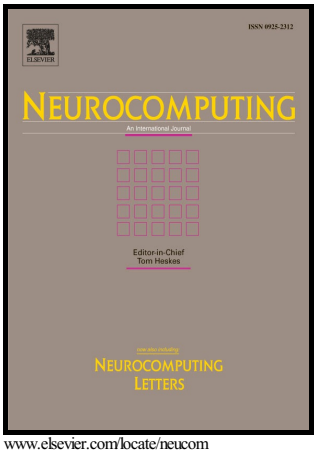
Generalized extreme learning machine autoencoder and a new deep neural network

Kai Sun, Jiangshe Zhang, Chunxia Zhang, Junying Hu



www.elsevier.com/locate/neucom

PII: S0925-2312(16)31503-X
DOI: http://dx.doi.org/10.1016/j.neucom.2016.12.027
Reference: NEUCOM17845

To appear in: *Neurocomputing*

Received date: 2 August 2016
Revised date: 23 November 2016
Accepted date: 8 December 2016

Cite this article as: Kai Sun, Jiangshe Zhang, Chunxia Zhang and Junying Hu Generalized extreme learning machine autoencoder and a new deep neural network, *Neurocomputing,* http://dx.doi.org/10.1016/j.neucom.2016.12.027

Generalized extreme learning machine autoencoder and a new deep neural network

Kai Sun, Jiangshe Zhang*, Chunxia Zhang, Junying Hu

School of Mathematics and Statistics, Xi'an Jiaotong University, China

Abstract

Extreme learning machine (ELM) is an efficient learning algorithm of training single layer feed-forward neural networks (SLFNs). With the development of unsupervised learning in recent years, integrating ELM with autoencoder has become a new perspective for extracting feature using unlabeled data. In this paper, we propose a new variant of extreme learning machine autoencoder (ELM-AE) called generalized extreme learning machine autoencoder (GELM-AE) which adds the manifold regularization to the objective of ELM-AE. Some experiments carried out on real-world data sets show that GELM-AE outperforms some state-of-the-art unsupervised learning algorithms, including k-means, laplacian embedding (LE), spectral clustering (SC) and ELM-AE. Furthermore, we also propose a new deep neural network called multilayer generalized extreme learning machine autoencoder (ML-GELM) by stacking several GELM-AE to detect more abstract representations. The experiments results show that ML-GELM outperforms ELM and many other deep models, such as multilayer ELM autoencoder (ML-ELM), deep belief network (DBN) and stacked autoencoder (SAE). Due to the utilization of ELM, ML-GELM is also faster than DBN and SAE.

Keywords: Extreme learning machine, generalized extreme learning machine autoencoder, manifold regularization, deep neural network, multilayer

*Corresponding author

Email addresses: kai.s@foxmail.com (Kai Sun), jszhang@mail.xjtu.edu.cn (Jiangshe Zhang), cxzhang@mail.xjtu.edu.cn (Chunxia Zhang), hujunyingmm@163.com (Junying Hu)

1. Introduction

In the past several decades, single-hidden layer feedforward networks (abbreviated as SLFNs) have been intensively studied. Because most of the existing algorithms for training SLFNs are time-consuming, Huang et al. [1, 2] proposed extreme learning machines (ELMs) which are simple but efficient. The key principle of ELMs is to randomly choose the weights from input layer to hidden layer and then analytically determine the output weights of SLFNs through minimizing a loss function. Since then, much theoretical and experimental evidence has proven the multiple advantages of ELMs, such as fast training, strong generalization ability and universal approximation capability [2]. So far, a large variety of ELMs have been successfully applied in many areas, for example, face classification [3], image segmentation [4], clustering [5] and etc.

Recently, deep learning (DL) [6, 7] has gained more and more attention in pattern recognition and machine learning fields since it has made excellent performance on many difficult tasks such as speech recognition, objection detection and image classification. In some cases, its recognition ability has even exceeded human's performance. To get good representations of data, deep learning models are specifically designed to imitate the hierarchical architecture of mammalian brain through using a deep architecture. Essentially, it can also be deemed as a neural network having many hidden layers. Since each hidden layer acts as a feature detector, a deep learning model is composed of many feature detector units. In general, lower layers extract simple features, feed into higher layers and higher layers can thus extract more abstract and complex features. Currently, the outstanding representatives of deep learning models include deep belief network (DBN, [6]), stacked autoencoders (SAEs, [8]), convolutional neural networks (CNNs, [9]) and so on.

In fact, the concept of deep learning is not new in machine learning field. However, people cannot find an efficient algorithm to learn the associated pa-

rameters until Hinton and Salakhutdinov [6] proposed a greedy layer-by-layer training manner, which can be seen as a breakthrough for training deep networks. Based on the efficient learning algorithm, deep learning began to exhibit its great power in various domains with the aid of computers' powerful computing ability and special devices (e.g., graphics processing units, abbreviated GPUs) as well as the availability of large amount of data. Nowadays, even though the computational cost of learning deep learning models are acceptable, it is still one of the hot topics in deep learning to investigate how to improve their efficiency.

In view of the high efficiency of ELMs, some researchers devoted to studying how to integrate ELMs into the common deep learning framework. For example, Kasun et al. [10] introduced the ELM autoencoder (ELM-AE) which represent the original data in another feature space. Inspired by the idea of deep learning, they also proposed to stack multiple ELM-AEs into a multilayer ELM autoencoder (ML-ELM). Tissera and McDonnel [11] presented a deep neural networks using ELMs as a stack of supervised autocoders, which is inspired by that used in discriminative restricted boltzmann machines [12]. Yu et al. [13] propose a stacked model, DrELM, to learn deep representations via ELM and random projection. Hu et al. [14] developed a stacked deep neural network, StURHN-SLFNs (SLFNs with randomly fixed hidden neurons), via unsupervised RHN-SLFNs according to staked generalization philosophy to deal with unsupervised problems.

To improve the performance of existing machine learning models, learning with local consistency of data has gained much attention of researchers. Laplacian embedding (LE) [15] and spectral clustering (SC) [16, 17] are two commonly used techniques. However, LE and SC may result in loss of information since they are essentially dimensionality reduction algorithms. In contrast, ELM-AE can avoid the loss of information because the dimension of new space could be greater than the original dimension. Nevertheless, ELM-AE doesn't take the data's intrinsic Manifold structure (i.e., data points which are near in the original space should be as close as possible in the new space) into account. Based

on this fact, we propose a discriminative graph regularized extreme learning machine autoencoder in this paper, also called generalized extreme learning machine autoencoder (GELM-AE). In GELM-AE, the constraint imposed on output weights enforces the output of similar samples to be close in new space. The constraint is formulated as a regularization term added to the objective of the basic ELM-AE model. In this manner, the output weights can be solved analytically. In order to further enhance the ability of GELM-AE to learn good features, we propose to stack several GELM-AEs into a deep model which is abbreviated as ML-GELM. The experiments carried out on some real-world datasets show that GELM-AE has good capacity to extract meaningful features. Meanwhile, ML-GELM shows the excellent results on the studied problems but with lower time complexity in comparison with several other deep learning methods.

The remainder of the paper is organized as follows. In Section 2, we give a brief review of ELM-AE and manifold regularization. Section 3 presents our proposed method GELM-AE and the corresponding algorithm. In section 4, multiple GELM-AEs are stacked into a ML-GELM by using the idea of deep learning. Section 5 devotes to examining and comparing the performance of GELM-AE and ML-GELM with other techniques. Finally, section 6 concludes this paper.

2. Preliminaries

To make the paper self-contained, we will briefly introduce ELM-AE and manifold regularization framework in this section.

2.1. Introduction of ELM-AE

Based on ELM, the extreme learning machine autoencoder (ELM-AE) [10] is a new unsupervised learning algorithm. In ELM-AE, there is a single hidden layer. Meanwhile, the input data are also used as output data. Fig.1 illustrates the specific network structure of a ELM-AE. For clarity of presentation, we give the following explanation of symbols.

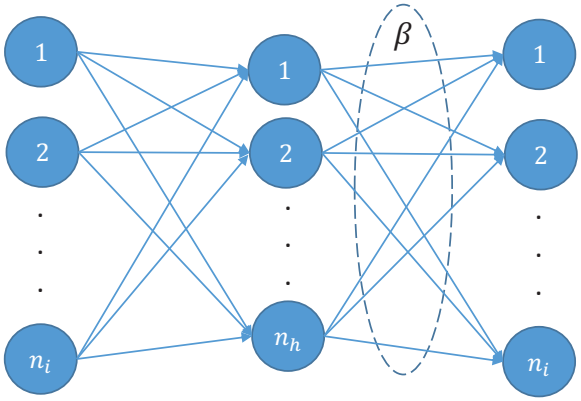


Figure 1: Illustration of the structure of a ELM-AE.

- N: the number of samples;
- n_i: the number of input units;
- n_h: the number of hidden units;
- $\boldsymbol{a}$: the input weights between the input layer and the hidden layer which is a $n_i \times n_h$ matrix;
- $\boldsymbol{b}$: the biases of hidden units;
- $\boldsymbol{\beta}$: the output weights between the hidden layer and the output layer which is $n_h \times n_i$ matrix.

The basic idea for training an ELM-AE is simple, which is a two-stage process. In the first stage, a number of hidden neurons map the original data into a n_h dimensional feature space. According to the size of n_i and n_h, an ELM-AE can represent the extracted features with three different architectures: 1) $n_i > n_h$, compressed architecture; 2) $n_i < n_h$, sparse architecture; 3) $n_i = n_h$, equal dimension architecture.

For the compressed architecture of an ELM-AE, Johnson-Lindenstrauss Lemma [18] shows that the euclidean distances between each input data points and the

corresponding euclidean distances in lower dimension space is equal. In this situation, the mapping of $\boldsymbol{x}_i$ to a n_h dimensional space can be calculated by

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}_i) &= g(\boldsymbol{a}^T\boldsymbol{x}_i + \boldsymbol{b}) \\ \boldsymbol{a}^T\boldsymbol{a} &= \boldsymbol{I}, \boldsymbol{b}^T\boldsymbol{b} = 1. \end{aligned} \tag{1}$$

where $\boldsymbol{h}(\boldsymbol{x}_i) \in \mathbb{R}^{n_h}$ is the output vector of the hidden layer with respect to $\boldsymbol{x}_i$, $g(\cdot)$ is an activation function which can be sigmoid function, gaussian function and so on, $\boldsymbol{I}$ is an identity matrix of order n_h.

As for the sparse ELM-AE architecture, the orthogonal hidden random parameters can be computed according to

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}_i) &= g(\boldsymbol{a}^T\boldsymbol{x}_i + \boldsymbol{b}) \\ \boldsymbol{a}\boldsymbol{a}^T &= \boldsymbol{I}, \boldsymbol{b}^T\boldsymbol{b} = 1. \end{aligned} \tag{2}$$

Then, the outputs of the network are given by

$$\boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{h}(\boldsymbol{x}_i)^T\boldsymbol{\beta}, \quad i = 1, \ldots, N. \tag{3}$$

In the second stage, ELM-AE needs to update the output weights $\boldsymbol{\beta}$ by adopting the squared loss of the prediction error. The mathematical model for training ELM-AE is as follows:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^{n_h\times n_i}} L_{ELM-AE} = \min_{\boldsymbol{\beta}\in\mathbb{R}^{n_h\times n_i}} \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{C}{2}\|\boldsymbol{X} - \boldsymbol{H}\boldsymbol{\beta}\|^2 \tag{4}$$

where the first term is a regularization term which controls the complexity of the model, and C is a penalty coefficient on the training errors. And $\boldsymbol{X} = [\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \ldots, \boldsymbol{x}_N^T]^T \in \mathbb{R}^{N\times n_i}$, $\boldsymbol{H} = [\boldsymbol{h}(\boldsymbol{x}_1)^T, \boldsymbol{h}(\boldsymbol{x}_2)^T, \ldots, \boldsymbol{h}(\boldsymbol{x}_N)^T]^T \in \mathbb{R}^{N\times n_h}$.

By setting the gradient of L_{ELM-AE} with respect to $\boldsymbol{\beta}$ to zero, we have

$$\nabla_{ELM-AE} = \boldsymbol{\beta} - C\boldsymbol{H}^T(\boldsymbol{X} - \boldsymbol{H}\boldsymbol{\beta}) = 0 \tag{5}$$

The above gradient equation is easy to solve and its solution will be discussed in the following two cases.

When the number of training samples N is larger than the number of the hidden neurons n_h, we can have the following closed-form solution

$$\boldsymbol{\beta}^* = \left(\boldsymbol{H}^T\boldsymbol{H} + \frac{\boldsymbol{I}_{n_h}}{C}\right)^{-1}\boldsymbol{H}^T\boldsymbol{X} \tag{6}$$

where $\boldsymbol{I}_{n_h}$is an identity matrix of dimension n_h.

If the number of training samples N is smaller than the number of hidden neurons n_h, $\boldsymbol{H}$ will have more columns than rows. In this case, we introduce additional constraints to $\boldsymbol{\beta} = \boldsymbol{H}^T\boldsymbol{\alpha}(\boldsymbol{\alpha} \in \mathbb{R}^{N\times n_i})$. Then the closed-form solution of (4) becomes

$$\boldsymbol{\beta}^* = \boldsymbol{H}^T\left(\boldsymbol{H}\boldsymbol{H}^T + \frac{\boldsymbol{I_N}}{C}\right)^{-1}\boldsymbol{X} \tag{7}$$

where $\boldsymbol{I}_N$ is an identity matrix of dimension N.

Given input data $\boldsymbol{X}$, we can obtain its representation in a n_h dimensional space as $\boldsymbol{X}_{new} = \boldsymbol{X}\boldsymbol{\beta}^T$. Subsequently, we can use $\boldsymbol{X}_{new}$ to replace the original data in the clustering and embedding tasks. Based on the above discussion, we summarized the main steps of ELM-AE for clustering in Algorithm 1.

Algorithm 1: ELM-AE Algorithm for clustering

Input: Data $\{\boldsymbol{X}\} = \{\boldsymbol{x}_i\}_{i=1}^N$, the number of hidden neurons n_h, the penalty coefficient C

Output: The cluster result

Step 1:

Initialize an ELM-AE of n_h hidden neurons with random orthogonal input weights and biases.

Step 2:

If $n_h \leq N$

Compute the output weights $\boldsymbol{\beta}$ using (6)

Else

Compute the output weights $\boldsymbol{\beta}$ using (7)

Step 3:

$\boldsymbol{X}_{new} = \boldsymbol{X}\boldsymbol{\beta}^T$

Step 4:

Treat each row of $\boldsymbol{X}_{new}$ as a point and cluster the N points into K clusters using k-means algorithm.

2.2. Introduction of manifold regularization

Manifold regularization exploits the geometry of marginal distribution, which has been quite successfully used for semisupervised tasks [19, 20]. Two assumptions are built in semisupervised learning: (1) both the labeled data $\boldsymbol{X}_l$ and the unlabeled data $\boldsymbol{X}_u$ come from the same marginal distribution P_X; (2) if two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are close to each other, then the conditional probabilities $p(\boldsymbol{y}|\boldsymbol{x}_1)$ and $P(\boldsymbol{y}|\boldsymbol{x}_2)$ should be similar as well. This is just the basic idea of manifold regularization. Generally, the manifold regularization can be formulated as

$$L_m = \frac{1}{2}\sum_{ij} W_{ij}\|p(\boldsymbol{y}|\boldsymbol{x}_i) - p(\boldsymbol{y}|\boldsymbol{x}_j)\|^2 \tag{8}$$

where W_{ij} is the pair-wise similarity between two patterns $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

Since it is difficult to compute the conditional probability, L_m defined in (8) is usually substituted by the following formula

$$\hat{L}_m = \frac{1}{2}\sum_{ij} W_{ij}\|\hat{\boldsymbol{y}}_i - \hat{\boldsymbol{y}}_j\|^2 \tag{9}$$

where $\hat{\boldsymbol{y}}_i$ and $\hat{\boldsymbol{y}}_j$ are the predictions with respect to samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively.

For the expression in (9), we can easily get the matrix expression, that is,

$$\hat{L}_m = Tr(\hat{\boldsymbol{Y}}^T L \hat{\boldsymbol{Y}}) \tag{10}$$

where $Tr(\cdot)$ is the trace of a matrix, $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ is known as the *graph Laplacian*, $\boldsymbol{W} = (w_{ij})$ is the similarity matrix which is usually sparse. Only when two patterns $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close, e.g., $\boldsymbol{x}_i$ is among the k nearest neighbors of $\boldsymbol{x}_j$ or $\boldsymbol{x}_j$ is among the k nearest neighbors of $\boldsymbol{x}_i$, we will assign a nonzero value to w_{ij}. Otherwise, we will assign zero to w_{ij}. The nonzero weights w_{ij} are usually computed using gaussian function ($exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma^2)$), or simply fixed to 1. $\boldsymbol{D}$ is a diagonal matrix given by $D_{ii} = \sum_j w_{ij}$. As discussed in [21], instead of using $\boldsymbol{L}$ directly, we can normalize it by $\boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$ or replace it by $\boldsymbol{L}^p$ (p is an integer) based on some prior knowledge.

3. Graph regularized extreme learning machine autoencoder

In this section, we will introduce a new discriminative graph regularized extreme learning machine autoencoder(GELM-AE), and give the model and as well as the main steps of its learning algorithm.

The structure of a GELM-AE is similar to that of an ELM-AE, as shown in Fig.1. It is well-known that unsupervised learning makes use of the unlabeled data to find the underlying structure of the original data. The ELM-AE could also handle the unsupervised learning and have excellent feature extraction ability. Nevertheless, GELM-AE introduces the manifold regularization into it. Introducing the manifold regularization guarantees that the original datas which are near each other will be closer in output space. So GELM-AE have the better result than the ELM-AE.

For a set of unlabeled data $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}$, where N is the number of training samples. The optimization objective function of GELM-AE can be formulated as

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^{n_h\times n_o}} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{\kappa}{2}\|\boldsymbol{X} - \boldsymbol{H}\boldsymbol{\beta}\|^2 + \frac{\lambda}{2}Tr(\boldsymbol{\beta}^T\boldsymbol{H}^T\boldsymbol{L}\boldsymbol{H}\boldsymbol{\beta}) \tag{11}$$

where $\boldsymbol{L}$ is *graph Laplacian*.

In general, we compute the gradient of the above objective function with respect to $\boldsymbol{\beta}$ and can have

$$\nabla \boldsymbol{L}_{GELM-AE} = \boldsymbol{\beta} - \boldsymbol{C}\boldsymbol{H}^T(\boldsymbol{X} - \boldsymbol{H}\boldsymbol{\beta}) + \lambda\boldsymbol{H}^T\boldsymbol{L}\boldsymbol{H}\boldsymbol{\beta} \tag{12}$$

where $\boldsymbol{C}$ is a $N \times N$ diagonal matrix with its diagonal elements $[\boldsymbol{C}]_{ii} = \kappa, i = 1, \ldots, N$. By setting the gradient to zero, we can get the closed-form solution for (11).

Similar to the situation of an ELM-AE, we will discuss the solution of a GELM-AE in two different cases. When the number of training samples is larger than the number of the hidden neurons, we have

$$\boldsymbol{\beta}^* = \left(\boldsymbol{I}_{n_h} + \boldsymbol{H}^T\boldsymbol{C}\boldsymbol{H} + \lambda\boldsymbol{H}^T\boldsymbol{L}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{C}\boldsymbol{X} \tag{13}$$

where $\boldsymbol{I}_{n_h}$ is an identity matrix of dimension n_h.

When the number of training samples is smaller than the number of hidden neurons, $\boldsymbol{H}$ will have more columns than rows. In this case, we introduce additional constraints to $\boldsymbol{\beta} = \boldsymbol{H}^T\boldsymbol{\alpha}(\boldsymbol{\alpha} \in \mathbb{R}^{N\times n_o})$. Under this circumstance, the closed-form solution of (11) is

$$\boldsymbol{\beta}^* = \boldsymbol{H}^T\left(\boldsymbol{I}_N + \boldsymbol{C}\boldsymbol{H}\boldsymbol{H}^T + \lambda\boldsymbol{L}\boldsymbol{H}\boldsymbol{H}^T\right)^{-1}\boldsymbol{C}\boldsymbol{X} \tag{14}$$

where $\boldsymbol{I}_N$ is an identity matrix of dimension N.

Now we get another representation $\boldsymbol{X}_{new}$ of the input data $\boldsymbol{X}$ in a n_h dimensional space as $\boldsymbol{X}_{new} = \boldsymbol{X}\boldsymbol{\beta}^T$. Subsequently, instead of the original data, we can use $\boldsymbol{X}_{new}$ for clustering and embedding tasks. Based on the above discussion, we summarized the main steps of GELM-AE for clustering into Algorithm 2.

Algorithm 2: GELM-AE Algorithm for Clustering

Input: Data $\{\boldsymbol{X}\} = \{\boldsymbol{x}_i\}_{i=1}^N$, the number of hidden neurons n_h, the penalty coefficient κ and λ

Output: The cluster results

Step 1:

Initialize an ELM of n_h hidden neurons with random input weights and biases.

Step 2:

If $n_h \leq N$

Compute the output weights $\boldsymbol{\beta}$ using (13)

Else

Compute the output weights $\boldsymbol{\beta}$ using (14)

Step 3:

$\boldsymbol{X}_{new} = \boldsymbol{X}\boldsymbol{\beta}^T$

Step 4:

Treat each row of $\boldsymbol{X}_{new}$ as a point and cluster the N points into K clusters using the k-means algorithm.

4. Multilayer graph regularized extreme learning machine auto-encoder(ML-GELM): stacked deep neural network based on GELM-AE

We now introduce a new deep architecture, multilayer graph regularized extreme learning machine autoencoder (ML-GELM) to capture the higher-level underlying structure in the data. The model structure and the details of the algorithm are given as follows.

ML-GELM is a stacked multilayer neural network model which employs unsupervised GELM-AE as a base building block. It trains GELM-AE in a feedforward way and uses the outputs of each GELM-AE as the inputs of the next GELM-AE, as shown in Fig.2. More specifically, we first train the bottom

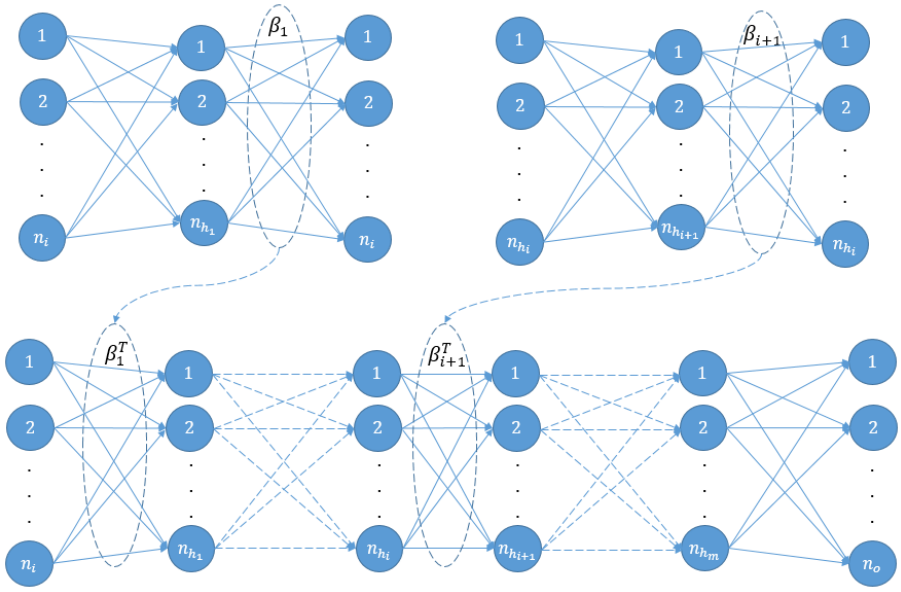


Figure 2: Illustration of the structure of a ML-GELM.

GELM-AE by Algorithm 2, as described in Section 3. The outputs of the bottom GELM-AE are then taken as the inputs of the 2nd layer GELM-AE to train the second GELM-AE. And by this analogy, we can get the whole structure of ML-GELM.

The main aim of ML-GELM is to find the underlying structure of the input

Algorithm 3: ML-GELM Algorithm for Classification

Input:

Train data: $\{\boldsymbol{X}_{train}, \boldsymbol{Y}_{train}\} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$;

Test data: $\{\boldsymbol{X}_{test}, \boldsymbol{Y}_{test}\} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{M}$;

The model depth: m;

The number of hidden nodes in the each GELM-AE: $n_{h_1}, n_{h_2}, \ldots, n_{h_m}$;

The new activation function: h_{new}.

Output: The classification results of the M testing data

Step 1:

Initialize $\boldsymbol{X}_1 = \boldsymbol{X}_{train}$.

Step 2:

- **For** $i = 1 : m$
  - Initialize the input weights $\boldsymbol{W}_i$ and biases $\boldsymbol{b}_i$ of the ith layer GELM-AE by some random numbers;
  - Train the output weights $\boldsymbol{\beta}_i$ of the ith layer GELM-AE;
  - Compute the outputs $\boldsymbol{X}_{i+1} = h_{new}(\boldsymbol{X}_i {\boldsymbol{\beta}_i}^T)$.
- **EndFor**

Step 3:

Map $\boldsymbol{X}_{m+1}$, the output of the mth layer, to the output layer.

Step 4:

Compute the classification results of test data by using the above trained ML-GELM model.

data. If the ML-GELM can be trained well, it will capture excellent potential structure and then can do a better job in handling classification or other tasks. Taking classification as an example, the detailed steps to train a ML-GELM are summarized as Algorithm 3.

5. Experimental results

In this section, to examine the performance of GELM-AE and ML-GELM, we conducted experiments on various kinds of benchmark data. And we compared our proposed methods with some other existing techniques. Moreover, we also verified the advantage of ML-GELM in terms of time complexity in comparison to some traditional deep model.

5.1. GELM-AE algorithms

In this subsection, we will investigate the feature extraction ability of GELM-AE and compare it with some other methods. The used benchmark data sets are summarized in Table 1. The data sets IRIS, WINE and GLASS are from UCI repository [22]. We also considered one face recognition data set YALEB [23], and one multiclass image classification data set COIL20 [24]. These characteristics of these data sets are also summarized in Table 1. Here, we compared our proposed algorithm GELM-AE with other unsupervised learning algorithms including k-means algorithms, spectral clustering (SC) [16, 17], laplacian eigenmaps (LE) [15] and ELM-AE [10].

Table 1: The data sets for clustering.

Data Sets	Cluster	Dimension	Sample size
IRIS	3	4	150
WINE	3	13	178
YALEB	15	1024	165
GLASS	6	10	214
COIL20	20	1024	1440

To assess the feature representation capacity of each algorithm, we employed clustering accuracy (ACC) in our experiments. In particular, ACC is defined as

$$ACC = \frac{\sum_{i=1}^{N} \delta(y_i, map(c_i))}{N} \tag{15}$$

where N is the number of training samples, y_i and c_i are the true label and the predicted label of x_i, respectively. Moreover, $\delta(y_i, map(c_i))$ is a indicator function that equals to 1 if $y_i = map(c_i)$ or 0 otherwise. The function $map(\cdot)$ is an optimal permutation function that maps each cluster label to a category label by the Hungarian algorithm [25]. The number of hidden neurons used in every data set was selected from 500, 1000 and 2000. The hyperparameter in each layer of the basic GELM-AE was selected from the exponential sequence $\{10^{-4}, 10^{-1}, \ldots, 10^{6}\}$ based on the clustering performance. For unsupervised GELM-AE, LE and SC, the same affinity matrix was used but the dimension of the embedded space was selected independently. After setting all the parameters, we conducted 50 trials for each data set. In each trial, k-Means algorithm was implemented in the original space while LE, SC, ELM-AE and GELM-AE were conducted in the embedded spaces, respectively.

Table 2 summarizes the obtained experimental results in 50 trials. For each method, average and std in the table are the mean and standard deviation of the accuracies obtained in 50 experiments, respectively. For the best accuracy, it indicates the highest accuracy in those trials. From Table 2, we can see that GELM-AE achieves the best results on all the data sets in comparison with other methods. To compare the embedding results, we gave a visualized example using IRIS data set. Fig 3 shows the first 2-D of the original data as well as the embedded data outputted from ELM-AE and GELM-AE. The red circles show the wrongly clustered data points by k-means. It can be observed that the data in the original space are not clustered well by k-means algorithms but GELM-AE produces much better results.

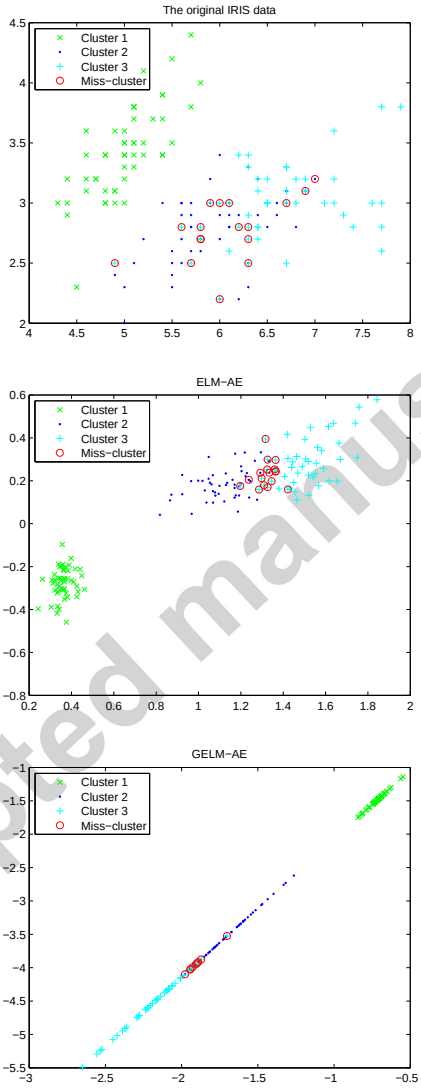


Figure 3: A visualization of the original IRIS data, and the embedded IRIS data generated by ELM-AE and GELM-AE.

Table 2: The mean clustering accuracy of each method.

Data Sets	Accuracy	K-Means	SC	LE	ELM-AE	GELM-AE
IRIS	Average±std	81.81±13.45	82.53±10.99	81.81±16.11	84.44 ± 0.63	92.44 ± 3.94
	Best	89.33	90	90.67	85.33	94
WINE	Average±std	95.89 ± 5.20	92.70 ± 8.34	92.63 ± 8.67	91.87 ± 0.90	96.37 ± 0.78
	Best	96.63	94.38	94.38	94.38	97.19
YALEB	Average±std	41.78 ± 3.65	38.53 ± 2.12	40.65 ± 2.63	42.87 ± 4.44	44.93 ± 3.51
	Best	49.7	43.03	44.85	50.91	50.3
GLASS	Average±std	55.31 ± 0.68	53.50 ± 5.58	53.87 ± 7.74	58.79 ± 7.04	64.55 ± 5.77
	Best	56.07	76.64	75.7	76.17	78.04
COIL20	Average±std	59.88 ± 4.71	54.98 ± 7.44	55.97 ± 6.58	59.39 ± 4.66	61.78 ± 4.99
	Best	69.17	68.4	65.35	68.96	71.25

5.2. ML-GELM algorithms

In this subsection, we compared our proposed ML-GELM algorithm to several other deep models and ELM. ELM is a easy model for semi-supervised learning and many deep models are used to handle the semi-supervised data. In this part of experiments, we took into account deep belief network (DBN) [6], stacked autoencoder (SAE) [8] and multilayer extreme learning machine (ML-ELM) [10], DrELM and DrELM^r [13].

The benchmark data sets are summarized in Table 3, which include five UCI data sets IRIS, WINE, GLASS, LIVER and SATIMAGE [22], one multiclass image classification data set COIL20 [24], and a USPST data set which is a subset (the testing set) of the handwritten digit recognition data set USPS. Training samples are half of each data set and the other half is for testing. To facilitate the comparison, Table 3 also summarizes the characteristics of these data sets.

The experimental setting is as follows. In order to make the comparison fairer, DBN, SAE, ML-ELM and ML-GELM were made to have the same net structure in most data sets (i.e. the same depth and number of hidden nodes). And the number of ELM's hidden nodes is same as that of ML-GELM's first hidden layer nodes. As for DrELM and DrELM^r, we set their parameters by cross-validation. The lost function used in DBN and SAE is cross-entropy error

Table 3: The data sets for classification.

Data Sets	number of classes	Dimension	Sample size
IRIS	3	4	150
WINE	3	13	178
USPST	10	256	2007
LIVER	2	6	345
GLASS	6	10	214
COIL20	20	1024	1440
SATIMAGE	6	36	6435

($L = -\sum_i p_i log\hat{p_i} - \sum_i (1-p_i) log(1-\hat{p_i})$). Besides, the number of hidden neurons, it varied from 1000 to 5000 with the interval of 100 for the most data sets. The specific number of the hidden neurons in every data set was chosen by the cross-validation on the training set. The hyperparameter in ML-GELM was selected from the exponential sequence $\{10^{-7}, 10^{-1}, \ldots, 10^{6}\}$. The used evaluation measure was classification accuracy. For each algorithm, we conducted 30 trials for each data set and reported the corresponding results in Table 4.

Since ML-GELM is based on a deep learning framework, some existing DL methods were taken into account to compare with ML-GELM. The first one is DBN [6] which is a stack of several restricted boltzmann machines(RBMs). The training of DBN involves a two-stage progress. In the first pretraining process, every RBM is trained by contrastive divergence [26] and the output of current RBM is provided as the input of next RBM. In the second fine-tuning process, the whole net is fine-tuned by using back-propagation of error. Stacked autoencoder(SAE) is similar to DBN. However, instead of a RBM, an autoencoder was the building block of SAE. DrELM and DrELM^r model are new deep models that combine the ELM and random projection. The only difference between DrELM and DrELM^r is the computation of X_{i+1} (X_{i+1} is the input data of the (i+1)th ELM). In DrELM, X_{i+1} updates with only the ith layer's prediction results. Nevertheless, X_{i+1} updates with the first i layers' prediction results in DrELM^r. Another two compared algorithms are ELM and

ML-ELM, which have been introduced in Section 2. Table 4 lists the average, standard deviation as well as the best accuracy of each algorithm in 30 trials. The results in Table 4 implies that ML-GELM performs best in most cases.

Table 4: The classification accuracies for each method.

Data Sets	Accuracy	ELM	DBN	SAE	ML-ELM	DrELM	DrELM^r	ML-GELM
IRIS	Average	83.49±2.00	96.67±1.44	95.07±2.95	93.73±2.78	94.62±3.49	95.11±2.65	97.15±0.54
	Best	88.00	98.67	100.00	97.33	100.00	100.00	97.33
WINE	Average	89.73±1.79	96.00±1.83	90.78±0.54	94.56±1.85	96.62±1.92	96.00±2.25	96.67±1.28
	Best	93.33	98.89	91.11	96.67	100.00	98.89	98.89
USPST	Average	91.59±0.75	88.96±0.81	93.69±0.33	89.78±1.58	92.45±0.93	92.50±0.87	93.72±0.48
	Best	93.03	90.25	94.13	92.04	94.53	94.33	94.53
LIVER	Average	54.16±2.64	59.65±0.53	61.97±3.96	58.03±3.61	68.42±3.35	68.54±3.56	70.08±2.96
	Best	59.54	60.69	65.90	64.74	73.99	74.57	75.72
GLASS	Average	73.23±2.14	71.38±0.95	83.95±0.99	75.23±1.83	83.06±4.55	82.87±3.22	82.94±1.31
	Best	78.90	73.39	85.32	78.90	89.91	88.07	84.40
COIL20	Average	94.69±0.92	98.85±0.09	98.81±0.20	96.58±0.70	95.58±1.18	95.59±0.87	99.90±0.09
	Best	96.39	99.03	99.17	97.50	97.92	97.36	100.00
SATIMAGE	Average	87.07±0.50	86.20±0.66	87.67±1.33	85.52±0.53	87.85±0.53	87.82±0.54	88.77±0.44
	Best	88.13	87.36	88.97	86.64	88.79	89.13	89.53
Average		81.99	85.39	87.42	84.78	88.37	88.35	89.89

To compare the time complexity of the considered algorithms, we recorded the running time consumed by them. It is worth pointing out that the configuration of the computer is 3.40GHz with 24GB of memory. Fig.4 presents the running time(measured in terms of seconds) of these algorithms on each data set. It can be observed that ML-GELM is far more efficient than DBN and SAE. Furthermore, ML-GELM spends a little more time than ML-ELM, but slightly less time than DrELM and DrELM^r. However, ML-GELM achieves better classification performance than other deep models as illustrated in Table 4.

6. Conclusions

In this paper we propose a new ELM variant called GELM-AE, which combines the simplicity of ELM-AE with the power of manifold regularization. Compared with LE, SC and ELM-AE, GELM-AE is empirically shown to have stronger ability to extract more suitable features for clustering. We also propose a stacked framework named ML-GELM to learn deep representations via GELM-AE. ML-GELM integrates the stacked generalization principle into the

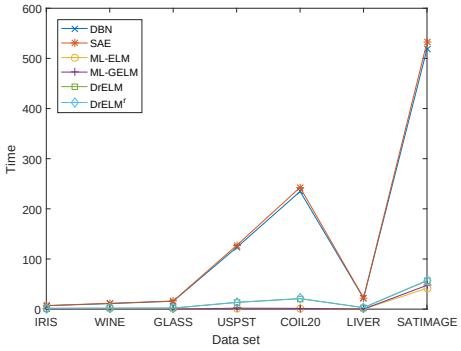


Figure 4: Running time of ML-GELM and other deep model in all data sets

process of deep representation learning. For classification tasks, the experimental results show that ML-GELM performs better than ELM and some other deep models, such as DBN, SAE, ML-ELM, DrELM and DrELM^r. What's more, ML-GELM spends less time than DBN and SAE.

Acknowledgments

This work is supported by the National Basic Research Program of China (973Program No. 2013CB329404), the National Natural Science Foundation of China (No. 61572393, 11501049, 11671317, 11131006).

References

[1] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of International Joint Conference on Neural Network, 2004, pp. 985–990.

[2] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocumputing (2006) 489–501.

[3] A. A. Mohammed, R. Minhas, Q. J. Wu, M. A. Sid-Ahmed, Human face recognition based on multidimensional pca and extreme learning machine, Pattern Recognition 44 (10) (2011) 2588–2597.

[4] C. Pan, D. S. Park, Y. Yang, H. M. Yoo, Leukocyte image segmentation by visual attention and extreme learning machine, Neural Computing and Applications 21 (6) (2012) 1217–1227.

[5] Q. He, X. Jin, C. Du, F. Zhuang, Z. Shi, Clutering in extreme learning machine feature space, Neurocomputing 128 (2014) 88–95.

[6] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.

[7] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Networks 61 (2015) 85–117.

[8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, in: Advances in neural information processing systems, 2007, pp. 153–160.

[9] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (1998) 2278–2324.

[10] L. L. C. Kasun, H. Zhou, G. B. Huang, C. M. Vong, Representational learning with ELMs for big data, IEEE Intelligent Systems 28 (6) (2013) 31–34.

[11] M. D. Tissera, M. D. McDonnell, Deep extreme learning machines: supervised autoencoding architecture for classification, Neurocomputing 174 (2016) 42–49.

[12] H. Larochelle, M. Mandel, R. Pascanu, Y. Bengio, Learning algorithms for the classification restricted boltzmann machine, Journal of Machine Learning Research 13 (2012) 643–669.

[13] W. C. Yu, F. Z. Zhuang, Q. He, Z. Z. Shi, Learning deep representations via extreme learning machines, Neurocomputing 149 (2015) 308–315.

[14] J. Y. Hu, J. S. Zhang, C. X. Zhang, J. Wang, A new deep neural network based on a stack of single-hidden-layer feedforward neural networks with randomly fixed hidden neurons, Neurocomputing 171 (2016) 63–72.

[15] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Computation 15 (6) (2003) 1373–1396.

[16] A. Y. Ng, M. I. Jordan, Y. Weiss, et al., On spectral clustering: Analysis and an algorithm, in: Advances in Neural Information Processing Systems, 2002, pp. 849–856.

[17] U. von Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (4) (2007) 395–416.

[18] W. B. Johnson, J. Lindenstrauss, Extensions of lipschitz mappings into a hilbert space, Contemporary Mathematics 26 (1984) 189–206.

[19] M. Belkin, P. Niyogi, V. Sindhwani, Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, Journal of Machine Learning Research 7 (2006) 2399–2434.

[20] M. Belkin, P. Niyogi, Using manifold stucture for partially labeled classification, in: Advances in Neural Information Processing Systems, 2002, pp. 929–936.

[21] V. Sindhwani, P. Niyogi, M. Belkin, Beyond the point cloud: from transductive to semi-supervised learning, in: Proceedings of the 22nd International Conference on Machine learning, ACM, 2005, pp. 824–831.

[22] M. Lichman, UCI machine learning repository (2013).
URL `http://archive.ics.uci.edu/ml`

[23] K. C. Lee, J. Ho, D. J. Kriegman, Acquiring linear subspaces for face recognition under variable lighting, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2005) 684–698.

[24] S. Melacci, M. Belkin, Laplacian support vector machines trained in the primal, Journal of Machine Learning Research 12 (2011) 1149–1184.

[25] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Courier Corporation, 1982.

[26] G. E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Computation 14 (8) (2002) 1771–1800.