Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 491587, 8 pages
http://dx.doi.org/10.1155/2015/491587

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

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 take the 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 1024×1024 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:

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

where $\mathbf{x} \in \mathbb{R}^d$ is the input vector and $h_i(\mathbf{x})$ is the ith output value of the hidden layer in three-layer neural network, or the ith output value of other two-layer model such as least square regression and logistic regression. β_i is the parameter vector which connects $h_i(\mathbf{x})$ 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 $h_i(\mathbf{x})$ is sometimes regarded as point in the feature space. Function $f(\cdot)$ controls the final output value according to

specific learning tasks. The definition of feature mapping function is

$$h_i(\mathbf{x}) = g\left(\mathbf{w}_i^T \mathbf{x} + \mathbf{b}_i\right), \tag{2}$$

where $\mathbf{w}_i \in \mathbb{R}^d$ is the weight vector that connects the input nodes and the ith hidden node in neural network models and $\mathbf{b}_i$ is the bias of the ith hidden node in this case. $g(\cdot)$ is probably a nonlinear continuous function. For linear regression models as well as back-propagation networks, the $\mathbf{w}_i$ 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]:

$$h_i(A) = g\left(\mathbf{u}_i^T A \mathbf{v}_i + \mathbf{b}_i\right), \tag{3}$$

where $\mathbf{u}_i \in \mathbb{R}^{d_1}$ and $\mathbf{v}_i \in \mathbb{R}^{d_2}$ are two weight vectors similar to $\mathbf{w}_i$ 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 $A \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_m}$ 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

$$\min_{\boldsymbol{\beta}} : \left\|\mathbf{H}\boldsymbol{\beta} - \mathbf{y}\right\|_2^2, \tag{4}$$

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

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} h_1(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_N) & \cdots & h_L(\mathbf{x}_N) \end{pmatrix}, \tag{5}$$

where $h_i(\mathbf{x}_j)$ is the output of the ith hidden node of the jth input vector and probably has the same form as (2).

The significant characteristic of ELM lies at the random choice of the weights $\mathbf{w}$ that connect the input layer and hidden layer as well as the bias $\mathbf{b}$ 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 $\boldsymbol{\beta}$ need to be learned. Under the ERM principle, the optimal solution $\boldsymbol{\beta}^*$ to (3) can be analytically resolved as

$$\boldsymbol{\beta}^* = \mathbf{H}^\dagger \mathbf{y}, \tag{6}$$

where $\mathbf{H}^\dagger$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

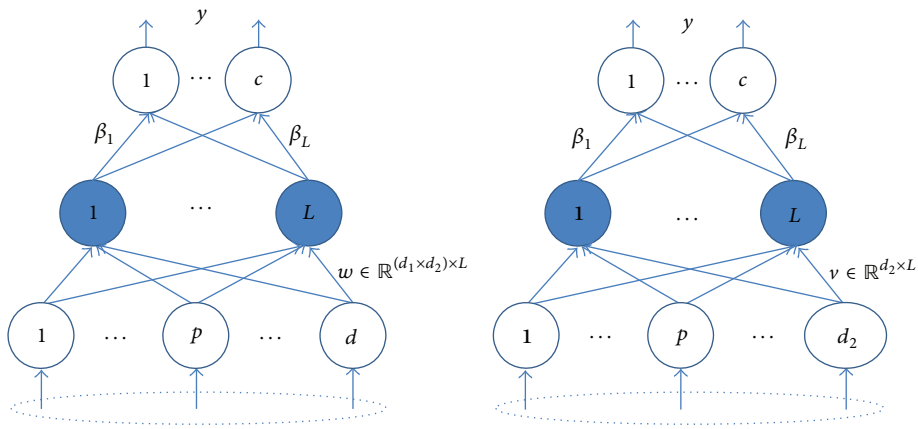

FIGURE 1: Two cases for single layer feedforward neural network on matrix pattern. (a) needs $d_1 \times d_2$ nodes in the first layer while (b) just needs d_2. reshape$(\cdot)$ here is the vectorization process.

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 $g(t) = 1/(1 + \exp(-t))$, 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 $\mathbf{u}_i$, $\mathbf{v}_i$, and $\mathbf{b}_i$ in (3), at the first layer in SLFN. As we mentioned, $\mathbf{u}^T A \mathbf{v}$ might be the simplest way to transform a matrix A into a scalar using vector inner products. The entries in hidden layer output matrix $(\mathbf{H})_{N \times L}$ of SLFN are formally defined as

$$\mathbf{H}_{ij} = g\left(\mathbf{u}_i^T A_j \mathbf{v}_i + \mathbf{b}_i\right), \tag{7}$$

where $\mathbf{H}_{ij}$ is the output of the ith hidden node of the jth input matrix sample. Each hidden node thus gets all entries' information of matrix A_j while keeping various via different random weights $\mathbf{u}_i$, $\mathbf{v}_i$, and $\mathbf{b}_i$.

For a complete learning model, we have random parameters $\mathbf{U} \in \mathbb{R}^{d_1 \times L}$, $\mathbf{V} \in \mathbb{R}^{d_2 \times L}$, and bias $\mathbf{b} \in \mathbb{R}^L$. $\mathbf{u}_i$ and $\mathbf{v}_i$ in (7) are the ith row of $\mathbf{U}$ and $\mathbf{V}$, 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 reformatting 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 $\boldsymbol{\beta}$ for ELM [9, 17] as well as 2DELM. The corresponding objective function is defined as (8), and λ is the parameters to balance the loss and L_2 regularizer. This problem can be analytically solved by $\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{H} + \lambda \mathbf{I})^{-1}\mathbf{H}^T\mathbf{y}$ where $\mathbf{I}$ is the identity matrix

$$\min_{\boldsymbol{\beta}} : \left\|\mathbf{H}\boldsymbol{\beta} - \mathbf{y}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_2^2. \tag{8}$$

The whole procedure of MatELM is illustrated in Algorithm 1. As we can see, using the same amount of hidden

Input: Training samples $\{(A_i, y_i)\}, i = 1, \ldots, N$
Samples $A_i \in \mathbb{R}^{d_1 \times d_2}$, labels $y_i \in \{1, \ldots, C\}$
Output: Model parameters $\boldsymbol{\beta}$.
(0) Determine the network architecture with d_2 input nodes, L hidden nodes and C output nodes;
(1) Randomly choose input parameters $\mathbf{U}$, $\mathbf{V}$ and bias $\mathbf{b}$;
(2) Compute the hidden layer output matrix $\mathbf{H}$ as in (7);
(3) Solve output parameters $\boldsymbol{\beta}$ by (6) or (8)

Algorithm 1: 2DELM.

nodes, ELM and MatELM have the same training speed to compute $\boldsymbol{\beta}$ since they share the size of $\mathbf{H}$ and $\mathbf{y}$. But in theory, the computational complexity to build $\mathbf{H}$ is different: $O(d_1 \times d_2)$ for ELM and $O(d_1 + d_2)$ for 2DELM. In practice the speed to compute $\mathbf{H}$ 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 import step lies in step (2) of Algorithm 1, that is, to compute the hidden layer output matrix $\mathbf{H}$. For tensor pattern $A \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_m}$, we can define the entries in $\mathbf{H}$ by

$$\mathbf{H}_{ij} = g\left(\left(\mathbf{u}_{(m-1)i}^T \left(\mathbf{u}_{(m-2)i}^T \cdots \left(\mathbf{u}_{1i}^T A_j\right)\right)\right)\mathbf{u}_{mi} + \mathbf{b}_i\right), \quad (9)$$

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

Castaño 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

$$T_{ij} = \begin{cases} 1 & \text{when } A_i \in j, \\ -1 & \text{otherwise.} \end{cases} \quad (10)$$

The primary solution of $\boldsymbol{\beta}_{L \times C}$ 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 λ 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 $\mathbf{t}$, and then the index of largest entry of $\mathbf{t}$ is taken as the label

$$\mathbf{t} = \boldsymbol{\beta}\mathbf{h}(A), \quad (11)$$

where $\mathbf{h}(A)$ 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.cad.zju.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 32×32 or 64×64. Figure 2(a) shows five sample images from this database.

(2) Face database (http://www.cad.zju.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 32×32 or 64×64. 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 112×92 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 640×480 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.

(a) Yale face samples

(b) ORL face samples

(c) Umist face samples

(d) Georgia tech face samples

(e) PIE face samples

(f) MNIST samples

Figure 2: The samples from face database in the experiments.

(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.

Dataset	# train	# test	# features	# classes	R	Separate
Yale_32 × 32	110	55	1024	15	16	No
Yale_64 × 64	110	55	4096	15	32	No
ORL_32 × 32	266	134	1024	40	16	No
ORL_64 × 64	266	134	4096	40	32	No
UMist	383	192	10304	20	50.5	No
GTface	500	250	307200	50	274.3	No
PIE	7703	3851	1024	68	16	No
Letter	15000	5000	16	26	2	Yes
Shuttle	43500	14500	9	7	1.5	Yes
USPS	7291	2007	256	10	8	Yes
MNIST	60000	10000	784	10	14	Yes

Fifty trials have been conducted for each problem when comparing ELM and 2DEM 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 trail. 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 $\boldsymbol{\beta}$ 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 • indicates this advantage is significant under pairwise t-tests at 95% significance level (∘ 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 trails.

Dataset	Training time		Testing time	
	ELM	2DELM	ELM	2DELM
Yale_32 × 32	**0.1909**	**0.1919**	0.0009	0.0022
Yale_64 × 64	0.0655	**0.0568**	0.0053	**0.0003**
ORL_32 × 32	0.1438	**0.1304**	**0.0069**	0.0075
ORL_64 × 64	**0.1460**	0.1685	**0.0069**	0.0081
UMist	**0.1987**	0.2290	0.0078	**0.0031**
GTface	0.6146	**0.6043**	0.0197	**0.0122**
PIE	**6.5056**	6.5642	**0.1532**	0.1554
Letter	11.1809	**11.0386**	0.1872	**0.1822**
Shuttle	29.7740	**28.0178**	0.4824	**0.4671**
USPS	**6.1127**	6.1936	**0.0699**	0.0736
MNIST	42.2897	**42.2875**	**0.3716**	0.3719

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.

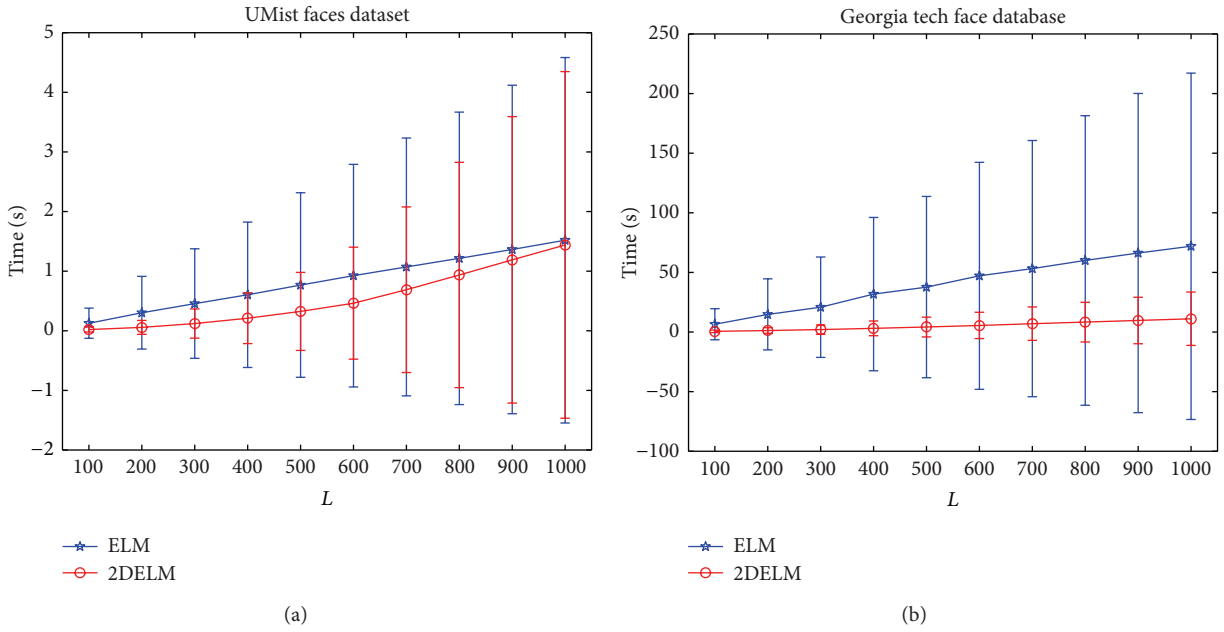

Figure 3: The time needed for getting hidden layer output matrix **H** on UMist data and Georgia tech face database.

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

Dataset	ELM		2DELM	
	Training	Testing	Training	Testing
Yale_32 × 32	1.0000 ± 0.0000	0.7578 ± 0.0585	1.0000 ± 0.0000	**0.7880** ± 0.0524•
Yale_64 × 64	1.0000 ± 0.0000	0.8473 ± 0.0454	1.0000 ± 0.0000	**0.8720** ± 0.0346•
ORL_32 × 32	1.0000 ± 0.0000	0.9333 ± 0.0233	1.0000 ± 0.0000	**0.9416** ± 0.0244•
ORL_64 × 64	1.0000 ± 0.0000	0.9251 ± 0.0258	1.0000 ± 0.0000	**0.9378** ± 0.0275•
UMist	1.0000 ± 0.0000	0.9649 ± 0.0139	1.0000 ± 0.0000	**0.9749** ± 0.0128•
GTface	1.0000 ± 0.0000	**0.9752** ± 0.0150◦	1.0000 ± 0.0000	0.9746 ± 0.0151
PIE	0.9766 ± 0.0016	0.9314 ± 0.0047	0.9888 ± 0.0009	**0.9586** ± 0.0028•
Letter	0.9543 ± 0.0011	**0.9366** ± 0.0020•	0.9489 ± 0.0009	0.9311 ± 0.0018
Shuttle	0.9995 ± 0.0000	**0.9977** ± 0.0001•	0.9989 ± 0.0001	0.9970 ± 0.0001
USPS	0.9910 ± 0.0006	0.9369 ± 0.0027	0.9918 ± 0.0007	**0.9395** ± 0.0033•
MNIST	0.9378 ± 0.0010	0.9357 ± 0.0016	0.9407 ± 0.0010	**0.9403** ± 0.0016•

• 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

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] H. Kong, L. Wang, E. K. Teoh, X. Li, J.-G. Wang, and R. Venkateswarlu, "Generalized 2D principal component analysis for face image representation and recognition," *Neural Networks*, vol. 18, no. 5-6, pp. 585–594, 2005.

[3] Z. Wang, S. Chen, J. Liu, and D. Zhang, "Pattern representation in feature extraction and classifier design: matrix versus vector," *IEEE Transactions on Neural Networks*, vol. 19, no. 5, pp. 758–769, 2008.

[4] Z. Wang and S. Chen, "New least squares support vector machines based on matrix patterns," *Neural Processing Letters*, vol. 26, no. 1, pp. 41–56, 2007.

[5] J. Yang, D. Zhang, A. F. Frangi, and J.-Y. Yang, "Two-dimensional PCA: a new approach to appearance-based face representation and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 1, pp. 131–137, 2004.

[6] J. Ye, R. Janardan, Q. Li et al., "Two-dimensional linear discriminant analysis," in *Proceedings of the Neural Information Processing Systems Conference (NIPS '04)*, vol. 4, p. 4, 2004.

[7] D. Tao, X. Li, X. Wu, W. Hu, and S. J. Maybank, "Supervised tensor learning," *Knowledge and Information Systems*, vol. 13, no. 1, pp. 1–42, 2007.

[8] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[9] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, vol. 42, no. 2, pp. 513–529, 2012.

[10] Q. Liu, Q. He, and Z. Shi, "Extreme support vector machine classifier," in *Advances in Knowledge Discovery and Data Mining*, pp. 222–233, Springer, New York, NY, USA, 2008.

[11] B. Frénay and M. Verleysen, "Using SVMs with randomised feature spaces: an extreme learning approach," in *Proceedings of the 18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN '10)*, pp. 315–320, April 2010.

[12] K. Choi, K.-A. Toh, and H. Byun, "Incremental face recognition for large-scale social network services," *Pattern Recognition*, vol. 45, no. 8, pp. 2868–2883, 2012.

[13] J. Cao, T. Chen, and J. Fan, "Fast online learning algorithm for landmark recognition based on bow framework," in *Proceedings of the 9th IEEE Conference on Industrial Electronics and Applications*, pp. 1163–1168, 2014.

[14] D. Li, Z. Pan, Z. Deng, and Y. Zhang, "Large scale extreme learning machine using MapReduce," *International Journal of Digital Content Technology and Its Applications*, vol. 6, no. 20, pp. 62–70, 2012.

[15] Y. Wang, D. Li, Y. Du, and Z. Pan, "Anomaly detection in traffic using L1-norm minimization extreme learning machine," *Neurocomputing*, vol. 149, pp. 415–425, 2015.

[16] A. E. Hoerl and R. W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[17] W. Deng, Q. Zheng, and L. Chen, "Regularized extreme learning machine," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM '09)*, pp. 389–395, April 2009.

[18] A. Castaño, F. Fernández-Navarro, and C. Hervás-Martínez, "PCA-ELM: a robust and pruned extreme learning machine approach based on principal component analysis," *Neural Processing Letters*, vol. 37, no. 3, pp. 377–392, 2013.

[19] D. B. Graham and N. M. Allinson, "Face recognition: from theory to applications," *NATO ASI Series F: Computer and Systems Sciences*, vol. 163, pp. 446–456, 1998.

[20] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 12, pp. 1615–1618, 2003.

[21] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Wiley & Sons, New York, NY, USA, 2012.