Neurocomputing 149 (2015) 308–315

Contents lists available at ScienceDirect

Neurocomputing

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

Learning deep representations via extreme learning machines

Wenchao Yu [a,b], Fuzhen Zhuang [a,b], Qing He [a], Zhongzhi Shi [a]

[a] The Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[b] University of Chinese Academy of Sciences, Beijing 100049, China

ARTICLE INFO

Article history:
Received 12 August 2013
Received in revised form
1 February 2014
Accepted 29 March 2014
Available online 8 September 2014

Keywords:
Extreme learning machine
Deep learning
Representation learning
Stacked ELMs
Stacked generalization
DrELM

ABSTRACT

Extreme learning machine (ELM) as an emerging technology has achieved exceptional performance in large-scale settings, and is well suited to binary and multi-class classification, as well as regression tasks. However, existing ELM and its variants predominantly employ single hidden layer feedforward networks, leaving the popular and potentially powerful *stacked generalization* principle unexploited for seeking predictive deep representations of input data. Deep architectures can find higher-level representations, thus can potentially capture relevant higher-level abstractions. But most of current deep learning methods require solving a difficult and non-convex optimization problem. In this paper, we propose a stacked model, DrELM, to learn deep representations via extreme learning machine according to stacked generalization philosophy. The proposed model utilizes ELM as a base building block and incorporates random shift and kernelization as stacking elements. Specifically, in each layer, DrELM integrates a random projection of the predictions obtained by ELM into the original feature, and then applies kernel functions to generate the resultant feature. To verify the classification and regression performance of DrELM, we conduct the experiments on both synthetic and real-world data sets. The experimental results show that DrELM outperforms ELM and kernel ELMs, which appear to demonstrate that DrELM could yield predictive features that are suitable for prediction tasks. The performances of the deep models (i.e. Stacked Auto-encoder) are comparable. However, due to the utilization of ELM, DrELM is easier to learn and faster in testing.

1. Introduction

Extreme learning machine (ELM) [1], in virtue of its ease of use, fast learning speed and exceptional performance, has been extensively studied in recent years [2–6]. It has shown that ELM and its variants perform well in regression as well as in large scale classification applications, and a unified learning framework of least square support vector machine, proximal support vector machine, and other regularization algorithms referred to extreme learning machine can be built [3]. In many areas of machine learning, one is often confronted with overwhelmingly complex features in the raw data and needs to obtain useful representations of the data. Specifically, we need to compute a "function" that can accurately capture the structure of the inputs. Recently, deep learning and representation learning attract many research interests with its remarkable success in many applications [7–11]. In these works, deep nonlinear network structure was learnt to achieve a more complex function approximation, thus help to disentangle the underlying factors of variation in which raw data holds [12]. Deep architectures can potentially capture relevant higher-level abstractions and characterize the data representations more accurately.

Motivated by the remarkable success of deep learning [8,9], we propose a new stacked architecture, deep representations learning via extreme learning machine (DrELM for short), which incorporates the simplicity of ELM with the power derived from deep architectures. DrELM follows the philosophy of "stacked generalization" [13], a scheme of building layer-by-layer architectures for estimating the errors of prediction functions when working on a particular learning set, and then correcting those errors [13,14]. Specifically, the new stacking technique utilizes ELM as the base building block. In each layer, DrELM integrates a random projection of the predictions obtained by ELM into the inputs, and then apply kernel functions to generate the outputs for the next layer as shown in Fig. 1. This strategy could be seen as transforming the original feature according to the predictions of each layer so that the original manifold could be disentangled, leading to better predictions for ELM in the subsequent layers.

In particular, using ELM in the proposed model enables easier to learn and faster in testing. The model randomly generates projection parameters W_{1i} and W_{2i}, instead of fine-tuning using back-propagation [7,8], suffices to achieve a significant efficiency improvement during both the training and testing phases. To show the effectiveness of DrELM, we apply it to both the synthetic and real-world data sets. The experimental results show that the proposed model, while keeping the simplicity and efficiency of ELM, can exploit non-linear dependencies with the stacked architecture. In addition, DrELM outperforms ELM and kernel ELMs, and the performances of the deep models are comparable. However, DrELM enables faster computation.

http://dx.doi.org/10.1016/j.neucom.2014.03.077
0925-2312/

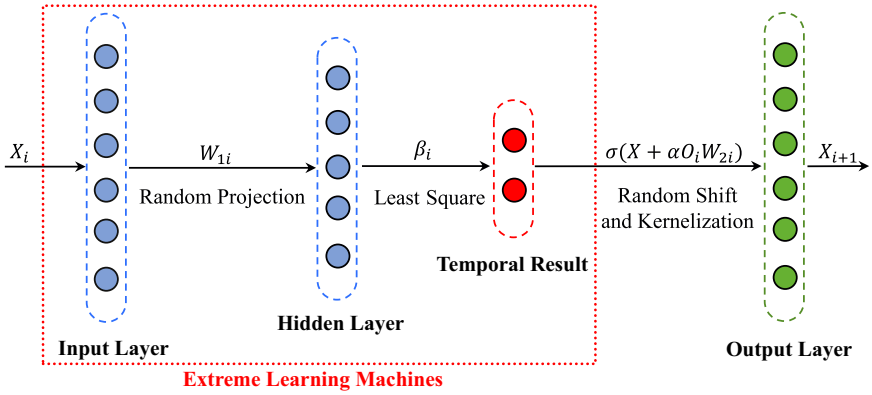

Fig. 1. Illustration of ith block building paradigm of DrELM. We first random project the input data $\boldsymbol{X}_i$ into hidden layer, and then solve the least square problem $\|\boldsymbol{W}_{1i}\boldsymbol{X}_i\boldsymbol{\beta}_i-\boldsymbol{T}\|$ to learn the weight $\boldsymbol{\beta}_i$ ($\boldsymbol{T}$ is the target value matrix corresponding to the input). We random shift the original feature based on the classification or regression results $\boldsymbol{O}_i$ and apply kernel functions $\sigma(\cdot)$ to avoid a trivial linear model.

In this paper, we describe DrELM, a stacked architecture which uses ELM as the base building block, along with the details of implementation and performance on simple synthetic examples and real-world data sets. The remainder of this paper is structured as follows: Section 2 reviews the preliminary knowledge of extreme learning machine. Section 3 describes DrELM, the proposed model which learns deep representations with extreme learning machine as a base building block. Section 4 presents empirical studies of DrELM. Section 5 compares DrELM to other deep architectures and discusses several related works. Finally, Section 6 concludes this study and mentions several directions for future work.

2. Brief review of extreme learning machine

Extreme learning machine (ELM) was originally derived from the single hidden layer feedforward neural networks (SLFNs) [1] and then extended to the generalized SLFNs. One of the typical implementations of ELM is to generate the weight matrix randomly between the input layer and the hidden layer, and then calculate the output weights by the least-square method. Different from traditional algorithms [15], ELMs tend to reach not only the smallest training error but also the smallest norm of output weights. ELM can be written as

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

where

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{h}(\boldsymbol{x}_1)\\ \vdots\\ \boldsymbol{h}(\boldsymbol{x}_m)\end{bmatrix}=\begin{bmatrix}G(\boldsymbol{a}_1,b_1,\boldsymbol{x}_1) & \cdots & G(\boldsymbol{a}_L,b_L,\boldsymbol{x}_1)\\ \vdots & \ddots & \vdots\\ G(\boldsymbol{a}_1,b_1,\boldsymbol{x}_m) & \cdots & G(\boldsymbol{a}_L,b_L,\boldsymbol{x}_m)\end{bmatrix} \tag{2}$$

$$\boldsymbol{\beta}=\begin{bmatrix}\beta_1^T\\ \vdots\\ \beta_L^T\end{bmatrix}_{L\times c} \quad \text{and} \quad \boldsymbol{T}=\begin{bmatrix}\boldsymbol{t}_1^T\\ \vdots\\ \boldsymbol{t}_m^T\end{bmatrix}_{m\times c} \tag{3}$$

$\boldsymbol{H}$ is the hidden layer output matrix, $(\boldsymbol{x}_i,\boldsymbol{t}_i)\in\mathcal{R}^d\times\mathcal{R}^c$ refers to arbitrary distinct samples, and $G(\boldsymbol{a}_i,b_i,\boldsymbol{x})$ is the hidden layer feature mapping function of the ith hidden node. If $\boldsymbol{H}$ is a nonsquare matrix and the smallest norm least-square solution of the above linear system is

$$\widehat{\boldsymbol{\beta}}=\boldsymbol{H}^{\dagger}\boldsymbol{T} \tag{4}$$

where $\boldsymbol{H}^{\dagger}$ is the Moore–Penrose generalized inverse of matrix $\boldsymbol{H}$. Thus, ELM can be summarized in Algorithm 1.

Algorithm 1. Basic ELM.

Input: A training set $\boldsymbol{X}=\{(\boldsymbol{x}_i,\boldsymbol{t}_i)|\boldsymbol{x}_i\in\mathcal{R}^d,\boldsymbol{t}_i\in\mathcal{R}^c,\ i=1,2,\ldots,m\}$, hidden node number L, hidden node transfer function $G(\cdot),j=1,2,\ldots,L$.

Output: The prediction function of ELM for classification and regression.

(1) Randomly generate hidden node parameters matrix $\boldsymbol{W}\in\mathcal{R}^{d\times L}$;

(2) Compute the hidden layer output matrix $\boldsymbol{H}=G(\boldsymbol{W},\boldsymbol{X})$;

(3) Compute output weight vector $\widehat{\boldsymbol{\beta}}=\boldsymbol{H}^{\dagger}\boldsymbol{T}$;

(4) Compute the result decision function:

$f(\boldsymbol{x})=\boldsymbol{h}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}=\boldsymbol{h}(\boldsymbol{x})\boldsymbol{H}^{\dagger}\boldsymbol{T}$.

3. Learning deep representations via ELM

Now we present a new deep architecture, DrELM, to learn deep representations via ELM. DrELM is a stacked model consisting of multiple layers of ELM in which the outputs of each layer are wired to the inputs of the successive layer. It explicitly employs ELM as a base building block and trains layer by layer in a feedforward way.

3.1. DrELM for multi-class classification and regression

We would like to use the idea of stacking technique to learn deep representations with ELM. The general approach we propose for DrELM is to compute the prediction results using *ELM with linear kernel*, then we integrate a random projection of the predictions into the original feature, and apply kernel functions to generate the resultant feature. As shown in Fig. 1, we stack each ELM block and bridge them by randomly shift and kernelize the original feature based on prediction results.

Formally, we consider a training set that contains m pairs of tuples $(\boldsymbol{x}^{(i)},\boldsymbol{t}^{(i)})$, where $\boldsymbol{x}^{(i)}\in\mathcal{R}^d$ is the feature vector, and $\boldsymbol{t}^{(i)}$ is the numerical value or class label corresponding to the input $\boldsymbol{x}^{(i)}$. We define a new matrix $\boldsymbol{X}$ for the entire training set as the concatenation of the m training samples, and $\boldsymbol{T}$ for the class label or real-value corresponding to the training samples. The ELM classifier is learned through Algorithm 1 with linear kernel, and we denote $\boldsymbol{O}^{(i)}$ for the classification or regression result corresponding to the sample $\boldsymbol{x}^{(i)}$. For the notation convenience, we drop the index and denote $\boldsymbol{X}$ for inputs. Starting from $\boldsymbol{X}_1=\boldsymbol{X}$ for the first layer as our initial input, we feed it to a linear ELM that gives the output $\boldsymbol{O}_1\in\mathcal{R}^{m\times c}$. In general, $\boldsymbol{O}_1$ is a linear classification or regression

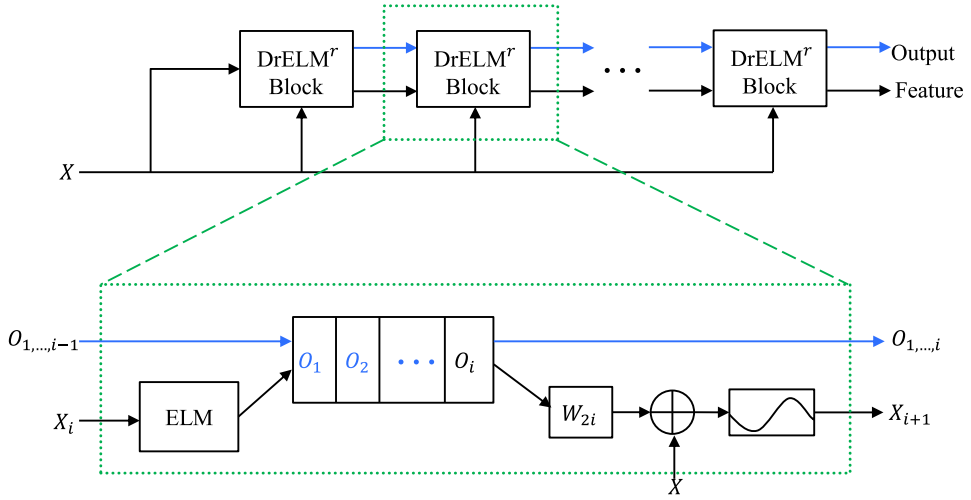

Fig. 2. Illustration of DrELM^r paradigm. In recursive version, $\boldsymbol{X}_{i+1}$ updates with all the prediction results in the previous layers $\boldsymbol{O}_1, \boldsymbol{O}_2, \ldots, \boldsymbol{O}_i$.

result and it would not be very precise, but it would be better than a random guess. We then random shift the original feature $\boldsymbol{X}$ and apply kernel functions to obtain new feature $\boldsymbol{X}_2$. In this way, we can obtain $\boldsymbol{X}_{i+1}$ via the following equation

$$\boldsymbol{X}_{i+1} = \sigma(\boldsymbol{X} + \alpha \boldsymbol{O}_i \boldsymbol{W}_i) \tag{5}$$

The kernel function $\sigma(\cdot)$ can be any kernel function such as sigmoid function and radial basis function; the projection matrix $\boldsymbol{W}_i \in R^{c \times d}$ whose elements are sampled from normal distribution $N(0,1)$; α is a weight parameter that controls the degree with which we shift the original data sample $\boldsymbol{x}$.

Algorithm 2. DrELM.

Input: The training set $\boldsymbol{X} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}]^T$ and $\boldsymbol{T} = [\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, \ldots, \boldsymbol{t}^{(m)}]^T$ corresponding to each training instance, $\boldsymbol{x}^{(i)} \in \mathcal{R}^d, \boldsymbol{t}^{(i)} \in \mathcal{R}^c$. The hidden node number L.
Output: The prediction function of DrELM for regression or classification.
(1) Choose model depth k;
(2) Initial $\boldsymbol{X}_1 = \boldsymbol{X}$;
(3) **for** i from 1 to k
(4) Randomly generate hidden node parameters matrix $\boldsymbol{W}_{1i} \in \mathcal{R}^{d \times L}$;
(5) Compute hidden layer output $\boldsymbol{H}_i = \boldsymbol{X}_i \boldsymbol{W}_{1i}$;
(6) Compute output weight vector $\widehat{\boldsymbol{\beta}}_i = \boldsymbol{H}_i^{\dagger} \boldsymbol{T}$;
(7) Compute the classification or regression results $\boldsymbol{O}_i = \boldsymbol{H}_i \widehat{\boldsymbol{\beta}}_i$;
(8) Randomly generate projection weight $\boldsymbol{W}_{2i} \in \mathcal{R}^{c \times d}$;
(9) Compute $\boldsymbol{X}_{i+1}$ by Eq. (5): $\boldsymbol{X}_{i+1} = \sigma(\boldsymbol{X} + \alpha \boldsymbol{O}_i \boldsymbol{W}_{2i})$;
(10) **end**
(11) Compute the prediction function $f_k(\boldsymbol{X}) = \boldsymbol{X}_k \boldsymbol{W}_{1k} \widehat{\boldsymbol{\beta}}_k$.

The central motivation of DrELM is that higher-level representations can potentially capture relevant higher-level abstractions, when well trained, tend to do a better job at disentangling the underlying factors of variation [12]. Random projections of previous predictions help to move apart the manifolds in a stacked fashion, in order to achieve better linear separability [9]. The utilization of kernel functions can avoid a trivial linear model. We develop a layer-by-layer learning algorithm to train DrELM. The algorithm is outlined in Algorithm 2.

Inspired by learning with recursive perceptual representations [9], we can design a recursive version of DrELM (DrELM^r) as shown in Fig. 2. The only difference between them is the computation of $\boldsymbol{X}_{i+1}$ in step (9) of Algorithm 2. In recursive version, $\boldsymbol{X}_{i+1}$ updates with all the prediction results in the previous layers: $\boldsymbol{X}_{i+1} = \sigma(\boldsymbol{X} + \alpha \sum_{j=1}^{i} \boldsymbol{O}_j \boldsymbol{W}_{2j})$.

4. Performance evaluation

To verify the classification and regression performance of DrELM, we conduct the experiments on various kinds of benchmark data sets. We begin with two case studies on synthetic data to intuitively show the feasibility of DrELM. Then we compare the performance of DrELM with linear ELM, kernel ELMs and deep learning methods in real-world benchmark data sets to show the prediction accuracy and time efficiency of DrELM.

4.1. Case studies on synthetic data

The performance of the proposed DrELM has been tested on the classical *XOR* problem [16] and *Spiral* data set [17]. The *XOR* problem data set contains 40 training samples in each class and is not linearly separable. The *Spiral* data set consisting of 97 in each class intertwines as two spirals in two-dimensional space. We train DrELM with 20 hidden neurons and radial basis function kernel. Fig. 3 visualizes the classification hyperplanes of DrELM with different depths on these two synthetic data sets. The linear hyperplanes on the left of Fig. 3(a) and (b) are obtained by one-layer DrELM which is identical to ELM. The right ones are obtained by two-layer DrELM and are able to obtain non-linear hyperplanes by staking linear ELM. Thus, it is obvious that the results of two-layer DrELM is more "semantic" since they adapt to the nonlinear characteristics of the data. The results also demonstrate that higher-level representations can potentially capture data manifold more precisely [12].

4.2. Experiments on real-world data sets

4.2.1. Data sets and baseline methods

Data sets: The benchmark data sets are summarized in Tables 1 and 2. The *MNIST* [18] data set consists of gray-level images with $28 \times 28 = 784$ pixels in 10 classes. It contains 60,000 training samples and 10,000 test samples. The *CIFAR*-10 [19] data set consists of 60,000 32×32 color images in 10 classes, with 50,000 training images and 10,000 test images. The other data sets are taken from the UCI repository [20] and StatLib [21]. Training

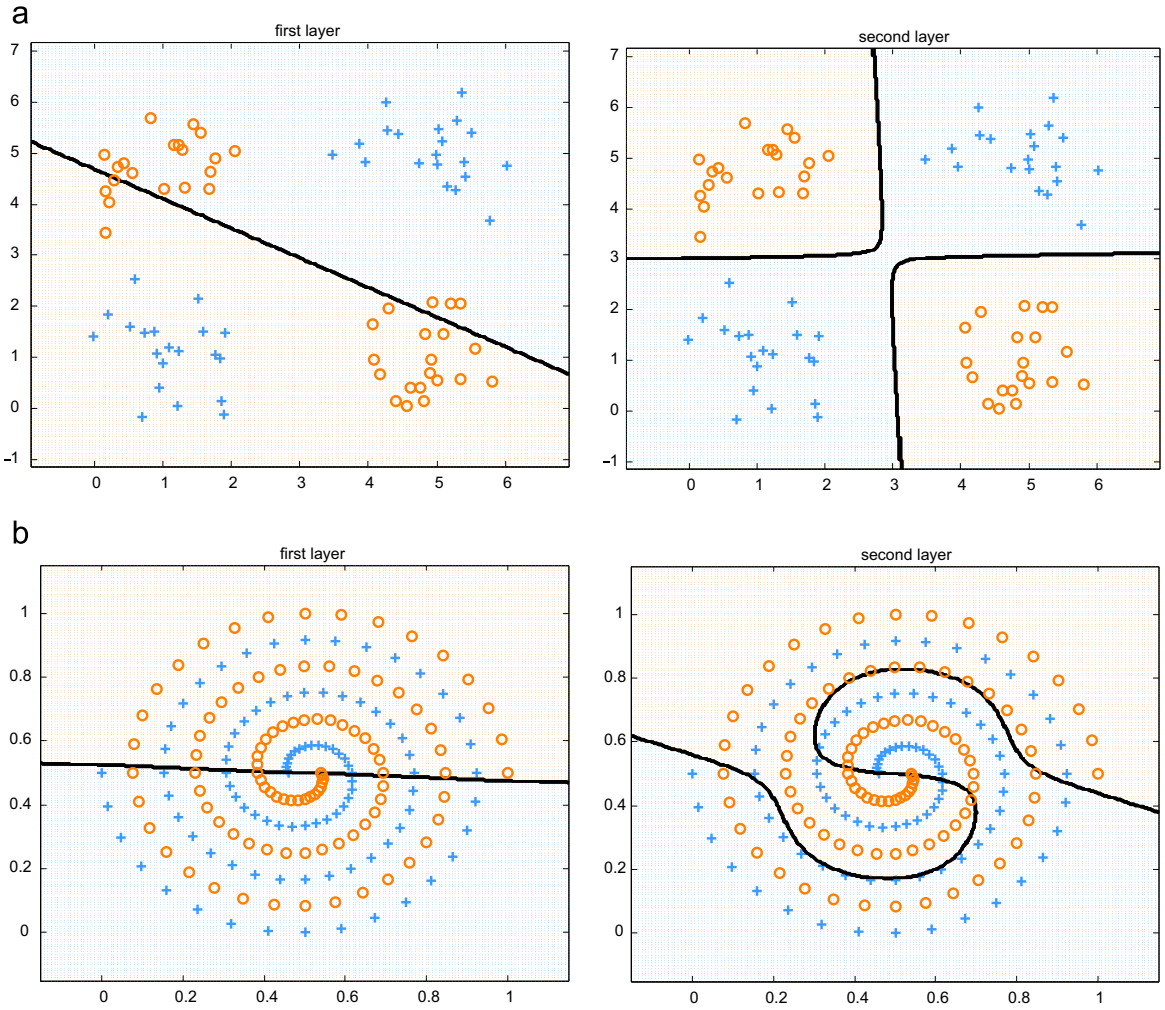

Fig. 3. Classification hyperplanes of DrELM with different depths on synthetic data sets. The linear hyperplanes on the left are obtained by one-layer DrELM, while the right ones are obtained by two-layer DrELM. (a) Classification hyperplane on *XOR* problem. (b) Classification hyperplane on *Spiral*.

Table 1
Specification of binary classification and multi-class classification data sets.

#	Data sets	#Training	#Testing	#Attributes	#Classes
1	Diabetes	512	256	8	2
2	Liver	230	115	6	2
3	Mushroom	2708	1354	22	2
4	Adult	21,708	10,854	123	2
5	Iris	100	50	4	3
6	Wine	119	59	13	3
7	Segment	1540	770	19	7
8	Satimage	4290	2145	36	6
9	MNIST	60,000	10,000	784	10
10	CIFAR-10	50,000	10,000	3072	10

Table 2
Specification of regression problems data sets.

#	Data sets	#Training	#Testing	#Attributes
1	Pyrim	49	25	28
2	Housing	337	169	14
3	Strike	416	209	7
4	Balloon	1334	667	2
5	Abalone	3177	1000	8

samples are two-thirds of each data set and the other one-third is for testing. All these data sets include 10 classification cases and 5 regression cases.

Baseline methods: In our experiments, we compare the proposed DrELM to the following baseline algorithms.

(1) *Linear ELM* [1,2]: Hidden node parameters remain fixed after randomly generated and compute the output weight vector $\boldsymbol{\beta}$ by solving the linear system $\boldsymbol{H\beta} = \boldsymbol{T}$.

(2) *Kernel based ELM* [3]: Gaussian and Sigmoid kernels are used to implement kernel based ELM.[1]

(3) *Optimally-Pruned Extreme Learning Machine* (*OP-ELM*) [6]: It is an improvement of ELM and in this paper we use OP-ELM Toolbox[2] to compute the results, the toolbox only supports binary classification.

(4) *Deep Belief Networks* (*DBN*)[3] [7]: The base building block of DBN is Restricted Boltzmann Machine. We fine-tune the results after layer-by-layer training.

(5) *Stacked Auto-encoder* (*SAE*)[4] [8]: The sparse auto-encoders [22] are stacked in a greedy layer-wise fashion for pre-training the

[1] http://www.ntu.edu.sg/home/egbhuang
[2] http://research.ics.aalto.fi/eiml/software.shtml
[3] https://github.com/rasmusbergpalm/DeepLearnToolbox
[4] http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

weights of a deep network and a final softmax classifier [23] is utilized. After the phase of training is complete, we fine-tune the parameters using back-propagation.

4.2.2. Prediction accuracy

In this subsection, we provide a quantitative evaluation of DrELM and Recursive DrELM (DrELM^r) on prediction performance for both classification and regression tasks.

Classification: We perform binary and multi-class classification on the data sets described in Table 1. For the baseline methods and DrELMs, the number of hidden nodes L is set as follows: For *MNIST*, *CIFAR*-10 and *Adult*, L varies from 20 to 2000 with the interval of 10; Others varies from 20 to 200 with the same interval. In the range of L we use 5-fold cross-validation on the training set to find the parameters. We use the same strategy to find the proper depth to train DBN, SAE and DrELMs, the depth varies from 1 to 10 (the values of depth will be analyzed in Section 4.2.3). α is set to 0.1 for all the experiments. After setting all the parameters, 50 trials have been conducted for each data set and take the average as the final results. The prediction accuracy of different models is shown in Table 3.

It can be seen that DrELM and DrELM^r perform better than linear ELM and kernel ELMs, while the performances of the deep models (i.e., DBN, Stacked Auto-encoder, DrELM and DrELM^r) are comparable. However DrELMs is easier to learn and faster in testing, as we shall see in Section 4.2.4.

Regression: Due to the utilization of ELM, DrELM is a general framework which compatible with multi-class classification and regression tasks. We evaluate DrELM on the regression data sets described in Table 2. The parameters are set by the same strategy described in classification tasks. We built DrELM use sigmoid kernel. The data are normalized by the following method: $\hat{\eta} = (\eta - \eta_{min})/(\eta_{max} - \eta_{min})$, where η_{max} is the maximum value and η_{min} is the minimum value of each feature. Table 4 shows the average results over 50 trials of the DrELMs and the baselines. We can see that DrELM can get better results than linear ELM and kernel ELMs, while the performances of DBN, SAE, DrELM and DrELM^r are comparable. This suggests that deep models can discover more predictive latent structure for difficult regression questions. Particularly, if we use kernel ELM instead of linear ELM to build the deep architecture, the accuracy will be improved, but the training time will increase accordingly.

4.2.3. Depth of DrELM

Depth is the key aspect to representation learning strategies. Fig. 4 shows the average depth for each data set when DrELM achieves the best prediction accuracy. As discussed in [13], there is no fast rules to identify which level of prediction functions should be use. In this paper, the quantitative evaluation on 15 data sets shows that the average depth is 6.66.

4.2.4. Time efficiency

In this section, we report empirical results on time efficiency in training. All models perform a feedforward way when testing, and there is no significant difference in testing time between the models with same layers. Thus we only report the average training time. All the following results are carried out with MATLAB 7.9.0 running on a standard desktop with a 3.30 GHz Intel processor and 16 G RAM.

Fig. 5 shows the average training time of different models on binary and multi-class classification tasks and regression tasks with 50 randomly initialized runs (the order of data sets on x-axis has been changed for better visualization). The parameters of DrELM and stacked auto-encoder are set by the same strategy described in Section 4.2.2. We can see from Fig. 5 that DrELM always runs faster than stacked auto-encoder (about 3.1–194.0 times faster in this figure).

Table 3
Classification results of different methods. We report the average accuracy of 50 trials over the hyper-parameters of DrELM and baseline methods. The data are all normalized on each feature and we built DrELM use sigmoid kernel.

#	Data sets	Linear	Kernel based ELM		OP-ELM	DBN	SAE	DrELM	DrELM^r
		ELM	Gaussian	Sigmoid					
1	Diabetes	0.7459	0.7732	0.7744	0.7799	0.7805	0.7783	0.7763	**0.7822**
2	Liver	0.6596	0.6735	0.6848	0.7048	0.6980	0.6962	**0.7507**	0.7421
3	Mushroom	0.9414	0.9658	0.9728	0.9890	**0.9994**	0.9986	0.9935	0.9847
4	Adult	0.8419	0.8468	0.8444	0.8437	0.8428	0.8448	0.8421	**0.8494**
5	Iris	0.8173	0.9462	0.9484	–	**0.9681**	0.9645	0.9477	0.9631
6	Wine	0.9881	0.9664	0.9847	–	**0.9901**	0.9898	0.9822	0.9747
7	Segment	0.8661	0.9425	0.9486	–	0.9551	0.9568	0.9530	**0.9579**
8	Satimage	0.7794	0.8648	0.8645	–	**0.8766**	0.8761	0.8481	0.8745
9	MNIST	0.8497	0.9201	0.9142	–	0.9627	**0.9675**	0.9478	0.9538
10	CIFAR-10	0.4043	0.3979	0.4239	–	**0.4362**	0.4338	0.4284	0.4312
	Average	0.7894	0.8297	0.8361	–	0.8509	0.8506	0.8470	**0.8514**

Table 4
Regression results of different methods. We report the average accuracy of 50 trials of DrELM, DrELM^r and baseline methods.

#	Data sets	Linear	Kernel based ELM		OP-ELM	DBN	SAE	DrELM	DrELM^r
		ELM	Gaussian	Sigmoid					
1	Pyrim	0.1524	0.1352	0.1284	0.1273	0.1268	0.1277	**0.1251**	0.1279
2	Housing	0.1022	0.0913	0.0878	0.0857	0.0840	0.0831	0.0821	**0.0784**
3	Strike	0.0942	0.0935	0.0973	0.0954	0.0984	0.0936	**0.0880**	0.0922
4	Balloon	0.0115	0.0133	0.0156	0.0108	0.0121	**0.0067**	0.0106	0.0082
5	Abalone	0.0925	0.0835	0.0817	0.0781	0.0718	0.0773	**0.0716**	0.0730
	Average	0.0906	0.0834	0.0822	0.0795	0.0786	0.0777	**0.0755**	0.0759

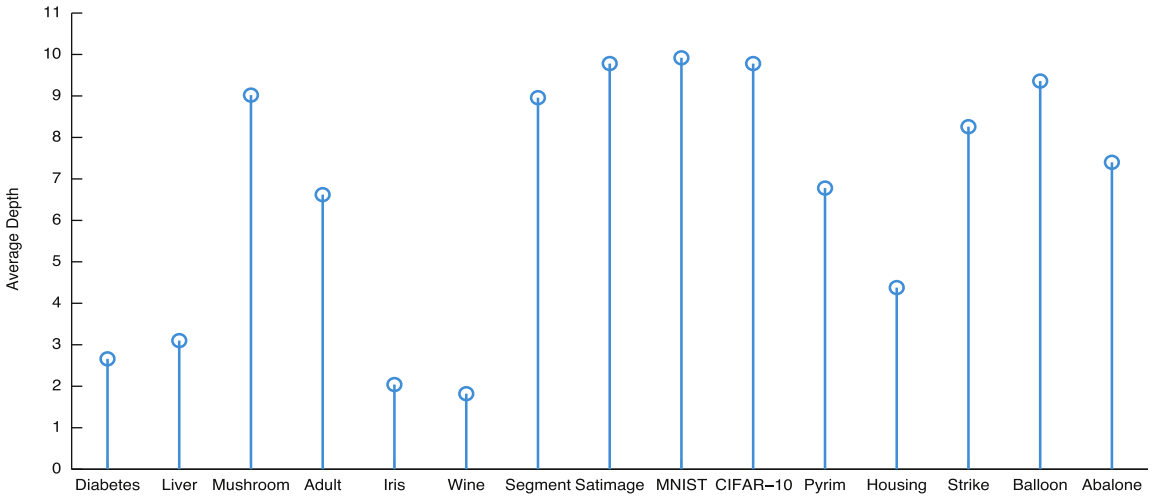

Fig. 4. The depth of each data set when DrELM achieves the best prediction accuracy. The average depth for 10 classification tasks is 6.37, and 7.24 for the rest 5 regression tasks. The total average is 6.66.

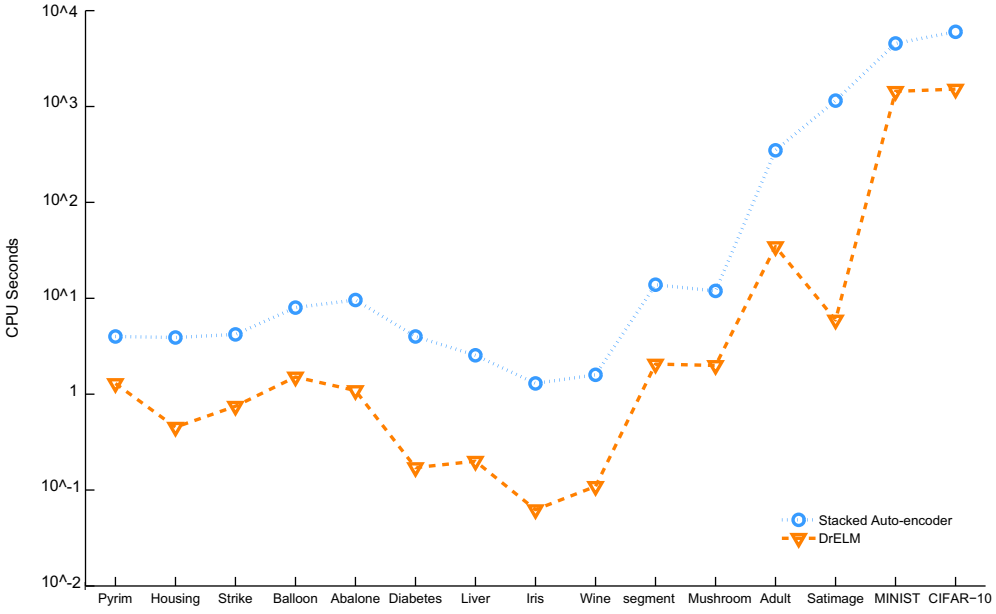

Fig. 5. Training time (CPU seconds in log-scale) of stacked auto-encoder and DrELM with respect to the data sets.

5. Related work

ELM as an emergent technology which overcomes the challenges such as slow learning speed, trivial human intervene and poor computational scalability faced by neural networks, support vector machines and other computational intelligence techniques [5]. The universal approximation capability of ELMs is investigated and it has been proved that SLFNs with randomly generated additive or RBF nodes can universally approximate any continuous target functions [2]. Recently, many variants of ELM have been proposed: a kernel matrix can be defined by the hidden layer feature mapping $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{h}(\boldsymbol{x}_i) \cdot \boldsymbol{h}(\boldsymbol{x}_j)$ thus one can implement ELMs with eligible kernel [3,24,25]. Online sequential ELMs are built for the applications in which training data come one by one or chunk by chunk [26]. Other approaches related to ELM such as incremental ELM [27], SVM with ELM feature mapping [4] are also fully studied by many researchers and a unified learning framework has been built referred to ELM for regression and multi-class classification [3]. The proposed DrELM is a deep architecture using ELMs as its base building block and follows the philosophy of stacked generalization [13].

Stacked generalization is a scheme for minimizing the generalization error of the prediction functions and deducing the biases with respect to the provided training set [13]. There are many different ways to combine the prediction functions to implement stacked generalization and various stacked implementation strategies have been developed. These strategies can be divided into two categories, one category utilizes stacking methodology to build deep architecture via layer-by-layer unsupervised pre-training, as exemplified in deep brief networks [7,28], deep auto-encoder [8,29] and convolutional neural networks [10,30]. The other stacking category is carried out using supervised information such as stacked regressions [14] and random recursive SVM [9]. However, many aspects of stacked generalization are “black art” at present. For example, there is no fast rules to identify which level of prediction functions should be used and the choice of stacking depth is a key aspect of representation learning strategies and still remains an open question. DrELM is a new stacking technique that has close connections

to the stacked methods mentioned above, but DrELM has its own merits, say, fast learning speed with better generalization ability due to the utilization of ELM.

The random recursive SVM (R^2SVM) [9] is closely related to our approach. Both methods follow stacked generations and use the outputs of previous layers to transform the input features thus improve the classification or regression performance. But DrELM is distinct from R^2SVM in the following aspects. Firstly, the base building block is different. DrELM utilizes ELMs to build deep architecture which has several advantages: ease of use, faster learning speed and higher generalization performance. Secondly, R^2SVM learns recursive perceptual representations for classification, however, DrELM is a general framework which is compatible with multi-class classification and regression tasks. In addition, DrELM can implement recursive and non-recursive versions with different kernel functions when transforming the features with the predictions of previous layers. In virtue of the flexible implementations, DrELM performs classification and regression tasks with more competitive efficiency and effectiveness.

5.1. Existing approaches to deep learning

Deep belief nets (DBNs) are probabilistic generative models which first trained only with unlabeled data and then fine-tuned in a supervised mode [7]. The base building block of DBN is Restricted Boltzmann Machine (RBM) [28]. The RBM is a bipartite graph with visible layer $\boldsymbol{v}$ and hidden layer $\boldsymbol{h}$. The energy of the joint configuration is

$$E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) = -\sum_{ij}\boldsymbol{W}_{ij}\boldsymbol{v}_i\boldsymbol{h}_j - \sum_i b_i\boldsymbol{v}_i - \sum_j a_j\boldsymbol{h}_j \tag{6}$$

where $\boldsymbol{\theta}=\{\boldsymbol{W},a,b\}$ is model parameters. The probability of the joint configuration is given by the Boltzmann distribution

$$P_\theta(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{\mathcal{Z}(\boldsymbol{\theta})}\exp(-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})) \tag{7}$$

where $\mathcal{Z}(\boldsymbol{\theta})=\sum_{\boldsymbol{v},\boldsymbol{h}}\exp(-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}))$ is the partition function. RBM possesses the useful property that the conditional distribution over the hidden units factorizes the given visible units and vice versa. This is trained with a procedure called contrastive divergence [28] whereby one pushes up the energy on training data and pushes down the energy on samples generated by the model.

The stacked auto-encoder is another deep learning scheme [8]. The auto-encoder is usually adopted as a base building block to initialize the stacked auto-encoder neural network. It is stacked in a greedy layer-wise fashion and trained to encode the inputs $\boldsymbol{x}^{(i)}, i=1,\ldots,m$ into some representations $f(\boldsymbol{x}^{(i)})=\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(i)}+b^{(1)})$ so that the resulting representations $h(\boldsymbol{x}^{(i)})=\sigma(\boldsymbol{W}^{(2)}f(\boldsymbol{W}^{(1)},b^{(1)};\boldsymbol{x}^{(i)})+b^{(2)})$ can revert to their original forms $\boldsymbol{x}^{(i)}$. Thus the reconstruction error needs to be minimized

$$\min_{\boldsymbol{W},b}\frac{1}{2}\sum_{i=1}^{m}\|h(\boldsymbol{W},b;\boldsymbol{x}^{(i)})-\boldsymbol{x}^{(i)}\|^2 \tag{8}$$

Sparse coding [31] is another technique that can be used as a base building block for deep architecture [32]. It also relates a latent representation $\boldsymbol{h}$ to the input data $\boldsymbol{x}$ through a linear mapping $\boldsymbol{W}$, which we refer as the dictionary. Specifically, sparse coding can be seen as recovering the code or feature vector associated with a new input $\boldsymbol{x}$ via

$$f(\boldsymbol{x}) = \arg\min_{\boldsymbol{h}}\|\boldsymbol{x}-\boldsymbol{W}\boldsymbol{h}\|_2^2+\lambda\|\boldsymbol{h}\|_1 \tag{9}$$

Learning the dictionary $\boldsymbol{W}$ can be accomplished by optimizing the following training criterion with respect to $\boldsymbol{W}$

$$\min_{\boldsymbol{W}}\sum_i\|x^{(i)}-\boldsymbol{W}\boldsymbol{h}^{(i)}\|_2^2 \tag{10}$$

where $\boldsymbol{x}^{(i)}$ is the input vector and $\boldsymbol{h}^{(i)}$ is the corresponding sparse codes computed by Eq. (9).

The stacking technique enjoys greater power derived from deep models and numerous deep models are introduced, such as sum-product networks (SPNs) [33] and random recursive SVM (R^2SVM) [9] and stacked regressions [14]. The last two follows the philosophy of stacked generations [13].

6. Conclusions

In this paper we proposed a stacked framework, DrELM, to learn deep representations via extreme learning machines. DrELM integrates the stacked generalization principle into the process of deep representation learning. The proposed framework utilizes ELM as a base building block and incorporates random shift and kernelization as stacking elements. We provide empirical evidence which demonstrates that DrELM could yield predictive features that are suitable for prediction tasks, such as regression and classification. Our empirical results on UCI data sets, *MNIST* and *CIFAR*-10 demonstrate that DrELM is an attractive stacked generalization which outperforms ELM and kernel based ELMs, while the performances of the deep models are comparable. In addition, the training time of DrELM is always less than deep auto-encoders by orders of magnitude. We plan to enhance the performance of DrELM by designing fine-tuning mechanisms. Also, we will investigate semi-supervised ELM in order to extend it to deep architectures with layer-by-layer unsupervised pre-training.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61175052612032297, 60933004, 61035003), National High-tech R&D Program of China (863 Program) (No. 2013AA01A606, 2012AA011003), National Program on Key Basic Research Project (973 Program) (No. 2013CB329502).

References

[1] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: IEEE International Joint Conference on Neural Networks, 2004, pp. 985–990.
[2] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: theory and applications, Neurocomputing (2006) 489–501.
[3] G. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst., Man, Cybern., Part B: Cybern. (2010) 1–17.
[4] Q. Liu, Q. He, Z. Shi, Extreme support vector machine classifier, Adv. Knowl. Discov. Data Min. (2008) 222–233.
[5] G. Huang, D.H. Wang, Y. Lan, Extreme learning machines: a survey, Int. J. Mach. Learn. Cybern. (2011) 107–122.
[6] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, Op-elm: optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162.
[7] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, Adv. Neural Inf. Process. Syst. (2007) 153.
[9] O. Vinyals, Y. Jia, L. Deng, T. Darrell, Learning with recursive perceptual representations, in: Advances in Neural Information Processing Systems, 2012, pp. 2834–2842.
[10] A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1106–1114.
[11] R. Socher, J. Pennington, E.H. Huang, A.Y. Ng, C.D. Manning, Semi-supervised recursive autoencoders for predicting sentiment distributions, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2011, pp. 151–161.
[12] Y. Bengio, G. Mesnil, Y. Dauphin, S. Rifai, Better mixing via deep representations, in: Proceedings of the 30th International Conference on Machine Learning, 2013.
[13] D.H. Wolpert, Stacked generalization, Neural Netw. (1992) 241–259.
[14] L. Breiman, Stacked regressions, Mach. Learn. (1996) 49–64.

[15] D. Rumelhart, G. Hintont, R. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.
[16] M. Minsky, S. Papert, Perceptron: An Introduction to Computational Geometry, expanded edition, vol. 19, The MIT Press, Cambridge, 1969, p. 88.
[17] A. Wieland, Twin spiral dataset, Available at: ⟨http://www-cgi.cs.cmu.edu~/afs/cs.cmu.edu/project/ai-repository/ai/areas/neural/bench/cmu/0.html⟩.
[18] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[19] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images (Master's thesis), Department of Computer Science, University of Toronto, 2009.
[20] C. Blake, C. Merz, Uci repository of machine learning databases, 1998.
[21] M. Mike, Statistical datasets. Available from: http://lib.stat.cmu.edu/datasets, 1989.
[22] A. Ng, Cs294a lecture notes: sparse autoencoder, in: Stanford University, 2010.
[23] W. Greene, C. Zhang, Econometric Analysis, Prentice Hall, Upper Saddle River, NJ, 1997.
[24] B. Frénay, M. Verleysen, Parameter-insensitive kernel in extreme learning for non-linear support vector regression, Neurocomputing 74 (16) (2011) 2526–2531.
[25] B. Frénay, M. Verleysen, Using svms with randomised feature spaces: an extreme learning approach, in: ESANN, 2010.
[26] G. Huang, N. Liang, H. Rong, P. Saratchandran, N. Sundararajan, Online sequential extreme learning machine, in: The IASTED International Conference on Computational Intelligence, 2005.
[27] G. Huang, L. Chen, C. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw, 2006, 879–892.
[28] G. Hinton, A practical guide to training restricted Boltzmann machines, Momentum 9 (1) (2010).
[29] Y. Bengio, Learning deep architectures for ai, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.
[30] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition, in: IEEE 12th International Conference on Computer Vision, 2009, pp. 2146–2153.
[31] B.A. Olshausen, et al., Emergence of simple-cell receptive field properties by learning a sparse code for natural images, Nature (1996) 607–609.
[32] K. Yu, Y. Lin, J. Lafferty, Learning image representations from the pixel level via hierarchical sparse coding, in: IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp. 1713–1720.
[33] H. Poon, P. Domingos, Sum-product networks: a new deep architecture, in: IEEE International Conference on Computer Vision Workshops, 2011, pp. 689–690.

Wenchao Yu is a Master candidate student in the Institute of Computing Technology, Chinese Academy of Sciences. He received the B.S degree from International School of Software, Wuhan University, Hubei, China, in 2011. His research interests include machine learning, data mining and deep learning. He has published several papers in relevant forums, such as ECMLPKDD.

Fuzhen Zhuang is an Assistant Professor in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include transfer learning, machine learning, data mining, distributed classification and clustering, natural language processing. He has published several papers in some prestigious refereed journals and conference proceedings, such as IEEE Transactions on Knowledge and Data Engineering, Information Sciences, Neurocomputing, ACM CIKM, ACM WSDM, IEEE ICDM, SIAM SDM.

Qing He is a Professor at the Institute of Computing Technology, Chinese Academy of Sciences (CAS), and he is a Professor of Graduate University of Chinese Academy of Sciences (GUCAS). He received the B.S. degree from Hebei Normal University, Shijiazhuang, P.R.C., in 1985, and the M.S. degree from Zhengzhou University, Zhengzhou, P.R.C., in 1987, both in Mathematics. He received the Ph.D. degree in 2000 from Beijing Normal University in Fuzzy Mathematics and Artificial Intelligence, Beijing, P.R.C. Since 1987–1997, he has been with Hebei University of Science and Technology. He is currently a doctoral tutor at the Institute of Computing Technology, CAS. His interests include data mining, machine learning, cloud computing and big data.

Zhongzhi Shi is a Professor in the Institute of Computing Technology, CAS, leading the Research Group of Intelligent Science. His research interests include intelligence science, multi-agent systems, semantic Web, machine learning and neural computing. He has won a 2nd-Grade National Award at Science and Technology Progress of China in 2002, two 2nd-Grade Awards at Science and Technology Progress of the Chinese Academy of Sciences in 1998 and 2001, respectively. He is a senior member of IEEE, member of AAAI and ACM, Chair for the WG 12.2 of IFIP. He serves as a Vice President for Chinese Association of Artificial Intelligence.