Can Threshold Networks be Trained Directly?

Guang-Bin Huang, *Senior Member, IEEE*, Qin-Yu Zhu, K. Z. Mao, Chee-Kheong Siew, *Member, IEEE*, P. Saratchandran, *Senior Member, IEEE*, and N. Sundararajan, *Fellow, IEEE*

***Abstract*—Neural networks with threshold activation functions are highly desirable because of the ease of hardware implementation. However, the popular gradient-based learning algorithms cannot be directly used to train these networks as the threshold functions are nondifferentiable. Methods available in the literature mainly focus on approximating the threshold activation functions by using sigmoid functions. In this paper, we show theoretically that the recently developed extreme learning machine (ELM) algorithm can be used to train the neural networks with threshold functions directly instead of approximating them with sigmoid functions. Experimental results based on real-world benchmark regression problems demonstrate that the generalization performance obtained by ELM is better than other algorithms used in threshold networks. Also, the ELM method does not need control variables (manually tuned parameters) and is much faster.**

***Index Terms*—Extreme learning machine (ELM), gradient descent method, threshold neural networks.**

I. Introduction

MULTILAYER neural networks have attracted a lot of interest in past decades. Although the neural networks with analog activation functions such as sigmoid and sine in hidden layers have great computational capabilities, the networks with the threshold or hard-limiting activation function in hidden layers

$$h(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{1}$$

are still desirable due to the following reasons.

1) The threshold units are easy for hardware implementation [1], [2].
2) The relationship between the size of the networks using threshold units and the complexity of the training are better understood [3], [4].

However, the widely used back-propagation (BP) learning algorithm [5] and its variants cannot be used to train the threshold neural networks directly as the threshold functions are nondifferentiable. Hence, in literature, many research efforts [2], [6], [7] have been pursued to modify the gradient-based learning methods to be applicable indirectly for networks with threshold units. BP and its variants are usually slow in learning and may face local minima problem. The validation process (selection of control parameters such as the learning rate, number of hidden neurons and learning epoches) is complicated and challenging to users, especially to those having little domain knowledge in the neural networks area. The large computational cost involved in the learning process makes implementing an online learning system on chips rather difficult. Thus, these algorithms are normally trained offline first and then the parameters (weights and biases) of neural networks are transfered to the threshold networks in hardware implementation.

Recently, a novel learning method for single-hidden-layer feedforward neural networks (SLFNs) named extreme learning machine (ELM) algorithm has been proposed in Huang *et al.* [8]. In this algorithm, the input weights (of the connections linking the input neurons to hidden neurons) and the bias of the hidden neurons are randomly generated based on continuous distribution probabilities (the uniform distribution probability is used in our simulations) and kept fixed. The output weights are then analytically determined. ELM learns much faster than the traditional BP without loss of generalization performance. As indicated in Huang *et al.* [8], ELM algorithm will work for the neural networks with threshold units as well. However, a detailed performance study of ELM for threshold neural networks has not been carried out so far and this paper attempts to fill this gap. The aim of this paper is two-fold: 1) to prove, in theory, that similar to ELM for sigmoid networks in theory the input weights and biases of the threshold networks can also be assigned randomly based on continuous distribution probabilities (such as uniform distribution probability used in our simulations) and thus ELM can be used to train such networks easily without any modifications; 2) to provide a detailed performance evaluation of ELM for threshold units based on a number of real-world benchmark regression problems. Simulations results show that the ELM for threshold networks achieves better generalization performance than those trained by other BP methods.

II. Brief Review of Learning Algorithms for Threshold Networks

As the threshold functions are nondifferentiable, the gradient descent learning algorithms for multilayer feedforward neural networks cannot be directly applied. Hence, a number of modifications to gradient descent methods have been proposed in the literature. Toms [6] proposed a gradient descent learning algorithm for networks with hybrid activation functions which linearly combine the sigmoid and threshold functions as

$$f(x) = bS(x) + (1 - b)\theta(x) \tag{2}$$

where the $S(x)$ is sigmoid function and $\theta(x)$ is a threshold function. The training is initiated with $b = 1$. During the training, b is gradually decreased to 0. Thus, the activation functions of hidden neurons gradually are transformed from pure analog units to pure threshold ones. The activation function $f(x)$ is not a direct threshold function and is differentiable everywhere except at $x = 0$.

Manuscript received September 23, 2004; revised April 25, 2005. This paper was recommended by Associate Editor E. N. Sanchez.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798 (e-mail: egbhuang@ntu.edu.sg).

Digital Object Identifier 10.1109/TCSII.2005.857540

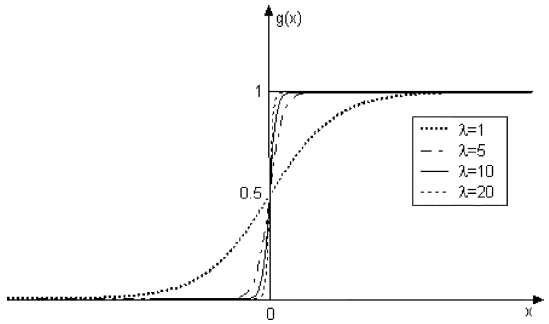

Fig. 1. Threshold unit can only be approximated by sigmoid unit with sufficiently large gain parameter λ.

Corwin *et al.* [2] has proposed an iterative method for training multilayer networks with threshold functions. The sigmoid function with a gain parameter λ is used in the training instead of the threshold function directly

$$g(x) = \frac{1}{1+e^{-\lambda x}}. \tag{3}$$

If the training error is small, the gain parameter λ is gradually increased during the training until the slope of the sigmoid is sufficiently large to allow a transfer to a threshold network with the same architecture. However, in many cases, the error may not be small enough to allow the λ to be increased. On the other hand, as shown in Fig. 1 λ may need to be big enough to let the sigmoid unit approximate the threshold unit properly.

In a method very similar to BP, Goodman and Zeng [7] compute the "pseudo-gradient" (instead of the true gradient), using the gradient of a sigmoid function as a heuristic hint in place of that of the hard-limiting function. As commented by Goodman and Zeng [7], the "inaccuracy" of the pseudo-gradient exists in hidden layers.

Bartlett and Downs [1] proposed a probability distribution based gradient descent approach. This approach assumes that the units' weights w_i are random variables with probability density functions $f_{w_i}(w_i)$. When these weights are normally distributed with mean μ_{w_i} and standard deviation σ_i, then the training of the networks can be easily achieved by adjusting the parameters of the cumulative distribution function (CDF) of each weight w_i. Finally, the mean w_i is considered as the weight w_i and its standard deviation σ_i as the weight of an additional connection between a Gaussian noise source and the unit. Because of the large computational cost involved in the training, Bartlett and Downs [1] has indicated that this algorithm is not suitable for online training.

Plagianakos *et al.* [9] have presented an algorithm for training threshold networks by using the differential evolution (DE) strategy. Although this algorithm need not compute the gradient of the error function, it does need an iterative learning scheme. Also, several control variables (mutation amplification, crossover constant, number of generalizations) need to be manually tuned making it slow especially for large size applications.

It should be noted that all the above algorithms are not suitable for online learning in environments where target functions may change often. They are only suitable for offline training (outside the chips), and parameters obtained after the completion of the training are then transferred to the threshold networks in chips.

III. ELM for Threshold Networks

Huang *et al.* [8] have alluded that ELM works for SLFNs with different activation functions including threshold functions. However, this has not been shown theoretically. In this section, we prove that in theory ELM can be used to directly train single hidden layer threshold networks.

A. Approximation Problem of SLFNs

For N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where the input $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in \mathbf{R}^n$ and the target $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in \mathbf{R}^m$, standard single-hidden-layer neural networks (SLFN) with L hidden neurons and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j, \qquad j = 1, \ldots, N \tag{4}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ is the weight vector connecting the ith hidden neuron and the input neurons, $\boldsymbol{\beta}_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector connecting the ith hidden neuron and the output neurons, and b_i is the bias of the ith hidden neuron. $\mathbf{w}_i \cdot \mathbf{x}_j$ denotes the inner product of $\mathbf{w}_i$ and $\mathbf{x}_j$.

The fact that standard SLFNs with L hidden neurons can approximate these N samples with zero error means that $\sum_{j=1}^{N} \|\mathbf{o}_j - \mathbf{t}_j\| = 0$, i.e., there exist $\boldsymbol{\beta}_i$, $\mathbf{w}_i$ and b_i such that

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j, \qquad j = 1, \ldots, N. \tag{5}$$

The above N equations can be written compactly as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{6}$$

where $\mathbf{H}$ is called the hidden layer output matrix of the SLFN [10], [11]; the ith column of $\mathbf{H}$, $[g(\mathbf{w}_i \cdot \mathbf{x}_1 + b_i), \ldots, g(\mathbf{w}_i \cdot \mathbf{x}_N + b_i)]^T$, is the ith hidden neuron output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. $\boldsymbol{\beta} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_L]^T$ and $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T$.

B. ELM Learning Algorithm for Threshold Networks

From the main results of [11], [12], the following lemma can be formulated.

Lemma 3.1: A SLFN with N hidden neurons with the activation function $g(x) = 1/(1+e^{-\lambda x})$ and with randomly chosen input weights and hidden biases can learn N distinct observations with any arbitrarily small error.

In fact, arbitrarily small error is not required in real applications. Instead, the training error E obtained using the network is only expected to be smaller than a given nonzero target error $\epsilon > 0$. Based on this, We have the following.

Theorem 3.1: For a SLFN with the activation function $g(x) = 1/(1+e^{-\lambda x})$ in the hidden layer, given any constant $\epsilon > 0$, there always exists an integer $L \leq N$ such that a SLFN with L hidden neurons and with randomly chosen input weights and hidden biases can learn N distinct observations with a training error less than ϵ.

Proof: According to Lemma 3.1, there always exists an integer $L \leq N$ such that $\sum_{j=1}^{N} \|\mathbf{o}_j - \mathbf{t}_j\|^2 < \epsilon$, otherwise as an extreme case we can simply choose $L = N$. ■

We now extend this theorem to threshold networks as given below.

Theorem 3.2: Suppose that threshold activation function $h(x) = 1_{x\geq 0} + 0_{x<0}$ is used in the hidden layer. Given any nonzero constant $\epsilon > 0$ there always exists an integer $L \leq N$ such that a SLFN with L such hidden neurons and with randomly chosen input weights and hidden biases can learn N distinct observations with its training error less than ϵ.

Proof: Since Theorem 3.1 holds for all positive gain parameter λ and $\lim_{\lambda\to+\infty} g(x) = h(x)$, the theorem holds. ■

Thus, very interestingly, similar to sigmoid SLFNs [11], [13] the input weights and hidden biases of threshold SLFNs are in fact not necessarily tuned and the hidden layer output matrix $\mathbf{H}$ can actually remain unchanged once random values have been assigned to input parameters and hidden neuron biases in the beginning of learning. According to Theorem 3.2, there exists L and randomly generated input weights $\mathbf{w}_i$ and hidden neuron biases b_i $(i = 1, \ldots, L)$ such that

$$E = \sum_{j=1}^{N} \left(\sum_{i=1}^{L} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) - \mathbf{t}_j \right)^2 < \epsilon. \qquad (7)$$

In most cases the number of required hidden neurons is much less than the number of distinct training samples, $L \ll N$, making $\mathbf{H}$ a nonsquare matrix. For fixed input weights $\mathbf{w}_i$ and the hidden layer biases b_i, from (7), *to train an SLFN* is simply equivalent to finding a least-squares solution $\hat{\boldsymbol{\beta}}$ of the linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$. The unique smallest norm least-squares solution of the above linear system is

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T} \qquad (8)$$

where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse of matrix $\mathbf{H}$. The learning procedure of ELM algorithm for threshold neural networks can therefore be described as follows.

ELM Algorithm:

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m\}$, threshold activation function $h(x) = 1_{x\geq 0} + 0_{x<0}$ and number of hidden neurons L

Step 1: Assign random input weight $\mathbf{w}_i$ and bias of hidden neurons b_i, $i = 1, \ldots, L$.

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$.

Step 3: Calculate the output weight $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}. \qquad (9)$$

C. Analysis of Generalization Performance

Unlike BP learning algorithms, ELM obtains both the smallest output weights and the smallest training error. As analyzed by Huang *et al.* [8], from the viewpoint of *Vapnik–Chervonenkis* (VC) dimension (and hence number of parameters) [4], this method tends to achieve good generalization performance. Gradient-based learning algorithms like back-propagation have the difficulty of reaching the good generalization performance since they only try to obtain the small training errors without considering the magnitude of the weights. (For detailed discussions, refer to [4], [8], [11])

IV. Simulation Results

In many applications, threshold networks have been approximated by sigmoid SLFNs with different gain parameters λ and then trained by BP and its variants. In this section, the performance comparison will be conducted between such BP based threshold network training algorithms and ELM algorithm for SLFN networks on 14 benchmark real life regression problems[1] which cover various fields of applications. The specifications of the datasets are listed in Table I. In our experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into [0, 1]. Both the input weights and hidden biases are randomly generated from $[-1,1]$. The average performance is obtained based on 50 trials of simulations done for each learning algorithm for each problem, and all datasets are randomly reshuffled into training set and testing set at each trial of simulations.

TABLE I
Specification of 14 Benchmark Real Life Datasets

Name	No. of Observations		Attributes
	Training	Testing	
Abalone	2000	2177	8
Delta Ailerons	3000	4129	6
Delta Elevators	4000	5517	6
Pole Telecomm	5000	10000	48
Computer Activity	4000	4192	8
Census (House16H)	10000	12784	16
Auto Price	80	79	15
Boston Housing	250	256	13
Pyrimidines	40	34	27
Triazines	100	86	60
Machine CPU	100	109	6
Servo	80	87	4
Breast Cancer	100	94	32
Stocks Domain	450	500	10

Since the BP algorithm cannot be applied to the threshold activation function directly, similar to Corwin *et al.* [2] sigmoid SLFNs with different gain parameters λ are used to approximate threshold networks in our experiments. BP and ELM are running in the MATLAB 6.5 (Windows version) environments. Levenberg–Marquardt (LM) learning algorithm, one of the fastest BP variant, is used in our simulations (cf. HELP of MATLAB). All the simulations have been run on the same PC with Pentium 4 3.0 GHZ and 768 MB RAM. It has been known that SLFN with too few hidden neurons may be incapable of learning whereas SLFN with too many hidden neurons may be overfitting. In both cases, the SLFNs do not produce good generalization. Thus, the size of the SLFN (number of hidden neurons) for both BP and ELM can be well chosen through an extensive validation process for each algorithm on each dataset, which provides almost the best validation performance for an algorithm in a dataset. For example, in our cases, the neuron numbers L of BP and ELM are searched from the

[1]The datasets can be downloaded from http://www.niaad.liacc.up.pt/~ltorgo/Regression/ds_menu.html

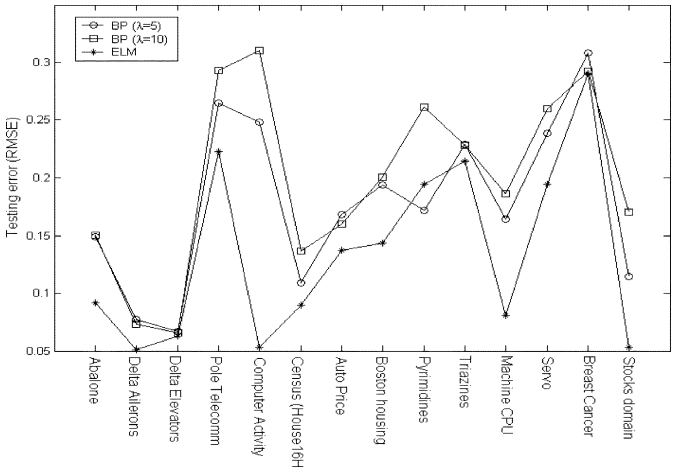

Fig. 2. Average generalization performance comparison of threshold networks directly trained by ELM algorithm and threshold networks approximated by analog networks trained by BP algorithm.

range $[5, 10, 15, \ldots, 390, 395, 400]$ and the best generalization performance obtained by BP and ELM are shown in this paper.

As shown in Fig. 1, sigmoid units can approximate threshold units only when the gain parameters λ are set large enough; otherwise one actually deals with a sigmoid network instead of threshold network. As shown in Fig. 2 and Table II, ELM for threshold networks usually obtains much better generalization performance than BP learning algorithms. Here the threshold networks are approximated by sigmoid networks with gain parameters $\lambda = 5$ and $\lambda = 10$ as set in Corwin *et al.* [2]. We have tried to gradually increase the gain parameter λ and check the relationship between the generalization performance and λ. As observed from Fig. 3, the generalization performance tends to decrease when the λ is increased.[2] Thus, the gradient descent method which gradually transforms the activation functions from sigmoid to threshold may not provide satisfactory performance when threshold units are approximated by sigmoid functions with higher gain parameters λ.

As shown in Table III the ELM algorithm is very stable and able to obtain much smaller standard deviation of testing errors. As shown in Table IV, the ELM algorithm runs much faster than BP. Since the input weights and hidden biases of ELM are randomly generated instead of being fine-tuned, ELM usually needs more neurons than other learning algorithms. For example, if the target function is a sigmoid function: $f(x) = 1/(1+e^{-x})$, BP with sigmoid activation function $g(x) = 1/(1+e^{-x})$ and one hidden neuron can approximate this target function after fine-tuning the input weights and hidden bias. Obviously, in general, ELM with one hidden neuron (with random input weights and hidden bias) cannot approximate this target function and ELM may need more neurons (around 10 in this case) so that the combination of all these neurons can approximate it.

V. Discussion and Conclusion

It is well known that compared with analog neural networks threshold networks can reduce the complexity of neural network implementation in hardware. The traditional backpropagation algorithm cannot be applied to such threshold networks since the required derivatives are not available. Many learning algorithms do not deal with threshold networks directly and instead use some analog networks to approximate them such that gradient-descent method can finally be used. Similar to conventional BP, these methods may face local minima and over fitting issues. To our knowledge, all these learning algorithms [1], [2], [6], [7], [9] may face difficulty for online and/or hardware implementation due to the following.

1) Besides network size many control variables (manually tuned parameters) are required to be manually selected in advance.
2) Learning parameters (connection weights and hidden neuron biases) can be transformed to threshold networks in hardware only after training is completed.

[2]For the sake of readability, only a partial of all simulation results are shown in Fig. 3.

TABLE II
Comparison of Average Generalization Performance (Average Root Mean Square Error (RMSE) of Testing) of Different Learning Algorithms

Dataset	BP ($\lambda = 5$)	BP ($\lambda = 10$)	ELM (threshold)
Abalone	0.14919	0.15017	0.091948
Delta Ailerons	0.077026	0.073125	0.050801
Delta Elevators	0.06706	0.065261	0.0630
Pole Telecomm	0.26473	0.29321	0.2225
Computer Activity	0.24818	0.31035	0.0530
Census (House16H)	0.109	0.13631	0.089744
Auto Price	0.16813	0.15976	0.13709
Boston housing	0.1939	0.20077	0.14325
Pyrimidines	0.17165	0.26147	0.19416
Triazines	0.22907	0.22836	0.21426
Machine CPU	0.16445	0.18598	0.08066
Servo	0.23886	0.26012	0.19474
Breast Cancer	0.3085	0.29246	0.29022
Stocks Domain	0.11432	0.17013	0.052997

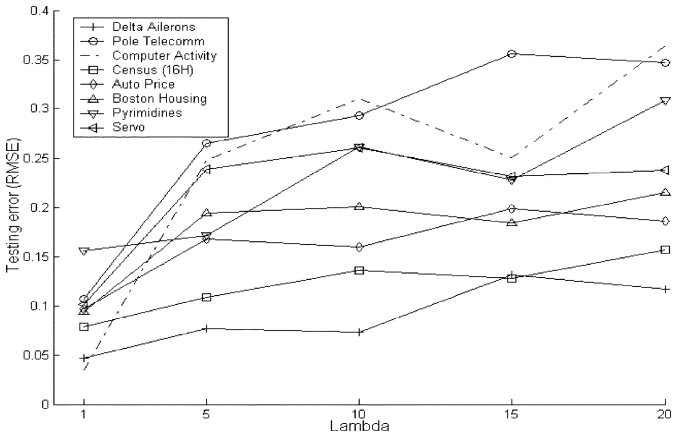

Fig. 3. Generalization performance of analog approximated threshold networks depends closely on the gain parameter λ.

TABLE III
COMPARISON OF STANDARD DEVIATIONS OF TESTING RMSE OF DIFFERENT LEARNING ALGORITHMS

Dataset	BP ($\lambda = 5$)	BP ($\lambda = 10$)	ELM (threshold)
Abalone	0.078597	0.093745	0.007065
Delta Ailerons	0.056256	0.033776	0.0055414
Delta Elevators	0.026206	0.019181	0.005523
Pole Telecomm	0.092299	0.11306	0.009235
Computer Activity	0.31722	0.32758	0.006495
Census (House16H)	0.034903	0.082774	0.0016687
Auto Price	0.078444	0.070159	0.024718
Boston Housing	0.1018	0.11585	0.021672
Pyrimidines	0.046733	0.16867	0.061273
Triazines	0.071486	0.052997	0.027469
Machine CPU	0.13201	0.12717	0.026763
Servo	0.11372	0.11216	0.021299
Breast Cancer	0.074661	0.035625	0.019929
Stocks Domain	0.080048	0.15098	0.0038313

TABLE IV
COMPARISON OF TRAINING TIME AND NUMBER OF HIDDEN NEURONS OF DIFFERENT LEARNING ALGORITHMS

Dataset	BP ($\lambda = 5$)		BP ($\lambda = 10$)		ELM-threshold	
	time (s)	neurons	time (s)	neurons	time (s)	neurons
Abalone	6.6219	5	7.3251	5	0.24004	80
Delta Ailerons	0.87926	5	0.86224	5	0.35364	80
Delta Elevators	1.2919	5	1.3016	5	0.7771	100
Pole Telecomm	8.0454	5	18.242	10	3.7634	300
Computer Activity	4.2516	10	1.8045	5	1.5155	180
Census (House16H)	17.472	10	16.301	10	9.1687	340
Auto Price	0.31904	5	0.3484	5	0.00188	20
Boston Housing	0.55224	10	0.57064	10	0.01726	70
Pyrimidines	0.3456	5	0.43542	10	0.00062	5
Triazines	0.59346	5	0.53716	5	0.00092	20
Machine CPU	0.2872	5	0.2154	5	0.01068	80
Servo	0.28686	5	0.36566	40	0.00156	20
Breast Cancer	0.46524	5	0.4335	5	0.00062	5
Stocks Domain	0.37216	5	0.55336	10	0.3582	200

3) All of these algorithms face difficulty when the learning tasks change over time, which does often happen in real applications.

In this paper, based on the results obtained for sigmoid networks [11], [12], we have shown in Theorem 3.2 that the input weights and hidden neurons biases for threshold networks can be randomly assigned instead of greedy tuning. Once the input weights and hidden neurons biases are randomly assigned, unlike the conventional gradient-descent based methods which often get stuck in local minima, ELM tends to reach global minimum directly by using Moore–Penrose generalized inverse method. We further showed that the ELM learning algorithm can directly be used to train threshold networks with good generalization performance. This has been verified by the simulation results based on a number of real-world benchmark applications. The results also show that ELM for threshold networks takes lesser training time and obtains stable performance. Compared to other learning algorithms, ELM with a given network architecture requires no parameters to be manually tuned and hence is easy to use, especially for ordinary users who do not have domain knowledge in neural networks. In order to encourage open development, testing and expansion of the ELM, ELM source codes have been released in the ELM website http://www.ntu.edu.sg/home/egbhuang/.

REFERENCES

[1] P. L. Bartlett and T. Downs, "Using random weights to train multilayer networks of hard-limiting units," *IEEE Trans. Neural Netw.*, vol. 3, pp. 202–210, 1992.

[2] E. M. Corwin, A. M. Logar, and W. J. B. Oldham, "An iterative method for training multilayer networks with threshold function," *IEEE Trans. Neural Netw.*, vol. 5, pp. 507–508, 1994.

[3] S. E. Hampson and D. J. Volper, "Representing and learning boolean functions of multivalued features," *IEEE Trans. Syst., Man Cybern.*, vol. 20, pp. 67–80, 1990.

[4] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network," *IEEE Trans. Inf. Theory*, vol. 44, pp. 525–536, 1998.

[5] D. E. Rumellhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distribution Processing: Explanations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[6] D. J. Toms, "Training binary node feedforward neural networks by back-propagation of error," *Electron. Lett.*, vol. 26, no. 21, pp. 1745–1746, 1990.

[7] R. M. Goodman and Z. Zeng, "A learning algorithm for multi-layer perceptrons with hard-limiting threshold units," in *Proc. IEEE Workshop of Neural Networks for Signal Processing*, 1994, pp. 219–228.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'04)*, Budapest, Hungary, Jul., 25–29 2004.

[9] V. P. Plagianakos, G. D. Magoulas, N. K. Nousis, and M. N. Vrahatis, "Training multilayer networks with discrete activation functions," in *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN'01)*, Washington, D.C., U.S.A., 2001.

[10] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Trans. Neural Netw.*, vol. 9, pp. 224–229, 1998.

[11] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Trans. Neural Netw.*, vol. 14, pp. 274–281, 2003.

[12] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layers versus three," *IEEE Trans. Neural Netw.*, vol. 8, pp. 251–255, 1997.

[13] E. B. Baum, "On the capabilities of multilayer perceptrons," *J. Complexity*, vol. 4, pp. 193–215, 1988.