Deep Learning: From Zero to Hero

Tal Ben-Haim

June 22, 2023

Machine learning, deep learning, and artificial intelligence (AI) are rapidly evolving fields that has transformed the way we live, work, and interact with technology. It has become an integral part of many industries, including healthcare, finance, transportation, and more. As the world becomes increasingly data-driven, the ability to extract insights from large datasets has become critical for businesses and individuals alike. In this book, we explore the fundamentals of machine learning, from the basics of data analysis and modeling to the latest advancements in artificial intelligence. Through practical examples and real-world applications, we provide a comprehensive overview of this exciting and dynamic field.

Disclaimer: This document will inevitably contain some mistakes— both simple typos and legitimate errors. Keep in mind that part of the notes are in the process of learning the material, so take what you read with a grain of salt.

Contributions to the book are highly encouraged. If you notice any errors, have suggestions for improvements, or would like to contribute additional sections or chapters, please feel free to submit pull requests or open issues in the GitHub repository and I will provide you with the latex source code. You can also reach me by email, in English or Hebrew, at talbenha@gmail.com.

Tal Ben-Haim,
Last Update: June 22, 2023,

Contents

Chpater 1: Signal Processing

1.1 Introduction

Machine learning and signal processing are two closely related fields that have become increasingly important in recent years. Machine learning is the process of teaching computers to learn from data, without being explicitly programmed. Signal processing, on the other hand, is the analysis and manipulation of signals, which can include anything from sound waves to images.

Despite their differences, machine learning and signal processing share a number of fundamental concepts and techniques. In particular, many of the algorithms used in machine learning are based on statistical methods that have their roots in signal processing. For example, principal component analysis (PCA), which is commonly used in machine learning for feature extraction, is based on the spectral analysis of signals (Eigenvalue Decomposition of the covariance matrix).

Conversely, many of the techniques used in signal processing can be seen as a form of learning, in which the goal is to extract meaningful features from data without explicit supervision. For example, convolutional neural networks (CNNs) are a type of deep learning model that rely on convolutional operations to extract features from input signals, making them well-suited for a wide range of signal processing problems. By automating feature extraction and leveraging the scalability of deep learning, CNNs have enabled state-of-the-art performance on tasks including images, speech signals, and texts, demonstrating the powerful connection between machine learning and signal processing.

1.2 Fourier Analysis

Fourier analysis is a mathematical technique used to represent a signal in terms of its frequency components. The basic idea behind Fourier analysis is that any complex signal can be decomposed into a sum of simple sinusoidal functions of different frequencies. This can be useful for a variety of signal processing applications.

1.2.1 Fourier Series

The Fourier series is a mathematical technique that allows us to represent a periodic signal as a sum of sinusoids of different frequencies. The Fourier series of a signal $x(t)$ with period T is given by:

$$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi nt/T} \tag{1}$$

where c_n are the Fourier coefficients, given by:

$$c_n = \frac{1}{T}\int_{t_0}^{t_0+T} x(t)e^{-j2\pi nt/T}dt \tag{2}$$

where t_0 is any point in the interval $[0, T]$.

The Fourier series allows us to represent a periodic signal as a sum of sinusoids of different frequencies, where each sinusoid has a frequency that is an integer multiple of the fundamental frequency $f_0 = 1/T$. The Fourier series is particularly useful for analyzing periodic signals, such as those found in electrical engineering and communications.

1.2.2 Fourier Transform (Continuous-Time Signal)

The Fourier series can be extended to non-periodic signals by using the Fourier transform. In this case, the Fourier coefficients are replaced by the Fourier transform that allows us to represent a non-periodic signal as a sum of sinusoids of different frequencies, where each sinusoid has a continuous frequency. The Fourier transform of a signal $x(t)$ is defined as:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{3}$$

where ω is the frequency in radians per second, and j is the imaginary unit. The Fourier transform takes a continuous-time signal in the time domain and converts it into a representation in the frequency domain.

The inverse Fourier transform allows us to convert a frequency-domain representation back into the time domain. The inverse Fourier transform of $X(\omega)$ is given by:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega \tag{4}$$

The Fourier transform is particularly useful for analyzing signals with continuous frequency content, such as analog signals. It allows us to decompose a signal into its constituent frequencies, making it easier to analyze and process. However, the Fourier transform cannot be directly applied to discrete-time signals, such as those sampled from a digital system.

1.2.3 Discrete Fourier Transform (DFT)

For discrete-time signals, we use the discrete Fourier transform (DFT) instead. The DFT is a digital implementation of the Fourier transform that can be used to compute the frequency content of a discrete-time signal. The DFT is closely related to the Fourier series, which allows us to represent a periodic signal as a sum of sinusoids of different frequencies. The DFT is a commonly used technique for analyzing the frequency content of discrete-time signals. The DFT of a sequence $x[n]$ with N samples is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{5}$$

where $X[k]$ is the kth complex DFT coefficient. The inverse DFT (IDFT) can be used to recover the original sequence $x[n]$ from its DFT coefficients:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N} \tag{6}$$

The DFT is commonly used in applications such as audio and image compression, as well as in digital signal processing for filtering and feature extraction.

1.2.4 Short-Time Fourier Transform (STFT)

The short-time Fourier transform (STFT) is a technique used to analyze the frequency content of signals that vary over time. The STFT is computed by applying the Fourier transform to short segments of the signal, known as windows, and then sliding the window along the signal to compute the frequency content at different time points. The STFT of a signal $x[n]$ can be computed as follows:

$$X(m,k) = \sum_{n=0}^{N-1} x[n+mH]w[n]e^{-j2\pi kn/N} \tag{7}$$

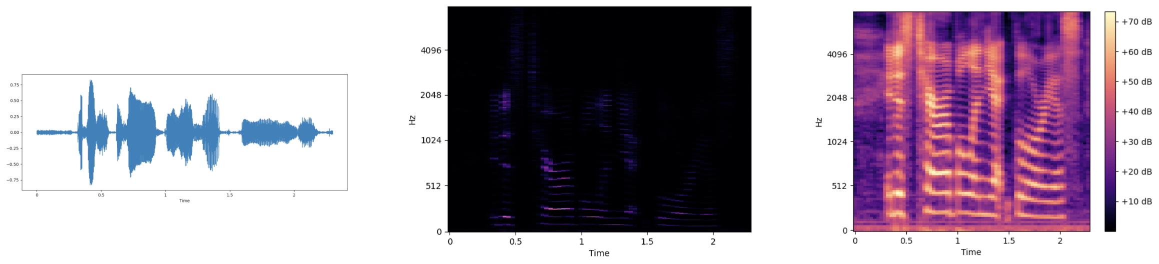
where $X(m,k)$ is the STFT coefficient at time m and frequency k, H is the hop size (i.e., the number of samples between adjacent windows), and $w[n]$ is the window function.

The inverse STFT (ISTFT) can be used to recover the original signal from its STFT coefficients:

$$x[n] = \frac{1}{N}\sum_{m=0}^{M-1}\sum_{k=0}^{N-1} X(m,k)w[n-mH]e^{j2\pi kn/N} = \frac{1}{N}\sum_{m=0}^{M-1} x_m[n-mH] \tag{8}$$

where $x_m[n]$ is the m-th segment of the signal, and $w[n]$ is the window function. The STFT is commonly used in applications such as speech and music processing, where the frequency content of a signal can change over time.

Time-frequency analysis. Time-frequency analysis enables the exploration of dynamic signals, facilitating audio tasks such as sound recognition, speech processing, and music analysis. It is a powerful approach that combines information from both the time and frequency domains to provide a more comprehensive understanding of signals. The STFT divides a signal into small overlapping segments and applies the Fourier transform to each segment, revealing the frequency content at different time intervals. This allows for the examination of how the signal's frequency components change over time. Another important technique in time-frequency analysis is the Mel spectrogram, which is derived from the STFT. The Mel spectrogram applies a non-linear transformation to the STFT magnitude spectrum, emphasizing perceptually important frequency bands while reducing sensitivity to absolute frequency. The Mel spectrogram is commonly used in applications like speech and audio processing, providing a more intuitive representation of the signal's frequency content that aligns with human auditory perception. Figure 1 showcases the raw samples of an audio signal containing the sentence "It is terrible and yet glorious.", along with its corresponding Short-Time Fourier Transform (STFT) and Mel spectrogram. The STFT provides insight into the frequency content of the signal at different time intervals, while the Mel spectrogram offers a perceptually meaningful representation of the signal's frequency content. Together, these visualizations illustrate the application of time-frequency analysis techniques to analyze the given audio signal.



(a) Raw Audio Signal (b) STFT (c) Mel Spectrogram

Figure 1: (a) The raw samples of an audio signal containing the sentence "It is terrible and yet glorious,", along with its STFT (b) and Mel Spectrogram (c).

1.2.5 Fast Fourier Transform (FFT)

The fast Fourier transform (FFT) is an efficient algorithm for computing the DFT of a sequence. The FFT takes advantage of the symmetry properties of the DFT to reduce the number of computations required. Specifically, the FFT computes the DFT of a sequence with length $N = 2^p$ in $O(N \log N)$ operations, compared to the $O(N^2)$ operations required for the standard DFT algorithm.

The FFT is commonly used in applications such as audio and image processing, where large sequences need to be analyzed quickly. The FFT can also be used in conjunction with the STFT to compute the frequency content of a signal over time

1.3 Filtering and Convolution

Filtering is the process of removing unwanted noise or features from a signal while preserving important information. This is often accomplished by convolving the signal with a filter kernel. The filter kernel, also known as the impulse response, specifies how much each input value contributes to the output of the filter. In this section, we will discuss convolution, a fundamental operation in signal processing, and filter design.

1.3.1 Convolution of Continuous Signals

The convolution of two continuous signals $f(t)$ and $g(t)$ is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{9}$$

This can be thought of as a sliding dot product between the two signals, with the dot product taken at each time point t. The resulting output signal is typically of length $2N-1$, where N is the length of the input signals.

1.3.2 Convolution of Discrete Signals

For discrete signals, the convolution can be defined as:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{10}$$

This is similar to the continuous case, but with the integral replaced by a sum.

1.3.3 Convolution using Matrix Multiplication

Convolution can also be formulated using linear algebra by representing signals as vectors and the convolution operation as a matrix-vector multiplication. Let $\mathbf{f} = [f_1, f_2, \ldots, f_n]^T$ be the input signal and $\mathbf{g} = [g_1, g_2, \ldots, g_m]^T$ be the filter kernel. We can construct a convolution matrix $\mathbf{A}$ such that the output signal $\mathbf{y} = [y_1, y_2, \ldots, y_{n+m-1}]^T$ is given by:

$$\mathbf{y} = \mathbf{A}\mathbf{f} \tag{11}$$

where $\mathbf{A}$ is an $(n+m-1) \times n$ Toeplitz matrix defined as:

$$\mathbf{A} = \begin{bmatrix} g_1 & 0 & 0 & \cdots & 0 \\ g_2 & g_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_m & g_{m-1} & g_{m-2} & \cdots & g_1 \\ 0 & g_m & g_{m-1} & \cdots & g_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & g_m \end{bmatrix} \tag{12}$$

Each row of $\mathbf{A}$ represents a shifted version of the filter kernel $\mathbf{g}$, and the matrix-vector multiplication essentially computes the dot product between the shifted filter kernel and the input signal for each shift. Note that the resulting output signal $\mathbf{y}$ has a length of $n+m-1$ to account for the potential overlap at the boundaries of the convolution.

This formulation can be useful in cases where it is computationally expensive to compute the Fourier transform, such as when dealing with very large signals.

1.3.4 Fourier transform and Convolution

The Fourier transform and convolution are closely related concepts. In fact, the convolution theorem states that the Fourier transform of a convolution of two signals is equal to the pointwise product of their individual Fourier transforms. Mathematically, this can be written as:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \tag{13}$$

where $\mathcal{F}\{\cdot\}$ denotes the Fourier transform operator, and f and g are the two signals being convolved. The pointwise multiplication in the frequency domain is equivalent to the convolution in the time domain.

This property is particularly useful because it allows us to perform convolutions more efficiently by taking advantage of the fast Fourier transform (FFT) algorithm. Instead of directly computing the convolution in the time domain, we can first compute the Fourier transform of the signals, multiply them in the frequency domain, and then inverse transform the result back to the time domain to obtain the convolution. The FFT algorithm is much faster than the direct computation of convolution, especially for large signals, and can significantly speed up many signal processing tasks.

1.3.5 Filter Design

Filter design is the process of designing a filter kernel that can be used to filter a signal. There are many different types of filters, each with its own design criteria and trade-offs. Some common types of filters include low-pass, high-pass, band-pass, and band-stop filters.

A low-pass filter, for example, allows low-frequency signals to pass through while attenuating high-frequency signals. The filter kernel for a low-pass filter can be designed using a windowing method, where a window function is applied to a sinc function. The resulting kernel is then scaled and shifted to obtain the desired frequency response.

1.3.6 Cross-Correlation

Cross-correlation is closely related to convolution, and can be thought of as a measure of similarity between two signals. Given two signals f and g, their cross-correlation is defined as:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau) g(\tau + t) d\tau \tag{14}$$

where $\star$ denotes the cross-correlation operation. Essentially, this operation slides one of the signals (in this case, g) along the other signal (in this case, f), and computes the overlap between the two signals at each point.

If we consider the case where one of the signals is flipped in time, the cross-correlation operation becomes equivalent to a convolution:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau = (f \star g(-t))(t) \tag{15}$$

where $*$ denotes the convolution operation. In this case, the signal g is flipped in time, which changes the sign of the time variable in the argument of the integral.

Cross-correlation is often used in signal processing to measure the similarity between two signals, and one important application of cross-correlation is in matched filtering.

Matched filtering is a technique used to detect a particular signal (called the "template" signal) in the presence of noise. The idea behind matched filtering is to correlate the incoming signal with a template signal that represents the expected pattern of the desired signal.

The cross-correlation function between the template signal $h(t)$ and the input signal $x(t)$ is given by:

$$y(t) = \int_{-\infty}^{\infty} \bar{h}(\tau) x(t + \tau) d\tau \tag{16}$$

where $\bar{h}(\tau)$ denotes the complex conjugate of $h(\tau)$.

This operation is also called the matched filter operation, because the filter is matched to the template signal.

The matched filter output $y(t)$ can be interpreted as a measure of the similarity between the template signal and the input signal at each point in time. When the input signal contains the desired signal, the matched filter output will be high at the time when the template signal matches the desired signal. By comparing the matched filter output to a threshold value, we can detect the presence of the desired signal in the input signal.

Matched filtering is widely used in various applications, such as radar signal processing, image processing, and communications.

Chpater 2: Machine Learning Fundamental

2.1 Nearest Neighbor

The Nearest Neighbor algorithm is a type of machine learning algorithm that makes predictions based on the closest training examples in a dataset. It is a type of instance-based learning, which means that it does not explicitly learn a model, but instead stores and memorizes the training data to make predictions.

Nearest Neighbor Classifier. In the Nearest Neighbor classifier, when a new input is given, the algorithm compares it to all the existing data points in the training set and identifies the closest one(s) based on some similarity metric, such as Euclidean distance or cosine similarity. The output label for the new input is then determined by a majority vote from the labels of the closest data points. If multiple nearest neighbors are utilized in the algorithm, it is referred to as K-Nearest Neighbor (K-NN).

Pros and Cons. One benefit of using the Nearest Neighbor classifier is its ability to handle intricate decision boundaries and non-linear input-output relationships. However, it can be susceptible to noisy data and high-dimensional feature spaces, and may require extensive preprocessing and feature selection for optimal performance. Additionally, unlike other classifiers, there is no explicit training phase, but computational costs are incurred during testing.

When comparing the distance between two images, the Nearest Neighbor classifier can be affected by the number of pixels and may not capture spatial information accurately (can be normalized by number of pixels). For instance, if an image is cyclically shifted by one pixel, comparing the original and shifted images may result in a high L1 and L2 distance. Therefore, an embedded representation is required to mitigate these issues.

2.2 Dimensionality Reduction

High-dimensional data can be difficult to visualize and analyze, and can lead to overfitting and slow computation times in machine learning algorithms. Dimensionality reduction techniques aim to overcome these challenges by reducing the number of features in the data while preserving the most important information.

2.2.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a feature extraction and dimensionality reduction technique that can be used in both supervised and unsupervised settings. In unsupervised settings, PCA is used to find the underlying structure of the data, while in supervised settings, it can be used to reduce the number of features in the data while preserving the most important information for the given classification task.

PCA works by finding the principal components of the data that explain the most variance in the data. The principal components are a set of orthogonal vectors that represent the direction of maximum variance in the data. The first principal component corresponds to the direction with the largest variance, and each subsequent component corresponds to the direction with the next largest variance, subject to the constraint that it is orthogonal to the previous components.

The principal components can be computed using the eigenvalue decomposition of the covariance matrix of the data:

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T \tag{17}$$

where Σ is the covariance matrix, n is the number of data points, x_i is the ith data point, and $\bar{x}$ is the mean of the data. The eigenvectors of the covariance matrix correspond to the principal components, and the eigenvalues represent the amount of variance explained by each principal component. The principal components can be used to project the data onto a lower-dimensional space while preserving the most important information in the data.

2.2.2 Linear Discriminant Analysis (LDA)

LDA is a **supervised** method that aims to find a linear combination of features that maximizes the separation between classes. The idea is to find a projection of the data onto a lower-dimensional space that maximizes the ratio of between-class variance to within-class variance. The between-class variance measures the distance between the means of different classes, while the within-class variance measures the spread of the data within each class. LDA can be used for feature extraction and dimensionality reduction, but it is also commonly used as a classification algorithm.

The projection that maximizes the ratio of between-class variance to within-class variance can be computed using the generalized eigenvalue problem:

$$S_W^{-1} S_B w = \lambda w \tag{18}$$

where S_W is the within-class scatter matrix, S_B is the between-class scatter matrix, w is the projection vector, and λ is the eigenvalue.

2.2.3 Comparison Between PCA and LDA.

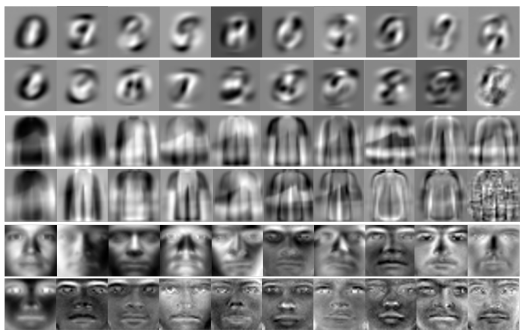
PCA and LDA are both linear transformation techniques used for dimensionality reduction, but they have different objectives and assumptions. PCA is an unsupervised method that aims to find the principal components that explain the most variance in the data, while LDA is a supervised method that aims to find a projection of the data that maximizes the separation between classes. PCA can be used for feature extraction and dimensionality reduction in both supervised and unsupervised settings, while LDA is primarily used for supervised classification tasks.

PCA is a simple and computationally efficient method that can handle high-dimensional data, but it may not be effective if the data has a nonlinear structure or if the relevant information is concentrated in a small subset of the dimensions. LDA can be a powerful method for classification tasks, but it requires labeled data and may not be effective if the classes are highly overlapping or if the within-class variance is large. Overall, both PCA and LDA are valuable tools for dimensionality reduction and feature extraction, and their choice depends on the specific problem and data at hand.

We conducted a comparative analysis between Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) using three distinct datasets: MNIST [1], Fashion-MNIST [2], and Yale Face Database B [3]. Figure 2 illustrates this comparison. Additionally, we visualized the column vectors of the projection matrix in Figure 2a, and presented the re-projection results for various images in both methods in Figure 2b. The subsequent analysis discusses the obtained results. For a more comprehensive comparison, please refer to [**?**].

Visualization of Projections. Gaining insights into the preservation objectives of these methods, we visually examined the column vectors of the projection matrix. In Figure 2a, the white regions in the column vectors denote areas containing critical information targeted by the methods. For instance, in the Yale B dataset, the first projection vector of LDA (Figure 2a, bottom left image) captures features such as the mouth, eyes, and their surrounding regions. In the case of MNIST, PCA performs better in preserving the shapes of the numbers. In the FMNIST dataset, both PCA and LDA projection columns reveal a focus on clothing forms. Regarding the illustrations in Yale B (Figure 2a, last two rows), the PCA method learns the illuminations present in the dataset, whereas LDA disregards them.

Reconstruction. In terms of image reconstruction, PCA outperforms LDA both visually and quantitatively [**?**]. Visual inspection, as shown in Figure 2b, confirms the superior quality of PCA's reconstructed images compared to LDA. PCA effectively preserves important image features, showcasing its efficacy in


(a) Projection Visualization.

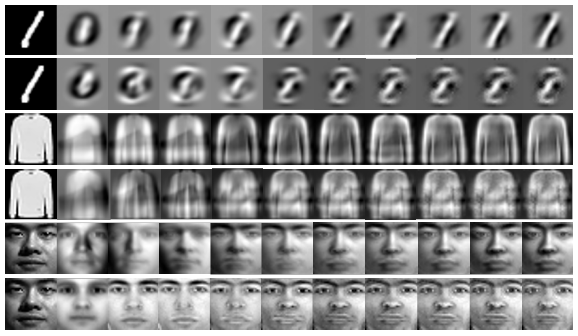

(b) Image Reconstruction.

Figure 2: **(a) Projection Visualization.** Each column represents a different column of the projection matrix used in the dimensionality reduction methods (PCA/LDA). The first column corresponds to the first matrix column. **(b) Reconstruction Visualization.** The first column shows the original image, while the remaining columns show the reconstructed images obtained by re-projecting the most column vectors numbered from (1) to (19). Each row corresponds to a specific projection and dataset. Each row in both images shows a combination of data and dimensionality reduction method: (i) PCA-MNIST, (ii) LDA-MNIST, (iii) PCA-FMNIST, (iv) LDA-FMNIST, (v) PCA-YaleB, (vi) LDA-YaleB.

capturing and retaining essential information during the reconstruction process. Although LDA also allows reconstruction using the right unitary matrix, as depicted in Figure 2b, it primarily focuses on identifying discriminative features, resulting in less accurate approximations of the original image. Specifically, second and last rows demonstrate instances where the reconstructed images deviate from the original image, highlighting the limitations of LDA in image restoration.

2.2.4 Non-negative Matrix Factorization (NMF)

NMF is a technique used to factorize a non-negative matrix $\mathbf{V}$ into two non-negative matrices $\mathbf{W}$ and $\mathbf{H}$, such that $\mathbf{V} \approx \mathbf{WH}$ (*Recap:* A non-negative matrix is a matrix in which all of its elements are non-negative).

More formally, given a non-negative matrix $\mathbf{V} \in \mathbb{R}^{m\times n}$, the goal of NMF is to find non-negative matrices $\mathbf{W} \in \mathbb{R}^{m\times r}$ and $\mathbf{H} \in \mathbb{R}^{r\times n}$, where r is the desired rank of the factorization, that minimize the following objective function:

$$\min_{\mathbf{W}\geq 0,\mathbf{H}\geq 0} \frac{1}{2}|\mathbf{V} - \mathbf{WH}|_F^2, \tag{19}$$

where $|\cdot|_F$ is the Frobenius norm. The non-negativity constraints ensure that the factors $\mathbf{W}$ and $\mathbf{H}$ are non-negative, which makes the factorization interpretable in many applications.

One popular algorithm for solving NMF is multiplicative update rules. The update rules for $\mathbf{W}$ and $\mathbf{H}$ are:

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{\mathbf{VH}^T}{\mathbf{WHH}^T}, \quad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T\mathbf{V}}{\mathbf{W}^T\mathbf{WH}}, \tag{20}$$

where $\odot$ denotes element-wise multiplication. These update rules iteratively refine the factors $\mathbf{W}$ and $\mathbf{H}$ until convergence. The algorithm is guaranteed to converge to a local minimum of the objective function.

NMF has been successfully applied in various applications. For example, NMF can be used for image compression . However, NMF has some limitations such as the sensitivity to the initial values of $\mathbf{W}$ and $\mathbf{H}$ and the difficulty of determining the appropriate rank r of the factorization.

2.2.5 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a nonlinear dimensionality reduction technique that is particularly effective for visualizing high-dimensional data in two or three dimensions. It works by finding a low-dimensional representation of the data that preserves the pairwise similarities between the high-dimensional data points.

Given a set of high-dimensional data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, t-SNE first constructs a probability distribution pij over pairs of high-dimensional data points, based on their similarity. The similarity between two points is defined as a Gaussian probability distribution:

$$p_{ij} = \frac{\exp(-|\mathbf{x}_i - \mathbf{x}j|^2/2\sigma_i^2)}{\sum k \neq l \exp(-|\mathbf{x}_k - \mathbf{x}_l|^2/2\sigma_k^2)} \tag{21}$$

where σ_i is the variance of the Gaussian distribution for point $\mathbf{x}_i$. The choice of σ_i is determined by a perplexity parameter that is specified by the user. t-SNE then constructs a similar probability distribution q_{ij} over pairs of low-dimensional data points $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, using a student-t distribution with one degree of freedom (i.e., a Cauchy distribution):

$$q_{ij} = \frac{(1 + |\mathbf{y}_i - \mathbf{y}j|^2)^{-1}}{\sum k \neq l(1 + |\mathbf{y}_k - \mathbf{y}_l|^2)^{-1}} \tag{22}$$

The variance of this distribution is again proportional to the distance between the points in the low-dimensional space. t-SNE then minimizes the Kullback-Leibler divergence between the two distributions, using gradient descent. The Kullback-Leibler divergence is a measure of the difference between the two probability distributions:

$$\mathrm{KL}(P|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{23}$$

Minimizing this divergence ensures that the low-dimensional representation preserves the pairwise similarities between the high-dimensional data points.

t-SNE is a computationally intensive algorithm and may not be suitable for large datasets. It is also highly dependent on the choice of hyperparameters, such as the perplexity parameter, which determines the balance between preserving global and local structure in the data (choosing an appropriate perplexity value is important to obtain meaningful and insightful embeddings). Despite these limitations, t-SNE has been widely used in a variety of applications, including image recognition, natural language processing, and single-cell RNA sequencing analysis. It can provide highly interpretable visualizations of high-dimensional data, and can reveal underlying structure and patterns that may not be apparent from the original data.

2.3 Clustering

Clustering is a technique in unsupervised learning that groups similar data points together based on their features. The goal of clustering is to partition the data into clusters such that data points within a cluster are more similar to each other than to data points in other clusters. Clustering can be used for a variety of tasks such as market segmentation, image segmentation, and anomaly detection.

2.3.1 k-Means Clustering

K-means clustering is a popular unsupervised algorithm used for clustering data points into k groups based on their similarity. The algorithm aims to minimize the sum of squared distances between the data points and their respective cluster centroids.

The steps of the k-means algorithm can be summarized as follows:

1. Choose the number of clusters k that you want to create.
2. Initialize k cluster centroids randomly.
3. Assign each data point to the nearest centroid based on its distance.

4. Recalculate the centroids based on the mean of the data points in each cluster.
5. Repeat steps 3 and 4 until the centroids no longer move or a maximum number of iterations is reached.

The distance between a data point x_i and a cluster centroid c_j can be calculated using the Euclidean distance formula:

$$d(x_i, c_j) = \sqrt{\sum_{j=1}^{n}(x_{ij} - c_{ij})^2} \tag{24}$$

where $x_i = (x_1, x_2, ..., x_n)$ is a data point, $c_j = (c_1, c_2, ..., c_n)$ is a cluster centroid, and n is the number of dimensions. The objective function of k-means algorithm is to minimize the sum of squared distances between each data point and its assigned centroid:

$$J = \sum_{i=1}^{m}\sum_{j=1}^{k} w_{ij} ||x_i - c_j||^2 \tag{25}$$

where m is the number of data points, k is the number of clusters, x_i is the ith data point, c_j is the jth cluster centroid, and $w_{ij} \in \{0, 1\}$ is a binary variable that indicates whether data point i belongs to cluster j ($\sum_{j=1}^{k} w_{ij} = 1$). The k-means algorithm seeks to find the optimal cluster centroids that minimize the objective function J. This is done through an iterative process of assigning data points to clusters and updating the centroids based on the mean of the data points in each cluster.

Overall, k-means is a simple yet effective algorithm for clustering data points into groups based on their similarity. Its computational efficiency and ease of implementation make it a popular choice for a variety of applications.

2.3.2 Hierarchical Clustering

Hierarchical Clustering is a popular clustering algorithm that aims to create a hierarchy of clusters in a dataset. It iteratively merges or splits clusters based on their similarities until a desired clustering structure is obtained. There are two main types of hierarchical clustering: Agglomerative and Divisive.

Agglomerative Hierarchical Clustering starts with each data point as an individual cluster and iteratively merges the closest clusters until a single cluster encompasses all the data points. The proximity between clusters is typically defined by a distance metric such as Euclidean distance or cosine similarity. The algorithm proceeds as follows:

1. Compute the distance matrix or similarity matrix between all pairs of data points.
2. Treat each data point as a separate cluster.
3. Repeat until a single cluster remains:
    (a) Find the two closest clusters based on the defined distance metric.
    (b) Merge the two closest clusters into a new cluster.
    (c) Update the distance matrix by recalculating the distances between the new cluster and the remaining clusters.
4. The resulting dendrogram or tree-like structure represents the hierarchical clustering.

Divisive Hierarchical Clustering, also known as top-down clustering, takes the opposite approach by starting with a single cluster containing all the data points and recursively splitting it into smaller clusters until each data point is in its own cluster. The algorithm proceeds as follows:

1. Treat all data points as a single cluster.
2. Repeat until each data point is in its own cluster:
    (a) Select a cluster to split.

(b) Divide the selected cluster into two subclusters using a specified criterion.

(c) Update the hierarchy.

Dendrograms - a dendrogram is a visual representation of the hierarchical clustering results. It displays the merging or splitting of clusters at different levels of the hierarchy. The x-axis represents the data points, and the y-axis represents the distance or similarity between clusters. Dendrograms help in interpreting the structure and relationships within the data.

2.3.3 DBSCAN Clustering

Given a dataset consisting of n data points, DBSCAN is an algorithm that groups the points into clusters based on their density in the feature space. It operates by defining a neighborhood around each data point and identifying core points, density-reachable points, and noise points. Few key terms used in DBSCAN are:

- Epsilon (ε): It represents the radius of the neighborhood around each data point. Points within this radius are considered neighbors.
- MinPts: It denotes the minimum number of points required to form a dense region or cluster. If a point has at least MinPts points within its ε-neighborhood, it is considered a core point.

The steps of the DBSCAN algorithm:

1. Randomly select an unvisited data point from the dataset.
2. Retrieve all the points within its ε-neighborhood. If the number of points within the neighborhood is less than MinPts, mark the point as noise and move to the next unvisited point.
3. If the selected point has at least MinPts points within its ε-neighborhood, it is marked as a core point, and a new cluster is created.
4. Expand the cluster by recursively adding density-reachable points to the cluster. A point is density-reachable if it has at least MinPts points within its ε-neighborhood and is also reachable from another core point.
5. Repeat steps 1 to 4 until all points have been visited.

The resulting clusters are formed by connecting core points and their density-reachable points. Any unvisited point that does not belong to any cluster is considered noise. Mathematically, the core point (p) can be defined as follows:

$$\text{core}(p) = q \mid \text{dist}(p, q) \leq \varepsilon, q \in D, |N(q)| \geq \text{MinPts} \tag{26}$$

where dist(p, q) is the distance metric used to measure the distance between two points p and q, N(q) represents the set of points within the ε-neighborhood of q, and D is the dataset.

The density-reachable points (q) from a core point (p) are given by:

$$\text{density-reachable}(p, q) = q \mid q \in D, \text{dist}(p, q) \leq \varepsilon, |N(q)| \geq \text{MinPts} \tag{27}$$

Finally, the clusters are formed by connecting core points and their density-reachable points. Any point that is not a core point and does not belong to any cluster is considered noise.

DBSCAN is advantageous because it can discover clusters of arbitrary shapes and is robust to noise and outliers. It is also less sensitive to the initial positions of the clusters than k-Means and does not require the number of clusters to be specified in advance. However, selecting appropriate values for ε and $minPts$ is crucial for obtaining meaningful clusters, which can be challenging in practice.

References

[1] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.

[2] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.

[3] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman, “From few to many: Illumination cone models for face recognition under variable lighting and pose,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 23, no. 6, pp. 643–660, 2001.