Sepehr Goshayeshi
February 9, 2020
Comprehensive Summary 1

Unlike supervised learning, in unsupervised learning, we are given *unlabeled* data [1]. The training set is in the form of $x^{(m)}$ without labels $y^{(i)}$ in unsupervised learning [1]. The goal is to use an algorithm to find some kind of underlying structure in the unlabeled data [1]. This paper will be structured as such:

I. Clustering Algorithms

K-Means Algorithm

One popular example includes *clustering algorithms*, which is very useful for grouping and organizing similar data points into coherent subsets. One popular clustering algorithm is the *k-means algorithm*.

Prior to beginning the algorithm, one must determine the number of clusters he or she wants to find (K). Then "centroids" (μ_k) are placed at random locations. Afterwards, there are two main steps done iteratively: cluster assignment and move centroid [1]. First for "cluster assignment", each data point (x_i), is assigned to a cluster ($c^{(i)}$) with the closest centroid (μ_k). Second for "move centroid", the mean of all data point is computed in each cluster, and the positions of the centroids (μ_k) are thus recomputed. These two steps are iterated until no data points change cluster memberships.

The optimization objective used is represented below [1]. It is trying to minimize the cost function (J), which is also known as "distortion":

$$J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-\mu_{c^{(i)}}\right\|^2$$

It is getting the squared distance between the data point ($x^{(i)}$) and the location of cluster centroid to which that data point was assigned (μ_c^{i}). During the cluster assignment step, the distortion function of the clusters ($c^{(m)}$) to which the data points are assigned is minimized while centroids are fixed. Moreover, in the "move centroid" step, the distortion function of the cluster centroid (μ_k) is minimized.

In the beginning, it is a good idea to have random initialization of number of centroids done many times in order to find good local optima [1]. The "elbow method" can sometimes help decide the number of centroids. It involves picking the number of centroids after which the reduction in the "within-cluster sum of squares" (WCSS) is computed as irrelevant [1-2]:

$$WCSS = \sum_{i=1}^{m} \left\| x^{(i)} - \mu_{c^{(i)}} \right\|^2$$

When this is plotted with cost function on vertical axis and number of clusters on x-axis, the "elbow" is the optimal number of centroids [2]. Also, one must decide on the number of clusters for his or her purposes.

K-Means clustering has many applications. Examples include: database of shop items can be grouped into different market segments (e.g., type or size of Iphone case), network analysis of people on social media platforms (e.g., based on likes), scientific data analysis (e.g., genetics and astronomy), organizing computing clusters, and so forth [1].

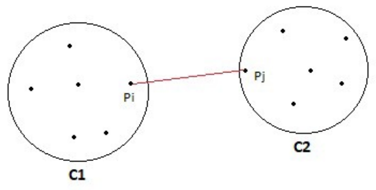
K-Means is easy to implement and fast, but it only works when there's no overlapping data or significant outliers. It also depends on one's initial choice of clusters and such, as previously explained.

Hierarchical Clustering

Another clustering algorithm is *hierarchical clustering* [3]. Hierarchical clustering has two techniques: agglomerative and divisive. *Agglomerative* starts from each data point as a separate cluster and aggregates that data by merging the most similar clusters until all data points merge into a single cluster whereas *divisive* begins from one cluster, containing the whole dataset, and recursively splits it until only clusters of individual data points remain. The data is either aggregated or split based on similarity, which refers to Euclidean distance. Hierarchical clustering is represented via a "dendogram", which is a tree-like diagram recording sequences of merges or splits [4]. It is good for purposes such as biological taxonomy, hierarchical visual cortical processing, or anything involving a strict hierarchy [3]. Unlike k-means, one advantage of hierarchical clustering is we do not need to specify the number of clusters, but one major disadvantage of hierarchical clustering is that because there is no objective function there is a higher time complexity.

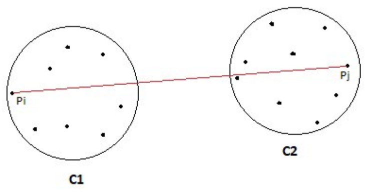
Similarity between two clusters is measured for merging or dividing clusters. There are five different approaches for this one can pick from: MIN, MAX, Group Average, Ward's method, and distance between centroids[4].

MIN involves picking the two closest points in Cluster 1 (C1) and Cluster 2 (C2) [3-4]. Not good for noise between clusters [4].



Source: [4]

MAX involves picking the two farthest points in C1 and C2 [4]. MAX is good for noise between clusters, but it is biased towards globular clusters and tends to break large clusters [4].



Source: [4]

Group average involves taking all pairs of points and calculating their average similarities. Ward's method is similar to group average, but it involves calculating the sum of the square of distances. Distance between centroids is basically computing centroids in clusters and then determining similarity.

II. Dimensionality Reduction Algorithms

Principle Component Analysis (PCA)

Another example of unsupervised learning problem is *dimensionality reduction* [1]. It is good to do data compression in order to reduce redundant or highly correlated features, which can help learning algorithms become more efficient. For example, consider the feature x_1 being length in cm and x_2 as length in inches, and we can reduce the data from two dimensions into a new one dimensional feature z_1. Likewise, one can also reduce 3D data as 2D if the former can be projected as a 2D-plane. Higher dimensional datasets can also be reduced to lower-dimensions. Dimensionality reduction is also good for *visualizing data* too. For example, one could have a dataset of many countries with 50 features such as GDP, per capita GDP, HDI, life expectancy, poverty index, mean household income, and more [1]. Then data is reduced to two features like z_1 and z_2, and then you can plot them in a 2D graph. The z_1 horizontal axis roughly corresponds to the overall country size/GDP whereas the vertical axis is the per person GDP or economic activity, and this can provide some general assessment of various countries based on these two reduced features [1].

The most popular algorithm for dimensionality reduction is the principal components analysis (PCA) [1]. It tries to finds the direction of lower dimensional surface or vector by which the data can be projected in order to minimize the average squared projection error (i.e., orthogonal distance between the points $x^{(i)}$ and the projection, $x^{(i)}_{approx}$) [1]:

$$\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x^{(i)}_{approx}\right\|$$

PCA differs from linear regression, even though they may look cosmetically similar. In linear regression, we are trying to predict y from the features of x. However, in PCAs, there is no value y which we're trying to predict. Instead, we have a bunch of features which are treated equally.

Before implementing PCAs to reduce n-dimensions to k-dimension, it is always necessary to perform preprocessing and standardization such as mean normalization and feature scaling on the unlabeled data. Afterwards, the procedure to reduce the n dimensions to k-dimensions, involves first computing the covariance matrix, sigma [1]:

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$$

From computing sigma, which is a nxn matrix, you then obtain the eigenvectors and eigenvalues [5]. To quote plot.ly [5], the next step involves "sorting eigenvalues in descending order and choosing the k eigenvectors that correspond to the k largest eigenvalues where k is the number of dimensions of the new feature subspace (k≤n)". A projection matrix is now made from the selected k eigenvectors, and you can transform the original dataset via the projection matrix in order to reduce the original n-dimensional matrix to k-dimensions [5]. One can also reconstruct the original high-dimensional data from the compressed representation [1].

Typically, the number of principal components is decided by dividing the average squared projection error and total variation in the data while ensuring 99% of the variance is retained [1]:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01$$

PCAs are very useful for reducing memory usage and also speeding up supervised learning [1]. They are also used to visualize and find patterns in financial, bioinformatic, and psychological data [1]. However, PCAs are bad to use for preventing overfitting. Regularization is preferable for preventing overfitting as explained in class, and PCAs have frequently been misused for preventing overfitting [1].

Autoencoders

Another unsupervised learning algorithm useful for dimensionality reduction are *autoencoders* [5-6]. They are a kind of feedforward neural architecture [6-7]. It takes in input data, such as an image, with a high dimensionality and compresses the data after running it through the network [6-7]. There are two main components to autoencoders, which both involve a bunch of hidden fully connected or convolutional layers: the encoder and decoder. The encoder takes an input image and compresses it down to less dimensions, mapping it to a latent space or fixed vector, which is called the "bottleneck", and then the decoder transforms it into an output with the same dimensions [6-7]. Training the autoencoder involves looking at the reconstructed, decoded version pixel-by-pixel at the end of the decoder network, and it gives a reconstruction loss with respect to the input. It is learning hidden or latent representations of the data through this optimization process. Autoencoders can be used in Convolutional Neural Networks too [6-7]. Self-driving cars also use them [6-7]. One can also train denoising variants of autoencoders to remove noise [6-7]. One example of reconstruction loss used is mean squared error, $J(x,z)=\sum\|x-z\|^2$, where z is the reconstructed input and x the original input, and backpropagation is done to update weights to minimize reconstruction loss during training [7]. Cross entropy loss is also popular [7].

Variational Autoencoders (VAEs) have become more popular due to being more robust [8, p. 299]. VAEs do not compress the input image as fixed code into the latent space. Instead, the image is mapped into a statistical distribution of mean and variance [8, p. 299]. The decoder takes one point that's randomly sampled from the distribution and reconstructs it [8, p. 299].

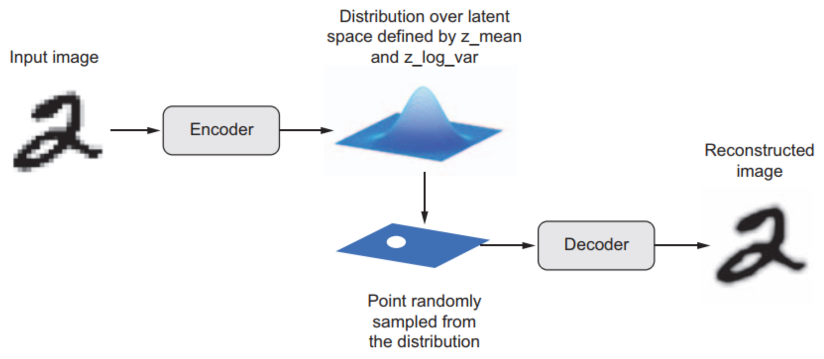

Input image
Distribution over latent space defined by z_mean and z_log_var
Encoder
Reconstructed image
Decoder
Point randomly sampled from the distribution

Figure 8.13 **A VAE maps an image to two vectors, `z_mean` and `z_log_sigma`, which define a probability distribution over the latent space, used to sample a latent point to decode.**

Source: [8, p. 299]

The loss function of the VAE can expressed as such:

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big)$$ *Source [9]*

The first part of this equation with the log is the loss function while the second part helps to stay close to a normally distributed Gaussian. Regularization loss helps to prevent overfitting also [8].

Much like PCAs, autoencoders are good for speeding up supervised learning due to dimensionality reduction, but unlike PCAs, autoencoders can also be used for regularization of neural networks [10]. VAEs are also good for image generation, such as generating never before seen faces and even altering smile vectors [8]. They are good for image editing like swapping faces, turning a frown to a smile, and so forth [8]. They can work with animations [8]. Moreover, VAEs have been used for discovering drug discovery and decoding fMRI BOLD signals in the visual cortex [11-12].

Generative Adversarial Networks (GANs)

GANs were introduced by Goodfellow et al. [13] in 2014. They are alternatives to VAE for learning an image's latent space and generation of realistic images [8, p. 305]. It involves a forger and expert network *competing* to best one another [8, p. 305]. For example, imagine a forger of paintings working together with an expert who spots fakes. The forger provides both real and fake paintings for the expert to distinguish. The forger's competency grows while the expert becomes even better at distinguishing fakes from the real ones. Eventually, the forger's and expert reach an equilibrium where the latter can no longer distinguish fake from real! This principle of competition between forger and expert drives how GANs work.

There are two parts to GANs: Generator and Discriminator network. The generator network takes the input as a random network to decode it into a synthetic image whereas the discriminator network takes either the real or synthetic image to predict whether it came from the training set or was synthetically generated [8, p. 305]. This is analogous to the forger and expert painting example.

What makes GAN interesting is it does not use gradient descent as optimization function [8, p. 306]. Instead, optimization involves seeking an equilibrium between the two networks, and for this reason training a GAN is very difficult [8, p. 306]. The difficulty of training is one huge disadvantage.

The tricks of optimizing a GAN are very difficult [8, p. 307]. What I noticed is GANs are not included in the class syllabus, which I surmise is due to their difficulty. For the loss function, Goodfellow et al. [13] initially used minimax loss function:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$ *Source: [14]*

D(x) is the estimated probability that the real data is real while D(G(z)) is the estimated probability that the fake image is real [14]. E_x is the expected value for the real data instances whereas E_z is the expected value for fake data estimates [14]. The goal is to minimize log(1-D(G(z))) [14]. Other modifications involve maximizing log(D(x)) [14].

References

[1] A. Ng, “Coursera Machine Learning: Week 8 – Unsupervised Learning,” *Coursera*. [Online]. Available: https://www.coursera.org/learn/machine-learning/home/week/8. [Accessed: 02-Feb-2020].

[2] V. Alto, “Unsupervised Learning: K-means vs Hierarchical Clustering,” *Towards Data Science*, 08-Jul-2019. [Online]. Available: https://towardsdatascience.com/unsupervised-learning-k-means-vs-hierarchical-clustering-5fe2da7c9554. [Accessed: 02-Feb-2020].

[3] S. Ullman, T. Poggio, D. Harari, D. Zysman, and D. Seibert, “Unsupervised learning: Clustering,” *MIT*, 2014. [Online]. Available: http://www.mit.edu/~9.54/fall14/slides/Class13.pdf. [Accessed: 02-Feb-2020].

[4] C. Reddy, “Understanding the concept of Hierarchical clustering Technique,” *Towards Data Science*, 10-Dec-2018. [Online]. Available: https://towardsdatascience.com/understanding-the- concept-of-hierarchical-clustering-technique-c6e8243758ec. [Accessed: 02-Feb-2020].

[5] “Principal Component Analysis in Python/v3,” *Plot.ly*. [Online]. Available: https://plot.ly/python/v3/ipython-notebooks/principal-component-analysis/. [Accessed: 02-Feb-2020].

[6] K. Srinivasan, “Guide to Autoencoders,” *Yale Data Science*, 29-Oct-2016. [Online]. Available: https://yaledatascience.github.io/2016/10/29/autoencoders.html. [Accessed: 05-Feb-2020].

[7] Arxiv Insights, “Variational Autoencoders,” *Youtube*, 25-Feb-2018. [Online Video]. Available: https://youtu.be/9zKuYvjFFS8. [Accessed: 05-Feb-2020].

[8] C. François, *Deep Learning with Python*. Shelter Island, NY: Manning Publications Co., 2018.

[9] C. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, “Understanding disentangling in β-VAE,” *Cornell University*, 10-Apr-2018. [Online]. Available: https://arxiv.org/abs/1804.03599. [Accessed: 05-Feb-2020].

[10] C. Will, "How to do Regularization of a Neural Network with an Autoencoder,” *Principles of Deep Learning*, 20-Aug-2018. [Online]. https://principlesofdeeplearning.com/index.php/2018/08/20/how-regularization-of-a-neural-network-can-be-done-with-an-auto-encoder/. [Accessed: 09-Feb-2020].

[11] A. Zhavoronkov, Y. A. Ivanenkov, A. Aliper, M. S. Veselov, V. A. Aladinskiy, A. V. Aladinskaya, V. A. Terentiev, D. A. Polykovskiy, M. D. Kuznetsov, A. Asadulaev, Y. Volkov, A. Zholus, R. R. Shayakhmetov, A. Zhebrak, L. I. Minaeva, B. A. Zagribelnyy, L. H. Lee, R. Soll, D. Madge, L. Xing, T. Guo, and A. Aspuru-Guzik, “Deep learning enables rapid identification of potent DDR1 kinase inhibitors,” *Nature Biotechnology*, vol. 37, no. 9, pp. 1038–1040, 2019.

[12] K. Han, H. Wen, J. Shi, K.-H. Lu, Y. Zhang, D. Fu, and Z. Liu, “Variational autoencoder: An unsupervised model for encoding and decoding fMRI activity in visual cortex,” *NeuroImage*, vol. 198, pp. 125–136, 2019.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Networks,” *arXiv*, 10-Jun-2018. [Online]. Available: https://arxiv.org/abs/1406.2661. [Accessed: 07-Feb-2020].

[14] “Loss Functions | Generative Adversarial Networks | Google Developers,” Google. [Online]. Available: https://developers.google.com/machine-learning/gan/loss. [Accessed: 07-Feb-2020].