Deep Learning Clustering

An Overview of the Current Techniques and Challenges

Jan Rodríguex Miret

June 1, 2021

1 Introduction

The field of deep clustering has increasingly received more attention and promises to open the door for new applications in many different fields like market segmentation, recommendation systems, biomedicine, and bioinformatics [Karim et al., 2020], where they are extensively used.

Deep clustering refers to the use of deep neural networks to transform the original input data to a new representation from which the clustering occurs. The typical deep clustering approach is to use a specific type of neural network for the task to solve and then apply a traditional clustering algorithm like k-means or some variation of it, although several frameworks are discussed throughout the document. This transformation of the input to a new more meaningful space has to be learned, in this case by a neural network, in what is called representation learning.

In clustering problems, the ground truth labels are not available, which is why this representation and clustering learning must be unsupervised or self-supervised (where we use some data augmentation of the input).

In this document, an overview of the state of the art of deep clustering is presented, starting with an introductory theory of different aspects related to deep clustering methods, and their corresponding classification. Then, a set of the most relevant algorithms in the literature is exposed with the distinctions of aspects explained. Finally, a discussion of the current advantages, challenges, and possible applications of this work is commented on.

Concretely, this document begins with an explanation of the most common architectures used. Then, a classification of the loss functions used for deep clustering follows, detailing the usual training phases of these algorithms. In order to understand better the development of deep clustering techniques, Sections 4 and 5 explain how are they validated and what is the set of features.

2 Neural network architectures for deep clustering

As already mentioned, the common approach to deep clustering is to use a neural network architecture to learn deep representations of the input data such that a clustering algorithm can achieve better results than if directly applied to the input feature space. For that, multiple types of neural network architectures are used in deep learning-based algorithms for clustering.

In this section, some common deep neural network architectures are summarized. These architectures presented serve as frameworks and most of the techniques proposed in the literature use them, or make some adaptations or combinations of them. They are exposed sequentially so the explanation of them can be more guided, but the properties of the algorithms based on each architecture are detailed in 6.

2.1 Basic case: Multilayer Perceptron (MLP)

A multilayer perceptron is the most basic deep neural network [Schmidhuber, 2015]. A neural network is considered *deep* when it has a sufficiently large number of hidden layers. The hidden layers are those between the input and the output, as can be visually seen in Figure 1.

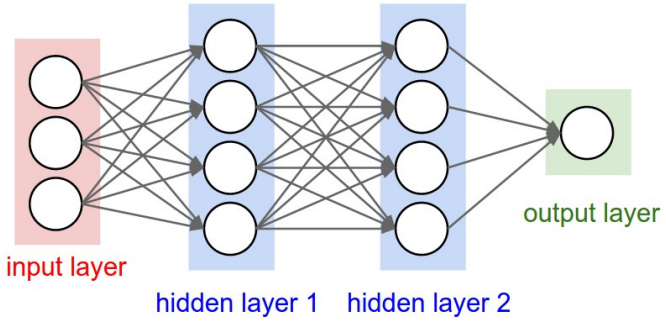

Figure 1: A multilayer perceptron with two hidden layers (in blue). The red and green layers denote the input and output layers. Note that the output layer may have more than one neuron. Extracted from Stack Exchange [https://stats.stackexchange.com/questions/256342/how-many-learnable-parameters-does-a-fully-connected-layer-have-without-the-bias].

The typical multilayer perceptron is a type of feedforward network in which contiguous layers are fully connected. That means that the input of each neuron at a layer l_n is a weighted sum of the outputs of all neurons of the layer

l_{n-1}. The neuron is then fired with a value equal to this mentioned sum but after being processed by some activation function, usually a Sigmoid or a Rectified Linear Unit (ReLU), which introduces some non-linearity.

During training, these weights between neurons are updated in a way that minimizes the loss function. A more detailed explanation of how the training is performed and the different losses used for that purpose is given in Section 3.

The rest of the architectures explained below follow a similar structure and training procedure since it is something common among all neural networks. Their technical peculiarities are described in each of the following subsections.

2.2 Convolutional Neural Network (CNN)

Inspired by how the visual cortex works in animals, convolutional neural networks are used for data that is shift-invariant and where spatial information is important. They have become the standard architecture for different image and text-related tasks, due to their good performance and computational efficiency compared to other neural networks (the neurons share weights, so fewer parameters have to be learned).

Simplifying, they are usually a concatenation of blocks with convolutional layers (where the convolution of the image features and the learned kernels takes place) and a subsampling layer (normally a max pooling).

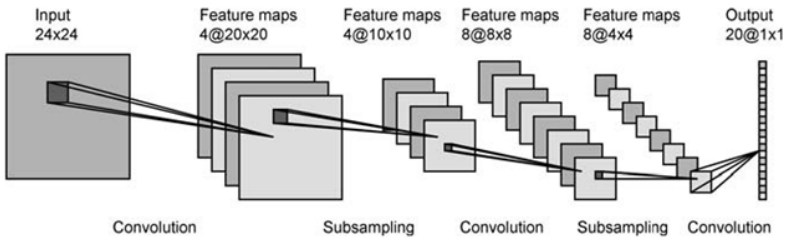

Figure 2: A convolutional neural network with the different operations performed at every layer. The output of the convolutions is then flattened to a vector of the feature maps, which often serves as input to a fully-connected network. Extracted from what-when-how [http://what-when-how.com/wp-content/uploads/2012/07/tmp725d63_thumb.png].

In deep clustering, though, it is more common to have a convolutional autoencoder, which is a convolutional adaptation of the autoencoder network architecture described later in Section 2.3.

2.3 Autoencoder (AE)

The autoencoder is a kind of neural network that is composed of two subnetworks: an encoder and a decoder. The dimensionality of the hidden layer connecting both subnetworks is usually lower than the input's, which ensures that this bottleneck layer contains a compressed and meaningful representation of the data.

The encoder part can be seen as a function $f(x)$ that is trying to capture these reduced set of latent features z, from which the decoder has to reconstruct the same input x by applying a function $g(z) = g(f(x))$.

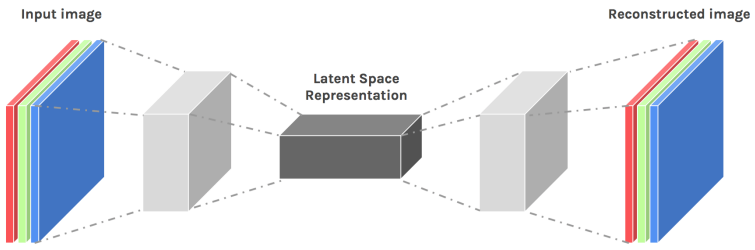

Figure 3: General architecture of a convolutional autoencoder. Extracted from sefiks.com [https://sefiks.com/2018/03/23/convolutional-autoencoder-clustering-images-with-neural-networks/].

A convolutional autoencoder (see Figure 3) is a special kind of autoencoder based on convolutional neural networks, instead of using a fully-connected neural architecture. The purpose of the network is the same.

Usually, in deep clustering, the decoder is not used after training, since the clustering is performed over the embedded representations h. More details on that are in Section 3.1.1 and 6.1.

2.4 Variational Autoencoder (VAE)

A variational autoencoder is a special type of autoencoder where the latent representations of data are not from an arbitrary function that the network learns, but have to follow a given distribution (typically a normal distribution). Apart from the reconstruction loss (3.1.1), it also uses the Kullback-Leibler divergence between the latent feature distribution and the assumed normal distribution. By imposing this constraint, and training for enough time and data, we obtain a latent variable model. A variational autoencoder is thus a generative model, so new instances can be sampled from this latent space Z.

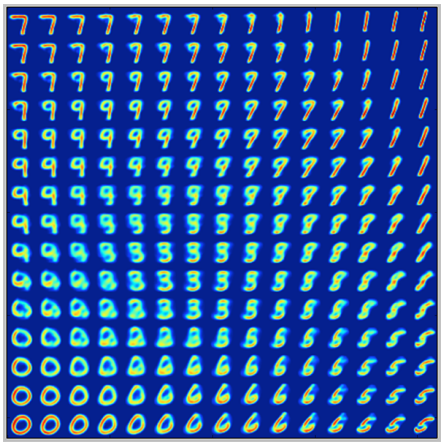

Figure 4: Visualization of a 2-dimensional latent manifold learned of a VAE trained over the MNIST handwritten digits data set. Sampling was performed at linear intervals. Extracted from The Keras Blog [https://blog.keras.io/building-autoencoders-in-keras.html].

Another benefit of this embedding space is that it allows for the interpolation of samples (gradually transforming one sample to another). See Figure 4.

2.5 Generative Adversarial Network (GAN)

A generative adversarial network is another kind of generative model, which was proposed in [Goodfellow et al., 2014]. In this case, two neural network models compete in a zero-sum game. The generator model G tries to learn a meaningful distribution to generate data samples while the discriminator model D is trained to distinguish real samples from those generated with G.

The key idea here is that the improvement of one model also forces the other to be better.

There are some deep clustering techniques that are based on GANs, which can be seen in Section 6.4.

3 Loss functions used in deep clustering

A loss function in a machine learning technique, like the ones described throughout the document, is used as a reference value to measure the goodness of the model. During training, the main objective is to minimize the loss function, which is computed at every training step. Then, an optimizer method is used to find the best way to adjust the model's parameters and reduce the loss as much as we can. This process is usually iterative.

In the case of neural networks, some of the most known optimizers are (Stochastic) Gradient Descent and Adam [Kingma and Ba, 2015], which then use the backpropagation algorithm [Rumelhart et al., 1986] to update the weights.

There are several types of loss functions used in deep learning-based approaches for clustering. We can divide them into two different groups depending on their general purpose: non-clustering and clustering loss functions.

Usually, we find both types of losses combined, as it is discussed later in Section 3.3, but we can also find some approaches that avoid using one of them, in which case special attention must be paid to obtain meaningful

representations and results. Furthermore, some VAE-based and GAN-based algorithms incorporate the two kinds of losses together, with no meaningful division to be made between clustering and non-clustering losses.

3.1 Non-clustering loss functions

The non-clustering loss functions do not depend on the clustering algorithms applied on top of the learned representations. They are aimed at restricting the deep neural network to obtain some desirable properties on the embedding feature space of the data that are agnostic to the clustering method used.

3.1.1 Autoencoder reconstruction loss

In an autoencoder, we have two distinguished subnetworks (an encoder and a decoder) that are connected with a bottleneck of much lower dimensionality. The encoder's function f is to map an input x to a representation z of this lower dimensional latent space Z. While training, the decoder applies a mapping g that tries to reconstruct the same input x from its latent representation z. With that, the autoencoder network can learn a meaningful and compressed representation of the data. After training, the decoder part is not used since the clustering algorithm is applied to these latent representations.

The autoencoder reconstruction loss L_{rec} is defined as some sort of distance between the original input x_i and the reconstructed output $g(f(x_i))$, which normally takes the form of a mean squared error between the two:

$$L_{rec} = \sum_i \|x_i - g(f(x_i))\|^2 \tag{1}$$

The key idea behind this loss function is that reconstruction from z to x is only possible if the learned representation of the data keeps the essential and sufficient information of the original one.

3.1.2 Self-Augmentation Loss

Proposed in [Hu et al., 2017], its goal is to have the representations of the original and their augmented samples close in the learned latent space, thus ensuring that semantically similar inputs have similar representations. This loss L_{SAT} of a particular input can be defined by the equation:

$$L_{SAT} = -\frac{1}{N} \sum_N s(f(x), f(T(x))), \tag{2}$$

where x is the original instance, T is the augmentation function, N is the number of augmented samples, $f(x)$ is the representation in the latent space obtained from the model and s is a similarity function (e.g. cross-entropy).

Significantly, the augmentation function T can be deterministic or stochastic.

This loss is used in the IMSAT algorithm, defined in the same work ((see Section 6.2). Many recent contrastive learning and self-supervised approaches such as [Niu et al., 2020] and [Asano et al., 2020] use this loss or some kind of adaptation of it.

3.1.3 Locality-preserving loss

It was proposed in [Huang et al., 2014] to keep the local information of similar data points in both input X and latent Z spaces.

The locality-preserving loss L_{lp} is formulated as:

$$L_{lp} = \sum_i \sum_{j \in N_k(i)} s(x_i, x_j) \|z_i - z_j\|^2, \tag{3}$$

where $N_k(i)$ is the set of the nearest k neighbors of the data point x_i and $s(x_i, x_j)$ is a similarity measure between the two data points in the input space, and z_i is the latent representation of x_i in the embedding space.

Note that since the term $s(x_i, x_j)$ is fixed, the optimization of the network can only try to minimize the difference between the two latent representations, with higher importance the more similar a neighbor is.

3.1.4 Without non-clustering loss

There is the possibility to just use a clustering loss, without any additional non-clustering loss function. In this case, the network is only constrained by the clustering loss (see Section 3.2), so it has to be defined in such a way that guarantees that the network learns a good representation of the data anyway [Yang et al., 2017].

A network that falls in this category is sometimes called Clustering Deep Neural Network (CDNN) in the literature [Guo et al., 2020], which is a term that is adopted from now on in this document.

3.2 Clustering loss functions

Apart from restricting the representations obtained from the network model via the non-clustering loss functions, which is agnostic of the clustering method used later, it is common to train the whole model by using a clustering loss that is specific to the clustering algorithm.

The main objective of this kind of loss functions is to obtain representations that are clustering-friendly, meaning that the latent representations have some desirable properties for the clustering algorithm to be applied on top, i.e. get useful embeddings for the specific clustering method.

There are multiple loss functions of this type that are commonly used in the literature, which are described in this section.

3.2.1 k-Means loss

Proposed in [Yang et al., 2017], it ensures that the representations data points learned are evenly distributed around the centroids. To achieve that, the neural network must be trained with the following loss function $L_{km}(\theta)$:

$$L_{km}(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} s_{ik} \|z_i - \mu_k\|^2, \tag{4}$$

where z_i is a representation in the latent space of some input x_i, μ_k is a cluster center and s_{ik} is a boolean indicating whether or z_i belongs to the cluster whose centroid is μ_k.

Thus, by using this loss function with respect to the network weights θ the network is minimizing the squared distance between each data point in the embedding space and its associated cluster center.

This loss is interesting when using k-Means or some variation of it, though it might be used with other partition-based clustering algorithms that have the same principles.

3.2.2 Cluster assignment hardening

In this case, the assignments of the data point representations to a cluster must be soft. A possible way to get these soft assignments is to measure the similarity between the data points and the cluster centers by using Student's-t-distribution [Maaten and Hinton, 2008]. The distribution Q can be expressed as the following:

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\nu)^{-\frac{\nu+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\nu)^{-\frac{\nu+1}{2}}}, \tag{5}$$

where z_i is a data point in the latent representation space, μ_j is the j_{th} cluster center and ν is a constant, normally $\nu = 1$.

This distribution is computing a normalized soft assignment between each data point and the centroids. Then, a hardening on these assignments takes place by approaching the Q distribution defined in equation (5) to a target distribution P, defined as follows:

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_{j'}(q_{ij'}^2 / \sum_i q_{ij'})} \tag{6}$$

Note that it is a squared and normalized version of the original Q distribution, which makes this target distribution P to be more polarized. In other words, assignments in the distribution are more close to 0 or 1, i.e. the density mass moves towards the extremes.

The divergence between both probability distributions is used as the cluster assignment hardening loss L_{hard}. The most common way to compute this divergence is the Kullback-Leibler's [Kullback and Leibler, 1951] KL, which is formulated as:

$$L_{hard} = KL(P\|Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{7}$$

Similarly to the k-means loss $L_{k-means}$, the weights θ of the neural network are optimized by minimizing this loss.

3.2.3 Balanced assignments loss

The balanced assignments loss tries to constrain the imbalance on the cluster assignments. It has been used with other loss functions like cluster assignment hardening in some approaches like DEPICT [Dizaji et al., 2017], which is described in Section 6.1.2.

The probability distribution G is defined as the probability of assigning a data point representation to each cluster, i.e. the sum of all probabilities q_{ik} between a representation z_i and a cluster k, divided by the total number of samples N:

$$g_k = P(y = k) = \frac{1}{N} \sum_i q_{ik} \tag{8}$$

The probability distribution G described above can be compared to the uniform distribution U to obtain the balanced assignments loss. Concretely, the loss is computed as the divergence between both probability distributions by using the same Kullback-Leibler divergence:

$$L_{ba} = KL(G\|U) \tag{9}$$

Note that it might not be good to suppose a uniform distribution of the assignments, so a different probability distribution other than U can be used if the prior distribution of the assignments is known.

3.2.4 Other losses

There are several other clustering losses used in the literature like group sparsity loss, cluster classification loss, and agglomerative clustering loss, which are more detailed below.

Group sparsity loss is used in [Xiuyan Ni, 2019], and takes inspiration from spectral clustering, which is explained in [Ng et al., 2001]. It is aimed to perform a feature selection over the latent feature space while training the network.

Cluster classification loss uses the assignments of the clustering algorithm applied as pseudo-ground truth for training the neural network. In [Hsu and Lin, 2017] a mini-batch k-means is used over randomly selected features in the latent space.

Agglomerative clustering loss defines a way to optimize the neural network in which the affinity or similarity between the computed clusters is maximized at every merging step. It is therefore only used with agglomerative clustering techniques, like in JULE [Yang et al., 2016] (see Section 6.2).

3.2.5 Without clustering loss

Usually, a clustering loss is used to restrict the representations that we obtain from the neural network in a way that the posterior clustering algorithm applied can perform better.

However, it is also possible to obtain useful representations for clustering by using only a non-clustering loss (see Section 3.1), although most of the time using a clustering loss results in a better performance [Xie et al., 2016].

3.3 Training schedules

In this subsection, a more detailed explanation of the training of the algorithm is given. Concretely, it is focused on how are the different losses combined into a final metric that can be used to train the system, by means of an optimizer, and the typical schedule approaches of the loss during training.

Naturally, as discussed throughout this section, there are some algorithms that do not have both types of losses (clustering and non-clustering) or have a specially defined loss to combine multiple losses, with the purpose of achieving some useful constraints for their specific approach.

Otherwise, if it utilizes the two kinds of losses, the most common way to combine the loss functions is to use a weighted sum of the clustering and non-clustering losses:

$$L(\theta) = \alpha L_c(\theta) + (1 - \alpha) L_n c(\theta) \tag{10}$$

Here, $L_c(\theta)$ is the clustering loss and $L_n c(\theta)$ is the non-clustering one. The α term is a value between 0 and 1 (both included) that indicates the importance of each of both functions.

Significantly, the hyperparameter α can be tuned for different parts of the training to achieve desirable behaviors and learn in a safer way. The most common schedules of the α hyperparameter are described below, which define usual training approaches found in the literature:

Pre-training and fine-tuning

The hyperparameter is initialized to 0, meaning that the network only uses the non-clustering loss to be trained. In this phase, the network is learning how to get meaningful representations in the latent space for its input samples, by using the reconstruction loss or some other loss to preserve the semantic similarity between samples. So far, no clustering is performed.

Note that we could use an already pre-trained network for that purpose, which can decrease the training time of the network if the domain of both networks is similar.

After this pre-training is complete, the layers of the network that will not be used for clustering can be removed and α is set to 1. Now, the network is only trying to optimize a friendly-clustering representation in the latent space.

It is important to highlight the fact that, with enough training, the network could lose the constraints introduced during the pre-training phase and thus lead to worse embedding features and clustering performance.

Simultaneous training

In this case, the value of α is a constant strictly between 0 and 1, i.e. $\alpha \in (0, 1)$. For instance, in the case $\alpha = 0.5$ both losses will account for the same weight.

Dynamic schedule of α

With a dynamic schedule, α takes a variable value depending on a specific schedule. Typically, it is initialized with a low value and increased towards 1, which results in a smoothened version of the pre-training and fine-tuning schedule explained above.

4 Validation of deep clustering models

Normally in machine learning, the performance of a model is evaluated by comparing the ground truth labels, the real ones, with the ones predicted by the model. This is always the case in supervised learning approaches, where the most common metrics to assess performance are accuracy, precision, recall, and F1-score. These metrics can be macro or micro (i.e. average of all classes or one class only, respectively). Furthermore, other statistics that generate a more complex report, like a confusion matrix, are typically used.

We can separate the metrics used in deep clustering, and clustering in general, into two different groups, depending on the availability of the true labels:

4.1 External metrics

This group of performance metrics uses real labels to evaluate the model. Some of them are adaptations from the supervised evaluation metrics aforementioned.

An example of these adapted metrics would be the unsupervised clustering accuracy, with normalized mutual information and adjusted rand index being also commonly used. These kinds of metrics are usually called external validation metrics since they use outside information to be computed (labels).

Unsupervised Clustering Accuracy (ACC)

The unsupervised clustering accuracy is the unsupervised version of the common classification accuracy. It is defined as:

$$ACC = max_m \frac{\sum_{i=1}^{n} 1\{y_i = m(c_i)\}}{n}, \tag{11}$$

where 1 is the indicator function and m is a mapping function between the assignment output c and the ground truth y for a sample i.

Basically, it is trying to find the best one-to-one mapping of the clustering assignments and the real ones. It can do so because the value of the cluster assignment itself does not have meaning, it is just to indicate what data points are grouped together.

Normalized Mutual Information (NMI)

The Normalized Mutual Information [Estevez et al., 2009] is an information-theoretic metric that uses the mutual information I metric between the ground truth labels Y and the clustering assignment labels C as follows:

$$NMI(Y, C) = \frac{I(Y, C)}{\frac{1}{2}[H(Y) + H(C)]}, \tag{12}$$

where H denotes the entropy. Note that the mutual information I is normalized by the average of the entropy of both Y and C.

It is important to note that most deep clustering techniques proposed use labeled data to evaluate the performance of their new method. The models are still trained in an unsupervised manner, but they can then use the ground truth labels to get more reliable and informative results on the performance of the algorithm.

4.2 Internal metrics

Nonetheless, in most real scenarios the ground truth labels are not available, in which case we are forced to evaluate the performance of deep clustering models by other means.

If the true labels are not given, there are other performance metrics used to get an idea of the goodness of the clustering model or to search for the best value for some hyperparameters like the number of clusters K, which may be also unavailable.

Some examples of these metrics include different indices like Silhouette [Rousseeuw, 1987], Davies-Bouldin [Davies and Bouldin, 1979] and Caliński-Harabasz [Caliński and Harabasz, 1974]. They are referred to as internal validation metrics.

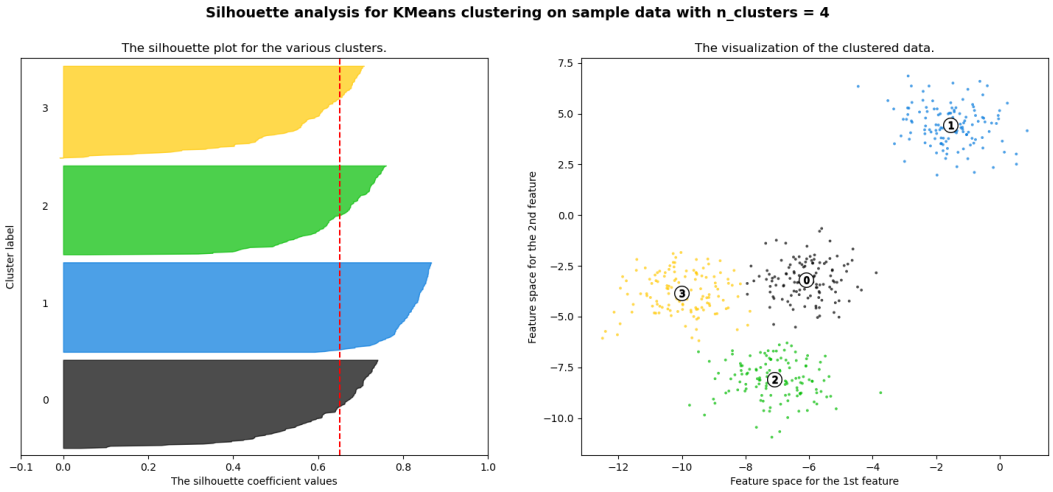

Figure 5: Silhouette analysis for k-means with $K = 4$. The red discontinuous line indicates the mean silhouette score (the higher the better). The silhouette analysis provides a useful insight into the distances between the samples and their assigned cluster centroids. Extracted from the Scikit-learn online documentation [https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html].

The definition and evaluation of performance, in this case, lose some meaning with respect to the labeled data case, but these validations are still necessary to achieve better results. Importantly, without labels, clusters might not have a clear separation and the goal of the clustering is not defined per se (e.g. data can be separated in many different ways, and all are valid). In other words, instead of a classification problem, it is a purely clustering problem.

5 Deep features used for clustering

The deep neural network used to obtain the latent representations of the data is encoding the information at each layer or branch of the encoding subnetwork. Thus, each embedding feature space generated at every layer can be used to represent the data, and not just the output layer of the encoder. Usually, the activation values of neurons at a layer l are used as features.

Depending on the number of layers used to build the representation vector, we can divide the techniques into two different groups: single-layer and multi-layer features, having some pros and cons with each one of them.

Note that the *encoder* mentioned is not just referring to the one used in an autoencoder-based architecture, but rather is loosely referring to the whole subnetwork until a layer l, from which the latent features will be extracted (maybe using some features of layers before l).

5.1 Single-layer features

This is the most common approach for deep clustering. The clustering algorithm uses the latent representation of a single layer, which is usually the output of the encoder subnetwork (that has been referred to as $f(x)$ along the document) to cluster the input data samples.

Some advantages of using a single layer are that the dimensionality of the representation space is lower and all features are presumably more independent than if using multiple layers.

5.2 Multi-layer features

More recently, some approaches were proposed where the representation is the concatenation of features extracted at different layers of the network. In this case, the representation contains more information, which can be used to find more complex patterns and achieve a better performance [Saito and Tan, 2017].

Nonetheless, a disadvantage is that clustering algorithms work better if the dimensionality is low. The densities and distances between points (measures used by the clustering algorithms) become pretty much useless if samples are very scattered. Furthermore, in this concatenated representation of features, they are far from being independent (in a feedforward network the neurons are activated based on the previous layer). Thus, depending on the task the layers used to extract the features should be tuned.

To address the problem of high-dimensionality when using multi-layer features, we can use some dimensionality reduction techniques such as Principal Component Analysis (PCA) or a discretization of the features, similar to the work of [Garcia-Gasulla et al., 2017] where they perform a full-network embedding of a CNN to solve a supervised classification problem.

6 Deep clustering algorithms

In this section, some of the most important and successful deep clustering algorithms are described. In order to have a broader view of the state-of-the-art techniques, they are presented grouped by the core architecture used in their approach: AE-based, CDNN-based, VAE-based, and GAN-based.

Significantly, the primitive architecture (described in Section 2) used in each method have many consequences on different properties of the model: how is it trained, advantages, disadvantages, computational complexity, etc., which are discussed throughout this section.

6.1 Autoencoder-based

These algorithms utilize a combination of the autoencoder reconstruction loss and some clustering loss. They optimize both autoencoder network and clustering parameters to achieve meaningful and clustering-friendly representation features.

One major advantage is that they are easier to implement compared to techniques based on other architectures while avoiding trivial solutions. On the counterpart, the hyper-parameter α must be scheduled and tuned to obtain better results.

These techniques may use a multilayer perceptron or a convolutional neural network as an autoencoder, with the latter especially useful for handling spatial-invariant data like images.

Many approaches add noise to the input to make the model more robust and avoid overfitting, which is referred to as a denoising autoencoder. Furthermore, the autoencoder can use not only the difference between the input and the reconstructed output, but also all the intermediate representations [Dizaji et al., 2017]. For that to make more sense, the autoencoder should have a symmetrical architecture.

Some representative algorithms that are autoencoder-based are Deep Embedding Networks (DEN) [Huang et al., 2014], Deep Subspace Clustering Networks (DSC-Nets) [Ji et al., 2017], and Deep Multi-Manifold Clustering (DMC) [Chen et al., 2017]. Two of the most important are further described as follows:

6.1.1 Deep Clustering Network (DCN)

Deep Clustering Network [Yang et al., 2017] uses a k-means on top of an autoencoder. First, it pre-trains an autoencoder with the reconstruction loss. Then the network is jointly optimized with k-means loss. It is one of the most simple approaches to deep clustering and can yield to good results if the task is not very complex.

6.1.2 Deep Embedded Regularized Clustering (DEPICT)

A more complex algorithm is the Deep Embedded Regularized Clustering (DEPICT), which was proposed in [Dizaji et al., 2017]. It uses a convolutional autoencoder with a softmax layer. It uses a relative entropy loss with a term that enforces a balanced cluster assignment. Therefore, it is good at avoiding outliers to be cluster centroids. It uses all layers for the reconstruction loss and added noise, making it more robust to corrupted latent representations. Still, it is highly efficient while achieving superior results.

6.2 CDNN-based

As mentioned in Section 3.1.4, some algorithms only use a clustering-specific loss, without any non-clustering loss. However, this clustering loss is designed such that it can guide the network's optimization process.

We can separate them between those pre-trained using a supervised approach, using an unsupervised one, and those not pre-trained at all.

Unsupervised pre-trained

In this case, the network is first pre-trained by using an unsupervised representation learning approach. Then the subnetwork responsible for obtaining the latent representations (encoder part of an autoencoder) is trained with the clustering loss.

Some works that used this unsupervised pre-trained method include Deep Non-parametric Clustering (DNC) [Chen, 2015], Deep Embedded Clustering (DEC) [Xie et al., 2016], Discriminatively Boosted Clustering (DBC) [Li et al., 2017].

Supervised pre-trained

For more complex data, the unsupervised pre-training may be insufficient. Thus, it is possible to use big models already pre-trained on large-scale data sets like ImageNet. Some of the most known architectures such as VGG-16, ResNet, and Inception can be utilized to speed up the convergence of the network.

This is the approach that some algorithms like Clustering Convolutional Neural Networks (CCNN) [Hsu and Lin, 2017] use, to deal with image classification tasks.

Non-pre-trained

It is possible to obtain good results even without pre-training the network at all with a sufficiently well-designed clustering loss function and some regularizations and additional techniques.

Some algorithms like Information Maximizing Self-Augmented Training (IMSAT) [Hu et al., 2017], Joint Unsupervised Learning [Yang et al., 2016], and Deep Adaptive Image Clustering (DAC) [Chang et al., 2017] are based on this idea. In the case of IMSAT, it learns a representation space in a self-supervised manner.

6.3 Variational Autoencoder-based

Autoencoder-based and CDNN-based have achieved a lot of improvement with respect to traditional clustering algorithms, but they are designed only for clustering and cannot be extended to other applications like sample generation, which is a limitation.

In contrast, variational autoencoders (VAE) can learn a latent representation of input data with a predefined distribution, which is one of the main reasons why they have been gaining attention in recent years.

Note that generally the prior distribution $p(z)$ of a VAE is assumed as an isotropic gaussian, but in deep clustering, this prior must describe the structure of the clusters. Many algorithms choose a mixture of Gaussians for that.

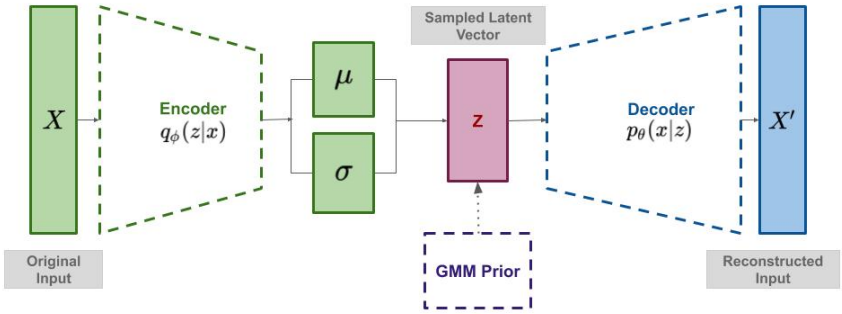

Figure 6: An overview of the Variational Deep Embedding (VaDE) architecture, where an encoder is mapping an input x to the z latent space that assumes a GMM prior. Extracted from deepnotes.io [https://deepnotes.io/deep-clustering].

Some techniques like Variational Deep Embedding (VaDE) [Jiang et al., 2017] (shown in Figure 6) and Gaussian Mixture VAE (GMVAE) [Dilokthanakul et al., 2017] use this kind of architecture, with the latter being more complex and empirically yielding worse results than the former.

6.4 Generative Adversarial Network-based

The Generative Adversarial Network [Goodfellow et al., 2014] (GAN) is another kind of deep generative model that has seen increasing popularity recently.

In it, two neural networks engage in a min-max adversarial game, with a generative model G and a discriminative model D. The two networks are optimized alternatively: first, it tries to maximize the loss of the discriminator D by updating the weights of the generative network G (G tries to fool D); then, it finds the corresponding gradient that minimizes the loss of the discriminator (D becomes better at discriminating real samples from G-generated samples). In this iterative process, the networks improve each other.

Some works that use GAN-based techniques are Deep Adversarial Clustering (DAC) [Warith Harchaoui, 2017], Categorial Generative Adversarial Network (CatGAN) [Springenberg, 2016], and Information Maximizing Generative Adversarial Network (InfoGAN) [Chen et al., 2016].

For InfoGAN, the objective function is defined as:

$$\min_{G,Q} \max_{D} V_{InfoGAN}(D, G, Q) = V(D, G) - \lambda L_I(G, Q), \tag{13}$$

where L_I is the variational lower bound of the mutual information $I(c; G(z, c))$ between c and the output of the generation (which is intractable but can be approximated). A similar approximation is found in some VAE-based algorithms where the objective function is the variational lower bound on the marginal likelihood with some prior distribution.

7 Discussion

Deep clustering is a mix of different fields of deep learning and machine learning clustering, that combines techniques and glances of many areas. By looking at the state-of-the-art of deep clustering, we can see that most of

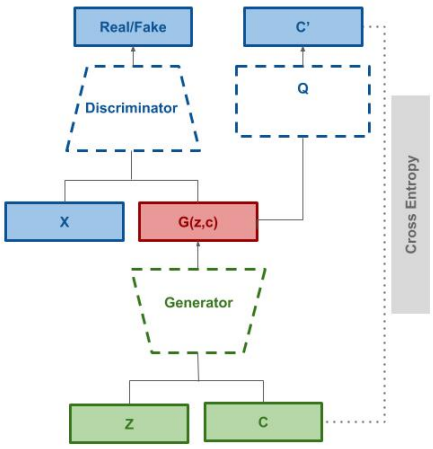

Figure 7: An overview of the Information Maximizing Generative Adversarial Network (InfoGAN) architecture. Input is composed of z, which is treated as incompressible noise, and c, which refers to the latent code. Extracted from deepnotes.io [https://deepnotes.io/deep-clustering].

the approaches are picked, reused, extended, or adapted from this field itself and its broader parent fields aforementioned. This means that many of the advantages and challenges of deep clustering come and depend on the evolution of general clustering techniques.

An overall discussion of the advantages, challenges, and applications of deep clustering techniques can be found in this section. For a more specific discussion of a given term, see its corresponding section.

7.1 Advantages

In deep clustering, the dimensionality of the representations can be reduced successfully without losing the semantic information. This is something intrinsic to these algorithms due to the architecture (bottleneck layer in an autoencoder) or the loss functions used to train the model. With that, the performance of the applied clustering algorithm can be highly improved.

Furthermore, this end-to-end framework allows the feature extraction and clustering assignments optimization to happen simultaneously. They are in a symbiotic environment and not completely independent pieces.

Another advantage is that networks can be scaled properly in most cases, to increase the complexity of the patterns and representations found. These solutions allow for transfer learning of big models, which is something very beneficial to the field.

Probably, the most important advantage is that most of these algorithms are trained in an unsupervised or self-supervised manner, i.e. without the target labels given, but are able to even outperform supervised learning approaches in some cases, which is something encouraging because most of the data in the real world is unlabeled.

7.2 Challenges

Despite its benefits, there are still several challenges that have to be addressed to keep making progress in this field and some related ones.

The first problem is common to all neural networks and is the fact that most of the time, especially with the bigger and bigger models, interpretability is lost. This means that we have difficulties understanding why exactly is the network giving us this specific output and not another. This does not mean the network is not working. In fact, if trained properly they can outperform humans in many tasks, but we are not able to know exactly his "thought process". This lack of interpretability is gaining attention and there are many researchers working on that, which is commonly referred to as explainable AI (XAI).

Another issue arises from the fact that if these models are trained in an unsupervised manner, without any target label to compare to, they might not extrapolate well enough in a real-case scenario. In supervised learning the model is validated many times during the model selection phase, to validate different combinations of hyperparameters and find the best one by some criteria. It is usually even further validated with a test set, which is typically reported as the final performance of the model. Therefore, we can be more certain of good modeling as

long as the real data is from the same (or nearly the same) distribution of data as the one it was used for training. But if labels are not given, like in most of the cases here, we have to rely only on internal metrics to tune our hyper-parameters.

Importantly, most algorithms described suppose that the number of clusters K is known, which is a common assumption not only in deep clustering but in clustering in general. It could be the case, though, where this is unknown and other clustering methods based on density might have to be applied. To my understanding, a semi-supervised or active learning approach could prove very useful in these situations.

7.3 Applications

Deep clustering has widened even more the range of possibilities for applications in many areas. Due to its neural network nature, and the huge amount and yet increasing resources that neural networks and AI, in general, are receiving, we can expect these proposals to only be multiplied.

A deep clustering algorithm, like the ones presented before, can be used to perform any clustering task that has to deal with text processing, image and video processing, knowledge discovery, and data mining, among many others.

They are especially used in big companies that collect a large amount of information from their users, but they are also used in some very specialized companies and sectors where technology must be cutting-edge.

These algorithms can be used for building powerful recommendation systems, market segmentation, image detection and auto-completion, speech source identification, document clustering, etc. In fact, some claims against the exploited use of these types of technologies and for stricter regulation have been increasingly made.

8 Conclusions

An overview of some of the most important deep clustering algorithms in different taxonomies has been presented in this document, which has been made to serve as a starting point for any non-expert person. This taxonomy distinguishes the algorithms in different classes regarding their network architecture, their losses and training phases, and the set of deep features used.

At the beginning of the work, the author did not have any deep knowledge of this field, so assume some errors could be present.

Several references appear throughout the work, so they can be reviewed for more detailed and original explanations of the algorithms and the techniques used.

References

[Asano et al., 2020] Asano, Y. M., Rupprecht, C., and Vedaldi, A. (2020). Self-labelling via simultaneous clustering and representation learning.

[Caliński and Harabasz, 1974] Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27.

[Chang et al., 2017] Chang, J., Wang, L., Meng, G., Xiang, S., and Pan, C. (2017). Deep adaptive image clustering. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5880–5888.

[Chen et al., 2017] Chen, D., Lv, J., and Zhang, Y. (2017). Unsupervised multi-manifold clustering by learning deep representation. In *AAAI Workshops*.

[Chen, 2015] Chen, G. (2015). Deep learning with nonparametric clustering.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[Davies and Bouldin, 1979] Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.

[Dilokthanakul et al., 2017] Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2017). Deep unsupervised clustering with gaussian mixture variational autoencoders.

[Dizaji et al., 2017] Dizaji, K. G., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization.

[Estevez et al., 2009] Estevez, P. A., Tesmer, M., Perez, C. A., and Zurada, J. M. (2009). Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201.

[Garcia-Gasulla et al., 2017] Garcia-Gasulla, D., Vilalta, A., Parés, F., Moreno, J., Ayguadé, E., Labarta, J., Cortés, U., and Suzumura, T. (2017). An out-of-the-box full-network embedding for convolutional neural networks.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial networks. *ArXiv*, abs/1406.2661.

[Guo et al., 2020] Guo, X., Liu, X., Zhu, E., Zhu, X., Li, M., Xu, X., and Yin, J. (2020). Adaptive self-paced deep clustering with data augmentation. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1680–1693.

[Hsu and Lin, 2017] Hsu, C.-C. and Lin, C.-W. (2017). Cnn-based joint clustering and representation learning with feature drift compensation for large-scale image data.

[Hu et al., 2017] Hu, W., Miyato, T., Tokui, S., Matsumoto, E., and Sugiyama, M. (2017). Learning discrete representations via information maximizing self-augmented training.

[Huang et al., 2014] Huang, P., Huang, Y., Wang, W., and Wang, L. (2014). Deep embedding network for clustering. In *2014 22nd International Conference on Pattern Recognition*, pages 1532–1537.

[Ji et al., 2017] Ji, P., Zhang, T., Li, H., Salzmann, M., and Reid, I. (2017). Deep subspace clustering networks.

[Jiang et al., 2017] Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. (2017). Variational deep embedding: An unsupervised and generative approach to clustering.

[Karim et al., 2020] Karim, M. R., Beyan, O., Zappa, A., Costa, I. G., Rebholz-Schuhmann, D., Cochez, M., and Decker, S. (2020). Deep learning-based clustering approaches for bioinformatics. *Briefings in Bioinformatics*, 22(1):393–415.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

[Li et al., 2017] Li, F., Qiao, H., Zhang, B., and Xi, X. (2017). Discriminatively boosted image clustering with fully convolutional auto-encoders.

[Maaten and Hinton, 2008] Maaten, L. V. D. and Hinton, G. E. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

[Ng et al., 2001] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 849–856, Cambridge, MA, USA. MIT Press.

[Niu et al., 2020] Niu, C., Zhang, J., Wang, G., and Liang, J. (2020). Gatcluster: Self-supervised gaussian-attention network for image clustering.

[Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[Saito and Tan, 2017] Saito, S. and Tan, R. T. (2017). Neural clustering: Concatenating layers for better projections.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[Springenberg, 2016] Springenberg, J. T. (2016). Unsupervised and semi-supervised learning with categorical generative adversarial networks.

[Warith Harchaoui, 2017] Warith Harchaoui, Pierre-Alexandre Mattei, C. B. (2017). Deep adversarial gaussian mixture auto-encoder for clustering.

[Xie et al., 2016] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 478–487. JMLR.org.

[Xiuyan Ni, 2019] Xiuyan Ni, Yang Yu, P. W. Y. L. S. N. Q. Q. C. C. (2019). Feature selection for facebook feed ranking system via a group-sparsity-regularized training algorithm.

[Yang et al., 2017] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017). Towards k-means-friendly spaces: Simultaneous deep learning and clustering.

[Yang et al., 2016] Yang, J., Parikh, D., and Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters.