Joint Deep Clustering: Classification and Review

Arwa Alturki, Ouiem Bchir, Mohamed Maher Ben Ismail
Department of Computer Science
King Saud University, Riyadh
Saudi Arabia

***Abstract*—Clustering is a fundamental problem in machine learning. To address this, a large number of algorithms have been developed. Some of these algorithms, such as K-means, handle the original data directly, while others, such as spectral clustering, apply linear transformation to the data. Still others, such as kernel-based algorithms, use nonlinear transformation. Since the performance of the clustering depends strongly on the quality of the data representation, representation learning approaches have been extensively researched. With the recent advances in deep learning, deep neural networks are being increasingly utilized to learn clustering-friendly representation. We provide here a review of existing algorithms that are being used to jointly optimize deep neural networks and clustering methods.**

Keywords—Clustering; deep learning; deep neural network; representation learning; clustering loss; reconstruction loss

I. INTRODUCTION

Clustering is a challenging problem in machine learning, as its purpose is to categorize objects into groups according to similarity measures. To achieve this, many clustering algorithms have been published in the literature [1]. These algorithms can be classified into two groups: hierarchical and partitional approaches. In hierarchical clustering, the data are organized into nested clusters that are merged into larger ones or divided into smaller ones. This yields a hierarchy of clusters called a dendrogram. Conversely, partitional clustering is based on the optimization of a specific cost function that allows separation between clusters. The performance of these different clustering algorithms depends on their accurate representation of the data. Hence, data representation learning is a critical step in the clustering process.

Over the past several decades, many traditional representation learning techniques have been proposed. Some of these techniques are designed to learn low-dimensional data representation with linear projections, such as unsupervised principal component analysis (PCA) [2], supervised linear discriminant analysis (LDA) [3], kernel-based PCA [4], and generalized discriminant analysis (GDA) [5]. To discover the intrinsic structure of high-dimensional data, manifold learning algorithms that are based on locality were introduced, such as isometric feature mapping (Isomap) [6] and locally linear embedding (LLE) [7]. In 2006, Hinton et al. [8, 9] introduced the concept of deep learning by utilizing artificial neural networks (ANNs) for dimensionality reduction. Specifically, they introduced a greedy layer-wise pretraining process and a finetuning framework for deep neural network (DNN) learning. The resulting performance was better than that of state-of-the-art algorithms on MNIST [9] handwritten digit recognition and document retrieval tasks. Following this groundbreaking work, a considerable number of deep representation learning algorithms were developed.

Recently, frameworks that perform deep representation learning and clustering procedures have attracted much attention. These frameworks are referred to as deep clustering algorithms, and they can be divided into (1) separated deep clustering and (2) combined deep clustering methods. In separated deep clustering, the deep representation is learned first, and then fed into a clustering algorithm. However, because these two tasks are optimized separately, the learned representation may not be suitable or sufficient for the clustering. In combined deep clustering, the deep representation learning and clustering are jointly optimized. This implies that the clustering assignments and network parameters are reciprocally affected in every learning iteration. Such an approach yields a representation that is more suitable for clustering. Two approaches to achieve combined optimization exist: the pretraining and finetuning approach, and the joint training approach. In the pretraining and finetuning approach, the DNN is pre-trained with nonsclustering loss (network loss) to initialize the network parameters and learn initial representation. Then, the clustering loss is used to train (finetune) the initialized network and output clusters. In contrast, in the joint training approach, the network is trained with a joint loss function that integrates the clustering loss with a nonclustering loss (network loss). In this review, we survey joint deep clustering algorithms by examining different network structures and analyzing the building blocks of these algorithms.

In Section 2, we introduce deep representation learning techniques. In Section 3, we will describe the clustering algorithms that are utilized in joint deep clustering. In Section 4, we provide a survey of the joint deep clustering approaches, and in Section 5, we present the conclusions from the results of this survey.

II. DEEP REPRESENTATION LEARNING

Deep representation learning techniques generate multiple levels or a hierarchy of representations. In this hierarchy, the high-level representations are constructed from multiple low-level ones. These techniques are based on deep ANNs. A typical (single-layer) neural network consists of input, hidden, and output layers. The input layer receives the raw input data, whereas the output layer produces the task results, such as object classification or clustering. The hidden layer applies nonlinear transformation to extract more abstract and composite representations from the input data. DNNs contain more than one hidden layer, to apply multiple nonlinear transformations and create the representation hierarchy. The word "deep" refers to the multiple hidden layers in the neural network.

DNNs apply a supervised learning process, where a set of input–output pairs is provided for training. This learning process is composed of two passes: a forward pass (forward propagation) and a backward pass (backpropagation). The forward pass first randomly initializes the network parameters, that is, the connections, weights, and biases. Then, the input data are passed through the network layers, in the forward direction, to calculate the predicted output. Next, the predicted output is compared with the actual output through a task-specific loss function. An optimization technique, namely, stochastic gradient descent (SGD), is then applied to minimize the loss function. Conversely, the backpropagation process is initiated by updating the network weights so that the predicted output is closer to the actual output. This can be achieved by minimizing the error of each output neuron in the entire network.

In the following subsections, we discuss three DNN types that have been used as representation learning techniques for clustering tasks. The first is feedforward neural networks (FNNs), which fall into two categories: completely connected networks (FCNs) [10] and convolutional neural networks (CNNs) [11]; the second is deep belief network (DBNs), which are composed of a stochastic probabilistic component called a restricted Boltzmann machine (RBM); and the third is the autoencoder (AE), which comes in two types: the stacked AE (SAE) and convolutional AE (CAE).

A. Feedforward Neural Networks

The FNN [12] is the simplest type of neural network, where the connection between neurons does not form a cycle. The information in this type of network moves forward (in one direction) from the input neurons to the output neurons. In this case, there is no feedback from the output toward the input neurons. FNNs are arranged in the form of layers, as are all neural networks. Depending on the number of layers, an FNN can be a single- or a multilayer network. As mentioned above, FNNs fall into two types: FCNs and CNNs.

An FCN, also known as a multilayer perceptron (MLP) [13], consists of multiple completely connected (FC) layers, where each neuron in one layer is connected to every neuron in the previous layer. In addition, every one of these connections has its own weight. FCNs are composed of an input layer, an output layer, and an arbitrary number of hidden layers. This type of feedforward network is tailored for supervised learning.

Inspired by biological process, the neuron connectivity pattern in CNNs mimics the organization of the animal visual cortex. The first and core building block of a CNN is the convolutional layer, where each neuron is connected to only a few neurons in the previous layer. The same set of weights is used for every neuron. The second layer is the rectified linear unit (ReLU) layer, which applies an elementwise nonlinear activation function to retain the positive parts of the inputs and remove the negative ones by replacing them with zero. The reason for applying ReLU layers in a CNN is to increase the nonlinearity of the inputs. A pooling layer is frequently inserted between two consecutive convolutional layers. The pooling layer applies a function to reduce the spatial size of the representation by combining the output of the set of neurons in one layer into a single neuron in the next layer. As a consequence, the number of parameters and computations throughout the network is reduced and overfitting is controlled. The final layer of a CNN is an FC layer to classify the input. Similar to FCNs, CNNs are designed for supervised learning, and specifically to classify image datasets.

Deep clustering algorithms that employ feedforward networks for unsupervised representation learning use clustering loss only to train the network. Hence, these algorithms aim to optimize the objective function,

$$\mathrm{L} = \mathrm{L_c} \tag{1}$$

where L is the algorithm loss function and L_c is the clustering loss function. In the absence of other measures and depending completely on the clustering loss, such deep clustering algorithms may lead to a distorted representation space, wherein all data points are assigned to tight clusters. Such a trivial solution results in a small amount of meaningless clustering loss. To alleviate this problem, and in addition to the careful design of the clustering loss function, suitable network parameter initialization is required to enhance the performance.

B. Deep Belief Network

DBNs [14] are a branch of DNNs, and are composed of a stack of RBMs [15] followed by a softmax layer that applies a softmax activation function to the input. An RBM is a two-layer neural network, where the first is the visible (input) layer and the second is the hidden layer. A DBN is trained by greedy layer-wise unsupervised learning with RBMs as the building blocks for each layer. Then, the parameters of the DBN are finetuned according to a task-specific loss function. DBN-based deep clustering algorithms finetune the network parameters using the clustering loss function only, and thus optimize an objective function similar to the feedforward network loss function in equation (1). Hence, careful clustering loss selection and good network parameter initialization affect the performance of the deep clustering algorithm.

C. Autoencoder

An AE [16] is a special type of neural network designed for unsupervised representation learning. It consists of three building blocks: an encoder, a bottleneck layer, and a decoder. The encoder maps the input x_i to its hidden representation z_i through a nonlinear function $f_{W_1}(\cdot)$, as in equation (2), and the decoder reconstructs the input x_i from its hidden representation z_i by using a transformation function $g_{W_2}(\cdot)$ as in equation (3).

$$\mathrm{z_i} = \mathrm{f}_{\mathrm{W_1}}(\mathrm{x_i}) \tag{2}$$

$$\mathrm{y_i} = \mathrm{g}_{\mathrm{W_2}}(\mathrm{z_i}) \tag{3}$$

Here, W_1 represents the encoding weight, and W_2 the decoding weight. The encoder and decoder can comprise an FC network to construct an SAE [17], or a CNN to form a CAE [18]. The bottleneck layer controls the amount of information that traverses the network by learning a compressed representation of the input data. The learning problem can be formulated as a supervised one that is aimed to output the reconstruction image y_i from the input x_i. The entire network can be trained by minimizing the reconstruction loss L_{r_AE}, which measures the differences between the original input x_i and the reconstructed image y_i:

$$\mathrm{L_r} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\|\mathrm{x_i} - \mathrm{y_i}\|^2 \tag{4}$$

AE-based deep clustering algorithms seek to optimize an objective function that combines clustering and reconstruction losses:

$$\mathrm{L} = \mathrm{L_r} + \gamma \mathrm{L_c} \tag{5}$$

where $\boldsymbol{\gamma}$ is a coefficient to control the distortion of the representation embedding space. The existence of the reconstruction loss forces the algorithm to avoid trivial solutions and learn more feasible representations.

D. Variational Autoencoder

VAE [19] is a generative variant of AE that enforces the latent code to follow a predefined distribution. This goal is achieved by encoding the input data into two vectors instead of one: mean value and standard deviation. Unlike the output of the standard AEs that points directly to the encoded value in the latent space, VAE outputs point to the area where the encoded value can be. To be more specific, VAE initializes a probability distribution where the mean value controls the location point of the encoding center, and the standard deviation defines the area in which encoding can vary from the mean. As a consequence, VAE allows interpolation and generation of new samples. Mathematically, VAE measures the Kullback–Leibler (KL) divergence [20] from a prior distribution to approximate the variational posterior distribution. The objective function can be formulated as the following:

$$\mathrm{L_{r\_VAE}} = \mathrm{E}_{\mathrm{q(z|x_i)}}[\log \mathrm{p(x_i|z)}] \tag{6}$$

$$\mathrm{L_{VAE}} = \mathrm{L_{r\_VAE}} - \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{KL(q(z|x_i)\|p(z))} \tag{7}$$

where L_{r_VAE} represents the reconstruction loss of the VAE, $p(z)$ is the prior over the latent variables, $q(z|x_i)$ is the variational posterior to approximate the true posterior $p(z|x_i)$, and $p(x_i|z)$ is the likelihood function. Gaussian distribution is the common choice as prior; however, VAE-based clustering algorithms should choose a distribution which can describe the structure of the clusters.

E. Adversarial Autoencoder

Similar to VAE, AAE [21] utilizes a prior distribution to control the encoding of the input data. Hence, the decoder learns only the mapping from the prior distribution to the data distribution. The output of the AAE encoder, i.e. the encoded value, is fed as input to the decoder and to a special generative adversarial network (GAN) [19]. In AAE, the encoder and decoder together form the generator model (G), while the GAN is known as discriminator (D). Through the learning process, AAE establishes a min–max adversarial game between its generator and the discriminator. While the generator tries to map a generated sample from a prior distribution to the data space, the discriminator computes the probability to detect whether its input a real sample from the data distribution or a fake sample from the generator. The training process of AAE is handled through two phases: (1) a reconstruction phase and (2) a regulation phase. During the reconstruction phase, the generator is trained to minimize the reconstruction loss of the generated sample and produce a reconstructed image of it. In the regulation phase, the discriminator parameters are updated to distinguish the real samples generated by the priori from the fake samples generated by the encoder. The discriminator network D is updated by the following discriminative loss (L_d):

$$\mathrm{L_d} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\left[\log \mathrm{D}(\widehat{\mathrm{z_i}}) + \log\left(1 - \mathrm{D(z_i)}\right)\right] \tag{8}$$

where $\widehat{z_i}$ and z_i are the sample from prior distribution and input sample, respectively. Then, the discriminator is fixed, and the encoder is updated to confuse the discriminator by increasing the classification error of D on the input latent representation with generation loss L_g, as in the following equation:

$$\mathrm{L_g} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}} \log\left(1 - \mathrm{D(z_i)}\right) \tag{9}$$

AAE-based deep clustering algorithms optimize a loss function that combines reconstruction loss, generation loss, and clustering loss:

$$\mathrm{L_{AAE}} = \mathrm{L_r} + \alpha \mathrm{L_g} + \beta \mathrm{L_c} \tag{10}$$

where L_r, L_g, and L_c represent the reconstruction loss defined in equation (4), the generation loss in equation (9), and a clustering loss, respectively. α and β are hyperparameters to balance the importance of the generation loss and the clustering loss, respectively.

III. Clustering Techniques

As stated previously, clustering techniques can be divided into two types: hierarchical and partitional clustering. Hierarchical clustering methods iteratively merge smaller clusters into larger ones, or split large clusters into smaller ones. The difference between hierarchical algorithms includes the similarity measures that are used to determine which clusters should be merged or split. The results of hierarchical clustering are organized in a tree called a dendrogram, which shows the relationships between clusters. Conversely, partitional clustering seeks to decompose data into a set of disjointed groups. This decomposition is achieved based on the minimization of a specific objective loss function. Centroid-based algorithms, such as K-means [22, 23] and KL-divergence [20] clustering, distribution-based algorithms such as Gaussian mixture clustering [24], graph-based clustering algorithms such as spectral clustering [25] and RCC [26], and density-based algorithms such as DBSCAN [27] are all subtypes of partitional clustering algorithms. As existing joint deep clustering utilizes only centroid- and graph-based clustering, these two techniques are explained in the following subsections. Finally, we introduce some auxiliary clustering losses that are used in conjunction with other losses to guide deep representation learning.

A. Centroid-Based Clustering

Given a dataset $X = \{x_1, \dots, x_n\}$ of n points together with its extracted representation $Z = \{z, \dots, z_n\}$, centroid-based clustering partitions the data points into clusters with central representatives called centroids. These cluster centroids, denoted by $\mathcal{M} = \{\mu_1, \dots, \mu_k\}$, where k is a predefined number of clusters, do not necessarily belong to the dataset. In joint deep clustering algorithms, two centroid-based algorithms are utilized: K-means and KL-divergence clustering.

1) K-means Clustering: K-means clustering first randomly selects k centroids from the input data representations, each of which represents a cluster. A K-means algorithm minimizes the total mean squared error between the input data and cluster centroids according to the loss function:

$$L_{KM} = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| z_i - \mu_j \right\|_2^2 \tag{11}$$

An additional variation of the K-means loss function is the weighted least squares error, referred to as weighted K-means. It optimizes the cost function as:

$$L_{WKM} = \sum_{j=1}^{k} \sum_{i=1}^{n} S_{ij} \left\| z_i - \mu_j \right\|_2^2 \tag{12}$$

where S_{ij} is a similarity weight that encodes the closeness of a data point to a cluster centroid; i.e., S_{ij} will be larger if the data point z_i is close to the centroid μ_j. In the K-means learning process, the following two steps are repeated until convergence is reached:

- Point assignment update, which is accomplished by (i) calculating the mean distance from the data point to every cluster centroid, and (ii) assigning points to the cluster with the minimum mean among all clusters.
- Centroid update, which is computed according to the following equation, where m_j is the number of points in the j^{th} cluster:

$$\mu_j = \left(\frac{1}{m_j}\right) \sum_{i=1}^{m_j} z_i \tag{13}$$

K-means perform well when the distribution of the points is in circular form. Otherwise, K-means will attempt to group the points in circular form, which will affect the clustering result. To remedy this issue, K-means should be updated to employ a distribution-based model instead of a distance-based model.

Gaussian Mixture Model (GMM) [24] is a probabilistic soft clustering technique which tends to group points with the same distribution together. The clustering process starts by initializing the means and covariances of the Gaussian distribution for k clusters. Then, the expectations of all points assignments are calculated for all clusters. Furthermore, the distribution parameters are re-estimated, and the log-likelihood function is computed. This process continues until a predefined convergence criterion is reached.

2) KL-divergence Clustering: KL-divergence clustering is a soft assignment clustering technique, in which each data point is assigned to all clusters with varying probabilities. This algorithm is initiated using K-means to obtain k initial centroids. Next, the learning process is executed to optimize the following Kullback–Leibler (KL) divergence loss function:

$$L_{KLD} = KL(P||Q) = \sum_i \sum_j p_{ij} \, log\left(\frac{p_{ij}}{q_{ij}}\right) \tag{14}$$

where P is an auxiliary target distribution and Q represents the data point soft assignments. The KL-divergence clustering algorithm refines the point assignments by learning from higher confidence points utilizing the auxiliary target distribution p_{ij}. Specifically, the algorithm matches the soft assignments q_{ij} with the target distribution p_{ij} by computing the KL divergence. The clustering algorithm iteratively performs the following steps until convergence is obtained or the maximum iteration is reached:

1) Calculation of $q_{ij,}$, the probability that a data point i belongs to cluster j. Two means of calculating q_{ij} exist: (1) student's t-distribution [28], as in equation (15), and (2) a multinominal regression [28] function, as in equation (16).

$$q_{ij} = \frac{\left(1+\left\|z_i-\mu_j\right\|_2^2\right)^{-1}}{\sum_j \left(1+\left\|z_i-\mu_j\right\|_2^2\right)^{-1}} \tag{15}$$

$$= \frac{\exp(\mu_j^T z_i)}{\sum_j \exp(\mu_j^T z_i)} \tag{16}$$

2) Computing p_{ij}, a higher confidence distribution that can be obtained by calculating the soft cluster frequencies by considering the formula:

$$p_{ij} = \frac{{q_{ij}}^2 / \sum_i q_{ij}}{\sum_j ({q_{ij}}^2 / \sum_i q_{ij})} \tag{17}$$

3) Updating clusters centroids according to:

$$\mu_j = \mu_j - \frac{\lambda}{n} \sum_{i=1}^{n} \frac{\partial L_{KLD}}{\partial \mu_j} \tag{18}$$

B. Graph-Based Clustering

Given a dataset $X = \{x_1, \dots, x_n\}$ of n points together with their corresponding representation $Z = \{z, \dots, z_n\}$, graph clustering techniques first construct an undirected similarity graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ denotes a set of vertices to represent the input data, and E is the set of edges between vertices. Several approaches for building a similarity graph [1] exist, two of which are specifically used in joint deep clustering. These approaches are the following:

- K-nearest neighbor (KNN) graph: this graph connects vertex v_i with vertex v_j, if v_j is within K-nearest neighbors of v_i. One problem common to KNN is that the graph is asymmetric, which means that if v_j is among the KNNs of v_i, then v_i is not necessarily among the KNNs of v_j. Hence, the constructed graph is a directed one. To alleviate this problem, there are two solutions; first, to insert an undirected edge between the two vertices v_i and v_j, if one of them is within the KNNs of the other; second, to restricts the edges, two vertices v_i and v_j are connected by an undirected edge only if they are both among the KNNs of each other. The resultant graph in the latter solution is called a mutual KNN graph.
- Completely connected graph: this graph simply connects all vertices with each other by weighted edges.

The weight of an edge w_{ij} between two vertices v_i and v_j represents the similarity between them. Because the graph should express the local neighborhood relationship, a Gaussian similarity function is usually utilized.

The graph is represented by an adjacency matrix, in which the similarity w_{ij} b tween every two vertices is included. Two graph-based clustering algorithms are utilized in joint deep clustering techniques: spectral clustering [25] and robust continuous clustering (RCC) [26]. We briefly explain these two approaches.

1) Spectral clustering: After the construction of the similarity graph and the extraction of the adjacency matrix, the spectral algorithm transforms the data into a low-dimensional space. To achieve this, another graph representation matrix is computed, the Laplacian matrix. The graph Laplacian matrix $\mathcal{L}$ is computed as:

$$\mathcal{L}_{ij} = \begin{cases} d_i, \text{if } i = j \\ w_{ij}, \text{if } (i,j) \in E \\ 0, \text{if } (i,j) \notin E \end{cases} \tag{19}$$

where d_i is the degree of the vertex v_i, which can be computed as:

$$d_i = \sum_{\{j|(i,j)\in E\}} w_{ij} \tag{20}$$

Then, the Laplacian matrix is utilized to find the eigenvalues λ and eigenvectors υ, such that.

$$\lambda\mathcal{L} = \lambda\upsilon \tag{21}$$

Once the eigenvectors have been obtained, the low-dimensional data transformation is completed. Finally, a K-means clustering algorithm, explained in section 3.1, is applied to the transformed data (eigenvectors) to create clusters.

2) Robust Continuous Clustering (RCC): This approach operates on a set of representations $U = \{u_1, \dots, u_n\}$ for the original dataset X , where X and U have the same dimensionality. This algorithm minimizes the loss function.

$$L_{RCC} = L_{data} + \lambda L_{pairwise} \tag{22}$$

where λ is a coefficient that balances the two objective terms. The first term L_{data} is the data loss that constrains the representations to remain near the corresponding data points. The data loss can be computed as:

$$L_{data} = \sum_{i=1}^{n} \|z_i - u_i\|_2^2 \tag{23}$$

The second term, which is the pairwise loss $L_{pairwise}$, is designed to encourage the representations to merge, and pulls them together according to.

$$L_{pairwise} = \sum_{(p,q)\in E} w_{pq}\rho\left(\left\|u_p - u_q\right\|_2; \mu\right) \tag{24}$$

where $\{w_{pq}\}$ represents appropriately defined weights, μ is a scale parameter, and ρ is a redescending M-estimator that can be calculated according to a scaled Geman–McClure function [29]:

$$\rho(x;\ \mu) = \frac{\mu x^2}{\mu + x^2} \tag{25}$$

The first stage in the RCC learning procedure is initialization, which includes the following steps:

1) Construction of the similarity graph $G_1 = (V, E)$ using mutual KNN.

2) Initialization of the data representation with $u_i = z_i$.

3) Initialization of the line process $\mathbb{L} = \{\ell_{pq}\}$, where ℓ_{pq} is an auxiliary variable between two connected vertices v_p and v_q with $\ell_{pq} = 1$.

4) Initialization of a scale parameter μ with $\mu \gg max\left\|z_p - z_q\right\|^2$.

The optimization is aimed to reveal the cluster structure latent in the data; thus, the number of clusters does not need to be known in advance. The following optimization steps are recursively repeated until a maximum iteration number is reached, or the difference between the clustering loss in two consecutive iterations is less than a predetermined threshold.

1) Update ℓ_{pq} according to the following formula.

$$\ell_{pq} = \left(\frac{\mu}{\mu + \left\|u_p - u_q\right\|_2^2}\right)^2 \tag{26}$$

2) Update the representations $U = \{u_1, \dots, u_n\}$ using the following equation:

$$UM = Z \tag{27}$$

where

$$M = I + \lambda\, A \tag{28}$$

I is the identity matrix, e_i is an indicator vector with the i^{th} element set to 1, and A is computed as the following:

$$A = \sum_{(p,q)\in E} w_{pq}\ell_{pq}(e_p - e_q)\,(e_p - e_q)^T \tag{29}$$

Update the value of λ as.

$$\lambda = \frac{\|Z\|_2}{\|A\|_2} \tag{30}$$

Update the value of μ as.

$$\mu = \max\left(\frac{\mu}{2}, \frac{\delta}{2}\right) \tag{31}$$

where δ is a threshold set to be the mean of the lengths of the shortest 1% of the edges in E. Then, RCC constructs a new graph $G_2 = (V, \mathcal{E})$ with $\varepsilon_{pq} = 1$ if $\left\|u_p^* - u_q^*\right\|_2 > \delta$. Finally, the algorithm outputs the clusters given by the connected vertices of G_2.

C. Auxiliary Clustering Losses

Some clustering loss functions are designed to guide deep representation learning techniques to extract feasible clustering-oriented representations; they cannot, however, output clusters. These functions are known as auxiliary clustering losses. Considering a dataset $X = \{x_1, \dots, x_n\}$ of

n points together with its extracted representations $Z = \{z, \dots, z_n\}$, we present the auxiliary clustering losses that have been used in joint deep representation clustering algorithms.

1) Balanced assignment loss: Balanced assignment loss is used in conjunction with other clustering loss to enforce balanced clustering assignments. The difference between two distributions, f and u, is measured based on KL divergence as follows:

$$L_{BA} = KL(f||u) = \frac{1}{n}\sum_i \sum_j p_{ij} \log\left(\frac{f_j}{u_j}\right) \tag{32}$$

where P is the target distribution proposed in equation (17) u is the uniform distribution, and f is the probability distribution, which can be calculated as.

$$f_j = \frac{1}{n}\sum_i p_{ij} \tag{33}$$

2) Locality-preserving loss: Locality-preserving loss preserves the local structure property of the original data by pushing the nearby points together as.

$$L_{LP} = \sum_{i,j \in N_k(i)} S_{ij} \left\| z_i - z_j \right\|_2^2 \tag{34}$$

where $N_k(i)$ is the set of k nearest neighbors of the data point x_i and S_{ij} is a similarity measure between x_i and x_j.

3) Group sparsity loss: Group sparsity loss was inspired by spectral clustering, where a block-diagonal similarity matrix is utilized for representation learning. Specifically, the hidden units are divided into k groups, where k is the number of clusters. For each data point x_i, after its representation z_i has been extracted, a k group unit $\{f^j(x_i)\}_{j=1}^{k}$ is obtained. Then, the group sparsity is computed as.

$$L_{GS} = \sum_{i=1}^{n} \sum_{j=1}^{k} \lambda \sqrt{n_g} \left\| f^j(x_i) \right\|_2 \tag{35}$$

where $f(x_i)$ is the representation encoding function, λ is a constant, and n_g is the group size.

4) Self-expressiveness loss: Self-expressiveness loss is a property where a point in a subspace can be expressed as a linear combination of other points in the same subspace. Let X be a column matrix of all data points; the self-expressiveness can then be represented as $X = XC$, where C is the self-representation coefficient matrix. By minimizing a certain norm of C, and under the assumption that the subspaces are independent, C is guaranteed to have a block-diagonal structure. This ensures that $c_{ij} \neq 0$, where x_i and x_j are two data points lying in the same subspace. The matrix C can then be leveraged by spectral clustering to construct the affinity matrix. Given this fact, each data representation z_i in a latent subspace is approximated by a weighted linear combination of other points $\{z_j\}_{j=1}^{n}$ with weights c_{ij} . To encode self-expressiveness, the following auxiliary clustering loss function is introduced:

$$L_{SE} = \lambda_1 \|C\|_p + \frac{\lambda_2}{2}\|Z - ZC\|^2, \text{s.t.} (\text{diag}(c) = 0) \tag{36}$$

where λ_1 and λ_2 are two regularization parameters to account for data corruption, and $\|\cdot\|_p$ represents an arbitrary matrix norm.

IV. Joint Deep Clustering

Given a dataset $X = \{x_1, \dots, x_n\}$ of n points, the goal of joint deep clustering techniques is simultaneously to learn a low-dimensional representation $Z = \{z_1, \dots, z_n\}$ for the data and to cluster it into groups jointly. This can be accomplished by optimizing a joint loss function that combines two losses: the representation learning loss and the clustering loss. Then, the low-dimensional representations, network parameters (weights and biases), and clustering parameters and assignments are updated jointly. In this section, we survey these algorithms, and provides a taxonomy from the perspective of clustering algorithms. Table I summarizes existing joint deep clustering algorithms.

A. Deep Kullback–Leibler Divergence Clustering

Guo et al. [28] proposed improved deep embedded clustering (IDEC), an algorithm that simultaneously learns low-level representation and cluster assignment. The IDEC algorithm consists of two phases: (1) parameter initialization, and (2) parameter optimization and clustering. In the initialization phase, IDEC initiates a denoising SAE [17], which reconstructs a data point x from a corrupted (noisy) version $\tilde{x}$ to force the encoder and decoder to capture implicitly the structure of data that generate distribution. The SAE is trained based on reconstruction loss to obtain initial values for the network's weights and biases. The clusters' centroids are initiated by applying K-means to the representations extracted from the encoder element. When the initialization is completed, IDEC removes noise from the data to apply clustering to the representation learned from the clean data. When noise has been removed, the denoising SAE degenerates into a traditional SAE, which constrains the dimension of the hidden representation Z to be less than the dimension of the input data X. Then, the optimization and clustering phase is executed by finetuning using KL divergence as clustering loss and SAE reconstruction loss. This results in the joint loss function

$$L_{IDEC} = L_r + \lambda L_{KLD} \tag{37}$$

where L_r is the reconstruction loss in equation (4), L_{KLD} is the KL-divergence clustering loss in equation (14), and λ is a regularization parameter to balance the two terms. Clustering is achieved by alternating between computing the soft assignment based on the student's t-distribution formula in equation (15), and auxiliary target distribution in equation (17). IDEC jointly optimizes the cluster centers μ_j and the network parameters θ using an SGD algorithm [30]. The gradient is calculated for the clustering loss L_c with respect to the cluster centroid μ_j and point representation z_i, and then is utilized in backpropagation. Experimental results have demonstrated the importance of locality preservation. Guo et al. [31] developed a deep clustering method with CAEs (DCEC) for image clustering; the DCEC framework is very similar to the IDEC model, but instead of an SAE, DCEC employs a CAE to better incorporate the relationship between image pixels. The effectiveness of CAE over SAE has also been demonstrated for image datasets.

TABLE I. SUMMARY OF JOINT DEEP CLUSTERING ALGORITHMS

Algorithm	Clustering Technique	Network Architecture	Joint Loss Function	Main Contribution
IDEC	**KL-divergence**	**SAE**	$L_{IDEC} = L_r + \lambda L_{KLD}$	**Joint version of DEC [35], the first well-known deep clustering algorithm.**
DCEC		**CAE**		**Improves on IDEC by using CAE instead of SAE.**
ADEC		**AAE**	$L_{ADEC} = L_r + \alpha L_g + \beta L_{KLD}$	**Preserve the relevance between representation learning and clustering and reach to better trade-off between feature drift and feature randomness issue.**
DEPICT		**CAE**	$L_{DEPICT} = L_{r\_DEPICT} + L_{KLD} + L_{BL}$	**Ensures balanced clustering assignments that provide robust and superior results over image datasets.**
DEN	**K-means**	**SAE**	$L_{DEN} = L_r + \alpha L_{LP} + \beta L_{GS}$	**Learns clustering-oriented representations with the following properties: (1) locality preservation and (2) group sparsity.**
DCN			$L_{DCN} = L_r + \lambda L_{KM}$	**First algorithm to perform K-means and representation learning simultaneously.**
DKM			$L_{DKM} = L_r + \lambda L_{WKM}$	**Updates DCN to use weighted K-means instead of traditional K-means.**
DMC			$L_{DMC} = (1-\alpha)L_r + \alpha L_{LP\_DMC} + \beta L_{WKM}$	**Utilizes deep SAE to improve the traditional multimanifold clustering algorithm.**
DSC-Nets	**Spectral Clustering**	**CAE**	$L_{DSC-Nets} = L_r + L_{SE}$	**Utilizes deep CAE to improve the traditional spectral clustering algorithm.**
DASC		**AAE**	$L_{DASC} = L_r + \alpha L_g + \beta L_{SE}$	**Learns subspace clustering-friendly representations using AAE and self-expressiveness constraint.**
DSC			$L_{DSC} = L_r + L_g + spectral\ clustering$	**More robust to noise; since it enforces the reconstruction constraints for the latent representations and their noisy versions.**
DCC	**RCC**	**SAE**	$L_{DCC} = \frac{1}{D} L_r + \frac{1}{d}(L_{data\_DCC} + \lambda L_{pairwise\_DCC})$	**Utilizes deep SAE to improve on the traditional RCC algorithm.**

Similar to IDEC, Zhou et al. [21] introduced Deep Embedded Clustering With Adversarial Distribution Adaptation (ADEC). Instead of SAE, ADEC utilizes AAE to learn from data space to feature space. With a backpropagation algorithm, ADEC iteratively optimizes the following objective function:

$$L_{ADEC} = L_r + \alpha L_g + \beta L_{KLD} \quad (38)$$

where L_r, L_g, L_{KLD} is the reconstruction loss defined in , the generation loss in (9), and the KL-divergence clustering loss in equation (14), respectively, and α and β are hyperparameters to balance the importance of the generation loss and the clustering loss, respectively. In deep learning, the optimization of a neural network's loss function whose secondary component highly competes with the primary one may lead to feature drift. As a result, the global learning process will be affected, since the features learned by the primary loss can be easily drifted by updating the secondary one. Discarding one of the primary or secondary losses will lead to substitution of a significant portion of true labels for random ones, known as feature randomness. Mrabah et al. [32] enhanced the IDEC approach to reach a better trade-off between feature drift and feature randomness using AAE complemented with data augmentation.

Dizaji et al. [33] proposed the deep embedded regularized clustering (DEPICT) model to learn data representation and perform the clustering task. DEPICT has a complicated network architecture composed of a softmax layer on top of a multilayer CAE. More specifically, DEPICT consists of four components: two encoders, one decoder, and one softmax layer. The encoder and decoder elements of the DEPICT network are referred to as paths.

Thus, there are three paths in the DEPICT architecture. The first path is called the noisy encoder, which is the encoder part of the denoising CAE that accepts noisy input data to infer noisy hidden representations. The second path is called the noisy decoder (or just decoder), and is the decoder element of the denoising CAE for reconstructing the input from the learned noisy representations. The decoder element consists of a strided CNN, which is similar to the traditional one, except that the value of the convolutional kernel stride is greater than 1. The third path is called the clean encoder, a CNN that accepts clean input data to infer clean hidden representations. The clean and the noisy encoder paths share the same network parameters, i.e., weights and biases. The softmax layer (the fourth component of the network) is stacked on top of the noisy encoder top layer and clean encoder top layer to obtain the clustering assignments. The first phase of the algorithm is initialization, where the network parameters, cluster centroids, and target distribution are initialized. Instead of initializing the network parameters randomly, DEPICT assigns the weights from a Gaussian distribution, where the input and output variances are the same for each layer. This initialization approach is known as Xavier (or normalized) initialization [34]. Next, DEPICT is trained with reconstruction loss only (without clustering loss) to obtain initial embedded representations for the input data. Then, the K-means clustering technique is applied to obtain the initial cluster centroids and the initial target distribution P , when the initialization phase is complete, the optimization and clustering phase starts. In the softmax layer, DEPICT iteratively minimizes the following three-term joint loss function:

$$\mathrm{L_{DEPICT}} = \mathrm{L_{r\_DEPICT}} + \mathrm{L_{KLD}} + \mathrm{L_{BL}} \quad (39)$$

where L_{KLD} and L_{BL} are the KL-divergence and balanced assignment losses that were introduced in equations (14) and (32), respectively. The first term is a data-dependent regularization term, which is a reconstruction loss function introduced in DEPICT designed to enhance the representation learning process and avoid the overfitting problem. The reconstruction loss between the noisy decoder and the clean encoder representations is computed as.

$$\mathrm{L_{r\_DEPICT}} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\sum_{\mathrm{l}=1}^{\mathrm{L}-1}\frac{1}{|\mathrm{z}_{\mathrm{i}}^{\mathrm{l}}|}\left\|\mathrm{z}_{\mathrm{i}}^{\mathrm{l}} - \hat{\mathrm{z}}_{\mathrm{i}}^{\mathrm{l}}\right\|_2^2 \quad (40)$$

where n is the size of the input data, L is the number of noisy decoder and clean encoder layers, l is the layer number, $|z_i^l|$ is the l^{th} layer output size, z_i^l is the l^{th} layer of clean representations (from the clean encoder), and $\hat{z}_i^l$ is the l^{th} layer of noisy representations (from the noisy decoder). The second term of the DEPICT joint loss function is the KL-divergence clustering loss. A multinominal logistic regression function is employed to perform the soft clustering assignment. Note that DEPICT computes the soft assignment predictions Q based on noisy representations that are extracted from the noisy encoder, whereas the target distribution P is computed from the clean representations extracted from the clean encoder path. The third term is a regularization term that encourages balanced cluster assignments and avoids the allocation of clusters to outlier samples. The effectiveness of DEPICT has been proven empirically, especially in terms of the running time complexity.

B. Deep K-Means Clustering

Huang et al. [36] introduced a deep embedding network, referred to as DEN, to learn clustering-oriented representations using a three-layer SAE. Similar to that of most deep clustering algorithms, the DEN learning procedure is composed of two phases: initialization (pretraining) and optimization. In the pretraining phase, a three-layer DBN [14] is trained based on the contrastive divergence loss only, to initialize the SAE parameters. Then, the learned representation from the DBN is fed into the three-layer SAE to begin the joint training optimization process. In this phase, the DEN minimizes the joint loss function.

$$\mathrm{L_{DEN}} = \mathrm{L_r} + \alpha\mathrm{L_{LP}} + \beta\mathrm{L_{GS}} \quad (41)$$

where L_r is the reconstruction loss in equation (4), L_{LP} is the locality-preserving auxiliary clustering loss defined in (34), and L_{GS} is the group sparsity auxiliary clustering loss expressed in equation (35) with $S_{ij} = exp\left(\left\|x_i - x_j\right\|_2^2/t\right)$. Further, α, β, and t are tuning parameters. By considering these two auxiliary clustering losses, the DEN imposes two constraints on the learned representations: the first is the locality-preserving constraint to preserve the local structure property of the original data, and the second is the group sparsity constraint. These are imposed to facilitate the clustering process, and ensure that the learned representation incorporates cluster information, and thus, is more suitable for clustering. After the optimization phase, the traditional K-means clustering algorithm is employed to perform clustering.

Yang et al. [37] proposed a dimensionality reduction and K-means clustering framework named the deep clustering network (DCN). A DNN, specifically an SAE, is utilized by the DCN for dimensionality reduction and representation learning. The DCN algorithm is initiated by a pretraining stage based on reconstruction loss to initialize the SAE weights and biases. To initialize the cluster centroids, K-means clustering is applied to the obtained representations from the pretraining. Then, the joint training phase is executed by iteratively optimizing the joint loss function.

$$\mathrm{L_{DCN}} = \mathrm{L_r} + \lambda\mathrm{L_{KM}} \quad (42)$$

where L_r is the reconstruction loss as defined in equation (4), L_{KM} is the K-means clustering loss function described in equation (11), and λ is a regularization parameter, which balances the reconstruction error by finding K-means-oriented hidden representations. Instead of applying the traditional SGD for the optimization process, the DCN introduces an alternating SGD optimization algorithm to update its parameters. There are three sets of parameters to be updated in a DCN: cluster centroids, data point cluster assignments, and network parameters. The proposed alternating SGD suggests that each set of parameters should be treated as a subproblem; thus, DCN optimizes the subproblems with respect to one of the cluster centroids, data point assignments, and network parameters while keeping the other two sets fixed. For instance, to update network parameters, both the cluster centroids and data point assignment are fixed, and then the corresponding gradient is calculated by backpropagation.

Fard et al. [38] proposed a deep K-means clustering algorithm named deep K-means (DKM), which is very similar to the DCN [37]. DKM differs from the DCN in the clustering loss only, where weighted K-means is employed instead of K-means. Equation (43) shows the DKM joint loss function:

$$\mathrm{L_{DKM}} = \mathrm{L_r} + \lambda\mathrm{L_{WKM}} \quad (43)$$

where L_r is the reconstruction loss as defined in equation (4), L_{WKM} is the weighted K-means clustering loss function described in equation (12), and λ regulates the trade-off between seeking good representation and good clustering results. The similarity weight of the K-means loss function is computed according to the softmax function.

$$\mathrm{S_{ij\_DKM}} = \frac{\exp\left(-\alpha\left\|\mathrm{z_i} - \mathrm{z}_{\mu_{\mathrm{j}}}\right\|_2^2\right)}{\sum_{\mathrm{j}'=1}^{\mathrm{k}}\exp\left(-\alpha\left\|\mathrm{z_i} - \mathrm{z}_{\mu_{\mathrm{j}'}}\right\|_2^2\right)} \quad (44)$$

where z_i is the learned representation of data point x_i, k is the number of clusters, z_{μ_j} is the representation of the cluster centroid μ_j, and α is a coefficient such that when its value is 0, all of the data points in the embedding space are very close, and when its value is relatively high, the points are sparse in the space. The network architecture and learning process of DKM is similar to that of DCN, except that instead of alternating between continuous gradient updates and discrete cluster assignment steps, DKM relies on the gradient update only to learn both the representation and clustering parameters.

Chen et al. [39] proposed a deep manifold clustering algorithm called deep manifold clustering (DMC) to address multimanifold clustering (MMC) [40].DMC's architecture is similar to that of DEN [36], where an SAE [17] is employed for representation learning and a DBN [14] is utilized to initialize the SAE parameters. In DMC, a locality-preserving auxiliary clustering loss is introduced such that the locality of a manifold can be interpreted as similar inputs, and therefore, should have similar representations. Thus, a data point can be recovered using the representation of its nearby point. Based on this observation, the DMC [39] locality-preserving loss function is defined as.

$$L_{LP\_DMC} = \frac{1}{k}\sum_{j\in N_k(i)}\left\|y_i - x_j\right\|_2^2 \tag{45}$$

where y_i is the reconstructed image of data point x_i and $N_k(i)$ is the indices set of k nearest neighbors of x_i. After the SAE weights and cluster centroids have been initialized, the joint training procedure proceeds by iteratively optimizing the joint loss function:

$$L_{DMC} = (1-\alpha)L_r + \alpha L_{LP_{DMC}} + \beta L_{WKM} \tag{46}$$

where L_r is the reconstruction loss defined in equation (4), L_{LP_DMC} is the locality-preserving loss function defined in equation (45), L_{WKM} is the weighted K-means clustering loss function presented in equation (12), α balances the importance between the reconstruction of x_i itself and its local neighborhood, and β is a parameter to balance the contribution of the first two terms and L_{WKM}. DMC uses the Gaussian-dependent kernel as the similarity weight of the weighted K-means loss function.

$$S_{ij\_DMC} = \frac{\exp\left(-\alpha\left\|z_i - z_{\mu_j}\right\|_2^2/2\sigma\right)}{\sum_{j'=1}^{k}\exp\left(-\alpha\left\|z_i - z_{\mu_{j'}}\right\|_2^2/2\sigma\right)} \tag{47}$$

Here, σ is the kernel bandwidth. The keystone point of DMC is to find the manifold center, because the cluster centers are most probably surrounded by nearby points with lower local density, and because they are at a relatively large distance from any points with a higher local density. Therefore, DMC calculates the density of the new representation by computing two metrics: the local density of a point, and its distance to points with higher density. The local density ρ_i of the representation z_i is defined as.

$$\rho_i = \sum_{j=1}^{n} e^{\frac{-\Delta_{ij}}{\hat{\Delta}}} \tag{48}$$

where Δ_{ij} is the distance between the representation z_i and z_j and $\widehat{\Delta}$ is a cut-off distance. Then, the points in the new embedding space are sorted based on their density in descending order, denoted by $\{\lambda_i\}_{i=1}^{n}$ with $\rho_{\lambda 1} \geq \rho_{\lambda 2} \geq \cdots \geq \rho_{\lambda n}$. The distance metric is therefore calculated as.

$$\xi_{\lambda_i} = \begin{cases} \min\limits_{\substack{\lambda_j \\ j<i}} \{\Delta_{\lambda_i\lambda_j}\}, i \geq 2 \\ \max\limits_{j\geq 2} \{\xi_{\lambda_j}\}, i = 1. \end{cases} \tag{49}$$

Next, a third metric is defined as

$$\gamma_i = \rho_i \xi_i \tag{50}$$

Similarly, the points in the new embedding space are sorted based on γ_i, as computed in equation (44) in descending order, and denoted by $\{\pi_i\}_{i=1}^{n}$ with $\gamma_{\pi 1} \geq \gamma_{\pi 2} \geq \cdots \geq \gamma_{\pi n}$. Assuming that the number of clusters k is known in advance, the cluster centers are determined by considering the k largest γ. The experiments reported in [39] showed that DMC outperformed the state-of-the-art multimanifold clustering methods.

C. Deep Spectral Clustering

Ji et al. [41] introduced deep subspace clustering networks (referred to as DSC-Nets) based on CAE [18] to learn nonlinear mapping. The network architecture of DSC-Nets includes three parts: a CNN encoder, a middle layer called the self-expressive layer, and a CNN decoder. In the self-expressive layer, the neurons are completely connected using linear weights without bias and nonlinear activation. The purpose of this FC layer is to encode the self-expressiveness property, as explained in section 3.3. Each node in this self-expressive layer is a representation z_i, and the weights correspond to the matrix C in equation (36) which are further used to construct affinities between all data points. Therefore, essentially, the self-expressive layer enables the network to learn the affinity matrix directly. First, DSC-Nets pre-train the CAE without the self-expressive layer to initialize the encoder and decoder parameters. Then, in the finetuning process, the DSC-Nets deep network is first trained, and the following joint loss function is recursively optimized:

$$L_{DSC-Nets} = L_r + L_{SE} \tag{51}$$

where L_r is the reconstruction loss defined in equation (4) and L_{SE} is the self-expressiveness loss as expressed in (36). When the training is completed, the parameters of the self-expressive layer are used to build an affinity matrix for spectral clustering, as explained in section 3.2. The experiments reported in [41] showed that DSC-Nets yielded superior results for small datasets. However, this method cannot be applied on large datasets because of the memory complexity of the algorithm [19].

Similar to DSC-Nets, in [42], Zhou et al. proposed deep adversarial subspace clustering (DASC) model which learns subspace clustering-friendly representations using AAE and self-expressiveness constraint. Given that, DASC optimizes the following objective function:

$$L_{DASC} = L_r + \alpha L_g + \beta L_{SE} \tag{52}$$

where L_r, L_g, and L_{SE} represent the reconstruction loss defined in (4), the generation loss in (9), and the self-expressiveness loss that defined in (36), respectively, and α and β are hyperparameters to balance the importance of the generation loss and the clustering loss, respectively. Upon the completion of the training process, spectral clustering is applied to the resulting affinity matrix.

Yang et al. in [43] presented a deep spectral clustering (DSC) approach based on AAE. In the proposed approach, the generator is a dual AE network (one encoder and two decoders) to enforce the reconstruction constraints for the latent representations and their noisy versions. As a consequence, the resulting latent representation will be more robust to noise. Hence, the reconstruction loss is updated to be in the following format:

$$L_{r\_DSC} = \frac{1}{n}\sum_{i=1}^{n} \|\widetilde{y_i} - y_i\|^2 + \delta L_r \tag{53}$$

where L_r is the reconstruction loss in (4), y_i is the reconstructed image of input x_i, $\widetilde{y_i}$ is the reconstructed image of the noisy version of the input x_i, and δ balances the strength of the two losses. Then, the mutual information estimation is employed to boost the discriminator with more information from the inputs. To achieve this, the feature map of the middle convolutional layer of the encoder is extracted and combined with the latent representation to obtain a new feature map. Therefore, the generation loss will be as follows:

$$L_{g\_DSC} = -\beta\left[\frac{1}{n}\sum_{i=1}^{n}\log D(x_i, z_i) + \log\left(1 - D(x_i, z_i)\right)\right] - \frac{\beta}{hw}\left[\sum_{i,j}\frac{1}{n}\sum_{k=1}^{n}\log D\left(C_{ij}, z_k\right) + \log\left(1 - D\left(C_{ij}, z_k\right)\right)\right] + \gamma L_{KL} \tag{54}$$

where D is the discriminator, C_{ij} represents the feature vector of the middle feature map at coordinates (i, j), z_i is the latent representation of input x_i, L_{KL} is the KL-divergence loss in equation (14), h and w represent the height and width of the feature map, and β and γ are balancing parameters. Furthermore, the latent representations are embedded into the eigenspace to cluster them using a spectral clustering technique.

D. More Deep Clustering Algorithms

Shah et al. [44] presented deep continuous clustering (DCC), a framework for joint nonlinear embedding learning and clustering. The DCC framework integrates an RCC algorithm [44] with an SAE [17] as a deep representation learning technique. DCC consists of two stages: initialization and optimization. During the initialization stage, the denoising SAE is trained based on reconstruction loss only to initialize the network parameters, i.e., weights and biases. Then, the SAE is finetuned, using the reconstruction loss only, to complete the initialization. At the end of this stage, the learned representation Z is obtained from the bottleneck layer to have the initialization $U = Z$. Then, the optimization is conducted by minimizing the joint loss function.

$$L_{DCC} = \frac{1}{D}L_r + \frac{1}{d}\left(L_{data_{DCC}} + \lambda L_{pairwise_{DCC}}\right) \tag{55}$$

where L_r is the AE reconstruction loss in equation (4), D is the dimensionality of the original input dataset, and d is the dimensionality of the lower-dimensional representations Z. DCC modifies the data loss introduced in RCC [44] as.

$$L_{data\_DCC} = \sum_{i=1}^{n}\rho(\|u_i - z_i\|_2; \mu_1) \tag{56}$$

where ρ is the scaled Geman–McClure function defined in equation (25). The pairwise loss is also modified by DCC as.

$$L_{pairwise\_DCC} = \sum_{(i,j)\in E} w_{ij}\rho\left(\left\|u_i - u_j\right\|_2; \mu_2\right) \tag{57}$$

The parameters μ_1 and μ_2 control the radii of the convex basins of the estimators. The weights w_{ij} are computed based on.

$$w_{ij} = \frac{\frac{1}{n}\sum_{k=1}^{N} N_k}{\sqrt{N_i N_j}} \tag{58}$$

where N_i is the degree of u_i in the graph. To balance the different terms, DCC sets λ and A according to equations (29) and (30), respectively. The network parameters, the representatives U, and the lower-dimensional representations Z are updated by an SGD optimization algorithm [45] through backpropagation. Other DCC parameters, such as λ, are iteratively updated during the optimization as in the RCC algorithm [44].

Jiang et al. [46] proposed Variational Deep Embedding (VaDE), a probabilistic generative clustering technique within a VAE framework. In VaDE, Mixture-of-Gaussian is assumed to be the prior of the probabilistic clustering. To model the data generative procedure, VaDE utilizes GMM to pick a cluster from which a latent embedding is generated. Then, VAE decodes the latent embedding into an observable. Then, VAE is trained to maximize the evidence lower bound (ELBO) [19] according to VAE loss (L_{VAE}) in equation (7). After maximizing the ELBO, the cluster assignment can be inferred by the learned GMM model. GMVA [47] is another probabilistic clustering algorithm based on VAE with a Gaussian mixture as a prior distribution. The main contribution of this algorithm is in introducing the minimum information constraint [48] to the VAE in order to overcome the problem of cluster degeneracy, caused by the over-regularization of the VAE. The GMVA approach is more complex than VaDE, and has shown worse results in practice [19]. However, both VaDE and GMVA suffer from high computational complexity [19].

Mukherjee et al. [49] addressed the problem of clustering in the latent space of GAN [19] by introducing the ClusterGAN framework. In order to establish non-smooth geometry of the latent space, a mixture of discrete and continuous latent variables is utilized. To accommodate that mixture of variables, a new backpropagation algorithm is introduced to obtain the latent variable given a data input. The experimental results showed that GAN is able to preserve latent space interpolation across different categories.

As shown in Table I, we compared the studied joint deep clustering algorithms based on their clustering technique, loss functions, and main contributions. From the presented review, deep clustering algorithms with autoencoders are the most common technique and this due to two reasons. We can summarize these two points as: (1) the ability to combine the autoencoders with the most clustering algorithm, (2) autoencoders reconstruction loss is capable to learn feasible representations and avoid trivial solutions. It is important to note that, the computational cost of autoencoder based deep clustering algorithms is highly affected by the clustering loss. However, for computational feasibility, such algorithms have limited network depth due to the symmetry architecture of autoencoder. On the other hand, deep clustering algorithms with VAE, AAE, and GAN minimize the variational lower bound on the marginal likelihood of data which make them theoretically guaranteed. Unfortunately, these clustering techniques suffer from high computational complexity. Comparing VAE deep clustering algorithms with AAE and GAN clustering algorithms, AAE and GAN algorithms are more flexible and diverse than VAE algorithms. Nonetheless, AAE and GAN based clustering algorithms have slow convergence rate.

V. Conclusion

Recently, clustering algorithms have benefited from the new deep learning research field. In fact, new active research studies are focused on integrating deep representation learning with clustering tasks. Beyond joint deep clustering algorithms, more recent algorithms have been proposed, some of which have been classified as separated deep clustering approaches, and others categorized as combined deep clustering techniques, but not joint. DeepCluster, clustering by unmasking, rank-constrained spectral clustering, SDEC, parameter-free clustering, and learning deep graph representation are all examples of not-joint deep clustering algorithms.

In this article, we reviewed the existing joint deep clustering algorithms by describing their network structure and analyzing their objective functions. Based on the survey of algorithms discussed here, theoretical analysis of how and why jointly optimizing reconstruction and clustering losses significantly improves the clustering performance is itself significant. Also, studying whether deep supervised learning techniques, such as data augmentation and regularization, are applicable and useful for deep unsupervised clustering is meaningful. Exploring the feasibility of applying the proposed joint deep clustering algorithms on sequential data is highly encouraged. Moreover, exploring the viability of combining deep clustering techniques with other unsupervised learning tasks such as transfer learning is strongly recommended.

Acknowledgment

The authors are grateful for the support of the Research Center of the College of Computer and Information Sciences, King Saud University. The authors thank the Deanship of Scientific Research and RSSU at King Saud University for their technical support.

References

[1] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms," Ann. Data Sci., vol. 2, no. 2, pp. 165–193, 2015.

[2] I. T. Jolliffe, "Principal Components in Regression Analysis," in Principal component analysis, Springer, 1986, pp. 129–155.

[3] M. Li and B. Yuan, "2D-LDA: A statistical linear discriminant analysis for image matrix," Pattern Recognit. Lett., vol. 26, no. 5, pp. 527–532, Apr. 2005.

[4] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," Neural Comput., vol. 10, no. 5, pp. 1299–1319, 1998.

[5] I. National and D. Recherche, "Generalized Discriminant Analysis Using a Kernel Approach," Neural Comput., vol. 12, no. 1, pp. 1–13, 1994.

[6] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," Science (80-.)., vol. 290, no. 5500, pp. 2319–2323, 2000.

[7] Yuexian Hou, Peng Zhang, Xingxing Xu, Xiaowei Zhang, and Wenjie Li, "Nonlinear Dimensionality Reduction by Locally Linear Inlaying," IEEE Trans. Neural Networks, vol. 20, no. 2, pp. 300–315, 2009.

[8] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets.," Neural Comput., vol. 18, no. 7, pp. 1527–54, 2006.

[9] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science (80-.)., vol. 313, no. 5786, pp. 504–507, 2006.

[10] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin, "Feed forward neural networks with random weights," in Proceedings - International Conference on Pattern Recognition, 1992, vol. 2, pp. 1–4.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Adv. Neural Inf. Process. Syst., pp. 1–9, 2012.

[12] D. Svozil, V. Kvasnieka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," 1997.

[13] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences," Atmos. Environ., vol. 32, no. 14–15, pp. 2627–2636, Aug. 1998.

[14] N. Lopes and B. Ribeiro, "Deep Belief Networks (DBNs)," in Machine Learning for Adaptive Many-Core Machines-A Practical Approach, Springer, 2015, pp. 155–186.

[15] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," Commun. ACM, vol. 54, no. 10, pp. 95–103, 2011.

[16] Y. Bengio, "Learning deep architectures for AI," Found. Trends Mach. Learn., vol. 2, no. 1, pp. 1–27, 2009.

[17] 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, pp. 3371–3408, 2010.

[18] B. Leng, S. Guo, X. Zhang, and Z. Xiong, "3D object retrieval with stacked local convolutional autoencoder," Signal Processing, vol. 112, pp. 119–128, 2015.

[19] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A Survey of Clustering with Deep Learning: From the Perspective of Network Architecture," IEEE Access, vol. 6, pp. 39501–39514, 2018.

[20] J. R. Hershey and P. A. Olsen, "Approximating the Kullback Leibler divergence between Gaussian mixture models," in ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2007, vol. 4, pp. IV-317-IV–320.

[21] W. Zhou and Q. Zhou, "Deep Embedded Clustering With Adversarial Distribution Adaptation," IEEE Access, vol. 7, pp. 113801–113809, 2019.

[22] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, vol. 1, pp. 281–296.

[23] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm," Appl. Stat., vol. 28, no. 1, p. 100, 1979.

[24] G. Celeux and G. Govaert, "Gaussian parsimonious clustering models," Pattern Recognit., vol. 28, no. 5, pp. 781–793, May 1995.

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

[26] S. A. Shah and V. Koltun, "Robust continuous clustering," Proc. Natl. Acad. Sci. U. S. A., vol. 114, no. 37, pp. 9814–9819, 2017.

[27] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[28] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation," in IJCAI International Joint Conference on Artificial Intelligence, 2017, pp. 1753–1759.

[29] S. Geman and D. E. McClure, “Statistical methods for tomographic image reconstruction,” Bull. Int. Stat. Inst, vol. 52, no. 4, pp. 5–21, 1987.

[30] L. Bottou, “Large-Scale Machine Learning with Stochastic Gradient Descent,” in Proceedings of COMPSTAT’2010, Physica-Verlag HD, 2010, pp. 177–186.

[31] X. Guo, X. Liu, E. Zhu, and J. Yin, “Deep Clustering with Convolutional Autoencoders,” in International conference on neural information processing, 2017, vol. 10635 LNCS, pp. 373–382.

[32] N. Mrabah, M. Bouguessa, and R. Ksantini, “Adversarial Deep Embedded Clustering: on a better trade-off between Feature Randomness and Feature Drift,” IEEE Trans. Knowl. Data Eng., pp. 1–1, 2020.

[33] K. G. Dizaji, A. Herandi, C. Deng, W. Cai, and H. Huang, “Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, vol. 2017-Octob, pp. 5747–5756.

[34] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Journal of Machine Learning Research, 2010, vol. 9, pp. 249–256.

[35] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised Deep Embedding for Clustering Analysis,” in International conference on machine learning, 2016.

[36] P. Huang, Y. Huang, W. Wang, and L. Wang, “Deep embedding network for clustering,” in 22nd International conference on pattern recognition, 2014, pp. 1532–1537.

[37] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, “Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering,” 2017.

[38] M. M. Fard, T. Thonet, and E. Gaussier, “Deep k-Means: Jointly clustering with k-Means and learning representations,” arXiv Prepr. arXiv1806.10069, 2018.

[39] D. Chen, J. Lv, and Z. Yi, “Unsupervised multi-manifold clustering by learning deep representation,” in AAAI Workshop - Technical Report, 2017, vol. WS-17-01-, pp. 385–391.

[40] R. Souvenir and R. Piess, “Manifold clustering,” in Proceedings of the IEEE International Conference on Computer Vision, 2005, vol. I, pp. 648–653.

[41] P. Ji, T. Zhang, H. Li, M. Salzmann, and I. Reid, “Deep subspace clustering networks,” in Advances in Neural Information Processing Systems, 2017, vol. 2017-Decem, pp. 24–33.

[42] P. Zhou, Y. Hou, and J. Feng, “Deep Adversarial Subspace Clustering,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2018, pp. 1596–1604.

[43] X. Yang, C. Deng, F. Zheng, J. Yan, and W. Liu, “Deep spectral clustering using dual autoencoder network,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019, vol. 2019-June, pp. 4061–4070.

[44] S. A. Shah and V. Koltun, “Deep Continuous Clustering,” arXiv Prepr. arXiv1803.01449, 2018.

[45] I. Goodfellow, Y. Bengio, and A. Courville, “Deep Learning,” Nature, vol. 521, no. 7553, p. 800, 2016.

[46] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, “Variational deep embedding: An unsupervised generative approach to Clustering,” in IJCAI International Joint Conference on Artificial Intelligence, 2017, pp. 1965–1972.

[47] N. Dilokthanakul et al., “Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders,” 2016.

[48] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, “Improved variational inference with inverse autoregressive flow,” in Advances in Neural Information Processing Systems, 2016, pp. 4743–4751.

[49] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, “ClusterGAN: Latent Space Clustering in Generative Adversarial Networks,” Proc. AAAI Conf. Artif. Intell., vol. 33, pp. 4610–4617, 2019.