Chapter 4
An Introduction to Deep Clustering

Gopi Chand Nutakki, Behnoush Abdollahi, Wenlong Sun, and Olfa Nasraoui

4.1 Introduction

Driven by the explosive growth in available data and decreasing costs of computation, Deep Learning (DL) has been paving a transformational path in machine learning and artificial intelligence [8, 17], with a dramatic impact on a variety of application domains, ranging from computer vision [15] and speech recognition [13] to natural language processing [5] and recommender systems [25, 27, 30]. DL found much of its fame in problems involving predictive modeling tasks such as classification and recommendation which are considered supervised learning. Deep learning has also been widely used to learn richer and better data representations from big data, without relying too much on human engineered features. Many of these deep representation networks rely on a preliminary unsupervised learning stage, referred to as unsupervised pretraining (e.g., autoencoders, matrix factorization, restricted Boltzmann machines, etc.), which learn better (deep) representations of the data that are known to drastically improve the results of supervised learning networks. Even though it started mostly within the realm of supervised learning, deep learning's success has recently inspired several deep learning-based developments in clustering algorithms which sit squarely within unsupervised learning. Most DL-based clustering approaches have exploited the representational power of DL networks for preprocessing clustering inputs in a way to improve the quality of clustering results and to make clusters easier to extract. However, clustering has not

G. C. Nutakki · B. Abdollahi · W. Sun
Knowledge Discovery & Web Mining Lab, University of Louisville, Louisville, KY, USA
e-mail: g0nuta01@louisville.edu; b0abdo03@louisville.edu; w0sun005@louisville.edu

O. Nasraoui (✉)
Department of Computer Engineering and Computer Science, University of Louisville, Louisville, KY, USA
e-mail: olfa.nasraoui@louisville.edu

O. Nasraoui, C.-E. Ben N'Cir (eds.), *Clustering Methods for Big Data Analytics*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-319-97864-2_4

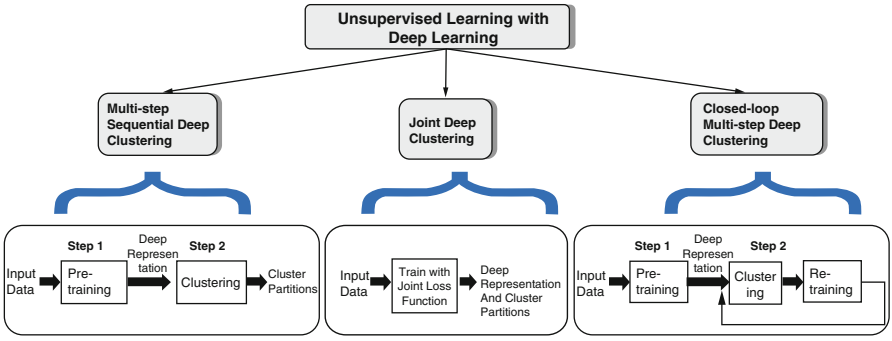


Fig. 4.1 Taxonomy of Deep Clustering presented in this chapter

always been the ultimate goal of these techniques, since some of them aim primarily to obtain richer deep representations, but employ clustering as an essential and enabling step to improve this representation. Yet, both approaches result in deep representations and clustering outputs, hence we will refer to all the approaches presented in this chapter as *Deep Clustering*.

Figure 4.1 helps further understand the taxonomy of Deep Clustering approaches that we present, which itself motivates the structure of this chapter. Deep Clustering approaches differ based on their overall algorithmic structure, network architectures, loss functions, and optimization methods for training (or learning the parameters). In this chapter, we focus on Deep learning for clustering approaches where those approaches either use deep learning for the purpose of grouping (or partitioning) the data and/or creating low rank deep representations or embeddings of the data, which among other possible goals, could play a significant supporting role as a building block of supervised learning. There may be several ways to come up with a taxonomy of deep clustering methods; our approach in this chapter is based on viewing the methods as a *process*. We thus present a simplified taxonomy based on the overall procedural structure or design of deep clustering methods. The simplified taxonomy helps both beginner and advanced readers. Beginners are expected to quickly grasp how almost all approaches are designed based on a small set of common patterns, while more advanced readers are expected to use and extend these patterns in order to design increasingly complex deep clustering pipelines that fit their own machine learning problem-solving aims. In our taxonomy, Deep Clustering approaches can be considered to fall into the following three broad families (see Fig. 4.1):

1. **Sequential multistep Deep Clustering approaches**: These approaches consist of two main steps. The first step learns a richer deep (also known as latent) representation of the input data, while the second step performs clustering on this deep or latent representation.
2. **Joint Deep Clustering approaches**: This family of methods include a step where the representation learning is tightly coupled with the clustering, instead of two separate steps for the representation learning and clustering, respectively.

The tight coupling is generally performed by optimizing a combined or joint loss function that favors good reconstruction while taking into account some form of grouping, clustering, or codebook representation of the data.

3. **Closed-loop multistep Deep Clustering approaches**: This family of methods contains two salient steps, similar to the first family (sequential multistep Deep Clustering); however, the steps alternate in an iterative loop instead of being performed in one feedforward linear fashion.

A common thread between all the above families is that clustering is performed on a different (so-called deep or latent) representation (DR) instead of the original data. The deep representation is typically of lower dimensionality and captures more easily the different hidden groups within the data. It is not surprising therefore that such deep representations are more conducive to better clustering, even using simple algorithms such as K-means although the latter tends to struggle when faced with high-dimensional data. It is well known that using a Deep Representation (DR) or preprocessing of the data using dimensionality reduction methods gives better clustering results. Deep Representation methods that are known to improve the results of K-means include: (1) linear mapping methods such as Principal Component Analysis (PCA) or Nonnegative Matrix Factorization (NMF) [4, 35] and (2) nonlinear approaches such as those used in Spectral Clustering [22, 33] and Deep Neural Network-based DR [3, 10, 12, 28, 32].

Underlying all three families are building blocks which are essential to designing most deep learning methods. These building blocks are:

- Deep representation models.
- Loss functions.

The following sections of the chapter will start by presenting the essential building blocks to Deep Clustering. Then, each family of Deep Clustering approaches will be discussed in order (sequential multistep, joint, and finally closed-loop multistep Deep Clustering).

4.2 Essential Building Blocks for Deep Clustering

Before we introduce deep clustering techniques, we first discuss the essential ingredients for any good clustering. The first important ingredient is that the algorithm must have good representation or features (and/or similarity or affinity measures). The second ingredient is a good cost or loss function that captures what a good representation or clustering is. For this reason, we consider that underlying all three families of Deep Clustering are the following set of building blocks which are essential to designing most deep learning methods:

- Deep representation models: These are unsupervised pretraining models which typically compute new deep or latent features or embeddings that are considered faithful but richer representations (DR) of the data.

- Loss functions: These are the objective or cost functions that are used to train the above deep representation models or to compute clusters from data.

These building blocks are described in the next subsections.

4.2.1 Learning Deep Representations

A high-level representation of features can be learned using deep neural networks (DNNs) that learn mappings from the input data to better (deep) representations (DR) that can in turn aid clustering algorithms to discover better partitions or cluster parameters. The features of these deep representations are generally extracted from one layer of the network, typically the latent hidden layer of an Autoencoder [31] or a Restricted Boltzmann Machine network [11]. The features can also be extracted from the concatenation of several layers [26].

Deep neural networks that can discover better feature mappings are typically trained in a similar way as unsupervised pretraining which is a common preliminary stage even in building supervised deep learning networks. Unsupervised pretraining strategies include denoising autoencoders [31] and Restricted Boltzmann Machines [11] which are known to reduce variance and learn hidden units that compute features of the input data that correspond to major factors of variation in the true input distribution. Hence, unsupervised pretraining can guide optimization towards basins of attraction of minima that allow better generalization from training data and add robustness to a deep architecture [6]. Autoencoders are in fact fundamentally connected to clustering as shown by Baldi [2] who presented a framework to study linear and nonlinear autoencoders, showing that learning in the Boolean autoencoder (which is the most nonlinear autoencoder) is equivalent to a clustering problem that can be solved in polynomial time for a small number of clusters, but becomes NP complete for a large number of clusters.

In addition to autoencoders, there are alternative representation extraction building blocks such as Matrix Factorization [4, 34]. Although linear, Matrix Factorization has proven to be a powerful strategy to learn latent representations that are useful in many applications. MF methods are no longer confined to be linear. In fact, a nonlinear extension of MF, called Generalized Matrix Factorization, was recently proposed and used in deep recommender systems [9]. Just like autoencoders and Restricted Boltzman Machines, they can easily be stacked to form hierarchical representation layers whose output is later used as DR for clustering or other tasks.

4.2.2 Deep Clustering Loss Functions

There are several types of loss functions that can be used in a Deep Clustering framework, of which we describe the four most important types ([1] discussed more loss function building blocks). The first type comes purely from learning a deep representation using deep learning techniques, and independent of any clustering

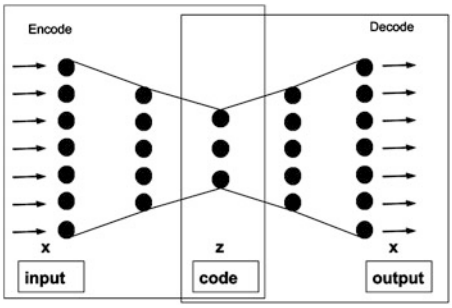


Fig. 4.2 The overall structure of an auto-encoder

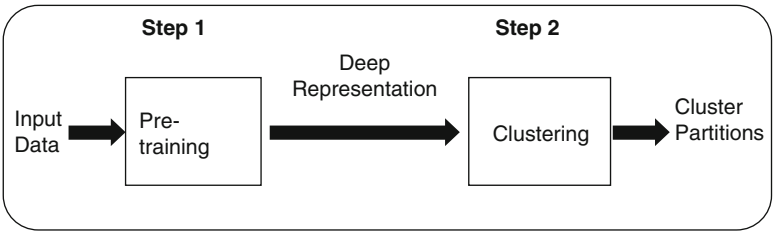


Fig. 4.3 Sequential Multistep Deep Clustering

(e.g., the reconstruction error of an autoencoder or matrix factorization). The second type is the *clustering loss*, which comes from a clustering process (e.g., the K-means' sum of squared errors objective function). The third type is a *joint loss function* that combines the first two types. The fourth type is a loss function meant to obtain crisper or harder cluster partitions in case of soft clustering. Soft or fuzzy clustering allows a data point to belong to multiple clusters with varying degrees of membership.

Autoencoder Reconstruction Loss An autoencoder consists of two parts: an encoder and a decoder [31] (see Fig. 4.3). The encoder maps (or encodes) its input data layer x to a hidden layer representation or a code z in a latent space Z. During training, the decoder tries to learn to make a faithful reconstruction x from the code z, making sure that useful information has not been lost by the encoding process. Once the training is done, the decoder part is no longer used, and only the encoder part (consisting of the input layer and hidden layer) is left for mapping its input to the latent space Z. This procedure allows autoencoders to learn useful representations in case the output's dimensionality is different from that of the inputs or when random noise is injected into the input [32]. They can also be used for dimensionality reduction [12]. The Autoencoder's reconstruction loss captures the distance or error between the input x_i to the autoencoder and the corresponding reconstruction or code $f(x_i)$, for instance, the mean squared error:

$$L = \sum_i \|x_i - f_{autoencoder}(x)\|^2. \tag{4.1}$$

Clustering Loss This type of loss is the objective function that guides a clustering process, such as K-means [20]. Given a set of data samples $\{\boldsymbol{x}_i\}_{i=1,\ldots,N}$ where $\boldsymbol{x}_i \in \mathbb{R}^M$, K-means strives to group the N data samples into K categories or cluster partitions of the data. This is done by minimizing the following loss or cost function:

$$\min_{\boldsymbol{M}\in\mathbb{R}^{M\times K},\{\boldsymbol{s}_i\in\mathbb{R}^K\}} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{M}\boldsymbol{s}_i\|_2^2 \tag{4.2}$$
$$\text{s.t. } s_{j,i} \in \{0,1\},\ \mathbf{1}^T\boldsymbol{s}_i = 1\ \ \forall i, j,$$

where $\boldsymbol{s}_i$ is the cluster membership assignment vector of data point i which has only one nonzero element for the cluster nearest to the data point; $s_{j,i}$ is the jth element of $\boldsymbol{s}_i$ and represents the membership assignment in the jth cluster (1 if this cluster is the nearest cluster to the data point and 0 otherwise); and the kth column of $\boldsymbol{M}$, $\boldsymbol{m}_k$, is the centroid of the kth cluster.

Joint Deep Clustering Loss Function A joint loss function is intended to jointly learn both DNN parameters and cluster parameters that aim to produce both better DR and better clusters. Hence, joint loss functions combine both DR and clustering objectives [23, 40]. Joint loss functions typically assume a generative model that generates an M-dimensional data sample by $\boldsymbol{x}_i = \boldsymbol{W}\boldsymbol{h}_i$, where $\boldsymbol{h}_i \in \mathbb{R}^R$ are latent factors and $\boldsymbol{W} \in \mathbb{R}^{M\times R}$ are encoding weights or loading coefficients from the data on those factors, and the latent factors' dimensionality $R \ll M$. They further assume that the data clusters are well-separated in latent domain (i.e., where $\boldsymbol{h}_i$ lives) but distorted by the transformation introduced by $\boldsymbol{W}$. A joint loss function can then be defined as follows [40]:

$$\min_{\boldsymbol{M},\{\boldsymbol{s}_i\},\boldsymbol{W},\boldsymbol{H}} \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F^2 + \lambda \sum_{i=1}^{N} \|\boldsymbol{h}_i - \boldsymbol{M}\boldsymbol{s}_i\|_2^2$$
$$+ r_1(\boldsymbol{H}) + r_2(\boldsymbol{W}) \tag{4.3}$$
$$\text{s.t. } s_{j,i} \in \{0,1\},\ \mathbf{1}^T\boldsymbol{s}_i = 1\ \ \forall i, j,$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$, $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N]$, and $\lambda \geq 0$ is a parameter for balancing data fidelity or reconstruction (first term) and the latent cluster structure (second term). In (4.3), the first term performs DR and the second term performs latent clustering. The terms $r_1(\cdot)$ and $r_2(\cdot)$ are regularization terms that try to favor nonnegativity or sparsity and are used to prevent trivial solutions, such as zero latent factor values [37].

Cluster Assignment Hardening Loss Assuming soft assignments of data points to clusters, obtained using fuzzy clustering techniques, probabilistic clustering methods, or normalized similarities between points and centroids. The *cluster assignment hardening* loss tries to enforce making soft assignment probabilities stricter or harder by encouraging cluster assignment probability distribution Q to approach an auxiliary (target) distribution P which guarantees this constraint. The

following auxiliary distribution [34] improves cluster purity and stresses data points that are assigned with high confidence:

$$p_{ij} = \frac{q_{ij}^2 / \Sigma_i q_{ij}}{\Sigma_{j'}(q_{ij'}^2 / \Sigma_i q_{ij'})}. \tag{4.4}$$

because it forces assignments to have stricter probabilities (closer to 0 and 1). This happens due to squaring the original distribution and then normalizing. Cluster hardening can finally be accomplished by minimizing the distance between the original cluster assignment probability distribution and the auxiliary or target crisp distribution, which can be done by minimizing (via neural network training) their Kullback–Leibler divergence [16], given by:

$$L = \mathrm{KL}(P \| Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{4.5}$$

4.3 Sequential Multistep Deep Clustering

Sequential multistep Deep Clustering approaches consist of two major steps, as shown in Fig. 4.3. The first step learns a richer deep (also known as latent) representation (DR) of the input data, while the second step performs clustering on this deep or latent representation. In the following, we describe a few representative algorithms that fall into this family of Deep Clustering methods.

4.3.1 Fast Spectral Clustering

Fast Spectral Clustering [38] is a classical multistep approach where an autoencoder is trained to produce a latent or deep representation (DR), which is later used as input to clustering with the K-means algorithm. In contrast to multistep methods that train the autoencoder directly on the input data, Fast Spectral Clustering first extracts an embedding S from the Laplacian of a precomputed similarity or kernel matrix W and then trains the autoencoder to encode this embedding (see Algorithm 1). The similarities are computed using a Gaussian kernel between every input data point and p landmark points. The p landmarks can be obtained in different ways. They

Algorithm 1 Fast Spectral Clustering (FSC)

Input: Input data $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$;
Output: k clusters of the data set 1: Select p landmarks 2: Compute similarity matrix W between data points and landmarks 3: Compute the degree matrix: $D = diag(W^\top \mathbf{w}^s)$ 4: Compute S, the input to the autoencoder: $\mathbf{s}_i = d_{ii}^{-1/2} \mathbf{w}_i$ 5: Train an autoencoder using S as its input 6: Run k-means on the latent space DR of the trained autoencoder

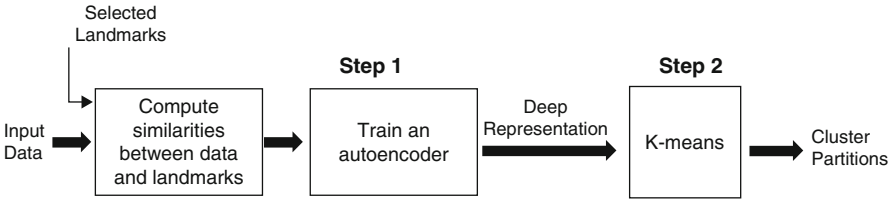


Fig. 4.4 Fast Spectral Clustering

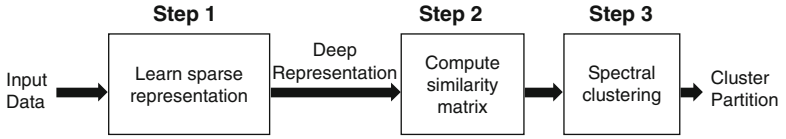


Fig. 4.5 Deep Sparse Subspace Clustering

can be randomly sampled from the original data set or selected from the centroids of p clusters by running k-means or using column subset selection methods [7]. The Laplacian matrix is computed using:

$$Ł_{norm} = D^{-1/2} M D^{-1/2} = D^{-1/2} W^{\top} W D^{-1/2} = S \cdot S^{T} \tag{4.6}$$

which yields the embedding $S = WD^{-1/2}$ that will be used as the input to the autoencoder (Fig. 4.4).

4.3.2 Deep Sparse Subspace Clustering (SSC)

Given a data set $X = [x_1, x_2, \ldots, x_n] \in R^{d \times n}$, SSC [24] seeks to linearly reconstruct the i-th sample $\mathbf{x}_i$ using a small number of other samples, hence producing representation coefficients that are sparse. The optimization consists of minimizing the reconstruction loss function as follows:

$$\min_{\mathbf{c}_i} \frac{1}{2} \|\mathbf{x}_i - \mathbf{X}\mathbf{c}_i\|_F^2 + \gamma \|\mathbf{c}_i\|_1 \qquad \text{s.t. } c_{ii} = 0 \tag{4.7}$$

where $\|\cdot\|_1$ denotes the ℓ_1-norm and c_{ii} denotes the i-th element in $\mathbf{c}_i$ which encourages $\mathbf{c}_i$ to be sparse, and the constraint $c_{ii} = 0$ avoids trivial solutions. After solving for the sparse representation $\mathbf{C}$, an affinity matrix $\mathbf{A}$ is calculated using $\mathbf{A} = |\mathbf{C}| + |\mathbf{C}|^T$. This matrix is finally used as input to spectral clustering to obtain clustering results (see Fig. 4.5).

4.3.3 Deep Subspace Clustering (DSC)

In most existing subspace clustering methods including SSC, each data input is encoded as a linear combination of the whole data set. However, linear combinations may not be sufficient for representing high-dimensional data which usually lie on nonlinear manifolds. Deep Subspace Clustering (DSC) [24] represents each input sample as a combination of others, similar to SCC. However, instead of a linear combination, DSC learns a mapping using explicit hierarchical transformations in a neural network and simultaneously learns the reconstruction coefficients $\mathbf{C}$. The neural network consists of $M + 1$ stacked layers with M nonlinear transformations, which takes a given data input x as the input to the first layer. The input to the first layer of the neural network is denoted as $h^{(0)} = x \in R^d$, while $h^{(m)}$ denotes the output of the subsequent (m-th) layers (where $m = 1, 2, \ldots, M$ indexes the layers), $W^{(m)} \in R^{d^{(m)} \times d^{(m-1)}}$ and $b^{(m)} \in R^{d^{(m)}}$ denote the weights and bias associated with the m-th layer, respectively, and $d^{(m)}$ is the dimension of the output of the m-th layer. If $H^{(M)}$ denotes the collection of the corresponding outputs given by the neural network,

$$\mathbf{H}^{(M)} = [\mathbf{h}_1^{(M)}, \mathbf{h}_2^{(M)}, \ldots, \mathbf{h}_n^{(M)}], \tag{4.8}$$

the optimization problem of DSSC can then be expressed as:

$$\begin{aligned} \min_{\mathbf{W}^{(m)}, \mathbf{b}^{(m)}, \mathbf{C}} & \frac{1}{2} \|\mathbf{H}^{(M)} - \mathbf{H}^{(M)}\mathbf{C}\|_F^2 + \gamma \|\mathbf{C}\|_1 \\ & + \frac{\lambda}{4} \sum_{i=1}^{n} \|(\mathbf{h}_i^{(M)})^T \mathbf{h}_i^{(M)} - 1\|_2^2 \\ & \text{s.t.} diag(\mathbf{C}) = 0, \end{aligned} \tag{4.9}$$

where λ is a positive trade-off parameter, the first term is designed to minimize the discrepancy between $\mathbf{H}^{(M)}$ and its reconstructed representation, the second term regularizes $\mathbf{C}$ for some desired properties, and the third term is designed to remove an arbitrary scaling factor in the latent space, without which the neural network may collapse in the trivial solutions $\mathbf{H}^{(M)} = \mathbf{0}$.

DSSC simultaneously learns M nonlinear mapping functions $\{\mathbf{W}^{(m)}, \mathbf{b}^{(m)}\}_{m=1}^{M}$ and n sparse codes $\{\mathbf{c}_i\}_{i=1}^{n}$ by alternatively updating one variable while fixing all others, until convergence. Stochastic sub-gradient descent (SGD) is used to update the parameters $\{\mathbf{W}^{(m)}, \mathbf{b}^{(m)}\}_{m=1}^{M}$. An ℓ_2-norm regularizer can also be added to avoid overfitting [15, 21]. After obtaining $\mathbf{C}$, a similarity graph is constructed using $\mathbf{A} = |\mathbf{C}| + |\mathbf{C}|^T$ and this graph is used as input to clustering.

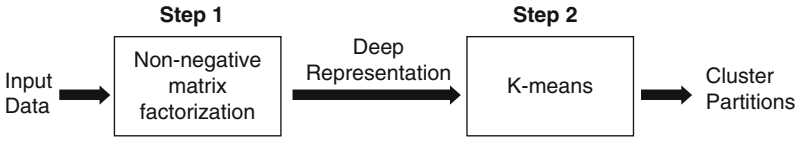
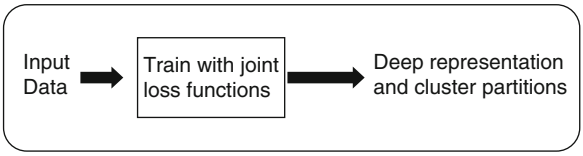


Fig. 4.6 Nonnegative matrix factorization (NMF) + K-means

Fig. 4.7 Joint deep clustering



4.3.4 Nonnegative Matrix Factorization (NMF) + K-Means

This approach first applies NMF [18] to learn a latent or deep representation, and then applies K-means to this reduced-dimension representation [39] (see Fig. 4.6). It is a common baseline for evaluating more complex architectures.

4.4 Joint Deep Clustering

Joint Deep Clustering is a family of methods that include a step, where the representation learning is tightly coupled with the clustering, instead of two separate steps for the representation learning and clustering, respectively. The tight coupling is generally performed by optimizing a combined or joint loss function that favors good reconstruction while taking into account some form of grouping, clustering, or codebook representation of the data. In the following, we describe a few representative algorithms that fall into this family of Deep Clustering methods (Fig. 4.7).

4.4.1 Task-Specific and Graph-Regularized Network (TAGnet)

The goal of TAGnet [41] is to learn a discriminative embedding that is optimal for clustering. Different from generic deep architectures, TAGnet is designed in a way to take advantage of the successful sparse code-based clustering pipelines. The TAGnet approach includes a feed-forward architecture, termed *Task-specific And Graph-regularized Network* (TAGnet), to learn discriminative features, and a joint clustering-oriented loss function (see Fig. 4.8). Its aim is to learn features that are optimized under clustering criteria, while encoding graph constraints to regularize the target solution. The solution to training the network parameters in TAGnet is

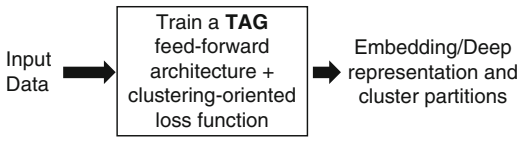


Fig. 4.8 The TAGnet approach

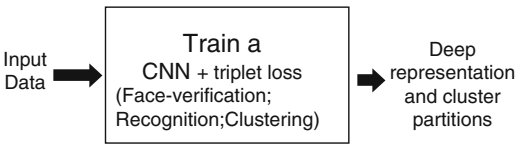


Fig. 4.9 The FaceNet approach

derived from a theorem that shows that the optimal sparse code can be obtained as the fixed point of an iterative application of a shrinkage operation on the degree matrix and the Laplacian derived from the graph's affinity matrix.

4.4.2 FaceNet

Designed for face recognition tasks, FaceNet [28] is a unified system for face verification, recognition, and clustering (see Fig. 4.9). The approach aims to learn a Euclidean embedding for images using a deep convolutional network. The network is trained to produce good embeddings such that the squared L2 distances in the embedding space directly correspond to the notion of face similarity; thus, faces of the same person should have small distances while faces of different people should be separated by large distances. Training is achieved by minimizing a triplet loss function that is used to simultaneously achieve face verification, recognition, and clustering. An embedding $f(x)$ is extracted from an image x into a feature space $\mathbb{R}^d$, where the squared distance between all faces of the same identity is small across all imaging conditions, whereas the squared distance between a pair of face images from different identities is large.

4.4.3 Deep Clustering Network (DCN)

Deep Clustering Network (DCN) [39] is a multistep approach (see Fig. 4.14) that starts by pretraining an autoencoder based on reconstruction loss minimization, then feeds the deep representation output to K-means for clustering. After clustering, the autoencoder is retrained by minimizing a joint loss function combining reconstruction loss and the K-means clustering loss. DCN then alternates between the autoencoder network training and cluster updates. DCN's results on the MNIST data set outperformed the results of the similar DEC approach, presented earlier (Fig. 4.10).

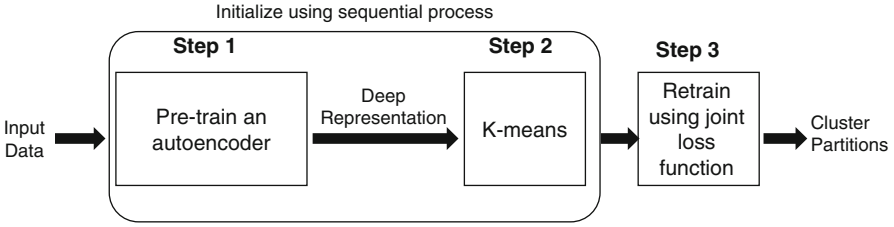


Fig. 4.10 Overview of a Deep Clustering Network (DCN) model

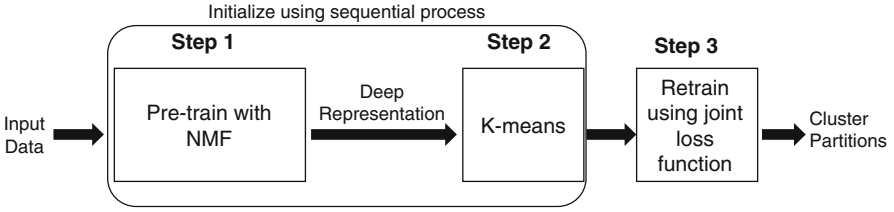


Fig. 4.11 Joint NMF + K-means (JNKM) model

4.4.4 Joint NMF and K-Means (JNKM)

Similar to DCN, JNKM [39] performs joint deep representation and K-means clustering in several steps, except that it uses NMF instead of an autoencoder to obtain the DR. This method has the same structure as DCN (see Fig. 4.11), replacing the autoencoder training with NMF, and was used as a baseline by [39]. Recall that the main difference between NMF and autoencoder networks is the nonlinearity of the latter's mapping.

4.5 Closed-Loop Multistep Deep Clustering

Closed-loop multistep Deep Clustering approaches are a family of methods that consist of two salient steps, similar to the first family (sequential multistep Deep Clustering); however, these steps alternate in a loop instead of being performed in one feedforward linear fashion. In the following, we describe a few representative algorithms that fall into this family of Deep Clustering methods (Fig. 4.12).

Deep Embedded Clustering (DEC) uses autoencoders as network architecture and initialization method, and uses K-means for clustering [34] (see Fig. 4.13). DEC first pretrains an autoencoder using a standard input reconstruction loss function. Then, the DR resulting from this autoencoder is fed to K-means to obtain clusters. Next, the autoencoder network is fine-tuned using the cluster assignment hardening loss (see the Building Blocks section above) and the clustering centers

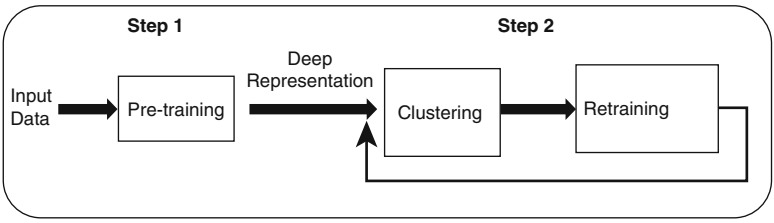


Fig. 4.12 Closed-loop multistep deep clustering

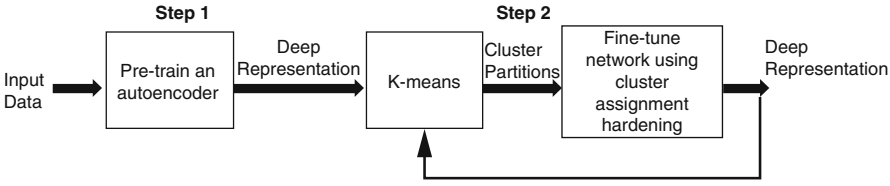


Fig. 4.13 Deep Embedded Clustering (DEC)

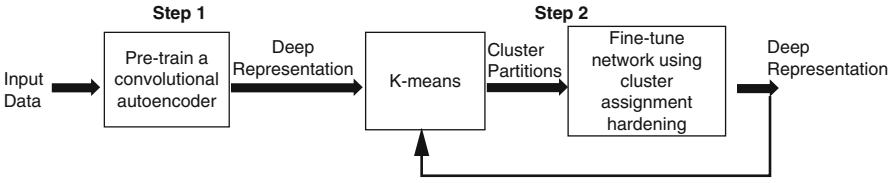


Fig. 4.14 Discriminatively Boosted Clustering (DBC)

are updated again. This process is then repeated. The clusters are iteratively refined by learning from their high confidence assignments with the help of the auxiliary target distribution.

Discriminatively Boosted Clustering (DBC) [19] shares many similarities with DEC except for using convolutional autoencoders. It also uses K-means for clustering and the same training method: pretraining with autoencoder reconstruction loss, clustering the autoencoder's output DR, then fine-tuning the autoencoder using a cluster assignment hardening loss (see Fig. 4.14). The main advantage of DBC over DEC is its superior performance on image data, as expected, due to using convolutional layers.

Clustering CNN (CCNN) uses a Convolutional Neural Network [29] to achieve joint clustering and deep representation learning. The features are extracted from one of the internal layers of the CCNN, while the cluster labels are considered to be the output predictions coming from the CCNN's softmax layer. The initial cluster centers that are used to compute the joint loss function in the pretraining phase are based on selecting k random images from the data set. After clustering the features coming from the DR using K-means, the assigned cluster labels are compared with the labels predicted by the CCN's softmax layer to compute a

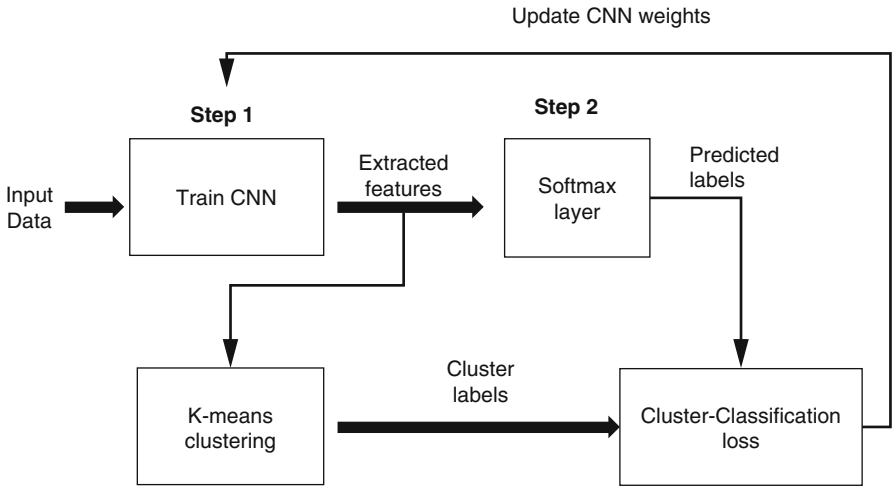


Fig. 4.15 Clustering Convolutional Neural Network (CCNN)

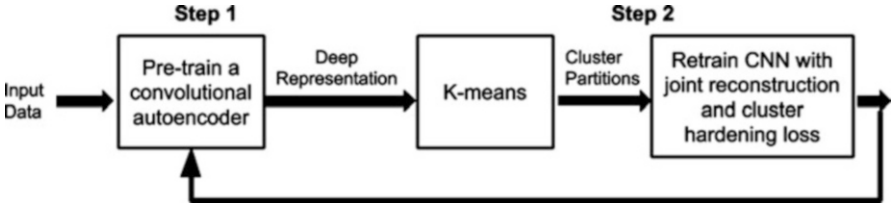


Fig. 4.16 Aljalbout's closed loop approach using a joint loss function

clustering-classification loss. This loss function is then used to retrain the CCNN parameters. The process is repeated with the extracted DR features again used to update the cluster centroids, essentially entering a repetitive loop of CCN training and clustering (see Fig. 4.15).

Aljalbout et al. [1] proposed a closed loop strategy that also iterates autoencoder training and K-Means clustering in a loop. Like CCNN, they used a convolutional architecture for extracting the DR in the first step to handle image data. However, the main difference is that they retrain the network weights using a joint loss function combining both reconstruction and cluster hardening loss (see Fig. 4.16). Their method was shown to outperform other strategies on the MNIST and COIL20 benchmark image data sets based on the external cluster evaluation metrics, Normalized Mutual Information and cluster accuracy [1].

4.6 Conclusions

In this chapter, we presented a simplified taxonomy with several representative examples of Deep Clustering methods which are built on algorithms that result in both deep or latent representations (DR) and (either as an explicit aim or as a byproduct) clustering outputs, such as a partition of the input data and cluster centroids. Our simplified taxonomy is based on the overall procedural structure or design of most Deep Clustering methods, thus dividing Deep Clustering approaches into three families: sequential multistep, joint, and closed-loop multistep Deep Clustering methods. In addition to describing several algorithms that fall under the three main families above, we discussed the most important building blocks that are essential to Deep Clustering methods.

Together with the Deep Clustering building blocks, our simplified taxonomy should help beginning readers get a quick grasp of how most approaches are designed, while guiding more advanced readers, and following the tradition of the DL community, to stack and combine the modular patterns that we have presented in order to design even more sophisticated deep clustering pipelines for their specific problem-solving needs.

While we have not detailed the implementation details or computational costs of Deep Clustering, scaling Deep Clustering follows along the steps of scaling DL in general. This means that strategies such as SGD play an essential role in fast optimization in the presence of big data sets, while GPU clusters play a critical role in accelerating computation due to the highly distributed nature of most DL architectures and weight updates. As with conventional clustering, graph-based, agglomerative, and spectral methods are notoriously costly in both computation time and memory requirements because of handling pairwise similarity matrices and Laplacians. However, this can be addressed by using a small number of selected landmarks instead of the entire data set when computing pairwise metrics, such as was done in Fast Spectral Clustering [38].

Deep Clustering has been applied to several big data domains, ranging from image to text and time series (earthquake) data, and we expect that in the future, it will have an impact on even more diverse domains and applications, in the same tradition of older clustering algorithms. Deep Clustering is hence expected to continue the tradition of clustering algorithms and to expand their ability to elucidate the hidden structure in big data and thus contribute to better understanding, retrieval, visualization, and organization of big data, in addition to being an important component of complex decision-making systems [14, 36].

References

1. E. Aljalbout, V. Golkov, Y. Siddiqui, D. Cremers, Clustering with deep learning: taxonomy and new methods (2018). Arxiv preprint arXiv:1801.07648
2. P. Baldi, Autoencoders, unsupervised learning, and deep architectures, in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning* (2012)

3. J. Bruna, S. Mallat, Invariant scattering convolution networks. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1872–1886 (2013)
4. D. Cai, X. He, J. Han, Locally consistent concept factorization for document clustering. IEEE Trans. Knowl. Data Eng. **23**(6), 902–913 (2011)
5. R. Collobert, J. Weston, A unified architecture for natural language processing: deep neural networks with multitask learning, in *Proceedings of the 25th International Conference on Machine Learning* (ACM, New York, 2008), pp. 160–167
6. D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. **11**, 625–660 (2010)
7. A.K. Farahat, A. Elgohary, A. Ghodsi, M.S. Kamel, Greedy column subset selection for large-scale data sets. Knowl. Inf. Syst. **45**(1), 1–34 (2015)
8. I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, *Deep Learning*, vol. 1 (MIT Press, Cambridge, 2016)
9. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee (2017), pp. 173–182
10. J.R. Hershey, Z. Chen, J. Le Roux, S. Watanabe, Deep clustering: discriminative embeddings for segmentation and separation, in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, Piscataway, 2016), pp. 31–35
11. G.E. Hinton, A practical guide to training restricted Boltzmann machines, in *Neural Networks: Tricks of the Trade* (Springer, Berlin, 2012), pp. 599–619
12. G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
13. G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath et al., Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
14. A.K. Jain, Data clustering: 50 years beyond k-means. Pattern Recogn. Lett. **31**(8), 651–666 (2010)
15. A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
16. S. Kullback, R.A. Leibler, On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951)
17. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436 (2015)
18. D.D. Lee, H.S. Seung, Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755), 788 (1999)
19. F. Li, H. Qiao, B. Zhang, X. Xi, Discriminatively boosted image clustering with fully convolutional auto-encoders (2017). Arxiv preprint arXiv:1703.07980
20. S. Lloyd, Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
21. G. Montavon, K.-R. Müller, Better representations: invariant, disentangled and reusable, in *Neural Networks: Tricks of the Trade* (Springer, Berlin, 2012), pp. 559–560
22. A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: analysis and an algorithm, in *Advances in Neural Information Processing Systems* (2002), pp. 849–856
23. V.M. Patel, H. Van Nguyen, R. Vidal, Latent space sparse subspace clustering, in *2013 IEEE International Conference on Computer Vision (ICCV)* (IEEE, Piscataway, 2013), pp. 225–232
24. X. Peng, J. Feng, S. Xiao, J. Lu, Z. Yi, S. Yan, Deep sparse subspace clustering (2017). ArXiv preprint arXiv:1709.08374
25. M. Quadrana, A. Karatzoglou, B. Hidasi, P. Cremonesi, Personalizing session-based recommendations with hierarchical recurrent neural networks, in *Proceedings of the Eleventh ACM Conference on Recommender Systems* (ACM, New York, 2017), pp. 130–137
26. S. Saito, L. Wei, L. Hu, K. Nagano, H. Li, Photorealistic facial texture inference using deep neural networks, in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, vol. 3 (2017)

27. R. Salakhutdinov, A. Mnih, G. Hinton, Restricted Boltzmann machines for collaborative filtering, in *Proceedings of the 24th International Conference on Machine Learning* (ACM, New York, 2007), pp. 791–798
28. F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: a unified embedding for face recognition and clustering, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015)
29. C.-J. Tsai, Y.-W. Tsai, S.-L. Hsu, Y.-C. Wu, Synthetic training of deep CNN for 3d hand gesture identification, in *2017 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)* (IEEE, Piscataway, 2017), pp. 165–170
30. A. Van den Oord, S. Dieleman, B. Schrauwen, Deep content-based music recommendation, in *Advances in Neural Information Processing Systems* (2013), pp. 2643–2651
31. P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in *Proceedings of the 25th International Conference on Machine Learning* (ACM, New York, 2008), pp. 1096–1103
32. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**, 3371–3408 (2010)
33. U. Von Luxburg, A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
34. J. Xie, R. Girshick, A. Farhadi, Unsupervised deep embedding for clustering analysis, in *International Conference on Machine Learning* (2016), pp. 478–487
35. R. Xu, Measuring explained variation in linear mixed effects models. Stat. Med. **22**(22), 3527–3541 (2003)
36. D. Xu, Y. Tian, A comprehensive survey of clustering algorithms. Ann. Data Sci. **2**(2), 165–193 (2015)
37. Y. Yang, M.C. Gupta, K.L. Dudley, Towards cost-efficient EMI shielding materials using carbon nanostructure-based nanocomposites. Nanotechnology **18**(34), 345701 (2007)
38. A.Y. Yang, S.S. Sastry, A. Ganesh, Y. Ma, Fast ℓ 1-minimization algorithms and an application in robust face recognition: a review, in *2010 17th IEEE International Conference on Image Processing (ICIP)* (IEEE, Piscataway, 2010), pp. 1849–1852
39. B. Yang, X. Fu, N.D. Sidiropoulos, M. Hong, Towards k-means-friendly spaces: simultaneous deep learning and clustering (2016). Arxiv preprint arXiv:1610.04794
40. J. Yang, D. Parikh, D. Batra, Joint unsupervised learning of deep representations and image clusters, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 5147–5156
41. M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, D. Cai, Graph regularized sparse coding for image representation. IEEE Trans. Image Process. **20**(5), 1327–1336 (2011)