Neurocomputing 159 (2015) 78–83

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A randomized ELM speedup algorithm

Chang-qian Men [a,b], Wen-jian Wang [a,b,*]

[a] School of Computer and Information Technology, Shanxi University, Shanxi, Taiyuan 030006, China
[b] Key Laboratory of Computational Intelligence and Chinese Information Processing (Shanxi University), Ministry of Education, Shanxi, Taiyuan 030006, China

ARTICLE INFO

Article history:
Received 19 December 2014
Received in revised form
5 February 2015
Accepted 8 February 2015
Communicated by G.-B. Huang
Available online 26 February 2015

Keywords:
Extreme learning machine
Randomized approximation
Kernel

ABSTRACT

Extreme learning machine (ELM) as an emergent technology has shown its good performance in classification applications. However, ELM algorithm needs to find the inversion of matrix in nature, which will limit its application on many occasions. This paper proposes an ELM speedup algorithm based on the analysis of ELM algorithm. By applying randomized approximation method, the proposed algorithm can approximate the key matrix (For example, the kernel matrix in the kernel-based ELM) with a low-rank matrix. By doing so, the complexity of the inversion can be reduced from $O(n^3)$ to $O(kn^2+k^3)$ (n is the size of the data set, and k is the numerical rank of the approximated matrix). On the premise of not decreasing the accuracy too much, the training time can be cut down substantially, which has important significance in practical application of machine learning algorithms. The experimental results on benchmark data sets demonstrate the effectiveness of the proposed algorithm.

1. Introduction

With the rapid development of machine learning theories in recent years, various of machine learning algorithms are applied to many practical problems, and have made great achievements. But there is a limitation existing in most of the algorithms, that is, due to the high computation complexity, these algorithms are infeasible in many practical problems. Some scholars have made many efforts to improve the learning efficiency of algorithms, but many machine learning algorithms are based on eigenvalue decomposition or matrix inversion, such as PCA, manifold learning, spectral clustering, and LDA [1–4], or ensembles of these [5]. These standard algorithms need to compute the eigenvalue decomposition of a dense $n \times n$ matrix with $O(n^3)$ time, which is prohibited on many occasions. So it is of great value to design an effective machine learning algorithm.

ELM is originally proposed by Huang [6–8] for the single-hidden layer feedforward neural networks and then extended to the generalized single-hidden layer feedforward neural networks. And there also exist many ELM versions [9–12], e.g., I-ELM, OS-ELM, P-ELM, and C-ELM, which have been proved to be effective in applications. There are also some algorithms combined with ELM that are proposed. For example [13] combines SVM and ELM and proposes the Extreme Support Vector Machine, which shows that it has a better generalization performance than traditional SVM. In efficiency, ELM is proved better than traditional machine learning algorithms [6]. ELM is an eigenvalue-decomposition-based algorithm in nature, and therefore is faced with the above-mentioned problem of high computation complexity.

To solve the eigenvalue decomposition efficiently, the matrix low-rank approximation methods can be used. If we can find a low-rank approximation of the matrix, the eigenvalue decomposition on the low-rank matrix can be computed efficiently. Among many matrix low-rank approximation methods, the random approximation methods are well studied. The simplest approach to random matrix approximation is the method of sparsification. The goal of sparsification is to replace the matrix with fewer nonzero entries (which are drawn independently at random from a distribution determined from the input matrix) [14]. The second approach to matrix approximation is based on the concept of dimension reduction. A random linear map provides an efficient, nonadaptive way to perform this reduction [15]. The third approach can be called the compact matrix decomposition which expresses $A \approx CUR$, where C and R denote small column and row submatrices (chosen according to a random algorithm) of A and where U is a small matrix [16].

Based on these researches, Halko et al. [17] proposed a new randomized matrix approximation method. They use random sampling to identify a subspace to capture the action of the matrix. In this way the key matrix of the data set can be approximated by a low-rank matrix, and then the inversion or eigenvalue decomposition of matrix can be acquired based on it. This approximation can effectively reduce

* Corresponding author at: School of Computer and Information Technology, Shanxi University, Shanxi, Taiyuan 030006, China.
E-mail address: wjwang@sxu.edu.cn (W.-j. Wang).

http://dx.doi.org/10.1016/j.neucom.2015.02.018
0925-2312/

the computation complexity from $O(n^3)$ to $O(k^3+kn^2)$. It is found in practice $k<n$ can be set without any significant decrease in accuracy of the solution.

This paper combines the ELM with the randomized approximation method to accelerate the training time of ELM, which is testified effective and reasonable by data simulation. The paper is organized as follows: Sections 2 and 3 give a brief introduction of ELM and the randomized approximation method. Section 4 explains the proposed algorithm in detail. Section 5 provides the experimental results and the analysis. The last section concludes the whole work.

2. ELM

Extreme learning machine (ELM) was originally proposed for the single-hidden layer feedforward networks and was extended to the generalized single-hidden layer feedforward networks where the hidden layer need not be neuron alike [6]. The output function of ELM for generalized SLFNs (take one output node case as an example) is

$$f_L(\mathbf{x})=\sum_{i=1}^{L}\beta_i h_i(\mathbf{x})=\mathbf{h}(\mathbf{x})\boldsymbol{\beta} \tag{1}$$

where $\boldsymbol{\beta}=[\beta_1,\beta_2,\ldots,\beta_L]^T$ is the vector of the output weights linking the hidden layer of L nodes to the output node, and $\mathbf{h}(\mathbf{x})=[h_1(\mathbf{x}),\ldots,h_L(\mathbf{x})]$ is the output vector with respect to the input $\mathbf{x}$. $\mathbf{h}(\mathbf{x})$ maps the data from d-dimensional input space to L-dimensional hidden layer feature space.

According to ELM learning theory, widespread type of feature mapping $\mathbf{h}(\mathbf{x})$ can be used in ELM so that ELM can approximate any continuous target function. That is, given any target continuous function $f(\mathbf{x})$, there exists a series of β_i such that

$$\lim_{L\to\infty}\|f_L(\mathbf{x})-f(\mathbf{x})\|=\lim_{L\to\infty}\|\sum_{i=1}^{L}\beta_i h_i(\mathbf{x})-f(\mathbf{x})\|=0 \tag{2}$$

The classification capability of SLFN can be described by the following theorem [6,7]:

Theorem 2.1. *Given a feature mapping* $\mathbf{h}(\mathbf{x})$*, if* $\mathbf{h}(\mathbf{x})\boldsymbol{\beta}$ *is dense in* $C(R^d)$*, then a generalized SLFN can separate arbitrary disjoint regions of any shape in* R^d.

Since ELM can approximate any target continuous function and the output of the ELM classifier $\mathbf{h}(\mathbf{x})\boldsymbol{\beta}$ can be as close to the class labels in the corresponding regions as possible. Here we consider the multi-class case, where the number of output nodes of the ELM is q (q is the number of classes). If x_i belongs to class p, then the output node is $\mathbf{t}_i=[0,\ldots,0,1,0,\ldots,0]^T$ (only the pth element of $\mathbf{t}_i$ is one; the others are set zero). The classification problem for ELM can be formulated as follows [6]:

$$\textbf{Minimize}: \quad L_{PELM}=\frac{1}{2}\|\boldsymbol{\beta}\|^2+C\frac{1}{2}\sum_{i=1}^{n}\|\boldsymbol{\xi}_i\|^2 \tag{3}$$

$$\textbf{s.t.}: \quad \mathbf{h}(\mathbf{x}_i)\boldsymbol{\beta}=\mathbf{t}_i^T-\boldsymbol{\xi}_i^T \tag{4}$$

where $\boldsymbol{\beta}=[\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_q]$, $\mathbf{t}_i=[t_{i1},\ldots,t_{iq}]^T$, $\boldsymbol{\xi}_i=[\xi_{i1},\ldots,\xi_{iq}]^T$. $\boldsymbol{\xi}_i$ is the training error vector of the q output nodes with respect to the training sample $\mathbf{x}_i$. $\boldsymbol{\beta}_j$ is the vector of the weights linking hidden layer to the jth output node.

Based on the Karush–Kuhn–Tucker theorem, to train ELM is equivalent to solve the following dual optimization problem:

$$L_{DELM}=\frac{1}{2}\|\boldsymbol{\beta}\|^2+C\frac{1}{2}\sum_{i=1}^{n}\|\boldsymbol{\xi}_i\|^2-\sum_{i=1}^{n}\sum_{j=1}^{q}\alpha_{i,j}(\mathbf{h}(\mathbf{x}_i)\boldsymbol{\beta}_j-t_{i,j}+\xi_{i,j}) \tag{5}$$

The corresponding optimality conditions are as follows:

$$\frac{\partial L_{DELM}}{\partial\boldsymbol{\beta}_j}=0\to\boldsymbol{\beta}_j=\sum_{i=1}^{n}\alpha_{i,j}\mathbf{h}(\mathbf{x}_i)^T \tag{6}$$

$$\frac{\partial L_{DELM}}{\partial\boldsymbol{\xi}_i}=0\to\boldsymbol{\alpha}_i=C\boldsymbol{\xi}_i \tag{7}$$

$$\frac{\partial L_{DELM}}{\partial\boldsymbol{\alpha}_i}=0\to\mathbf{h}(\mathbf{x}_i)\boldsymbol{\beta}-\mathbf{t}_i^T+\xi_i^T=0 \tag{8}$$

where $\boldsymbol{\alpha}_i=[\alpha_{i1},\ldots,\alpha_{iq}]^T$, $\boldsymbol{\alpha}=[\alpha_1,\ldots,\alpha_n]^T$.

Substituting (6) and (7) into (8), the equation can be written as follows:

$$\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)\boldsymbol{\alpha}=\mathbf{T} \tag{9}$$

where

$$T=\begin{bmatrix}\mathbf{t}_1^T\\ \vdots\\ \mathbf{t}_n^T\end{bmatrix}=\begin{bmatrix}t_{11} & \cdots, & t_{1q}\\ \vdots & \vdots & \vdots\\ t_{n1} & \cdots, & t_{nq}\end{bmatrix} \tag{10}$$

And the weights matrix $\boldsymbol{\beta}$ is

$$\boldsymbol{\beta}=\mathbf{H}^T\left(\frac{I}{C}+HH^T\right)^{-1}T \tag{11}$$

The output function of ELM classifier is

$$\mathbf{f}(\mathbf{x})=\mathbf{h}(\mathbf{x})\boldsymbol{\beta}=h(x)H^T\left(\frac{I}{C}+HH^T\right)^{-1}T \tag{12}$$

The label of the sample x is

$$\text{label}(\mathbf{x})=\arg\max f_i(\mathbf{x}) \tag{13}$$

where $\mathbf{f}(\mathbf{x})=[f_1(\mathbf{x}),\ldots,f_q(\mathbf{x})]^T$.

If a feature mapping $\mathbf{h}(\mathbf{x})$ is unknown, one can apply Mercers conditions on ELM. The kernel matrix for ELM can be written as follows:

$$\Omega_{ELM}=\mathbf{H}\mathbf{H}^T:\Omega_{ELM_{ij}}=\mathbf{h}(\mathbf{x}_i)\cdot h(x_j)=K(\mathbf{x}_i,\mathbf{x}_j) \tag{14}$$

and the output function is

$$\mathbf{f}(\mathbf{x})=\mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{I}{C}+HH^T\right)^{-1}T=\begin{bmatrix}\mathbf{K}(\mathbf{x},\mathbf{x}_1)\\ \vdots\\ \mathbf{K}(\mathbf{x},\mathbf{x_n})\end{bmatrix}^T\left(\frac{I}{C}+\Omega_{ELM}\right)^{-1}T \tag{15}$$

3. Randomized approximation

For the ELM algorithm, the most important step to solve the problem is to calculate the inversion of the kernel matrix. However, it is not an easy task in most cases. If the size of the training data set is n, the computation complexity of finding the inversion of kernel matrix is about $O(n^3)$. It will be time consuming. So the algorithm is infeasible in practice. Huang et al. [6] gives an alterative solution to this problem. The output function can be rewritten as follows:

$$\mathbf{f}(\mathbf{x})=\mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{\mathbf{C}}+\mathbf{H}^\mathbf{T}\mathbf{H}\right)^{-1}\mathbf{H}^\mathbf{T}\mathbf{T} \tag{16}$$

By doing so, the size of the matrix which needs to find the inversion is $L\times L$ (L is the number of nodes in the hidden layer, $L\ll n$). The cost of computation can be reduced. In such cases, the feature mapping $\mathbf{h}(\mathbf{x})$ must be known, and that will limit the use of ELM algorithm on many occasions.

To handle this problem, Halko et al. [17] give a low-rank approximation method to a given matrix. By doing so, the time complexity of SVD on a low-rank matrix can be reduced to a large

extent. The method can be split naturally into two stages. The first is to construct a low-dimensional subspace that captures the action of the matrix. The second is to restrict the matrix to the subspace and then compute a standard factorization (QR, SVD, etc.) of the reduced matrix. It can be expressed as follows:

Stage 1: Compute an approximate basis for the range of the input matrix A. In other words, an orthonormal matrix Q can be found to satisfy

$$A \approx QQ^T A$$

The basis matrix Q should contain as few columns as possible, but it is even more important to have an accurate approximation of the input matrix.

Stage 2: Matrix Q can be used to compute a standard factorization (QR, SVD, etc.) of A.

The detail of the algorithm is described in Algorithm 1 [17].

Algorithm 1. The approximate SVD algorithm.

Input: Matrix $A_{m\times n}$.

Output: The approximate SVD decomposition of A, $A \approx U\Sigma V^T$.

1: Construct the matrix $Q_{m\times k}$, which makes $A \approx QQ^T A$.
2: Calculate the matrix $B_{k\times n}$, where $B = Q^T A$.
3: Perform SVD on B where $B = \hat{U}\Sigma V^T$.
4: Obtain $U = Q\hat{U}$

How to find the orthonormal matrix Q is a key problem to be solved. Halko use random sampling to identify such matrix Q. Suppose that to seek a basis for the range of matrix A with exact rank k. Draw a random vector $\boldsymbol{\omega}$, and form the product $\mathbf{y} = \mathbf{A}\boldsymbol{\omega}$. For now, the precise distribution of the random vector is unimportant; just think of y as a random sample from the range of A. Repeat this sampling process k times:

$$y^{(i)} = A\omega^{(i)}, \quad i = 1, 2, \ldots, k$$

Owing to the randomness, the set $\{\omega^{(i)} : i = 1, 2, \ldots, k\}$ of random vectors is likely to be in general linear position. In particular, the random vectors form a linearly independent set and no linear combination falls in the null space of A. As a result, the set $y^{(i)} : i = 1, 2, \ldots, k$ of sample vectors is also linearly independent, so it spans the range of A. Therefore, to produce an orthonormal basis for the range of A, we just need to orthonormalize the sample vectors.

The most natural way to draw a random matrix Ω can be from the standard Gaussian distribution. That is, each entry of Ω is an independent Gaussian random variable with mean zero and variance one. The detail can be described in Algorithm 2.

Algorithm 2. The randomized approximate SVD algorithm.

Input: Matrix $A_{m\times n}$.

Output: The approximate SVD decomposition of A, $A \approx U\Sigma U^T$.

1: Generate a $n \times k$ Gaussian random matrix Ω.
2: Construct the matrix $Y_{m\times k}$, $Y = A\Omega$.
3: Construct a matrix Q whose columns form an orthonormal basis for the range of Y.
4: Construct the matrix $Q_{m\times k}$, which makes $A \approx QQ^T A$.
5: Calculate the matrix $B_{k\times n}$, where $B = Q^T A$.
6: Perform SVD on B where $B = \hat{U}\Sigma V^T$.
7: Obtain $U = Q\hat{U}$.

The goal of Algorithm 2 is to produce an orthonormal matrix Q with few columns that achieves

$$\|A - QQ^T A\| \le \varepsilon$$

To achieve the given tolerance ε, the column number of Q is usually larger than the target rank k, which means $l = p + k$ columns are needed. p is referred to as the oversampling parameter. The size of the oversampling parameter depends on several factors: (1) The matrix dimensions. Very large matrices may require more oversampling. (2) The singular spectrum. The more rapid the decay of the singular values, the less oversampling is needed. (3) The random matrix. For Gaussian matrices, it is adequate to choose the oversampling parameter to be a small constant, such as $p = 5$ or $p = 10$.

Halko et al. [17] also give a bound on the expectation of the error for Gaussian matrices:

Theorem 3.1. *Suppose that A is a $m \times n$ matrix, target rank is k, and an oversampling parameter is $p \ge 2$. The singular value of A is $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Then*

$$E\|A - QQ^T A\|_F \le \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \tag{17}$$

and

$$E\|A - QQ^T A\| \le \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \tag{18}$$

This theorem shows that the error bound is related to the singular value of matrix A. As is known, $\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$ is the minimal Frobenius-norm error when approximating A with a rank-k matrix. The error bound in the theorem gives similar results. When the singular values exhibit some decay, the error of random approximation is rather small. If A has an exact rank k or even less, then $A = QQ^T A$ with very high probability.

4. The proposed algorithm

According to the analysis above, if we can get the random approximation of M (For kernel-based ELM, $M=K$, K is the kernel matrix. For non-kernel based ELM, $M = H^T H$.) in the form $M = UDU^T$, the inversion of matrix $I/C + M$ can be solved easily by using WoodBury formula:

$$\left(\frac{I}{C} + M\right)^{-1} \approx \left(\frac{I}{C} + \tilde{M}\right)^{-1} = C\left(I - G\left(\frac{I}{C} + G^T G\right)^{-1} G^T\right) \tag{19}$$

where $G = UD^{1/2}$ (In fact, $\tilde{M}$ can be reformulated as $\tilde{M} = GG^T$, and G is a matrix with the size $n \times k$). It is clear that we only need to calculate the inversion of matrix with the size $k \times k$ instead of the inversion of a matrix with the size $n \times n$. The total computation complexity of calculating the inversion of matrix $\left((1/C)I + \tilde{M}\right)^{-1}$ is $O(kn^2 + k^3)$. Algorithm 3 is the proposed algorithm.

Algorithm 3. The randomized approximate ELM algorithm.

Input Training set $X_{n\times m}$ (n is the size of training set, and m is the feature number of the data), k, C, other parameters(i.e. kernel parameter) and test sample $\mathbf{x}_{1\times m}$.

Output The label of the test sample $\mathbf{x}$ (which belongs to the set $\{1, 2, \ldots, q\}$).

1: Construct the matrix M. (For kernel-based ELM, $M=K$, where $K_{ij} = k(x_i, x_j)$, $x_i, x_j \in X = \{x_k\}_{k=1}^n$. For non-kernel based ELM, $M = H^T H$.)
2: Generate a $n \times (k+p)$ Gaussian random matrix Ω.
3: Form a $n \times (k+p)$ matrix Y, $Y = M\Omega$.

4: Construct a matrix Q whose columns form an orthonormal basis for the range of Y (This can be done through QR decomposition.)

5: Form a $(k+p)\times n$ matrix B, $B=Q^TM$.

6: Form a $(k+p)\times(k+p)$ matrix T, $T=BQ$.

7: Perform SVD decomposition on T, $T=\widehat{U}D\widehat{U}^T$.

8: Calculate the $n\times(k+p)$ matrix U, $U=Q\widehat{U}$ ($\tilde{M}$ is the random approximate of M, and $\tilde{M}=UDU^T$).

9: Obtain $G=UD^{1/2}$, $\tilde{M}=GG^T$.

10: Calculate $(M+I/C)^{-1}\approx C\left(I-G\left(\frac{I}{C}+G^TG\right)^{-1}G^T\right)$.

11: Compute the output function (use kernel-based form as an example):

$$\mathbf{f}(\mathbf{x})=\mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{\mathbf{C}}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T}=\begin{bmatrix}\mathbf{k}(\mathbf{x},\mathbf{x}_1)\\ \cdots\\ \mathbf{k}(\mathbf{x},\mathbf{x}_N)\end{bmatrix}^T\mathbf{C}\left(\mathbf{I}-\mathbf{G}\left(\frac{\mathbf{I}}{\mathbf{C}}+\mathbf{G}^\mathbf{T}\mathbf{G}\right)^{-1}\mathbf{G}^\mathbf{T}\right)\mathbf{T} \quad (20)$$

The label of $\mathbf{x}$ is argmax $f_i(x)$.

The computation complexity can be estimated as follows. To simplify the discussion, we use k to replace $k+p$ (compared with k, p is relatively small, $k\approx k+p$). According to Algorithm 3, in steps 3 and 5, we need to calculate the matrix Y and B. The complexity is $O(kn^2)$ (the size of M is $n\times n$, and the size of Ω is $n\times k$, so the complexity of matrix multiplication is $O(kn^2)$). In steps 6 and 8, the time complexity is $O(k^2n)$. As is known, in step 7, the complexity of SVD on a matrix sized $k\times k$ is $O(k^3)$. And in step 4, to construct orthonormal matrix Q, we perform QR decomposition on matrix $Y_{n\times k}$ whose time complexity is $O(nk^2)$. In step 1, to construct the given matrix, we need $n\times n$ operations. If the time cost of generating the Gaussian random matrix is T_{ran}, the total time complexity of the proposed algorithm is $T_{ran}+c_1kn^2+c_2nk^2+c_3k^3$. In most cases, $k\le n$ and $T_{ran}<kn^2$, so we can write it in the form $O(kn^2+k^3)<O(n^3)$.

Table 1
Data set used.

Datasets	Training data	Testing data	Classes	Features
dna	1400	1186	3	180
letter	2861	1003	5	16
optdigits	3823	1797	10	64
segment	2061	249	7	19
svmguide1	3089	4000	2	4
waveform	5000	1000	3	21
usps	7291	2007	10	256

5. Experiments and analysis

In order to verify the proposed algorithm, we will test it on benchmark data sets. And to evaluate this algorithm, we will compare the randomized approximate ELM algorithm with the traditional one [6]. The used data sets are listed in Table 1. The data sets *dna, letter* and *svmguide1* are from LIBSVM data sets [18] (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/), and the data sets *optdigits, segment, waveform, usps* and *shuttle* are from UCI benchmark data sets.

From Table 1, there are altogether 26 classes in the *letter* data set, corresponding to the 26 English letters. We select 5 classes from it, i.e. A–E. The experiments are carried out in MATLAB 7.10.0 environment running in Dual-Core, 2.6-GHZ CPU with 2-GB RAM, and the operating system is 32bit Win7. Thirty trials have been conducted for each data set. The corresponding standard deviations (Dev) are given in the simulation results.

Table 2 is the experiment results of comparing the standard ELM with the Randomized Approximate ELM whose output weight β is in the form

$$\boldsymbol{\beta}=\mathbf{H}^T\left(\frac{I}{C}+HH^T\right)^{-1}T \quad (21)$$

Here the algorithm 3 is used to approximate the matrix $\left(I/C+HH^T\right)^{-1}$.The function of the hidden node is sigmoid, and the number of the hidden node is 1000. The values of parameter C are chosen ranging from 2^{-15} to 2^{15}. The best one is selected as the optimal parameter. Due to the fact that the values of $\{a_i,b_i\}$ (parameters in the sigmoid function) in ELM are generated randomly and the Randomized Approximate ELM is also a random algorithm, the values of C in the two algorithms are set differently. To validate the speedup performance, in the Randomized Approximate ELM algorithm we choose $k=0.01n, 0.05n, 0.1n$ (n is the size of training data set). Here we use the word Randomized AELM standing for Randomized Approximate ELM.

From Table 2, it can be seen that as the k value increases, the test rates of Randomized Approximate ELM are as good as the standard ELM. When $k=0.1n$, the test rates of most data sets are almost the same as the standard ELM, and the Dev values also become very small. Compared with the standard ELM, the training time of Randomized Approximate ELM is reduced to a large extent. However, as k increases, the training time of Randomized Approximate ELM increases too. So there is a tradeoff between the speedup performance and the accuracy. Another fact to be mentioned is that compared with the standard ELM, the performance of Randomized Approximate ELM is more sensitive to the value C. The reason may be that there are too much Randomness in the algorithm.

Table 2
Comparisons between the ELM with the randomized approximate ELM with $\beta=H^T\left(H^TH+\frac{I}{C}\right)^{-1}T$.

Datasets	Random ELM				Randomized AELM								
	C	Testing rate (%)	Dev	Training time (s)	C	$k=0.01n$			$k=0.05n$			$k=0.1n$	
						Testing rate (%)	Dev	Training time (s)	Testing rate (%)	Dev	Training time (s)	Testing rate (%)	Dev	Training time (s)
dna	2^{-7}	93.03	0.38	2.98	2^{-9}	84.91	1.27	1.09	89.03	1.00	1.35	92.46	0.49	1.53
optdigits	2^{-5}	96.83	0.34	38.46	2^{-8}	87.09	2.19	8.72	96.25	0.41	12.62	96.42	0.33	18.95
letter	2^{-2}	98.19	0.22	17.13	2^{-5}	68.24	4.49	4.98	97.22	0.45	6.50	97.47	0.36	8.90
segment	2^{3}	92.78	1.10	8.50	2^{-5}	62.62	4.40	2.56	89.06	1.86	3.02	91.57	1.05	4.36
svmguide1	2^{10}	94.29	0.46	20.92	2^{-1}	61.82	6.39	5.57	75.34	2.99	7.60	94.90	0.72	10.29
waveform	2^{4}	92.16	0.47	69.57	2^{-5}	64.18	3.25	14.61	87.92	1.00	21.46	90.30	0.55	37.85
usps	2^{-1}	93.66	0.23	292.85	2^{-8}	74.57	2.08	17.15	93.52	0.44	31.08	93.44	0.20	53.87

Table 3
Comparison between the ELM with approximate ELM with Gaussian kernel.

Datasets	Parameter		Gaussian kernel ELM		Gaussian kernel Randomized AELM											
	C	γ	Testing rate (%)	Training time (s)	$k=0.01n$				$k=0.05n$				$k=0.1n$			
					Testing rate (%)	Dev	Training time (s)	Fnorm	Testing rate (%)	Dev	Training time (s)	Fnorm	Testing rate (%)	Dev	Training time (s)	Fnorm
dna	2^{-1}	2^{-6}	92.83	1.76	88.54	0.71	0.29	33.36	91.50	0.68	0.51	21.81	93.08	0.34	0.92	12.01
letter	2^{0}	2^{-2}	99.20	12.68	97.30	0.76	1.27	93.90	96.76	0.95	2.91	68.64	98.90	0.29	5.78	54.95
optdigits	2^{-1}	2^{-6}	98.33	26.18	98.07	0.31	2.21	66.14	98.11	0.57	6.34	63.34	98.21	0.06	11.97	60.50
segment	2^{-1}	2^{-5}	97.19	26.86	96.67	0.21	1.52	57.90	96.95	0.27	4.24	53.90	96.93	0.27	7.13	49.70
svmguide1	2^{-1}	2^{-6}	96.88	15.48	93.45	1.42	2.23	129.61	93.79	0.88	6.15	79.60	96.40	0.23	10.79	54.32
waveform	2^{-1}	2^{-9}	87.30	59.33	87.27	0.06	4.07	0.77	87.30	0	11.33	0.08	87.30	0	28.03	0.01
usps	2^{0}	2^{1}	95.07	112.24	92.20	0.19	6.42	41.44	94.82	0.10	24.05	40.46	95.03	0.03	52.24	39.27

Table 4
Comparisons between the ELM with the randomized approximate ELM with linear kernel.

Datasets	C	Linear kernel ELM			Linear kernel randomized AELM			Time ratio	Fnorm
		Testing rate (%)	Dev	Training time (s)	Testing rate (%)	Dev	Training time (s)		
dna	2^{-8}	92.24	0	1.56	92.24	0	1.06	1.50	1.12e−9
optdigits	2^{-9}	91.82	0	29.45	91.82	0	3.04	9.71	3.87e−7
letter	2^{-9}	87.14	0	12.75	87.14	0	1.20	10.66	2.79e−8
segment	2^{1}	81.12	0	5.23	81.12	0	0.73	7.28	1.01e−6
svmguide1	2^{8}	76.90	0	15.44	76.90	0	1.23	12.58	8.08e−7
waveform	2^{9}	85.80	0	61.02	85.80	0	3.41	17.93	1.15e−8
usps	2^{-9}	87.39	0	273.27	87.39	0	16.97	16.04	6.38e−8

Tables 3 and 4 are the experiment results of kernel matrix approximation ELM. Here the chosen kernel functions are mainly linear kernel and Gaussian kernel, with the form of the latter being $k(x,y)=\exp(-\gamma\|x-y\|^2)$ [19]. Meanwhile the value of $\varepsilon=\|K-\tilde{K}\|_F$(Fnorm) is given in the experiments, i.e. the approximate error of the kernel matrix. The parameter C and kernel parameter γ of ELM also need to be chosen appropriately. For each data set, we have used 21 different values of C and 21 different values of γ ranging from 2^{-15} to 2^{15}, i.e. $\{2^{-15},2^{-14},\ldots,2^{14},2^{15}\}$. We choose the best combination of $\{C,\gamma\}$ as the optimal parameters. Here it should be noticed that when applying kernel mapping, the standard ELM is no longer a randomized algorithm, and the Dev values in the experiments are nearly 0.

Table 3 is the experiment results by applying the Gaussian kernel. Different from the linear kernel, due to the non-linear mapping, we could not know the rank of the Gaussian kernel matrix. So here we also choose $k=0.01n, 0.05n, 0.1n$ to validate the speedup effectiveness. It can also be seen from Table 3 that the larger the k, the closer the test rate of Randomized Approximate ELM is to the standard ELM. But the training time of the proposed method gets longer and longer. Obviously there is also a tradeoff between the speedup performance and the accuracy as we have concluded in Table 2. Another fact we can draw from Table 3 is that for different data sets, to achieve desirable speedup performance without losing too much accuracy, the k value is different. The choice of k depends on the property of the data set. For example, in Table 3, the *usps* data set needs only $k=0.01n$ to achieve the same test rate with the standard ELM, while for the *dna* and *optdigits* data set the k value needs to be $0.1n$. How to choose k value is an important problem which needs to be explored. During the execution of the algorithm, most of the time cost is the matrix multiplication. If the matrix multiplication operations can be improved, the time of the proposed algorithm can be cut down further.

Table 4 is the experiment results by applying linear kernel when $k=\min\{n,m\}$ (n is the number of samples, and m is the number of features. Here $m\ll n$, so we choose $k=m$.). To evaluate the speedup performance, the *Time Ratio* is defined as Training Time of the standard ELM/Training Time of the Randomized Approximated ELM. When applying linear kernel, we have the following simple fact that

$$\mathrm{rank}(K)=\mathrm{rank}(XX^T)\le\mathrm{rank}(X)\le\min\{m,n\}$$

It can be observed that the approximate errors of kernel matrix in most data sets are rather small, and the test rates of the proposed algorithm are almost the same with the standard ELM.

In Theorem 3.1, the upper bound of matrix approximation error is determined by $\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$. As mentioned above, the Frobenius norm of the best rank-k approximation error is $\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$. So according to the relation $\varepsilon=\|K-\tilde{K}\|_F>\min_{rank(\tilde{K})=k}=\|K-\tilde{K}\|_F=\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$, the smaller the Fnorm ($\varepsilon=\|K-\tilde{K}\|_F$) value, the better the matrix approximation. In the experiments, we can see that the Fnorm ($\varepsilon=\|K-\tilde{K}\|_F$) values in both linear kernel and Gaussian kernel are rather small. So the matrix approximation errors are correspondingly small. This also explains the reason that if we choose a larger k, the Fnorm value is decreased and the matrix approximation error also decreases, and the test rates are increased (especially for linear kernel, if we choose $k=\min\{m,n\}$, the Fnorm values of approximation error are close to 0, so Linear kernel ELM and Linear kernel Randomized Approximate ELM lead to similar results).

6. Conclusion

This paper proposes an ELM speedup algorithm based on the analysis of the ELM algorithm. By applying the randomized

approximation method, the proposed algorithm approximates the key matrix with a low-rank matrix. The data experiments verify the effectiveness of the proposed algorithm. On the premise of not lowering the accuracy too much, the training time can be cut down substantially, proving the practical value of the machine learning algorithm.

Also, it can be concluded that, when $k < n$, i.e. $O(kn^2 + k^3)$ the needed training time depends on k value. Obviously, the smaller the k value, the less the training time, and the less the accuracy. So k value is an important parameter affecting the classification performance of the algorithms. And how to select an appropriate k value is worth our in-depth research. Another drawback of the proposed algorithm is that it must store the key matrix (e.g. kernel matrix K) in the memory. In large-scale data sets, the algorithm may not work efficiently.

Acknowledgments

The work was partially supported by the National Natural Science Foundation of China (Nos. 60975035,71031006,61273291), Doctoral Fund of Ministry of Education of China (No. 20091401110003), Research Project Supported by Shanxi Scholarship Council of China (No. 2012-008), and Scientific and Technological Project of Shanxi Province (No. 20120321027-1).

References

[1] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in: Advances in Neural Information Processing Systems, vol. 14, MIT Press, Cambridge MA, 2002, pp. 585–591.
[2] J. Tenenbaum, V. Silva, J. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (2000) 2319–2323.
[3] V. Luxburg, A tutorial on spectral clustering, Stat. Comput. 17 (4) (2007) 395–416.
[4] E. Cambria, Y. Song, H. Wang, N. Howard, Semantic multi-dimensional scaling for open-domain sentiment analysis, IEEE Intell. Syst. 29 (2) (2014) 44–51.
[5] E. Cambria, T. Mazzocco, A. Hussain, Application of multi-dimensional scaling and artificial neural networks for biologically inspired opinion mining, Biol. Inspir. Cogn. Arch. 4 (2013) 41–53.
[6] G.B. Huang, H.M. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multi-class classification, IEEE Trans. Syst. Man Cybern.—Part B: Cybern. 42 (2) (2012) 513–529.
[7] G.B. Huang, Y.-Q. Chen, H.A. Babri, Classification ability of single hidden layer feedforward neural networks, IEEE Trans. Neural Netw. 11 (3) (2000) 799–801.
[8] G.B. Huang, X. Ding, H. Zhou, Optimization method based extreme learning machine for classification, Neurocomputing 74 (2010) 155–163.
[9] G.B. Huang, X. Ding, H. Zhou, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[10] H. Rong, Y.S. Ong, A.H. Tan, Z. Zhu, A fast pruned extreme learning machine for classification problem, Neurocomputing 72 (2008) 359–366.
[11] S. Decherchi, P. Gastaldo, R. Zunino, E. Cambria, J. Redi, Circular-ELM for the reduced-reference assessment of perceived image quality, Neurocomputing 102 (2013) 78–89.
[12] N. Liang, G.B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate on-line sequential learning algorithm for feedforward networks, IEEE Trans. Neural Netw. 17 (6) (2006) 1411–1423.
[13] Q.G. Liu, Q. He, Z.Z. Shi, Extreme support vector machine classifier, in: T. Washio, E. Suzuki, K. Ting, A. Inokuchi (Eds.), Advances in Knowledge Discovery and Data Mining, vol. 5012 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 222–233.
[14] D. Achlioptas, F. Mcsherry, Fast computation of low-rank matrix approximations, J. ACM 54 (2007).
[15] V. Rokhlin, A. Szlam, M. Tygert, A randomized algorithm for principal component analysis, SIAM J. Matrix Anal. Appl. 31 (2009) 1100–1124.
[16] R.K. Petros Drineas, M.W. Mahoney, Fast monte carlo algorithms for matrices III: computing a compressed approximate matrix decomposition, SIAM J. Comput. 36 (1) (2006) 184–206.
[17] N. Halko, P.G. Martinsson, J.A. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, J. SIAM Rev. (2011) 217–288.
[18] C. Chih-Chung, L. Chih-Jen, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 1–27.
[19] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.

Chang-qian Men received his M.S. degree in Computer Application Technology from Shanxi University, China, in 2006, and Ph.D. degree in Systems Engineering from Shanxi University, China, in 2013. He has been working at Department of Computer Science in Shanxi University since 2006. His current research interests include support vector machines, machine learning theory, kernel methods and extreme learning machine.

Wen-jian Wang received her B.S. degree in Computer Application Technology from Shanxi University, China, in 1990, M.S. degree in Computer Science from Hebei Polytechnic University, China, in 1993, and Ph.D. degree in Applied Mathematics from Xi'an Jiao Tong University, China, in 2004. She worked as a research assistant at the Department of Building and Construction, in City University of Hong Kong from May 2001 to May 2002. She has been worked at Department of Computer Science in Shanxi University since 1993, where she was promoted to associate professor in 2000 and full-time professor in 2004, and now serves as a Ph.D. supervisor in Computer Science and Technology, Soft Engineering. She has published more than 70 academic papers on her research areas like machine learning, computational intelligence, and data mining. Her current research interests include support vector machines, machine learning theory, kernel methods, granular computing and environmental computations.