Research Article

Two-Dimensional Extreme Learning Machine

Bo Jia, Dong Li, Zhisong Pan, and Guyu Hu

College of Command Information System, PLA University of Science and Technology, Nanjing 210007, China

Correspondence should be addressed to Guyu Hu; huguyu@189.cn

Received 17 August 2014; Revised 5 November 2014; Accepted 6 November 2014

Academic Editor: Amaury Lendasse

Copyright © 2015 Bo Jia et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Extreme learning machine (ELM) has achieved wide attention due to faster learning speed compared with conventional neural network models like support vector machine (SVM) and back-propagation (BP) networks. However, like many other methods, ELM is originally proposed to handle vector pattern while nonvector patterns in real applications need to be explored, such as image data. We propose the two-dimensional extreme learning machine (2DELM) based on the very natural idea to deal with matrix data directly. Unlike original ELM which handles vectors, 2DELM takes matrices as input features without vectorization. Empirical studies on several real image datasets show the efficiency and effectiveness of the algorithm.

1. Introduction

Pattern representation is probably one of the basic problems in machine learning; almost all learning algorithms aim to build the mapping functions from the input to output. The output value of a learning model is always straightforward while different input representations could influence the results much. For statistical learning, the input pattern is commonly represented by a vector which contains the values belongs to corresponding features. Even though the original data is not sampled as vectors, there exists a standard preprocessing method named vectorization, which aims to transform the original data into vectors for the convenience of computation. Taking the face image, for example, each sample of a $d_1$-by-$d_2$ face image is always transformed into a $d_1 \times d_2$-length vector by concatenating all columns or rows, so that the sample can be processed by popular learning algorithms such as support vector machine (SVM) or artificial neural networks. Input vectors almost become another name for input samples, and some of them have discriminative ability which define the margin of largest separation are called support vectors in SVM [1].

On the one hand, vectorization helps the input data to fit in mature models as well as to accelerate computation procedure using popular linear algebra libraries. On the other hand, the drawbacks of vectorizing image data are obvious from at least two aspects [2, 3]. (1) Structural or contextual information may be lost during the transformation due to the changes of relative position of the pixels, and the reason is quite intuitive. (2) Vectorization needs more parameters and thus leads to the curse of dimensionality. For example, in order to classify $1024 \times 1024$ images by neural networks with 1000 hidden nodes, one need $10^9$ parameters in the first layer. The feedforward computation can be slow.

Now look at the general class of mapping function adopted by many discriminative models, which take the sample vector as input and classification label or regression value as the output:

$$ y = f\left(\sum_{i=1}^{m} \beta_i h_i(x)\right), \quad (1) $$

where $x \in \mathbb{R}^d$ is the input vector and $h_i(x)$ is the $i$th output value of the hidden layer in three-layer neural network, or the $i$th output value of other two-layer model such as least square regression and logistic regression. $\beta_i$ is the parameter vector which connects $h_i(x)$ and the final output value. In order to have a scalar output $y$ easily, a linear or nonlinear transformation needs to be conducted on the input space; thus $h_i(x)$ is sometimes regarded as point in the feature space. Function $f(\cdot)$ controls the final output value according to
Mathematical Problems in Engineering

specific learning tasks. The definition of feature mapping function is

\[ h_i(x) = g \left( w_i^T x + b_i \right), \]  

(2)

where \( w_i \in \mathbb{R}^d \) is the weight vector that connects the input nodes and the \( i \)th hidden node in neural network models and \( b_i \) is the bias of the \( i \)th hidden node in this case. \( g(\cdot) \) is probably a nonlinear continuous function. For linear regression models as well as back-propagation networks, the \( w_i \) are the main parameters that need to be learned. The feature mapping stage here is a linear transformation, and the output of each hidden node is a linear combination of input units and corresponding weights.

Similar to vector case, the feature mapping function for the matrix pattern \( A \in \mathbb{R}^{d_1 \times d_2} \) looks differently as the following form [4]:

\[ h_i(A) = g \left( u_i^T A v_i + b_i \right), \]  

(3)

where \( u_i \in \mathbb{R}^{d_1} \) and \( v_i \in \mathbb{R}^{d_2} \) are two weight vectors similar to \( w_i \) in the vector pattern. This might be the simplest way to transform a matrix into a scalar using vector inner product similar to (2), since matrix-vector product is essentially sum of several vector inner products.

We can see that there are only \( d_1 + d_2 \) parameters needed instead of \( d_1 \times d_2 \) in (2) for each hidden node. From this point, using matrix pattern could reduce model complexity with fewer parameters, even if the original sample is not matrix as long as the vector can be recombination into matrix. Take the single layer feedforward neural network (SLFN), for example, here, as Figure 1 shows the differences between two input patterns: (a) needs \( d_1 + d_2 \) nodes in the input layer while (b) just needs \( d_2 \) for the same input sample.

As opposed to vector case learning methods, two-dimensional methods have been used on feature extraction as well as conventional learning models, in the last decade. Yang et al. [5] proposed two-dimensional principle component analysis (2DPCA) for image representation, which turned out to be advantageous over PCA in several aspects. Ye et al. [6] proposed two-dimensional linear discriminant analysis (2DLDA) which works with data in matrix representation and could overcome the singularity problem in conventional LDA. Wang et al. [3] provided a fully matrixed approach, applied in both feature extraction and classifier design, including his previous work [4] which proposed MatLSSVM, that is, least squares support vector machines (LS-SVM) based on matrix patterns and its fuzzy version. Empirical studies in these literatures showed that two-dimensional methods helped to improve classification performance and reduce the computational and space complexity, compared with the base models.

A more general representation pattern over matrix is tensor, which takes \( A \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_m} \) as the input. Tao et al. [7] described a supervised tensor learning framework and the alternating projection optimization to obtain the solution. Conventional models like SVM and Fisher discriminant analysis were contained in this framework. Possible solutions of tensor based ELM will be discussed later.

Inspired by the very natural idea to let ELM process matrices directly and matrix pattern related works [3], we propose the two-dimensional extreme learning machine (2DELM) in this paper, and our main contributions can be summarized as follows:

(i) providing a simple method to process matrix pattern for SLFN;
(ii) analyzing the random feature mapping from a probabilistic perspective for both ELM and 2DELM;
(iii) comparing the proposed algorithm with original ELM on image datasets based on a statistical approach.

The remainder of this paper is organized as follows. Section 2 reviews the vector based ELM. Section 3 describes the 2DELM and related concepts, including a sparse version, kernels tricks, and tensor based ELM. We evaluate our methods on several image data in Section 4. Finally Section 5 concludes this paper.

2. Extreme Learning Machine: A Vector Case

Extreme learning machine [8] was proposed as an efficient learning algorithm for single hidden layer feedforward neural networks, which outperforms the gradient-based methods to learn the same architecture. The structure is also shown in Figure 1(a). According to a general principle for learning machine, that is, to minimize the empirical risk (ERM), ELM aims to reach the smallest training error by

\[ \min_{\beta} : \|H\beta - y\|^2, \]  

(4)

where \( H \) is defined as the hidden layer output matrix of \( N \) training samples, \( \beta \) is output weights vector that connect hidden layer and output layer (with \( L \) hidden nodes), and \( y \in \mathbb{R}^N \) is the target vector that contains real values for regression and class labels for classification:

\[ H = \begin{pmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{pmatrix} = \begin{pmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_N) & \cdots & h_L(x_N) \end{pmatrix}, \]  

(5)

where \( h_l(x_i) \) is the output of the \( i \)th hidden node of the \( j \)th input vector and probably has the same form as (2).

The significant characteristic of ELM lies at the random choice of the weights \( w \) that connect the input layer and hidden layer as well as the bias \( b \) of hidden layer, which is different from traditional algorithms like back-propagation where all parameters need to be tuned. This makes the hidden layer output matrix by hand and only the output weights \( \beta \) need to be learned. Under the ERM principle, the optimal solution \( \beta^* \) to (3) can be analytically resolved as

\[ \beta^* = H^T y, \]  

(6)

where \( H^T \) is the Moore-Penrose generalized inverse of matrix \( H \).
The idea that hidden node parameters need not to be learned has been extended to many other models beyond neural network models like SVM, RBF networks, and so forth [9]. The simplicity of ELM has been also extended to form a unified framework, which mainly takes three steps as follows.

1. Randomly choose parameters at first layer of SLFN for feature mapping.
2. Use various activation function to generate new feature representations.
3. Fast solution of required parameters at last layer of SLFN.

Kernel tricks such as in SVM could be used in ELM to obtain more powerful classification ability [10, 11]. The fast solution in step (3) makes online learning and real time prediction possible. The whole procedure is also suitable for many other models in ensemble learning; the weights of multiple predictors can be determined by a similar way in (5). To be more specific, the last layer of SLFN can be viewed as a linear combination of multiple weak predictors to form a strong predictor, which is consistent with the design of ensemble learning.

Because of above properties, ELM and its variants have been widely used in many areas like face recognition [12], object recognition [13], large scale data analysis [14], network security [15], and so forth. Almost all these application examples deal with vector pattern, even if the objects are images. It is necessary to extend ELM to matrix pattern, so that we can use it in a more generalized form in practise.

3. Two-Dimensional ELM

3.1. Basic Formulas. The goals of 2DELM are to process matrix pattern directly, instead of vectorizing by concatenating all columns or rows at first. At the feature mapping stage, which is corresponding to the input layer and the hidden layer in SLFN architecture, each hidden node encode all original features of a sample someway.

Assume activation function is sigmoid \( g(t) = \frac{1}{1 + \exp(-t)} \), vector based ELM takes a linear combination of all features as input of activation function \( t \), and the linear weights are randomly generated. Inspired by ELM, the first layer parameters which actually do feature mapping need not to be tuned in SLFN. In order to get random features in the hidden layer like ELM, we can randomly choose \( u_i \), \( v_i \), and \( b \) in (3), at the first layer in SLFN. As we mentioned, \( u^T \cdot A \cdot v \) might be the simplest way to transform a matrix \( A \) into a scalar using vector inner products. The entries in hidden layer output matrix \( (H)_{N \times L} \) of SLFN are formally defined as

\[
H_{ij} = g(\mathbf{u}_i^T \cdot A \cdot \mathbf{v}_j + b_i),
\]

where \( H_{ij} \) is the output of the \( i \)th hidden node of the \( j \)th input matrix sample. Each hidden node thus gets all entries’ information of matrix \( A_j \) while keeping various via different random weights \( u_i \), \( v_i \), and \( b \).

For a complete learning model, we have random parameters \( U \in \mathbb{R}^{d_1 \times L} \), \( V \in \mathbb{R}^{d_2 \times L} \), and bias \( b \in \mathbb{R}^L \). \( U \) and \( V \) in (7) are the \( i \)th row of \( U \) and \( V \), respectively. Having the hidden layer output matrix by hand, the next thing would be the same as ELM: to solve the optimal weights by (6). With \( L \) hidden nodes, we can see that \((d_1 + d_2) \times L \) input parameters are needed here while the number is \((d_1 \times d_2) \times L \) after vectorization. Conversely, we could also conduct reformattting a vector into a matrix to reduce the parameters as long as the length of the vector is not a prime.

In order to get a stable solution, ridge regression [16] could be applied to solve \( \beta \) for ELM [9, 17] as well as 2DELM. The corresponding objective function is defined as (8), and \( \lambda \) is the parameters to balance the loss and \( L_2 \) regularizer. This problem can be analytically solved by \( \beta = (H^T \cdot H + \lambda I)^{-1} \cdot H^T \cdot y \) where \( I \) is the identity matrix

\[
\min_\beta \| H \beta - y \|_2^2 + \lambda \| \beta \|_2^2.
\]

The whole procedure of MatELM is illustrated in Algorithm 1. As we can see, using the same amount of hidden
nodes, ELM and MatELM have the same training speed to compute $\beta$ since they share the size of $H$ and $y$. But in theory, the computational complexity to build $H$ is different: $O(d_1 \times d_2)$ for ELM and $O(d_1 + d_2)$ for 2DELM. In practice the speed to compute $H$ also depends on how the original data are stored: 2DELM tends to outperform ELM if samples are stored in matrices and vice versa.

3.2. Further Discussion. As mentioned in Section 1, tensor is a more general representation pattern over matrix pattern. The essence of learning model is to transform the input to considerable output, regarding tensor based ELM; we can extend it similar to 2DELM. The most important step lies in step (2) of Algorithm 1, that is, to compute the hidden layer output matrix $H$. For tensor pattern $A \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_n}$, we can define the entries in $H$ by

$$H_{ij} = g \left( \left( u_{(m-1)j}^T \left( u_{(m-2)j}^T \cdots (u_{ij}^T A) \right) \right) u_{mj} + b_j \right),$$

(9)

where the weights $u_{ij} \in \mathbb{R}^{d_i}$ are randomly chosen. Once the hidden layer output matrix $H$ is ready, the rest of training is the same as ELM. The sparse weight vector $\beta$ can be also obtained by $L_1$ norm regularizer in tensor based ELM as well.

Castano et al. [18] proposed PCA-ELM, a robust and pruned ELM based PCA, which aims to determine the hidden nodes in ELM with the information retrieved from principal components analysis of training data. PCA-ELM reduced the model parameters by taking low-dimension training data, which is different from our method. Explicit vectorization is needed in these methods; however the frameworks are not in contradiction with 2DELM since the latter focuses on the pattern representation. In other words, PCA related techniques can be combined with the idea of 2DELM in practice.

4. Experiments

In this section, we mainly compare 2DELM and ELM on image datasets for multiclass classification. Assume the number of classes is $C$; we transform the label vector $y$ into ground truth matrix $T_{N \times C}$ in both training and testing stage. The definition of entries of $T$ is

$$T_{ij} = \begin{cases} 1 & \text{when } A_j = i, \\ -1 & \text{otherwise}. \end{cases}$$

(10)

The primary solution of $\beta_{DSC}$ in all experiments is based on Moore-Penrose generalized inverse (we replace the $y$ by $T$ in (6)) since it needs only one user-defined parameter: the number of hidden nodes $L$. In practice, it is very time consuming to choose the parameter $\lambda$ when using ridge regression in a wide range; moreover, in many cases, Moore-Penrose generalized inverse solution tends to be stable as well. At predicting stage, for each tested sample $A$, we use (11) to get the output vector $t$, and then the index of largest entry of $t$ is taken as the label

$$t = \beta h(A),$$

(11)

where $h(A)$ is hidden layer output vector of sample $A$, sized by $L$-length, and each entry has the same form as (7).

4.1. Data Description. In order to show the effectiveness of 2DELM, we get several popular image datasets, and the application background covers face recognition and other image classifications. In specific, there are five face databases and four OCR datasets in the following discussion, the data size and dimensions vary in wide range.

(1) Yale database (http://www.antd.yju.edu.cn/home/dengcai/Data/FaceData.html): the Yale database contains 165 images from 15 persons, and each person has 11 images. These images were various from facial expressions and lighting conditions: center-light, happy, left-light, sad, and so forth. Each image has size of $32 \times 32$ or $64 \times 64$. Figure 2(a) shows five sample images from this database.

(2) Face database (http://www.antd.yju.edu.cn/home/dengcai/Data/FaceData.html): the ORL has 400 images belonging to 40 persons, and each person has 10 images. These images were taken from different times, and the lighting, facial expressions (open/closed eyes), and facial details (glasses or no glasses) are various. Moreover, all the images were taken in front of the same dark background with the individual in an upright position. Each image has size of $32 \times 32$ or $64 \times 64$. Figure 2(b) shows five sample images from this database.

(3) UMist faces (http://www.cs.nyu.edu/~roweis/data.html): there are 575 grayscale face images from 20 different people. These pictures were taken with the individuals having different side angle against the camera. Each image is $112 \times 92$ size and is manually cropped by Graham and Allinson at UMist [19]. Figure 2(c) shows five sample images from this database.

(4) Georgia tech face database (http://www.anefian.com/research/face_reco.htm): this database contains 750 images of 50 people, each person has 15. All people in the database are represented by 15 color JPEG images with cluttered background taken at resolution $640 \times 480$ pixels. The database also provides coordinates of the face rectangle. Here we only use the grayscale images for classification. Figure 2(d) shows five sample images from this database.
(5) PIE face: the CMU Pose, Illumination, and Expression (PIE) database, provided by [20]. There are 41,368 images of 68 people which were collected in 2000. These images were taken with each person under 13 different poses, 43 different illumination conditions, and with 4 different expressions. We get a sunset containing 11,554 images in our experiment, and each has the size 32 × 32. Figure 2(e) shows five sample images from this database.

(6) Letter, shuttle, and USPS datasets (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html) and MNIST Handwritten Digits (http://www.cs.nyu.edu/~roweis/data.html): these four OCR datasets were provided with separate training set and testing set. There are 7,291 training samples and 2,007 testing samples in USPS, and each has size 16 × 16. There are 15,000 training samples and 5,000 testing samples in letter, and each has size 4 × 4. There are 43,500 training samples and 14,500 testing samples in shuttle, and each has size 3 × 3. MNIST has 60,000 training samples and 10,000 testing samples, and each has size 28 × 28. Figure 2(f) shows five sample images from MNIST.

More details of these datasets are provided in Table 1. Similar to [4], we introduce the ration $R = (d_1 \times d_2)/(d_1 + d_2)$ to indicate the input parameters needed ration for vector pattern versus matrix pattern. The last column of these tables indicates whether the training data and the testing data are provided separately.

4.2. Experiment Settings. The simulations of ELM (http://www.ntu.edu.sg/home/egbhuang/elm_random_hidden_nodes.html) and 2DELM (https://github.com/fairmiracle/MatELM/) on all datasets are carried out in Matlab 2013a environment running in Intel Core i5 CPU, with 16 GB memory.
Table 1: Summary of the image datasets for multiclass classification, \( R = (d_1 \times d_2)/(d_1 + d_2) \). “Separate” indicates whether the training data and the testing data are provided separately.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># train</th>
<th># test</th>
<th># features</th>
<th># classes</th>
<th>( R )</th>
<th>Separate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yale,32 \times 32</td>
<td>110</td>
<td>55</td>
<td>1024</td>
<td>15</td>
<td>16</td>
<td>No</td>
</tr>
<tr>
<td>Yale,64 \times 64</td>
<td>110</td>
<td>55</td>
<td>4096</td>
<td>15</td>
<td>32</td>
<td>No</td>
</tr>
<tr>
<td>ORL,32 \times 32</td>
<td>266</td>
<td>134</td>
<td>1024</td>
<td>40</td>
<td>16</td>
<td>No</td>
</tr>
<tr>
<td>ORL,64 \times 64</td>
<td>266</td>
<td>134</td>
<td>4096</td>
<td>40</td>
<td>32</td>
<td>No</td>
</tr>
<tr>
<td>UMist</td>
<td>383</td>
<td>192</td>
<td>10304</td>
<td>20</td>
<td>50.5</td>
<td>No</td>
</tr>
<tr>
<td>GTface</td>
<td>500</td>
<td>250</td>
<td>307200</td>
<td>50</td>
<td>274.3</td>
<td>No</td>
</tr>
<tr>
<td>PIE</td>
<td>7703</td>
<td>3851</td>
<td>1024</td>
<td>68</td>
<td>16</td>
<td>No</td>
</tr>
<tr>
<td>Letter</td>
<td>15000</td>
<td>5000</td>
<td>16</td>
<td>26</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Shuttle</td>
<td>43500</td>
<td>14500</td>
<td>9</td>
<td>7</td>
<td>1.5</td>
<td>Yes</td>
</tr>
<tr>
<td>USPS</td>
<td>7291</td>
<td>2007</td>
<td>256</td>
<td>10</td>
<td>8</td>
<td>Yes</td>
</tr>
<tr>
<td>MNIST</td>
<td>60000</td>
<td>10000</td>
<td>784</td>
<td>10</td>
<td>14</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Fifty trials have been conducted for each problem when comparing ELM and 2DELM on all datasets. For the face datasets, which do not provide separate training set and testing set, we randomly choose the 2/3 of the total samples as training set and the rest as testing in each trial. For other datasets, we conduct the experiments with fifty different random initializations of ELM and 2DELM. The averaged training accuracy, testing accuracy, training time, and testing time of all trials are recorded, and the comparison of testing accuracy is based on pairwise \( t \)-tests at 95% significance level.

4.3. Comparison Results. We first compare the processing time of ELM and 2DELM when dealing with matrix pattern. Since most of the datasets were provided with vectors in Table 1, we find two image databases with samples stored in original pictures. We put the cell structure in Matlab which contains matrices as the elements as the input and count the cpu-time for each trial. Note that implementation by vectorization is accelerated by some linear algebra libraries; here we count tics for getting \( \mathbf{H} \) without vectorization in ELM due to the same conditions with 2DELM. The time needed by calculating hidden layer output matrix \( \mathbf{H} \) for ELM or 2DELM is also depend on the number of hidden nodes \( L \). Figure 3 shows the results when \( L \) is in the range \([100, 200, \ldots, 1000]\). With each \( L \), mean time and standard deviation of ten trails are shown on the figures. We can see that 2DELM achieves faster speed under the same implementation condition and tends to be more stable with random parameters.

We set the number of hidden nodes \( L = 1000 \) in both ELM and 2DELM in the following comparison. Table 2 shows the average training time and testing time. We can see that the training time (time needed for calculating \( \mathbf{H} \) with \( \mathbf{H} \) by hand) in ELM and 2DELM keeps roughly the same, due to the same size of \( \mathbf{H} \) they share, so as the testing time.

The accuracy comparison results are shown in Table 3. Bold number indicates better mean testing accuracy, and \( \circ \) indicates this advantage is significant under pairwise \( t \)-tests at 95% significance level (\( \circ \) otherwise). We can see that 2DELM achieves better testing accuracy than ELM in most cases, with the same amounts of hidden nodes and much less input parameters.

Table 2: Training time and testing time (in seconds) for ELM and 2DELM, average values on fifty trials.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training time</th>
<th>Testing time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ELM</td>
<td>2DELM</td>
</tr>
<tr>
<td>Yale,32 \times 32</td>
<td>0.1909</td>
<td>0.1919</td>
</tr>
<tr>
<td>Yale,64 \times 64</td>
<td>0.0655</td>
<td>0.0568</td>
</tr>
<tr>
<td>ORL,32 \times 32</td>
<td>0.1438</td>
<td>0.1304</td>
</tr>
<tr>
<td>ORL,64 \times 64</td>
<td>0.1460</td>
<td>0.1685</td>
</tr>
<tr>
<td>UMist</td>
<td>0.1987</td>
<td>0.2290</td>
</tr>
<tr>
<td>GTface</td>
<td>0.6146</td>
<td>0.6043</td>
</tr>
<tr>
<td>PIE</td>
<td>6.5056</td>
<td>6.5642</td>
</tr>
<tr>
<td>Letter</td>
<td>11.1809</td>
<td>11.0386</td>
</tr>
<tr>
<td>Shuttle</td>
<td>29.7740</td>
<td>28.0178</td>
</tr>
<tr>
<td>USPS</td>
<td>6.1127</td>
<td>6.1936</td>
</tr>
<tr>
<td>MNIST</td>
<td>42.2897</td>
<td>42.2875</td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper we propose a matrix pattern representation based ELM algorithm 2DELM, which take matrices as input instead of commonly used vectors in the SLFN. The key difference between 2DELM and ELM lies at the feature mapping stage; vectorization is not needed when dealing with matrices and thus reduces the input weights compared with vector pattern case. The learning stage keeps the same as ELM and inherits most characteristics of ELM. The comparing experiments on several image datasets show the effectiveness of the proposed algorithms. In most cases, 2D could achieve better or comparable testing accuracy as ELM while using fewer input weights parameters.

From ELM to 2DELM, we aim to simplify the learning model by reducing parameters while keeping the predicting accuracy under the basic ELM framework. The method also keeps consistent with the general principle of Occam's razor [21] in classifier design. What is more, for dealing with high dimensional data, the matrix or tensor pattern representation may provide another perspective besides traditional dimension reducing techniques.
Figure 3: The time needed for getting hidden layer output matrix $H$ on UMist data and Georgia tech face database.

Table 3: Mean accuracy ($\pm$ standard deviation) of fifty-trial comparison of ELM and 2DELM. Testing accuracy is also compared based on pairwise $t$-tests at 95% significance level.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ELM Training</th>
<th>ELM Testing</th>
<th>2DELM Training</th>
<th>2DELM Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yale, $32 \times 32$</td>
<td>1.0000 ± 0.0000</td>
<td>0.7578 ± 0.0585</td>
<td>1.0000 ± 0.0000</td>
<td>0.7880 ± 0.0524*</td>
</tr>
<tr>
<td>Yale, $64 \times 64$</td>
<td>1.0000 ± 0.0000</td>
<td>0.8473 ± 0.0454</td>
<td>1.0000 ± 0.0000</td>
<td>0.8720 ± 0.0346*</td>
</tr>
<tr>
<td>ORL, $32 \times 32$</td>
<td>1.0000 ± 0.0000</td>
<td>0.9333 ± 0.0233</td>
<td>1.0000 ± 0.0000</td>
<td>0.9416 ± 0.0244*</td>
</tr>
<tr>
<td>ORL, $64 \times 64$</td>
<td>1.0000 ± 0.0000</td>
<td>0.9251 ± 0.0258</td>
<td>1.0000 ± 0.0000</td>
<td>0.9378 ± 0.0275*</td>
</tr>
<tr>
<td>UMist</td>
<td>1.0000 ± 0.0000</td>
<td>0.9649 ± 0.0139</td>
<td>1.0000 ± 0.0000</td>
<td>0.9749 ± 0.0128*</td>
</tr>
<tr>
<td>GTFace</td>
<td>1.0000 ± 0.0000</td>
<td>0.9752 ± 0.0150*</td>
<td>1.0000 ± 0.0000</td>
<td>0.9746 ± 0.0151</td>
</tr>
<tr>
<td>PIE</td>
<td>0.9766 ± 0.0016</td>
<td>0.9314 ± 0.0047</td>
<td>0.9888 ± 0.0009</td>
<td>0.9586 ± 0.0028*</td>
</tr>
<tr>
<td>Letter</td>
<td>0.9543 ± 0.0011</td>
<td>$\textbf{0.9366} \pm 0.0020^*$</td>
<td>0.9489 ± 0.0009</td>
<td>0.9311 ± 0.0018</td>
</tr>
<tr>
<td>Shuttle</td>
<td>0.9995 ± 0.0000</td>
<td>$\textbf{0.9977} \pm 0.0001^*$</td>
<td>0.9989 ± 0.0001</td>
<td>0.9970 ± 0.0001</td>
</tr>
<tr>
<td>USPS</td>
<td>0.9990 ± 0.0006</td>
<td>0.9369 ± 0.0027</td>
<td>0.9918 ± 0.0007</td>
<td>$\textbf{0.9395} \pm 0.0033^*$</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.9378 ± 0.0010</td>
<td>0.9357 ± 0.0016</td>
<td>0.9407 ± 0.0010</td>
<td>$\textbf{0.9403} \pm 0.0016^*$</td>
</tr>
</tbody>
</table>

* or □ indicates whether the advantage is significant or not.

Conflict of Interests
The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments
This research was supported by the National Technology Research and Development Program of China (863 Program) 2012AA01A510. The authors would also like to thank the anonymous reviewers for their patient work and suggestions to improve the paper.

References