An Improved Back Propagation Neural Network Algorithm on Classification Problems

Nazri Mohd Nawi[1], R.S. Ransing[2], Mohd Najib Mohd Salleh[1], Rozaida Ghazali[1], and Norhamreeza Abdul Hamid[1]

[1] Faculty of Information Technology and Multimedia
Universiti Tun Hussein Onn Malaysia
86400, Parit Raja, Batu Pahat, Johor, Malaysia
[2] Civil and Computational Engineering Centre, School of Engineering
University of Wales Swansea
Singleton Park, Swansea, SA2 8PP, United Kingdom
{nazri,najib,rozaida}@uthm.edu.my,
r.s.ransing@swansea.ac.uk, gi090007@siswa.uthm.edu.my

Abstract. The back propagation algorithm is one the most popular algorithms to train feed forward neural networks. However, the convergence of this algorithm is slow, it is mainly because of gradient descent algorithm. Previous research demonstrated that in 'feed forward' algorithm, the slope of the activation function is directly influenced by a parameter referred to as 'gain'. This research proposed an algorithm for improving the performance of the back propagation algorithm by introducing the adaptive gain of the activation function. The gain values change adaptively for each node. The influence of the adaptive 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. The efficiency of the proposed algorithm is compared with conventional Gradient Descent Method and verified by means of simulation on four classification problems. In learning the patterns, the simulations result demonstrate that the proposed method converged faster on Wisconsin breast cancer with an improvement ratio of nearly 2.8, 1.76 on diabetes problem,, 65% better on thyroid data sets and 97% faster on IRIS classification problem. The results clearly show that the proposed algorithm significantly improves the learning speed of the conventional back-propagation algorithm.

Keywords: Back-propagation Neural Networks; Gain; Activation Function; Learning Rate; Training Efficiency.

1 Introduction

A neural network is a computing system made up of a number of simple, interconnected processing neurons or elements, which process information by its dynamic state response to external inputs [1]. The development and application of neural networks are unlimited as it spans a wide variety of fields. This could be attributed to the fact that these networks are attempts to model the capabilities of human. It had successfully

Y. Zhang et al. (Eds.): DTA/BSBT 2010, CCIS 118, pp. 177–188, 2010.

implemented in the real world application which are accounting and finance [2,3], health and medicine [4,5], engineering and manufacturing [6,7], marketing [8,9] and general applications [10,11,12]. Most papers concerning the use of neural networks have applied a multilayered, feed-forward, fully connected network of perceptions [13,14]. Reasons for the use of simple neural networks are done by the simplicity of the theory, ease of programming, good results and because this type of NN represents an universal function in the sense that if the topology of the network is allowed to vary freely it can take the shape of any broken curve [15]. A standard multi-layer feed forward neural network has an input layer of neurons, a hidden layer of neurons and an output layer of neurons. Every node in a layer is connected to other node in the adjacent forward layer. Several types of learning algorithm have been used in the literature. However, back-propagation algorithm is the most popular, effective, and easy to earn model for complex, multilayered networks. This algorithm is used more than all other combined and used in many different types of applications [16]. A back-propagation is a supervised learning technique that uses a gradient descent rule which attempts to minimize the error of the network by moving down the gradient of the error curve [1]. When using the back-propagation to train a standard multi-layer feed forward neural network, the designer is required to arbitrarily select parameters 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. Improper selection 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, can easily get stuck at local minima. Hence, improving the application of back-propagation remains an important research issue.

In recent years, a number of research studies have attempted to overcome these problems. These involved the development of heuristic techniques, based on studies of properties of the conventional back-propagation algorithm. These techniques include such idea as varying the learning rate, using momentum and gain tuning of activation function. Perantonis et al. [17] proposed an algorithm for efficient learning in feed forward neural networks using momentum acceleration. Nevertheless, it could be found that more training time is spent on the computation of constrained conditions in the algorithm. Kamarthi and Pittner [18] presented a universal acceleration technique for the back-propagation algorithm based on extrapolation of each individual interconnection weight. This requires the error surface to have a smooth variation along the respective axes, therefore extrapolation is possible. For performing extrapolation, at the end of each epoch, the converge behaviour of each network weight in back-propagation algorithm is individually examined. They also focused on the use of standard numerical optimization techniques. Though, this technique often must be tuned to fit a particular application. Møller [19] explained how conjugate gradient algorithm could be used to train multi-layer feed forward neural networks. In this algorithm a search is performed along conjugate directions, which generally leads to faster convergence than steepest gradient descent directions. The error function is guaranteed not to increase consequently of the weights update. However, if it reaches a local minimum, it remains forever, as there is no mechanism for this algorithm to escape. Lera et al. [20] described the use of Levenberg-Marquardt algorithm for training multi-layer feed forward neural networks. Though, the training times required strongly depend on neighbourhood size.

Nazri et al. [21] demonstrated that changing the 'gain' value adaptively for each node can significantly reduce the training time. Based on [21], this paper proposed an improved algorithm that will change the gain value adaptively which significantly improve the performance of the back-propagation algorithm. In order to verify the efficiency of the proposed algorithm, and to compare it with the Gradient Descent Method (GDM) proposed by Nazri et al. [21], some simulation experiments was performed on four benchmark classification problems including Wisconsin breast cancer [22], diabetes [23], thyroid [24] and IRIS [25].

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

2 Using Activation Function with Adaptive Gain

An activation function is used for limiting the amplitude of the output of 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

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

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, directly influences the slope of the activation function. For large gain values $(c \le 1)$, the activation function approaches a 'step function' whereas for small gain values $(0 < c \le 1)$ 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. 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 which used in back-propagation algorithms. The recent results [27] show that learning rate, momentum constant and gain of the activation function have a significant impact on training speed. Unfortunately, higher values of learning rate and/or gain cause instability [28]. Thimm et al. [29] also proved that a relationship between the gain value, a set of initial weight values, and a learning rate value exists. Looney [30] 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. [31] proposed a method for automatic gain tuning using a fuzzy logic system. Nazri et al. [21] proposed a method to change adaptively gain value on other optimisation method such as conjugate gradient.

3 The Proposed Algorithm

In this section, an improved algorithm for improving the training efficiency of back-propagation is proposed. The proposed algorithm modifies the initial search direction by changing the gain value adaptively for each node. The following subsection describes the proposed 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.

There are two different ways in which this gradient descent can be implemented which are incremental mode and batch mode. In this paper, the batch mode was used for training process. As in batch mode training, the weights, biases and gains are updated after one complete presentation of the entire training set. An epoch is defined as one complete presentation of the 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.

3.1 Algorithm

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

```
For a given epoch,
   For each input vector,
       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 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 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.

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

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

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

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

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

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

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

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

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

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

$$\Delta w_{jk} = \eta\left(t_k - o_k\right) o_k\left(1 - o_k\right) c_k o_j \tag{10}$$

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

$$\Delta \theta_k = \eta\left(t_k - o_k\right) o_k\left(1 - o_k\right) c_k \tag{11}$$

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

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

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

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

3.2 Advantages of Using Adaptive Gain

An algorithm has been proposed in this paper for the efficient calculation of the adaptive gain value in 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 [29]. 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.

4 Results and Discussion

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 Wisconsin breast cancer [22], diabetes [23], thyroid [24] and IRIS [25]. 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 (GDM) [21]
2) The proposed Gradient descent with momentum and Adaptive Gain (GDM/AG)

To compare the performance of the proposed algorithm with gradient descent method [21], 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, 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 is 0.3 and the momentum term value is 0.7. The initial value used for the gain parameter is 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.

4.1 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 [22]. This problem tries to diagnosis of breast cancer by trying to classify a tumor as

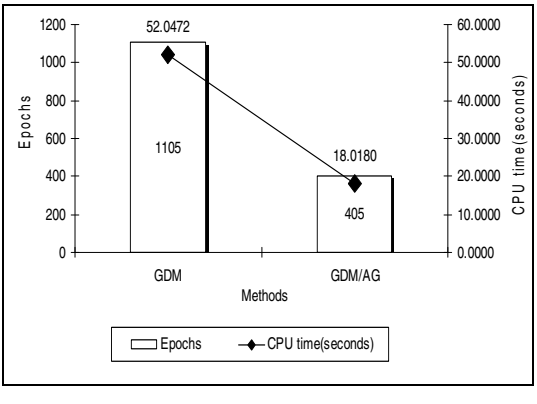

Fig. 1. Performance comparison of GDM/AG with GDM for Breast Cancer Classification Problem

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.

Figure 1 shows that the proposed algorithm (GDM/AG) exhibit very good average performance in order to reach target error. Furthermore the number of failures for the proposed algorithm is smaller as compared to GDM. The proposed algorithm (GDM/AG) needs only 405 epochs to converge as opposed to the standard GDM at about 1105 epochs. Apart from speed of convergence, the time required for training the classification problem is another important factor when analyzing the performance. For numerous models, training process may suppose a very important time consuming process. The results in Figure 1 clearly show that the proposed algorithm (GDM/AG) outperform GDM with an improvement ratio, nearly 2.8, for the total time of converge.

4.2 IRIS Classification Problem

This is a classical classification dataset made famous by Fisher [25], 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

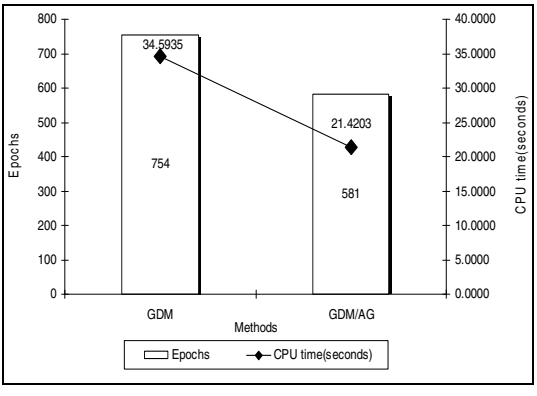

Fig. 2. Performance comparison of GDM/AG with GDM for IRIS Classification Problem

Figure 2 shows that the proposed algorithm (GDM/AG) still outperforms other algorithms in term of CPU time and number of epochs. The proposed algorithm (GDM/AG) only required 581 epochs in 21.4203 seconds CPU times to achieve the target error, whereas GDM required 754 epochs in 34.5935 seconds CPU times. As we can see that the number of success rate for the proposed algorithm (GDM/AG) was 97% as compared to GDM in learning the patterns. Furthermore, the average number of learning iterations for the proposed algorithm was reduced up to 1.3 times faster as compared to GDM.

4.3 Diabetes Classification Problem

This dataset was created based on the 'Pima Indians Diabetes' problem dataset [23] 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.

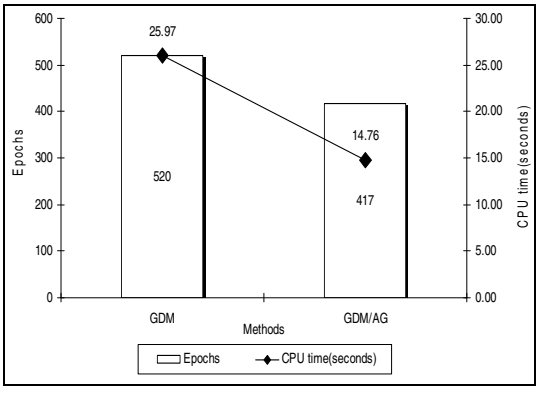

Fig. 3. Performance comparison of GDM/AG with GDM for Diabetes Classification Problem

From Figure 3, it is worth noticing that the performance of the proposed GDM/AG is almost 1.8 faster as compared to GDM. GDM/AG took only 417 epochs to reach the target error as compared to GDM which took about 520 epochs to converge. Still the proposed algorithm outperform the traditional GDM in term of the total time of converge.

4.4 Thyroid Classification Problem

This dataset was created based on the 'Thyroid Disease' problem dataset [24] from the UCI repository of machine learning database. The dataset was obtained from the Garvan Institute. This dataset deals with diagnosing a patient thyroid function. Each pattern has 21 attributes and can be assigned to any of three classes which were hyper-, hypo- and normal function of thyroid gland. The selected architecture of the Feed-forward Neural Network is 21-5-3. The target error is set to 0.05.

Figure 4 reveal that GDM need 316.4905 seconds with 3441 epochs to converge. However, the proposed algorithm (GDM/AG) performed significantly better with only 111.631 seconds and 1114 epochs to converge. The result shown that the GDM/AG perform better as compared to GDM.

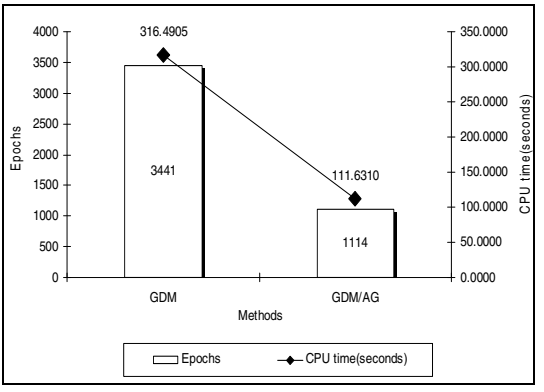

Fig. 4. Performance comparison of GDM/AG with GDM for Thyroid Classification Problem

Consequently, from the previous results, we can claim that in general, the proposed algorithm (GDM/AG) presents better performance than GDM. The experimental results from four classification problems allow to compares the proposed algorithm (GDM/AG) with the traditional GDM, in terms of speed of convergence, measured in number of iterations and CPU time. Moreover, when comparing the proposed algorithm with GDM, it has been empirically demonstrated that the proposed algorithm (GDM/AG) performed highest accuracy than GDM. This conclusion enforces the usage of the proposed algorithm as alternative training algorithm of back propagation neural networks.

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. It changes the gain value adaptively for each node. The effectiveness of the proposed algorithm has been compared with the Gradient Descent Method (GDM) [21], verified by means of simulation on four classification problems including Wisconsin breast cancer with an improvement ratio nearly 2.8 for the total time of converge, diabetes almost 1.76 faster respectively, thyroid took almost 65% less time to converge and IRIS the proposed algorithm outperformed the traditional GDM with 97% faster in learning the patterns.

Acknowledgment

The authors would like to thank the Ministry of Science, Technology and Innovation, Malaysia for supporting this research under the Science Fund Research Grant.

References

1. Majdi, A., Beiki, M.: Evolving Neural Network Using Genetic Algorithm for Predicting the Deformation Modulus of Rock Masses. International Journal of Rock Mechanics and Mining Science, vol 47(2), 246–253 (2010)
2. 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), 1–16 (2005)
3. Landajo, M., Andres, J.D., Lorca, P.: Robust Neural Modeling for the Cross-Sectional Analysis of Accounting Information. European Journal of Operational Research 177(2), 1232–1252 (2007)
4. Razi, M.A., Athappily, K.: A Comparative Predictive Analysis of Neural Networks (NNs), Nonlinear Regression and Classification and Regression Tree (CART) Models. Expert Systems with Applications 2(1), 65–74 (2005)
5. Behrman, M., Linder, R., Assadi, A.H., Stacey, B.R., Backonja, M.M.: Classification of Patients with Pain Based on Neuropathic Pain Symptoms: Comparison of an Artificial Neural Network against an Established Scoring System. European Journal of Pain 11(4), 370–376 (2007)
6. Yesilnacar, E., Topal, T.: Landslide Susceptibility Mapping: A Comparison of Logistic Regression and Neural Networks Methods in a Medium Scale Study, Hendek region (Turkey). Engineering Geology 79(3–4), 251–266 (2005)
7. Dvir, D., Ben-Davidb, A., Sadehb, A., Shenhar, A.J.: Critical Managerial Factors Affecting Defense Projects Success: A Comparison between Neural Network and Regression Analysis. Engineering Applications of Artificial Intelligence 19, 535–543 (2006)
8. Gan, C., Limsombunchai, V., Clemes, M., Weng, A.: Consumer Choice Prediction: Artificial Neural Networks Versus Logistic Models. Journal of Social Sciences 1(4), 211–219 (2005)
9. Chiang, W.K., Zhang, D., Zhou, L.: Predicting and Explaining Patronage Behavior toward Web and Traditional Stores Using Neural Networks: A Comparative Analysis with Logistic Regression. Decision Support Systems 41, 514–531 (2006)
10. Chang, L.Y.: Analysis of Freeway Accident Frequencies: Negative Binomial Regression versus Artificial Neural Network. Safety Science 43, 541–557 (2005)
11. Sharda, R., Delen, D.: Predicting box-office success of motion pictures with neural networks. Expert Systems with Applications 30, 243–254 (2006)
12. Nikolopoulos, K., Goodwin, P., Patelis, A., Assimakopoulos, V.: Forecasting with cue information: A comparison of multiple regression with alternative forecasting approaches. European Journal of Operational Research 180(1), 354–368 (2007)
13. Curteanu, S., Cazacu, M.: Neural Networks and Genetic Algorithms Used For Modeling and Optimization of the Siloxane-Siloxane Copolymers Synthesis. Journal of Macromolecular Science, Part A 45, 123–136 (2007)
14. Lisa, C., Curteanu, S.: Neural Network Based Predictions for the Liquid Crystal Properties of Organic Compounds. Computer-Aided Chemical Engineering 24, 39–45 (2007)
15. Fernandes, Lona, L.M.F.: Neural Network Applications in Polymerization Processes. Brazilian Journal Chemical Engineering 22, 323–330 (2005)
16. Mutasem, K.S.A., Khairuddin, O., Shahrul, A.N.: Back Propagation Algorithm: The Best Algorithm Among the Multi-layer Perceptron Algorithm. International Journal of Computer Science and Network Security 9(4), 378–383 (2009)

17. Perantonis, S.J., Karras, D.A.: An Efficient Constrained Learning Algorithm with Momentum Acceleration. Neural Networks 8(2), 237–249 (1995)
18. Kamarthi, S.V., Pittner, S.: Accelerating Neural Network Training using Weight Extrapolations. Neural Networks 12, 1285–1299 (1999)
19. Møller, M.F.: A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. Neural Networks 6(4), 525–533 (1993)
20. Lera, G., Pinzolas, M.: Neighborhood based Levenberg-Marquardt Algorithm for Neural Network Training. IEEE Transaction on Neural Networks 13(5), 1200–1203 (2002)
21. Nawi, N.M., Ransing, M.R., Ransing, R.S.: An Improved Conjugate Gradient Based Learning Algorithm for Back Propagation Neural Networks. International Journal of Computational Intelligence 4(1), 46–55 (2007)
22. Mangasarian, O.L., Wolberg, W.H.: Cancer Diagnosis via Linear Programming. SIAM News 23(5), 1–18 (1990)
23. Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S.: Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. In: Proceedings of the Symposium on Computer Applications and Medical Care, pp. 261–265. IEEE Computer Society Press, Los Alamitos (1988)
24. Coomans, D., Broeckaert, I., Jonckheer, M., Massart, D.L.: Comparison of Multivariate Discrimination Techniques for Clinical Data—Application to The Thyroid Functional State. Methods of Information Medicine 22, 93–101 (1983)
25. Fisher, R.A.: The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics 7, 179–188 (1936)
26. Holger, R.M., Graeme, C.D.: The Effect of Internal Parameters and Geometry on the Performance of Back-Propagation Neural Networks. Environmental Modeling and Software 13(1), 193–209 (1998)
27. Hollis, P.W., Harper, J.S., Paulos, J.J.: The Effects of Precision Constraints in a Backpropagation Learning Network. Neural Computation 2(3), 363–373 (1990)
28. Thimm, G., Moerland, F., Fiesler, E.: The Interchangeability of Learning Rate and Gain in Backpropagation Neural Networks. Neural Computation 8(2), 451–460 (1996)
29. Looney, C.G.: Stabilization and Speedup of Convergence in Training Feed Forward Neural Networks. Neurocomputing 10(1), 7–31 (1996)
30. 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)