Embedding with Autoencoder Regularization

Wenchao Yu[1,2], Guangxiang Zeng[3], Ping Luo[4], Fuzhen Zhuang[1],
Qing He[1], and Zhongzhi Shi[1]

[1]The Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
[3]University of Science and Technology of China, [4]HP Labs China
{yuwenchao,heqing}@ict.ac.cn,{guangxiang.zeng,ping.luo}@hp.com,
{zhuangfz,shizz}@ics.ict.ac.cn

Abstract. The problem of embedding arises in many machine learning applications with the assumption that there may exist a small number of variabilities which can guarantee the "semantics" of the original high-dimensional data. Most of the existing embedding algorithms perform to maintain the *locality-preserving* property. In this study, inspired by the remarkable success of representation learning and deep learning, we propose a framework of embedding with autoencoder regularization (EAER for short), which incorporates embedding and autoencoding techniques naturally. In this framework, the original data are embedded into the lower dimension, represented by the output of the hidden layer of the autoencoder, thus the resulting data can not only maintain the locality-preserving property but also easily revert to their original forms. This is guaranteed by the joint minimization of the embedding loss and the autoencoder reconstruction error. It is worth mentioning that instead of operating in a batch mode as most of the previous embedding algorithms conduct, the proposed framework actually generates an *inductive* embedding model and thus supports incremental embedding efficiently. To show the effectiveness of EAER, we adapt this joint learning framework to three canonical embedding algorithms, and apply them to both synthetic and real-world data sets. The experimental results show that the adaption of EAER outperforms its original counterpart. Besides, compared with the existing incremental embedding algorithms, the results demonstrate that EAER performs incremental embedding with more competitive efficiency and effectiveness.

Keywords: Embedding, Autoencoder, Representation Learning, Unsupervised Dimensionality Reduction

1 Introduction

In many real-world applications, one is often confronted with overwhelmingly complex features in the raw data and needs to obtain more useful data representations. The manifold hypothesis, that real-world data presented in high-dimensional spaces usually concentrate near a lower-dimensional non-linear manifold, brings a rich geometric perspective to the problem of learning meaningful representations [15]. Illustrated by this assumption, various embedding methods have been developed [19, 24, 1]. However, most of them only focus on the *locality-preserving* property of embedding, namely the relative distance between the points in the high dimension is preserved in the low dimension space.

Recently, deep learning and representation learning attract many research interests with its remarkable success in many applications [9, 2, 3, 23]. In these works, autoencoder is usually adopted as a basic building block to initialize the deep neural network [3, 23]. It is trained to encode the inputs into some representations so that the resulting representations can revert to their original forms. It has been shown that autoencoding is a powerful way to learn the hidden representation of the data.

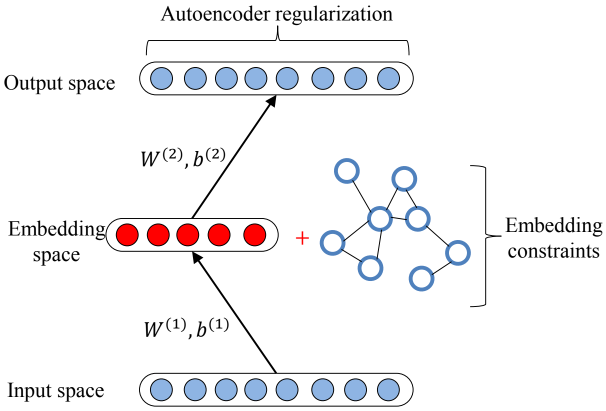


Fig. 1. Illustration of embedding with autoencoder regularization: view from autoencoders

Motivated by the remarkable success of deep learning with autoencoders, we solve the embedding problem collaboratively with autoencoding techniques. Specifically, we propose a framework of embedding with autoencoder regularization (EAER for short) with its two views from autoencoder and embedding respectively. First, from the view of autoencoder in Figure 1, EAER actually trains an autoencoder with the embedding constraints. Specifically, it contains the input layer, hidden layer and output layer from bottom to top. In this framework, besides minimizing the reconstruction error in the original autoencoder, the embedding loss at the hidden layer is also minimized simultaneously. Second, from the view of embedding in Figure 2, each data point with D dimensions is embedded into a d-dimensional space. Simultaneously, it is required that the data points in the hidden space can be recovered to their original form. In other words, the autoencoder is used here as a complementary regularizer to the embedding process. Hopefully, by this joint minimization we can derive the embedding with more semantic representations.

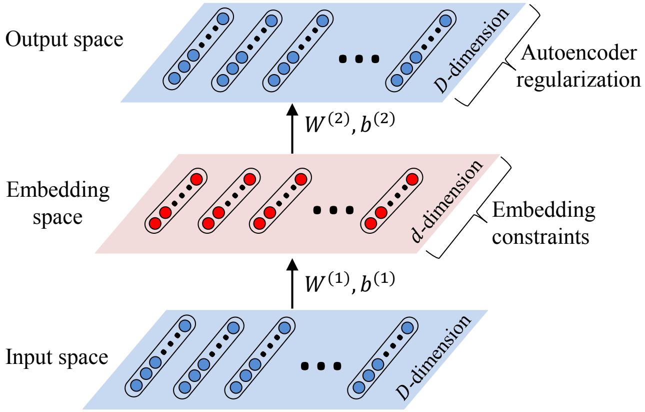


Fig. 2. Illustration of embedding with autoencoder regularization: view from embedding

It should be noted that the training of the proposed framework actually generates an *inductive* embedding model, the function between the input and hidden layers of the autoencoder. It can directly map an instance into the low-dimensional space without accessing the original training data set. Thus, it supports incremental embedding more efficiently compared with most of the existing embedding algorithms which perform in a batch mode.

To show the effectiveness of EAER we adapt this joint learning framework to three well known embedding algorithms, namely Laplacian eigenmaps [1], multi-dimensional scaling [5] and margin-based embedding [7], and apply them to both the synthetic and real-world data sets. The experimental results show that the adaption of EAER outperforms its original counterpart. Also, we demonstrate that compared with the existing incremental embedding algorithms, EAER performs incremental embedding more efficiently with the competitive effectiveness.

In this paper, we describe the EAER framework, along with the details of implementation and performance on a simple synthetic example and real-world data sets. The organization of this paper is as follows: In Section 2, we review the preliminary knowledge of embedding algorithms and autoencoder. In Section 3, we describe EAER framework of learning a low dimensional mapping with autoencoder regularization. In Section 4, we illustrate the algorithm's performance by adapting this joint learning framework to three canonical embedding algorithms. In Section 5, we compare EAER framework to other unsupervised embedding algorithms and discuss several related works. Finally, in Section 6 we conclude this study and mention several directions for future work.

2 Preliminaries

In this section, the summary of general embedding algorithms will be given, followed by a brief review of autoencoders.

2.1 Embedding Algorithms

Many well known embedding algorithms can be described as a rather general form [26]: given the data set $x^{(1)}, ..., x^{(m)}$ find an embedding $f(x^{(i)})$ of each point $x^{(i)}$ by solving the following optimization problem

$$\sum_{1 \le i < j \le m} L(f(W, b; x^{(i)}), f(W, b; x^{(j)}), \varphi_{ij}) \tag{1}$$

where $f(W, b; x) \in R^d$ is the embedding result for a given input $x \in R^D$. $L(\cdot)$ is the loss function between pairs of inputs. φ_{ij} is the weight between $x^{(i)}$ and $x^{(j)}$. We define $\varphi_{ij} = 0$ if $i = j$. For certain loss function $L(\cdot)$ such as Equation (2), constraints may need to remove arbitrary factors in the embedding.

To compute the parameter set φ, one can construct an adjacency graph by putting an edge between between $x^{(i)}$ and $x^{(j)}$ if they are "similar". The similarity of any two data points can be evaluated with k-nearest neighbors (kNN) or ε-neighborhoods. Nodes $x^{(i)}$ and $x^{(j)}$ are connected by an edge if $x^{(i)}$ is among k nearest neighbors of $x^{(j)}$. If one chooses ε-neighborhoods metric, nodes are connected by an edge if $\|x^{(i)} - x^{(j)}\|^2 < \varepsilon$ where the norm is usually the Euclidean norm. The edges are weighted with Euclidean distance $\varphi_{ij} = \|x^{(i)} - x^{(j)}\|^2$ or Gaussian kernel $\varphi_{ij} = e^{-\frac{\|x^{(i)} - x^{(j)}\|^2}{\tau}}$ $(\tau \in R)$ if $x^{(i)}$ and $x^{(j)}$ are connected. Another simple way for weighting the edges is to set $\varphi_{ij} = 1$ if nodes are connected, otherwise $\varphi_{ij} = 0$.

We consider the following embedding algorithms which fit into this framework.

Laplacian Eigenmaps. Laplacian eigenmaps (LE) [1] is a coherent framework for embedding by emphasizing the preservation of the locality. One constructs the adjacency graph of input samples and encodes them into d-dimensional Euclidean space. The embedding is given by the $d \times m$ matrix $\mathcal{F} = [f_1, f_2, ..., f_m]$, f_i is short for $f(W, b; x^{(i)})$. $L = D - \varphi$ is the Laplacian matrix where D is a diagonal weight matrix $D_{ii} = \sum_j \varphi_{ji}$. Then, we need to minimize

$$\sum_{i<j} L(f_i, f_j, \varphi_{ij}) = \sum_{i<j} \|f_i - f_j\|^2 \varphi_{ij} = \mathrm{Tr}(\mathcal{F} L \mathcal{F}^T) \tag{2}$$

subject to: $\mathcal{F} D \mathcal{F}^T = I$ and $\mathcal{F} D 1 = 0$.

Multidimensional Scaling. Multidimensional scaling (MDS) [5] is a canonical form of linear embedding that attempts to find an embedding from the input data into a low-dimensional space such that distances between data points are preserved. Usually, MDS is formulated as an optimization problem

$$\sum_{i<j} L(f_i, f_j, \varphi_{ij}) = \sum_{i<j} (\|f_i - f_j\| - \varphi_{ij})^2 \tag{3}$$

Kernel PCA can be interpreted as a form of metric MDS when the kernel function is isotropic [27].

Isomap [24] is one of several widely used low-dimensional embedding methods which works by defining the geodesic distance to be the sum of edge weights along the shortest path between two nodes and then performs low-dimensional embedding with classical MDS based on the pairwise distance between data points.

Margin-based Embedding. The margin-based loss function proposed for learning a globally coherent nonlinear function that maps the data evenly to the output manifold and relies solely on neighborhood relationships [7]. The following optimization is used:

$$L(f_i, f_j, \varphi_{ij}) = \begin{cases} \|f_i - f_j\|^2 & \text{if } \varphi_{ij} = 1 \\ \max(0, l - \|f_i - f_j\|^2) & \text{if } \varphi_{ij} = 0 \end{cases} \tag{4}$$

which ensures that the data in the embedding space have a distance of at least l from each other when they are similar in the input space. In our experiments l is set to 1. The weight of edges $\varphi_{ij} = 1$ if $x^{(i)}$ and $x^{(j)}$ are connected, otherwise $\varphi_{ij} = 0$.

2.2 Autoencoders

An autoencoder neural network is an unsupervised learning algorithm that applies back-propagation [20], setting the target values to be equal to the inputs. In terms of embedding, the network learns to encode the inputs into a small number of dimensions and then decode it back into the original space. Specifically, given an unlabeled data set $x^{(i)}, i = 1, ..., m$, we want to learn representations

$$f(W^{(1)}, b^{(1)}; x^{(i)}) = \sigma(W^{(1)} x^{(i)} + b^{(1)}) \tag{5}$$

such that the output hypotheses

$$h(W, b; x^{(i)}) = \sigma(W^{(2)} f(W^{(1)}, b^{(1)}; x^{(i)}) + b^{(2)}) \tag{6}$$

is approximately $x^{(i)}$. Thus we use L_2 norm to minimize the reconstruction error $J(W, b; x)$

$$J(W, b; x) = \frac{1}{2} \sum_{i=1}^{m} \|h(W, b; x^{(i)}) - x^{(i)}\|^2 \tag{7}$$

We consider sparse autoencoder with sparsity parameter ρ and penalize it with the Kullback-Leibler (KL) divergence [10]. We then define the overall cost function to be

$$J(W, b; x) + \beta \sum_{j=1}^{d} \mathrm{KL}\left(\rho \| \hat{\rho}_j\right) + \frac{\lambda}{2} \|W\|^2 \tag{8}$$

where $\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m} f_j(W^{(1)}, b^{(1)}; x^{(i)})$ is the average activation of hidden unit j; m is the number of inputs and d is the number of hidden units; The last term is a weight decay term that tends to decrease the magnitude of the weights, and helps prevent over-fitting. β and λ control the weight of the corresponding penalty terms;

$$\mathrm{KL}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{9}$$

is the KL divergence between Bernoulli random variables with mean ρ and $\hat{\rho}_j$ respectively.

3 Learning a Low Dimensional Mapping with Autoencoder Regularization

We consider the problem of finding a function that maps high-dimensional input data to lower-dimensional representations given the neighborhood relationships between the data in the input space.

3.1 Embedding with Autoencoder Regularization

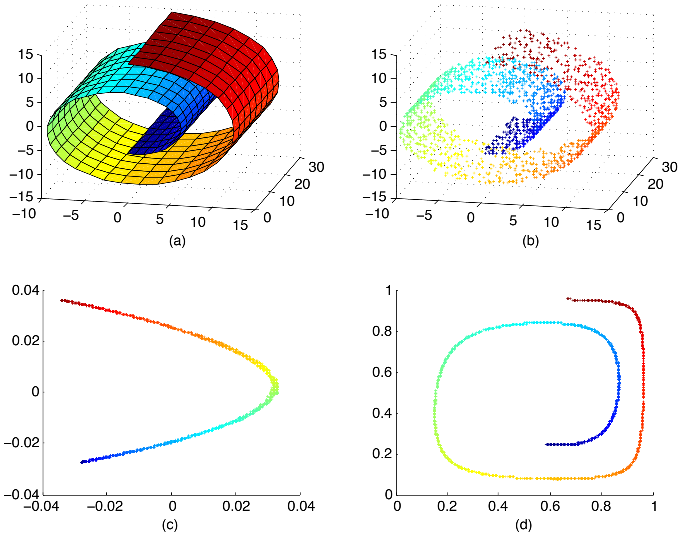
We would like to use the ideas developed in autoencoding for embedding. The general approach we propose for EAER is to add an autoencoder regularizer to the embedding optimization function. As shown in Figure 1 and 2, we aim to simultaneously minimize the autoencoder reconstruction error at the output layer and the embedding loss in the hidden layer. The general form of this joint loss function is as follows:

$$\begin{aligned} J_{em}(W, b, \varphi; x) = &\sum_{1 \le i < j \le m} L(f(W^{(1)}, b^{(1)}; x^{(i)}), f(W^{(1)}, b^{(1)}; x^{(j)}), \varphi_{ij}) \\ &+ \gamma J(W, b; x) + \beta \sum_{j=1}^{d} \mathrm{KL}\,(\rho \| \hat{\rho}_j) + \frac{\lambda}{2}\|W\|^2 \end{aligned} \tag{10}$$

where $L(\cdot)$ is the embedding loss function between pairs of the data (its detailed form can be any of the functions (2), (3) and (4) for different embedding algorithms). Here, $f(W^{(1)}, b^{(1)}; x^{(i)})$ actually maps $x^{(i)}$ to the lower dimension; $J(W, b; x)$ is the autoencoder reconstruction error defined by Equation (7); the last two terms are sparsity penalty term and weight decay term discussed in Section 2.2; γ, β and λ control the balance between these penalty terms. The idea that injecting an autoencoder regularization may help to guide the embedding towards better data representations, and we use a synthetic swiss roll example to illustrate this conjecture.

3.2 A Case Study on Synthetic Data

The swiss roll, considered in [1, 24, 19], is a flat two-dimensional sub-manifold of R^3 which is shown in Figure 3(a), and the data set of 2000 points chosen at random from the swiss roll is shown in Figure 3(b). We build the adjacency graph



(a)

(b)

(c)

(d)

Fig. 3. A synthetic swiss roll example. (a) the synthetic swiss roll manifold, (b) 2000 points chosen at random from the swiss roll, (c) embedding result of LE, (d) embedding result of EAER-LE.

with 8 nearest neighbors, and set the weight $\varphi_{ij} = 1$ if node i is among the 8 nearest neighbors of node j, otherwise $\varphi_{ij} = 0$. For EAER-LE (the EAER framework applied to the LE algorithm), we have the following parameter settings in Equation (10): $\gamma = 0.65$, $\lambda = 0.003$, and $\beta = 0$. We compute the two-dimensional representations by LE and EAER-LE, and their results are respectively shown in Figure 3(c) and Figure 3(d).

The curve in Figure 3(c) is only a half ellipse, while the curve in Figure 3(d) maintains the roll in the two-dimension space. Thus, it is obvious that the result of EAER-LE is more "semantic" since it preserves the curve of the original manifold together with the locality properties. As shown in Section 4, we can also obtain such meaningful results when EAER is applied to the other embedding algorithms.

3.3 Model Learning

Our goal is to minimize $J_{em}(W, b, \varphi; x)$ as a function of W, b and φ. One can construct a weighted graph by the method of k-nearest neighbors (kNN) or ε-neighborhoods to compute φ and then train this regularized neural network parameterized by W and b. The key step of model learning is computing the partial derivatives $\frac{\partial}{\partial W^{(l)}} J_{em}(W, b, \varphi; x)$ and $\frac{\partial}{\partial b^{(l)}} J_{em}(W, b, \varphi; x)$ with respect to the input x. We will use an efficient way to compute the partial derivatives in light of the intuition behind the back-propagation algorithm [20]. In order to measure how much the nodes of the same layer is "responsible" for the errors of the output hypotheses, we introduce an “error term” vector δ, and define $\delta^{(1)}$ and $\delta^{(2)}$ for the hidden layer and output layer respectively. To incorporate the KL-divergence term into the derivative calculation, $\delta^{(1)}$ and $\delta^{(2)}$ are computed as follows [16]:

$$\delta^{(1)} = \left(\left(W^{(2)} \right)^T \delta^{(2)} + \beta \frac{\hat{\rho} - \rho_0}{\hat{\rho}(1 - \rho_0)} \right) \cdot \sigma'(z^{(1)}) \qquad (11)$$

$$\delta^{(2)} = \frac{\partial}{\partial z^{(2)}} \frac{1}{2} \| h(W, b; x) - x \|^2 = - \left(h(W, b; x) - x \right) \cdot \sigma'(z^{(2)}) \qquad (12)$$

where $\hat{\rho} = \frac{1}{m} \sum_{i=1}^{m} f(W^{(1)}, b^{(1)}; x^{(i)})$ is the average activation of embedding layer; $\rho_0 \in R^d$ is a vector with all entries ρ; “$\cdot$” denotes the element-wise product operator; $z^{(1)} = W^{(1)} x + b^{(1)}$ and $z^{(2)} = W^{(2)} f(W^{(1)}, b^{(1)}; x) + b^{(2)}$. In detail, the procedure can be described in Algorithm 1.

```
Algorithm 1 Partial Derivatives Computation
Input: The input sample $x$
Output: Partial derivatives of the function defined by Equation (10): $\frac{\partial}{\partial W^{(l)}} J_{em}(W, b, \varphi; x)$ and $\frac{\partial}{\partial b^{(l)}} J_{em}(W, b, \varphi; x)$.

1. Randomly initialized $W^{(l)}$ and $b^{(l)}$, $(l = 1, 2)$.
2. Perform a feedforward pass, computing the activations for the embedding layer
   and output layer.
3. For the output layer, compute $\delta^{(2)}$ by Equation (12).
4. Compute $\delta^{(1)}$ by Equation (11).
5. Compute the partial derivatives:
   $\frac{\partial}{\partial W^{(2)}} J_{em}(W, b, \varphi; x) = \gamma \delta^{(2)} \left( f(W^{(1)}, b^{(1)}; x) \right)^T + \lambda W^{(2)}$;
   $\frac{\partial}{\partial b^{(2)}} J_{em}(W, b, \varphi; x) = \gamma \delta^{(2)}$;
   $\frac{\partial}{\partial W^{(1)}} J_{em}(W, b, \varphi; x) = \frac{\partial}{\partial W^{(1)}} \sum_{ij} L(\cdot) + \gamma \delta^{(1)} x^T + \lambda W^{(1)}$;
   $\frac{\partial}{\partial b^{(1)}} J_{em}(W, b, \varphi; x) = \frac{\partial}{\partial b^{(1)}} \sum_{ij} L(\cdot) + \gamma \delta^{(1)}$ where
   $L(\cdot) = L\left( f(W^{(1)}, b^{(1)}; x^{(i)}), f(W^{(1)}, b^{(1)}; x^{(j)}), \varphi_{ij} \right)$.
```

In Step 5 of Algorithm 1, we compute $\frac{\partial}{\partial W^{(1)}} \sum_{ij} L(\cdot)$ according to its concrete form of different embedding algorithms, such as LE, MDS and margin-based embedding. For certain embedding loss function, the gradient can be incorporated into the “error term” $\delta^{(1)}$ in order to speed up the algorithm. Pseudocode of the full approach is given in Algorithm 2, where α is the learning rate. To train this model, we can now repeatedly take steps of gradient descent to reduce our cost function $J_{em}(W, b, \varphi; x)$. Note that EAER is a general framework, which can be adapted for different embedding algorithms. In the experiments we adapt the framework to LE, MDS and the margin-based embedding algorithm for comparison.

Algorithm 2 Algorithm for EAER Framework

```
Input: The input data set $\{x^{(i)}\}_{i=1}^m$.
Output: Results of embedding layer $f(W^{(1)}, b^{(1)}; x)$.

1. Construct the adjacency graph of $\{x^{(i)}\}_{i=1}^m$ and compute $\varphi_{ij}$.
2. while not stopping criterion do
3.     Set $\varDelta W^{(l)} = 0$, $\varDelta b^{(l)} = 0$ for all $l = 1, 2$.
4.     Use Algorithm 1 to compute $\frac{\partial}{\partial W^{(l)}} J_{em}(W, b, \varphi; x^{(i)})$ and $\frac{\partial}{\partial b^{(l)}} J_{em}(W, b, \varphi; x^{(i)})$
   for all $x^{(i)}$.
5.     Compute $\varDelta W^{(l)} = \sum_{i=1}^m \frac{\partial}{\partial W^{(l)}} J_{em}(W, b, \varphi; x^{(i)})$;
6.     Compute $\varDelta b^{(l)} = \sum_{i=1}^m \frac{\partial}{\partial b^{(l)}} J_{em}(W, b, \varphi; x^{(i)})$.
7.     Update: $W^{(l)} = W^{(l)} - \alpha \left( \frac{1}{m} \varDelta W^{(l)} \right)$, $b^{(l)} = b^{(l)} - \alpha \left( \frac{1}{m} \varDelta b^{(l)} \right)$.
8. end while
9. Compute the embedding results $f(W^{(1)}, b^{(1)}; x)$.
```

3.4 Incremental Embedding with EAER

Most of the embedding algorithms operate in a "batch" mode. That is, all data need to be available during training. If the new data come, the naive method is to re-run the training on the union of the original and new data, which prohibitively involves with expensive computing. To address this point, some incremental version of embedding algorithms, such as incremental Isomap [12] and incremental locally linear embedding (LLE) [22], were proposed. Their basic idea is to identify the k-nearest neighbors in the original training data for the new point and use these neighbors to represent the new one in the embedding space. It is clear that the original training data must be accessed in these methods of incremental embedding.

As to the EAER framework it is naturally an inductive embedding model. After all the model parameters are learned, any instance x can be embedded to $f(W^{(1)}, b^{(1)}; x)$ in the lower dimension directly without accessing the original training data. It should be noted that, when we use the Sigmoid function as the activation function, the input data need to be normalized to the range $[0, 1]$. Therefore, it is convenient to apply incremental embedding to those data sets with known data intervals, say MNIST [13], with each element ranging from 0 to 255.

Assume that we have m points for training and n for testing. EAER only takes linear time $O(n)$ to compute the low-dimensional embedding for the new n points. However, the incremental versions of the existing embedding algorithms need to compute its k-nearest neighbors among the training set. Thus, the overall time complexity of these algorithms is $O(mn)$ (assuming the linear scanning method for k-nearest neighbors is adopted here). Therefore, EAER dramatically reduces the running time for incremental embedding. The experiments in Section 4 also show the effectiveness of EAER for incremental embedding.

4 Experimental Evaluation

To verify the embedding performance of EAER framework, we conduct the experiments on various kinds of benchmark data sets. We begin with a simple synthetic example to intuitively show the embedding performance of EAER in Section 3. In this section, we build softmax classifiers [6] based on the embedded features of the real-world data sets and check the performance on classification accuracy. Then, we compare EAER with the incremental version of the baseline methods.

4.1 Benchmark Data Sets and Baseline Methods

The benchmark data sets are summarized in Table 1. One is MNIST [13] and the other six are taken from the UCI repository [4]. All these data sets are provided with the class labels.

Table 1. Data sets description

#	Data Sets	#Instances	#Attributes	#Classes
1	Iris	150	4	3
2	Wine	178	13	3
3	Glass	214	9	6
4	Diabetes	768	8	2
5	Segment	2310	19	7
6	Satimage	6435	36	6
7	MNIST	70000	784	10

In our experiments, we adapt the proposed EAER framework to the following embedding algorithms.

- **LE**: Laplacian Eigenmaps with the loss function defined by Equation (2) [1];
- **MDS**: Classical multidimensional scaling [5] with the loss function in Equation (3), where the norm adopted is the Euclidean distance;
- **Margin-based**: Embedding with the margin-based loss function defined by Equation (4) [7].

The embedding algorithms adapted to EAER are denoted as EAER-LE, EAER-MDS and EAER-Margin respectively.

4.2 Experimental Results on Real-world Data Sets

The experiments are conducted on 7 real-world data sets listed in Table 1. For each data set we first apply any embedding algorithm to it and then build the classification model on the resultant features. The classification accuracy on the training data is used as the evaluation measure for embedding.

The model parameters are set as follows. In the joint loss function of Equation (10), we set $\rho = 0.1$, $\beta = 0.2$, $\lambda = 0.003$, and W, b are randomly initialized. We set the weight $\varphi_{ij} = 1$ if node i is among the nearest neighbors of node j, otherwise $\varphi_{ij} = 0$. Then, for the rest three parameters, namely the number d of hidden units, the number k of the nearest neighbors in the adjacency graph and the weight γ of the autoencoder regularizer, their ranges are given as follows. γ is sampled from 0.1 to 1 with the interval of 0.1, k varies from 2 to 10 with the interval of 1, and d is set as $2 \leq d \leq \frac{D+1}{2}$ for the UCI data sets and $2 \leq d \leq 20$ for the MNIST data set, where D is the dimension of the original data sets. The data values are all normalized on each feature.

Table 2. Results on the data sets described in Table 1. We report the best accuracy for each method over the parameter ranges. For MNIST, we randomly select 1000 and 5000 samples and evaluate the performances on them respectively. The last column "Original" is the classification results on the original data sets.

Methods	LE	EAER-LE	MDS	EAER-MDS	Margin-based	EAER-Margin	Original
Iris	0.8933	0.9467	0.9400	0.9733	0.9667	**0.9800**	0.9467
Wine	0.9663	0.9719	0.9831	**0.9944**	0.9775	0.9888	**0.9944**
Glass	0.5654	0.5374	0.5935	0.6916	0.6355	**0.6963**	0.6075
Diabetes	0.6680	0.6576	0.7083	0.7552	0.7578	0.7669	**0.7813**
Segment	0.7758	0.8823	0.6390	0.7643	0.9028	**0.9443**	0.9130
Satimage	0.8362	0.8410	0.7984	0.8413	0.8522	**0.8738**	0.8578
MNIST^{1k}	0.8140	0.9520	0.8320	0.9020	0.9510	0.9840	**1.0000**
MNIST^{5k}	0.8370	0.9340	0.8520	0.9260	0.9370	0.9900	**1.0000**
Average	0.7945	**0.8404**	0.7933	**0.8560**	0.8726	**0.9030**	0.8876

For each embedding algorithm we train the softmax classifier on the resultant features and the original data sets, and record its classification accuracy on these training data. Among the parameter ranges, the results with best training accuracy are reported in Table 2. It shows that the EAER adaption is usually better than its original counterpart except that on the two data sets of *Glass* and *Diabetes*, EAER-LE is slightly worse than LE. On the whole, EAER increases the average accuracy over all the data sets by 4.59%, 6.27%, 3.04% compared with the corresponding three baseline methods respectively. It is worth mentioning that the average accuracy of EAER-Margin is better than the classification performance with original data sets, which again verifies EAER can learn better semantic representations.

Figure 4 shows the average accuracy of EAER-LE, EAER-MDS and EAER-Margin (denoted as "EAER" in the figure) and baseline methods (denoted as "Embedding" in the figure) when the embedding layer dimensionality d varies. The parameter setting is identical to previous experiment. Figure 4 demonstrates that when the embedding layer dimensionality d increases, the embedding accuracy increases. Yet we also observed that embedding methods adapted to

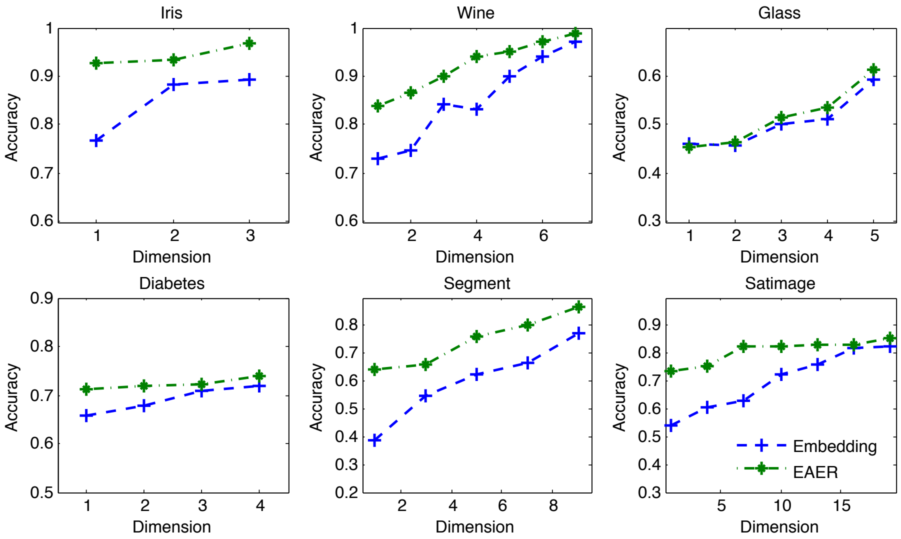


Fig. 4. The average accuracy of EAER-LE, EAER-MDS and EAER-Margin (denoted as "EAER" in the figure) and baseline methods (denoted as "Embedding" in the figure) when the embedding layer dimensionality d varies. The experiment was conducted on six UCI data sets.

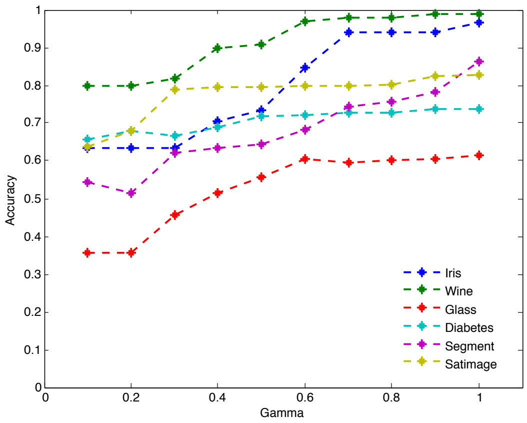


Fig. 5. The average accuracy of embedding methods adapted to EAER (EAER-LE, EAER-MDS and EAER-Margin) on each data set when γ changes. γ is sampled from 0.1 to 1 with the interval of 0.1, the experiment was conducted on six UCI data sets.

EAER framework led to comparatively higher accuracy even when d is small. So to speak, embedding methods adapted to EAER framework is able to preserve more data information. Figure 5 shows the average accuracy of embedding methods adapted to EAER when γ changes. As can be seen from the figure, when γ is small, the accuracy increases as γ goes up. When γ reaches a certain value (usually 0.6 to 0.8, the value varies according to different data sets), the result becomes comparatively stable. However, larger γ does not indicate better result. When its value exceeds a certain value (say 3 or 5), the accuracy begins to fall off (not shown in this figure).

4.3 Incremental Embedding Results

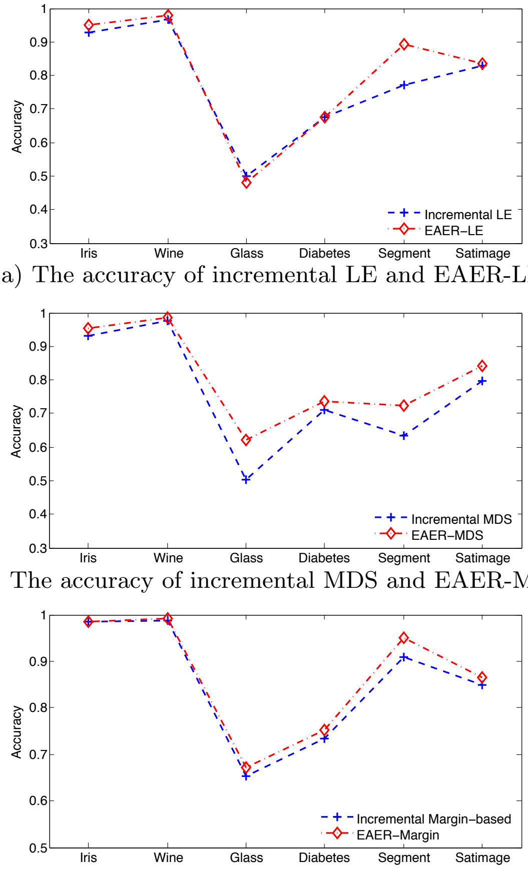
The EAER framework is an inductive embedding model. After all the model parameters are learned, any instance x can be embedded to a lower-dimensional space via $f(W^{(1)}, b^{(1)}; x)$. Thus, EAER can naturally handle incremental embedding. Since LE, MDS and Margin-based methods all perform in a batch mode, we adapt them to handle new data as follows, similar to the method in [22]. For a new instance we first identify its k-nearest neighbors in the original data and then construct the output by combining the embedded features of these neighbors.

To evaluate the performance of incremental embedding, each data set is randomly divided into two parts for training and testing respectively. On the training data we first apply the embedding algorithm and then train a classifier based on the resultant features. Next, for each instance in the testing data we apply the incremental embedding on it and then use the classifier to test on its embedded features. Thus, the classification accuracy on the testing data can be used as the evaluation measure for incremental embedding. We randomly sample the training and testing data for 10 times and average the testing accuracy values for each method. All the results are shown in Figure 6. We can find that EAER achieves highest accuracy among 16 of the whole 18 cases, and 1 tie-break.

5 Related Work

Two canonical forms of linear embedding are eigenvector methods of principle component analysis (PCA) [11] and multidimensional scaling (MDS) [5]. Recently there have been lots of nonlinear approaches to compute a low-dimensional embedding, including Isomap [24], LLE [19], Laplacian eigenmaps [1], margin-based embedding algorithms [7] and their variants [17, 8]. The proposed EAER framework is different from these methods in the following aspects. First, with the autoencoder regularization it can be used as a general approach to complement any existing embedding method. Second, it embeds the new data using the resultant inductive model without accessing the original training data.

Autoencoders are usually used as basic building blocks to train the deep neural network [3] and currently variants of autoencoder have been investigated, such as contractive autoencoders [18] and denoising autoencoders [25]. These



(a) The accuracy of incremental LE and EAER-LE

(b) The accuracy of incremental MDS and EAER-MDS

(c) The accuracy of incremental Margin-based embedding and EAER-Margin

Fig. 6. The accuracy of LE, MDS, Margin-based embedding and the corresponding methods adapted to EAER

are often called regularized autoencoders, where some regularization terms are proposed to improve the data reconstruction performance. However, in the proposed framework, an autoencoder as a whole is used as a regularizer to improve the embedding algorithms.

There are also some other works related to EAER. First, in [26], the embedding-based regularizer is plugged into the layers of deep architectures as an auxiliary task for semi-supervised embedding. The focus of this work is on semi-supervised learning, while ours is for unsupervised embedding with the autoencoder regularizer. Second, in [14] the stacked restricted Boltzmann machines (RBMs) are pre-trained and then fine-tuned with the supervised embedding constraints. It is clear that this approach is a disjoint way of autoencoding and embedding while our method trains them jointly. In [21], a multilayer neural network is pre-trained and fine-tuned to learning a nonlinear embedding by preserving class neighborhood structure. However, EAER is a general framework which compatible with different types of embedding algorithms.

Often we need to generalize the embedding results for the new data. LLE [19] was extended to its incremental version [22] by identifying the k-nearest neighbors of the new input and construct its output with its neighbors. Also, the incremental version of Isomap was proposed [12]. EAER naturally generates an inductive embedding model, whereas the methods mentioned embed new data in a transductive way.

6 Conclusion

In this paper we proposed an *embedding with autoencoder regularization* (EAER) framework for unsupervised nonlinear dimensionality reduction. By minimizing the embedding loss and the autoencoder reconstruction error simultaneously, EAER can learn more semantic representations of the inputs. We adapt the framework to the embedding algorithms of Laplacian eigenmaps, multidimentional scaling and margin-based method, and the results demonstrate that the embedding methods adapted to EAER outperform the original counterparts when applying the embedding codes to the classification tasks. The EAER framework proposed in the paper is naturally an inductive model, thus can embed the new data more efficiently. We plan to further investigate the performance of EAER by extending it to deep architectures or combining the advanced autoencoders, such as contractive autoencoders and denoising autoencoders.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (No. 61175052, 61203297, 60933004, 61035003), National High-tech R&D Program of China (863 Program) (No. 2013AA01A606, 2012AA011003), and National Program on Key Basic Research Project (973 Program) (No. 2013CB329502). Thanks to the internship program of Hewlett Packard Labs China.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. In: Neural computation. pp. 1373–1396 (2003)
2. Bengio, Y., Courville, A., Vincent, P.: Unsupervised feature learning and deep learning: A review and new perspectives. In: CoRR (2012)
3. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems. pp. 153–160 (2007)
4. Blake, C., Merz, C.: Uci repository of machine learning databases (1998) (1998)
5. Cox, T., Cox, M.: Multidimensional scaling. Chapman & Hall, London, UK (1994)
6. Greene, W., Zhang, C.: Econometric analysis. In: Prentice hall Upper Saddle River (1997)

7. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 1735–1742 (2006)
8. He, X., Cai, D., Yan, S., Zhang, H.: Neighborhood preserving embedding. In: Tenth IEEE International Conference on Computer Vision. pp. 1208–1213 (2005)
9. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science pp. 504–507 (2006)
10. Hinton, G.: A practical guide to training restricted boltzmann machines. Momentum (2010)
11. Jolliffe, I.: Principal component analysis. In: Springer, New York (1986)
12. Law, M., Jain, A.: Incremental nonlinear dimensionality reduction by manifold learning. IEEE Trans. Pattern Analysis and Machine Intelligence pp. 377–391 (2006)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. pp. 2278–2324 (1998)
14. Min, R., van der Maaten, L., Yuan, Z., Bonner, A., Zhang, Z.: Deep supervised t-distributed embedding. In: Proceedings of the 27th International Conference on Machine Learning (2010)
15. Narayanan, H., Mitter, S.: Sample complexity of testing the manifold hypothesis. In: Advances in Neural Information Processing Systems. pp. 1786–1794 (2010)
16. Ng, A.: Cs294a lecture notes: Sparse autoencoder. In: Stanford University (2010)
17. Niyogi, X.: Locality preserving projections. In: Advances in Neural Information Processing Systems. pp. 153–160 (2004)
18. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contractive auto-encoders: Explicit invariance during feature extraction. In: Proceedings of the 28th International Conference on Machine Learning (2011)
19. Roweis, S., Saul, L.: Nonlinear dimensionality reduction by locally linear embedding. Science pp. 2323–2326 (2000)
20. Rumelhart, D., Hintont, G., Williams, R.: Learning representations by back-propagating errors. Nature pp. 533–536 (1986)
21. Salakhutdinov, R., Hinton, G.: Learning a nonlinear embedding by preserving class neighbourhood structure. In: AI and Statistics (2007)
22. Saul, L., Roweis, S.: Think globally, fit locally: unsupervised learning of low dimensional manifolds. In: The Journal of Machine Learning Research. pp. 119–155 (2003)
23. Socher, R., Pennington, J., Huang, E.H., Ng, A.Y., Manning, C.D.: Semi-supervised recursive autoencoders for predicting sentiment distributions. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. pp. 151–161 (2011)
24. Tenenbaum, J., De Silva, V., Langford, J.: A global geometric framework for non-linear dimensionality reduction. Science pp. 2319–2323 (2000)
25. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on Machine learning (2008)
26. Weston, J., Ratle, F., Collobert, R.: Deep learning via semi-supervised embedding. In: Proceedings of the 25th International Conference on Machine Learning (2008)
27. Williams, C.: On a connection between kernel pca and metric multidimensional scaling. In: Machine Learning. pp. 11–19. Springer (2002)