Neural Process Lett (2008) 28:81–95
DOI 10.1007/s11063-008-9083-z

Universal Approximation and QoS Violation Application of Extreme Learning Machine

Lei Chen · LiFeng Zhou · Hung Keng Pung

Published online: 29 August 2008

Abstract Neural networks have been successfully applied to many applications due to their approximation capability. However, complicated network structures and algorithms will lead to computational and time-consuming burdens. In order to satisfy demanding real-time requirements, many fast learning algorithms were explored in the past. Recently, a fast algorithm, Extreme Learning Machine (ELM) (Huang et al. 70:489–501, 2006) was proposed. Unlike conventional algorithms whose neurons need to be tuned, the input-to-hidden neurons of ELM are randomly generated. Though a large number of experimental results have shown that input-to-hidden neurons need not be tuned, there lacks a rigorous proof whether ELM possesses the universal approximation capability. In this paper, based on the universal approximation property of an orthonormal method, we firstly illustrate the equivalent relationship between ELM and the orthonormal method, and further prove that neural networks with ELM are also universal approximations. We also successfully apply ELM to the identification of QoS violation in the multimedia transmission.

Keywords Feedforward neural network · Universal approximation · Radial basis function (RBF) · Extreme learning machine (ELM) · Randomhidden neurons · QoS

1 Introduction

In past decades feedforward neural networks (FNNs) have been investigated extensively from both theoretical and applied aspects. Because of their approximation capability [2–4], neural networks have been successfully applied to many real applications. One of neural network advantages is that they can learn the interrelations of all factors from the observed data automatically. However, such a high artificial intelligence (AI) also brings the extremely computational burden, which may not satisfy the real-time requirement. Meanwhile the

L. Chen (✉) · L. Zhou · H.K. Pung
Network Systems and Service Lab, Department of Computer Science, National University of Singapore, Singapore, Singapore
e-mail: dcschen@nus.edu.sg; chen_lei@pmail.ntu.edu.sg

real-time support is a common requirement in many practical applications. Designing a FNN to achieve a better generalization performance with fast training speed is always a challenging topic in neural networks.

Recently, an effective and non-iterative technique, ELM, has been proposed in [1]. The basic principle of ELM is to randomly generate hidden neurons (the input-to-hidden parameters are generated based on some continuous distribution probabilities) and analytically determine the hidden-to-output weights of neural networks one time. Due to the random character of hidden neurons, ELM has been proven to be an effective method to make neural networks achieve a better generalization performance at a fast learning speed. Though many experimental results show that the input-to-hidden neurons do not need to be adjusted at all, so far there lacks a rigorous theoretical justification whether neural networks with ELM process the universal approximation capability.

In order to solve the question, we firstly refer to the universal approximation of a Gram-Schmidt orthonormal neural network. The orthonormal network was first proposed in [5], where hidden kernel neurons are transformed into an orthonormal set of neurons by using Gram-Schmidt orthonormalization. After this transformation, FNNs do not recomputate the already existing weights of hidden neurons, which can remarkably reduce the computing time. Based on the universal approximation of the orthonormal network, we firstly illustrate the equivalent relationship between ELM and the orthonormal network, thus we can naturally obtain the conclusion: neural networks with ELM are also universal approximation. In order to verify the equivalent relationship, several benchmarking regression simulations support our conclusion. Moreover, in this paper we successfully apply ELM to the identification of QoS violations in multimedia network transmission to replace conventional rule-base methods. A violation possibly occurs at runtime due to fluctuating resource availabilities, which could adversely affect the performance of multimedia applications. A simple way to detect a violation is to employ rules with thresholds set for a few key parameters. However, such a common method is not effective identifying a violation if multiple environment variables are involved. In view of this, we classify and identify a QoS violation through analyzing a vector of flow statistics and application QoS metrics by using ELM and the orthonormal neural networks. The experimental results of the two algorithms enjoy the advantage of both real-time processing and high classification accuracy in the same precision, which also supports our conclusions with respect to the approximation analysis.

This paper is organized as follows. Section 2 provides some preliminaries, such as inner products and symbols of a standard SLFN, which are required for the analysis in the following sections. In order to prove the approximation of ELM, we also need to show the universal approximation of Gram-Schmidt orthonormal neural networks in this section. Section 3 details our main result: the universal approximation of ELM, and their performance evaluation is presented in Sect. 4. Discussions and conclusions are given in Sect. 5.

2 Preliminaries

Before we discuss our main results, we need to introduce some terminologies and background information.

2.1 Extreme Learning Machine (ELM)

The output of an standard single hidden layer feedforward network (SLFN) with L hidden neurons can be represented by $f_L = \sum_{i=1}^{L} \beta_i g(\mathbf{a}_i, b_i, \mathbf{x})$, where $\mathbf{a}_i$ and b_i are the learning

parameters of hidden neurons and β_i is the weight connecting the i-th hidden neuron to the output neurons; $g(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the i-th hidden neuron with respect to the input $\mathbf{x}$. Seen from the viewpoint of network architecture, two main SLFN network architectures have been investigated, additive neurons and kernel neurons. For the additive neurons, the activation function $g(x)$: $R \to R$ takes the form $g(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i)$, where $\mathbf{a}_i \in R^n$ is the weight vector connecting the input layer to the i-th hidden neuron, and $b_i \in R$ is the bias of the i-th hidden neuron; $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in R^n. For the kernel neurons, the activation function $g(x)$: $R \to R$ takes the form $g(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|)$, where $\mathbf{a}_i \in R^n$ is the center of the i-th RBF neuron and $b_i \in R^+$ is the impact of the i-th RBF neuron. R^+ indicates the set of all positive real value.

For a series of N arbitrary distinct training samples $(\mathbf{x}_i, \mathbf{t}_i)$, $i = 1, \ldots, N$, where $\mathbf{x}_i = [x_{i1}, \ldots, x_{in}]^T \in \mathbf{R}^n$ is an input vector and $\mathbf{t}_i = [t_{i1}, \ldots, t_{im}]^T \in \mathbf{R}^m$ is a target vector. A standard SLFN with L hidden neurons with activation function $g(x)$ can be expressed as

$$\sum_{i=1}^{L} \beta_i g(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{o}_j, \quad j = 1, \ldots, N,$$

where $\mathbf{o}_j$ is the actual output of SLFN.

A standard SLFN with L hidden neurons can learn N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, $i = 1, \ldots, N$, with **zeroerror** means that there exist parameters $\mathbf{a}_i$ and b_i, for $i = 1, \ldots, L$, such that

$$\sum_{i=1}^{N} \|\mathbf{o}_i - \mathbf{t}_i\| = 0.$$

Thus our ideal objective is to find proper parameters $\mathbf{a}_i$ and b_i such that

$$\sum_{i=1}^{L} \beta_i g(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \quad j = 1, \ldots, N,$$

The above N equations can be expressed as

$$\mathbf{H}\beta = \mathbf{T} \tag{1}$$

where $\beta = [\beta_1, \ldots, \beta_L]^T$, $\mathbf{T} = [t_1, \ldots, t_N]^T$ and

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & g(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ g(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & g(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L} \tag{2}$$

where the matrix $\mathbf{H}$ is called as the hidden layer matrix of the SLFN.

Lemma 2.1 *[1] Given a standard SLFN with N hidden neurons and the activation function* g: $\mathbf{R} \to \mathbf{R}$, *which is infinitely differentiable in any interval, for N arbitrary distinct samples* $(\mathbf{x}_i, t_i)$, *where* $\mathbf{x}_i \in \mathbf{R}^d$ *and* $t_i \in \mathbf{R}^m$, *for* $\mathbf{a}_i$ *and* b_i *randomly chosen from any intervals of* $\mathbf{R}^d$ *and R, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix* $\mathbf{H}$ *of the SLFN is invertible and* $\|\mathbf{H}\beta - \mathbf{T}\| = 0$.

Lemma 2.1 illustrates that when the number of neurons L is equal to the number of samples N, neural networks can precisely express observed samples. However, the number of hidden neurons is normally much less than the number of distinct training samples, i.e., $L \ll N$. It means that $\mathbf{H}$ is a nonsquare matrix. Huang further pointed out that neural networks

with randomly generating $\{\mathbf{a}_i, b_i\}$ and determining β_i by generalized inverse $\beta = \mathbf{H}^{\dagger}\mathbf{T}$ can approach training target with small errors, where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of the matrix $\mathbf{H}$. The fast learning algorithm is also called as Extreme Learning Machine (ELM) [6].

2.2 Universal Approximation of Orthonormal Networks

Due to the random character of $\{\mathbf{a}_i, b_i\}_{i=1}^{L}$, ELM has been proven to be an effective method to make neural networks achieve a better generalization performance at an extremely fast learning speed. Up till now, there is no proof that neural networks by ELM with randomly generated hidden neurons can have universal approximation capability. In order to prove the universal approximation of ELM, we firstly need to prove the approximation capability of an orthonormal network.

Assuming the training samples generated with uniform probability distribution and all the functions belong to the space L^2. Similar to [5, p. 1179], the inner product of two functions is defined as

$$\langle u(\mathbf{x}), v(\mathbf{x})\rangle = \sum_{i=1}^{N} u(\mathbf{x}_i)v(\mathbf{x}_i) \tag{3}$$

where N is the number of training samples. Here we should note that the above inner product expression is based on the statistics, which can be approximation by the interpolation. In fact, when the number of training data is large enough, the inner product can be easily deduced by using the limitation theory. Thus without loss of generality, we denote the inner product by using the interpolation in this paper.

We say that the nonzero vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are orthogonal if $\langle \mathbf{e}_i, \mathbf{e}_j\rangle = 0$, for $i \neq j$ and orthonormal if $\langle \mathbf{e}_i, \mathbf{e}_i\rangle = 1$. In fact, for any linearly independent sequence $\{g_1(\mathbf{x}), \ldots, g_L(\mathbf{x})\}$ in Hilbert space, we can construct an orthonormal basis $\{e_1, e_2, \ldots, e_L\}$ by some orthonormal transformations, such as the Gram-Schmidt process [7, pp. 167–168].

Next we will introduce how to use Gram-Schmidt orthonormalization to construct networks. According to ELM, we should find proper parameters such that

$$\beta_1 g_1(\mathbf{x}_i) + \cdots + \beta_L g_L(\mathbf{x}_i) = f(\mathbf{x}_i), \quad i = 1, \ldots, N \tag{4}$$

where $f(\mathbf{x}_i) = t_i$ and $g_j(\mathbf{x}_i) = g(\mathbf{a}_j, b_j, \mathbf{x}_i)$ for $j = 1, \ldots, L$. The function $f(\mathbf{x})$ can be regarded as the target function to be approximated by neural networks.

Multiplying Eq. 4 by $g_j(\mathbf{x}_i)$, we have

$$\beta_1 g_1(\mathbf{x}_i)g_j(\mathbf{x}_i) + \cdots + \beta_L g_L(\mathbf{x}_i)g_j(\mathbf{x}_i) = f(\mathbf{x}_i)g_j(\mathbf{x}_i), \quad i = 1, \ldots, N, \quad j = 1, \ldots, L \tag{5}$$

Calculating the sum of $\{\mathbf{x}_i\}_{i=1}^{N}$ in the above L equations, we have

$$\beta_1 \sum_{i=1}^{N} g_1(\mathbf{x}_i)g_j(\mathbf{x}_i) + \cdots + \beta_L \sum_{i=1}^{N} g_L(\mathbf{x}_i)g_j(\mathbf{x}_i) = \sum_{i=1}^{N} f(\mathbf{x}_i)g_j(\mathbf{x}_i), \quad j = 1, \ldots, L \tag{6}$$

Following Eqs. 3 and 6 can be rewritten as

$$\beta_1\langle g_1(\mathbf{x}), g_j(\mathbf{x})\rangle + \cdots + \beta_L\langle g_L(\mathbf{x}), g_j(\mathbf{x})\rangle = \langle f(\mathbf{x}), g_j(\mathbf{x})\rangle, \quad j = 1, \ldots, L \tag{7}$$

The above L equations can be rewritten as

$$\widetilde{\mathbf{H}}\beta = \widetilde{\mathbf{T}}$$

where

$$\widetilde{\mathbf{T}} = \begin{pmatrix} \langle f(\mathbf{x}), g_1(\mathbf{x})\rangle \\ \vdots \\ \langle f(\mathbf{x}), g_L(\mathbf{x})\rangle \end{pmatrix}$$

and

$$\widetilde{\mathbf{H}} = \begin{bmatrix} \langle g_1(\mathbf{x}), g_1(\mathbf{x})\rangle & \cdots & \langle g_L(\mathbf{x}), g_1(\mathbf{x})\rangle \\ \vdots & \ddots & \vdots \\ \langle g_1(\mathbf{x}), g_L(\mathbf{x})\rangle & \cdots & \langle g_L(\mathbf{x}), g_L(\mathbf{x})\rangle \end{bmatrix}_{L\times L}$$

where $\widetilde{\mathbf{H}}$ is named as the inner product hidden layer matrix. If $\{g_k(\mathbf{x})\}_{k=1}^{L}$ are orthonormal each other, the solutions of above equations can be calculated as $\beta_k = \langle f(\mathbf{x}), g_k(\mathbf{x})\rangle = \sum_{i=1}^{N} t_i g_k(\mathbf{x}_i)$. However, as Kaminski and Strumillo [5] has pointed out, $\{g_k(\mathbf{x})\}_{k=1}^{L}$ are not orthonormal each other normally. Hence similar to [5], we apply the standard Gram-Schmidt orthonormalization to transform $\{g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_L(\mathbf{x})\}$ into an orthonormal set of basis functions $\{u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_L(\mathbf{x})\}$, i.e.,

$$[u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_L(\mathbf{x})] = [g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_L(\mathbf{x})] \cdot \mathbf{V} \tag{8}$$

where $\mathbf{V}$ is an upper triangular matrix whose detailed expression can be found in [5, p. 1182].

We denote $\|f\|^2 = \langle f, f\rangle$. For any f and $\{e_k\}_{k=1}^{\infty}$ in Hilbert space $\mathcal{H}$, the identity

$$\|f\|^2 = \sum_{j=1}^{\infty} |a_k|^2, \text{ where } a_k = \langle f, e_k\rangle,$$

which is called *Parseval identity*, holds if and only if $\{e_k\}_{k=1}^{\infty}$ is also an orthonormal basis [7, p. 166]. Based on Parseval identity, we can easily obtain the following approximation property of the orthonormal basis:

Theorem 2.1 *For any $f \in \mathcal{H}$, suppose $e_1, e_2, \ldots$ is an orthonormal basis in $\mathcal{H}$, then*

$$\|f - \sum_{j=1}^{L} \langle f, e_j\rangle e_j\| \to 0, as L \to \infty.$$

Remark 1 In the classic textbooks of functional analysis, we can find similar theorem in one-dimension space. In fact, the similar property can be extended into multi-dimension space by using Parseval identity.

Theorem 2.2 *For any bounded, integrable function $g(\mathbf{a}_i, b_i, \mathbf{x}) \in L^2$, if it is an infinitely differentiable additive function or kernel function, then neural networks by Gram-Schmidt transformation are universal approximation.*

Proof Lemma 2.1 illustrates that if activation function $g(x)$ is infinitely differentiable, then for almost all the parameters, the column vectors of $\mathbf{H}$ are linearly independent of each other. In fact, the orthogonal or orthonormal relationship is a kind of the linear independence as well. The linear independence of the column vectors in $\mathbf{H}$ leads to that the inner product of the column vectors in $\mathbf{H}$ is not zero each other. Thus based on inner product definition (3) and the orthogonal definition, we can conclude that $g_i(x)_{i=1}^{L}$ are linearly independent when the column vectors of $\mathbf{H}$ are linearly independent each other.

For any kernel function, the statements of the paper [5, p. 1179] illustrated that if centers are arbitrary, the system (7) has an unique solution. Meantime, according to the statement of Subsect. 2.2, we know that the linear system (4) is equivalent to the system (7), i.e.,

$$\mathbf{H}^{\dagger}\mathbf{T} = \widetilde{\mathbf{H}}^{-1}\widetilde{\mathbf{T}} \tag{9}$$

In another word, the system (4) has also a unique solution. Therefore the corresponding column vectors of matrix $\mathbf{H}$ are also linearly independent, which is consistent with Lemma 2.1, but the only difference is that we extend infinitely differentiable kernel functions to any kind of kernel functions.

Summarizing the above statement, we have the following conclusion: for infinitely differentiable additive function or any kernel function $g(x)$, when $\mathbf{a}_i$ and b_i randomly chosen from any intervals of $\mathbf{R}^d$ and R, $\{g_k(\mathbf{x})\}_{k=1}^{L}$ are linearly independent each other, where the linear independence of $\{g_k(\mathbf{x})\}_{k=1}^{L}$ is judged by inner product expression (3) under the condition of large enough training data.

Based on Eq. 8, we apply the Gram-Schmidt to transform $\{g_1(\mathbf{x}), \ldots, g_L(\mathbf{x})\}$ into an orthonormal set of basis functions $\{u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_L(\mathbf{x})\}$, i.e.,

$$[u_1(\mathbf{x}), u_2(\mathbf{x}), \ldots, u_L(\mathbf{x})] = [g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_L(\mathbf{x})] \cdot \mathbf{V} \tag{10}$$

According to the orthonormal property of $\{u_i\}_{i=1}^{L}$ and Theorem 2.1, such weights $\{\alpha\}_{i=1}^{L}$ can ensure the approximation capability of neural networks, i.e.,

$$\| f - \sum_{i=1}^{L} \alpha_i u_i(\mathbf{x}) \| < \epsilon, \text{ as} L \to \infty. \tag{11}$$

where $\alpha_i = \langle f(\mathbf{x}), u_i(\mathbf{x}) \rangle$. □

Remark 2 Here, Theorem 2.2 only mentions an orthonormal transformation. In fact, for other orthonormal transformations, such as QR decomposition or singular value decomposition (SVD), the universal approximation result is also correct. Theorem 2.2 explains why the orthonormal neural networks proposed in [5] can achieve a good generalization performance.

3 Universal Approximation of ELM

In this section, capitalizing on the approximation property of orthonormal neural networks, we will prove that neural networks with ELM are universal approximation.

Theorem 3.1 *For any bounded, integrable function $g(\mathbf{a}_i, b_i, \mathbf{x}) \in L^2$, neural networks constructed by ELM and with infinitely differentiable additive neurons or with any kernel neurons in L^2 are universal approximation.*

Proof Based on Theorem 2.2, when the number of training data is large enough, we have: for infinitely differentiable additive function or any kernel function $g(x)$, when $\mathbf{a}_i$ and b_i randomly chosen from any intervals of $\mathbf{R}^d$ and R, $\{g_k(\mathbf{x})\}_{k=1}^{L}$ are linearly independent each other. Then based on Eq. 11 in Theorem 2.2, after Gram-Schmidt transformation, the new hidde-to-output weights $\{\alpha_i\}_{i=1}^{L}$ expressed as $\alpha_i = \langle f(\mathbf{x}), u_i(\mathbf{x}) \rangle$ can ensure universal approximation of neural networks.

We set $\beta = \mathbf{H}^{\dagger}\mathbf{T}$ and

$$\mathbf{U} = \begin{bmatrix} u_1(\mathbf{x}_1) & \cdots & u_L(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ u_1(\mathbf{x}_N) & \cdots & u_L(\mathbf{x}_N) \end{bmatrix}_{N \times L}$$

According to Eq. 8, we have $\mathbf{U} = \mathbf{HV}$.
Based on (9), we have

$$\alpha = \widetilde{\mathbf{U}}^{-1}\widetilde{\mathbf{T}} = \mathbf{U}^{\dagger}\mathbf{T} = \mathbf{U}^{\dagger}\mathbf{H}\beta \tag{12}$$

where refereing to the definition of $\widetilde{\mathbf{H}}$, $\widetilde{\mathbf{U}}$ is the inner product matrix of $\mathbf{U}$.
Based on Eqs. 10–12, we have

$$\begin{aligned} \|f(\mathbf{x}) - \sum_{i=1}^{L} \beta_i g_i(\mathbf{x})\| &= \|f(\mathbf{x}) - [g_1(\mathbf{x}), \ldots, g_L(\mathbf{x})] \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_L \end{bmatrix} \| \\ &= \|f - [u_1(\mathbf{x}), u_2(\mathbf{x}), \cdots, u_L(\mathbf{x})]\, \mathbf{V}^{\dagger}\beta\| \\ &= \|f - [u_1(\mathbf{x}), u_2(\mathbf{x}), \cdots, u_L(\mathbf{x})]\, \mathbf{V}^{\dagger}\mathbf{H}^{\dagger}\mathbf{U}\alpha\| \\ &= \|f - [u_1(\mathbf{x}), u_2(\mathbf{x}), \cdots, u_L(\mathbf{x})]\, \alpha\| \\ &= \|f - \sum_{i=1}^{L} \alpha_i u_i(\mathbf{x})\| \to 0, L \to \infty \end{aligned} \tag{13}$$

The theorem is proved. □

Remark 3 Such orthonormal neural networks have been proved to be able to achieve good generalization performance at a fast speed [5]. However, the Eq. 13 in Theorem 3.1 states an equivalent relationship between Gram-Schmidt transformation and ELM: the hidden-to-out weights directly determined by the hidden layer matrix $\mathbf{H}$ is the same solution as the hidden-to-out weights calculated by the orthonormal basis $\{u_i(\mathbf{x})\}_{i=1}^{L}$. Hence we can naturally obtain the following conclusion that such orthonormal transformation is not necessary for neural networks. The following simulation result comparison between ELM and the orthonormal neural networks based on benchmark regression and QoS classification applications will support our conclusion.

Remark 4 Original ELM is only suitable for infinitely differentiable functions. Theorem 3.1 shows that ELM can be applied to any kernel function, which simultaneously enlarge the scope of activation functions.

Remark 5 Unlike many classic approximation results, where hidden neurons must be calculated by a complicated process, Theorem 3.1 shows that neural networks with random hidden neurons are also universal approximation. The universal approximation proof of neural networks trained by another random neuron algorithm can be found in the paper Huang et al. [8], but the algorithm is a growing algorithm, which calculates hidden-to-output weights one by one in an incremental way. However, ELM is a kind of batch learning algorithms, which calculates all the hidden-to-output weights one time. Theorem 3.1 explains why ELM can provide a good generalization performance.

4 Performance Evaluation

In [1], authors have demonstrated that ELM can outperform many popular algorithms like BP, SVM and other well-known algorithms in many cases, moreover the main purpose of this paper is to illustrate the universal approximation of ELM and the equivalent relationship between ELM and the orthonormal algorithms, thus we may only need to compare ELM with the orthonormal methods in this paper. In this section, we will show that ELM and Gram-Schmidt algorithms present the same performance in our experiments (Benchmarking regression problems and QoS violation classification). The two learning procedures can be summarized in the following steps:

Algorithm Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots, N\}$ and L hidden neurons

1) ELM

   a) Assign random parameters $\mathbf{a}_i$ and b_i, for $i = 1, \ldots, L$.
   b) Calculate hidden layer matrix $\mathbf{H}$.
   c) Calculate the hidden-to-output weights by $\mathbf{H}^{\dagger}\mathbf{T}$.

2) Orthonormal Neural Networks

   a) Assign random parameters $\mathbf{a}_i$ and b_i, for $i = 1, \ldots, L$.
   b) Calculate hidden layer matrix $\mathbf{H}$.
   c) After any orthonormal transformation, obtain orthonormal matrix $\mathbf{U}$.
   d) Calculate the hidden-to-output weights by $\alpha_i = \langle f(\mathbf{x}), u_i(\mathbf{x}) \rangle$, for $i = 1, \ldots, L$.

All the benchmark regression problems are from UCI datasets [9]. For simplicity, all the input data are normalized into the range $[-1, 1]$ in our experiments. Neural networks with ELM and with Gram-Schmidt are both assigned the same of number of hidden neurons, i.e. 30 neurons. All the simulations run in MATLAB 6.5 environment and the same PC with Pentium 4 3.0 GHZ CPU and 1G RAM. The activation function used in our proposed algorithms is a simple Gaussian kernel function $g(\mathbf{x}) = \exp(-\gamma \|\mathbf{x} - \mu\|^2)$, the centers μ_i are randomly chosen from the range $[-1, 1]$ whereas the impact factor γ is chosen from the range $(0, 0.5)$.

4.1 Benchmarking with Regression Problem

Based on nine real world benchmark regression datasets, the performance comparisons between ELM and Gram-Schmidt are given out. Table 1 shows the characteristics of these regression datasets and the corresponding number of hidden neurons.

For each problem, 50 trials are done. As shown in Table 2, the two neural networks both achieve good generalization performances with almost the same error level, which also verifies their equivalent relationship. Here, we should note that for the RMSE of some cases, there may be small differences between the two algorithms. In fact, deficient training samples or incomplete distribution may lead to the imprecise expression of inner product in the orthonormal neural networks. But for ELM, since the calculation of inner product is not necessary, its computational errors are avoided, which is also an advantage of ELM. Since the training data and testing data are randomly regenerated in every simulation trial, every trial leads to different results for the same case. The corresponding deviation of the results reflects the steady state of the corresponding algorithms. Table 2 shows the deviation of the training error and testing error at the 30th neuron. Since the corresponding deviation results are very small, it shows that our simulation results are very stable and repeatable.

Table 1 Specification of nine Benchmark Regression Datasets

Name	No. of observations		Attributes	No. of neurons
	Training data	Testing data		
Abalone	2,000	2,177	8	30
Ailerons	7,154	6,596	39	30
Airplane	450	500	9	30
Bank	4,500	3,692	8	30
Boston	250	256	13	30
California	8,000	12,640	8	30
Census	10,000	12,784	8	30
Delta Ailerons	3,000	4,129	5	30
Delta Elevators	4,000	5,517	6	30

Table 2 Comparison of Average Root Mean Square Error (Mean) and Devations (Dev)of Training Error and Testing Error

Name	ELM				Gram-Schmidt			
	Training		Testing		Training		Testing	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
Abalone	0.0756	0.0008	0.0784	0.0027	0.0754	0.0010	0.0772	0.0022
Ailerons	0.0617	0.0053	0.0625	0.0053	0.0639	0.0053	0.0643	0.0047
Airplane	0.0444	0.0037	0.0481	0.0032	0.0447	0.0032	0.0491	0.0045
Bank	0.0593	0.0049	0.0603	0.0053	0.0627	0.0086	0.0629	0.0085
Boston	0.0923	0.0087	0.1095	0.0123	0.0909	0.0088	0.1117	0.0100
California	0.1367	0.0028	0.1377	0.0024	0.1368	0.0026	0.1383	0.0025
Census	0.0749	0.0019	0.0758	0.0015	0.0750	0.0021	0.0760	0.0022
Delta Ailerons	0.0456	0.0047	0.0469	0.0052	0.0444	0.0043	0.0450	0.0046
Delta Elevators	0.0596	0.0061	0.0604	0.0063	0.0600	0.0059	0.0606	0.0073

In order to verify our approximation theory, we also draw the training error and testing error curves from the 5th neuron to the 80th neuron for Airplane case in Fig. 1a. As shown in Fig. 1a, the neural networks achieve better generalization performance with the growth of neurons, which also supports our universal approximation conclusion.

The mean training time of ELM and Gram-Schmidt neural networks with the 30th neuron is illustrated in Table 3 as well. From Table 3, we know that neural networks without orthonormal transformation take less training time than neural networks with orthonormal transformation. As shown in Fig. 1b, with the growth of neuron number, the difference between ELM and Gram-Schmid neural networks becomes greater, which shows the efficiency of ELM.

4.2 QoS Violation

4.2.1 QoS Violation in Multimedia Transmission

An end-to-end multimedia transmission is subject to various runtime impairments such as process failure, network congestion or link error. In view of this, a bundle of adaptation-based QoS systems (hereinafter referred as adaptive QoS systems) have been designed,

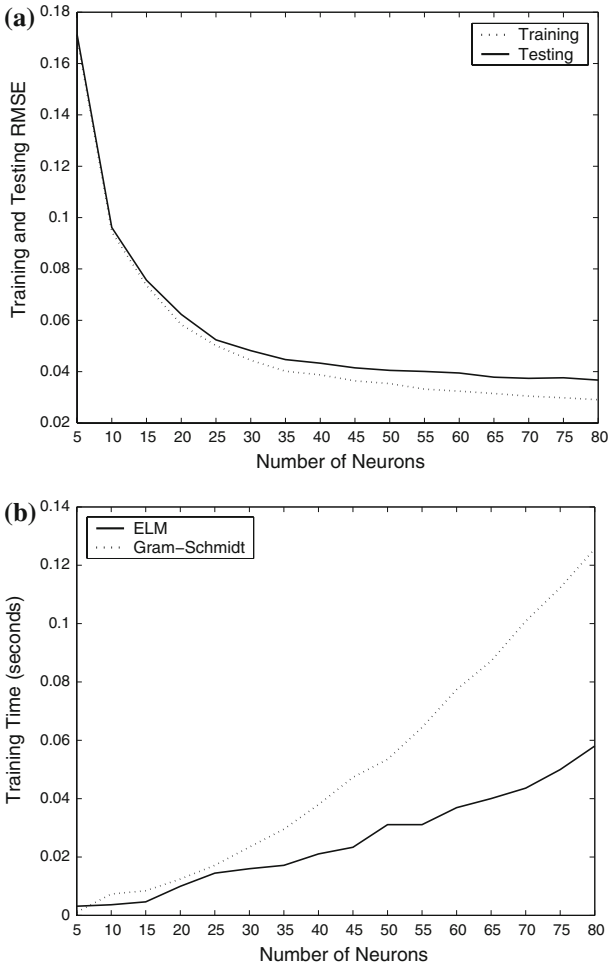

Fig. 1 Performance of the Airplane regression problem: (**a**) training and testing error curves with Gaussian kernel neurons, (**b**) comparison of training time curves with Gaussian kernel neurons between ELM and Gram-Schmidt networks

each of which tries to maintain runtime transmission quality through re-composition or re-configuration of the end-to-end system. However, before any adaptation technique can be invoked, it is essential to firstly identify the nature of a QoS violation for a correct solution. An illustrating example is that the same external phenomenon of packet loss in wireless communications can originate from various reasons (e.g., channel error or transmission congestion). Without a clear knowledge of the cause of an observed violation, one may not be able to apply correct remedies.

A bundle of QoS adaptation techniques have been proposed for dealing with QoS detection of applications [10,11]. An essential step before applying any effective adaptation solution is to correctly identify a violation. For example, if a delay violation is caused by incapability of receiving host to decode video frames, it makes no senses to re-allocate network resource for a higher throughput. The detailed description of QoS violation analysis is out of the scope of

Table 3 Comparison of Average Mean Training Time for Different Learning Algorithms

Name	ELM (seconds)	Gram-Schmidt (seconds)
	Training time	Training time
Abalone	**0.0717**	0.0942
Ailerons	**0.4379**	0.5656
Airplane	**0.0159**	0.0234
Bank	**0.1203**	0.1990
Boston	**0.0087**	0.0187
California	**0.1932**	0.3049
Census	**0.2853**	0.4280
Delta Ailerons	**0.0669**	0.0718
Delta Elevators	**0.0927**	0.1449

this paper and will be presented elsewhere. QoS violation classification problems have usually been implemented by using if-then rules. One advantage of rule based methods is that it can provide real-time (online) response. However, most current work tackles QoS violations through simple if-then rules, the performance of which is not convincing in practice. Defining and managing those rules or clauses has proven to be a difficult and time consuming task. Moreover, once rules are determined, they are difficult to change with respect to changes in the operating environment. Hence for every new case, one needs to redesign its rules. Neural networks can provide powerful high level artificial intelligence (AI) methods for classification with good accuracy. However, this high level AI comes with a computational burden, which leads to sluggish performance and hence is not acceptable for real-time applications. Since ELM can provide good generalization performance within a fast response time, we have successfully applied the fast learning algorithm to QoS violation detection, and showed that neural networks can correctly differentiate a QoS violation through the monitored data.

4.2.2 Data Collection

We conduct our experiments through monitoring and collecting data from audio/video streamings in our testbed. A stream sender (a desktop with Core 2 processor 2.13 GHz and 2 GB RAM) delivers datato a stream receiver (a desktop with Pentium 4 processor 2 GHz and 1 GB RAM). The sender and receiver are located in two sub-domains separated by a router running Fedroa 6. Monitoring facilities are deployed in the end-to-end system to collect performance data. A violation is asserted by observing application QoS while its type is identified through classification.

We setup a testbed for experiments as shown in Fig. 2. Two groups of PCs are separated by a router in between and forms two subnets. We deliver audio and video flows from the stream sender to the stream receiver. The audio source outputs an audio flow of PCM format, 44,100 Hz and Stereo quality. It is transcoded into either G723/RTP or MPA/RTP format for media streaming over the network. The video source outputs an video flow of format mpeg-1, 640×256 resolution and constant frame rate (30 fps). It is transcoded into either MPEG/RTP or JPEG/RTP format for transmission. We write the A/V streaming application on J2SE 6 platform using JMF library.[1] The RTP streaming between the sender and receiver uses UDP as the transport layer protocol. Monitors are placed at both streaming sender and receiver to

[1] http://www.java.sun.com/products/java-media/jmf/

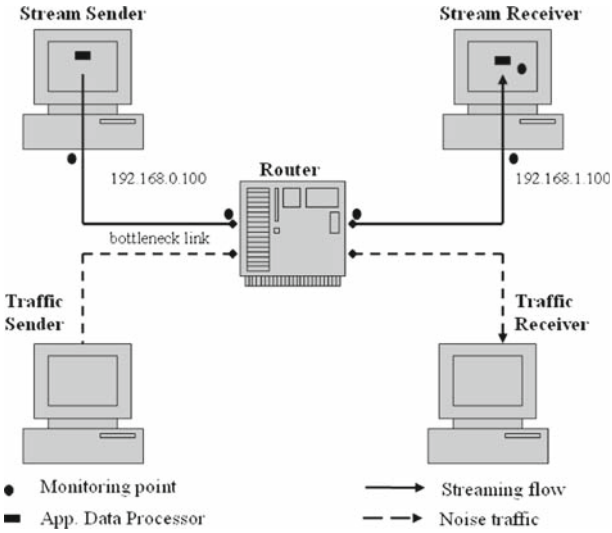

Fig. 2 Testbed environment

record end-to-end QoS. We use WinPcap[2] library to capture incoming traffics on the NIC of the receiver. We also integrate a frame rate monitor with the streaming application which measures the velocity of video display. As explained, video frame rate is defined as the QoS violation indicator with the delimiter set to 25 fps.

We first conduct a CPU violation test and collect corresponding violation data. A utility program is designed which can occupy CPU time slices at several scales, e.g., minimal (occupy 10–20% CPU time), medium (around 50%) or maximum (80–100%). By tuning the 'volume' of the utility program at either the sender or receiver, a CPU violation can be observed in the streaming application. Figure 3 shows change of application QoS (i.e., frame rate) and flow descriptors in a 120-second test, where 3,690 video frames are transmitted (Fig. 3a–c). The monitoring interval of application QoS is set to 2 s (Fig. 3d). In the first 60 s, no external interference is injected and hence the video streaming presents satisfactory performance. In the next 60 s, the CPU utility program is launched to contend with the streaming application for CPU time slice. A few violations can thus be observed where frame rate falls below a pre-defined level. Please note that the flow statistics (e.g., packet delay and jitter) shown here is the relative one-way values due to the clock screw between the sender and receiver.

Similarly, we have designed a network congestion case. A traffic generator is programmed which can generate noise traffic of either constant rate (emulate background traffics such as constant rate ftp flows) or normal distribution (emulate the arriving traffic of a core Internet router). The traffic generator is launched during streaming which delivery data to the traffic receiver sitting on the other subnet. By outputting a large volume of traffic, the traffic generator creates a congestion link on the end-to-end path of the streaming application. Figure 4 shows the measured performance of the video stream in a 120-second test. In the first 60 s, the traffic generator program yields a large volume traffic of normal distribution and leads to the performance degradation of the video flow. Packet loss is observed during this period which indicate the overflow of the router queue. The traffic generator is removed in the next

[2] http://www.winpcap.org

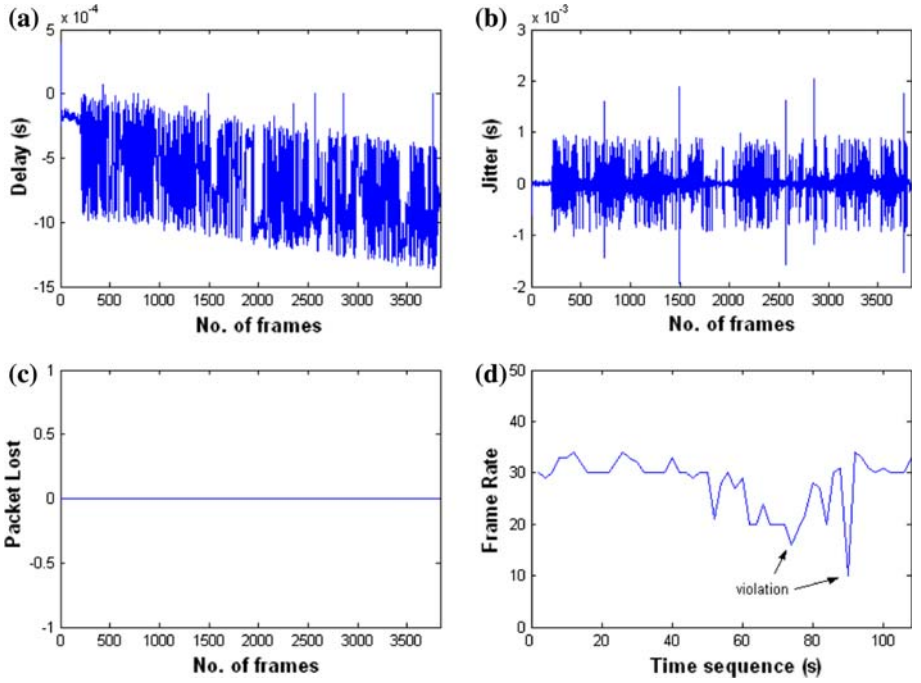

Fig. 3 Observation of end-to-end QoS w/ and w/o CPU contention

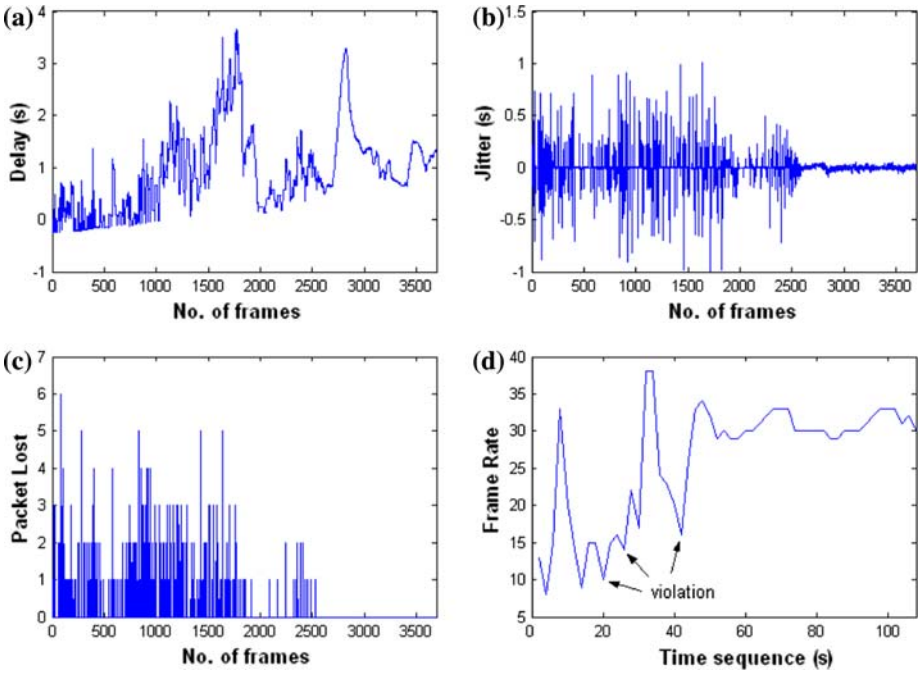

Fig. 4 Observation of end-to-end QoS w/ and w/o network congestion

Table 4 Specification of QoS violation datasets

[a] Three classes: Normal, CPU and Congestion

Name	No. of observations		Attributes
	Training data	Testing data	
QoS	4,000	3,524	3[a]

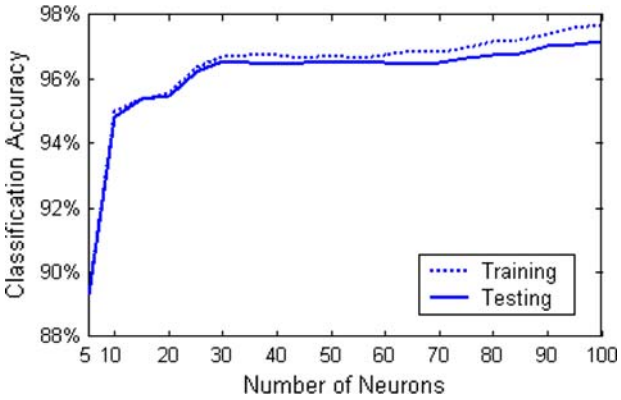

Fig. 5 Performance of the proposed ELM on QoS violation classification

Table 5 Classification accuracy for QoS violations

Name	Training accuracy		Testing accuracy		Time (s)	No. of neurons
	Mean (%)	Dev	Mean (%)	Dev		
ELM	96.70	0.0030	96.54	0.0036	0.0714	30
Gram-Schmidt	96.72	0.0035	96.48	0.0029	0.2526	30

60 s testing. As can be seen from the last part of Fig. 4d, the video flow gradually restores its quality, where frame rate varies within the acceptable range.

4.2.3 Performance Analysis

We use end-to-end flow descriptors such as packet delay, jitter and packet lost to distinguish among a normal transmission, a CPU violation and a congestion violation. We collected 7,524 groups of data for all the three cases and randomly selected 4,000 groups as training data. The rest are used as testing data. Table 4 gives the characteristics of QoS violation Datasets.

The activation function used in our proposed algorithms is a sigmoidal function $g(x) = 1/(1 + e^{-x})$ for additive neurons. The input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-1, 1]$. The experiment results are computed based on 50 trials. We have gradually increased the number of neurons from 5 to 100 with the increscent pace of 5 in each round and measured corresponding classification accuracy. The data analysis results are shown in Fig. 5. It shows that higher classification accuracy can be achieved with the growth of neuron numbers, which also verifies our approximation conclusion.

Table 5 shows the training and testing RMSE of ELM and Gram-Schmidt orthonormal neural networks with the 30th neuron. Seen from Table 5, the two neural networks both

achieve good generalization performances with the same error level, which also supports our equivalent relationship and universal approximation conclusions. From Table 5, ELM takes less training time than the orthonormal method. Therefore ELM is a more effective method, such an orthonormal transformation is not necessary.

5 Conclusion

In this paper, we systemically analyze two fast learning algorithms, which both have the same property, i.e., the parameters of hidden neurons are random. Though many experimental results show that neural networks with ELM can achieve a good generalization performance at a very fast speed, so far there lacks a strict theoretical justification whether they are universal approximation. Here we first refer to the universal approximation of neural networks transformed by the orthonormal transformation, and then by illustrate the equivalent relationship between ELM and the orthonormal networks, we successfully prove that neural networks with ELM are also universal approximation. Without the need for orthonormal transformation, ELM can achieve faster speed processing while yielding the same generalization performance. Simulation results based on benchmark regression and internet traffic QoS violation classification support our theories.

Acknowledgements The author would like to thank Dr. Guang-Bin Huang, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, for his valuable research. His suggestions are tremendously helpful to our study.

References

1. Huang G-B, Zhu Q-Y, Siew C-K (2006) Extreme learning machine: theory and applications. Neurocomputing 70:489–501
2. Ito Y (1991) Approximation of functions on a compact set by finite sums of a sigmoid function without scaling. Neural Netw 4:817–826
3. Leshno M, Lin VY, Pinkus A, Schocken S (1993) Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Netw 6:861–867
4. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4:251–257
5. Kaminski W, Strumillo P (1997) Kernel orthonormalization in radial basis function neural networks. IEEE Trans Neural Netw 8(5):1177–1183
6. Huang G-B, Zhu Q-Y, Siew C-K (2006) Real-time learning capability of neural networks. IEEE Trans Neural Netw 17(4):863–878
7. Stein EM, Shakarchi R (2005) Real analysis: measure theory, integration, and Hilbert spaces. Princeton University Press, NJ
8. Huang G-B, Chen L, Siew C-K (2006) Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans Neural Netw 17(4):879–892
9. Blake C, Merz C (1998) UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html, Department of Information and Computer Sciences, University of California, Irvine, USA
10. Poellabauer C, Abbasi H, Schwan K (2002) Cooperative run-time management of adaptive applications and distributed resources. In: Proceedings of ACM multimedia
11. Li B, Nahrstedt K (2000) Qualprobes: middleware qos profiling services for configuring adaptive applications. In: Proceedings of IFIP international conference on distributed systems platforms and open distributed processing