Adaptive Self-Paced Deep Clustering with Data Augmentation

Xifeng Guo, Xinwang Liu, *Member, IEEE*, En Zhu, Xinzhong Zhu, Miaomiao Li, Xin Xu, *Senior Member, IEEE*, and Jianping Yin

Abstract—Deep clustering gains superior performance than conventional clustering by jointly performing feature learning and cluster assignment. Although numerous deep clustering algorithms have emerged in various applications, most of them fail to learn robust cluster-oriented features which in turn hurts the final clustering performance. To solve this problem, we propose a two-stage deep clustering algorithm by incorporating data augmentation and self-paced learning. Specifically, in the first stage, we learn robust features by training an autoencoder with examples that are augmented by random shifting and rotating the given clean examples. Then, in the second stage, we encourage the learned features to be cluster-oriented by alternatively finetuning the encoder with the augmented examples and updating the cluster assignments of the clean examples. During finetuning the encoder, the target of each augmented example in the loss function is the center of the cluster to which the clean example is assigned. The targets may be computed incorrectly, and the examples with incorrect targets could mislead the encoder network. To stabilize the network training, we select most confident examples in each iteration by utilizing the adaptive self-paced learning. Extensive experiments validate that our algorithm outperforms the state of the arts on four image datasets.

Index Terms—Deep clustering, self-paced learning, data augmentation, unsupervised feature learning

1 Introduction

Clustering has been intensively studied in the data mining and machine learning community. Conventional clustering algorithms such as k-means [1], Gaussian Mixture Model (GMM) [2], and hierarchical clustering [3] generally group data on handcrafted features according to intrinsic similarity. It is well known that these features are designed for general purpose and may not be suitable for a specific task. Some clustering algorithms, including spectral clustering (SC) [4] and kernel k-means [5], [6], transform data into a new feature space in which the clustering task becomes much easier. However, these methods generally have a limited capacity for transformation or suffer from high computational complexity.

With the development of deep learning, Deep Neural Networks (DNNs) have shown amazing power of highly nonlinear transformation (or feature learning). Recently, some researches [7], [8], [9], [10], [11], [12] adopt DNNs to perform clustering, showing dramatic improvement on clustering performance. The basic idea is that good feature helps produce good clustering result, and the latter in turn guides DNNs to learn the better feature. These two processes are seamlessly connected to achieve superior performance. The resultant feature is task-specific, i.e., more suitable for clustering. In this paper, we refer to as *Deep Clustering* the algorithm that jointly performs feature learning and clustering by DNNs.

Most existing deep clustering algorithms tune the parameters of the DNN by using a loss function defined by the cluster centers and assignments, which are generally obtained based on the outputs of the DNN in the last iteration. However, we observe that these methods do not explicitly consider the effect of marginal examples on the network training. As the goal of the DNN is to learn features that are more suitable for clustering, the examples near cluster boundaries may not provide convincing guidance. This is in contrast to supervised learning where all target labels are given beforehand and hence all examples can give trustworthy supervisory signals. Actually, marginal examples in supervised learning play a more important role in searching class boundaries. We empirically validate that, in deep clustering, unreliable examples near cluster boundaries could confuse or even mislead the training process of the DNN, leading to unsatisfying performance. On the other hand, these clustering algorithms also overlook the technique of data augmentation which has been widely employed in supervised deep learning models to improve the generalization. Neglecting these two ingredients leads to the failure of leaning robust cluster-oriented features.

- *X. Guo, X. Liu, and E. Zhu are with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: guoxifeng1990@163.com, {xinwangliu, enzhu}@nudt.edu.cn.*
- *X. Zhu is with the College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, Jinhua, Zhengjiang 321004, China. E-mail: zxz@zjnu.edu.cn.*
- *M. Li is with the College of Computer, and Department of Computer, Changsha College, National University of Defense Technology, Changsha 410073, China. E-mail: miaomiaolinudt@gmail.com.*
- *X. Xu is with the Institute of Unmanned Systems, College of Intelligence Science, National University of Defense Technology, Changsha 410073, China. E-mail: xuxin_mail@263.net.*
- *J. Yin is with the Dongguan University of Technology, Guangdong 511700, China. E-mail: jpyin@dgut.edu.cn.*

Manuscript received 6 Aug. 2018; revised 22 Jan. 2019; accepted 10 Apr. 2019. Date of publication 17 Apr. 2019; date of current version 5 Aug. 2020. (Corresponding authors: Xinwang Liu, En Zhu, and Jianping Yin.) Recommended for acceptance by A. Bouguettaya. Digital Object Identifier no. 10.1109/TKDE.2019.2911833

In this paper, we propose an Adaptive Self-Paced deep Clustering with Data Augmentation (ASPC-DA) algorithm which is comprised of two stages: pretraining and finetuning. In the stage of pretraining, we train an autoencoder with augmented data by minimizing the reconstruction loss. As we know, the autoencoder can transform data from relatively high dimensional and sparse space to low dimensional and compact representation space. We use augmented data to enhance the smoothness of the manifold on which the learned representations lie. Then in the second stage, we finetune the encoder (feature extractor) by using a clustering loss defined as the within-cluster sum of squares. To stabilize the training process, we adopt adaptive self-paced learning to select "easy" (near cluster centers) examples as training set and gradually add harder examples. Different from typical self-paced learning, our adaptive variant is free of hyper-parameters and always keeps marginal examples out of training. We also adopt the same type of data augmentation as in the pretraining stage to further facilitate the feature learning. Our experiments show a great competitive edge over state-of-the-art clustering algorithms on various datasets. The main contributions of this paper are highlighted as follows:

- To learn robust cluster-oriented features, we propose a simple but effective deep clustering model that incorporates self-paced learning and data augmentation. To the best of our knowledge, this is the first work to introduce these two well studied techniques in supervised learning into unsupervised deep clustering field.
- We derive an adaptive self-paced learning algorithm without extra hyper-parameter. By using the adaptive self-paced learning, our model can eliminate the negative effect of the examples near cluster boundaries in the process of feature learning.
- Extensive experiments are conducted and the results validate the effectiveness of our ASPC-DA algorithm. The ablation study shows how the adaptive self-paced learning and data augmentation affect the proposed deep clustering algorithm, and provides possible ways to extend existing deep clustering algorithms.

The rest of this paper is organized as follows: Section 2 presents a review of related work. Section 3 is devoted to our ASPC-DA algorithm. Extensive experiments are conducted in Section 4 to support our claims. Some variants are proposed and validated based on our ASPC-DA in Section 5. We finish the paper with the conclusion and potential future work in Section 6.

2 Related Work

Our work falls into the category of deep clustering and incorporates self-paced learning and data augmentation. To facilitate the description of our algorithm, we shall review existing deep clustering algorithms, self-paced learning, and data augmentation techniques in turn.

2.1 Deep Clustering

Deep clustering represents a family of clustering algorithms that adopt deep neural networks to learn cluster-oriented features. As shown in Fig. 1, their loss functions are typically comprised of network loss L_n and clustering loss L_c where the network loss is used to learn feasible features and the clustering loss encourages feature points to form groups. The network loss can be the reconstruction loss of an autoencoder (AE), the variational loss of a variational autoencoder (VAE) [13], or the adversarial loss of a generative adversarial network (GAN) [14]. And the clustering loss can be the loss of any existing clustering algorithms like k-means [1], Gaussian Mixture Model (GMM) [2], and hierarchical clustering [3]. Some work designs a new clustering loss to incorporate the function of network loss, in which case the network loss can be removed. We refer to as Clustering DNN (CDNN) the network that is trained only by clustering loss L_c. Based on the type of DNN, existing deep clustering algorithms can be divided into four categories: AE-based, VAE-based, GAN-based, and CDNN-based deep clustering.

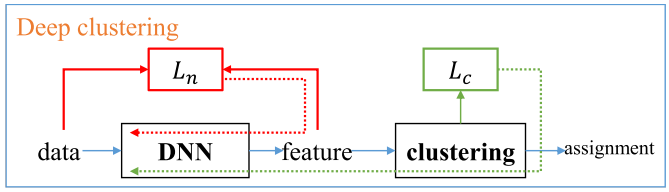

Fig. 1. The basic framework of deep clustering algorithms. The network loss L_n enforces the learned features feasible and clustering loss L_c makes it cluster-friendly. L_n can be removed if L_c is designed to absorb the function of L_n.

AE-based deep clustering has been extensively studied. They directly incorporate the prior knowledge that helps perform clustering into the objective function of an autoencoder. For example, in [12], [15], [16], [17], the authors choose the objective of k-means as the clustering loss L_c. Beyond the hard-assignment k-means loss, Jabi et al. [18] further propose a soft and regularized deep k-means algorithm. Peng et al. [8] incorporate a prior sparsity information into the hidden representation of autoencoders, in order to be adaptive to the local and global subspace structure simultaneously. Dizaji et al. [10] and Guo et al. [11] both borrow the objective of DEC [7] as the clustering loss, but the former utilizes a convolutional autoencoder and the latter uses a fully connected autoencoder. Ji et al. [19] propose to incorporate the self-expressiveness property of subspace clustering into the middle layer of a fully connected autoencoder. The advantage of AE-based deep clustering is that existing shallow clustering algorithms and regularizations can be easily adopted to work with the training of autoencoders. But a hyper-parameter has to be introduced to balance the reconstruction loss and clustering loss.

VAE-based clustering algorithms enforce the latent code to follow a predefined distribution which can describe the cluster structure. Jiang et al. [20] model the latent code by Gaussian mixture model. After maximizing the evidence lower bound, the cluster assignment is inferred by the learned GMM model. Dilokthanakul et al. [21] propose a similar formulation but their results are worse than Jiang et al. [20] empirically. This kind of algorithms are able to generate realistic images in addition to outputting clustering results. However, they suffer from high computational complexity.

GAN-based algorithms share the same idea with VAE-based algorithms. As a typical example, InfoGAN [22] maximizes the mutual information between a fixed small subset of the GAN's noise variables and the observations where the noise variables are sampled from categorical distribution. ClusterGAN [23] uses discrete-continuous mixtures for sampling noise variables of a GAN to enable the clustering in latent space. DASC [24] incorporates the self-expressiveness property of subspace clustering into an adversarial variational autoencoder. Same with [19], DASC is not scalable to large scale dataset. Yu and Zhou [25] introduce a GAN Mixture Model (GANMM) aiming to extend the Gaussian Mixture Model (GMM). GAN-based algorithms have the same problems of GANs, e.g., hard to converge and mode collapse.

The last category of deep clustering, CDNN-based algorithms, explicitly define a loss which is used as the objective of both clustering model and DNNs. Yang et al. [26] propose a recurrent framework with a unified weighted triplet loss which integrates clustering and feature learning processes into a single model. Effective as the algorithm is, it is quite time-consuming due to its recurrent process. Xie et al. [7] propose a Deep Embedded Clustering (DEC) algorithm by defining a KL divergence between distributions P and Q, where Q is the distribution of soft labels measured by Student's t-distribution and P is the target distribution derived from Q. Minimizing the KL divergence can obtain cluster-oriented features and cluster assignments simultaneously. Then Li et al. [27] extend DEC by incorporating convolutional neural networks and modifying the normalization of the target distribution. Peng et al. [9] define the clustering loss as the discrepancy between pair-wise example-centers distributions. They assume that the distribution between a given example and cluster centers is invariant to different distance metrics on the manifold. Based on existing deep clustering frameworks, Guérin and Boots [28] replace singular CNN with multiple pretrained CNNs to extract features. Lin et al. [29] propose a density clustering algorithm employed on pretrained deep features to cluster unconstrained face images. Caron et al. [30], [31] directly use the loss function of k-means to train a convolutional network and achieve promising clustering performance on large scale dataset. This category has a simple and graceful objective which, however, needs to be carefully designed.

However, none of these deep clustering algorithms have explicitly considered the effect of marginal examples. We observe that examples far away from cluster centers are harmful to the training of DNNs. We solve this problem by incorporating adaptive self-paced learning where examples close to cluster borders are totally excluded.

2.2 Self-Paced Learning

Self-paced learning [32] simulates the procedure of human learning: from easy to hard. Given some examples of a new task, we tend to first select easiest examples to learn basic knowledge. After the knowledge about the task improved, we can collect harder examples to acquire more knowledge gradually. At the end of this progress, we may acquire all the knowledge about the task. This strategy of learning is deemed to be more effective. The key problem is how to define "easiness". Depending on current knowledge we have, the closer the answer we give gets to the true answer, the easier the example (or problem) should be.

In machine learning problems, the value of loss function serves as the measure of "easiness". How easy examples should be used for training is controlled by a threshold λ. Formally, given training examples $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and learning model f with parameters $\mathbf{w}$, the traditional machine learning problem is

$$\min_{\mathbf{w}} \sum_{i=1}^{n} L(f_{\mathbf{w}}(x_i), y_i). \tag{1}$$

The the objective of self-paced learning is

$$\begin{aligned} \min_{\mathbf{w},\mathbf{v}} & \sum_{i=1}^{n} v_i L(f_{\mathbf{w}}(x_i), y_i) + g(\lambda, v_i), \\ & s.t. \quad v_i \in [0, 1], \end{aligned} \tag{2}$$

where $\mathbf{v} = [v_1, v_2, \ldots, v_n]^\top$ are weights of examples and $g(\lambda, v_i)$ is called self-paced regularization term. The $\mathbf{w}$ and $\mathbf{v}$ can be optimized using Alternative Search Strategy (ASS). Consider the simple hard-weighting self-paced learning where $g(\lambda, v_i) = -\lambda v_i$ and $v_i \in \{0, 1\}$, i.e.,

$$\begin{aligned} \min_{\mathbf{w},\mathbf{v}} & \sum_{i=1}^{n} v_i L(f_{\mathbf{w}}(x_i), y_i) - \lambda v_i, \\ & s.t. \quad v_i \in \{0, 1\}. \end{aligned} \tag{3}$$

Given example weights $\mathbf{v}$, the minimization over $\mathbf{w}$ is a weighted loss minimization problem. And when the model parameter $\mathbf{w}$ is fixed, the optimal v_i is determined by the closed form

$$v_i = \begin{cases} 1, & \text{if } L_i < \lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Self-paced learning has been successfully used in various applications. Kumar et al. [32] demonstrate that self-paced learning algorithm outperforms the state-of-the-art methods for learning a latent structural SVM on 4 applications: object localization, noun phrase coreference, motif finding and handwritten digit recognition. Then Jiang et al. [33] propose linear self-paced learning for zero-example multimedia search. Mixture self-paced learning algorithm is introduced by Xu et al. [34] to improve the optimization of multi-view clustering problem. Jiang et al. [35] incorporate diversity information by adding a $l_{2,1}$-norm regularization on example weights $\mathbf{v}$. Fan et al. [36] study a group of self-paced implicit regularization term that is deduced from robust loss function.

Self-paced learning algorithms can not avoid searching the best values for hyper-parameters λ and the step size δ that controls the amount of increasing λ at each iteration. In unsupervised learning, hyper-parameters are hard to set. Maybe that's why none of existing work, to the best of our knowledge, incorporates self-paced learning to deep clustering framework. We fill this gap by proposing an adaptive self-paced learning variant which is hyper-

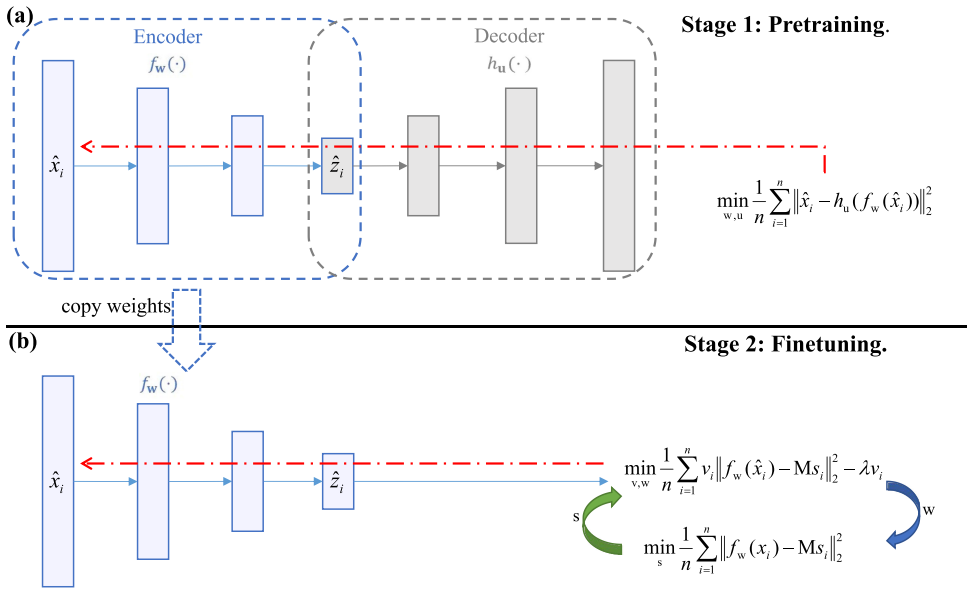

Fig. 2. The proposed ASPC-DA algorithm. The solid line with the filled arrow indicates the data flow in the forwarding path and the dash-dot line with unfilled arrow represents the backpropagation. (a) The stage of pretraining. The fully connected autoencoder is trained by minimizing the Mean Squared Error (MSE) loss L_r between the input augmented examples $\hat{x}_i$ and output reconstructed ones $h_{\mathbf{u}}(f_{\mathbf{w}}(\hat{x}_i))$. (b) The stage of finetuning. We first excerpt the encoder from the pretrained autoencoder and obtain the feature points z_i of the input clean examples x_i. The clean features z_i are fed to the k-means algorithm to determine the partitioning, i.e., cluster centers M. Then, we alternatively update $\mathbf{w}$, $\mathbf{v}$, and the cluster assignment s_i. We use the augmented examples $\hat{x}_i$ to finetune the encoder and clean examples x_i to update the cluster assignment s_i.

parameter free and is easy to be plugged into deep clustering models.

2.3 Data Augmentation

As we all know, DNNs are easy to overfit the data which can be solved by resorting to acquiring more training data, restricting the capacity of DNNs, or early stopping. Data augmentation is an effective way to expanding training datasets. For a given image example, we transform it by random rotating, shifting, cropping, shearing, etc. before feeding it to the network. In this way, DNNs have to fit more examples than the original dataset. Data augmentation is in fact introducing the prior knowledge that transforming an image in appropriate ways does not change its identity. In most deep learning models, e.g., AlexNet [37], ResNet [38], and CapsNet [39], data augmentation plays an important role to improve the generalization. However, in unsupervised cluster analysis, data augmentation has been overlooked, to the best of our knowledge. We give a simple and intuitive solution to bring data augmentation to deep clustering problem and validate the effectiveness of this solution.

3 Adaptive Self-Paced Deep Clustering with Data Augmentation

In this section, we first introduce the basic two-stage deep clustering model in which a new clustering loss is defined. Then in Section 3.2, we develop an adaptive self-paced learning algorithm to stabilize the training of the proposed deep clustering model by excluding the examples near cluster boundaries. Furthermore, we incorporate data augmentation technique into both of stages of the basic model in Section 3.3. After introducing these two techniques, we obtain our final algorithm termed Adaptive Self-Paced deep Clustering with Data Augmentation (ASPC-DA). Finally, we elaborate the optimization procedure for our ASPC-DA in Section 3.4. We also give an example to demonstrate how our algorithm works in practice in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2019.2911833. The whole structure of our ASPC-DA is illustrated in Fig. 2.

3.1 Basic Deep Clustering Model

Consider a dataset with n examples $X = \{x_i \in \mathbb{R}^D\}_{i=1}^n$. An encoder network $f_{\mathbf{w}}(\cdot)$ transforms each example $x_i \in \mathbb{R}^D$ to $z_i \in \mathbb{R}^d$ and a decoder network $h_{\mathbf{u}}(\cdot)$ transforms z_i back to $x_i' \in \mathbb{R}^D$. The number of clusters K is given as a priori knowledge and the center of the jth cluster is represented by $\mathbf{m}_j \in \mathbb{R}^d$. Define cluster matrix as $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K] \in \mathbb{R}^{d\times K}$. Let $s_i \in \{0,1\}^K$ denote the cluster index assigned to the example z_i. Then $y_i = \mathbf{M}s_i$ indicates the center of the cluster to which the example z_i belongs.

We aim to find a good neural network (feature extractor) $f_{\mathbf{w}}(\cdot)$ which enables the transformed points $\{z_i\}_{i=1}^n$ to be more suitable for clustering. To this end, we propose a two-stage deep clustering model to train $f_{\mathbf{w}}(\cdot)$.

3.1.1 Pretraining

In the first stage, we stack a decoder network $h_{\mathbf{u}}(\cdot)$ on the top of $f_{\mathbf{w}}(\cdot)$ to build an autoencoder which is trained by minimizing the reconstruction loss

$$L_r = \frac{1}{n}\sum_{i=1}^{n}\|x_i - h_{\mathbf{u}}(f_{\mathbf{w}}(x_i))\|_2^2. \tag{5}$$

The resulting transformed point $z_i = f_{\mathbf{w}}(x_i)$ is in a much lower dimensional feature space than x_i. So the encoder can serve as a dimensionality reduction (DR) method and is expected to be more powerful than other DR methods like Principal Component Analysis (PCA), Isomap, and Locally

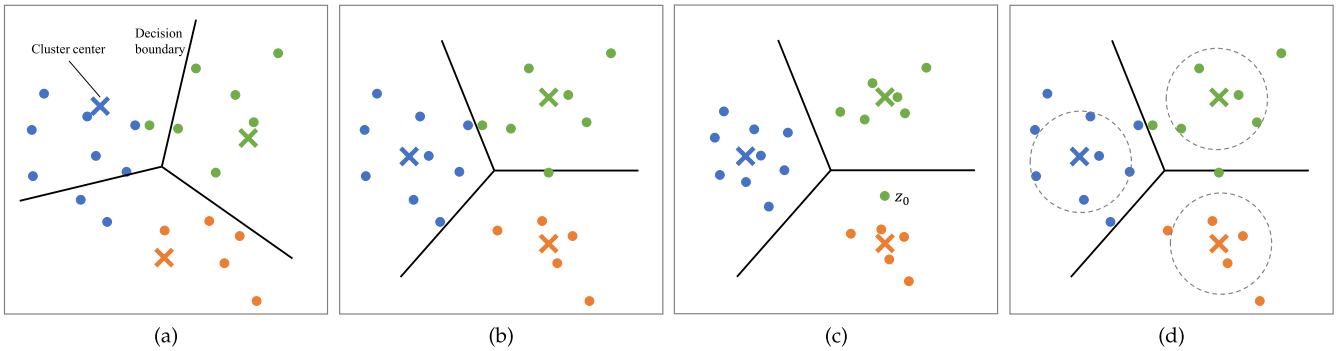

Fig. 3. The process of finetuning stage. Circles "o" indicate examples and crosses "×" represent cluster centers. Different clusters are rendered with different colors. Solid lines indicate decision boundaries which are perpendicular bisectors of adjacent cluster centers. (a) Random initialization of the cluster centers. (b) The result of k-means. It determines the best positions of cluster centers as well as the decision boundaries. (c) Training the encoder with cluster centers (as well as the decision boundaries) fixed. The examples in the same cluster are pulled closer to the cluster center in the feature space. But example z_0 near boundary is easily pulled towards the incorrect cluster. (d) Different with (b), training the encoder by using reliable examples (those in the dashed circles) is more meaningful.

Linear Embedding (LLE). That is, the z_i is a low dimensional and compact representation for the example x_i. Performing clustering on $\{z_i\}_{i=1}^n$ is more effective and efficient. But the training of the autoencoder is not task-specific (or clustering-specific), thus the feature point z_i is not necessarily suitable for clustering. Our solution is to further finetune the encoder by using a clustering loss.

3.1.2 Finetuning

After pretraining, we propose to finetune the network by using the following objective:

$$\begin{aligned} &\min_{\mathbf{w},\mathbf{s}} \frac{1}{n}\sum_{i=1}^{n} \|f_{\mathbf{w}}(x_i) - \mathbf{M}s_i\|_2^2, \\ &s.t.\ s_i \in \{0,1\}^K,\ \mathbf{1}^\top s_i = 1, \end{aligned} \tag{6}$$

where $\mathbf{M} \in \mathbb{R}^{d\times K}$ is a matrix comprising all cluster centers, s_i is the cluster assignment of the ith example, and $\mathbf{1}$ is a column vector with every element being 1. As we can notice, this objective is identical to the objective of k-means algorithm except for the parameters to optimize. The objective of k-means is

$$\begin{aligned} &\min_{\mathbf{M},\mathbf{s}} \frac{1}{n}\sum_{i=1}^{n} \|f_{\mathbf{w}}(x_i) - \mathbf{M}s_i\|_2^2, \\ &s.t.\ s_i \in \{0,1\}^K,\ \mathbf{1}^\top s_i = 1, \end{aligned} \tag{7}$$

which separates data points by alternatively updating cluster centers $\mathbf{M}$ and assignments $\mathbf{s}$, i.e., by gradually adjusting the decision boundaries as shown from Figs. 3a to 3b. We shall use k-means to initialize the cluster centers $\mathbf{M}$ and then optimize (6). The main reason of fixing $\mathbf{M}$ in (6) after initialization is to avoid the degenerate solution in which all examples converge to one point and the objective equals zero. Once $\mathbf{M}$ is fixed, decision boundaries are also determined. Because decision boundaries are perpendicular bisectors of adjacent cluster centers. Then it will be impossible to squash all examples together by optimizing (6) (see from Figs. 3b to 3c). We shall call this deep clustering model BasicDC.

We can alternatively optimize (6) regarding $\mathbf{w}$ and $\mathbf{s}$. When $\mathbf{s}$ is fixed, (7) degrades to

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \|f_{\mathbf{w}}(x_i) - y_i\|_2^2, \tag{8}$$

where $y_i = \mathbf{M}s_i$ is the center of the cluster that example x_i belongs to in feature space. This objective can be interpreted as the objective of supervised learning for the DNN. The target label of example x_i is its cluster center y_i. So minimizing (8) forces examples in feature space to scatter in groups. Note that squashing examples in the same cluster does not guarantee that these examples remain in the same cluster forever, since the neural network $f_{\mathbf{w}}$ has to model the structure of all clusters.

On the other hand, the target y_i of the example x_i is determined by using feature points z in the last iteration. Therefore the target y_i is not 100 percent correct, especially for examples near cluster borders. This is in contrast to supervised learning where targets y are given along with examples x. The false target will misguide the training of the neural network (see the point z_0 in Fig. 3c). What makes it worse is that the gradient of (8) with respect to $f_{\mathbf{w}}(x_i)$ is proportional to $(f_{\mathbf{w}}(x_i) - y_i)$. That is, the farther away the feature point is from its cluster center, the larger gradient will be fed back to the network. This is obviously not what we want. We would rather only use the target with high confidence to tune our network. We accomplish this goal by deriving an adaptive self-paced learning algorithm.

3.2 Incorporating Adaptive Self-Paced Learning

To stabilize the optimization of (8), we incorporate self-paced learning to select the most confident examples gradually. Substituting (8) into (3) to get the objective

$$\min_{\mathbf{v},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} v_i\|f_{\mathbf{w}}(x_i) - y_i\|_2^2 - \lambda v_i, \quad s.t. \quad v_i \in \{0,1\}, \tag{9}$$

where $\mathbf{v} = [v_1, v_2, \ldots, v_n]^\top$ are weights of training examples, and λ is the age parameter which controls the number of selected examples.

As introduced in Section 2.2, typical self-paced learning selects all examples into training set at the end of training. But in our problem, targets of examples on cluster borders are not reliable. Therefore, we shall prevent the self-paced

learning from selecting examples near boundaries, even at the end of training. Another drawback of traditional self-paced learning is introducing two additional hyper-parameters: the age parameter λ for controlling the learning pace and step size δ for increasing λ. Typically λ is set to the median of losses at beginning. Then λ increases by a step δ every several iterations. Since losses of examples are also decreasing during training, step size δ is difficult to choose.

Based on the above analysis, we propose to set λ according to the statistics of losses during training

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

where L^t denotes all losses in the tth iteration, T is the number of maximal iterations, $\mu(\cdot)$ and $\sigma(\cdot)$ are average and standard deviation of losses. As T is determined by the deep learning model, the λ now is adaptive to the losses of examples, not an independent hyper-parameter any more. Fig. 3d shows examples selected by using the adaptive self-paced learning at the beginning.

Substituting $y_i = \mathbf{M}s_i$ into (9), we get the objective of Adaptive Self-Paced deep Clustering algorithm (ASPC)

$$\begin{aligned} &\min_{\mathbf{v},\mathbf{W}} \frac{1}{n}\sum_{i=1}^{n} v_i \|f_{\mathbf{W}}(x_i) - \mathbf{M}s_i\|_2^2 - \lambda v_i, \\ &s.t.\ v_i \in \{0,1\}, \end{aligned} \tag{11}$$

and

$$\begin{aligned} &\min_{\mathbf{s}} \frac{1}{n}\sum_{i=1}^{n} \|f_{\mathbf{W}}(x_i) - \mathbf{M}s_i\|_2^2, \\ &s.t.\ s_i \in \{0,1\}^K,\ \mathbf{1}^\top s_i = 1, \end{aligned} \tag{12}$$

where λ is computed by (10).

3.3 Incorporating Data Augmentation

Since data augmentation, such as random rotation, shifting, and cropping, has been widely employed in supervised deep learning community, we shall attempt to explore it in unsupervised learning too. In this section, we give a direct and intuitive way of incorporating data augmentation to both of stages of the proposed deep clustering algorithm. For the sake of convenience, we define a mapping $\mathcal{T}$ to represent data augmentation function which can be any combination of random rotation, shifting, cropping, wrapping, etc. Then the augmented example is $\hat{x}_i = \mathcal{T}(x_i)$.

3.3.1 Pretraining Stage

We directly replace the clean example x_i in (5) with the augmented one $\hat{x}_i$ to obtain the reconstruction loss

$$L_r = \frac{1}{n}\sum_{i=1}^{n} \|\hat{x}_i - h_{\mathbf{u}}(f_{\mathbf{W}}(\hat{x}_i))\|_2^2. \tag{13}$$

Since autoencoders take the input examples as targets, this form is analogous to supervised learning. Thus it is intuitive to adopting (5), (6), (7), (8), (9), (10), (11), (12), and (13). Also note the fact that augmented examples share the same manifold with original examples. Ideally, the manifold learned by using the augmented examples should be more continuous and smooth than that learned by clean examples. That is to say, incorporating data augmentation can help the autoencoder learn more representative and robust features.

3.3.2 Finetuning Stage

In (11) and (12), there are three parameters, $\mathbf{w}, \mathbf{v}, \mathbf{s}$, corresponding to feature learning, reliable examples selection, and cluster assignment, respectively. As our ultimate goal is to improve the clustering performance, we shall compute assignments $\mathbf{s}$ for clean examples instead of augmented ones. Once $\mathbf{s}$ is computed, the target for the example x_i will be $y_i = \mathbf{M}s_i$. As same as in supervised deep learning, the clean example x_i should share the target y_i with the augmented counterpart $\hat{x}_i$. So feature learning (updating $\mathbf{w}$) can use augmented data. In summary, we shall utilize the augmented examples only in (11). The final objectives of our Adaptive Self-Paced deep Clustering with Data Augmentation (ASPC-DA) are (12) and

$$\begin{aligned} &\min_{\mathbf{v},\mathbf{W}} \frac{1}{n}\sum_{i=1}^{n} v_i \|f_{\mathbf{W}}(\hat{x}_i) - \mathbf{M}s_i\|_2^2 - \lambda v_i, \\ &s.t.\ v_i \in \{0,1\}, \end{aligned} \tag{14}$$

where λ is governed by (10). Note that the only difference between (11) and (14) is x_i versus $\hat{x}_i$.

3.4 Optimization

The pretraining stage can be directly optimized by using SGD and backpropagation, so we only focus on the finetuning stage. We alternatively optimize (12) and (14) with respect to $\mathbf{s}, \mathbf{v}, \mathbf{w}$.

3.4.1 Update $\mathbf{s}$ with $\mathbf{w}$ Fixed

When $\mathbf{w}$ is fixed in (12), we can immediately solve $\mathbf{s}$ in closed form

$$s_{ij} = \begin{cases} 1, & \text{if } j = \arg\min_k \|f_{\mathbf{W}}(x_i) - \mathbf{m}_k\|_2^2, \\ 0, & \text{otherwise}, \end{cases} \tag{15}$$

where s_{ij} is the entry at the ith row jth column of $\mathbf{s}$, and $\mathbf{m}_k$ is the kth column of $\mathbf{M}$, i.e., the center of the kth cluster. Not surprisingly, this solution is identical to k-means'.

3.4.2 Update $\mathbf{v}$ with $\mathbf{w}, \mathbf{s}$ Fixed

When $\mathbf{w}, \mathbf{s}$ is fixed in (14), the optimal $\mathbf{v}$ can be easily obtained in closed form

$$v_i = \begin{cases} 1, & \text{if } \|f_{\mathbf{W}}(\hat{x}_i) - \mathbf{M}s_i\|_2^2 < \lambda, \\ 0, & \text{otherwise}, \end{cases} \tag{16}$$

where λ is computed by (10).

3.4.3 Update $\mathbf{w}$ with $\mathbf{v}, \mathbf{s}$ Fixed

With $\mathbf{v}, \mathbf{s}$ fixed, (14) is simplified to

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} v_i \|f_{\mathbf{W}}(\hat{x}_i) - \mathbf{M}s_i\|_2^2, \tag{17}$$

which is consistent with supervised learning. Thus we can resort to backpropagation and SGD.

TABLE 1
Dataset Statistics

Dataset	# Points	# Classes	Size	Dimension
MNIST-full	70,000	10	28×28	784
MNIST-test	10,000	10	28×28	784
USPS	9,298	10	16×16	256
Fashion	70,000	10	28×28	784

3.4.4 Stopping Criterion

If the change of predicted cluster labels between two successive iterations is smaller than a threshold δ, we will stop the training. Formally, the stopping criterion is

$$1 - \frac{1}{n}\sum_{i,j} s_{ij}^{t} s_{ij}^{t-1} < \delta, \tag{18}$$

where s_{ij}^{t-1} and s_{ij}^{t} are indicators for whether example x_i is assigned to the jth cluster at the $(t-1)$th and tth iteration, respectively. We empirically set $\delta = 0.001$ in our experiment. The whole algorithm is summarized in Algorithm 1.

Algorithm 1. Adaptive Self-Paced Deep Clustering with Data Augmentation (ASPC-DA)

Input: Dataset X; Augmentation mapping $\mathcal{T}$; Number of clusters K; Stopping threshold δ; Maximum iterations T.
Output: Cluster assignments $\mathbf{s}$.

```
1: Initialize w by minimizing (13) where x̂_i = T(x_i);
2: Initialize M, s by deploying k-means on f_w(x_i), i.e., the
   output of the pretrained encoder;
3: for t = 0 to T do
4:    Update cluster assignments s by (15);
5:    Update example weights v by (16);
6:    Update network parameters w by optimizing (17);
7:    if Stopping criterion (18) is met then
8:       Stop training.
9:    end if
10: end for
```

3.4.5 Time Complexity

Suppose the maximum number of neurons in each layer of the autoencoder is $\tilde{D}$ and the maximum epochs for pretraining is T_1, then the time complexity of the pretraining stage is $\mathcal{O}(T_1 n\tilde{D}^2)$ (line 1 of Algorithm 1). The k-means (line 2) is $\mathcal{O}(T_2 ndK)$ where T_2 is the maximum iterations. The time complexity of updating $\mathbf{s}, \mathbf{v}, \mathbf{w}$ (line 4-6) is $\mathcal{O}(Tnd + Tn\tilde{D}^2 + TndK)$. So the total time of Algorithm 1 is $\mathcal{O}(T_1 n\tilde{D}^2 + T_2 ndK + Tn\tilde{D}^2 + TndK + Tnd) = \mathcal{O}((T_1 + T)n\tilde{D}^2 + (T_2 + T)ndK)$ which is linear to the number of examples n. Therefore, the proposed ASPC-DA algorithm is theoretically efficient and scalable.

4 Experiments

We conduct extensive experiments to validate the effectiveness of the proposed ASPC-DA algorithm. After introducing the basic experimental settings, we present the comparison with state-of-the-art clustering methods. Then we conduct ablation study to explore the effect of each part of our ASPC-DA. At last we provide sensitivity analysis for some hyper-parameters.

4.1 Experimental Settings

All experiments are conducted on a PC with Intel Core i7-7700 CPU @ 4.00 GHz and a NVIDIA GTX 1080Ti GPU.

4.1.1 Datasets

We conduct experiments on four image datasets:

- MNIST-full: A dataset consisting of 10 handwritten digits [40], total 70,000 examples. Each example is a 28x28 gray image.
- MNIST-test: A dataset that only contains the test set of MNIST-full, with 10,000 examples.
- USPS[1]: A dataset contains 9,298 gray digit images with size of 16x16 pixels divided into 10 categories.
- Fashion: A dataset of Zalando's article images, consisting of 70,000 examples each of which is a 28 x 28 gray image, divided into 10 classes [41].

The statistics of these datasets are summarized in Table 1. All datasets are rescaled to [0,1] for each element before being fed to clustering algorithms.

4.1.2 Baseline Methods

The proposed ASPC-DA is compared with both conventional shallow clustering algorithms and state-of-the-art deep clustering. Shallow ones include k-means [1], normalized-cut spectral clustering (SC) [42], locality preserving non-negative matrix factorization (NMF-LP) [43], agglomerative clustering (AC) [3], Gaussian mixture models (GMM) [2], and robust continuous clustering (RCC) [44]. Deep clustering algorithms include DeepCluster [30], [31], deep clustering networks (DCN) [12], deep k-means (DKM) [17], deep embedded clustering (DEC) [7], improved deep embedded clustering (IDEC) [11], cascaded subspace clustering (CSC) [9], soft regularized k-means (SR-k-means) [18], variational deep embedding (VaDE) [20], clustering with GAN (ClusterGAN) [23], joint unsupervised learning (JULE) [26], deep embedded regularized clustering (DEPICT) [10], discriminatively boosted clustering (DBC) [27], and deep adaptive clustering (DAC) [45]. We report the results by excerpting from the corresponding papers or by running their released code when available.

4.1.3 Evaluation Metrics

We evaluate all clustering methods with clustering ACCuracy (ACC) and Normalized Mutual Information (NMI). The ACC is defined as the best match between ground truth $\mathbf{y}$ and predicted labels $\mathbf{c}$

$$\mathrm{ACC}(\mathbf{y}, \mathbf{c}) = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\{y_i = g(c_i)\}}{n}, \tag{19}$$

where y_i and c_i are the ground truth and predicted label of example x_i respectively, and g is a one to one mapping from predicted label to ground truth label. The best mapping can be efficiently computed by the Hungarian algorithm [46]. Note that predicted label c_i can be calculated according to

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

TABLE 2
Summary of Some Key Hyper-Parameters Used in Our ASPC-DA Algorithm

Stage	Parameter	Value
Pretraining	Neurons	$[500, 500, 2000, 10, 2000, 500, 500, D]$
	Optimizer	SGD (lr=1.0, momentum=0.9)
	Epochs	500
	Batch size	256
Finetuning	Neurons	[500,500,2000,10]
	Optimizer	Adam (lr=0.0001)
	Iterations T	100
	Batch size	256
	Threshold δ	0.001
	Mapping $\mathcal{T}$	Random shift $\frac{\sqrt{D}}{10}$ pixels & rotation 10°

cluster assignment s_i, i.e., $c_i = \arg\max_j s_{ij}$. The NMI is defined as

$$\mathrm{NMI}(\mathbf{y}, \mathbf{c}) = \frac{2I(\mathbf{y}, \mathbf{c})}{H(\mathbf{y}) + H(\mathbf{c})}, \tag{20}$$

where I and H are mutual information and entropy, respectively. Both of ACC and NMI are in the range $[0, 1]$. The higher value corresponds to the better clustering performance.

4.1.4 Implementation Details

In the pretraining stage, the autoencoder is composed of eight fully connected layers with dimensions $D - 500 - 500 - 2000 - 10 - 2000 - 500 - 500 - D$, where D is the dimension of input examples. Except for the input, output, and embedding (with 10 neurons) layers, all internal layers are activated by ReLU [47]. The autoencoder is trained in an end-to-end manner for 500 epochs by using SGD optimizer with learning rate (lr) 1.0 and momentum 0.9. We do not employ tedious layer-wise pretraining [48] or any other regularization techniques like Dropout [49], Batch Normalization [50]. During finetuning, the encoder has four layers with dimensions $D - 500 - 500 - 2000 - 10$. The maximum number of iterations is set to $T = 100$. In each iteration, we train the encoder for one epoch using Adam [51] optimizer with initial learning rate 0.0001. The batch size is fixed to 256. The threshold in stopping criterion is $\delta = 0.001$. The transform function $\mathcal{T}$ for data augmentation is the combination of random rotation for up to 10° and random shifting for up to $\sqrt{D}/10$ (i.e., 10 percent of the width or height of the input image) pixels in each direction. The input examples are represented as D-dimensional vectors which will be reshaped to $\sqrt{D} \times \sqrt{D}$ matrices. Then we can rotate and shift the matrices by using $\mathcal{T}$. At last the transformed matrices are flattened back to D-dimensional vectors which can be fed to the autoencoder. A summary of these hyper-parameters is listed in Table 2. We fix the above settings over all datasets and avoid dataset specific tuning. On each dataset, we run our algorithm for 5 times and report the average result. Our implementation is based on Python and Keras [52]. The source code is publicly available on https://github.com/XifengGuo/ASPC-DA.

4.2 Comparisons with State-of-the-Art Methods

The clustering results of all methods are reported in Table 3. As we can see, in most cases, deep clustering algorithms (from DCN [12] to ASPC-DA) outperform shallow ones (from k-means [1] to RCC [44]) by a large margin. Comparison among deep clustering algorithms validates the superiority of algorithms using convolutional networks (from ClusterGAN [23] to DAC [45]) over that using fully connected networks (from DeepCluster [30] to VaDE [20]). But our ASPC-DA algorithm achieves the best performance in terms of all metrics on all datasets except ACC on Fashion.

TABLE 3
Clustering Performances of Different Algorithms in Terms of ACC and NMI

	MNIST-full		MNIST-test		USPS		Fashion	
	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
k-means [1]	0.532	0.500	0.546	0.501	0.668	0.627	0.474	0.512
SC [42]	0.656	0.731	0.660	0.704	0.649	0.794	0.508	0.575
NMF-LP [43]	0.471	0.452	0.479	0.467	0.652	0.693	0.434	0.425
AC [3]	0.621	0.682	0.695	0.711	0.683	0.725	0.500	0.564
GMM [2]	0.433	0.366	0.540	0.493	0.551	0.530	0.556	0.557
RCC [44]	–	0.893*	–	0.828	–	0.742	–	0.614
DeepCluster [30]	0.797	0.661	0.854	0.713	0.562	0.540	0.542	0.510
DCN [12]	0.830*	0.810*	0.802	0.786	0.688	0.683	0.501	0.558
DKM [17]	0.840*	0.796*	–	–	0.757*	0.776*	–	–
DEC [7]	0.863	0.834	0.856	0.830	0.762	0.767	0.518	0.546
IDEC [11]	0.881*	0.867*	0.846	0.802	0.761*	0.785*	0.529	0.557
CSC [9]	0.872*	0.755*	0.865*	0.733*	–	–	–	–
SR-k-means [18]	0.939*	0.866*	0.863*	0.873*	0.901	0.912	0.507	0.548
VaDE [20]	0.945	0.876	0.287	0.287	0.566	0.512	0.578	0.630
ClusterGAN [23]	0.950*	0.890*	–	–	–	–	**0.630***	0.640*
JULE [26]	0.964*	0.913*	0.961*	0.915*	0.950	0.913	0.563	0.608
DEPICT [10]	0.965*	0.917*	0.963*	0.915*	0.899	0.906	0.392	0.392
DBC [27]	0.964*	0.917*	–	–	–	–	–	–
DAC [45]	0.978*	0.935*	–	–	–	–	–	–
ASPC-DA	**0.988 ± 0.001**	**0.966 ± 0.002**	**0.973 ± 0.003**	**0.936 ± 0.003**	**0.982 ± 0.001**	**0.951 ± 0.002**	0.591 ± 0.022	**0.654 ± 0.021**

All results of baseline algorithms are reported by running their released code except the ones marked by (∗) on top which are excerpted from the corresponding paper. The mark "–" denotes that the result is unavailable from the paper or the code.

TABLE 4
Running Time (in Seconds) of Different Algorithms

	MNIST-full	MNIST-test	USPS	Fashion
k-means	150	16	4	108
SC [42]	500	110	35	480
NMF-LP [43]	143	15	5	115
AC [3]	1162	22	7	1171
GMM [2]	1300	106	31	830
DeepCluster [30]	2200	120	100	1800
DCN [12]	1000	110	68	780
DEC [7]	1100	1170	900	1060
IDEC [11]	1400	800	560	1200
SR-k-means [18]	14000	1500	1400	4500
VaDE [20]	3000	420	90	2800
JULE [26]	20000	3600	2800	14000
DEPICT [10]	9000	2100	1600	7680
ASPC-DA	925	129	92	877

The state-of-the-art performance on the most widely used MNIST-full dataset is increased by our algorithm from 97.8 to 98.8 percent in terms of ACC and from 93.5 to 96.6 percent regarding NMI.

It is worth emphasizing that our ASPC-DA algorithm does not tune hyper-parameters for different datasets. This consistent setting makes ASPC-DA easier be adapted to new datasets. In contrast, DEC [7] introduces a hyper-parameter to control the frequency of updating the target distribution, which needs to be tuned for each dataset. CSC [9] tunes 4 hyper-parameters for each dataset as shown in Table 1 in their paper. DCN [12] designs an individual network structure for each dataset and tunes 4 additional parameters.

In Table 4, we report the running time of different algorithms. Note that the results of some algorithms are absent due to that their codes are not publicly available. As can be seen, some conventional clustering algorithms (from k-means [1] to NMF-LP [43]) are efficient but ineffective. Our ASPC-DA is computationally comparable to or even better than other deep clustering algorithms (from DeepCluster [30] to DEPICT [10]). Especially, JULE [26] and DEPICT [10] spend up to ten times as long as our ASPC-DA. To conclude, in addition to achieving the state-of-the-art clustering accuracy, our ASPC-DA algorithm remains high computational efficiency.

4.3 Contributions of Different Parts of ASPC-DA

In this section, we perform ablation study to analyze the contributions of three parts of the proposed ASPC-DA: data augmentation in the pretraining stage (DA-S1), data augmentation in the finetuning stage (DA-S2), and the adaptive self-paced learning (ASP). Removing DA-S1 or DA-S2 from ASPC-DA means replacing $\hat{x}$ with x (or setting $\mathcal{T}$ as identity mapping) in (13) or (14). And disabling ASP in ASPC-DA corresponds to fixing $\mathbf{v} = \mathbf{1}$ in (14). We regard the configuration of removing all of the three parts as the BasicDC corresponding to Section 3.1.

Table 5 shows the results of ASPC-DA with different configurations. When we individually add one of DA-S1, DA-S2 and ASP to the BasicDC, the performance improves in most cases. And the most considerable improvement is seen by adding DA-S1. In fact, the results at the last four rows are consistently better than that at the first four rows. This result reveals that the DA-S1 is the most crucial part of our ASPC-DA algorithm. When adding two of the three parts together, there is a huge performance improvement except for the combination of AE-S2 and ASP where the performances on MNIST-test, USPS, and Fashion degenerate abnormally. The possible reason is that the feature space constructed by clean examples in the pretraining stage is easy to be corrupted by augmented examples that are first exposed to the neural network in the finetuning stage. The ASP further prompts the corruption of the feature space by only feeding part of augmented examples. Finally, by being equipped with all of the three parts, our ASPC-DA succeeds to achieve the best performance on all datasets. We further validate the effectiveness of BasicDC on more datasets in Appendix B, available in the online supplemental material.

Based on the above analysis, we can directly use data augmentation to extend any existing deep clustering algorithm which involves pretraining an autoencoder. If there exists a process of setting and updating "targets" of examples, we can incorporate data augmentation and ASP as same as in Sections 3.2 and 3.3. We shall avoid introducing data augmentation only to the finetuning stage.

4.4 Convergence

We record the variation of two loss values of our method with iterations, i.e., the feature learning loss $L(\mathbf{v}, \mathbf{w})$ and clustering loss $L(\mathbf{s})$. They are defined as

TABLE 5
Performances of ASPC-DA with Different Configurations

	DA-S1	DA-S2	ASP	MNIST-full		MNIST-test		USPS		Fashion	
				ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
BasicDC	×	×	×	0.912	0.857	0.764	0.709	0.701	0.721	0.562	0.633
ASPC	×	×	✓	0.924	0.881	0.785	0.757	0.688	0.748	0.561	0.627
–	×	✓	×	0.958	0.936	0.668	0.775	0.693	0.788	0.542	0.614
–	×	✓	✓	0.945	0.938	0.531	0.690	0.552	0.711	0.533	0.608
–	✓	×	×	0.972	0.931	0.930	0.867	0.953	0.896	0.592	0.650
–	✓	×	✓	0.978	0.942	0.947	0.887	0.958	0.901	**0.596**	0.652
–	✓	✓	×	0.987	0.963	0.970	0.931	0.977	0.939	0.584	0.651
ASPC-DA	✓	✓	✓	**0.988**	**0.966**	**0.973**	**0.936**	**0.982**	**0.951**	0.591	**0.654**

DA-S1 (or DA-S2) represents data augmentation in the pretraining (or finetuning) stage. ASP denotes adaptive self-paced learning. Whether to use a specific part (DA-S1, DA-S2, or ASP) in ASPC-DA algorithm is marked by "✓" (yes) or "×" (no).

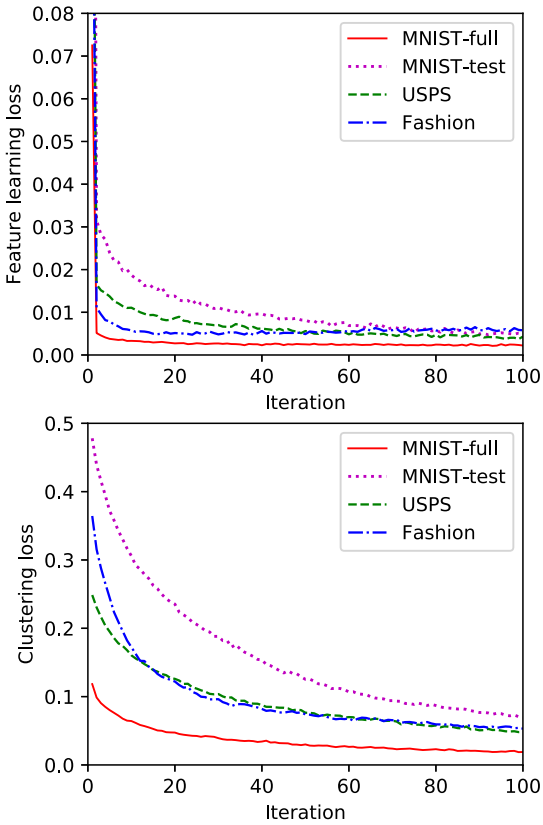

Fig. 4. Training losses. Both feature learning loss and clustering loss keep decreasing for all datasets, which indicates our method is able to converge in practice.

$$L(\mathbf{v}, \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} v_i \| f_{\mathbf{w}}(\hat{x}_i) - \mathbf{M}s_i \|_2^2 - \lambda v_i, \tag{21}$$

$$L(\mathbf{s}) = \frac{1}{n}\sum_{i=1}^{n} \| f_{\mathbf{w}}(x_i) - \mathbf{M}s_i \|_2^2. \tag{22}$$

The results are plotted in Fig. 4. For all four datasets in our experiments, both feature learning loss (21) and clustering loss (22) decrease quickly toward 0 when the training iteration increases. That is to say, our method usually converges in practice.

We further investigate the convergence of our algorithm by visualizing the learned features in different time of the optimization. We randomly sample 1,000 examples from MNIST-full dataset, and use PCA to project them into two-dimension space from raw pixel space, feature space with iteration $t = 0, 10, 100$ during finetuning stage, respectively. As shown in Fig. 5, data points projected from raw pixel space are highly overlapped, implying the difficulty of the clustering. After the pretraining stage, i.e., at the beginning of the finetuning stage, feature points extracted from the encoder are more separable than raw examples. But there are still many inseparable points. As the finetuning process goes, say at the tenth iteration, most data points in the same cluster have crowded together, and only a few points are near cluster borders. At the 100th iteration, feature points reach stability. We also show the t-SNE [53] visualization of the final result which confirms the final feature points are nicely separated. The

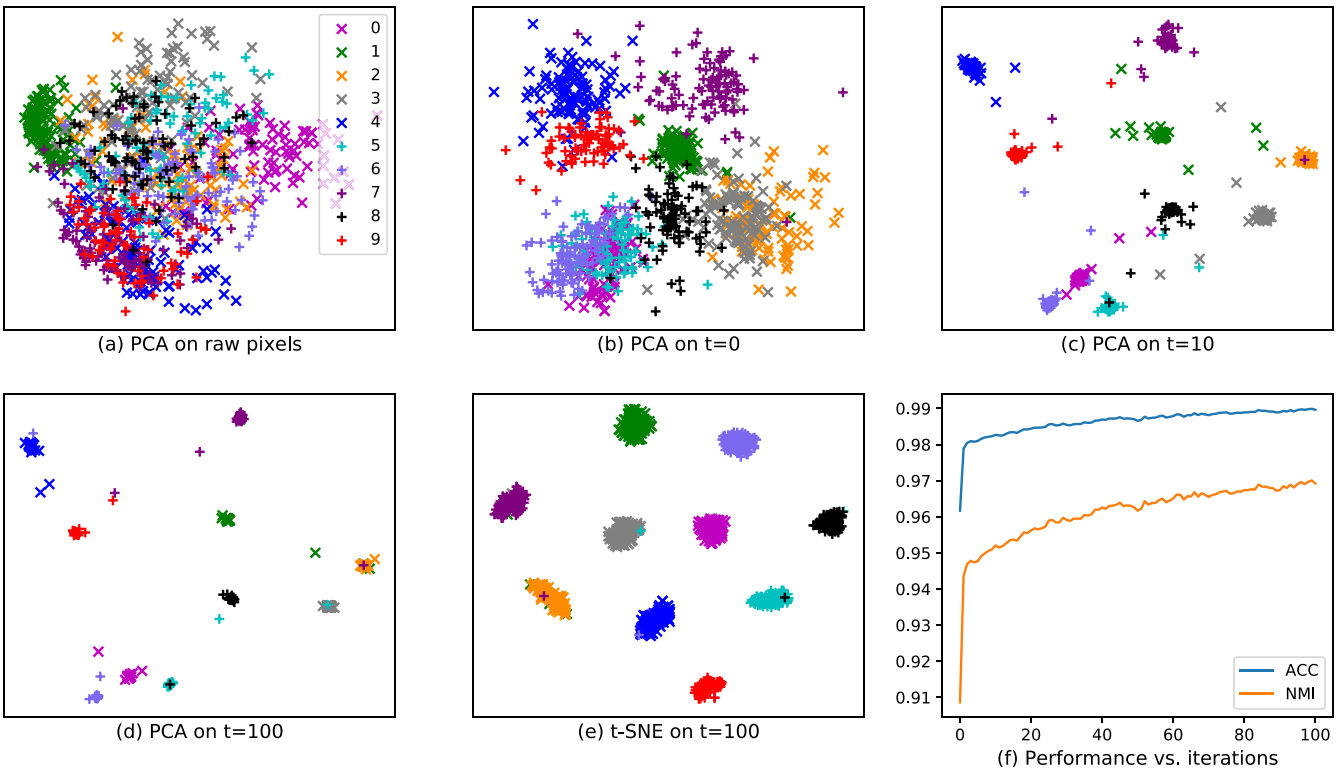

Fig. 5. Visualization on a subset of MNIST-full with 1,000 examples for different phase of our algorithm. (a) PCA results on examples in raw pixel space. The embedded points are highly overlapped. (b) PCA results on features extracted from the network at the beginning of finetuning. Data points are more separable than (a) and this validate the effectiveness of the pretraining stage. (c) PCA on features when the network is finetuned for 10 iterations. (d) PCA results and (e) t-SNE results on features at the end of finetuning stage. Most data points are perfectly separated. (f) The trend of ACC and NMI regarding the iterations of finetuning.

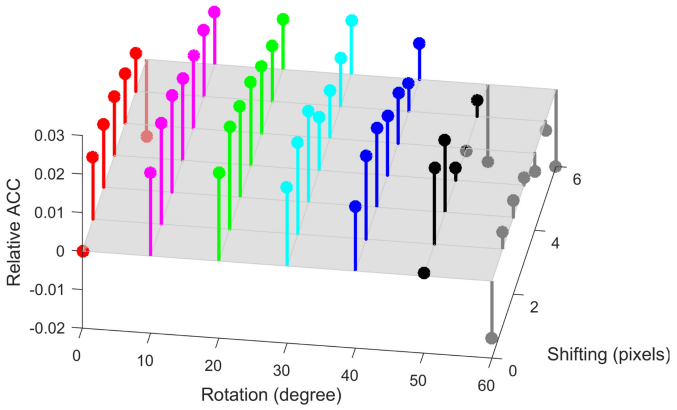

Fig. 6. Sensitivity analysis for the transformations (rotation and shifting) used in data augmentation on MNIST-full. Relative ACC is the value relative to the performance at rotation 0 degree and shifting 0 pixel. In a wide range, our ASPC-DA shows performance improvement regarding the baseline.

performance keeps increasing in the first few iterations and reaches stability at last, as shown in Fig. 5f.

4.5 Sensitivity of Hyper-Parameters

As shown in Table 2, there are some hyper-parameters which can not be avoided when involving DNNs. It is impossible to search in the whole parameter space. So we determine most of hyper-parameters by following the same setting in prior work of Xie et al. [7]. Here we first study the sensitivity of hyper-parameters in data augmentation which are first introduced in this paper. In our experiments, only random rotation and shifting are used. We sample $0^\circ, 10^\circ, \ldots, 60^\circ$ for rotation, and 0, 1, ..., 6 pixels for shifting transformation. Then on each of the resulting 49 grids, we run our ASPC-DA algorithm on MNIST-full dataset twice and report the average result. In Fig. 6, we report relative ACC which is relative to the performance of ASPC, i.e., ASPC-DA with rotation 0 degree and shifting 0 pixel. As can be seen, our ASPC-DA outperforms the baseline ASPC in a wide range. When shifting for up to 6 pixels and rotation for up to 60 degree, the performance of our ASPC-DA drops dramatically. This is not a surprise because there will be a lot of information lost in this case. In the range of $[0^\circ, 40^\circ]$ for rotation and $[0, 4]$ pixels for shifting, our ASPC-DA performs stably well. In a word, our ASPC-DA algorithm is insensitive to the hyper-parameters introduced by data augmentation in a wide range. For simplicity, we set 10 degree for rotation and $\sqrt{D}/10$ pixels for shifting and keep this setting fixed over all experiments. Actually, we can further improve the current ACC (NMI) of our ASPC-DA in Table 3 from 0.988 (0.966) to 0.990 (0.971) by using rotation 10 degree and shifting 2 pixels.

Then we analyze how the network structure affects the clustering performance. As mentioned before, the encoder model is composed of four fully connected layers with number of neurons 500, 500, 2000, 10, respectively. We set the fourth layer to 10 neurons because the input datasets all have 10 clusters. So we keep it fixed. Due to the computing resource and time limit, for each trial, we only change one layer, and for each layer, we change the number of neurons to one of $\{125, 250, 500, 1000, 2000, 4000\}$. The accuracies on MNIST-full dataset are depicted in Fig. 7. No matter how

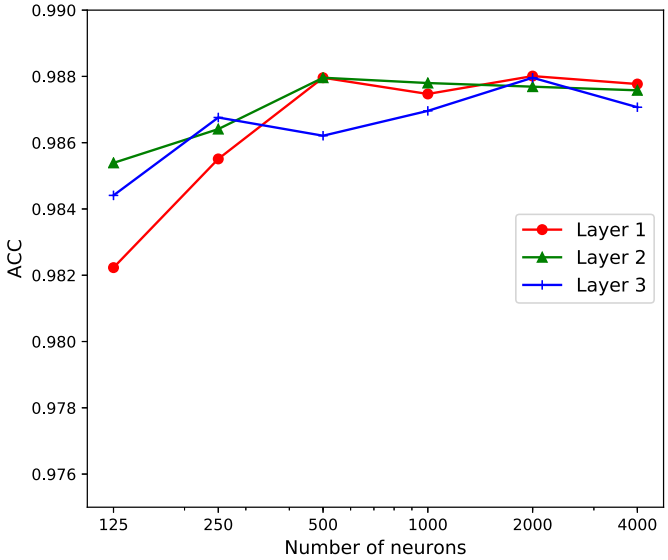

Fig. 7. Sensitivity of the network structure on MNIST-full dataset. In a wide range, the clustering accuracy is insensitive to the number of neurons of the network.

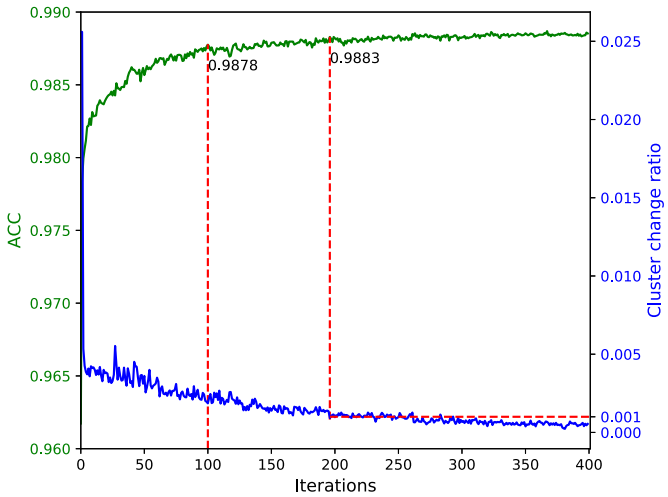

Fig. 8. The trends of clustering accuracy and cluster change ratio regarding the finetuning iterations. After the 100th iteration, the accuracy stays stable.

the number of neurons changes, the accuracy stably stays in $[0.982, 0.988]$, which is still larger than that existing deep clustering algorithms can achieve (see Table 3). In a word, our ASPC-DA is insensitive to the network structure.

Finally, we discuss the stopping criterion (18) in more detail. We shall analogize the stopping criterion in this paper to the early stopping mechanism in supervised deep neural networks. Therefore, it is not necessary to meet (18) before ending the training. Instead, the maximum iterations T serves as the main parameter to control when to stop training. Fig. 8 shows the trends of clustering accuracy and cluster change ratio (i.e., the left side of (18)) regarding the finetuning iterations on MNIST-full dataset. It is easy to notice that clustering accuracy and cluster change ratio have reached a stable level after the 100th iteration. And the cluster change ratio nearly stops decreasing after 0.001. Therefore, setting the stopping threshold $\delta = 0.001$ as the convergence condition is a reasonable choice. On the other hand, if we wait for (18) being met, we will end with

TABLE 6
Clustering Performances of Our Variants in Terms of ACC and NMI

	MNIST-full		MNIST-test		USPS		Fashion	
	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
k-means	0.532 ± 0.002	0.500 ± 0.000	0.546 ± 0.001	0.501 ± 0.000	0.668 ± 0.000	0.627 ± 0.001	0.474 ± 0.000	0.512 ± 0.000
SC	0.656 ± 0.000	0.731 ± 0.000	0.660 ± 0.000	0.704 ± 0.000	0.649 ± 0.000	0.794 ± 0.000	0.508 ± 0.000	0.575 ± 0.000
AC	0.621 ± 0.000	0.682 ± 0.000	0.695 ± 0.000	0.711 ± 0.000	0.683 ± 0.000	0.725 ± 0.000	0.500 ± 0.000	0.564 ± 0.000
GMM	0.433 ± 0.009	0.366 ± 0.011	0.540 ± 0.017	0.493 ± 0.016	0.551 ± 0.003	0.530 ± 0.008	0.556 ± 0.012	0.557 ± 0.010
RCC [44]	–	0.893 ± 0.000	–	0.828 ± 0.000	–	0.742 ± 0.000	–	0.614 ± 0.000
ASPC-DA (baseline)	$\mathbf{0.988 \pm 0.001}$	$\mathbf{0.966 \pm 0.002}$	0.973 ± 0.003	0.936 ± 0.003	0.982 ± 0.001	0.951 ± 0.002	0.591 ± 0.022	0.654 ± 0.021
ASPC-DA-SC	$\mathbf{0.988 \pm 0.001}$	$\mathbf{0.966 \pm 0.001}$	0.879 ± 0.003	0.908 ± 0.003	$\mathbf{0.983 \pm 0.000}$	$\mathbf{0.952 \pm 0.000}$	$\mathbf{0.637 \pm 0.013}$	$\mathbf{0.706 \pm 0.009}$
ASPC-DA-AC	$\mathbf{0.988 \pm 0.001}$	$\mathbf{0.966 \pm 0.002}$	0.973 ± 0.001	0.937 ± 0.002	$\mathbf{0.983 \pm 0.001}$	0.951 ± 0.002	0.564 ± 0.024	0.649 ± 0.016
ASPC-DA-GMM	$\mathbf{0.988 \pm 0.001}$	0.965 ± 0.001	$\mathbf{0.976 \pm 0.002}$	$\mathbf{0.940} \pm 0.003$	0.982 ± 0.000	0.949 ± 0.000	$\mathbf{0.637 \pm 0.013}$	0.675 ± 0.014
ASPC-DA-RCC	–	0.941 ± 0.019	–	0.896 ± 0.014	–	0.951 ± 0.000	–	0.646 ± 0.003

RCC [44] and ASPC-DA-RCC can estimate the number of clusters automatically, while others require the number of clusters pre-specified.

accuracy 0.9883 and computing time 392 seconds. However, at the 100th iteration, the accuracy has reached 0.9878 and computing time is 200 seconds. It means that we can only increase 0.0005 of accuracy at the cost of 192 seconds. To choose a better trade off between accuracy and computing time, we set maximum iterations $T = 100$.

5 Extensions

As shown in the line 2 of Algorithm 1, we initialize cluster centers $\mathbf{M}$ and assignment labels $\mathbf{s}$ by performing k-means on the features extracted from the pretrained autoencoder. This makes the ASPC-DA algorithm inherit the drawbacks of k-means: 1) performing bad on data with non-flat geometry, and (2) requiring the number of clusters predetermined. However, we realize that the k-means is not the only choice for the initialization. We can naturally replace k-means with spectral clustering (SC), Agglomerative Clustering (AC), Gaussian Mixture Models (GMM), Robust Continuous Clustering (RCC) [44], or any other clustering algorithms. These algorithms will output the cluster assignments $\mathbf{s} = [s_1, s_2, \ldots, s_n]^\top$. If the cluster centers are not defined and output, the center of the jth cluster, $\mathbf{m}_j$, can be defined as the average of data points z_i with $s_i = j$ for $i = 1, 2, \ldots, n$. Finally, we obtain the cluster center matrix $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K] \in \mathbb{R}^{d \times K}$, where d and K are the dimension of the feature space and the number of clusters, respectively. We refer to as ASPC-DA-X the ASPC-DA initialized by X algorithm, where X can be one of SC, AC, GMM, and RCC.

By using the same experimental settings with ASPC-DA, we show the performances of these extended variants in Table 6. First, ASPC-DA-X consistently outperforms X algorithm by a large margin which validates that the proposed ASPC-DA framework is responsible for the good performance. Second, all the variants are comparable with the baseline ASPC-DA algorithm. This indicates that changing the initialization clustering algorithm is feasible and effective. Third, ASPC-DA-SC sets the new state-of-the-art performance on Fashion dataset and leads its components by a notable advantage. Finally, we emphasize that the good performance of ASPC-DA-RCC algorithm is obtained without the ground truth number of clusters pre-specified. And the estimated numbers of clusters of MNIST-full, MNIST-test, USPS, and Fashion datasets are 19.6 ± 3.9, 22.3 ± 2.1, 24.4 ± 2.8, 19.0 ± 0.0, respectively, which are in a reasonable range. In Appendix C, available in the online supplemental material, we also demonstrate by toy datasets that ASPC-DA-SC can handle data with non-flat geometry. More exploration on these variants will be our future work.

6 Conclusion

We proposed an Adaptive Self-Paced deep Clustering with Data Augmentation (ASPC-DA) algorithm to learn robust cluster-oriented features. Our ASPC-DA excludes the examples near cluster boundaries from training by gradually adding "easy" (close to cluster centers) examples. We formulated the process of selecting examples and proposed an adaptive self-paced learning algorithm which does not introduce extra hyper-parameters. Data augmentation was naturally incorporated as we related the unsupervised feature learning process to supervised learning. Extensive experiments showed that the proposed ASPC-DA outperforms the state-of-the-art clustering methods. The ablation study and sensitivity analysis further validated the effectiveness of our algorithm. In the future, we plan on studying the augmentation techniques for non-image data. Directly adding Gaussian noise seems to be a potential choice, but the effectiveness has not been observed. It is also worthy of adopting techniques in this paper to other existing deep clustering algorithms. At last, we would like to apply our algorithm to real-world applications.

Acknowledgments

This work was supported by the National Key R & D Program of China 2018YFB1003203 and the National Natural Science Foundation of China (project no. 61773392 and 61672528).

References

[1] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. Berkeley Symp. Math. Statist. Probability*, 1967, pp. 281–297.
[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.
[3] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.

[4] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proc. Inf. Conf. Neural Inf. Process. Syst.*, 2002, pp. 849–856.

[5] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: Spectral clustering and normalized cuts," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 551–556.

[6] X. Liu, Y. Dou, J. Yin, L. Wang, and E. Zhu, "Multiple kernel k-means clustering with matrix-induced regularization," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1888–1894.

[7] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 478–487.

[8] X. Peng, S. Xiao, J. Feng, W.-Y. Yau, and Z. Yi, "Deep subspace clustering with sparsity prior," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 1925–1931.

[9] X. Peng, J. Feng, J. Lu, W.-Y. Yau, and Z. Yi, "Cascade subspace clustering," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 2478–2484.

[10] K. G. Dizaji, A. Herandi, C. Deng, W. Cai, and H. Huang, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," *Int. Conf. Comput. Vision, {ICCV}*, 2017, pp. 5747–5756.

[11] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 1753–1759.

[12] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards K-means-friendly spaces: Simultaneous deep learning and clustering," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3861–3870.

[13] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. Int. Conf. Learn. Representations*, 2014.

[14] I. J. Goodfellow, J. Pouget-Abadie , M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[15] Q. Yang, H. Wang, T. Li, and Y. Yang, "Deep belief networks oriented clustering," in *Proc. Int. Conf. Intell. Syst. Knowl. Eng.*, 2015, pp. 58–65.

[16] C. Song, Y. Huang, F. Liu, Z. Wang, and L. Wang, "Deep auto-encoder based clustering," *Intell. Data Anal.*, vol. 18, no. 6S, pp. S65–S76, 2014.

[17] M. M. Fard, T. Thonet, and E. Gaussier, "Deep k-means: Jointly clustering with k-means and learning representations," *CoRR*, vol. abs/1806.10069, 2018. http://arxiv.org/abs/1806.10069

[18] M. Jabi, M. Pedersoli, A. Mitiche, and I. B. Ayed, "Deep clustering: On the link between discriminative models and k-means," *CoRR*, vol. abs/1810.04246, 2018. http://arxiv.org/abs/1810.04246

[19] P. Ji, T. Zhang, H. Li, M. Salzmann, and I. D. Reid, "Deep subspace clustering networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 23–32.

[20] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: An unsupervised and generative approach to clustering," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 1965–1972.

[21] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep unsupervised clustering with Gaussian mixture variational autoencoders," *CoRR*, vol. abs/1611.02648, 2016. http://arxiv.org/abs/1611.02648

[22] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2172–2180.

[23] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "ClusterGAN : Latent space clustering in generative adversarial networks," in *Proc. AAAI Conf. Artif. Intell.*, 2019.

[24] P. Zhou, Y. Hou, and J. Feng, "Deep adversarial subspace clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1596–1604.

[25] Y. Yu and W. Zhou, "Mixture of GANs for clustering," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 3047–3053.

[26] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5147–5156.

[27] F. Li, H. Qiao, and B. Zhang, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognit.*, vol. 83, pp. 161–173, 2018. https://doi.org/10.1016/j.patcog.2018.05.019

[28] J. Guérin and B. Boots, "Improving image clustering with multiple pretrained cnn feature extractors," *Brit. Mach. Vision Conf. 2018, {BMVC}*, Northumbria University, Newcastle, UK, Sept.2018.

[29] W. Lin, J. Chen, C. D. Castillo, and R. Chellappa, "Deep density clustering of unconstrained faces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8128–8137.

[30] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 1692–1700.

[31] M. Caron, "Unsupervised representation learning with clustering in deep convolutional networks," 2018.

[32] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 1189–1197.

[33] L. Jiang, D. Meng, T. Mitamura, and A. G. Hauptmann, "Easy samples first: Self-paced reranking for zero-example multimedia search," in *Proc. ACM Int. Conf. Multimedia*, 2014, pp. 547–556.

[34] C. Xu, D. Tao, and C. Xu, "Multi-view self-paced learning for clustering," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 3974–3980.

[35] L. Jiang, D. Meng, S.-I. Yu, Z.-Z. Lan, S. Shan, and A. G. Hauptmann, "Self-paced learning with diversity," in *Proc. Advances Neural Inform. Process. Syst. (NIPS)*, 2014.

[36] Y. Fan, R. He, J. Liang, and B.-G. Hu, "Self-paced learning: An implicit regularization perspective," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1877–1883.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[39] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3859–3869.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[41] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017. http://arxiv.org/abs/1708.07747

[42] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. & Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[43] D. Cai, X. He, X. Wang, H. Bao, and J. Han, "Locality preserving nonnegative matrix factorization," in *Proc. Int. Joint Conf. Artif. Intell.*, 2009, pp. 1010–1015.

[44] S. Shah and V. Koltun, "Robust continuous clustering," *Proc. Nat. Academy Sci. United States America*, vol. 114, no. 37, pp. 9814–9819, 2017.

[45] J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan, "Deep adaptive image clustering," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5880–5888.

[46] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Res. Logistics Quart.*, vol. 2, no. 1/2, pp. 83–97, 1955.

[47] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *J. Mach. Learn. Res.*, vol. 15, pp. 315–323, 2011.

[48] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. Dec., pp. 3371–3408, 2010.

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[50] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[51] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Representations, {ICLR}*, 2015. http://arxiv.org/abs/1412.6980

[52] F. Chollet, et al., "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras

[53] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. Nov., pp. 2579–2605, 2008.

Xifeng Guo received the MS degree in computer science from the National University of Defense Technology, China, in 2016. He is currently working toward the PhD degree in the College of Computer, National University of Defense Technology. His research interests include deep learning, unsupervised learning, transfer learning, and computer vision. He has published papers in highly regarded journals and conferences such as *Pattern Recognition*, IJCAI, etc.

Xinwang Liu received the PhD degree from the National University of Defense Technology (NUDT), China. He is now an assistant researcher of the College of Computer, NUDT. His current research interests include kernel learning and unsupervised feature learning. He has published more than 40 peer-reviewed papers, including those in highly regarded journals and conferences such as the *IEEE Transactions on Image Processing*, the *IEEE Transactions on Neural Networks and Learning Systems*, ICCV, AAAI, IJCAI, etc. He served on the technical program committees of IJCAI 2016, and 2017 and AAAI2017. He is a member of the IEEE.

En Zhu received the PhD degree from the National University of Defense Technology (NUDT), China. He is now a professor with the School of Computer Science, NUDT, China. His main research interests include pattern recognition, image processing, machine vision, and machine learning. He has published more than 60 peer-reviewed papers, including the *IEEE Transactions on Circuits and Systems for Video Technology*, the *IEEE Transactions on Neural Networks and Learning Systems*, the *Pattern Recognition*, AAAI, IJCAI, etc. He was awarded the China National Excellence Doctoral Dissertation.

Xinzhong Zhu received the MS degree in software engineering from the National University of Defense Technology, in 2005, and the PhD degree in signal and information processing, in 2018. He is a professor with the College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, China. His research interests include machine learning, computer vision, manufacturing informatization, robotics and system integration, and intelligent manufacturing. Specifically, he is currently focusing on kernel learning and feature selection, multi-view clustering algorithms, real-time object detection (e.g., pedestrian detection, vehicle detection, general object detection, etc.) and deep learning, and their applications.

Miaomiao Li received the PhD degree from the National University of Defense Technology, China. She is now a lecturer at Changsha College, Changsha, China. Her current research interests include kernel learning and multi-view clustering. She has published several peer-reviewed papers such as AAAI, IJCAI, *Neurocomputing*, *Knowledge-Based Systems*, etc. She serves on the technical program committees of IJCAI 2017/2018.

Xin Xu (M'07-SM'12) received the PhD degree in control science and engineering from the National University of Defense Technology (NUDT), China. He has been a visiting professor with Hong Kong Polytechnic University, the University of Alberta, the University of Guelph, and the University of Strathclyde, United Kingdom, respectively. He is currently a professor with the College of Mechatronics and Automation, NUDT, China. He has co-authored more than 160 papers in international journals and conferences, and co-authored four books. His research interests include intelligent control, reinforcement learning, approximate dynamic programming, machine learning, robotics, and autonomous vehicles. He received the Fork Ying Tong Youth Teacher Fund of China, in 2008 and the 2nd class National Natural Science Award of China, in 2012. He serves as the co-editor-in-chief of the *Journal of Intelligent Learning Systems and Applications*. He is an associate editor of *Information Sciences*, *Intelligent Automation and Soft Computing*, and *Acta Automatica Sinica*. He was a guest editor of the *International Journal of Adaptive Control and Signal Processing* and the *International Journal of Social Robotics*. He is a member of the IEEE CIS Technical Committee on Approximate Dynamic Programming and Reinforcement Learning and the IEEE RAS Technical Committee on Robot Learning. He is a senior member of the IEEE.

Jianping Yin received the PhD degree from the National University of Defense Technology (NUDT), China. He is now a distinguished professor with the Dongguan University of Technology. His research interests include pattern recognition and machine learning. He has published more than 100 peer-reviewed papers, including the *IEEE Transactions on Circuits and Systems for Video Technology*, the *IEEE Transactions on Neural Networks and Learning Systems*, *Pattern Recognition*, AAAI, IJCAI, etc. He was awarded as the China National Excellence Doctoral Dissertation' Supervisor and National Excellence Teacher. He served on the technical program committees of more than 30 international conferences and workshops.