing direction to the previous update, the momentum may cause the weight to be adjusted up the slope of the error surface instead of down the slope as desired [18]. In order to make learning more effective, the momentum should be varied adaptively rather than being fixed during the training process [19].

On the other hand, Nazri *et al.* [20] demonstrated that changing the 'gain' value adaptively for each node can significantly reduce the training time without changing the network topology. Therefore, this research proposed a further improvement on [20] by adjusting activation function of neurons in the hidden layer in each training patterns. The activation functions are adjusted by the adaptation of gain parameters together with adaptive momentum and adaptive learning rate value during the learning process. The proposed algorithm which known as 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 that caused by the neuron saturation in the hidden layer. In order to verify the efficiency of the proposed algorithm, the performance of the proposed algorithm will be compared with the conventional BP algorithm and back propagation gradient descent with adaptive gain (BPGD-AG) proposed by [20]. Some simulation experiments were performed on two classification problems including iris [21] and card [22] problems and the validity of our method is verified.

The remaining of the paper is organised as follows. In Section 2, the effect of using activation function with adaptive gain is reviewed. While in section 3 presents the proposed algorithm. The performance of the proposed algorithm is simulated on benchmark dataset problems in Section 4. This paper is concluded in the final section.

2 The Effect of Gain Parameter on the Performance of Back Propagation Algorithm

An activation function is used for limiting the amplitude of the output neuron. It generates an output value for a node in a predefined range as the closed unit interval $[0,1]$ or alternatively $[-1,1]$ which can be a linear or non-linear function. 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

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

where,

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

where,

o_j	output of the j^{th} unit.
o_i	output of the i^{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, momentum directly influences the slope of the activation function. For large gain values $(c \geq 1)$, the activation function 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.

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, and so forth. Also, a unit value for gain has generally being used for most of the research reported in the literature, though a few authors have researched the relationship of the gain parameter with other parameters which used in BP algorithms. Results in [23] demonstrate that the learning rate, momentum coefficient and gain of the activation function have a significant impact on training speed. Unfortunately, higher values of learning rate or gain may cause instability [24]. Thimm *et al.* [25] also proved that a relationship between the gain value, a set of initial weight values, and a learning rate value exists. Eom *et al.* [26] proposed a method for automatic gain tuning using a fuzzy logic system. Nazri *et al.* [20] proposed a method to change the gain value adaptively on other optimisation method such as conjugate gradient.

3 The Proposed Algorithm

In this section, a further improvement on the current working algorithm proposed by [20] for improving the training efficiency of back propagation is proposed. The proposed algorithm modifies the initial search direction by changing the three terms adaptively for each node. Those three terms are; gain value, momentum and learning rate. The advantages of using an adaptive gain value together with momentum and learning rate 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 following iterative algorithm is proposed for the batch mode of training. The weights, biases, gains, learning rates and momentum terms are calculated and update 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, momentum coefficient and learning rate value.
     Step 2.
     Use the weight and bias value calculated in Step (1) to
     calculate the new gain, momentum coefficient and learning
     rate value.
  Repeat Steps (1) and (2) for each input vector and sum all the
  weights, biases, learning rate, momentum coefficient and gain
  updating terms
Update the weights, biases, gains, momentum coefficients and
learning rates 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 the predefined value.
```

The gain, momentum and learning rate 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:

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

For output unit, $\frac{\partial E}{\partial \alpha_k}$ needs to be calculated while for hidden units, $\frac{\partial E}{\partial \alpha_j}$ is also required.

The respective momentum values would then be updated with the following equations:

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

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

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

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

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

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

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

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

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:

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

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

$$\Delta\,\theta_k=\eta\,(t_k-o_k)\,o_k(1-o_k)\alpha_k \tag{11}$$

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

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

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

$$\Delta\theta_j=\eta\left[\sum_k \alpha_k w_{jk} o_k(1-o_k)(t_k-o_k)\right]\alpha_j o_j(1-o_j) \tag{13}$$

The learning rate, η value is adjusted after every training iteration in order to attempt accelerating the leveling speed of back propagation convergence. The η for all weights are adjusted as shown in (14):

$$\eta_{i+1}=\eta_i-\frac{\partial E}{\partial \eta}=\eta_i-\frac{\partial E}{\partial w}.\frac{\partial w}{\partial \eta} \tag{14}$$

where the $\frac{\partial w}{\partial \eta}$ is the gradient of the new weight with respect to the learning rate.

4 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 have been used to verify our algorithm. Two classification problems have been tested and verified using two benchmark dataset namely iris [21] and card [22] 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, the following three algorithms were analysed and simulated.

1) The Back Propagation Gradient Descent (BPGD)
2) The Back Propagation Gradient Descent with Adaptive Gain (BPGD-AG) [20]
3) The Back Propagation Gradient Descent with Adaptive Gain, Adaptive Momentum and Adaptive Learning Rate (BPGD-AGAMAL)

To compare the performance of the proposed algorithm with conventional BPGD and BPGD-AG [20], network parameters such as network size and architecture (number of nodes, hidden layers, etc), values for the initial weights and gain parameters were kept the 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 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 value was modified, the weights and biases were updated using the new value of gain, momentum coefficient and learning rate. To avoid oscillations during training and to achieve convergence, an upper limit of 1.0 is set for the gain value. The initial value used for the gain parameter is set to one. The momentum term and learning rate is randomly generated from range $[0,1]$ by using trial and error method. The best momentum term and learning rate value were selected. For each run, the numerical data is stored in two files - the results file and the summary file. The result file lists the data about each network. 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. The networks that failed to converge are obviously excluded from the calculations of the mean and SD and were consider to be reported as failures. For each problem, 50 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 50 runs, the mean of the number of iterations (mean), the 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.

4.1 Iris Classification Problem

This dataset is a classical classification dataset made famous by Fisher, who used it to illustrate principles of discriminant analysis [21]. The classifying of Iris 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 feedforward neural network is 4-5-3 with target error was set to 0.001. The best momentum term and learning rate value for conventional BPGD and BPGD-AG for the iris dataset is 0.4 and 0.6 while BPGD-AGAMAL is initialised randomly in range $[0.1, 0.4]$ for momentum term and $[0.4, 0.8]$ for learning rate value.

Table 1. Algorithm performance for Iris Classification Problem [21]

	BPGD	BPGD-AG	**BPGD-AGAMAL**
Mean	1081	721	533
Total CPU time (s) of converge	1.23×10^1	5.89	4.26
CPU time(s)/Epoch	1.14×10^{-2}	8.17×10^{-3}	7.99×10^{-3}
SD	1.4×10^2	4.09×10^2	2.45×10^2
Accuracy (%)	91.9	90.3	93.1
Failures	1	0	0

Table 1 shows that the proposed algorithm (BPGD-AGAMAL) exhibit very good average performance in order to reach the target error. The proposed algorithm (BPGD-AGAMAL) needs only 533 epochs to converge as opposed to the conventional BPGD at about 1081 epochs while BPGD-AG needs 721 epochs to converge. Apart from speed of convergence, the time required for training the classification problem is another important factor when analysing the performance. For numerous models, training process may suppose a very important time consuming process. The results in Fig. 1 clearly show that the proposed algorithm (BPGD-AGAMAL) outperform conventional BPGD with an improvement ratio, nearly 2.9 seconds while BPGD-AG, the proposed algorithm outperformed 1.38 seconds for the total time of converge. Furthermore, the accuracy of BPGD-AGAMAL is much better than BPGD and BPGD-AG algorithm.

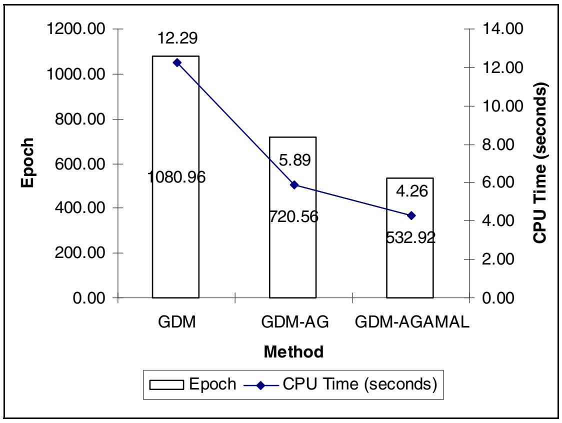

Fig. 1. Performance comparison of BPGD-AGAMAL with BPGD-AG and conventional BPGD on Iris Classification Problem

4.2 Card Classification Problem

This dataset contains all the details on the subject of credit card applications. It predicts the approval or non-approval of a credit card to a customer [22]. Descriptions of each attribute name and values are enclosed for confidentiality reason. This dataset classified whether the bank granted the credit card or not. The selected architecture of

Neural Network is 51-5-2 and the target error was set as 0.001. The best momentum term for conventional BPGD and BPGD-AG is 0.4 meanwhile the best learning rate value is 0.6. The momentum value for BPGD-AGAMAL is initialised randomly in range $[0.1, 0.4]$ and $[0.4, 0.8]$ for learning rate value.

Table 2. Algorithm performance for Card Classification Problem [22]

	BPGD	BPGD-AG	**BPGD-AGAMAL**
Mean	8645	1803	1328
Total CPU time (s) of converge	$5.47\text{x } 10^{2}$	$4.72 \text{ x } 10^{1}$	$2.2 \text{ x } 10^{1}$
CPU time(s)/Epoch	$6.33\text{x } 10^{-2}$	$2.61 \text{ x } 10^{-2}$	$1.66 \text{ x } 10^{-2}$
SD	$2.76 \text{ x } 10^{-3}$	$6.55 \text{ x } 10^{-1}$	$6.75 \text{ x } 10^{-2}$
Accuracy (%)	83.45	82.33	83.9
Failures	41	0	0

Table 2 reveals that BPGD needs 547 seconds with 8645 epochs to converge, whereas BPGD-AG needs 47.2 seconds with 1803 epochs to converge. Conversely, the proposed algorithm (BPGD-AGAMAL) performed significantly better with only 41.1 seconds and required 1328 epochs to converge. From Fig. 2, it is worth noticing that the performance of the BPGD-AGAMAL is almost 96% faster than BPGD and 53.4% faster than BPGD-AG.

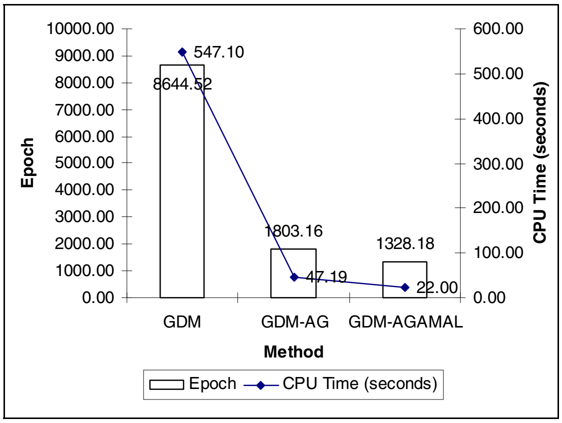

Fig. 2. Performance comparison of BPGD-AGAMAL with BPGD-AG and conventional BPGD on Card Classification Problem

The results show that the BPGD-AGAMAL perform better as compared to BPGD and BPGD-AG. Moreover, when comparing the proposed algorithm with BPGD and BPGD-AG, it has been empirically demonstrated that the proposed algorithm (BPGD-AGAMAL) performed highest accuracy than BPGD and BPGD-AG algorithm. This conclusion enforces the usage of the proposed algorithm as alternative training algorithm of BP neural networks.

5 Conclusions

Although BP algorithm is widely implemented in the most practical neural networks applications and performed relatively well, this algorithm still needs some improvements. We have proposed a further improvement on the current working algorithm proposed by Nazri *et al.* [20]. The proposed algorithm adaptively changes the gain parameter of the activation function together with momentum coefficient and learning rate to improve the learning speed. The effectiveness of the proposed algorithm has been compared with the conventional Back Propagation Gradient Descent (BPGD) and Back Propagation Gradient Descent with Adaptive Gain (BPGD-AG) [20]. The three algorithms were been verified by means of simulation on two classification problems including iris dataset with an improvement ratio nearly 2.8 seconds for the BPGD and 1.33 seconds better for the BPGD-AG in terms of total time to converge. Meanwhile, card dataset indicates almost 92.5% and 12.92% faster compared to BPGD and BPGD-AG respectively. The results show that the proposed algorithm (BPGD-AGAMAL) has a better convergence rate and learning efficiency as compared to conventional BPGD and BPGD-AG [20].

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. Nawi, N.M., Ransing, R.S., Salleh, M.N.M., Ghazali, R., Hamid, N.A.: An Improved Back Propagation Neural Network Algorithm on Classification Problems. In: Zhang, Y., Cuzzocrea, A., Ma, J., Chung, K.-i., Arslan, T., Song, X. (eds.) DTA and BSBT 2010. Communications in Computer and Information Science, vol. 118, pp. 177–188. Springer, Heidelberg (2010)
2. Nawi, N.M., Ghazali, R., Salleh, M.N.M.: The Development of Improved Back-Propagation Neural Networks Algorithm for Predicting Patients with Heart Disease. In: Zhu, R., Zhang, Y., Liu, B., Liu, C. (eds.) ICICA 2010. LNCS, vol. 6377, pp. 317–324. Springer, Heidelberg (2010)
3. Sabeti, V., Samavi, S., Mahdavi, M., Shirani, S.: Steganalysis and payload estimation of embedding in pixel differences using neural networks. Pattern Recogn. 43, 405–415 (2010)
4. Mandal, S., Sivaprasad, P.V., Venugopal, S., Murthy, K.P.N.: Artificial neural network modeling to evaluate and predict the deformation behavior of stainless steel type AISI 304L during hot torsion. Applied Soft Computing 9, 237–244 (2009)
5. Subudhi, B., Morris, A.S.: Soft computing methods applied to the control of a flexible robot manipulator. Applied Soft Computing 9, 149–158 (2009)
6. Yu, L., Wang, S.-Y., Lai, K.K.: An Adaptive BP Algorithm with Optimal Learning Rates and Directional Error Correction for Foreign Exchange Market Trend Prediction. In: Wang, J., Yi, Z., Żurada, J.M., Lu, B.-L., Yin, H. (eds.) ISNN 2006. LNCS, vol. 3973, pp. 498–503. Springer, Heidelberg (2006)

7. Lee, K., Booth, D., Alam, P.: A comparison of supervised and unsupervised neural networks in predicting bankruptcy of Korean firms. Expert Systems with Applications 29, 1–16 (2005)
8. Sharda, R., Delen, D.: Predicting box-office success of motion pictures with neural networks. Expert Systems with Applications 30, 243–254 (2006)
9. Popescu, M.-C., Balas, V.E., Perescu-Popescu, L., Mastorakis, N.: Multilayer perceptron and neural networks. WSEAS Trans. Cir. and Sys. 8, 579–588 (2009)
10. Fung, C.C., Iyer, V., Brown, W., Wong, K.W.: Comparing the Performance of Different Neural Networks Architectures for the Prediction of Mineral Prospectivity. In: Proceedings of 2005 International Conference on Machine Learning and Cybernetics, pp. 394–398 (2005)
11. Alsmadi, M.K.S., Omar, K., Noah, S.A.: Back Propagation Algorithm: The Best Algorithm Among the Multi-layer Perceptron Algorithm. International Journal of Computer Science and Network Security 9, 378–383 (2009)
12. Otair, M.A., Salameh, W.A.: Speeding Up Back-Propagation Neural Networks. In: Proceedings of the 2005 Informing Science and IT Education Joint Conference, Flagstaff, Arizona, USA, pp. 167–173 (2005)
13. Bi, W., Wang, X., Tang, Z., Tamura, H.: Avoiding the Local Minima Problem in Backpropagation Algorithm with Modified Error Function. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E88-A, 3645–3653 (2005)
14. Wang, X.G., Tang, Z., Tamura, H., Ishii, M., Sun, W.D.: An improved backpropagation algorithm to avoid the local minima problem. Neurocomputing 56, 455–460 (2004)
15. Ng, W.W.Y., Yeung, D.S., Tsang, E.C.C.: Pilot Study On The LocalizedGeneralization Error Model For Single Layer Perceptron Neural Network. In: Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian, August 13-16, pp. 3078–3082 (2006)
16. Ji, L., Wang, X., Yang, X., Liu, S., Wang, L.: Back-propagation network improved by conjugate gradient based on genetic algorithm in QSAR study on endocrine disrupting chemicals. Chinese Science Bulletin 53, 33–39 (2008)
17. Sun, Y.-J., Zhang, S., Miao, C.-X., Li, J.-M.: Improved BP Neural Network for Transformer Fault Diagnosis. Journal of China University of Mining and Technology 17, 138–142 (2007)
18. Hongmei, S., Gaofeng, Z.: A New BP Algorithm with Adaptive Momentum for FNNs Training. In: WRI Global Congress on Intelligent Systems, GCIS 2009, pp. 16–20 (2009)
19. Hamid, N.A., Nawi, N.M., Ghazali, R.: 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, pp. 178–184. Hotel Equatorial Bangi-Putrajaya, Malaysia (2011)
20. Nawi, N.M., Ransing, R.S., Ransing, M.S.: An Improved Conjugate Gradient Based Learning ALgorithm for Back Propagation Neural Networks. International Journal of Information and Mathematical Sciences 4, 46–55 (2008)
21. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of Eugenics 7, 179–188 (1936)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, San Francisco (1993)
23. Maier, H.R., Dandy, G.C.: The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study. Environmental Modelling and Software 13, 193–209 (1998)

24. Hollis, P.W., Paulos, J.J.: The effects of precision constraints in a backpropagation learning network. In: International Joint Conference on Neural Networks, IJCNN, vol. 2, p. 625 (1989)
25. Thimm, G., Moerland, P., Fiesler, E.: The interchangeability of learning rate and gain in backpropagation neural networks. Neural Comput. 8, 451–460 (1996)
26. Eom, K., Jung, K., Sirisena, H.: Performance improvement of backpropagation algorithm by automatic activation function gain tuning using fuzzy logic. Neurocomputing 50, 439–460 (2003)