Soft Computing
https://doi.org/10.1007/s00500-021-05934-8

FOCUS

Deep clustering based on embedded auto-encoder

Xuan Huang[1,2] · Zhenlong Hu[3,4] · Lin Lin[5]

Accepted: 2 June 2021

Abstract
Deep clustering is a new research direction that combines deep learning and clustering. It performs feature representation and cluster assignments simultaneously, and its clustering performance is significantly superior to traditional clustering algorithms. The auto-encoder is a neural network model, which can learn the hidden features of the input object to achieve nonlinear dimensionality reduction. This paper proposes the embedded auto-encoder network model; specifically, the auto-encoder is embedded into the encoder unit and the decoder unit of the prototype auto-encoder, respectively. To ensure effectively cluster high-dimensional objects, the encoder of model first encodes the raw features of the input objects, and obtains a cluster-friendly feature representation. Then, in the model training stage, by adding smoothness constraints to the objective function of the encoder, the representation capabilities of the hidden layer coding are significantly improved. Finally, the adaptive self-paced learning threshold is determined according to the median distance between the object and its corresponding the centroid, and the fine-tuning sample of the model is automatically selected. Experimental results on multiple image datasets have shown that our model has fewer parameters, higher efficiency and the comprehensive clustering performance is significantly superior to the state-of-the-art clustering methods.

Keywords Deep clustering · The embedded auto-encoder · Feature representation

1 Introduction

Clustering is an unsupervised machine learning technique that uses similarity or distance metrics to measure unlabeled objects, and categorize the objects into one cluster based on the measurement results. In the initial stage of data analysis, clustering is very suitable for exploring the underlying structure of the datasets that get a well-classified subset of objects, automatically learn the object labels. Therefore, clustering is an important preprocessing technique in machine learning and it is the core of many data-driven based learning technologies.

The traditional clustering techniques are roughly categorized into the partition-based approaches, the density-based approaches, the hierarchical-based approaches and so on. The partition-based clustering algorithms usually need to the number of clusters or initial centroids in advance, and iteratively group objects into the same clusters. K-Means and its variants belong to this category of approach, which is very simple and easy to implement. They have good performance on multiple datasets, but they are powerless serve for samples with non-convex samples. The density-based clustering mainly groups objects according to its density distribution; they can cluster any

Communicated by Suresh Chandra Satapathy.

✉ Zhenlong Hu
1179129@mail.dhu.edu.cn

Xuan Huang
christy.huang@my.swjtu.edu.cn

Lin Lin
narcia_lin2006@163.com

[1] The School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China

[2] Chengdu College of University of Electronic Science and Technology of China, Chengdu 611731, China

[3] Jiyang College of Zhejiang A&F University, Zhuji 311800, Zhejiang, China

[4] Zhejiang Yuexiu University, Shaoxing 312000, Zhejiang, China

[5] College of Information Engineering, Chengdu Aeronautic Polytechnic, Chengdu 610100, China

Published online: 18 June 2021

distributed objects. The disadvantage of this category approach is that it is very sensitive to parameter settings, especially for large capacity datasets, the convergence time will become longer. The hierarchical clustering approaches are mainly classified into two categories: the bottom-up and the top-down. The bottom-up hierarchical clustering is also termed the agglomerative clustering, and most hierarchical clustering algorithms fall into this category. These approaches treat each object as a cluster, and in each iteration, the two most similar clusters are selected and merged into a new cluster according to the specific conditions, and then calculate the centroid of the new cluster. Repeat this process until the required number of clusters is obtained. The hierarchical clustering method can show the clustering results of data objects on different scales, but its computational complexity is relatively high, and the singular value has a greater impact on the clustering results.

With the advancement of hardware technology, the performance of data acquisition equipment has improved significantly. It is easier to obtain various types of information, the data volume increases explosively, and the data also exhibit high-dimensional characteristics. The increase of the data dimension makes the search feature space increase exponentially, and the model complexity increases significantly, so the corresponding analysis approaches become more complicated. The traditional clustering methods consider the entire feature set of the object to perform clustering. It is difficult for them to find meaningful clusters in the high-dimensional feature space, which leads to the curse of dimensionality (Huang et al. 2019). For high-dimensional data, traditional clustering methods first map the objects from the raw feature space to the low-dimensional feature space or after feature transformation and then clustering. However, these methods perform linear embedding learning on the data, which may result in the loss of the important features. The result of dimensionality reduction or feature transformation directly affects the effect of clustering.

For these reasons, deep learning has become popular in clustering and has become the key technology for clustering high-dimensional data. The deep neural networks (DNNs) is the neural network with at least one hidden layer, which can simulate complex nonlinear relationships. The DNNs is the application basis of many artificial intelligence technologies and has outstanding performance in computer vision, speech recognition, text mining and other fields. The rise of deep learning, its combination with clustering has produced a new research direction, which is deep clustering. The deep clustering approaches extract the higher-level features from the raw features of the input object through DNNs, then learn an effective feature representation from a large amount of input objects, and then use the learned features for clustering.

In the past decade, deep learning is developing rapidly, and many new deep network architectures have emerged in different fields (Aljalbout et al. 2018; Min et al. 2018). Some popular architectures, such as the auto-encoder (AE) (Rumelhart et al. 1986), the recursive neural network (RvNN), the convolutional neural network (CNN), the generative adversarial network (GAN), are all proposed for different application. Each of these architectures has subtleties. Among them, the auto-encoder (Rumelhart et al. 1986) is an artificial neural network model, which consists of an encoder and a decoder. The encoder of the model learns the hidden features of the input object by training a mapping function; this process is encoding. The decoder can reconstruct the object from the hidden layer coding. The auto-encoder is a type of neural network used in semi-supervised learning and unsupervised learning. It is widely used for dimensionality reduction or feature learning. The learning goal of the model is to train a function to make the model output approximately equal to the input. It uses the idea of sparse coding, uses sparse features to reconstruct the input and learns the cluster-friendly representations. In 2006, Hinton improved the prototype of the auto-encoder architecture to form the deep auto-encoder (DAE) (Hinton et al. 2006), which is the neural network with multiple hidden layers and powerful feature representation capabilities.

In this paper, we propose a novel deep clustering approach, termed the deep clustering based on embedded auto-encoder (EmAEC), which is mainly used to cluster unlabeled high-dimensional data. Based on the prototype auto-encoder architecture, we embed the auto-encoder into the encoder units and decoder units, respectively. The embedded auto-encoder will perform an encoding–decoding operation before the final encoding the input. Since the decoder unit and encoder unit of the model are mirror-symmetrical, the embedded auto-encoder will also perform an encoding–decoding operation before decoding. The EmAEC model compresses and expands the object multiple times by performing the encoding and decoding process; it has a powerful feature representation capabilities and can effectively detect the key feature of the object. We have conducted experiments on various image datasets, and the results show that our model has significant advantages than the state-of-the-art clustering approaches.

For clarity, the main contributions of this paper are summarized as follow:

(1) A novel embedded auto-encoder architecture is proposed by embedding the auto-encoder into the coding unit and the decoding unit of the prototype auto-encoder, respectively. Compared with the existing auto-encoder based model, our model has fewer parameters and better clustering performance.

(2) In order to obtain more smoother and continuous intermediate layers manifold, in the training stage of the model, we impose a smoothness constraints on the hidden layer coding, and use the parameter to control the compromise between the reconstruction accuracy and the smoothing constraint. It significantly improves the representation ability of hidden layer coding.

(3) During the fine-tuning of the model, we use the adaptive self-paced learning approach to automatically select the fine-tuning samples of the model, and propose to use the median value of the distance between all objects and their corresponding centroids to determine the threshold to select the objects. The experimental results show that this strategy can better prevent the objects of the cluster boundary from participating in the training, and has better convergence.

The remainder of this paper is organized as follow: the related work is reviewed in Sect. 2. The proposed deep clustering approach termed EmAEC is introduced in Sect. 3. In Sect. 4, we introduced experimental procedures in detail, reported the experimental results and analyzed them. Section 5 concludes this paper.

2 Related work

2.1 Deep clustering

There has been extensive literature on deep learning over the past decade (Aljalbout et al. 2018; Min et al. 2018). Deep neural networks (DNNs) are network architectures with multiple hidden layers that can learn nonlinear transformations of objects. Deep clustering is a combination of deep learning and clustering. It uses DNNs to learn the representation of the raw features of the input object, and uses these cluster-friendly representations as the input of the specific clustering algorithm (Aljalbout et al. 2018). Unsupervised deep learning models mainly include the auto-encoder (AE), the clustering deep neural networks (CDNN) and deep generative models, the deep belief networks (DBNs) model, etc. (Min et al. 2018). The clustering deep neural networks train the feedforward neural network (FNN) by the clustering loss, which has applications on large-scale image datasets. In addition, the generative adversarial networks (GAN) (Goodfellow et al. 2014) and variational auto-encoders (VAE) (Kingma and Welling 2013) are the more popular depth generative models in recent years. They can perform clustering and obtain new samples generated by clusters at the same time. Deep belief networks (DBNs) is a generative model stacked by multiple the restricted Boltzmann machines (RBMs) (Hinton et al. 2006). It learns the weights between neurons by establishing a joint distribution between objects and its labels. Hinton uses an unsupervised layer-by-layer greedy algorithm to pre-train and to obtain the weights of the generative model (Hinton et al. 2006). The architecture can be used to learn the hierarchical representation of input objects, and is often used in fields such as image recognition and clustering.

The prototype auto-encoder model is composed of symmetrical encoder unit and decoder unit. It is a neural network whose model output is approximately equal to the input, and can be used for unsupervised representation learning. The model training should ensure that the reconstruction loss between the output of the encoder and the input data is minimized. Since the dimensionality of the hidden layer is usually smaller than the dimensionality of the input object, it can learn the most significant features of the input. The mathematical representation of the auto-encoder model is described in detail as follows.

Suppose $X = \{x_i \in \mathbb{R}^D\}_{i=1}^n$ denotes the dataset that contains n objects and their dimension is D. $f(\cdot)$ represents the encoder network function, which converts object x_i into the clustering-friendly representation $z_i \in \mathbb{R}^d (d < D)$. $\tilde{f}(\cdot)$ represents the decoder network function, which decodes the hidden layer coding z_i to obtain $\tilde{x}_i \in R^D$. The output of the auto-encoder is:

$$\tilde{x}_i = \tilde{f}(z_i; \tilde{\omega}) = \tilde{f}(f(x_i; \omega); \tilde{\omega}) \tag{1}$$

where ω and $\tilde{\omega}$ represent the weights of the encoder and the decoder, respectively. The learning goal of auto-encoder is to learn the identity function:

$$x = \mathcal{F}_{\text{ae}}(x) \tag{2}$$

where x is the input of the model, and $\mathcal{F}_{\text{ae}}(\cdot)$ is the mapping function of the entire auto-encoder.

The AE model reconstructs the input object x_i as much as possible. By adding the constraints to limit the number of neurons in the hidden layer, the network can discover potentially useful cluster information in the datasets. Deep clustering is to learn the feature representation of the raw objects through the deep neural network, and then perform clustering. Assuming that m_j represents the centroid of the j-th cluster, the centroid of all clusters can be expressed as $M = [m_1, m_2, \ldots, m_k] \in \mathbb{R}^{d \times K}$. Let $s_i = \{0, 1\}^K$ be the cluster allocation index of the hidden layer coding z_i corresponding to the object x_i, and $y_i = Ms_i$ represents the centroid of the cluster to which x_i belongs.

Recently, many researchers have also proposed clustering algorithm based on the auto-encoder, such as: VaDE (Jiang et al. 2017), DEC (Xie et al. 2016), IDEC (Guo et al. 2017a), N2D (McConville, et al. 2019) and ASPC-DA (Guo et al. 2020). Jiang et al. proposed the variational deep embedding (VaDE) algorithm (Jiang et al. 2017), which is an unsupervised generative clustering approach, that combines the Gaussian mixture model (GMM) and the variational auto-encoder model (VAE). The VaDE first selects the cluster through a Gaussian mixture model, and uses the cluster to generate potential embedding variables, uses the deep neural network to encode the potential embedding variables into the observable variables and then uses a decoder to decode the observable variables. They use random gradient variational Bayes (SGVB) and the reparameterization trick to maximize the evidence lower bound of VaDE to optimize the model. Xie et al. used nonlinear transformation to map the object to the latent feature space to learn the centroids in the feature space. They proposed the deep embedding for clustering (DEC) (Xie et al. 2016), which first initializes the parameters with the deep auto-encoder, and then performs clustering. Iteratively optimizes by calculating auxiliary target distribution and minimizing Kullback–Leibler (KL) divergence. The goal of DEC is to simultaneously learn feature representation and cluster assignments by calculating the KL divergence between the two distributions. The disadvantage of DEC is that the clustering loss defined by the model cannot guarantee the preservation of the local structure of the feature space, which leads to meaningless features that affect the clustering performance. Subsequently, Guo et al. (2017a) considered the data structure preservation and proposed the improved deep embedded clustering algorithm (IDEC), which manipulate feature space to scatter data points using a clustering loss as guidance. IDEC considers both the clustering loss and the loss of the auto-encoder, and uses mini-batch gradient descent and back-propagation to optimize. The N2D (McConville, et al. 2019) uses manifold learning technology to effectively replaces the clustering network with the Auto-encoded based representations. It reduces the depth of the network and obtains superior clustering performance. Guo et al. (2020) proposed a deep clustering algorithm, which incorporates data augmentation technology and the adaptive self-paced learning termed ASPC-DA. The algorithm first trains the auto-encoder model, learns features suitable for clustering and alternately use data augmentation samples to fine-tune the encoder, and then clusters the samples. Guo also uses the adaptive self-paced learning to selectively update the weight of the encoder.

2.2 Adaptive self-paced learning

The self-paced learning (SPL) is developed from the curriculum learning (Pawan Kumar et al. 2010). They all simulate the human learning process from easy to difficult. The main difference is that the curriculum learning is to determine the order of sample learning according to the given prior knowledge, while self-paced learning is to determine the next learning sample through the iterative process of learning. Therefore, the self-paced learning process is to first learn the basic knowledge of a specific task through the simplest and most confident samples, and when the learned knowledge improves, gradually choose more difficult and less confident samples for learning. In the process of model training, let the model learn simple samples first, then gradually increase the difficulty of learning, and add more complex samples for the model to learn. The mathematics of the self-paced learning is described as follows:

Suppose $X = \{x_i \in \mathbb{R}^D\}_{i=1}^n$ represents the sample set, let $f(x)$ denotes the SPL model function, which is the predicted output of the corresponding sample x. Let y be the ground-truth label of x, then the loss function of the SPL model is $L(f(x), y)$. The objective function of the SPL model as follows:

$$\arg\min_{\omega,\upsilon} \sum_{i=1}^{n} \upsilon_i L(f(x_i,\omega), y_i) - \lambda \sum_{i=1}^{n} \upsilon_i \quad s.t. \;\; \upsilon \in [0,1]^n \tag{3}$$

where υ_i is an auxiliary variable corresponding to sample x_i, it determines whether x_i can participate in training. $\lambda \sum_{i=1}^{n} \upsilon_i$ is the regular term, it automatically selects simple and high-confidence samples for training. λ is the hyper-parameter, which controls the step length of the model training.

In the clustering problem, we intuitively measure the distance between the hidden layer coding of the model and the centroid corresponding to the sample to determine the confidence that the sample belongs to a specific cluster. The distance is expressed as $d_{ij} = \psi(z_i, o_j)$, where $\psi(\cdot)$ can be any function that measures the distance between two vectors, such as Euclidean distance, cosine distance, Manhattan distance, Chebyshev distance, etc. Therefore, the objective function of the self-paced learning in the clustering is:

$$\arg\min_{\omega,\gamma} \frac{1}{n} \sum_{i=1}^{n} \gamma_i \| f(x_i;\omega) - Ms_i \|_2^2 + \phi(\lambda, \gamma_i) \quad s.t. \;\; \gamma_i \in [0,1] \tag{4}$$

where $\gamma_i (i \in [1, 2, \ldots, n])$ is the weight of x_i, and $\phi(\lambda, \gamma_i)$ is the regular term of the Self-Paced learning.

3 Deep clustering based on embedded auto-encoder

3.1 The network architecture

Given the input dataset $X = \{x_i \in \mathbb{R}^D\}_{i=1}^n$, it contains n unlabeled objects, and the dimension of each object is D. The learning goal of our model is to construct a better neural network encoder $f(\cdot;\omega)$ to make its hidden layer coding $\{z_i\}_{i=1}^n$ more suitable for clustering.

In order to enable the encoder to detect the high-level features of the input object more effectively and improve the model's representation capabilities, we embed the auto-encoder into the encoder unit of the prototype auto-encoder, that is, we use $D-1024-256-1024-d$ architecture in the encoder units. Before the final encoding of the input object, the embedded auto-encoder can perform an encoding–decoding operation in advance, which makes the dimensionality reduction more effective. The proposed model is mirror-symmetrical. Therefore, we also embed the same auto-encoder in the decoder unit of the prototype auto-encoder, that is, uses $d-1024-256-1024-D$ architecture in the decoder unit. The embedded auto-encoder performs an encoding–decoding in advance before the final reconstruction of the hidden layer coding. The overall architecture of the embedded auto-encoder model is illustrated in Fig. 1. Since the model allows multiple compression and expansion of objects in the encoding and decoding process, this pre-encoding–decoding operation allows the model to have stronger representation capabilities, and can effectively detect the important associated attributes of the objects.

3.2 Model training

The entire deep clustering algorithm based on embedded auto-encoder is divided into two stages: the pre-training and fine-tuning. The pre-training of the model mainly performs feature extraction or dimensionality reduction on the input object, while fine-tuning is mainly to adjust the extracted features to make it more suitable for clustering.

3.2.1 Pre-training

In the pre-training phase, the model is trained by minimizing the reconstruction loss between the input and output of the model. This paper uses the mean square error loss function as the reconstruction loss function (Xie et al. 2016; Guo et al. 2017a,2020). The reconstruction loss function of the model is:

$$\mathcal{L}_{\text{rec}}(\omega,\tilde{\omega}) = \frac{1}{n}\sum_{i=1}^{n}\|x_i - \mathcal{F}_{\text{ae}}(x_i)\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\left\|x_i - \tilde{f}(f(x_i;\omega);\tilde{\omega})\right\|_2^2 \tag{5}$$

After pre-training, the obtained hidden layer coding $z_i = f(x_i;\omega)$ is the low-dimensional representation of the raw object x_i. On the premise of preserving the key information of the input object, the hidden layer coding has a stronger representation capabilities than the raw features of the input object. In order to explicitly learn a smoother and more continuous intermediate layer manifold and obtain a more significant representation of the hidden layer, we add the smoothing constraint to the training function of the model. During the pre-training process, the objective function of the encoder is:

$$\mathcal{L}_{\text{rec}}(\omega,\tilde{\omega}) = \frac{1}{n}\sum_{i=1}^{n}\left\|x_i - \tilde{f}(f(x_i;\omega);\tilde{\omega})\right\|_2^2 + \varepsilon\|f(x_i;\omega)\|_2^2 \tag{6}$$

In (6), the first term on the right is the reconstruction term or fidelity term, which is used to ensure the reconstruction accuracy of the model; the second term is the regularization term, which is mainly used to impose smoothing constraints on hidden layer coding. The weight ε represents a compromise between reconstruction accuracy and smoothing constraints.

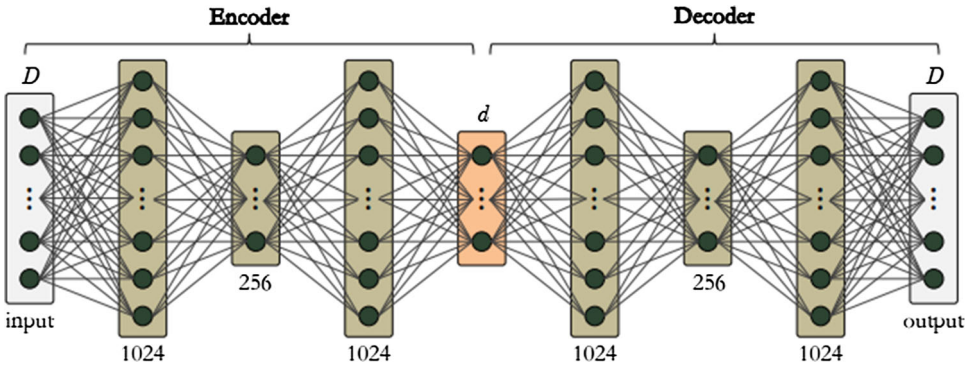


Fig. 1 The architecture of the embedded auto-encoder

3.2.2 Fine-tuning

After pre-training, traditional clustering algorithms can be used to cluster the hidden layer coding to obtain the cluster assignment of each object. During the fine-tuning process, the model is trained in the supervised manner, and the centroid of the cluster corresponding to the object is used as the label of the object. In order to detect cluster-friendly features, we need to fine-tune the encoder. Therefore, during the fine-tuning process, the encoder is trained by minimizing the following objective function:

$$\mathcal{L}_{\text{clu}}(\omega, s) = \frac{1}{n}\sum_{i=1}^{n} \|f(x_i;\omega) - Ms_i\|_2^2 \quad s.t.\ \ s_i \in \{0,1\}^K,\ \ 1^{\mathrm{T}}S_i = 1 \tag{7}$$

where $s = [s_1, s_2, \ldots, s_n] \in \{0,1\}^{K\times n}$ denotes the cluster assignment matrix of all objects, and $M \in \mathbb{R}^{d\times K}$ represents the matrix formed by the centroids of the corresponding clusters.

In the fine-tuning process, we first fix the cluster allocation matrix, at the same time, the centroid o_i of the cluster corresponding to x_i is the label of supervised training, and the weight ω of the encoder is updated through supervised learning. Then, the encoder weight ω is fixed, the hidden layer coding z_i of the object x_i is obtained through feedforward propagation, and the cluster assignment matrix of the object is updated according to the distance between z_i and the corresponding centroid o_i. Through the above steps, the weights of the encoder and the cluster assignment of the objects are alternately updated to obtain the optimal solution of the model.

3.2.3 Adaptive self-paced learning

The self-paced Learning is a learning strategy that simulates humans from easy to difficult. In the learning process, it gradually integrates all samples into the training process through a selection strategy. In the learning process, the simple sample refers to a sample with a smaller loss, and the complex sample refers to a sample with a larger loss. During the encoder fine-tuning process, some incorrectly labeled samples may mislead the encoder training. Therefore, manual screening of samples is required. Inspired by Guo et al. (2020), we adopted adaptive self-paced learning strategy to select the fine-tuning samples of the encoder. Specifically, we use the distance between the hidden layer coding of the sample and the centroid of its corresponding cluster as the confidence that the sample belongs to the certain cluster. Then, the fine-tuning samples of the encoder are selected by setting the confidence threshold of the samples.

In clustering, the objective function of the self-paced learning is formula (4). The regular term is added to the objective function of Self-Paced learning, using $\phi(\lambda, \gamma_i) = -\lambda\gamma_i$; therefore, the objective function is rewritten as:

$$\arg\min_{\omega,\gamma} \frac{1}{n}\sum_{i=1}^{n} \gamma_i \|f(x_i;\omega) - Ms_i\|_2^2 - \lambda\gamma_i \quad s.t.\ \ \gamma_i \in \{0,1\} \tag{8}$$

It can be solved by alternately optimizing the weight of the sample and the encoder parameters. When γ is fixed, (8) is the weighted objective function minimization problem; when ω is fixed, it is solved by the threshold λ of sample confidence:

$$\gamma_i = \begin{cases} 1 & \text{if } d_i < \lambda \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where d_i denotes the distance between the hidden layer coding z_i of x_i and its corresponding centroid o_i.

Guo et al. (2020) used the following formula to determine the threshold and select the fine-tuning sample.

$$\lambda = \mu(L^t) + \frac{t}{T}\sigma(L^t) \tag{10}$$

where $L = \psi(z_i, o_i)$ denotes the clustering loss, and L^t denotes the clustering loss of the t-th iteration, $\mu(\cdot)$ and $\sigma(\cdot)$, respectively, represent the mean function and the standard deviation function of the clustering loss, t and T, respectively, represent the current number of iterations and the total number of iterations.

During the fine-tuning process, the objective function of the encoder is defined as shown in (7). Solve by alternately optimizing the weight of the sample and the parameters of the encoder. The sample confidence threshold in (9) is used to solve the parameters and weights of the encoder. In the process of fine-tuning, we use the following formula to set the fine-tuning threshold

$$\lambda = k(L^t) + \frac{t}{T}\mu(L^t) \tag{11}$$

where $\mu(\cdot)$ is the median function. In subsequent experiments, we observed that setting the fine-tuning threshold of the model by (11) can make the model produce better convergence. In Sect. 4, we conduct experiments to verify the strategy.

3.2.4 Overall algorithm

In this section, we introduce the overall algorithm of the deep clustering based on embedded auto-encoder (EmAEC). In Table 1, we summarize all the mathematical symbols that are used throughout the paper.

Table 1 Summary of notations

Notations	Meanings
$X = \{x_i \in \mathbb{R}^D\}_{i=1}^n$	Input dataset, it contains n objects, each object has D features
$\tilde{x_i} \in R^D$	Decoder output
$z_i \in \mathbb{R}^d$	The hidden layer coding corresponding to x_i
$f(\cdot)$	The mapping function of the encoder network
$\tilde{f}(\cdot)$	The mapping function of the decoder network
$\mathcal{F}_{ae}(\cdot)$	The mapping function of the entire auto-encoder network
$\mathcal{H}(\cdot)$	The mapping function of data augmentation
$\psi(\cdot)$	The distance function between two vectors
$\phi(\cdot)$	The self-paced learning regularization items
$k(\cdot)$	The median function
$\mu(\cdot)$	The mean function
$\sigma(\cdot)$	The standard deviation function
ω	The weight of encoder
$\tilde{\omega}$	The weight of decoder
λ	The object confidence threshold
ε	The weight for smoothing items
$M = [m_1, m_2, \ldots, m_k] \in \mathbb{R}^{d \times K}$	The matrix corresponding to the centroids of all clusters
$\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_n]^{\mathrm{T}}$	The weight coefficient vector of the object
$s_i = \{0, 1\}^K$	The cluster assignment index of x_i
y_i	The ground-truth label of x_i
$\mathcal{L}_{\mathrm{rec}}(\cdot)$	The reconstruction loss function between model input and output
$\mathcal{L}_{\mathrm{clu}}(\cdot)$	The objective function of encoder training
$\mathcal{L}$	The clustering loss function
t	The current iteration number
T	The total number of iterations

In the model training process, in order to obtain more training data to prevent over-fitting and improve the generalization ability of the algorithm, we adopted the data augmentation technology (Guo et al. 2018,2020). Specifically, the input of the model is transformed by random rotations and translations first, and then, the transformed data are input into the model. Assume that the mapping function corresponding to data augmentation is $\mathcal{H}(\cdot)$, the augmented data are represented as follows, which is the actual input of the model.

$$\hat{x_i} = \mathcal{H}(x_i) \tag{12}$$

Input the augmented transformed data into the model, and rewrite the objective function of the auto-encoder as:

$$\mathcal{L}_{\mathrm{rec}}(\omega, \tilde{\omega}) = \frac{1}{n}\sum_{i=1}^{n} \left\|\hat{x_i} - \tilde{f}(f(\hat{x_i}; \omega); \tilde{\omega})\right\|_2^2 + \varepsilon \|f(\hat{x_i}; \omega)\|_2^2 \tag{13}$$

Correspondingly, the objective function of the fine-tuning stage is rewritten as:

$$\arg\min_{\omega, \gamma} \frac{1}{n}\sum_{i=1}^{n} \gamma_i \|f(\hat{x_i}; \omega) - Ms_i\|_2^2 - \lambda\gamma_i \quad s.t. \;\; \gamma_i \in \{0, 1\} \tag{14}$$

Solve (14) alternately through the following two steps:

Step 1: When s and ω are fixed, solve the weight γ of the samples in the adaptive self-paced learning according to the following:

$$\gamma_i = \begin{cases} 1 & \text{if } \|f(x_i; \omega) - Ms_i\|_2^2 < \lambda \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where λ is designated by (11).

Step 2: When γ and s are fixed, update the weight ω of the encoder according to (15).

$$\min_{\omega} \frac{1}{n}\sum_{i=1}^{n} \gamma_i \left\|f(\hat{x_i}; \omega) - Ms_i\right\|_2^2 \tag{16}$$

Then, the samples selected by the threshold, and the back propagation algorithm are used to solve (16). Finally, the cluster assignment of objects is updated by (17), and get the final output of the model.

$$s_{ij} = \begin{cases} 1 & \text{if } j = \arg\min_k \|f(x_i;\omega) - m_k\|_2^2 \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

Equations (15)–(17) are solved one by one through alternate optimization until the algorithm converges or the termination condition is satisfied.

The deep clustering based on embedded auto-encoder is given in Algorithm 1.

Algorithm 1

Input: $X, \mathcal{H}(\cdot), K, \mathcal{E}, T$

1: Data augmentation, get $\hat{x}_i$ by (12);

2:Train $\mathcal{F}_{ae}(\cdot)$, and find the initial value of the weight ω of the Auto-encoder by (13);

3:Cluster z_i, and initialize M and S ;

4:for $t=0$ to T do

5: Update sample weight γ by (15);

6: Update encoder weight ω by (16);

7: Update cluster assignment vector S by (17);

8: If $t>T$ then

9: Stop Model Training;

10: end if;

11:end for;

Output S

4 Experiments

In this section, we conduct different experiments on multiple real-world datasets and compare the proposed approaches against the state-of-the-art clustering algorithm to prove the effectiveness of the proposed approaches.

4.1 Experimental settings

All experiments are conducted in MATLAB 9.0.0.341360 (R2016a) 64-bit on PC with Intel (R) Core (TM) i7-6500 CPU (2.50 GHz) and a NVIDIA GeForce 1080Ti GPU.

4.2 Datasets

In our experiment, we use seven real-world datasets, namely, MNIST-full (LeCun et al. 1998), MNIST-test (LeCun et al. 1998), USPS[1], Yale (Georghiades et al. 2001), Fashion (Xiao et al. 2017), Pen digits (Alimoglu and Alpaydin 1997), SEMEION. These datasets are all image datasets, where MNIST-full, MNIST-test and Yale are the classic datasets, Pen digits and SEMEION are from the UCI machine learning repository[2], while the rest are from some references or personal homepages.

The details of the experimental datasets are given in Table 2.

MNIST-full: The dataset consists of 250 handwritten numbers 0–9 by different people. The training set contains 60,000 samples, and the test set contains 10,000 samples, for a total of 70,000 samples. Each picture consists of 28 × 28 pixels.

MNIST-test: The dataset is the test set in the MNIST-full and contains 10,000 samples.

USPS: A database of handwritten digits, including 10 categories, with a total of 9298 images composed of 16 × 16 pixels.

Fashion: A dataset of Zalando's article images consists of fashion products, containing 70,000 images in 10 categories. Each image is 28 × 28 Gy image.

Pen digits: This dataset contains 10,992 records in 10 categories, where each record with size of 4 × 4 pixels.

Yale (32 × 32): The dataset contains 15 people, each with 15 images, a total of 165 Gy images. Image size is 32 × 32.

Yale (64 × 64): The dataset contains 15 people, each with 15 images, a total of 165 Gy images. Image size is 64 × 64.

COIL-100: The dataset composed of 100 objects at different angles in 360° rotation. Each object has 72 poses, and the image size is 32 × 32.

SEMEION: The handwritten dataset, containing 1593 handwritten digits from 0 to 9, each image size is 16 × 16.

The details of the benchmark datasets are given in Table 2.

4.3 Evaluation methodology

In our experiment, we utilized three widely used clustering performance evaluation criterion to measure the quality of all clustering algorithms, namely clustering accuracy (ACC), normalized mutual information (NMI) and adjusted rand index (ARI).

The NMI can measure the similarity between the test clustering solution and the ground-truth clustering solutions based on the shared information. We use the following formula to calculate the NMI score of the test cluster solution with respect to the ground-truth clustering solutions

$$\mathrm{NMI}\left(P, P^G\right) = \frac{\sum_{i=1}^{n^P}\sum_{j=1}^{n^G} n_{ij}\log\frac{n_{ij}n}{n_i^P n_j^G}}{\sqrt{\sum_{i=1}^{n^P} n_i^P \log\frac{n_i^P}{n}\sum_{j=1}^{n^G} n_j^G \log\frac{n_j^G}{n}}} \tag{18}$$

where P is the test clustering solution, P^G is the ground-truth clustering, n^P and n^G represent the number of clusters in P and P^G, respectively. n_i^P is the number of objects in the i-th cluster in P, n_j^G is the number of objects in the j-th cluster in P^G, and n is the number of objects in X.

The ACC is an evaluation criteria based on clustering accuracy, which is the ratio of correctly classified samples to the total number of samples in the category. The ACC scores of the test clustering solution with respect to the ground-truth clustering solutions are calculated as follows:

$$\mathrm{ACC}\left(P, P^G\right) = \frac{\sum_{i=1}^{n}\delta(P^G, P)}{n} \tag{19}$$

where $\delta(\cdot)$ is the indicator function, which is defined as follows:

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

The ARI is a commonly used clustering evaluation criterion, which is a function to calculate the distribution similarity between the test clustering and the ground-truth clustering. It is defined as:

$$\mathrm{ARI}\left(P, P^G\right) = \frac{r_0 - r_3}{\frac{1}{2}(r_1 + r_2) - r_3} \tag{21}$$

where

$$r_0 = \sum_{i=1}^{n^P}\sum_{j=1}^{n^G}\binom{\left|C_i^P \cap C_j^{P^G}\right|}{2}$$

, $r_1 = \sum_{i=1}^{n^P}\binom{\left|C_i^P\right|}{2}$, $r_2 = \sum_{j=1}^{n^G}\binom{\left|C_j^{P^G}\right|}{2}$, $r_3 = \frac{2r_1 r_2}{n(n-1)}$ C_i^P and $C_j^{P^G}$ represent the i-th clusters in P and the j-th clusters in P^G, respectively.

In the above evaluation criterion, the ACC and NMI scores are in the range [0, 1]; the ARI score is in the interval of [−1, 1]. Note that the larger evaluation scores indicate better quality of clustering.

4.4 Hyper parameter settings

As depicting in Fig. 1, the embedded auto-encoder model proposed in this paper has 8 fully connected layers, and the number of neurons in each layer is $D - 1024 - 256 - 1024 - d - 1024 - 256 - 1024 - D$, where D represents the dimension of the input objects, and d is the dimension of hidden layer coding. We set d as the ground-truth number of clusters in the dataset. In addition, except for the input layer, output layer and the hidden coding layer, the activation functions of the other layers all use the ReLU function (Atul Shah and Koltun 2017). The smoothing constraint parameter in (6) and (13) is set to $\varepsilon = 0.001$. In the pre-training phase, the auto-encoder uses the stochastic gradient descent (SGD) optimizer, the learning rate is 1.0, the momentum is set to 0.99, the batch size is 256, and the total training has 1 million iterations. The fine-tuning stage uses the Adam optimizer (Kingma and Ba 2015), the learning rate is 0.0001, the batch size is also set to 256, the maximum number of iterations is set to

Table 2 Description of the datasets

Datasets	#Objects	#Classes	Size	Dimension
MNIST-full	70,000	10	28 × 28	784
MNIST-test	10,000	10	28 × 28	784
Fashion	70,000	10	28 × 28	784
USPS	9298	10	16 × 16	256
Pen digits	10,992	10	4 × 4	16
Yale (32 × 32)	165	15	32 × 32	1024
Yale (64 × 64)	165	15	64 × 64	4096
COIL-100	7200	100	32 × 32	1024
SEMEION	1593	10	16 × 16	256

[1] http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html

[2] https://archive.ics.uci.edu/ml/index.php

$T = 100$, and the number of iterations of the fine-tuning encoder in each iteration is set to 5000.

In the data augmentation operation, we only use random rotation and translation, where the angle of rotation is in the interval of $[-10°, +10°]$, and the range of the number of pixels for shifted up and down translation does not exceed 10% of the image height and width, respectively. Note that since the data augmentation is based on the random rotation and translation of the two-dimensional image, the data need to be convertible into the two-dimensional image. Before the object x_i is input to the augmented mapping function $\mathcal{H}(\cdot)$, it is first converted into the two-dimensional image of size $\sqrt{D} \times \sqrt{D}$. After random rotation and translation of the augmented function, it is converted into a one-dimensional vector.

All hyper parameter settings in the experiment are described in Table 3.

4.5 Comparison with other clustering approaches

In this section, we compare the proposed EmAEC with state-of-the-art clustering approaches, including the traditional clustering method, such as K-means (MacQueen, et al. 1967), spectral clustering (SC) (Chen et al. 2010), graph regularized nonnegative matrix factorization (GNMF) (Cai et al. 2010), Gaussian mixture models (GMM), landmark-based spectral clustering (LSC) (Cai and Chen 2014) and sampling-based scalable sparse subspace clustering (SSC) (Matsushima and Brbic 2019). Also includes some deep clustering approaches, for example, robust continuous clustering (RCC) (Atul Shah and Koltun 2017), N2D (McConville, et al. 2019), deep clustering with convolutional auto-encoders (DCEC) (Guo et al. 2017b), deep embedding clustering (DEC) (Xie et al. 2016) and improved deep embedded clustering (IDEC) (Guo et al. 2017a). All the experimental results we report are obtained by running the source code provided in the literature.

4.6 Experiment and analysis

4.6.1 Ablation experiment

In this section, we perform the ablation experiments to analyze the proposed model architecture and explain the advantages of its innovations. In the pre-training phase, we proposed the novel embedded auto-encoder architecture, as shown in Fig. 1. The previous methods, such as N2D (McConville, et al. 2019), DCEC (Guo et al. 2017b), DEC (Xie et al. 2016), IDEC (Guo et al. 2017a) and ASPC-DA (Guo et al. 2020), all used the $D - 500 - 500 - 2000 - d - 2000 - 500 - 500 - D$. In order to compare the performance of these two models, we set the comparison variable Network, Network = 0 represents the model used in the cited literature, and Network = 1 corresponds to our proposed model.

At the same time, in the pre-training stage of the model, we introduced the L_2 constraint to the objective function ((8) and (14)). Therefore, in order to compare them, we also introduce another comparison variable Smooth. Smooth = 0 represents that there is no smoothing constraint, and Smooth = 1 represents that there is a smoothing constraint. The results of the ablation experiment in the pre-training stage are reported in Table 4.

In the experiment, we first compared the proposed model with the auto-encoding model used in the references (Guo et al. 2020) and analyzed their quality of clustering on five benchmark datasets. It can be seen that on all datasets, our model has improved ACC and NMI scores compared with the ASPC-DA model. For the MNIST-full, the ACC score increased from 0.8916 to 0.9102, and the NMI score increased from 0.9463 to 0.9589; For the MNIST-test, the ACC score increased from 0.8611 to 0.8895, and the NMI score increased from 0.9388 to 0.9491. For the other three datasets USPS, Fashion and SEMEION, our model has a significant advantage.

Table 3 Hyper parameter settings of deep clustering model

Stage	Pre-training	Fine-tuning
Network architectures	$D - 1024 - 256 - 1024 - d - 1024 - 256 - 1024 - D$	$D - 1024 - 256 - 1024 - d$
Data augmented	Random rotation $[-10°, +10°]$ translation $\sqrt{D}/10$	Random rotation $[-10°, +10°]$ translation $\sqrt{D}/10$
Batch size	256	256
Optimizer	SGD, learning rate = 1.0, momentum = 0.99	Adam, learning rate = 0.0001
Number of iterations	1 million	5000
Fine-tuning times	/	$T = 100$
Smooth constraint	$\varepsilon = 0.001$	/

Table 4 Ablation experiment of architectures and smoothing constraints in pre-training (ACC/NMI)

Pre-training	Network	Smooth	MNIST-full	MNIST-test	USPS	Fashion	SEMEION
Models	0	0	0.8916/0.9463	0.8611/0.9388	0.8657/0.9238	0.5887/0.5439	0.5983/0.7001
	0	1	0.8962/0.9519	0.8723/0.9427	0.8748/0.9481	0.5957/0.5537	0.6351/0.7182
	1	0	0.9102/0.9589	0.8895/0.9491	0.8916/0.9506	0.6086/0.5621	0.6805/0.7316
	1	1	0.9184/0.9658	0.8978/0.9576	0.8994/0.9553	0.6170/0.5704	0.7080/0.7407

Then, we separately compare the effects of the proposed model and ASPC-DA (Guo et al. 2020) before and after adding constraints. The comparison results show that on the above datasets, after adding the smoothing constraint, the ACC and NMI scores of the two architectures have been improved. For MNIST-full, before adding constraints, the ACC score of the model proposed in this paper is 0.9102, and the NMI value is 0.9589; and after constraints are added to the model, the ACC score increases to 0.9184 and the NMI value increases to 0.9658. Similarly, after the ASPC-DA (Guo et al. 2020) adds constraints, the ACC scores increase from 0.8916 to 0.8962, and the NMI scores increase from 0.9463 to 0.9519. And the trend is the same on the remaining datasets. It can be seen that adding the smoothing constraints to the model can significantly improve the representation capabilities of the hidden layer coding of the model.

In the fine-tuning stage, we improved the threshold setting method for object selection to select fine-tuning samples of the model (10) and (11)). We set Network = 1 and Smooth = 1 to compare the impact of the threshold selection strategies in this paper and the literature (Guo et al. 2020) on the model performance. The results are reported in Table 5.

Comparing the experimental results, we observe that in the fine-tuning of the model, the selection strategy proposed in this paper to determine the weight threshold of the regular term of the self-paced learning not only shows better clustering performance on each dataset than the strategy used by Guo et al. (2020). It also shows a relatively stable convergence process.

4.6.2 Comparison with state-of-the-art clustering approaches

The quantitative comparison results of the EmAEC and other clustering algorithms, including the classical clustering algorithm and the deep clustering algorithms, are reported in Table 6a and b.

All results of the comparison algorithms are reported by running their released code. Some algorithms do not provide source code or cannot run on our platform, so some reported values come from original literature. In the table, we use the symbol "_" to mark the clustering results cited from the corresponding literature, and the symbol "N/A" indicates that the approaches are computationally infeasible or cannot be performed on the corresponding dataset. In addition, the best ACC and NMI scores are signaled in red in the table, and the second best scores are signaled in blue in the table.

As shown in Table 6a and b, the performance of the deep clustering algorithm is significantly improved compared to the traditional clustering algorithm. The proposed EmAEC algorithm obtains the best ACC and NMI scores on the datasets, and has the best overall clustering effect. Especially, for the USPS, FASHION and Yale (32×32), the proposed EmAEC achieves significant improvements in terms of NMI and ACC scores compared to other deep clustering methods.

4.6.3 Execution time

In this section, we compare the execution time of different clustering methods on different data sets of different sizes. The details of the benchmark dataset are illustrated in

Table 5 The influence of different threshold selection strategies on model performance in fine-tuning (Network = 1, Smooth = 1, ACC/NMI)

Fine-tuning	Threshold	MNIST-full	MNIST-test	USPS	Fashion	SEMEION
Models	Equation (10)	0.9885/0.9681	0.9781/0.9438	0.9837/0.9564	0.6217/0.6672	0.8007/0.7689
	Equation (11)	0.9892/0.9683	0.9806/0.9498	0.9852/0.9590	0.6898/0.6752	0.8230/0.7866

Table 6 Clustering performances (in terms of ACC and NMI) of different algorithms

(a) Methods\datasets	MNIST-full	MNIST-test	USPS	Fashion
K-Means (MacQueen, et al. 1967)	0.5807/0.4853	0.6098/0.5099	0.6998/0.6042	0.5545/0.5119
SC (Chen et al. 2010)	0.7258/0.7122	0.6679/0.6478	0.7921/0.7552	0.6258/0.6122
GNMF (Cai et al. 2010)	0.7752/0.7656	0.7438/0.7072	0.7890/0.7901	0.5659/0.5590
GMM (McLachlan and Peel 2004)	0.3153/0.2446	0.3102/0.2180	0.4792/0.4416	0.3880/0.4374
LSC (Cai and Chen 2014)	0.7821/0.7315	0.7345/0.7065	0.7987/0.7898	0.6718/0.6389
RCC (Atul Shah and Koltun 2017)	0.8125/0.8931	0.8016/0.8280	0.9709/0.7655	0.3865/0.6582
Deep cluster (Caron et al. 2018)	**0.7970/0.6610**	**0.8540/0.7130**	**0.5620/0.5400**	**0.5420/0.5100**
N2D (McConville, et al. 2019)	0.9770/0.9380	0.9486/0.8851	0.9393/0.8618	0.6218/0.6589
S5C (Matsushima and Brbic 2019)	0.5958/0.5958	0.5435/0.4896	0.7289/0.6465	0.6374/0.6184
DCEC (Guo et al. 2017b)	0.8789/0.8725	0.8690/0.8506	0.7926/0.8283	0.6388/0.6584
DEC (Xie et al. 2016)	0.8865/0.8740	0.8365/0.8406	0.7647/0.7601	0.5423/0.5479
IDEC (Guo et al. 2017a)	0.8983/0.8623	0.8384/0.8347	0.7755/0.7583	0.5360/0.5548
DSC-DAN (Yang et al. 2019)	**0.9780/0.9410**	**0.9800/0.9460**	**0.8690/0.8570**	**0.6620/0.6450**
ASPC-DA (Guo et al. 2020)	**0.9880/0.9660**	**0.9730/0.9360**	**0.9820/0.9510**	**0.5910/0.6540**
The proposed	0.9892/0.9683	0.9806/0.9498	0.9852/0.9590	0.6898/0.6752

(b) Methods\datasets	Yale (32 × 32)	Yale (64 × 64)	Pen digits	SEMEION
K-Means (MacQueen, et al. 1967)	0.3818/0.4679	0.5152/0.5568	0.7150/0.6724	0.6196/0.5396
SC (Chen et al. 2010)	0.4485/0.4759	0.4606/0.5034	0.7883/0.7878	0.60890.5684
GNMF (Cai et al. 2010)	0.3333/0.3857	0.4667/0.5354	0.7203/0.7412	0.6390/0.5885
GMM (McLachlan and Peel 2004)	0.2606/0.2562	0.2727/0.2415	0.5902/0.5991	0.3999/0.3207
LSC (Cai and Chen 2014)	0.4788/0.5281	0.5818/0.6048	0.7775/0.7601	0.6773/0.6298
RCC (Atul Shah and Koltun 2017)	0.1212/0.1019	0.0667/0.0000	0.8411/0.8551	0.6836/0.6153
Deep cluster (Caron et al. 2018)	N/A	N/A	N/A	N/A
N2D (McConville, et al. 2019)	0.4242/0.4664	0.4849/0.5103	0.8744/0.8484	0.6083/0.6445
S5C (Matsushima and Brbic 2019)	0.4848/0.5403	0.6303/0.6207	0.7725/0.7478	0.6277/0.5272
DCEC (Guo et al. 2017b)	0.4424/0.5335	0.5576/0.6178	N/A	0.7382/0.6777
DEC (Xie et al. 2016)	0.3091/0.4421	0.4727/0.5081	0.7404/0.6965	0.6190/0.5756
IDEC (Guo et al. 2017a)	0.3576/0.4485	0.4727/0.5076	0.7518/0.7461	0.5405/0.5218
DSC-DAN (Yang et al. 2019)	N/A	N/A	N/A	N/A
ASPC-DA (Guo et al. 2020)	N/A	N/A	N/A	N/A
The proposed	0.6477/0.6718	0.6401/0.6538	0.8923/0.8681	0.8230/0.7866

Table 2. Since deep cluster (Caron et al. 2018), DSC-DAN (Yang et al. 2019) and ASPC-DA (Guo et al. 2020) are not performed on the test dataset, they are ignored here. In addition, the deep learning model requires a lot of calculations in the pre-training stage, so the model training is accelerated by GPU. When we compare the efficiency of deep learning algorithms, we ignore the training process and only discuss the inference time. For a fair comparison, all algorithms run on the CPU, and algorithms based on deep learning only report the inference time.

After training the model, we use the K-means algorithm to cluster the hidden layer coding of the model. The execution time of the deep clustering algorithm reported in Table 7 is the inference time, which includes the time to run the K-means and the number of iterations, as well as the time to update the parameters in steps 5, 6 and 7 of Algorithm 1. The experiment was conducted on different image datasets, and their dimensions range from 16 to 4096. Note that the time of the deep clustering method is slightly higher than that of traditional clustering methods, such as K-means and SC, especially for some large volume datasets. The main reason is that for the same dataset, the deep learning method has more iterations. Compared with other deep clustering methods, the EmAEC has obvious

Table 7 The running time of different algorithms (seconds)

Database\methods	K-Means	SC	GNMF	GMM	LSC	RCC	N2D	S5C	DCEC	DEC	IDEC	Ours
MNIST-full	62.66	8.75	86.82	2656.35	28.25	9523.69	101.59	558.81	1065.72	296.22	251.61	182.91
MNIST-test	4.04	0.89	10.98	216.43	4.37	506.52	38.54	79.95	93.86	34.71	138.94	82.74
USPS	0.99	0.65	4.58	69.86	2.07	166.58	36.29	84.34	68.89	76.92	320.12	56.08
Fashion	25.35	13.43	111.98	1548.75	33.64	2883.15	107.63	595.02	1379.83	718.56	932.10	201.53
Yale (32×32)	0.10	0.12	0.28	0.35	0.14	1.89	10.23	2.73	11.50	23.74	15.32	12.92
Yale (64×64)	0.16	0.24	1.03	0.41	0.13	9.60	9.83	6.16	24.99	13.27	14.99	19.77
Pen digits	0.11	1.19	2.41	5.72	2.73	19.43	28.70	133.92	N/A	148.70	20.74	177.28
SEMEION	0.16	0.17	0.69	2.39	0.66	6.88	14.42	8.60	26.99	19.02	23.68	11.98

efficiency advantages. Especially for the large sample dataset, such as MNIST-full, the execution time of our method is 182.91 s, which is far less than the 9523.69 s of RCC and 1065.72 s of DCEC. Except for N2D (McConville, et al. 2019), the EmAEC has shorter inference time on multiple datasets and its efficiency is significantly superior to the other algorithms.

Due to the different dimensions of the dataset, calculated amount and number of parameters of the model are different on different datasets, but the relative relationship of statistical results will not change. We conduct experiments on the MNIST-full to compare the parameter scales and the amount of calculation of the two different auto-encoder network architectures. The results are reported in Table 8.

Compared with the auto-encoder architecture used in Xie et al. (2016), McConville, et al. (2019), Guo et al(. 2017a), Jiang et al. (2017), the EmAEC proposed in this paper reduces the number of the model parameters by about 20%, and the corresponding efficiency increases by about 20%. Therefore, the reduction of model parameters and the shortening of the inference time are the essential reasons for the improvement of computational efficiency. Compared with the previous experiments, our proposed algorithm has the best quantitative performance on many datasets. Therefore, the model has a better compromise between the clustering effect and the computational cost.

4.6.4 Convergence analysis

In this part, we analyze the convergence of the proposed model. For the MNIST-full (Huang et al. 2019), we visualize the clustering results of the learned feature representation at different stages of the model's pre-training and fine-tuning process to verify the convergence of the proposed method. The results are plotted in Figs. 2 and 3, respectively.

The experiment first randomly selects 1000 samples from the dataset and projects them into a two-dimensional space as shown in Fig. 2a. We use K-means to cluster the hidden layer coding of the selected samples, and finally uses t-SNE (Maaten and Hinton 2008) to visualize the clustering results. Figure 2b–f, respectively, shows the clustering results of the model in 50 K, 300 K, 500 K, 750 K and 1000 K iterations in the pre-training.

Comparing Fig. 2a–f, it can be seen that it is difficult to cluster directly by the raw features of the object. After the model pre-training, it is easier to classify the object by clustering the features representations learned from the hidden layer coding. As the number of iterations increases, more and more objects can be correctly clustered, but from the visualization, we observe that there are still a few objects that are incorrectly grouped into other clusters, and cannot be categorized correctly.

Similarly, Fig. 3 visualizes the convergence of the model in the fine-tuning process on MNIST-Full. At the beginning of the fine-tuning phase, as shown in Fig. 3a, most objects have been clustered in the fifth iteration; there still some objects that cannot be grouped correctly. As the number of iterations increases, objects that are not correctly clustered are also be well grouped. It shows that after 100 iterations, the model is stable.

In addition, we also compared the loss curves of the model on different datasets during the pre-training stage.

Table 8 Comparison of calculation amount and parameters of different models

Architecture	FLOPs	Parameters
$D-500-500-2000-d-2000-500-500-D$	6.6616 M	3.3308 M
$D-1024-256-1024-d-1024-256-1024-D$	5.3602 M	2.6801 M

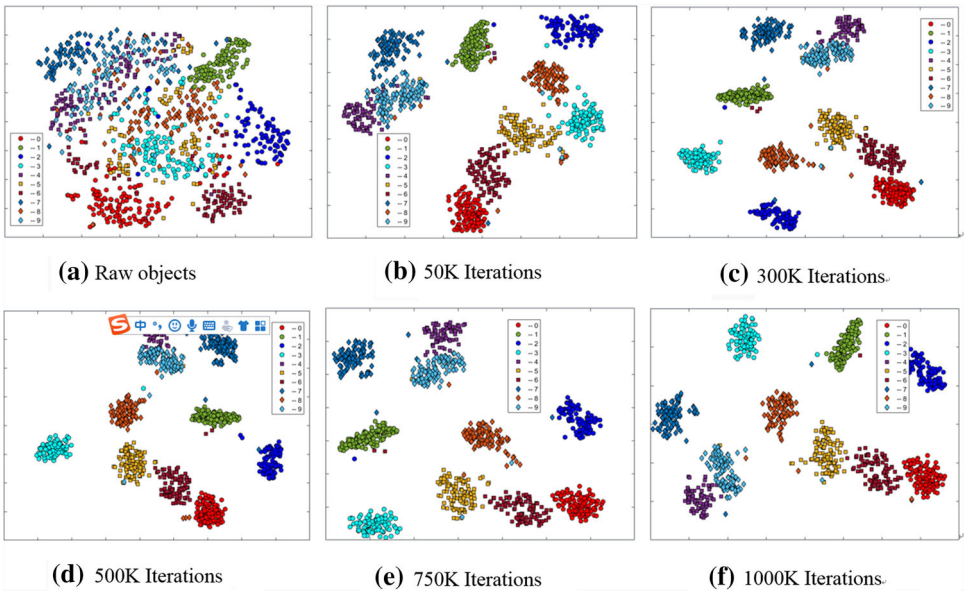


Fig. 2 The convergence process of the model in the pre-training on MNIST-full (t-SNE)

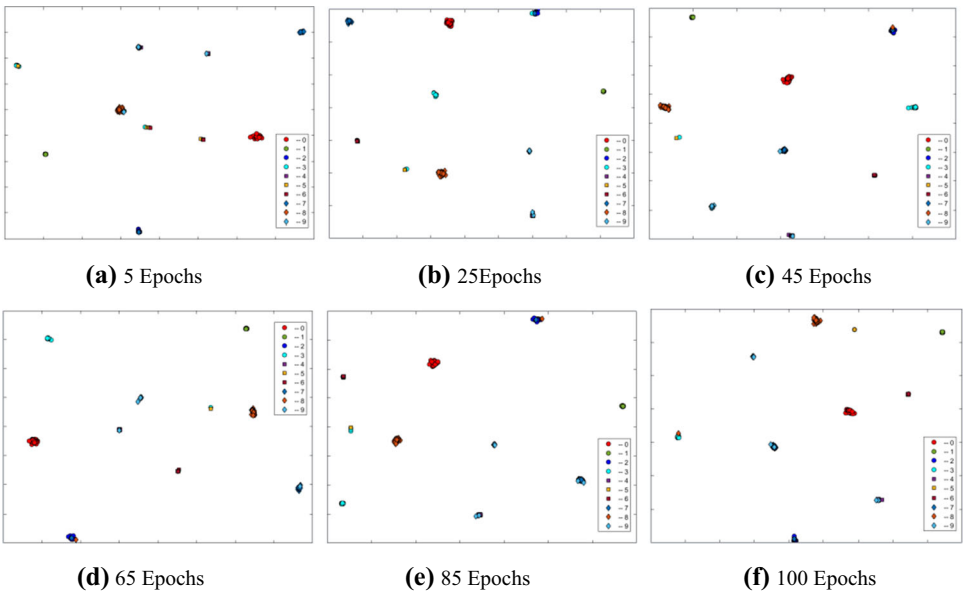
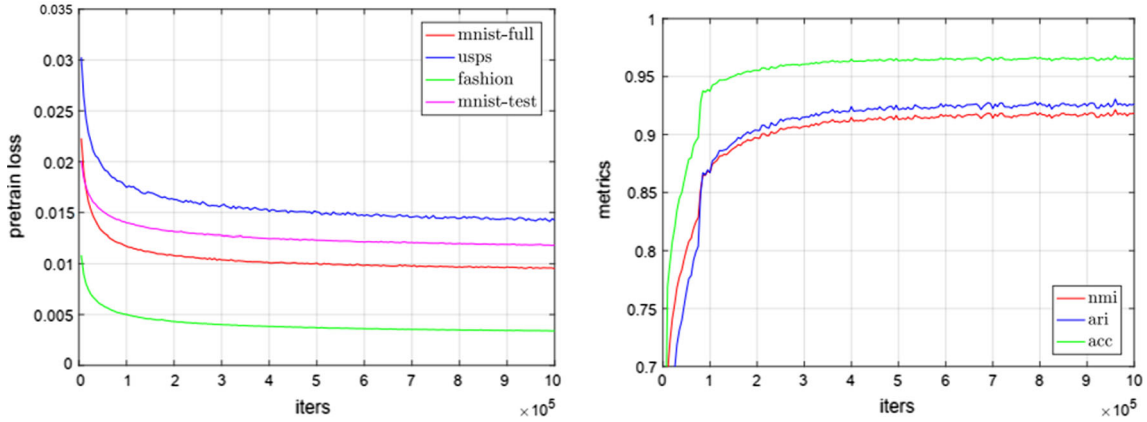


Fig. 3 The convergence process of the model in the fine-tuning on MNIST-full (t-SNE)

As shown in Fig. 4a, it can be seen that in the pre-training, the model converges well on different datasets. Figure 4b shows the trend of ACC, ARI and NMI scores of the model on the MNIST-full. The training process of the model is convergent.

5 Conclusion

In this paper, we propose a novel deep clustering based on embedded auto-encoder, which embeds the auto-encoder on the prototype auto-encoder, that is, it embeds the auto-encoder in the encoder unit and the decoder unit of the prototype auto-encoder. The deep model performs an encoding–decoding operation before the final encoding of the object, which can more effectively improve the representation capabilities of the model. At the same time, the encoding–decoding operation is also performed before finally reconstructing the object. In the model pre-training, we proposed to add a smoothing constraint to the objective function to learn a smoother and more continuous hidden layer manifold. Experimental results show that this greatly improves the representation capabilities of the hidden layer coding of the model. In addition, we also use the adaptive self-paced learning to automatically select the fine-tuning samples of the model encoder. Extensive experiment has been conducted on multiple image datasets, and the experimental results show that our algorithm is



(a) Model loss curve in pre-training phase

(b) Three clustering index curves on MNIST-full

Fig. 4 The convergence curve of the model in the pre-training

significantly superior in clustering effect and efficiency compared with the state-of-the-art clustering approaches.

Author contributions Xuan Huang was involved in writing and editing; Zhenlong Hu helped in data analysis; and Lin Lin was involved in implementing research process.

Funding There is no project funding for this article.

Declaratios

Conflict of interest All authors declare that they have no conflict of interest.

Informed consent This article does not contain any studies with human participants performed by any of the authors, so there is no informed consent involved.

References

Alimoglu F, Alpaydin E (1997) Combining multiple representations and classifiers for pen-based handwritten digit recognition. In: Proceedings of the 4th international conference on document analysis and recognition (ICDAR '97), vol 2, pp 637–640. Washington DC.

Aljalbout E, Golkov V, Siddiqui Y, Cremers D (2018) Clustering with deep learning: taxonomy and new methods

Atul Shah S, Koltun V (2017) Robust continuous clustering. Proc Natl Acad Sci (PNAS) 114(37):9814–9819

Cai D, Chen X (2014) Large scale spectral clustering via landmark-based representation. IEEE Trans Cybern 45(8):1669–1680

Cai D, He X, Han J, Huang TS (2010) Graph regularized nonnegative matrix factorization for data representation. IEEE Trans Pattern Anal Mach Intell 33(8):1548–1560

Caron M, Bojanowski P, Joulin A, Douze M (2018) Deep clustering for unsupervised learning of visual features. European conference on computer vision (ECCV). Munich, Germany, pp 132–149

Chen W, Song Y, Bai H et al (2010) Parallel spectral clustering in distributed systems. IEEE Trans Pattern Anal Mach Intell 33(3):568–586

Georghiades AS, Belhumeur PN, Kriegman DJ (2001) From few to many: Illumination cone models for face recognition under variable lighting and pose. IEEE Trans Pattern Anal Mach Intell 23(6):643–660

Goodfellow IIJ, Pouget-Abadie J, Mirza M, et al (2014) Generative adversarial nets. In: Proceedings of advances in neural information processing systems (NIPS), Montreal, Quebec, Canada, pp 2672–2680.

Guo X, Gao L, Liu X, et al (2017) Improved deep embedded clustering with local structure preservation. In: Proceedings of the international joint conference on artificial intelligence (IJCAI), Melbourne, Australia, pp 1753–1759.

Guo X, Liu X, Zhu E, Yin J (2017) Deep clustering with convolutional autoencoders. In: Proceedings of the 31st international conference on neural information processing systems (NIPS'17), Long Beach, CA, pp 373–382.

Guo X, Zhu E, Liu X, Yin J (2018) Deep embedded clustering with data augmentation. In: Proceedings of the 10th Asian conference on machine learning (ACML), Beijing, China, pp 550–565.

Guo X, Liu X, Zhu E et al (2020) Adaptive self-paced deep clustering with data augmentation. IEEE Trans Knowl Data Eng 32(9):1680–1693

Hinton GE, Osindero S, The YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554

Huang X, Wu L, Ye Y (2019) A review on dimensionality reduction techniques. Int J Pattern Recogn Artif Intell 33(10):1950017

Jiang Z, Zheng Y, Tan H, et al (2017) Variational deep embedding: an unsupervised and generative approach to clustering. In: Proceedings of the 26th international joint conference on artificial intelligence (IJCAI 2017), Melbourne, Australia, pp 1965–1972.

Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: International conference on learning representations (ICLR), San Diego, CA.

Kingma DP, Welling M (2013) Auto-encoding variational Bayes

LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324

MacQueen J, et al (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley symposium on mathematical statistics and probability. Oakland, CA, USA, pp 281–297.

Matsushima S, Brbic M (2019) Selective sampling-based scalable sparse subspace clustering. In: Advances in neural information processing systems (NIPS 2019), Vancouver, Canada, pp 12416–12425.

McConville R, Santos-Rodriguez R, et al (2019) N2D: (not too) deep clustering via clustering the local manifold of an autoencoded embedding

McLachlan GJ, Peel D (2004) Finite mixture models. Wiley, Hoboken

Min E, Guo X, Liu Q et al (2018) A survey of clustering with deep learning: from the perspective of network architecture. IEEE Access 6:39501–39514

Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on international conference on machine learning (ICML'10), pp 807–814

Pawan Kumar M, Packer B, Koller D (2010) Self-paced learning for latent variable models. In: Advances in neural information processing systems (NIPS 2010), Vancouver Canada, pp 1189–1197.

Rumelhart D, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536

van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9(86):2579–2605

Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms

Xie J, Girshick R, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: Proceedings of the 33rd international conference on machine learning (ICML 2016), New York, America, pp 478–487.

Yang X, Deng C, Zheng F, et al (2019) Deep spectral clustering using dual autoencoder network. In: Proceedings of IEEE conference on computer vision and pattern recognition (CVPR), Long Beach, CA, pp 4066–4075.