Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 346949, 9 pages
http://dx.doi.org/10.1155/2013/346949

Research Article

Fast Linear Adaptive Skipping Training Algorithm for Training Artificial Neural Network

R. Manjula Devi, S. Kuppuswami, and R. C. Suganthe

Department of Computer Science and Engineering, Kongu Engineering College, Erode 638 052, India

Correspondence should be addressed to R. Manjula Devi; rmanjuladevi.gem@gmail.com

Received 12 April 2013; Accepted 27 May 2013

Academic Editor: Ker-Wei Yu

Artificial neural network has been extensively consumed training model for solving pattern recognition tasks. However, training a very huge training data set using complex neural network necessitates excessively high training time. In this correspondence, a new fast Linear Adaptive Skipping Training (LAST) algorithm for training artificial neural network (ANN) is instituted. The core essence of this paper is to ameliorate the training speed of ANN by exhibiting only the input samples that do not categorize perfectly in the previous epoch which dynamically reducing the number of input samples exhibited to the network at every single epoch without affecting the network's accuracy. Thus decreasing the size of the training set can reduce the training time, thereby ameliorating the training speed. This LAST algorithm also determines how many epochs the particular input sample has to skip depending upon the successful classification of that input sample. This LAST algorithm can be incorporated into any supervised training algorithms. Experimental result shows that the training speed attained by LAST algorithm is preferably higher than that of other conventional training algorithms.

1. Introduction

An artificial neural network (ANN) is a nonlinear knowledge processing model that have been successfully utilized training models for solving supervised pattern recognition tasks [1, 2] due to its ability to generalize the real-world problems. The two phases of ANN operation are training (or learning) phase and testing phase. Among these two phases, the training phase is extremely time-consuming phase of an ANN. Training speed is the one of the most significant things to be examined in training a neural network as the training phase generally consumes excessively high training time. The training speed of the neural network is determined by the rate at which the ANN trains. The factors that influence the training rate of ANN are size of the neural network, size of training dataset, initial weight value, problem category, and training algorithm. Among these factors, the size of training dataset is examined and the way in which it affects the training rate is discussed. In order to generalize the neural network well, both underfitting and overfitting of the training dataset should be avoided. Ample amount of training dataset is required to elude overfitting. However, large amount of training data normally requires very long training time [3] which affects the training rate.

Speeding the ANN training is still a focus of research attention in neural network to improve network for faster processing. Many research works have been explored on different amendments by estimating optimal initial weight [4, 5], adaptive learning rate and momentum [6], and using second order algorithm [7–9] in favor of ameliorating the training speed and maintaining generalization. First, proper initialization of neural network initial weights reduces the iteration number in the training process thereby increasing the training speed. Many weight initialization methods have been proposed for initialization of neural network weights. Nguyen and Widrow allocate the nodes' initial weight within the specified range which results in the reduction of the epoch number [4]. Varnava and Meade Jr formulated a new initialization method by approximating the networks parameter using polynomial basis function [5]. Second, the learning rate is used to control the step size for reconciling the network weights. The constant learning rate secures the convergence but considerably slows down the training process. Hence, several methods based on heuristic

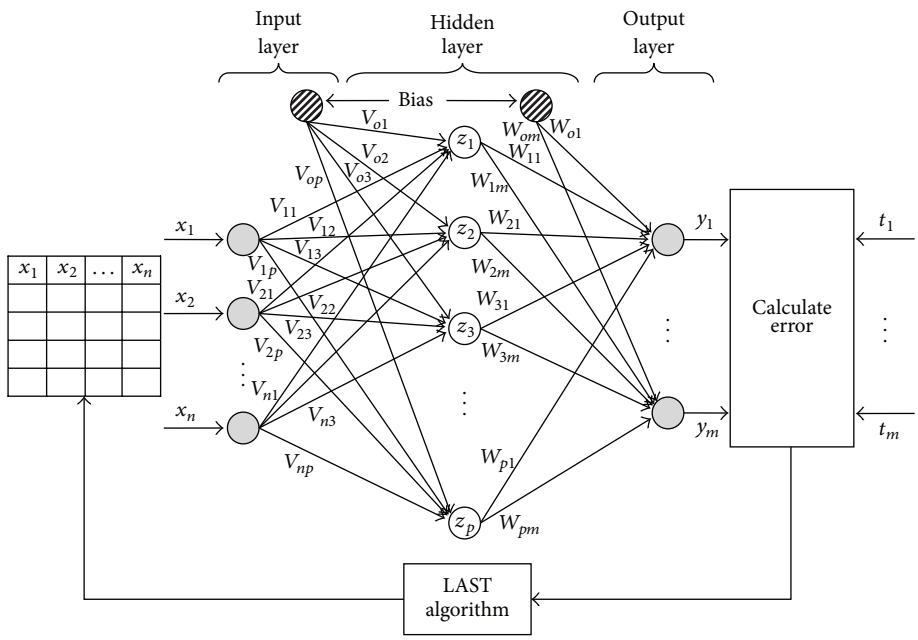

Figure 1: LAST incorporated in neural network architecture.

factor have been proposed for changing the training rate dynamically. Behera et al. applied convergence theorem based on Lyapunov stability theory for attaining the adaptive learning rate [6]. Last, second order training algorithms employ the second order partial derivatives information of the error function to perform network pruning. This algorithm is very apt for training the neural network that converges quickly. The most popular second order methods such as conjugate gradient (CG) methods, quasi-Newton (secant) methods, or Levenberg-Marquardt (LM) method are considered popular choices for training neural network. Nevertheless, it is not certain that these methods are very computationally expensive and require large memory particularly for large networks. Ampazis and Perantonis presented Levenberg-Marquardt with adaptive momentum (LMAM) and optimized Levenberg-Marquardt with adaptive momentum (OLMAM) second order algorithm that integrates the advantages of the LM and CG methods [7]. Wilamowski and Yu incorporated some modification in LM methods by rejecting Jacobian matrix storage and also replacing Jacobian matrix multiplication with the vector multiplication [8, 9] which results in the reduction of memory cost for training very huge training dataset.

All of the previously mentioned efforts are focused on speeding the training process by reducing the total number of epochs or by converging quickly. But each and every technique employs all the input samples in the training dataset to the network for classification at each and every single epoch. If a large amount of training data with high dimension is rendered for classification, then the mentioned technique introduces a problem by slowing down classification. There is a real fact that the correctly classified input samples are not involved in the weight updation since the error rate is calculated based on the misclassification rate. So, the intention of this research is to impart a simple and new algorithm for training the ANN in a fast manner. The core idea of LAST is when an input pattern is categorized perfectly by the network, and that particular pattern will not be presented again for the subsequent n epochs (epoch is one thorough cycle of populating the network with the entire training samples once). Only the patterns that do not categorize perfectly will be presented again for the next epoch which has reduced the total number of trained input samples employed by the network. Thereby, reducing the total amount of training dataset has reduced the training time which improved the training rate.

The gist of this research paper is systematized as follows. Section 2 describes how the new LAST algorithm is incorporated in standard BPN algorithm. Section 3 exhibits the experimental results. Finally, in Section 4, the overall conclusions of this research are drawn.

2. The Modified BPN with LAST Algorithm

2.1. LAST Neural Network Architecture. The LAST algorithm that is incorporated in the prototypical multilayer feedforward neural network architecture is sketched in Figure 1.

Such network is furnished with n input nodes, p hidden nodes, and m output nodes which are normally aligned in layers. Let P symbolize the number of training patterns. The input presented to the network is given in the form of matrix X with p rows and n columns. The number of network's input nodes is equivalent to the P, column value of the input matrix, X. Each row in the Matrix X, is considered to be a real-valued vector $x \in \mathfrak{N}^{n+1}$ which is symbolized by $\{x_0, x_1, x_2, \ldots, x_n\}$ where x_0 is a bias signal. The summed real-valued vector

$z \in \mathfrak{N}^{p+1}$ generated from the hidden layer is symbolized by $\{z_0, z_1, z_2, \ldots, z_p\}$ with z_0 being the bias signal. The estimated output real-valued vector $y \in \mathfrak{N}^m$ by the output layer is symbolized by $\{y_1, y_2, \ldots, y_m\}$ and the corresponding target vector $t \in \mathfrak{N}^m$ is symbolized by $\{t_1, t_2, \ldots, t_m\}$. Let (it) signify the (it)th iteration number.

The network parameter symbols employed in this algorithm are addressed here. Let $f_N(x)$ and $f_L(x)$ be the nonlinear logistic activation function and linear activation function of the hidden and output layer, respectively.

Since the network is fully interconnected, each layer nodes is integrated with all the node in the next layer. Let v_{ij} be the $n \times p$ matrix carries input-to-hidden weight coefficient for the link from the input node i to the hidden node j and v_{oj} the bias weight to the hidden node j. Let w_{jk} be the $p \times m$ matrix hidden-to-output weight coefficient for the link from the hidden node j to the output node k and w_{ok} the bias weight to the output node k.

2.2. LAST Essence. In BPN algorithm, each output unit correlates its computed activation y_k with its target value t_k to assess the associated error for that pattern with that unit. The factor δ is computed for the patterns that do not exhibit the correct value of y_k. The weights and biases are renovated according to the δ factor.

The core idea of LAST is when an input pattern is categorized perfectly by the network, and that particular pattern will not be presented again for the subsequent n epochs (epoch is one thorough cycle of populating the network with the entire training samples once). Only the patterns that do not categorize perfectly will be presented again for the next epoch which will decrease the training time largely.

2.3. Working Principle of LAST. The working principle of the LAST algorithm that is incorporated in the BPN algorithm is documented as follows.

Weight Initialization

Step 1. Determine the magnitude of the connection initial weights (and biases) to the disseminated values within the precised range and also the learning rate, α.

Step 2. While the iteration terminating criterion is attained, accomplish Steps 3 through 17.

Step 3. Iterate through the Steps 4 to 15 for each input training vector to be classified whose probability value, prob, is 1.

Furnish the Training Pattern

Step 4. Trigger the network by rendering the training input matrix to the input nodes in the network input layer.

Feedforward Propagation

Step 5. Disseminate the input training vector from the input layer towards the subsequent layers.

Step 6. Hidden Layer Activation net value is as follows.

(a) Each hidden node (z_j, $j = 1, 2, \ldots, p$) input is aggregated by multiplying input values with the corresponding synaptic weights

$$z_{inj}(\text{it}) = v_{oj}(\text{it}) + \sum_{i=1}^{n} x_i(\text{it}) \cdot v_{ij}(\text{it}). \tag{1}$$

(b) Apply nonlinear logistic sigmoid activation function to estimate the actual output for each hidden node j, $1 \le j \le p$. Consider

$$z_j(\text{it}) = \frac{1}{1 + e^{-z_{inj}}}. \tag{2}$$

Attaining the differential for the aforementioned activation function,

$$\frac{\partial\left(z_j(\text{it})\right)}{\partial x} = z_j(\text{it}) \times \left(1 - z_j(\text{it})\right). \tag{3}$$

Step 7. Output Layer Activation net value is next.

(a) Each output node (y_k, $k = 1, 2, \ldots, m$) input is aggregated by multiplying input values with the corresponding synaptic weights

$$y_{ink}(\text{it}) = w_{ok}(\text{it}) + \sum_{j=1}^{p} z_j(\text{it}) \cdot w_{jk}(\text{it}). \tag{4}$$

(b) Apply non-linear logistic sigmoid activation function to estimate the actual output for each output node k, $1 \le k \le m$. Consider

$$y_k(\text{it}) = \frac{1}{1 + e^{-y_{ink}}}. \tag{5}$$

Attaining the differential for the aforementioned activation function,

$$\frac{\partial\left(y_k(\text{it})\right)}{\partial x} = y_k(\text{it}) \times \left(1 - y_k(\text{it})\right). \tag{6}$$

Accumulate the Gradient Components (Back Propagation)

Step 8. For each output unit k, $1 \le k \le m$, the error gradient calculation for the output layer is formulated as

$$\delta_k(\text{it}) = y_k(\text{it}) \cdot \left[1 - y_k(\text{it})\right] \cdot \left[t_k - y_k(\text{it})\right]. \tag{7}$$

Step 9. For each hidden unit j, $1 \le j \le p$, the calculation of error gradient for the hidden layer is formulated as

$$\delta_j(\text{it}) = \left[\sum_{k=1}^{m} \delta_j(\text{it}) \cdot w_{jk}(\text{it})\right] z_j(\text{it}) \cdot \left[1 - z_j(\text{it})\right]. \tag{8}$$

Weight Amendment Using Delta-Learning Rule

Step 10. For each output unit, Consider the following.

The weight amendment is yielded by the following updating rule:

$$W_{jk}(\text{it}+1) = W_{jk}(\text{it}) + \Delta W_{jk}(\text{it}), \tag{9}$$

where

$$\Delta W_{jk}(\text{it}) = \alpha(\text{it}) \cdot \delta_k(\text{it}) \cdot z_j(\text{it}). \tag{10}$$

The bias amendment is yielded by the following updating rule:

$$W_{ok}(\text{it}+1) = W_{ok}(\text{it}) + \Delta W_{ok}(\text{it}), \tag{11}$$

where

$$\Delta W_{ok}(\text{it}) = \alpha(\text{it}) \cdot \delta_k(\text{it}). \tag{12}$$

Step 11. For each hidden unit, Consider the following.

The weight amendment is yielded by the following updating rule

$$V_{ij}(\text{it}+1) = V_{ij}(\text{it}) + \Delta V_{ij}(\text{it}), \tag{13}$$

where

$$\Delta V_{ij}(\text{it}) = \alpha(\text{it})\,\delta_j(\text{it})\,x_i(\text{it}). \tag{14}$$

The bias amendment is yielded by the following updating rule:

$$V_{oj}(\text{it}+1) = V_{oj}(\text{it}) + \Delta V_{oj}(\text{it}), \tag{15}$$

where

$$\Delta V_{oj}(\text{it}) = \alpha(\text{it})\,\delta_j(\text{it}). \tag{16}$$

LAST Algorithm Steps

Step 12. Measure the dissimilarity between the target and true value of each input sample (x_i, $i = 1, 2, \ldots, n$) which imitates the utter error value

$$\left|t_k - y_k(\text{it})\right|, \quad k = 1, 2, \ldots, m. \tag{17}$$

Step 13. Accomplish collation between the utter error value, $|t_k - y_k|$, and error threshold, $d_{\max}$, $|t_k - y_k(\text{it})| < d_{\max}$. If so, go to Step 14. Otherwise, go to Step 16.

Step 14. Compute the possibility value for presenting the input sample in the next iteration:

$$\text{prob}\left(x^i\right) = \begin{cases} 0, & \text{if } x^i \text{ is classified correctly and} \\ & \text{number of epochs is less than } n, \\ 1, & \text{otherwise.} \end{cases} \tag{18}$$

Step 15. Calculate n, number of epochs to be skipped.

(i) Initialize the value of n to zero for a particular sample x^i.

(ii) If x^i is classified correctly, then increment the value of n by c, where $c \rightarrow$ Linear Skipping Factor.

Step 16. Construct the new probability-based training dataset to be presented in the next epoch.

Step 17. Inspect for the halting condition such as applicable mean square error (MSE), elapsed epochs, and desired accuracy.

Table 1: Concrete quantity of the data sets used in the research.

Datasets	No. of attributes	No. of classes	No. of instances
Iris	4	3	150
Waveform	21	3	5000
Heart	13	2	270
Breast cancer	31	2	569

3. Experimental Results

The proposed LAST algorithm has been analyzed for the categorization problem concomitant with two-class and multiclass. The real-world workbench data sets wielded for training and testing the algorithms are Iris data set, Waveform data set, Heart data set and Breast Cancer data set which are possessed from the UCI (University of California at Irvine) Machine Learning Repository [10]. The concrete quantity of the data sets used in the research is provided in Table 1.

The magnitude of the initial weights is materialized with uniform random values in the range −0.5 to +0.5 using the Nguyen-Widrow (NW) initialization method for faster learning. All the bias nodes are enforced with the unit value.

3.1. Multiclass Problems

3.1.1. Iris Data Set. The real database of Iris flowering plant consists of measurements of 150 flower samples. For each flower, the four facets weighed for each flower are positioned here: a flower Sepal Length and Width and a flower Petal Length and Width. In fact, these four facets are involved in the categorization of each flower plant into apposite Iris flower genus: Iris Setosa, Iris Versicolour, and Iris Virgincia. The 150 flower samples are equally scattered amidst the three iris flower classes. Iris setosa is linearly separable from the other 2 genera. But Iris Virgincia and Iris Versicolour are nonlinearly detachable. Out of these 150 flower samples, 90 flower samples are employed for training and 60 flower samples for testing.

The neural network is orderly structured with 4, 5, and 1 neurons in the input, hidden, and output layers, respectively, for training Iris flowering plant database with the step size of 0.3 and momentum constant of 0.8. Such skeleton is put into training for 675 epochs by exploiting BPN and LAST algorithms.

The visual representation of the number of trained input samples and training time seized by BPN and LAST algorithms at every single epoch is laid out in Figures 2 and 3, respectively.

The empirical outcomes of BPN and LAST algorithms are rendered in Table 2. From this table, the LAST algorithm yield improved computational training speed in terms of the total number of trained input samples as well as total training time. Also the LAST algorithm is yielded with the performance of identical recognition accuracy.

3.1.2. Waveform Data Set. The Waveform database generator data set is holding the measurements of 5000 wave's samples.

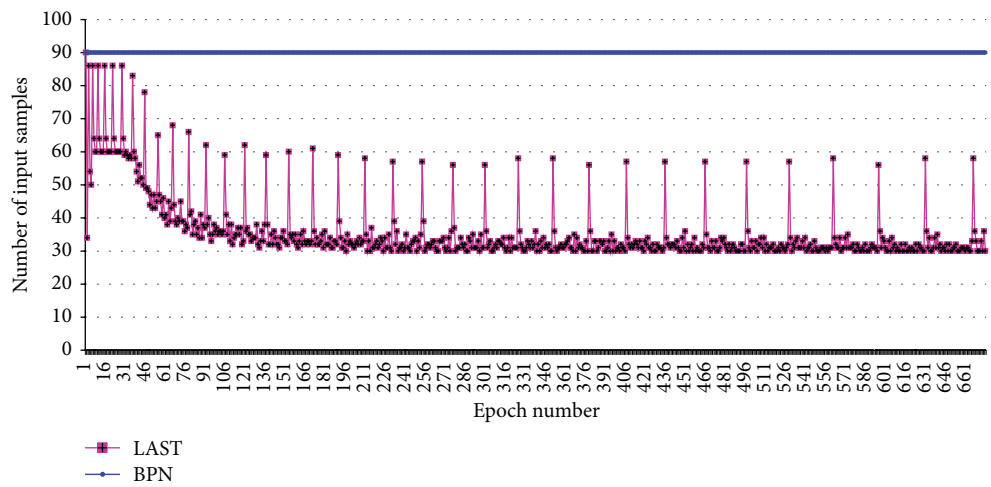

Figure 2: IRIS dataset: epoch-wise input samples taken by BPN and LAST algorithms.

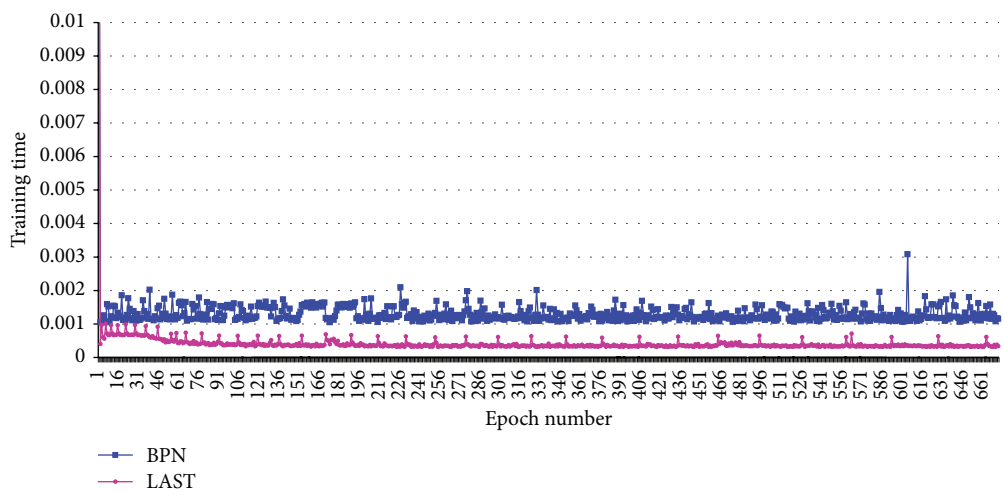

Figure 3: IRIS dataset: epoch-wise training time taken by BPN and LAST algorithms.

Table 2: Result comparison of BPN and LAST algorithms for the IRIS dataset.

Neural network algorithm	Network topology	Number of epochs	Total number of input samples	Training time (in sec.)	Accuracy (%)
BPN	$4 \times 5 \times 1$	675	60750	0.036189	91.67
LAST	$4 \times 5 \times 1$	675	24148	0.016641	91.67

The 5000 wave's samples are equally distributed, about 33%, into three wave families [10]. These waves are recognized with the 21 numerical features. Among these 5000 wave's samples, 4800 waves and 200 waves are randomly selected to form the training set and testing set.

A 3-layer feedforward neural network, containing 21, 10, and 1 units in the input, hidden, and output layers, respectively, is trained with the learning rate parameters $\alpha = 1e - 2$ and a momentum constant of 0.7. The previous structure is put into training for 675 epochs by exploiting BPN and LAST algorithms.

The visual representation of the number of trained input samples and training time seized by BPN and LAST algorithms at every single epoch is laid out in Figures 4 and 5, respectively.

The empirical results of BPN and LAST algorithms are rendered in Table 3. From this table, the LAST algorithm yield improved computational training speed in terms of the total number of trained input samples as well as total training time. Also the LAST algorithm is yielded with the performance of high recognition accuracy.

3.2. Two-Class Problems

3.2.1. Heart Data Set. The Statlog Heart disease database consists of 270 patient's samples. Each patient's characteristics are recorded using 13 attributes. These 13 features are involved

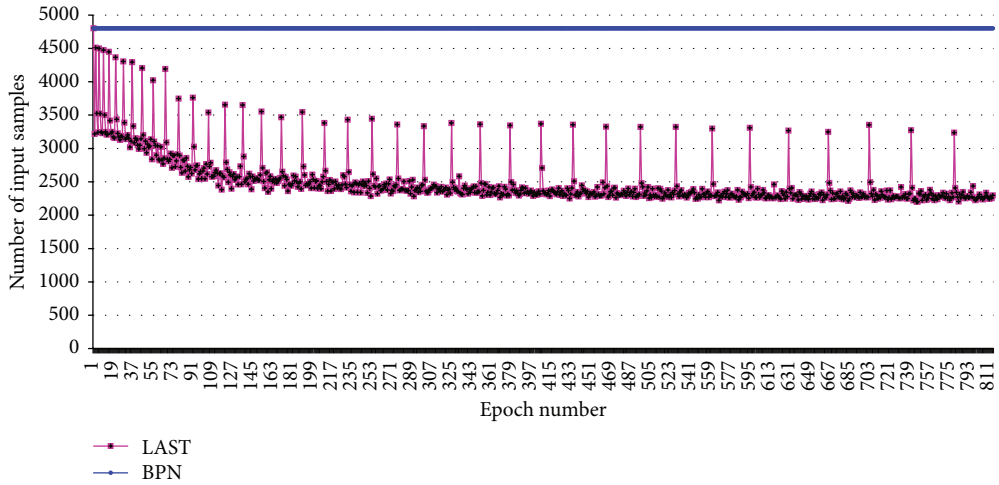

Figure 4: Waveform dataset: epoch-wise input samples taken by BPN and LAST algorithms.

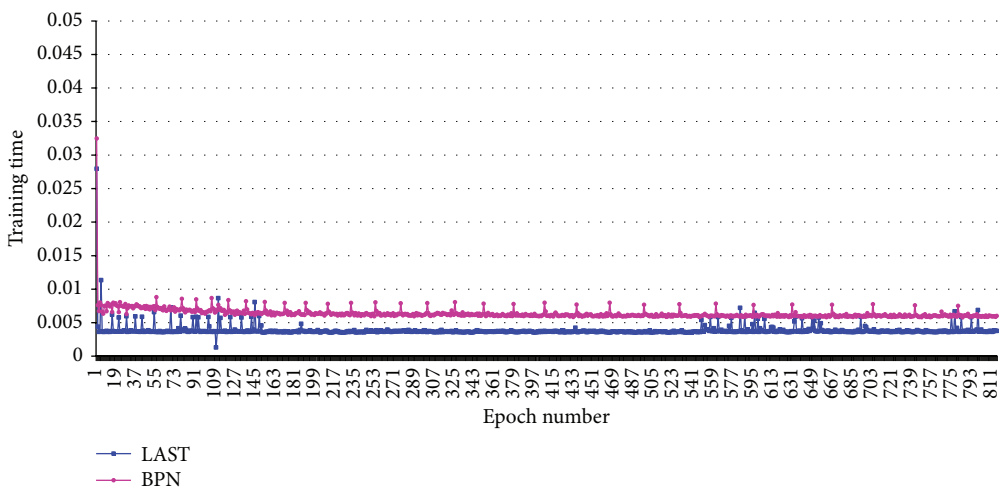

Figure 5: Waveform dataset: epoch-wise training time taken by BPN and LAST algorithms.

Table 3: Result comparison of BPN and LAST algorithms for the waveform dataset.

Neural network algorithm	Network topology	Number of epochs	Total number of input samples	Training time (in sec.)	Accuracy (%)
BPN	21×10×1	815	3912000	0.005806	97.00
LAST	21×10×1	815	2035031	0.000328	97.50

in the detection of the presence and absence of heart disease for the patients.

The neural network is orderly structured with 13, 5, and 1 neurons in the input, hidden, and output layers, respectively, for training Breast Cancer database with the step size of 0.3 and momentum constant of 0.9. Such skeleton is put into training for 619 epochs by exploiting BPN and LAST algorithms.

The visual representation of the number of trained input samples and training time seized by BPN and LAST algorithms at every single epoch is laid out in Figures 6 and 7, respectively.

The empirical results of BPN and LAST algorithms are rendered in Table 4. From this table, the LAST algorithm yield improved computational training speed in terms of the total number of trained input samples as well as total training time. Also the LAST algorithm is yielded with the performance of high recognition accuracy.

3.2.2. Breast Cancer Data Set. The Wisconsin Breast Cancer Diagnosis Dataset contains 569 patient's breasts samples among which 357 diagnosed as benign and 212 diagnosed as malignant class. Each patient's characteristics are recorded using 32 numerical features.

The neural network is orderly structured with 31, 15, and 1 neurons in the input, hidden, and output layers, respectively,

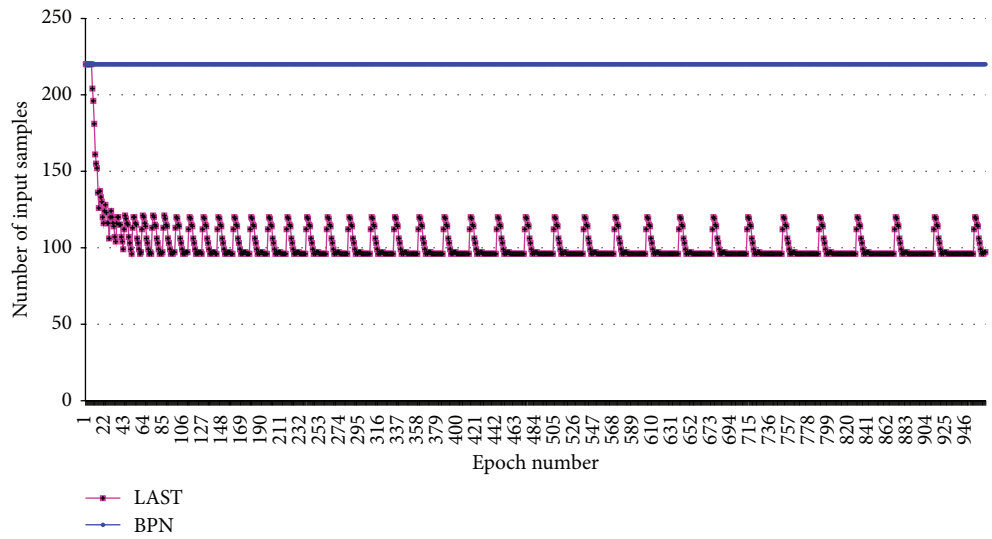

Figure 6: Heart dataset: epoch-wise input samples taken by LAST and BPN algorithms.

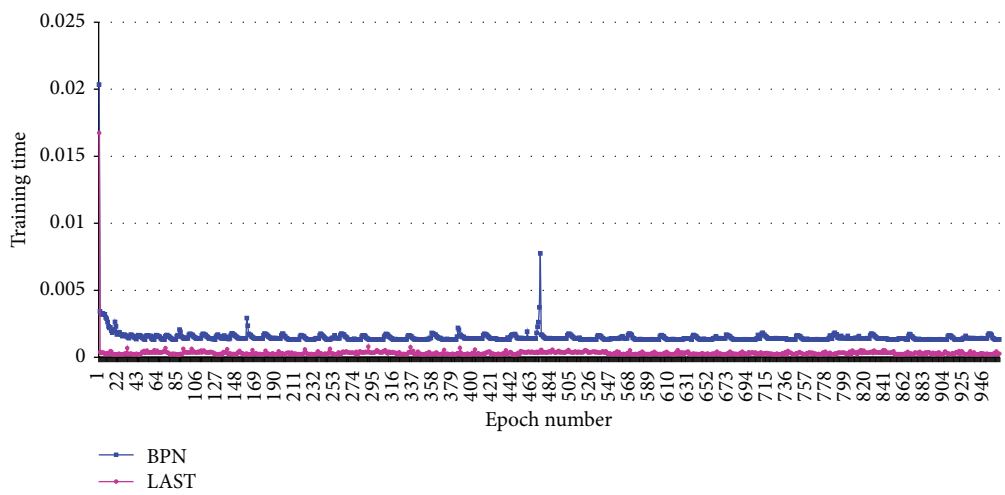

Figure 7: Heart dataset: epoch-wise training time taken by BPN and LAST algorithms.

Table 4: Result comparison of BPN and LAST algorithms for the Heart dataset.

Neural network algorithm	Network topology	Number of epochs	Total number of input samples	Training time (in sec.)	Accuracy (%)
BPN	$13 \times 5 \times 1$	964	212080	0.024903	90.00
LAST	$13 \times 5 \times 1$	964	98976	0.004097	92.00

for training Breast Cancer database with the step size of 0.3 and momentum constant of 0.9. Such skeleton is put into training for 619 epochs by exploiting BPN and LAST algorithms.

The visual representation of the number of trained input samples and training time seized by BPN and LAST algorithms at every single epoch is laid out in Figures 8 and 9, respectively.

The empirical results of BPN and LAST algorithms are rendered in Table 5. From this table, the LAST algorithm yield improved computational training speed in terms of the total number of trained input samples as well as total training time. Also the LAST algorithm is yielded with the performance of equal recognition accuracy.

3.2.3. Result Comparison. From Tables 2, 3, 4, and 5, it is concluded that the proposed LAST algorithm attains the higher training performance in terms of trained input samples and time. The comparison results of the training time for the

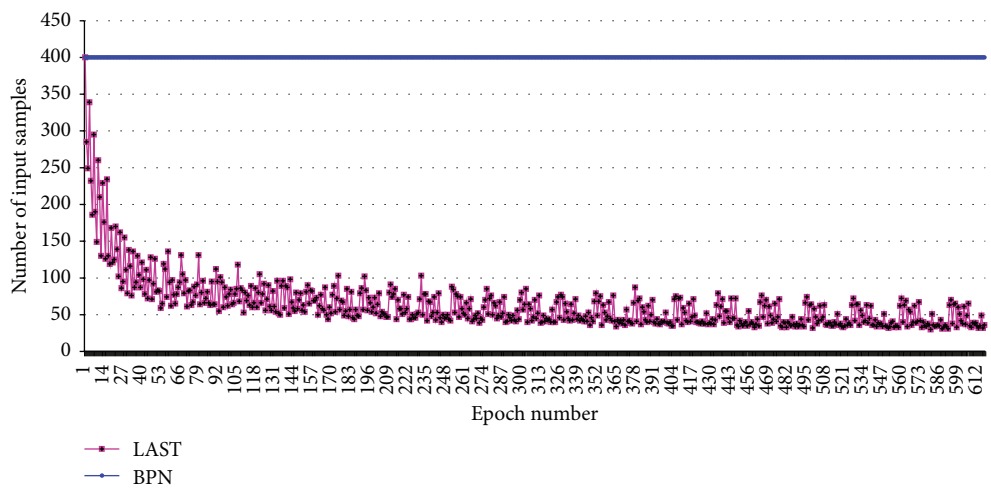

Figure 8: Breast Cancer dataset: epoch-wise input samples taken by LAST and BPN algorithms.

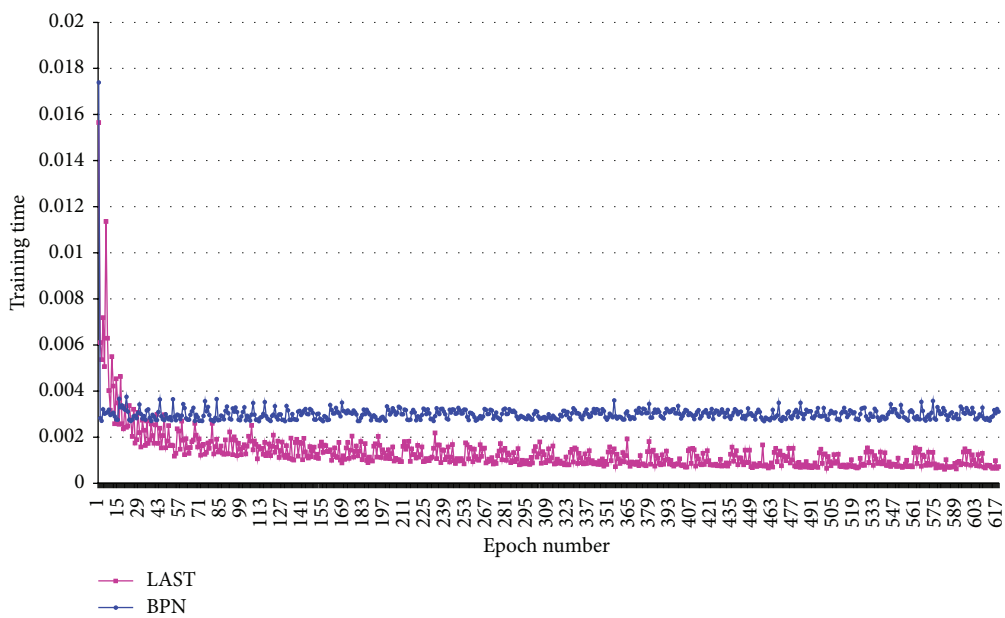

Figure 9: Breast Cancer dataset: epoch-wise training time taken by LAST and BPN algorithms.

Table 5: Result comparison of BPN and LAST algorithms for the Breast Cancer dataset.

Neural network algorithm	Network topology	Number of epochs	Total number of input samples	Training time (in sec.)	Accuracy (%)
BPN	31×15×1	619	247600	0.023736	95.27
LAST	31×15×1	619	39388	0.013930	95.27

previously mentioned dataset are consolidated in Figure 10. From this figure, the total training time of LAST algorithm is reduced to an average of nearly 50% of BPN algorithm.

4. Conclusion

A simple and new Linear Adaptive Skipping Training algorithm for training MFNN is systematically investigated in order to speed up the training phase of MFNN. The previously empirical results demonstratively proved that the LAST algorithm dynamically reduces the total number of training input samples presented to the MFNN at every single cycle. Thus decreasing the size of the training set can reduce the training time, thereby increasing the training speed. Finally, the proposed LAST algorithm seems to be faster than the standard BPN algorithm in training MFNN, and also the LAST Algorithm can be incorporated with any supervised task for training all real-world problems.

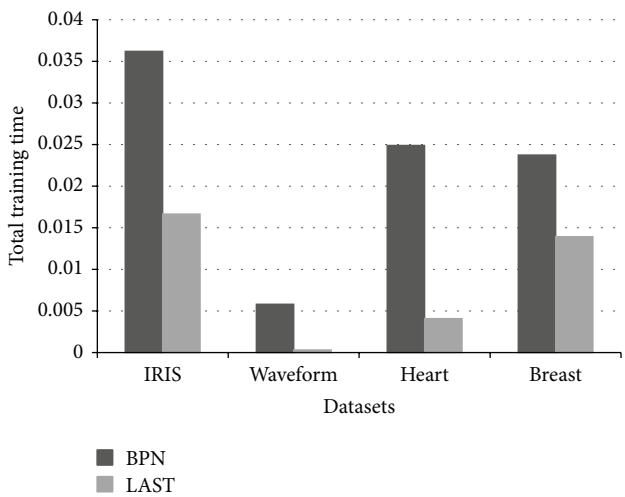

Figure 10: Total training time taken by BPN and LAST algorithm.

Both algorithms were implemented on a machine with the aforementioned configuration: Intel Core I5-3210M processor, CPU speed of 2.50 GHz, and 4 GB of RAM. The MATLAB version used for implementation is R2010b.

References

[1] P. Mehra and B. W. Wah, *Artificial Neural Networks: Concepts and Theory*, IEEE Computer Society Press, 1992.

[2] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[3] A. J. Owens, "Empirical modeling of very large data sets using neural networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '2000)*, vol. 6, pp. 302–307, July 2000.

[4] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '90)*, vol. 3, pp. 21–26, San Diego, Calif, USA, June 1990.

[5] T. M. Varnava and A. J. Meade Jr., "An initialization method for feedforward artificial neural networks using polynomial bases," *Advances in Adaptive Data Analysis*, vol. 3, no. 3, pp. 385–400, 2011.

[6] L. Behera, S. Kumar, and A. Patnaik, "On adaptive learning rate that guarantees convergence in feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 5, pp. 1116–1125, 2008.

[7] N. Ampazis and S. J. Perantonis, "Two highly efficient second-order algorithms for training feedforward networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 5, pp. 1064–1074, 2002.

[8] B. M. Wilamowski and H. Yu, "Improved computation for levenbergmarquardt training," *IEEE Transactions on Neural Networks*, vol. 21, no. 6, pp. 930–937, 2010.

[9] H. Yu and B. M. Wilamowski, "Neural network training with second order algorithms," *Advances in Intelligent and Soft Computing*, vol. 99, pp. 463–476, 2012.

[10] A. Asuncion and D. J. Newman, "UCI Machine Learning Repository," School of Information and Computer Science, University of California, Irvine, Calif, USA ,2007, http://www.ics.uci.edu/~mlearn/MLRepository.html.