International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 07 – Issue 02, March 2018

Comparative Performance of Several Recent Supervised Learning Algorithms

Tony Bazzi*
Department of Electrical and Systems Engineering
Oakland University
Rochester, MI 48309, USA
**Email: tbazzi [AT] oakland.edu*

Rana Ismail
Optimal Medical Center
Dearborn, MI, 48124

Mohamed Zohdy
Department of Electrical and Systems Engineering
Oakland University
Rochester, MI 48309, USA

Abstract— **A wide variety of optimization algorithms have been developed, however their performance is still unclear across optimization landscapes. The manuscript presented herein discusses methods for modeling and training neural networks on a small dataset. The algorithms include conventional gradient descent, Levenberg-Marquardt, Momentum, Nesterov Momentum, ADAgrad, and RMSprop learning methodologies. The work aims to compare the performance, efficiency, and accuracy of the different algorithms utilizing the fertility dataset available through the UC Irvine machine learning repository.**

Keywords: Neural Networks, Back Propagation, Lebenberg-Marquardt, Momentum, Nesterov, ADAgrad, RMSprop, Hyperparameters, Newton Method, Supervised Learning, gradient descent, delta rule, Python, Fertility

I. Introduction

Background

An artificial neural network is a supervised machine learning algorithm used by computers to model complex high dimensional datasets and make predictions without being explicitly programmed. Neural networks are biologically inspired through mimicking neurons operation in a human brain. The first neural network was modeled in 1957 by Frank Rosenblatt and was referred to as perceptron. Lately, neural networks have been gaining popularity through deep learning such as recurrent and convolution networks for speech recognition and autonomous applications. Several back-propagation learning algorithms were developed to assist with the convergence speed and performance of the networks. Algorithms such as the momentum back propagation algorithm has been widely used in neurocomputing [1][2] and several adaptive methodologies were then produced to dynamically vary the momentum parameter as described in [3][4][5][6][7]. Nesterov's momentum is one of the adaptive algorithms that gained popularity where it was developed by Yuri Nesterov in 1983 [8]. Levenberg's algorithm is another back propagation where it combines Gauss-Newton algorithm (GNA) with the conventional gradient descent published in 1944. Levenberg's contribution modifies the GNA to include a hyperparameter μ allowing the learning to increase in the initial phases of training and decreases when the solution tends towards convergence depending on the rate of change of the cost function s(w) [9]. On the other hand, Marquardt's extension in 1963 suggests scaling each component of the gradient per the curvature so there are large movement where the rate of change of the cost function is smaller [10]. In 2012, an adaptive gradient method or ADAgrad that incorporates knowledge of the geometry of the data being observed was proposed by Duchi et al. [11]. Root mean square propagation or RMSprop is a modified version of the ADAgrad that attempts to reduce its aggressiveness while monotonically decreasing the learning rate. The latter method was presented by Geoff Hinton in his Coursera class. While the RMSprop is an unpublished training algorithm but it is being adapted by the neurocomputing community, hence its addition to our scope of work in this manuscript. Numerous training algorithms were presented, discussed, and utilized in artificial neural networks with one hidden layer on one front and deep learning neural networks such as recurrent and convoluted networks on the other front. Comparison of neural network back-propagation algorithms have been carried out in the literature addressing specific case studies such as stream flow forecasting, determination of lateral stress in cohesion-less soils, electricity load forecasting, radio network planning tool, power quality monitoring, software defect prediction, sleep disorders, and electro-static precipitation for air quality control [12][13][14][15][16][17][18]. However, none of them compare and study the effectiveness of the algorithms covered in our work especially when modeling complexities in high dimensional small datasets. To bridge the gap, we employ the fertility database from UC Irvine machine learning repository to compare the performance and the effectiveness of conventional gradient descent, Levenberg-Marquardt, Momentum, Nesterov Momentum, ADAgrad, and RMSprop learning methods through studying the mean square error propagation versus the number of iterations and visually comparing actual versus predicted observations for our labeled dataset. The fertility dataset ivolves analyzing semen samples from 100 volunteers per the 2010 world health organization criteria such as socio-demographic data, environmental factors, health status, and life habits. The neural network along with the different training algorithms presented in our scope of work are programmed in Python version 2.7 utilizing the Spyder 2.3.8 development environment employing the "numpy" fundamental package for scientific computing. It is important to mention that all the algorithms and the neural network were developed and programmed without using generic Python scripts or APIs such as Scipy and Kerras.

II. THEORY

Neural Network

The end design of the neural network is illustrated in figure 1. The network receives an input matrix with 9 features and 100 data point hence our small data set criteria. The hidden layer includes 9 nodes while the output layer includes a single node. The hidden nodes are activated with hyperbolic functions while the output node or fertility predictor is a pure linear activation function. The choice of a single hidden layer is a result of the smaller number of attributes. Nonetheless, Hornik [19] showed that given enough hidden units, a single layer feedforward network can approximate any continuous function described in [18].

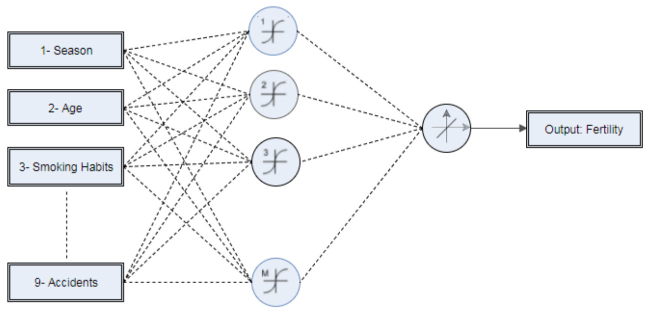

Figure 1: Neural interpretation diagram for the network used.

Training Algorithms

The process of tuning the neural network parameters or updating the weights to emulate the characteristics of an input-output pattern is performed by the following back propagation algorithms.

Gradient Descent

The gradient descent back propagation training algorithm is commonly referred to as the MIT or Delta rule. Suppose we have a cost function $J(\theta)$ where θ depicts the weights components, the delta rule performs a gradient search while updating and tuning the parameters to minimize the cost function. The gradient descent rule is described mathematically per the following formula.

$$\Delta\theta_t = -\gamma\frac{\partial J}{\partial\theta} = -\gamma\left(\frac{\partial s(\theta)}{\partial\theta}\right)^T$$
$$\theta_{new} = \theta_{old} + \Delta\theta_t$$
$$\gamma = \text{ Learning rate} \qquad (1)$$
$$\theta = \text{ Weights}$$
$$J = \text{ Cost function}$$
$$\text{t = Number of iteration}$$

Adaptive Momentum Back-Propagation

The momentum back-propagation algorithm for training neural network has been widely used in neuro-computing [21][22]. The convergence behavior has also been analyzed in [20][23][24][25][26]. It was first suggested that the momentum coefficient should remain constant but it has been observed that it should be changed dynamically throughout the training of the network hence the nomenclature BPAM (back-propagation with adaptive momentum) [27]. The algorithm is similar in nature to the gradient descent but involves an additional hyper-parameter μ that starts initially with a typical value of 0.5 and then progressively annealed as the number of iterations increases. The mathematical formulation used programmed for our work is in accordance with the following equations.

$$v_t = -\gamma\frac{\partial J}{\partial\theta} + \mu_{t-1} * v_{t-1}$$
$$\Delta\theta_t = \Delta\theta_{t-1} + v_t$$
$$\theta_{new} = \theta_{old} + \Delta\theta_t$$
$$\gamma = \text{ Learning rate} \qquad (2)$$
$$\theta = \text{ Weights}$$
$$J = \text{ Cost function}$$
$$\mu = \text{ Momentum Hyperparameter}$$
$$\text{v= Velocity parameter}$$

Nesterov's Momentum

Nesterov's adaptive gradient is a first order optimization technique that helps the stability and convergence of gradient descent [28]. The algorithm is computed as follows.

$$v_t = -\gamma\frac{\partial J}{\partial\theta} + \mu_{t-1} * v_{t-1}$$
$$\theta_{new} = \theta_{old} - \mu_{t-1}v_{t-1} + (1+\mu) * v$$
$$\gamma = \text{ Learning rate} \qquad (3)$$
$$\theta = \text{ Weights}$$
$$J = \text{ Cost function}$$
$$\mu = \text{ Momentum Hyperparameter}$$
$$\text{v= Velocity parameter}$$

Adaptive Gradient Learning Algorithm

Duchi et al. presented sub gradient methods that dynamically incorporate knowledge of the geometry of the data in earlier iterations for more informative back propagation learning [11]. The algorithm is more suitable for sparse data and it is used to train large scale neural networks such as the work presented by Pennington et al.for Glove word embeddings and Google for detecting images of cats in youtube videos [29] [30]. The idea about ADAgrad is that it performs larger updates for infrequent and smaller updates for frequent parameters. In our work, we aim to test the algorithm on a small database to gauge its performance and accuracy.

$$\Delta\theta_t = \frac{-\gamma\frac{\partial J}{\partial\theta}}{\sqrt{G_t+\varepsilon}}$$

$$G_t = \left(\frac{\partial J}{\partial\theta}\right)^2$$

$$\theta_{new} = \theta_{old} + \Delta\theta_t$$

$\gamma =$ Learning rate (4)

$\theta =$ Weights

$J =$ Cost function

t = Number of iteration

$\varepsilon =$ Tuning parameter to prevent division by Zero

Root Mean Squared Propagation

Geoffrey Hinton in his Coursera lectures introduces an unpublished method to train neural networks called RMSprop. RMSprop modifies the ADAgrad method reducing its aggressive monotonically decreasing learning rate by employing a moving average of the squared gradients Gt,ii.

$$\Delta\theta_t = \frac{-\gamma\frac{\partial J}{\partial\theta}}{\sqrt{G_t+\varepsilon}}$$

$$G_t = 0.9 * G_{t-1} + 0.1 * \left(\frac{\partial J}{\partial\theta}\right)^2$$

$$\theta_{new} = \theta_{old} + \Delta\theta_t$$

$\gamma =$ Learning rate (5)

$\theta =$ Weights

$J =$ Cost function

t = Number of iteration

$\varepsilon =$ Tuning parameter to prevent division by Zero

Gauss-Newton Algorithm

The Gauss-Newton algorithm or GNA is a modification of Newton's algorithm without having the complexity of calculating the second derivatives. The methodology involves solving nonlinear least square problems to minimize the sum of squared of our cost function $J(\theta)$ or $s(\theta)$ per the following formulation.

$$\mathbf{e}(\theta) = z - g(\theta) = 0$$

Expanding $e(\theta)$ by Taylor series to:

$$\mathbf{e}(\theta) = z - \left(\mathbf{g}(\theta)\Big|_{\theta=\theta_0} + \frac{\partial(\theta)}{\partial\theta}\Big|_{\theta=\theta_0} + h.o.t\right) = 0$$

Set $\mathbf{y} = \mathbf{z} - \mathbf{g}(\theta)\big|_{\theta=\theta_0}$ and $\mathbf{A} = \frac{\partial\mathbf{g}(\theta)}{\partial\theta}$ or Jacobian matrix gives:

$$\mathbf{e}(\theta) = y - A\theta \approx 0 \qquad (6)$$

We want to find θ so the sum of squared error is minimized:

$$\mathrm{J}(\theta) = s(\theta) = e(\theta)^T e(\theta) = (y - A\theta)^T (y - A\theta)$$

Perturbing $s(\theta)$ and appyling partial derivation with respect to θ yields:

$$s(\theta+\Delta\theta) = (\mathbf{y} - \mathbf{A}(\theta+\Delta\theta))^T (\mathbf{y} - \mathbf{A}(\theta+\Delta\theta)) = (\mathbf{y} - \mathbf{A}\theta - \mathbf{A}\Delta\theta)^T (\mathbf{y} - \mathbf{A}\theta - \mathbf{A}\Delta\theta)$$

$$\frac{\partial s(\theta+\Delta\theta)}{\partial\Delta\theta} = \frac{\partial(\mathbf{y} - \mathbf{A}\theta - \mathbf{A}\Delta\theta))^T (\mathbf{y} - \mathbf{A}\theta - \mathbf{A}\Delta\theta)}{\partial\Delta\theta} = \left(2(\mathbf{y} - \mathbf{A}\theta - \mathbf{A}\Delta\theta))^T\right)(-\mathbf{A}) = 0$$

$$\Delta\theta = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T(\mathbf{y} - \mathbf{A}\theta)$$

Levenberg's Contribution

Levenberg modifies the Gauss-Newton algorithm to include the parameter μ multiplied by an identity matrix in the following manner.

$$\Delta\theta = \left(\mathbf{A}^T\mathbf{A} + \mu I\right)^{-1}\mathbf{A}^T\left(\mathbf{y} - \mathbf{A}\theta\right) \qquad (7)$$

When our cost function J decreases rapidly, μ is set to a small value where our weights update $\Delta\theta$ becomes $\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\left(\mathbf{y} - \mathbf{A}\theta\right)$ which is the GNA. On the other hand, if the cost function decreases slowly μ is set to a large value and the weight update $\Delta\theta$ becomes $\left(\mu I\right)^{-1}\mathbf{A}^T\left(\mathbf{y} - \mathbf{A}\theta\right)$ or the traditional gradient descent illustrated by $\frac{\partial J}{\partial\theta}$ transposed.

Marquardt's Contribution

Marquardt suggests replacing the identity matrix by the diagonal matrix consisting of the diagonal elements of $\left(\mathbf{A}^T\mathbf{A}\right)$. The latter mitigates slow convergence in the direction of small gradients hence the following Levenberg-Marquardt's algorithm.

$$\Delta\theta = \left(\mathbf{A}^T\mathbf{A} + \mu.diag(\mathbf{A}^T\mathbf{A}) + \mu_0.I\right)^{-1}\mathbf{A}^T\left(\mathbf{y} - \mathbf{A}\theta\right) \qquad (8)$$

$\mu_0.I$ prevents the singularity condition of $\mathbf{A}^T\mathbf{A}$

III. RESULTS

The neural network parameters used in our work include nine input, nine hidden, and 1 output node. The learning rate is set to 0.4 for all cases while the momentum parameter is set to 0.4 for both Nesterov and regular momentum methods. On the other hand, for the Levenberg-Marquardt algorithm, the parameter μ is initialized to 0.001, 0.1 decrease factor, and 10 increase factor. The weights matrices initialization is constant for all cases. The training stops when the mean squared error reaches 0.0001 for all cases. The results are presented visually in figures 2 through 7 while table 1 provides a summary comparing the models in terms of performance and accuracy. The graphical representation includes two plots where the first subplot shows the mean square error propagation versus the number of iterations while the second subplot includes the neural network predicted outputs and the dataset actual outputs.

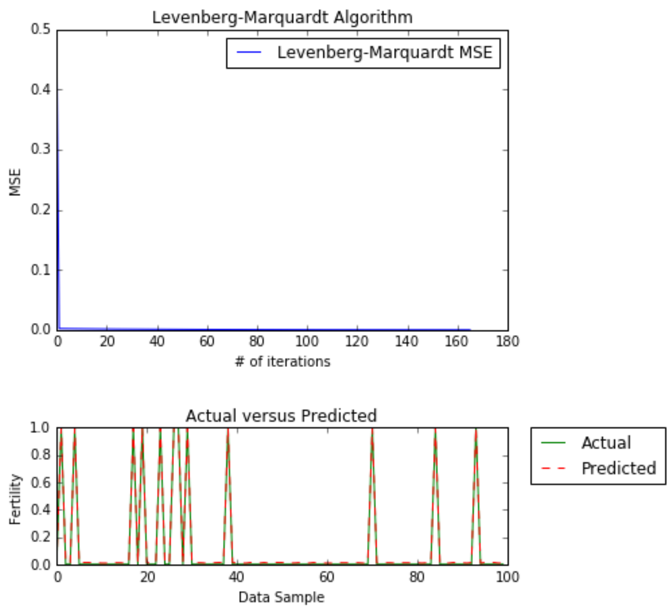

Figure 2: Levenberg-Marquardt algorithm implementation

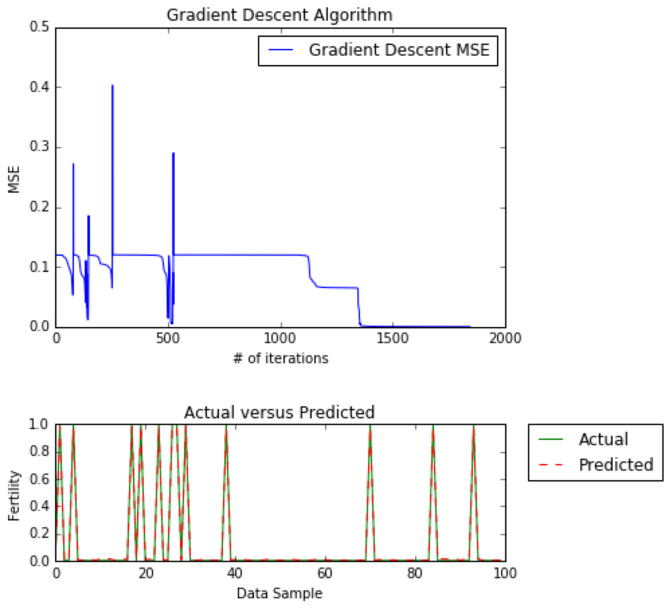

Figure 3: Gradient Descent algorithm implementation

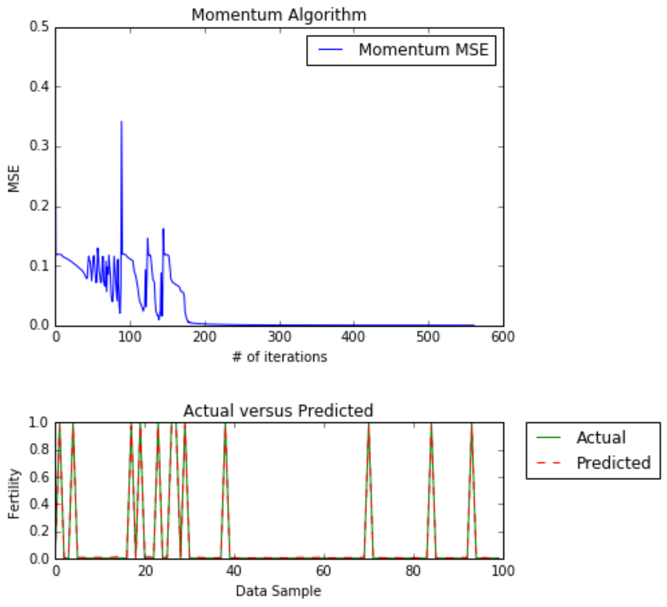

Figure 4: Momentum algorithm implementation

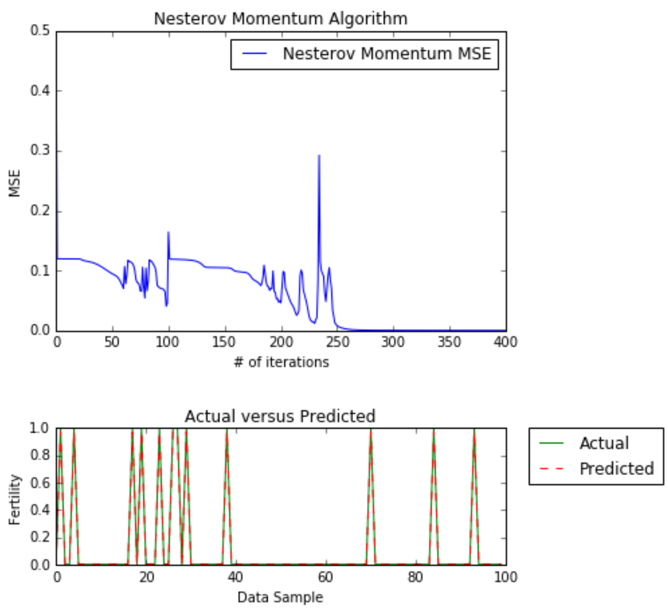

Figure 5: Nesterov Momentum algorithm implementation

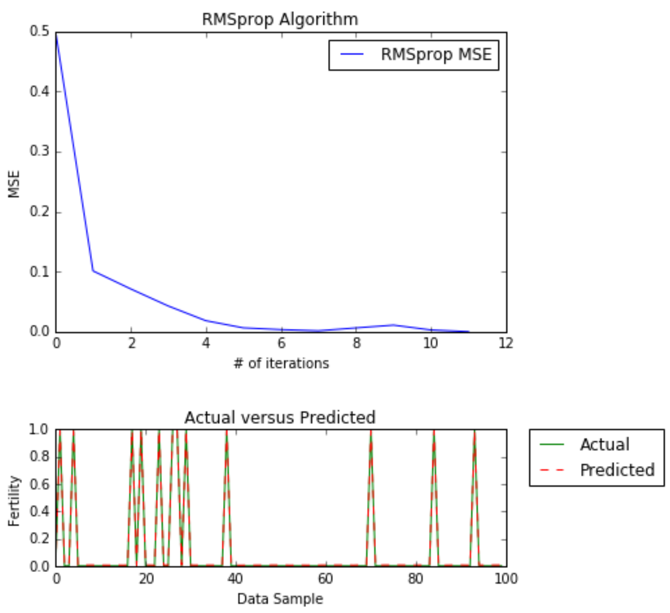

Figure 7: RMSprop algorithm implementation

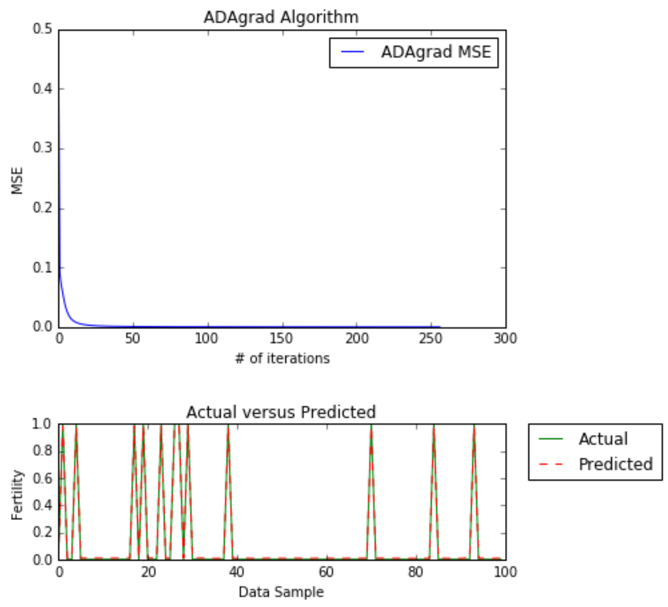

Figure 6: ADAgrad algorithm implementation

Algorithm	**Convergence (#of iterations)**	**Accuracy (%)**
Gradient Descent	1844	100%
Momentum	562	100%
Nesterov Momentum	401	100%
ADAgrad	257	100%
Levenberg-Marquardt	165	100%
RMSprop	11	100%

Table1 summarizes the convergence speed in number of iteration and the performance of the model in terms of percent accuracy

IV. CONCLUSION

The work performed in this manuscript suggests that all learning models sustained a 100% accuracy. The RMSprop algorithm was the fastest algorithm and attained convergence in 11 iterations compared to 165 iterations for the Levenberg-Marquardt algorithm ranked second. The RMSprop algorithm is more efficient costing less computing power than the Levenberg-Marquardt due to having to perform matrix inversion in the latter. In addition, the RMSprop algorithm reduces the number of hyper-parameters compared to the LM methodology. Although our dataset is considered small but it does provide a good benchmarking tool when considering large scale datasets or deep learning such as convolution neural networks. The RMSprop,ADAgrad, and LM algorithms convergence were more stable as they progressed towards the global minimum compared with the gradient descent, conventional momentum, and Nesterov's momentum methodologies.

REFERENCES

[1] D.E. Rumelhart, J.L. McClelland, the PDP Research Group, Parall Distributed Processing—Explorations in the Microstructure of Cognition, MIT Press, MA, 1986.

[2] M.R. Meybodi, H. Beigy, A note on learning automata-based schemes for adaptation of BP parameters, Neurocomputing 48 (2002) 957–974.] S. J. Miller, *The method of least squares*, Mathematics Department, Brown University..

[3] L.W. Chan, F. Fallside, An adaptive training algorithm for backpropagation networks, Computer Speech and Language 2 (1987) 205–218.

[4] G. Qiu, M.R. Varley, T.J. Terrell, Accelerated training of backpropagation networks by using adaptive momentum step, IEE Electronics Letters 28 (4)(1992) 377–379.

[5] X. Yu, N.K. Loh, W.C. Miller, A new acceleration technique for the backpropagation algorithm, in: IEEE International Conference on Neural Networks, vol. 3, 1993, pp. 1157–1161.

[6] E. Istook, T. Martinez, Improved backpropagation learning in nerural networks with windowed momentum, International Journal of Neural System 12 (3–4) (2002) 303–318.

[7] C. Yu, B. Liu, A backpropagation algorithm with adaptive learning rate and momentum coefficient, Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN'02), vol. 2, 2002, pp. 1218–1223.

[8] Conference on Neural Networks (IJCNN'02), vol. 2, 2002, pp. 1218–1223. H.M. Shao, G.F. Zheng, A new BP algorithm with adaptive momentum for FNNs training, in: 2009 WRI Global Congress on Intelligent Systems (GCIS'09), vol. 4, 2009, pp. 16–20.

[9] Levenberg, Kenneth (1944). "A Method for the Solution of Certain Non-Linear Problems in Least Squares". Quarterly of Applied Mathematics. 2:164–168.

[10] Marquardt, Donald (1963). "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". SIAM Journal on Applied Mathematics. 11 (2): 431–441. doi:10.1137/0111030.

[11] Duchi et al. (2012)."Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". Journal of Machine Learning Research. 12, 2121-2159.

[12] Xinxing Pan; Lee, B.; Chunrong Zhang, "A comparison of neural network backpropagation algorithms for electricity load forecasting," Intelligent Energy Systems, 2013 IEEE International Workshop on, pp.22,27, 14-14

[13] Zakaria Nouir, Berna Sayrac and Benoit Fourestie, Walid Tabbara and Francoise Brouaye, Comparison of Neural Network Learning Algorithms for Prediction Enhancement of a Planning Tool, 13th European Wireless Conference, 2007

[14] Kisi O, Uncuoglu E. Comparison of the three backpropagation training algorithms for two case studies. Indian J Eng Mater Sci 2005;12(5):434–42. [37] Hornik K. Multilayer feedforward networks are universal approximators. Neural Networks 1989;2:359–66.

[15] C. B. Khadse, M. A. Chaudhari and V. B. Borghate, "Comparison between three back-propagation algorithms for power quality monitoring," *2015 Annual IEEE India Conference (INDICON)*, New Delhi, 2015, pp.1-5. doi: 10.1109/INDICON.2015.7443766

[16] I. Arora and A. Saha, "Comparison of back propagation training algorithms for software defect prediction," 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Noida, 2016, pp. 51-58. doi: 10.1109/IC3I.2016.7917934

[17] V. K. Garg and R. K. Bansal, "Comparison of neural network back propagation algorithms for early detection of sleep disorders," 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, 2015, pp. 71-75. doi: 10.1109/ICACEA.2015.7164648

[18] Bazzi T., Estes M., Scherer B., 2016/8. "Artificial Intelligence For Precipitator Diagnostics". Power Plant Pollutant Control MEGA Symposium (MEGA 2016), Air and Waste Management Association (A&WMA), Vol 1, ISBN 9781510829862.

[19] K. Hornik. Approximation capabilities of multilayer feedforward net-works. Neural Networks, 4(2):251-257, 1991.

[20] Park C., Ki M., Namkung J., Paik J. (2006) Multimodal Priority Verification of Face and Speech Using Momentum Back-Propagation Neural Network. In: Wang J., Yi Z., Zurada J.M., Lu BL., Yin H. (eds) Advances in Neural Networks - ISNN 2006. ISNN 2006. Lecture Notes in Computer Science, vol 3972. Springer, Berlin, Heidelberg.

[21] D.E. Rumelhart, J.L. McClelland. The PDP Research Group, Parallel Distributed Processing- Explorations in the Microstructure of Cognition, MIT Press, MA, 1986.

[22] M.R Meybodi, H. Beigy. A note on learning automated-based schemes for adaptation of BP parameters, Neurocomputing 48 (2002) 957-974.

[23] M. Gori, M. Maggini, Optimal convergence of on-line backpropagation,IEEE Transactions on Neural Networks (1996) 251–254.

[24] V.V. Phansalkar,P.S. Sastry, Analysis of the back-propagation algorithm with momentum, IEEE Transactions on Neural Networks 5(3) (1994) 505–506.

[25] M. Torii, M.T. Hagan,Stability of steepest descent with momentum for quadratic functions,IEEE Transactions on NeuralNetworks 13(3) (2002) 752–756.

[26] Z. Zeng , Analysis of global convergence and learning parameters of the back-propagation algorithm for quadratic functions, Lecture Notes in Computer Science, vol. 4682, 2007, pp.7–13.

[27] E. Istook,T. Martinez, Improved backpropagation learning in nerural networks with windowed momentum,International Journal of Neural System 12 (3–4) (2002) 303–318.

[28] Y. Bengio, N. Boulanger-Lewandowski and R. Pascanu, "Advances in optimizing recurrent networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, 2013, pp. 8624-8628. doi: 10.1109/ICASSP.2013.6639349

[29] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543.

[30] Clark, L. (2012, June 26). GOOGLE'S ARTIFICIAL BRAIN LEARNS TO FIND CAT VIDEOS. Retrieved from https://www.wired.com/2012/06/google-x-neural-network/