Real-Time Learning Capability of Neural Networks

Guang-Bin Huang, *Senior Member, IEEE*, Qin-Yu Zhu, and Chee-Kheong Siew, *Member, IEEE*

***Abstract*—In some practical applications of neural networks, fast response to external events within an extremely short time is highly demanded and expected. However, the extensively used gradient-descent-based learning algorithms obviously cannot satisfy the *real-time learning needs* in many applications, especially for large-scale applications and/or when higher generalization performance is required. Based on Huang's constructive network model, this paper proposes a simple learning algorithm capable of real-time learning which can automatically select appropriate values of neural quantizers and *analytically* determine the parameters (weights and bias) of the network at one time only. The performance of the proposed algorithm has been systematically investigated on a large batch of benchmark real-world regression and classification problems. The experimental results demonstrate that our algorithm can not only produce good generalization performance but also have real-time learning and prediction capability. Thus, it may provide an alternative approach for the practical applications of neural networks where real-time learning and prediction implementation is required.**

***Index Terms*—Backpropagation (BP), extreme learning machine, feedforward networks, generalization performance, k-NN, real-time learning, real-time prediction.**

I. Introduction

THE widespread popularity of multilayer feedforward networks in many fields is mainly due to their ability 1) to approximate complex unknown nonlinear mappings directly from the input training samples [1]–[8] and 2) to form disjoint decision regions with arbitrary shapes and to determine unknown classes [9]. Whether neural networks can have real-time learning capability is still a challenging and open question. A neural network system is called a real-time learning system if it can finish a learning procedure with good generalization performance for a new application within expected fast response time defined by external requirements. This expected response time could be any reasonable time for an application users can wait. It could be microseconds, milliseconds, or even seconds. Real-time learning capability of neural networks is highly expected whenever a new application is faced, where a new knowledge map has to be built.

Iterative searching methodology has been widely used in the learning algorithms of neural networks [10], e.g., backpropagation (BP) algorithm [11] and its variants [12]–[19]. Since the parameters of the network are updated gradually in each iteration, the learning is apparently time consuming, especially for applications with large-scale observations and/or high accuracy expectation. It is not surprising that this kind of learning algorithm may spend hours on training and may also face trivial issues such as learning parameters (learning epochs, learning rate, etc.), stopping criteria, and/or local minima. Thus, these learning algorithms may face unexpected difficulties in time-critical applications.

Therefore, an alternative algorithm that can implement real-time learning is highly demanded by time-critical applications and fast changing environments, and can be widely used in human–computer interface, unmanned aerial vehicles, robotics, tracking and navigation, meteorology, geographical information processing, expert systems, intelligent networking, etc. Even for offline applications, speed is still a need, and a real-time learning algorithm that reduces training time and human effort to nearly zero would always be of considerable value.

Some researchers have proposed constructive methods for multilayer feedforward networks that can learn the observations just in one iteration. For instance, Huang and Babri [20] show that a single-hidden layer feedforward neural networks (SLFNs) with at most N hidden neurons and with almost any nonlinear activation function can learn N distinct observations with *zero* error. However, the number of hidden neurons required there is too large, especially for large-scale applications, which cannot be supported by the computational capabilities of most current ordinary computers.

Although many applications can be implemented by both SLFNs and two-hidden layer feedforward neural networks (TLFNs) [9], TLFNs may be more powerful than SLFNs in most cases [1], [21], [22]. Huang [1] proved in a novel constructive method that a TLFN can learn any N distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$ with any arbitrarily small error with at most $2\sqrt{(m+2)N} (\ll N)$ hidden neurons, which remarkably reduces the space complexity of a network especially for large-scale problems and makes it feasible to implement large-scale systems in an ordinary computer. In this method, the observations are divided into several groups, and each group is learned by a *common standard* TLFN. Furthermore, this trained common standard TLFN and some additional neuron quantizers are combined to form a special *nonconventional* TLFN, which can finally represent all observations with arbitrarily high accuracy (small error). Huang's network[1] has several features.

1) It has larger first hidden layer and narrower second hidden layer, which comprises a standard TLFN and several neuron quantizers, each linking to the input layer and one or several neurons in the second hidden layer only.
2) All the weights of the connections linking the input layer to the first hidden layer can be simply pre-fixed, and most of them can be assigned *randomly*.

Manuscript received June 21, 2005; revised January 3, 2006.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore (e-mail: gbhuang@ieee.org).

Digital Object Identifier 10.1109/TNN.2006.875974

[1]For the sake of convenience, Huang's constructive network model [1] is abbreviated as "Huang's network" in the context of this paper.

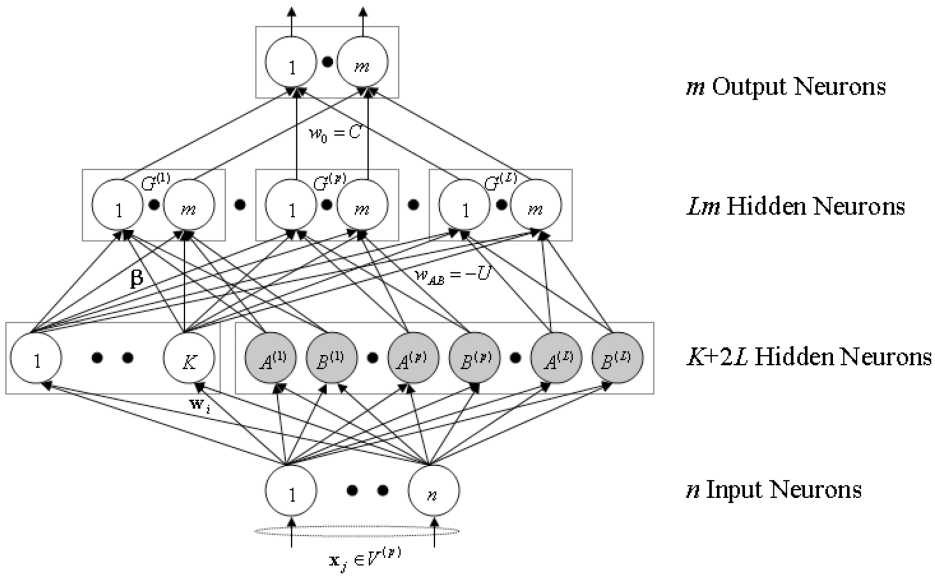

Fig. 1. Huang's network.

3) The weights of the connections linking the first hidden layer and second hidden layer can be determined *analytically* at one time, instead of the conservative iterative adjustment method popularly adopted in most neural network learning algorithms nowadays.
4) The weights of the connections linking the second hidden layer and the output layer can be simply *analytically* determined and remain as a constant value C.
5) It may be trained by only adjusting weight size factor λ or C and quantizer factors T and U so as to optimize generalization performance.

Huang [1] also analyzes that a good generalization performance may be achieved by the network.

During our latest study, the results of some experiments show that the generalization performance of the original constructive method may not be very stable. And the appropriate way to adjust the quantizer factors T and U in Huang's network are not provided yet. In this paper, an appropriate method to estimate the suitable value for the quantizer factors T and U are proposed. Further, an improved learning algorithm for Huang's network is suggested, which can finish training procedure with good generalization performance for many applications within seconds or milliseconds. Thus, it may be able to provide an alternative approach for some practical real-time learning applications. Extensive experiments on different types of benchmark real problems have been done to systematically evaluate the performance (learning and prediction speeds and generalization performance) of the proposed algorithm. The experimental results show that our proposed algorithm can not only produce good generalization performance but also have real-time learning and prediction capability.

This paper is organized as follows. Section II provides a review on Huang's constructive network model [1]. Based on Huang's constructive method, an appropriate parameters selection method and the proposed new algorithm are given in Section III. The comprehensive and comparative experimental results are shown in Sections IV and V. Conclusions are given in Section VI. The meanings of the notations and symbols used in this paper are the same as those in Huang [1].

II. Review of Huang's Constructive Network Model

A. Basic Structure of Huang's Network

It has been shown [1] that Huang's network with sigmoid activation function $g(x) = 1/(1 + e^{-x})$ can approximate N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in \mathbf{R}^m$, with arbitrarily small error. As shown in Fig. 1, Huang's network has larger first hidden layer and narrower second hidden layer, which consists of a standard TLFN and L neuron quantizers in the first hidden layer, each linking to the input layer and second hidden layer of the sub-TLFN, where L is a user-defined constant indicating how many neuron quantizers are used in the network. There are $K(K = N/L)$ neurons in the first hidden layer and mL neurons in the second hidden layer in its standard sub-TLFN. Each neuron quantizer consists of two sigmoidal neurons called A-type neuron and B-type neuron, respectively. A-type neuron and B-type neuron of the pth quantizer are denoted as neuron $A^{(p)}$ and neuron $B^{(p)}$. The pth neuron quantizer only links to the pth neuron group $G^{(p)}$ in the second hidden layer, $p = 1, \ldots, L$, each group $G^{(p)}$ consisting of m neurons. The training of Huang's network mainly consists of two phases: 1) determination of weights and bias of standard sub-TLFN and 2) determination of weights and biases of neural quantizers.

B. Determination of Weights and Bias of Standard Sub-TLFN

One of the major features of Huang's network is that weight $\mathbf{w}_i$ linking the ith neuron in the first hidden layer of the sub-TLFN to all the input neurons and bias b_i of the ith neuron in

the first hidden layer of the sub-TLFN ($i = 1, \ldots, K$, $K = N/L$) can be chosen randomly. Since all the given $\mathbf{x}_i$ are distinctive, a vector $\mathbf{w}$ can be almost randomly chosen such that $\mathbf{w} \cdot \mathbf{x}_1, \ldots, \mathbf{w} \cdot \mathbf{x}_N$ are all different [1], [20]. Without loss of generality, assume that the index i is reindexed such that

$$\mathbf{w} \cdot \mathbf{x}_1 < \mathbf{w} \cdot \mathbf{x}_2 < \ldots < \mathbf{w} \cdot \mathbf{x}_N. \tag{1}$$

For simplicity, consider N to be an integral multiple of user-defined positive integer L. The finite N inputs $\mathbf{x}_i$ are separated into L groups consisting of N/L input vectors each. These L input vector groups are denoted as $V^{(1)}, \ldots, V^{(L)}$

$$V^{(p)} = \left\{\mathbf{x}_i | \mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} \leqslant \mathbf{w} \cdot \mathbf{x}_i \leqslant \mathbf{w} \cdot \mathbf{x}_{pN/L}\right\} \tag{2}$$

where $p = 1, \ldots, L$.

Randomly choose the weight $\mathbf{w}_i$ of the connection linking the input to the ith neuron of the first hidden layer of the sub-TLFN and the bias b_i of the ith neuron of the first hidden layer of the sub-TLFN, $i = 1, \ldots, K$, $K = N/L$.

Define matrix $\boldsymbol{\beta}^{(p)}$ as

$$\boldsymbol{\beta}^{(p)} = \left[\beta_1^{(p)}, \ldots, \beta_K^{(p)}\right]^T \tag{3}$$

where $\beta_i^{(p)}$ denotes the weight vector connecting the ith neuron in the first hidden-layer of the sub-TLFN to the pth neuron group $G^{(p)}$ (m neurons) in the second hidden layer, $i = 1, \ldots, K$, $K = N/L$. $\boldsymbol{\beta}^{(p)}$ can be determined as

$$\boldsymbol{\beta}^{(p)} = \left(\mathbf{H}^{(p)}\right)^{-1} \mathbf{T}^{(p)}. \tag{4}$$

Matrix $\mathbf{T}^{(p)}$, $p = 1, \ldots, L$, is defined in (5), as shown at the bottom of the page, and matrix $\mathbf{H}^{(p)}$, $p = 1, \ldots, L$, is defined in (6), as shown at the bottom of the page, where $K = N/L$ and weight size factor C can be chosen any value to make

$$-0.5 < t_{ij}/C < 0.5. \tag{7}$$

All the weights of connections linking to the output layer are set to C, and bias of the output neuron is set as $-0.5C$.

C. Determination of Weights and Biases of Neural Quantizers

As shown in Fig. 1, the weights of connections linking the inputs to neuron $A^{(p)}$ and neuron $B^{(p)}$ can be chosen as $\overline{\mathbf{w}}_{A^{(p)}} = T \cdot \mathbf{w}$ and $\overline{\mathbf{w}}_{B^{(p)}} = -T \cdot \mathbf{w}$, respectively, where parameter T is called a quantizer factor by Huang [1] and should be large enough. That is, all the weights of connections linking the inputs to all A-type neurons can be chosen as $T \cdot \mathbf{w}$ and all the weights of connections linking the inputs to all B-type neurons chosen as $-T \cdot \mathbf{w}$. The biases $\overline{b}_{A^{(p)}}$ and $\overline{b}_{B^{(p)}}$ of neuron $A^{(p)}$ and neuron $B^{(p)}$ are simply analytically calculated as

$$\overline{b}_{A^{(p)}} = -T\left(\mathbf{w} \cdot \mathbf{x}_{pN/L} + \frac{1}{2}\min_{i,j}|\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{x}_j|\right) \tag{8}$$

and

$$\overline{b}_{B^{(p)}} = T\left(\mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} - \frac{1}{2}\min_{i,j}|\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{x}_j|\right). \tag{9}$$

Quantizer factor T can be chosen large enough such that for any input $\mathbf{x}_i$ within input vector group $V^{(p)}$, $p = 1, \ldots, L$, only the outputs of both neurons $A^{(p)}$ and $B^{(p)}$ are almost zero while one of the outputs of neurons $A^{(l)}$ and $B^{(l)}$ of each neural quantizer is almost one, where $l \neq p$. In other words, T can be chosen large enough such that for each input vector group $V^{(p)}$, only the output of the pth neural quantizer consisting of neurons $A^{(p)}$ and $B^{(p)}$ is almost zero, and the outputs of the other $L-1$ quantizers are almost one.

All the weights w_{AB} of the connections linking these $2L$ neurons $A^{(p)}$ and $B^{(p)}$ to their corresponding second hidden-layer neurons are chosen a same value $w_{AB} = -U$, where parameter U is also called a quantizer factor by Huang [1] and can be set large enough such that the input to the pth neuron group $G^{(p)}$ in the second hidden layer from neurons $A^{(p)}$ and $B^{(p)}$ (pth quantizer) has small values ϵ_i for any input $\mathbf{x}_i$ within input vector group $V^{(p)}$ and large negative values E_i for any input within other vector groups $V^{(l)}$, $p = 1, \ldots, L$, and $l \neq p$. E_i and ϵ_i can be made arbitrarily large and small, respectively, by setting

$$\begin{aligned}\mathbf{T}^{(p)} &= \mathbf{T}\left(\mathbf{t}_{(p-1)\frac{N}{L}+1}, \ldots, \mathbf{t}_{p\frac{N}{L}}\right) \\ &= \begin{bmatrix} \ln\left(\frac{0.5+t_{(p-1)\frac{N}{L}+1,1}/C}{0.5-t_{(p-1)\frac{N}{L}+1,1}/C}\right) & \cdots & \ln\left(\frac{0.5+t_{(p-1)\frac{N}{L}+1,m}/C}{0.5-t_{(p-1)\frac{N}{L}+1,m}/C}\right) \\ \cdots & \cdots & \cdots \\ \ln\left(\frac{0.5+t_{p\frac{N}{L},1}/C}{0.5-t_{p\frac{N}{L},1}/C}\right) & \cdots & \ln\left(\frac{0.5+t_{p\frac{N}{L},m}/C}{0.5-t_{p\frac{N}{L},m}/C}\right) \end{bmatrix}\end{aligned} \tag{5}$$

$$\begin{aligned}\mathbf{H}^{(p)} &= \mathbf{H}\left(\mathbf{x}_{(p-1)\frac{N}{L}+1}, \ldots, \mathbf{x}_{p\frac{N}{L}}, b_1, \ldots, b_K\right) \\ &= \begin{bmatrix} g\left(\mathbf{w}_1 \cdot \mathbf{x}_{(p-1)\frac{N}{L}+1} + b_1\right) & \cdots & g\left(\mathbf{w}_K \cdot \mathbf{x}_{(p-1)\frac{N}{L}+1} + b_K\right) \\ \cdots & \cdots & \cdots \\ g\left(\mathbf{w}_1 \cdot \mathbf{x}_{p\frac{N}{L}} + b_1\right) & \cdots & g\left(\mathbf{w}_K \cdot \mathbf{x}_{p\frac{N}{L}} + b_K\right) \end{bmatrix}\end{aligned} \tag{6}$$

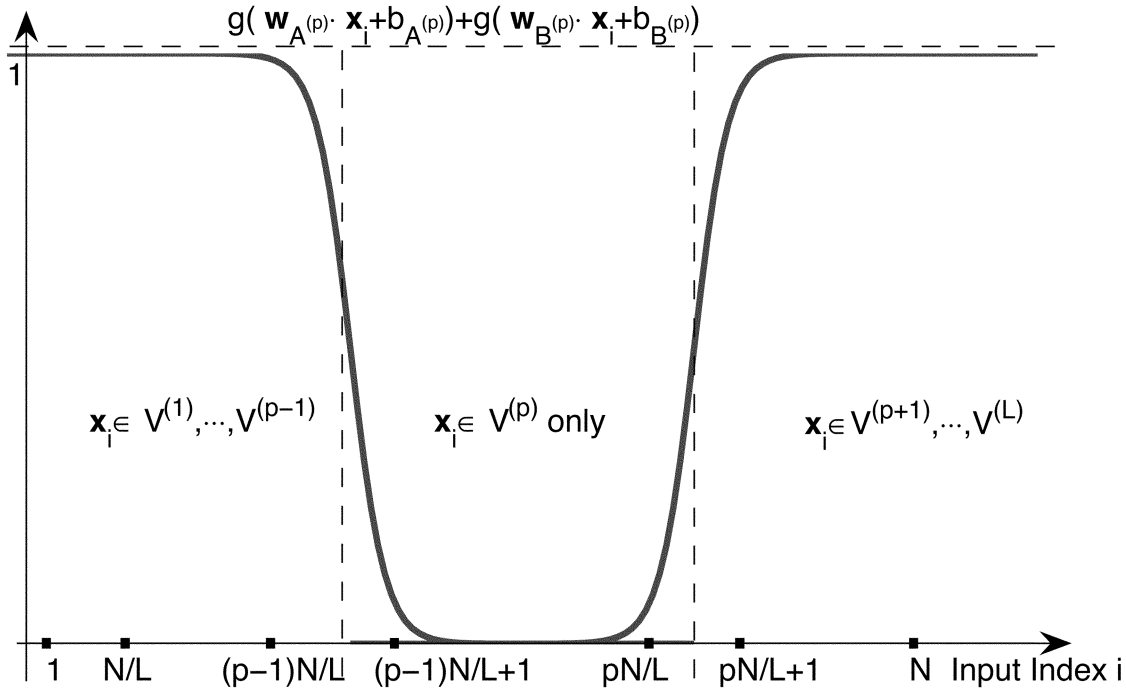

Fig. 2. Original output of pth neural quantizer of Huang's network.

quantizer factors T and U sufficiently large. E_i goes to negative infinity and ϵ_i goes to zero (see Fig. 2).

Seen from the above review, in Huang's network, the weights (matrix) $\boldsymbol{\beta}^{(p)}$ of the connections linking the first hidden layer of the sub-TLFN to the second hidden layer can be analytically calculated, while all the rest parameters (weights and biases) of Huang's network can be randomly preassigned (e.g., $\mathbf{w}_i$ and b_i) or pre-fixed (e.g., C). One may only adjust the quantizer factors T and U which may affect the weights of the neurons of each quantizer. In many applications Huang's network may be fast enough to learn new stimuli from and response to external events within an expected short time. It is capable of real-time learning of new knowledge in large-scale time-critical applications; thus, Huang's constructive method and an appropriate estimation of quantizer factors can form a real-time learning algorithm (RLA).

III. Proposed Real-Time Learning Algorithm

A. Improved Biases Selection of Neural Quantizers

During the latest study, it is found that the originally proposed selection [1] of biases of neurons $A^{(p)}$ and $B^{(p)}$ [see (8) and (9)] can be further improved in order to make the generalization performance of Huang's network much more stable. It is guaranteed that by the originally proposed selection of biases of neurons $A^{(p)}$ and $B^{(p)}$, any training input vector $\mathbf{x}_i$ will be always within only one of groups $V^{(p)}$, the output of one and only one neural quantizer is almost zero, and the outputs of the rest neural quantizers are almost one. Thus, the training error can be made arbitrarily small using the originally proposed biases selection (refer to Section II-C for detailed discussions). However, when a testing input vector $\mathbf{x}_0$ happens to satisfy $\mathbf{w}\cdot\mathbf{x}_{pN/L}+(1/2)\min_{i,j}|\mathbf{w}\cdot\mathbf{x}_i-\mathbf{w}\cdot\mathbf{x}_j|<\mathbf{w}\cdot\mathbf{x}_0<\mathbf{w}\cdot\mathbf{x}_{pN/L+1}-(1/2)\min_{i,j}|\mathbf{w}\cdot\mathbf{x}_i-\mathbf{w}\cdot\mathbf{x}_j|$, the outputs of two neural quantizers would be almost one, instead of one neural quantizer only, which causes the testing error to be possibly larger and affects the generalization performance of the network [see Fig. 3(a)].

To prevent this worse case happening and to achieve a better generalization performance, the selection of the biases $\overline{b}_{A(p)}$ and $\overline{b}_{B^{(p)}}$ of neural quantizers can be improved as shown in (10) and (11) at the bottom of the page. By this bias selection of neural quantizers, the probability that a testing input vector $\mathbf{x}_0$ happens to make $\mathbf{w}\cdot\mathbf{x}_0=(1/2)\mathbf{w}\cdot\mathbf{x}_{pN/L+1}+(1/2)\mathbf{w}\cdot\mathbf{x}_{pN/L}$ should be zero. Thus, for any testing input vector $\mathbf{x}_0$, the output of one and only one neural quantizer is almost zero and the outputs of the rest neural quantizers are almost one, which improves the generalization performance of the network [see Fig. 3(b)].

$$\overline{b}_{A^{(p)}}=\begin{cases}-T\left(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L}+\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L+1}\right), & \text{if } p\neq L\\ -T\left(\mathbf{w}\cdot\mathbf{x}_N+\max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)\right), & \text{if } p=L.\end{cases} \tag{10}$$

$$\overline{b}_{B^{(p)}}=\begin{cases}T\left(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L+1}+\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L}\right), & \text{if } p\neq 1\\ T\left(\mathbf{w}\cdot\mathbf{x}_1-\max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)\right), & \text{if } p=1.\end{cases} \tag{11}$$

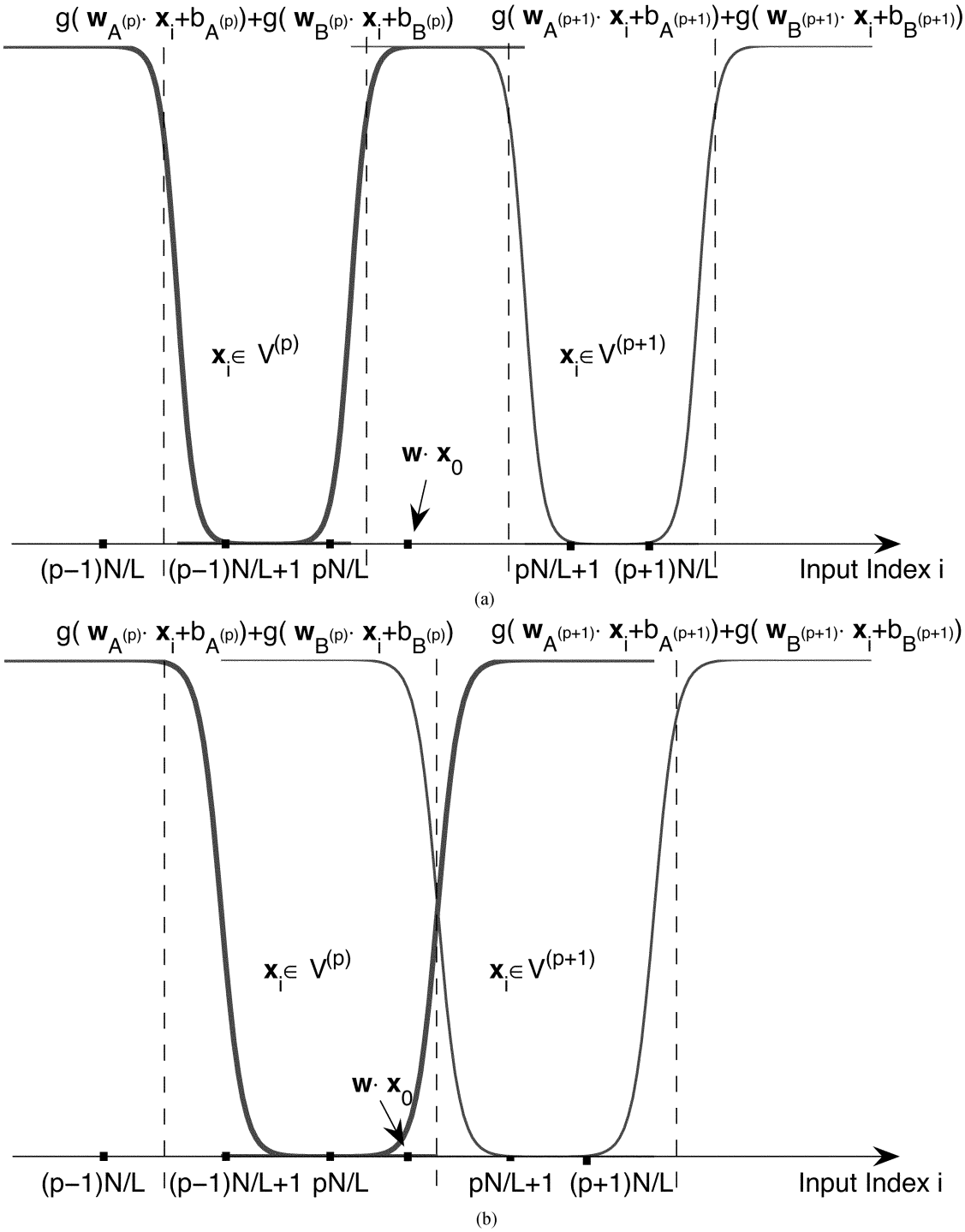

Fig. 3. Outputs of two contiguous neural quantizers. (a) Using previous selection of $\overline{b}_{A^{(p)}}$ and $\overline{b}_{B^{(p)}}$ there may exist a big gap not covered by both contiguous quantizers. (b) Using improved selection criteria, in theory the size of the gap not covered by both contiguous quantizers becomes zero.

B. Appropriate Weights Selection of Neural Quantizers

In order to determine the weights of neurons $A^{(p)}$ and $B^{(p)}$, one only needs to determine appropriate values for quantizer factors T and U. In fact, quantizer factors T and U are the only parameters to be estimated for Huang's network. In this section, an appropriate method to estimate the suitable values of the quantizer factors T and U is proposed. These values can be decided based on Theorem III.3. Two lemmas are needed to derive Theorem III.3.

Lemma III.1: Given an arbitrarily small positive value $\eta < 0.5$, there exists a constant T_0^A

$$T_0^A = \frac{2\ln\left(\frac{1-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$$

such that when $T \geqslant T_0^A$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the corresponding output $O_{A^{(p)}}(\mathbf{x}_i)$ of neuron $A^{(p)}$ satisfies

$$O_{A^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \eta, & \text{if } p \geqslant k \\ \geqslant 1-\eta, & \text{if } p < k \end{cases}. \tag{12}$$

Proof: It can be proved in three cases.

Case 1: when $p \geqslant k$ and $p \neq L$.

According to (10), since

$$\begin{aligned} O_{A^{(p)}}(\mathbf{x}_i) &= \frac{1}{1+e^{-\left(T\mathbf{w}\cdot\mathbf{x}_i+\overline{b}_A^{(p)}\right)}} \\ &= \frac{1}{1+e^{-T\left(\mathbf{w}\cdot\mathbf{x}_i-\frac{1}{2}\left(\mathbf{w}\cdot\mathbf{x}_{p\frac{N}{L}+1}+\mathbf{w}\cdot\mathbf{x}_{p\frac{N}{L}}\right)\right)}} \end{aligned} \tag{13}$$

to make $O_{A^{(p)}}(\mathbf{x}_i) \leqslant \eta$, T should satisfy

$$T\left(\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})\right) \leq \ln\left(\frac{\eta}{1-\eta}\right)$$

$\forall \mathbf{x_i} \in V^{(k)}$, and $p \geqslant k$, $\mathbf{w}\cdot\mathbf{x}_i \leqslant \mathbf{w}\cdot\mathbf{x}_{pN/L}$, then T should satisfy

$$T \geqslant \frac{\ln\left(\frac{\eta}{1-\eta}\right)}{\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})}. \tag{14}$$

Since obviously we have

$$\begin{aligned} &\frac{\ln\left(\frac{\eta}{1-\eta}\right)}{\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})} \\ &\qquad \leq \frac{2\ln\left(\frac{1-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \end{aligned}$$

we can set $T_0^{A1} = (2\ln(1-\eta/\eta)/\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j))$ such that when $T \geqslant T_0^{A1}$ (14) holds.

Case 2: when $p \geqslant k$ and $p = L$.

According to (10)

$$\begin{aligned} &O_{A^{(p)}}(\mathbf{x}_i) \\ &\quad = \frac{1}{1+e^{-(T\mathbf{w}\cdot\mathbf{x}_i - T(\mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)))}}. \end{aligned}$$

Similarly, to make $O_{A^{(p)}}(\mathbf{x}_i) \leqslant \eta$, T should satisfy

$$T \geq \frac{\ln\left(\frac{1-\eta}{\eta}\right)}{-\mathbf{w}\cdot\mathbf{x}_i + \mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}. \tag{15}$$

However, obviously we have

$$\begin{aligned} &\frac{\ln\left(\frac{1-\eta}{\eta}\right)}{-\mathbf{w}\cdot\mathbf{x}_i + \mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \\ &\qquad < \frac{\ln\left(\frac{1-\eta}{\eta}\right)}{\max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}. \end{aligned} \tag{16}$$

Thus, we can set $T_0^{A2} = (\ln(1-\eta/\eta)/\max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j))$ such that when $T \geqslant T_0^{A2}$ (15) holds.

Case3: when $p < k$. Similarly, according to (10), to make $O_{A^{(p)}}(\mathbf{x}_i) \geqslant 1-\eta$, T should satisfy

$$T \geqslant \frac{\ln\left(\frac{1-\eta}{\eta}\right)}{\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})}. \tag{17}$$

However, obviously we have

$$\begin{aligned} &\frac{\ln\left(\frac{1-\eta}{\eta}\right)}{\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})} \\ &\qquad \leq \frac{2\ln\left(\frac{1-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \end{aligned} \tag{18}$$

thus, we can set $T_0^{A3} = (2\ln(1-\eta/\eta)/\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j))$ such that when $T \geqslant T_0^{A3}$ (17) holds.

According to Cases 1–3, finally we can set

$$\begin{aligned} T_0^A &= \max\left\{T_0^{A1}, T_0^{A2}, T_0^{A3}\right\} \\ &= \frac{2\ln\left(\frac{1-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \end{aligned}$$

when $T \geqslant T_0^A$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, and any vector $\mathbf{w}$ satisfying $\mathbf{w}\cdot\mathbf{x}_i \neq \mathbf{w}\cdot\mathbf{x}_j$ $(i \neq j)$, the corresponding output $O_{A^{(p)}}(\mathbf{x}_i)$ of neuron $A^{(p)}$, $p = 1, \ldots, L$, satisfies (12). This completes the proof. ■

Similarly, we have the following lemma.

Lemma III.2: Given an arbitrarily small positive value $\eta < 0.5$, there exists a constant T_0^B

$$T_0^B = \frac{2\ln\left(\frac{1-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$$

such that when $T \geqslant T_0^B$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the corresponding output $O_{B^{(p)}}(\mathbf{x}_i)$ of neuron $B^{(p)}$ satisfies

$$O_{B^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \eta, & \text{if } p \leqslant k \\ \geqslant 1-\eta, & \text{if } p > k \end{cases}. \tag{19}$$

Proof: The proof is linearly similar to the proof of Lemma III.1. ■

From Lemmas III.1 and III.2, we have the following theorem.

Theorem III.1: Given an arbitrarily small positive value $\eta < 1$, there exists a constant T_0

$$T_0 = \frac{2\ln\left(\frac{2-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$$

such that when $T \geqslant T_0$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the output $O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i)$ of pth neural quantizer satisfies

$$O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \eta, & \text{if } p = k \\ \geqslant 1-\eta, & \text{if } p \neq k \end{cases}. \tag{20}$$

Proof: Set T_0 as

$$T_0 = \frac{2\ln\left(\frac{2-\eta}{\eta}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}.$$

According to Lemma III.1 when $T \geqslant T_0$, the output $O_{A^{(p)}}(\mathbf{x}_i)$ of neuron $A^{(p)}$ satisfies

$$O_{A^{(p)}}(\mathbf{x}_i)\begin{cases}\leqslant \frac{1}{2}\eta, & \text{if } p \geqslant k\\ \geqslant 1-\frac{1}{2}\eta, & \text{if } p < k\end{cases} \tag{21}$$

and according to Lemma III.2, when $T \geqslant T_0$, the output $O_{B^{(p)}}(\mathbf{x}_i)$ of neuron $B^{(p)}$ satisfies

$$O_{B^{(p)}}(\mathbf{x}_i)\begin{cases}\leqslant \frac{1}{2}\eta, & \text{if } p \leqslant k\\ \geqslant 1-\frac{1}{2}\eta, & \text{if } p > k\end{cases}. \tag{22}$$

Thus, when $T \geqslant T_0$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the output $O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i)$ of pth neural quantizer satisfies (20). ■

Remark 1: Theorem III.1 actually states, consistent with the design concept of Huang's network [1], that when the quantizer factor $T > T_0$, the output of one and only one neural quantizer is near to zero and the outputs of the rest are near to one. T_0 is the lower bound of the quantizer factor T. Now, we can further determine the appropriate value for quantizer factor U. For the sake of simplicity and convenience, we can consider the single output case first.

Theorem III.2: Give arbitrarily small positive value $\epsilon < 1$, the quantizer factors T and U can be set as

$$U = \ln\left(\frac{2CL}{\epsilon} - 1\right) + \max_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}} \left\{\mathbf{H}_q^{(p)} \cdot \beta^{(s)}\right\} + \min_{i=1,\ldots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right) \tag{23}$$

$$T = \frac{2\ln\left(2U/\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)-1\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)} \tag{24}$$

such that $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the output $O(\mathbf{x}_i)$ of Huang's network for single output case satisfies

$$|O(\mathbf{x}_i) - t_i| \leqslant \epsilon \tag{25}$$

where $\mathbf{H}_j^{(k)}$ denotes the jth row of the $\mathbf{H}^{(k)}$ matrix.

Proof: According to Theorem III.1, set T as

$$T = \frac{2\ln\left(\frac{2-\epsilon_0}{\epsilon_0}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \tag{26}$$

the output $A^{(p)} + B^{(p)}$ of the pth neural quantizer satisfies

$$O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i)\begin{cases}\leqslant \epsilon_0, & \text{if } p = k\\ \geqslant 1-\epsilon_0, & \text{if } p \neq k\end{cases}$$

where ϵ_0 is a small positive value which will be decided later.

Let $o_j(\mathbf{x}_i)$ denote the actual output of jth neuron in the second hidden layer for the corresponding input $\mathbf{x}_i$, $i = 1, \ldots, N$ and $j = 1, \ldots, L$. Since $O(\mathbf{x}_i) = C(\sum_{j=1}^{L} o_j(\mathbf{x}_i) - 0.5)$, we have

$$|O(\mathbf{x}_i) - t_i| \leqslant |Co_k(\mathbf{x}_i) - 0.5C - t_i| + \left|C\sum_{j=1, j\neq k}^{L} o_j(\mathbf{x}_i)\right|.$$

Hence, if we can make

$$|Co_k(\mathbf{x}_i) - 0.5C - t_i| \leqslant \frac{\epsilon}{2} \tag{27}$$

and

$$\left|C\sum_{j=1, j\neq k}^{L} o_j(\mathbf{x}_i)\right| \leqslant \frac{\epsilon}{2} \tag{28}$$

then (25) can be satisfied. Since $U(O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i)) > 0$ we have

$$\begin{aligned} Co_k(\mathbf{x}_i) - 0.5C - t_i =& \frac{C}{1+e^{-\left(t_i^{(k)} - U\left(O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i)\right)\right)}} \\ & - \frac{C}{1+e^{-t_i^{(k)}}} \\ <& 0 \end{aligned} \tag{29}$$

where $t_i^{(k)} = \ln((0.5 + t_i/C)/(0.5 - t_i/C))$, then (27) can be transformed into

$$\begin{aligned} Co_k(\mathbf{x}_i) \geqslant & -0.5\epsilon + 0.5C + t_i \\ \frac{C-2t_i}{C+2t_i} e^{U\left(O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i)\right)} \leqslant & \frac{C+\epsilon-2t_i}{C-\epsilon+2t_i}. \end{aligned} \tag{30}$$

[In fact, (29) and (30) are valid as long as $\mathbf{H}^{(k)}\boldsymbol{\beta}^{(k)}$ is close enough to $\mathbf{T}^{(k)}$ instead of $\mathbf{H}^{(k)}\boldsymbol{\beta}^{(k)} = \mathbf{T}^{(k)}$, $k = 1, \ldots, L$.]

Since $((C - 2t_i)/(C + 2t_i)) > 0$, we have

$$e^{U\left(O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i)\right)} \leq \frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}$$

then we have

$$U \leqslant \frac{\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)}{O_{A^{(k)}}(\mathbf{x}_i) + O_{B^{(k)}}(\mathbf{x}_i)}. \tag{31}$$

From Theorem III.1, we know $0 < O_{A^{(k)}}(\mathbf{x}_i) + O_{B^{(k)}}(\mathbf{x}_i) \leqslant \epsilon_0$. Hence, if we can make

$$U \leqslant \frac{1}{\epsilon_0} \min_{i=1,\ldots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right) \tag{32}$$

then (27) can be satisfied.

To satisfy (28), if for any j from 1 to L and $j \neq k$, there exists a U to make

$$|o_j(\mathbf{x}_i)| \leqslant \frac{\epsilon}{2CL} \tag{33}$$

(28) can be satisfied. Since $o_j(\mathbf{x}_i) > 0$, (33) is simply equivalent to

$$o_j(\mathbf{x}_i) \leqslant \frac{\epsilon}{2CL} \tag{34}$$

that is

$$\frac{1}{1+e^{-\left(\mathbf{H}_{i-(k-1)L}^{(k)}\cdot\beta^{(j)} - U\left(O_{A^{(j)}}(\mathbf{x}_i)+O_{B^{(j)}}(\mathbf{x}_i)\right)\right)}} \leqslant \frac{\epsilon}{2CL} \tag{35}$$

$$U \geqslant \frac{\ln\left(\frac{2CL}{\epsilon} - 1\right) + \mathbf{H}_{i-(k-1)L}^{(k)} \cdot \beta^{(j)}}{O_{A^{(j)}}(\mathbf{x}_i) + O_{B^{(j)}}(\mathbf{x}_i))}. \tag{36}$$

[In fact, (35) and (36) are valid as long as $\mathbf{H}^{(k)}\boldsymbol{\beta}^{(k)}$ is close enough to $\mathbf{T}^{(k)}$ instead of $\mathbf{H}^{(k)}\boldsymbol{\beta}^{(k)} = \mathbf{T}^{(k)}$, $k = 1, \ldots, L$.]

Since

$$\frac{\ln\left(\frac{2CL}{\epsilon}-1\right)+\mathbf{H}^{(k)}_{i-(k-1)L}\cdot\beta^{(j)}}{O_{A^{(j)}}(\mathbf{x}_i)+O_{B^{(j)}}(\mathbf{x}_i))} \leq \frac{\ln\left(\frac{2CL}{\epsilon}-1\right)+\max\limits_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\{\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\}}{1-\epsilon_0}$$

if U can satisfy

$$U \geqslant \frac{\ln\left(\frac{2CL}{\epsilon}-1\right)+\max\limits_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\{\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\}}{1-\epsilon_0} \quad (37)$$

then (28) can be satisfied. Considering (32) and (37), ϵ_0 should be chosen such that

$$\frac{\ln\left(\frac{2CL}{\epsilon}-1\right)+\max_{p=1,\ldots,L,q=1,\ldots,N/L,s=1,\ldots,L}\left\{\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\}}{1-\epsilon_0} \leqslant \frac{1}{\epsilon_0}\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right). \quad (38)$$

Thus, ϵ_0 can be simply chosen as shown in (39) at the bottom of the page. Furthermore, seen from (37), U can be simply chosen as

$$U=\ln\left(\frac{2CL}{\epsilon}-1\right)+\max_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\{\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\} +\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right) \quad (40)$$

and from (26) we have

$$T=\frac{2\ln\left(1+2\frac{\ln\left(\frac{2CL}{\epsilon}-1\right)+\max\limits_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\{\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\}}{\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)}\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)}. \quad (41)$$

This completes the proof. ■

Remark 2: Observe from the proof of Theorem III.2 that there is a tradeoff of quantizer factor U. As reviewed in Section II-C, quantizer factor U cannot be set too small. In fact, the lower bound of U has been given in (36). Although Huang [1] requires U large enough, as shown in (32), there is also an upper bound on U, meaning U cannot be set too large either. If U is set too large, the actual outputs of all the neurons in the second hidden layer may be nearly zero and all the output may be inhibited. Thus, Theorem III.2 gives an appropriate value for quantizer factor U.

Theorem III.2 can be easily extended to the multioutput case:

TABLE I
SPECIFICATIONS OF SMALL REAL-WORLD REGRESSION PROBLEMS

Datasets	# Observations		# Attributes	
	Training	Testing	continuous	norminal
Auto Price	80	79	14	1
Boston Housing	250	256	13	0
Pyrimidines	40	34	27	0
Triazines	100	86	60	0
Machine CPU	100	109	6	0
Servo	80	87	0	4
Breast Cancer	100	94	32	0
Stocks	450	500	10	0

Theorem III.3:
Given arbitrarily small positive value $\epsilon < 1$, the quantizer factors T and U can be set as

$$U=\ln\left(\frac{2\sqrt{m}CL}{\epsilon}-1\right)+\max_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\|\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\|_\infty +\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right) \quad (42)$$

$$T=\frac{2\ln\left(2U/\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right)-1\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)} \quad (43)$$

such that $\forall\mathbf{x}_i \in V^{(k)}$, $k=1,\ldots,L$, the output $\mathbf{O}(\mathbf{x}_i)$ of Huang's network satisfies

$$\|\mathbf{O}(\mathbf{x}_i)-\mathbf{t}_i\| \leqslant \epsilon. \quad (44)$$

Proof: According to Theorem III.2, replacing ϵ with $\epsilon/\sqrt{m}$, we can set

$$U=\ln\left(\frac{2\sqrt{m}CL}{\epsilon}-1\right)+\max_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\|\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\|_\infty +\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right) \quad (45)$$

$$\epsilon_0=\frac{\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)}{\ln\left(\frac{2CL}{\epsilon}-1\right)+\max\limits_{\substack{p=1,\ldots,L\\ q=1,\ldots,N/L\\ s=1,\ldots,L}}\left\{\mathbf{H}^{(p)}_q\cdot\beta^{(s)}\right\}+\min\limits_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)} \quad (39)$$

$$T=\frac{2\ln\left(2U/\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right)-1\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)} \tag{46}$$

$\forall\mathbf{x}_i$, we have

$$|O_j(\mathbf{x}_i)-t_{ij}|\leqslant\frac{\epsilon}{\sqrt{m}}$$

where O_j denotes the output of the jth output neuron of Huang's network. Then we have

$$\|\mathbf{O}(\mathbf{x}_i)-\mathbf{t}_i\|=\sqrt{\sum_{l=1}^{m}|O_j(\mathbf{x}_i)-t_{ij}|^2}\leqslant\epsilon.$$

■

Remark 3: Theorem III.3 gives a proper way to estimate the quantizer factor T and U based on the given goal of error ϵ.

C. Determination of Weight $\boldsymbol{\beta}$

In Huang's constructive method, the weight $\boldsymbol{\beta}$ of sub-TLFN is determined as

$$\boldsymbol{\beta}^{(p)}=\left(\mathbf{H}^{(p)}\right)^{-1}\mathbf{T}^{(p)},\qquad p=1,\ldots,L. \tag{47}$$

Huang [1] proved that the $\mathbf{H}^{(p)}$ is invertible when the number of neurons in the first hidden layer of sub-TLFN is N/L, which means that the total number of hidden neurons is $N/L+(m+2)L$. However, $N/L+(m+2)L$ hidden neurons may still be large for some problems, especially for those with large training set. In fact, zero training error is not required in real applications. Huang *et al.* [23] has the following result for SLFNs.[2]

Theorem III.4: [23] For an SLFN with the activation function $g(x)=1/(1+e^{-x})$ in the hidden layer, given any constant $\epsilon>0$, there always exists an integer $K\leq N/L$ such that an SLFN with K hidden neurons and with randomly chosen input weights and hidden biases can learn N/L distinct observations with a training error less than ϵ.

Actually, the validity of this theorem is obvious; otherwise one can simply choose $K=N/L$. However, $\mathbf{H}^{(p)}$ may not be invertible when $K\leq N/L$. In this case, for SLFNs, the weights linking the hidden layer and output layer can be calculated by the Moore–Penrose generalized inverse [25], [26]. Hence, in the proposed new learning algorithm, the weights β can be calculated as follows:

$$\boldsymbol{\beta}^{(p)}=\left(\mathbf{H}^{(p)}\right)^{\dagger}\mathbf{T}^{(p)},\qquad p=1,\ldots,L \tag{48}$$

where $(\mathbf{H}^{(p)})^{\dagger}$ is the Moore–Penrose generalized inverse of $\mathbf{H}^{(p)}$. Proofs of Theorems III.2 and III.3 do not require $\mathbf{H}^{(p)}$ invertible. Seen from all (29), (30), (35), and (36), which actually use $\mathbf{H}^{(p)}$, $\mathbf{H}^{(p)}\boldsymbol{\beta}^{(p)}$ needs only to be close enough to $\mathbf{T}^{(p)}$ instead of the strict condition $\mathbf{H}^{(p)}\boldsymbol{\beta}^{(p)}=\mathbf{T}^{(p)}$, $p=1,\ldots,L$. According to Theorem III.4, Theorems III.2 and III.3 are valid for some appropriate number $K\leq N/L$. The number of neurons in the first hidden layer of sub-TLFN no longer needs to be N/L but some appropriate number K which can be set by users. Thus, it can help eliminate overfitting in noisy data sets.

[2]One may refer to Huang *et al.* [24] for the rigorous proof of the universal approximation capability of such SLFNs with random hidden nodes.

D. Proposed Real-Time Learning Algorithm

Based on Huang's constructive method (see Section II) and the estimation on quantizer factors (Theorem III.3), a simple RLA can be proposed as follows.

The Proposed RLA Algorithm

Input:

1) Given N arbitrary distinct samples $(\mathbf{x}_i,\mathbf{t}_i)$, where $\mathbf{x}_i=[x_{i1},x_{i2},\ldots,x_{in}]^T\in\mathbf{R}^n$ and $\mathbf{t}_i=[t_{i1},t_{i2},\ldots,t_{im}]^T\in\mathbf{R}^m$.
2) Given the expected learning accuracy $\epsilon<0$.
3) Given the number of groups L.
4) Given the number of neurons K of the first hidden layer of the sub-TLFN.

Learning procedure of RLA

step 1: **Sorting and grouping inputs.**

a) Random choose vector $\mathbf{w}\in\mathbf{R}^n$ and reindex i of inputs such that $\mathbf{w}\cdot\mathbf{x}_1<\mathbf{w}\cdot\mathbf{x}_2<\ldots<\mathbf{w}\cdot\mathbf{x}_N$.

b) Group sorted inputs into L groups $V^{(p)}$, $p=1,\ldots,L$

$$V^{(p)}=\left\{\mathbf{x}_i|\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L+1}\leqslant\mathbf{w}\cdot\mathbf{x}_i\leqslant\mathbf{w}\cdot\mathbf{x}_{pN/L}\right\} \tag{49}$$

step 2: **Determination of weights and bias of standard sub-TLFN.**

a) Randomly choose the weights $\mathbf{w}_i$ and biases b_i, $i=1,\ldots,K$, where $K\leqslant N/L$ is the number of neurons in the first hidden layer of sub-TLFN and it is set by user.

b) Choose $C=a\max_{i=1,\ldots,N;j=1,\ldots,m}|t_{ij}|$, a can be any positive value larger than 2.

c) Calculate matrix $\boldsymbol{\beta}^{(p)}=[\beta_1^{(p)},\ldots,\beta_K^{(p)}]^T$:

$$\boldsymbol{\beta}^{(p)}=\left(\mathbf{H}^{(p)}\right)^{\dagger}\mathbf{T}^{(p)} \tag{50}$$

step 3: **Determination of weights and bias of neural quantizers.**

a) Set the quantizer factor T and U as

$$T=\frac{2\ln\left(2U/\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right)-1\right)}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)} \tag{51}$$

$$U=\ln\left(\frac{2\sqrt{m}CL}{\epsilon}-1\right)+\max_{\substack{p=1,\ldots,L\\q=1,\ldots,N/L\\s=1,\ldots,L}}\left\|\mathbf{H}_q^{(p)}\cdot\beta^{(s)}\right\|_{\infty}+\min_{i=1,\ldots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right) \tag{52}$$

b) Set the weights $\mathbf{w}_{A(p)}$ and $\mathbf{w}_{B(p)}$ of the connections linking the input layer to neurons $A^{(p)}$ and $B^{(p)}$, $p=1,\ldots,L$, as

$$\begin{cases}\mathbf{w}_{A^{(p)}}=T\cdot\mathbf{w}\\ \mathbf{w}_{B^{(p)}}=-T\cdot\mathbf{w}.\end{cases} \tag{53}$$

c) Set the biases of neurons $A^{(p)}$ and $B^{(p)}$, $p=1,\ldots,L$ as shown in (54) and (55) at the bottom of the next page

TABLE II
PERFORMANCE COMPARISON OF RLA AND BP ON SMALL REAL-WORLD REGRESSION PROBLEMS: TESTING ROOT MEAN SQUARE ERROR (RMSE) AND ITS STANDARD DEVIATION (DEV), TRAINING AND TESTING TIME (SECONDS), AND NETWORK ARCHITECTURES

Datasets	Algorithms	(L, K) or K	Testing		Time (s)	
			RMSE	SD	Training	Testing
Auto Price	RLA	(2, 10)	0.1059	0.0124	0.0002	0.0001
	BP	5	0.1157	0.0245	0.2456	0.0066
Boston Housing	RLA	(2, 30)	0.0950	0.0078	0.0030	0.0014
	BP	5	0.0976	0.0051	0.3755	0.006
Pyrimidines	RLA	(2, 10)	0.1443	0.0408	0.0031	0.0016
	BP	5	0.1391	0.0550	0.2675	0.0152
Triazines	RLA	(2, 10)	0.2107	0.0208	0.0015	0.001
	BP	5	0.2197	0.0564	0.5481	0.0145
Machine CPU	RLA	(2, 10)	0.0651	0.02560	0.0001	0.0001
	BP	10	0.0826	0.0165	0.2354	0.0084
Servo	RLA	(2, 15)	0.1264	0.0279	0.0003	0.0001
	BP	10	0.1276	0.0754	0.2447	0.0166
Breast Cancer	RLA	(2, 5)	0.2736	0.0135	0.0004	0.0002
	BP	5	0.3155	0.0891	0.3856	0.0073
Stocks Domain	RLA	(2, 60)	0.0317	0.0023	0.0314	0.0109
	BP	20	0.0358	0.0056	1.0487	0.0105

d) Set the weights w_{AB} of the connections linking neurons $A^{(p)}$ and $B^{(p)}$ to the second hidden layer as

$$w_{AB} = -U \tag{56}$$

Remark 4:

Observe that the basic algorithm of Huang's network [1] has the same form as the new proposed RLA algorithm except that for the basic algorithm 1) the number of neurons K of the first hidden layer of the sub-TLFN was set N/L instead of some appropriate number less than N/L, and thus $\mathbf{H}^{(p)}$ of (50) was always a square matrix; 2) T and U are manually predefined by users instead of automatically computed as in (51) and (52); and 3) the biases of neurons $A^{(p)}$ and $B^{(p)}$ were set according to (8) and (9) instead of (54) and (55). In RLA, the limitation that the number of neurons K of the first hidden layer of the sub-TLFN is equal to N/L (number of training data in each group $V^{(p)}$) has been removed. Setting K to some appropriate number less than N/L can not only improve the generalization performance by preventing RLA from overfitting in noisy data sets but also further reduce the network complexity of Huang's network.

Remark 5: The proposed algorithm requires very small memory in computation. 1) Huang's network is not a fully-connected standard TLFN. It is a sparse network and it has small number of trainable parameters. 2) The L matrix functions $\mathbf{H}^{(p)}$ in (50) share and are computed from a single common

$$\overline{b}_{A^{(p)}} = \begin{cases} -T\left(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{p\frac{N}{L}} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{p\frac{N}{L}+1}\right), & \text{if } p \neq L \\ -T\left(\mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)\right), & \text{if } p = L \end{cases} \tag{54}$$

$$\overline{b}_{B^{(p)}} = \begin{cases} T\left(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)\frac{N}{L}+1} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)\frac{N}{L}}\right), & \text{if } p \neq 1 \\ T\left(\mathbf{w}\cdot\mathbf{x}_1 - \max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)\right), & \text{if } p = 1 \end{cases} \tag{55}$$

TABLE III
SPECIFICATIONS OF MEDIUM TO LARGE REAL-WORLD REGRESSION PROBLEMS

Datasets	# Observations		# Attributes	
	Training	Testing	continuous	norminal
Abalone	2000	2177	7	1
Delta Ailerons	3000	4129	6	0
Delta Elevators	4000	5517	6	0
2D Planes	10000	30768	10	0
Kinematics of Robot Arm	4000	4192	8	0
Census (House8L)	10000	12784	8	0
Pumadyn	4500	3692	8	0
Bank	4500	3692	8	0
California Housing	8000	12460	8	0

TABLE IV
PERFORMANCE COMPARISON OF RLA AND BP ON MEDIUM TO LARGE REAL-WORLD REGRESSION PROBLEMS: TESTING RMSE AND ITS STANDARD DEVIATION (DEV), TRAINING AND TESTING TIME (SECONDS), AND NETWORK ARCHITECTURES

Datasets	Algorithms	(L, K)	Testing		Time (s)	
		or K	RMSE	Dev	Training	Testing
Abalone	RLA	(5, 10)	0.0851	0.0089	0.0108	0.0171
	BP	10	0.0874	0.0055	1.7562	0.0514
Delta Ailerons	RLA	(5, 20)	0.0467	0.0062	0.039	0.0406
	BP	10	0.0481	0.0013	2.7525	0.0140
Delta Elevators	RLA	(5, 20)	0.0575	0.0055	0.0468	0.0563
	BP	5	0.0592	0.0087	1.1939	0.0261
2D Planes	RLA	(5, 360)	0.0452	0.0010	16.513	2.6219
	BP	5	0.0429	0.0036	21.67	0.1048
Kinematics	RLA	(5, 300)	0.0876	0.0061	6.0594	0.289
	BP	90	0.0741	0.0013	134.98	0.1847
Census (house8L)	RLA	(5, 40)	0.0677	0.0019	0.3281	0.1954
	BP	10	0.0685	0.0038	8.0647	0.0426
Pumadyn Domains	RLA	(5, 140)	0.1325	0.0044	1.1766	0.1282
	BP	5	0.1474	0.0052	0.2735	0.0103
Bank Domains	RLA	(5, 80)	0.0359	0.0012	0.3966	0.0841
	BP	20	0.0379	0.0041	7.506	0.0643
California Housing	RLA	(5, 60)	0.1271	0.0027	0.5422	0.2469
	BP	10	0.1285	0.0088	6.532	0.8376

TABLE V
SPECIFICATIONS OF REAL-WORLD CLASSIFICATION PROBLEMS

Datasets	# Attributes	# Classes	# Observations	
			Training	Testing
Glass	9	7	110	104
Ionosphere	33	2	175	176
Liver Disorders	6	2	170	175
Page Blocks	10	5	2,700	2,773
Pima	8	2	380	388
Vehicle	18	4	420	426
Heart	13	2	135	135
German	24	2	500	500
Image Segmentation	19	7	1,100	1,210
Satellite Image	36	6	4,400	2,035
Letter Recognition	16	26	10,000	10,000
Forest Cover Type	54	2	100,000	481,012
Spam Emails	57	2	3,000	1,601

$N/L \times K$ matrix $\mathbf{H}$, instead of L separated $N/L \times K$ matrices stored in the memory. The L matrix functions $\mathbf{T}^{(p)}$, $p = 1, \ldots, L$, share and are computed from a single common $N/L \times m$ matrix $\mathbf{T}$ as defined in (5) as well. The algorithm spends most of its learning time in the matrix computation of $\mathbf{H}$ and $\mathbf{T}$. However, in applications, these two matrices are normally not very large, and the computation is very simple and extremely fast. Thus, in general, the algorithm can provide a seconds or milliseconds learning solution to many applications, including large-scale complex systems.

IV. BENCHMARKING WITH REAL-WORLD REGRESSION PROBLEMS

Numerous real-world regression data sets from the UCI Machine Learning Repository [27] have been tested to verify the performance of the proposed algorithm. All the simulations are conducted in MATLAB 6.5 environment on an ordinary PC with Pentium 4 3.0E and 1 GB RAM. Two algorithms, i.e., the proposed RLA and BP learning algorithm, are compared on regression problems. Levenberg–Marquardt (LM) learning algorithm [18] is used on behalf of BP, as it is known as one of the fastest variants of BP. SLFNs are trained by BP to learn the data sets. The number K of hidden neurons of the SLFN is gradually increased for each data set. The results reported in this paper are averaged over 50 trials of simulations for each case. The training and testing sets are randomly regenerated before each trial of simulation. The results listed here are those with the best mean generalization performance. The same process takes place when using the RLA. Different number L of groups and number K of hidden neurons in the sub-TLFN are also gradually increased and the network architecture of RLA shown in different tables is therefore denoted by (L, K). We simple set $\epsilon = 10^{-6}$ for all cases. Cross-validation is used in the training of BP to avoid overfitting, and the training time of BP shown in this paper has excluded the time spent on cross-validation. If cross-validation time is considered, the total time spent on training BP would be much longer than the training time shown in this paper. Learning parameter μ of the LM learning algorithm [18] is divided by some factor α after each successful step (reduction in performance function) and is multiplied by α only when a tentative step would increase the performance function. As done usually (see MATLAB HELP), the initial value of μ is set to 0.001 and α is set to 10. Maximum learning epochs for BP is set to 1000. In fact, the maximum learning epochs and the target accuracy ϵ are not reached in all cases because BP is early stopped by cross-validation so that overfitting can be avoided.

A. Small Real-World Regression Problems

Some small regression data sets (with less than 1000 total instances) are tested here. The specifications of these data set are listed in Table I. As observed from Table II, RLA wins over the BP on most of the data sets in terms of testing accuracy and obtains a much shorter training time on all the data sets.

TABLE VI
PERFORMANCE COMPARISON OF RLA AND BP ON SMALL TO MEDIUM REAL-WORLD CLASSIFICATION PROBLEMS: SUCCESSFUL CLASSIFICATION RATE ON TESTING DATA AND ITS STANDARD DEVIATION (DEV), TRAINING AND TESTING TIME (SECONDS), AND NETWORK ARCHITECTURES

Datasets	Algorithms	(L, K) or K	Testing		Time (s)	
			Rate	Dev	Training	Testing
Glass	RLA	(2, 20)	65.442%	3.937%	0.0015	0.00001
	BP	10	65.154%	4.728%	0.8938	0.0031
Ionosphere	RLA	(5, 15)	86.489%	3.066%	0.0077	0.0032
	BP	5	85.727%	4.037%	0.7955	0.0107
Liver Disorders	RLA	(2, 10)	68.171%	3.278%	0.0015	0.00001
	BP	20	68.066%	3.870%	0.6282	0.0031
Page Blocks	RLA	(5, 40)	95.914%	0.284%	0.0923	0.0984
	BP	10	95.515%	0.665%	63.983	0.0124
Pima	RLA	(2, 20)	75.747%	1.779%	0.0045	0.0047
	BP	10	75.216%	2.059%	0.6891	0.0045
Vehicle	RLA	(2, 60)	79.808%	2.292%	0.0311	0.0127
	BP	25	79.502%	3.685%	66.089	0.0126
Heart	RLA	(5, 10)	78.741%	3.826%	0.0032	0.00001
	BP	5	78.657%	5.847%	0.4062	0.0094
Germen	RLA	(2, 20)	72.660%	1.455%	0.0047	0.0064
	BP	10	72.360%	2.812%	2.5219	0.0108
Image Segment	RLA	(2, 100)	94.388%	0.637%	0.1486	0.0468
	BP	60	94.030%	0.886%	1030.7	0.019
Satellite Image	RLA	(5, 140)	89.892%	0.526%	1.2157	0.1421
	BP	90	89.440%	0.824%	228.48	0.0469

The proposed RLA shows its real-time learning and prediction capability on these data sets. In some data sets like Auto Price,Pyrimidines, etc., the training time of RLA is even less than 0.001 s.

B. Medium to Large Real-World Regression Problems

The performance of RLA is also tested on some medium to large regression data sets. The specifications of these data sets are listed in Table III. As observed from Table IV, RLA achieves the shorter training time and the higher testing accuracy in most data sets as expected. RLA still can complete the training procedure within 1 s for most of these medium to large regression problems.

TABLE VII
PERFORMANCE COMPARISON BETWEEN RLA AND BP ON A LARGE CLASSIFICATION APPLICATION—HANDWRITTEN LETTER RECOGNITION: SUCCESSFUL CLASSIFICATION RATE ON TESTING DATA AND ITS STANDARD DEVIATION (DEV), TRAINING AND TESTING TIME (SECONDS), AND NETWORK ARCHITECTURES

Algorithms	(L, K) or K	Testing		Time (s)	
		Rate	Dev	Training	Testing
RLA	(5, 740)	93.239%	0.559%	88.077	4.3611
RLA	(10, 420)	86.506%	0.750%	32.863	4.9843
BP	100	85.390%	1.636%	3977.2	0.1925

V. BENCHMARKING WITH REAL-WORLD CLASSIFICATION PROBLEMS

The data sets for classification are also selected from the UCI Machine Learning Repository [27]. The experimental settings

TABLE VIII
PERFORMANCE COMPARISON BETWEEN RLA AND BP ON A VERY LARGE CLASSIFICATION APPLICATION—FOREST COVER TYPE PREDICTION: SUCCESSFUL CLASSIFICATION RATE ON TESTING DATA AND ITS STANDARD DEVIATION (DEV), TRAINING AND TESTING TIME (SECONDS), AND NETWORK ARCHITECTURES

Algorithms	(L, K) or K	Testing		Time (s)	
		Rate	Dev	Training	Testing
RLA	(10, 350)	92.476%	1.493%	139.535	65.648
RLA	(20, 200)	91.164%	1.924%	102.928	68.484
BP	20	89.918%	0.9722%	1857.4	9.244
SVM [31]	31806 SVs	89.897%	-	28813.8	12660
SVM Modular Network [31]	38023 SVs	89.74%	-	1542	1523.4
Hard Probabilistic Mixture [30]	-	91.07%	-	17460	-

of classification problems are the same as regressions. In addition, the performance of RLA and k nearest neighbor (k-NN) algorithm are discussed in Section V-E.

A. Small to Medium Classification Problems

The performance of RLA and BP is compared on ten small to medium classification data sets which have less than 10 000 instances. The specifications of these data sets are listed in Table V, and the simulation results are shown in Table VI. In terms of testing accuracy, the performances of the both algorithms are very close in almost all the data sets, though RLA is slightly better than BP. Observed from Table V, RLA learns hundreds of times faster than BP in most cases. RLA learns 6700 times than BP in the Image Segment case. It can be seen that the prediction time spent by RLA is very short as well.

B. Large Classification Application: Letter Recognition

The data set with 16 input attributes and 26 class labels was constructed by Slate [27]. The objective is to classify each of a large number of black and white rectangular pixel displays as one of the 26 capital letters of the English alphabet. The character images produced were based on 20 different fonts, and each letter within these fonts was randomly distorted to produce a file of 20 000 unique images. For each image, 16 numerical attributes were calculated using edge counts and measures of statistical moments which were scaled and discretized into a range of integer values from 1 to 15 (see [28]). Either training set or testing set has 10 000 instances. As observed from Table VII, the successful classification rate on testing data obtained by RLA is more than 93%, which is much higher than the testing accuracy obtained by BP. The best testing accuracy obtained by BP through cross-validation method is just above 85%. The best testing accuracy obtained by Frey and Slate [29] using other methods is only a little over 80%, which is much lower than the testing accuracies obtained by RLA and BP.

TABLE IX
PERFORMANCE COMPARISON BETWEEN RLA AND BP ON A CLASSIFICATION APPLICATION REQUIRING REAL-TIME LEARNING—SPAM EMAIL PREDICTION: SUCCESSFUL CLASSIFICATION RATE ON TESTING DATA AND ITS STANDARD DEVIATION (DEV), TRAINING AND TESTING TIME (SECONDS), AND NETWORK ARCHITECTURES

Algorithms	(L, K) or K	Testing		Time (s)	
		Rate	Dev	Training	Testing
RLA	(2, 140)	91.861%	0.440%	0.8358	0.0939
RLA	(5, 80)	90.657%	0.756%	0.2547	0.075
RLA	(10, 70)	89.327%	0.578%	0.2468	0.0952
BP	50	91.836%	0.745%	4641.9	0.0294

C. Very Large Classification Application: Forest Cover Type Prediction

This is an extremely large complex classification problem with seven classes. The forest cover type for a 30 × 30 m cell was obtained from the U.S. Forest Service (USFS) Region 2 Resource Information System data. It has 581 012 total instances and 54 attributes for each instance. As usually done in the literature [30]–[32], it was modified as a binary classification problem where the goal was to separate class 2 from the other six classes. One hundred thousand instances are used as training data and the rest of the 481 012 instances are used for testing. As observed from Table VIII, the successful prediction rate obtained by RLA is above 92%. Single SVM [30] and SVM modular network [30] spent 480.23 and 25.7 min on training and obtained 89.897% and 89.74% prediction rates, respectively, which are similar to BPs. Hard probabilistic mixture of SVM with 20 SVM experts and an MLP gater with 150 hidden neurons spent 291 min on training and obtained 91.07% prediction rate for this case (see [31, Table 5]). SVM [32], [30] and SVM modular network [30] need more than 30 000 support vectors (SVs), which is much larger than the hidden neurons required by RLA and BP. Thus, SVM needs much longer prediction testing time (>3.5 h) than BP and RLA.

D. Application Requiring Real-Time Learning: Spam Emails Prediction

This data set consists of spam emails and nonspam emails. Classifiers are used to predict whether an incoming email is

TABLE X
PERFORMANCE COMPARISON OF RLA AND k-NN ON REAL-WORLD CLASSIFICATION PROBLEMS

Datasets	RLA				k-NN			
	(L, K)	Rate	Dev	Time (s)	k	Rate	Dev	Time (s)
Glass	(2, 20)	65.442%	3.937%	1.0e-5	1	65.308%	4.510%	0.0016
Ionosphere	(5, 15)	86.489%	3.066%	0.0032	2	86.466%	2.076%	0.0281
Liver Disorders	(2, 10)	68.171%	3.278%	1.0e-5	11	61.714%	2.721%	0.0156
Page Blocks	(5, 40)	95.914%	0.284%	0.0984	1	95.316%	0.416%	3.1109
Pima	(2, 20)	75.747%	1.779%	0.0047	15	73.995%	0.744%	0.0594
Vehicle	(2, 60)	79.808%	2.292%	0.0127	1	68.169%	1.795%	0.0938
Heart	(5, 10)	78.741%	3.826%	1.0e-5	15	77.556%	1.667%	0.0109
Germen	(2, 20)	72.660%	1.455%	0.0064	14	72.320%	0.862%	0.1484
Image Segment	(2, 100)	94.388%	0.637%	0.0468	1	93.694%	0.409%	0.6781
Satellite Image	(5, 140)	89.892%	0.526%	0.1421	3	88.747%	0.613%	8.3781
Letter Recognition	(5, 740)	93.239%	0.559%	4.3611	1	93.119%	0.180%	87.783
Forest Type	(10, 350)	92.476%	1.493%	65.648	1	92.081%	0.154%	93893
Spam Prediction	(2, 140)	91.861%	0.440%	0.0939	1	89.963%	0.626%	8.4877

spam or not. Sometimes some new types of spam and on-spam emails may be indicated by users explictly; in this case, the classifiers should be able to online learn the new spam and nonspam emails patterns as fast as possible such that it is able to recognize and ban the spam emails from the upcoming emails. There are 4601 instances, and each instance has 57 attributes. In the simulations, 3000 randomly selected instances compose the training set and all the rest are used for testing. As shown in Table IX, the RLA achieves good testing accuracy at very fast learning speed ($<$1 s); however, BP needs to spend 4641.9 s on learning, which is not realistic in such a practical real-time application.

E. Comparison Between RLA and k-NN Algorithm

RLA is also compared with k-nearest neighbor approach (k-NN) on these real-world classification applications. Like the other two algorithms, the parameter k of k-NN is also gradually increased to search for the best generalization performance. Table X shows the comparison between RLA and k-NN on all these classification problems. In this experiment, the testing accuracy achieved by k-NN is worse than RLA. Its testing/prediction time is much longer than the other two algorithms especially on the data sets having large numbers of testing data. For example, the forest-type prediction problem which has 100 000 training data and 481 012 testing data, the testing time of k-NN can be as long as 26 h, which is more than one day. Single SVM spent more than 3.5 h on predicting such a large number of unknown testing data [30].

VI. CONCLUSION

Based on our previous theoretical work [1], an extremely fast learning algorithm has been proposed for Huang's network in this paper. Three improvements have been made in the proposed new algorithm. 1) New selection criteria for the biases of neural quantizer has been proposed to improve the generalization performance. 2) An automatic method to select the appropriate values of the quantizer factors T and U has been introduced. In fact, the lower bound of T and lower and upper bounds of U have been derived and given in this paper (see Remarks 1 and 2 for details). 3) Hidden output matrix $\mathbf{H}$ need not be invertible and the MP generalized inverse of $\mathbf{H}$ is introduced, so that the number of neurons in the first hidden layer of sub-TLFN K can be set to some appropriate number smaller than N/L and thus the network architecture can be further shrinked. The proposed RLA algorithm has been successfully tested on many benchmarking data sets of different types. Huang's network is trained in a simple efficient one-time way instead of gradient-based and/or iterative methods: 1) all the weights of the connections linking the input layer to the first hidden layer of the sub-TLFN are randomly generated; 2) all the biases of the first hidden neurons of the sub-TLFN are also randomly generated; 3) all the rest parameters (weights and biases) of Huang's network can be simply analytically computed based on a small size of first hidden output matrix of the sub-TLFN. Huang's network [1] is a sparse TLFN and needs small memory for computation. Thus, RLA can implement many large-scale systems in an ordinary PC.

Since the proposed RLA algorithm for Huang's network is purely based on analytical method, it outperforms the other conventional gradient-based and/or iterative approaches without requiring stopping criterion and avoiding hidden local minima issues. Local minima problem existing in conventional learning algorithms make networks almost incapable of learning in some applications.

The performance evaluation of the proposed RLA has been systematically investigated in this paper. Seen from the simulation results on many real-world regression and classification problems, the RLA can not only reach good generalization performance but also provide very short learning and prediction time in many cases, which may be within most users' expected time in many applications. In other words, the proposed RLA algorithm or Huang's network would provide real-time alternative solutions to time-critical applications. Thus, the proposed RLA algorithm may be a good candidate to the applications 1) which have large training samples and require high learning accuracy/generalization performance and/or high learning speed and 2) where the application patterns and environments change fast and long time learning is not allowed. Even for those applications where long training time is allowed, the proposed RLA can be used and tried as the first choice since almost zero-time "effort" needs to be made for this trial. If the performance is accepted, tedious time-consuming trials of other algorithms can be prevented.

REFERENCES

[1] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 274–281, Mar. 2003.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–366, 1989.

[3] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[4] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Netw.*, vol. 2, pp. 183–192, 1989.

[5] A. Gallant and H. White, "There exists a neural network that does not make avoidable mistakes," in *Artificial Neural Networks: Approximation and Learning Theory*, H. White, Ed. Oxford, U.K.: Blackwell, 1992, pp. 5–11.

[6] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *Artificial Neural Networks: Approximation and Learning Theory*, H. White, Ed. Oxford, U.K.: Blackwell, 1992, pp. 29–40.

[7] Y. Ito, "Approximation of continuous functions on $\mathbf{R}^d$ by linear combinations of shifted rotations of a sigmoid function with and without scaling," *Neural Netw.*, vol. 5, pp. 105–115, 1992.

[8] T. Chen, H. Chen, and R.-W. Liu, "Approximation capability in $C(\overline{\mathbf{R}}^n)$ by multilayer feedforward networks and related problems," *IEEE Trans. Neural Netw.*, vol. 6, no. 1, pp. 25–30, Jan. 1995.

[9] G.-B. Huang, Y.-Q. Chen, and H. A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 799–801, May 2000.

[10] D. Erdogmus, O. Fontenla-Romero, J. C. Principe, A. Alonso-Betanzos, and E. Castillo, "Linear-least-squares initialization of multilayer perceptrons through backpropagation of the desired response," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 325–337, Mar. 2005.

[11] D. E. Rumellhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distrib. Process.*, vol. 1, pp. 318–362, 1986.

[12] R. P. Brent, "Fast training algorithms for multilayer neural nets," *IEEE Trans. Neural Netw.*, vol. 2, no. 3, pp. 346–354, May 1991.

[13] R. S. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks," *IEEE Trans. Signal Process.*, vol. 40, no. 1, pp. 202–210, Jan. 1992.

[14] A. Sperduti and A. Starita, "Speed up learning and network optimization with extended back propagation," *Neural Netw.*, vol. 6, pp. 365–383, 1993.

[15] A. Parlos, B. Fernandez, A. Atiya, J. Muthusami, and W. Tsai, "An accelerating learning algorithm for multilayer perceptron networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 3, pp. 493–497, May 1994.

[16] X. H. Yu, G. A. Chen, and S. X. Cheng, "Dynamic learning rate optimization of the backpropagation algorithm," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 669–677, May 1995.

[17] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*. Boston, MA: PWS, 1996.

[18] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[19] S. Abid, F. Fnaiech, and M. Najim, "A fast feedforward training algorithm using a modified form of the standard backpropagation algorithm," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 424–430, Mar. 2001.

[20] 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, no. 1, pp. 224–229, Jan. 1998.

[21] S. Ferrari and R. F. Stengel, "Smooth function approximation using neural networks," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 24–38, Jan. 2005.

[22] C. Xiang, S. Q. Ding, and T. H. Lee, "Geometrical interpretation and architecture selection of mlp," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 84–96, 2005.

[23] G.-B. Huang, Q.-Y. Zhu, K. Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 3, pp. 187–191, Mar. 2006.

[24] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, Jul. 2006.

[25] 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 (IJCNN2004)*, Budapest, Hungary, Jul. 25–29, 2004, vol. 2, pp. 985–990.

[26] ——, "Extreme learning machine: Theory and applications," *Neurocomput.*, to be published.

[27] C. Blake and C. Merz, UCI repository of machine learning databases [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html Dept. Inf. Comp. Sci., Univ. California. Irvine, CA, 1998

[28] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood, 1994.

[29] P. W. Frey and D. J. Slate, "Letter recognition using Holland-style adaptive classifiers," *Mach. Learn.*, vol. 6, no. 2, pp. 161–182, 1991.

[30] G.-B. Huang, K. Z. Mao, C.-K. Siew, and D.-S. Huang, "Fast modular network implementation for support vector machines," *IEEE Trans. Neural Netw.*, vol. 16, no. 6, pp. 1651–1663, Nov. 2005.

[31] R. Collobert, Y. Bengio, and S. Bengio, "Scaling large learning problems with hard parallel mixtures," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 17, no. 3, pp. 349–365, 2003.

[32] ——, "A parallel mixtures of SVMs for very large scale problems," *Neural Comput.*, vol. 14, pp. 1105–1114, 2002.

Guang-Bin Huang (M'98–SM'04) received the B.Sc. degree in applied mathematics and the M.Eng. degree in computer engineering from Northeastern University, China, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore, in 1999.

From June 1998 to May 2001, he was a Research Fellow with the Singapore Institute of Manufacturing Technology (formerly known as Gintic Institute of Manufacturing Technology), where he led/implemented several key industrial projects. Since then, he has been an Assistant Professor in the School of Electrical and Electronic Engineering, Nanyang Technological University. His current research interests include machine learning, bioinformatics, and networking.

Dr. Huang is an Associate Editor of *Neurocomputing* and IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, PART B: CYBERNETICS.

Qin-Yu Zhu received the B.Eng. degree from Shanghai Jiao Tong University, China, in 2001. He is currently working toward the Ph.D. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

His research interests include neural networks and evolutionary algorithms. He has published a number of papers in international journals and conferences.

Chee-Kheong Siew (M'92) received the B.Eng. degree in electrical engineering from the University of Singapore, in 1979, and the M.Sc. degree in communication engineering from Imperial College, London, U.K., in 1987.

He is currently an Associate Professor in the School of Electrical and Electronics Engineering, Nanyang Technological University (NTU), Singapore. From 1995 to 2005, he was Head of the Information Communication Institute of Singapore (ICIS) after he managed the transfer of ICIS to NTU and rebuilt the institute in the university environment. After six years in industry, he joined NTU in 1986 and became the Head of the institute in 1996. His current research interests include neural networks, packet scheduling, traffic shaping, admission control, service curves and admission control, QoS framework, congestion control, and multipath routing.