MODULE-4

PROBABILISTIC PCA

Probabilistic PCA (PPCA) is a linear dimensionality reduction technique with a probabilistic framework. It can be thought of as a generalization of standard PCA, which incorporates a probabilistic model for handling uncertainties, such as noise in the data.

- Probabilistic Principal Component Analysis (PPCA) is a statistical method used to reduce the dimensions of a dataset, similar to standard Principal Component Analysis (PCA).
- It makes use of probability theory to provide a more flexible approach to modeling data with missing values and estimating uncertainty.
- Probabilistic Principal Component Analysis (PPCA) is a statistical method that helps perform dimensionality reduction by assuming that the data is generated from a probabilistic model. PPCA with some mathematical examples are given below:

1. Mathematical Model of PPCA

- In PPCA, assume that our observed high-dimensional data is generated from some lower-dimensional latent variables through a linear transformation + Gaussian noise.

- **Observed Data:** $\mathbf{x} \in \mathbb{R}^D$ (high-dimensional data, with dimension D)
- **Latent Variable:** $\mathbf{z} \in \mathbb{R}^d$ (lower-dimensional representation, with dimension d, where $d < D$)
- **Loading Matrix:** $\mathbf{W} \in \mathbb{R}^{D \times d}$ (linear transformation matrix)
- **Mean:** $\boldsymbol{\mu} \in \mathbb{R}^D$ (mean vector of the observed data)
- **Noise:** $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ (Gaussian noise with variance σ^2)

The generative model for PPCA can be described as follows:

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \epsilon$$

where:

- $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a latent variable drawn from a standard normal distribution.
- $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ is Gaussian noise added to model real-world imperfections.

2. Likelihood of Observed Data

To estimate $\mathbf{W}$, $\boldsymbol{\mu}$, and σ^2, PPCA uses a maximum likelihood approach. The likelihood of the observed data $\mathbf{x}$ can be derived based on the above generative model.

Using properties of Gaussian distributions, we can derive that the observed data $\mathbf{x}$ follows a multivariate normal distribution:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$$

where the covariance matrix $\mathbf{C}$ is given by:

$$\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$$

3. Example Calculation

Suppose we have a dataset consisting of three variables (e.g., $D = 3$), and we assume there is a lower-dimensional representation of dimension $d = 2$. We can model the relationship between the observed data and latent variables using PPCA.

1. **Step 1: Define the Latent Variables and Loading Matrix**

   Let's say our latent variable $\mathbf{z} \in \mathbb{R}^2$ is drawn from a standard normal distribution:

   $$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

   And let's assume the loading matrix $\mathbf{W}$ is:

   $$\mathbf{W} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

2. **Step 2: Generate Observed Data**

   To generate an observed data point $\mathbf{x}$, we use:

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \epsilon$$

   Suppose $\boldsymbol{\mu} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$ and the noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ with $\sigma^2 = 0.1$.

   Let's consider a latent variable realization:

$$\mathbf{z} = \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix}$$

   Then:

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \epsilon$$

   Calculating $\mathbf{W}\mathbf{z}$:

$$\mathbf{W}\mathbf{z} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

   Adding the mean vector $\boldsymbol{\mu}$:

$$\mathbf{x} = \begin{bmatrix} 0.5 \\ -1.0 \\ -0.5 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1.0 \\ 1.5 \end{bmatrix}$$

   Finally, adding Gaussian noise (assuming ϵ is small), we get an observed data point close to:

$$\mathbf{x} \approx \begin{bmatrix} 2.5 \\ 1.0 \\ 1.5 \end{bmatrix}$$

3. Step 3: Finding Principal Components

   To find the principal components, PPCA estimates the parameters $\mathbf{W}$, $\boldsymbol{\mu}$, and σ^2 by maximizing the likelihood of the observed data. This involves using techniques like Expectation-Maximization (EM) to iteratively refine the estimates.

4. Step 4: Dimensionality Reduction

   Once we have the parameters, we can project the observed data $\mathbf{x}$ to the lower-dimensional latent space. The principal components are given by:

$$\mathbf{z} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T(\mathbf{x} - \boldsymbol{\mu})$$

   This gives us the reduced representation of $\mathbf{x}$ in the d-dimensional space.

- PPCA assumes that the high-dimensional data $\mathbf{x}$ is generated from a lower-dimensional latent variable $\mathbf{z}$ through a linear transformation $\mathbf{W}$ and Gaussian noise ϵ.
- The observed data is modeled as $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \epsilon$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$

  .
- PPCA uses maximum likelihood estimation to estimate the parameters $\mathbf{W}$, $\boldsymbol{\mu}$, and σ^2.
- The latent variable $\mathbf{z}$ serves as a lower-dimensional representation of the observed data $\mathbf{x}$, which can be used for dimensionality reduction.

PPCA is particularly useful for handling missing data, as it models the data generation process probabilistically, allowing for robust estimation even in the presence of missing values.

Independent Component Analysis (ICA)

- It is a computational technique used to separate a multivariate signal into additive, independent components.
- It is particularly useful in scenarios where you have mixed signals and want to extract the original sources based on the assumption that they are statistically independent and non-Gaussian.

Key Concepts of ICA

1.Independence: ICA assumes that the source signals are independent from each other.

2.Non-Gaussianity: ICA often relies on the fact that the original source signals are not normally distributed. This is essential because Gaussian distributions are fully characterized by their mean and variance, making it difficult to distinguish between independent components.

Real-Life Example: The Cocktail Party Problem

Imagine you are at a party with multiple conversations happening simultaneously. You have two microphones recording the sound, and each microphone picks up a mixture of voices. Your goal is to isolate each person's voice from the mixed recordings.

1. Problem Setup

Let's denote the mixed signals and original sources as follows:

- **Mixed Signals:**

$$\mathbf{x} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

- **Original Sources:**

$$\mathbf{s} = \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$$

- **Mixing Matrix:** The relationship between mixed signals and original sources can be expressed as:

$$\mathbf{x} = \mathbf{A}\mathbf{s}$$

where $\mathbf{A}$ is the mixing matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

2. Recovering the Original Signals

To recover the original sources $\mathbf{s}$ from the mixed signals $\mathbf{x}$, we need to find an **unmixing matrix** $\mathbf{W}$ such that:

$$\mathbf{s} = \mathbf{W}\mathbf{x}$$

2.1 The Mathematical Equation for Reconstruction

To reconstruct the original signals, we need to derive $\mathbf{W}$ from the mixed signals:

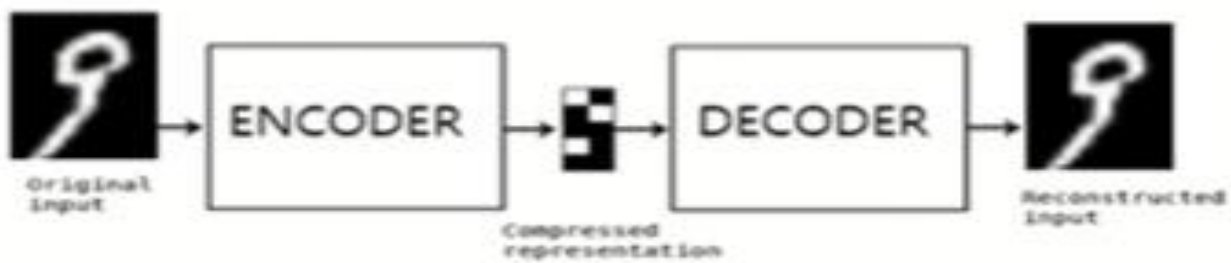
$$\mathbf{W} = \mathbf{A}^{-1}$$

If $\mathbf{A}$ is invertible, then the inverse can be computed, allowing us to write:

$$\mathbf{s} = \mathbf{W}\mathbf{x} = \mathbf{A}^{-1}\mathbf{x}$$

AUTOENCODERS

- An auto encoder is a simple (Relatively simple) ML algorithm which acquires input image and it will reconstruct the same. I.e. the image is compressed.
- This is also called as dimensionality reduction.
- Well, The dimensionality reduction certainly is used in the data preprocessing (we reduce/compress)
- The process of dimensionality reduction reduces the dimensionality of the considered dataset.
- There could be many features in the considered dataset. Do we use all those features? Not exactly. We use some features of them and such features are to be identified.
- Dimensionality reduction certainly helps in identifying the features which are useful (i.e. of interest).
- Auto encoders are meant for this Dimensionality Reduction.



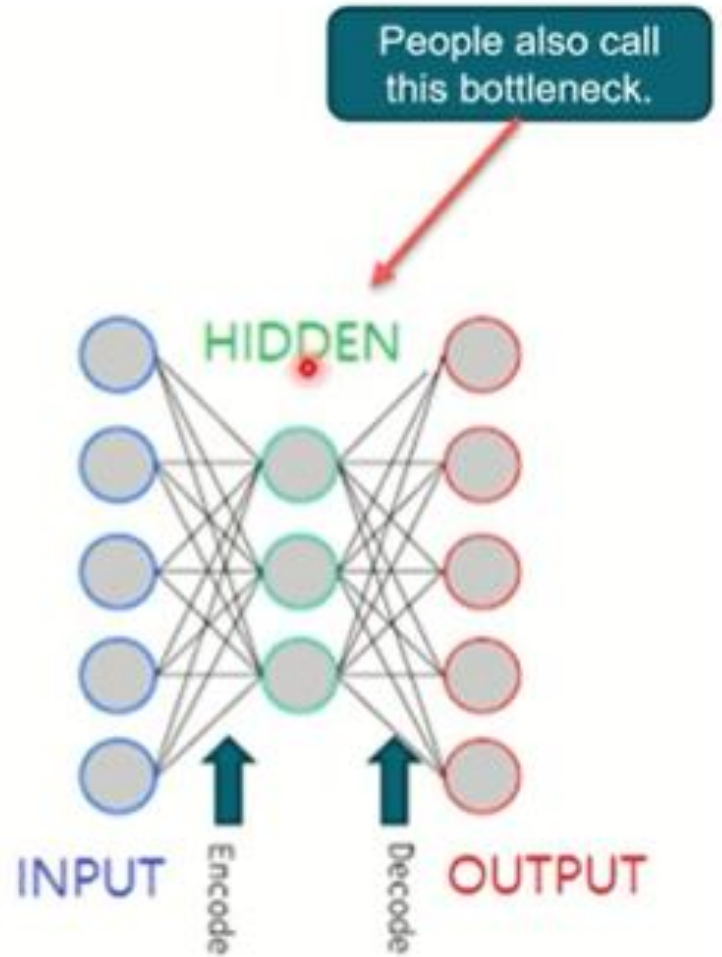
- We know what PCA – Principal Component Analysis (PCA). They do dimensionality reduction.
- The conversion from higher dimension to lower dimension happens. They also with PCA try to preserve the features which are important. This done with Linear Transformation. But, Auto encoders are using Non – Linear transformation.
- That's the major difference.

LAYERS OF AUTOENCODERS

- Input Layer
- Hidden Layer (People refer it as bottleneck
- Output Layer

Contd.,

- We can understand a little more now..
- Refer to the simple Auto Encoder Architecture presented in the RHS (Don't worry about the name and other details of the same)
- It has **two layers** (three, input ignored). It has one hidden layer as you could see.
- An instance for a quicker reference. Assume we feed an image with **8 pixel values as input** to the Auto Encoder.
- This is now **compressed** by the **encoder, to 5 Pixels (Let's say) at the hidden layer i.e. bottleneck layer or middle layer.**
- **Now, comes the decoder. It will try to reconstruct the 5 pixels to 8. (See, it will try to!!)**

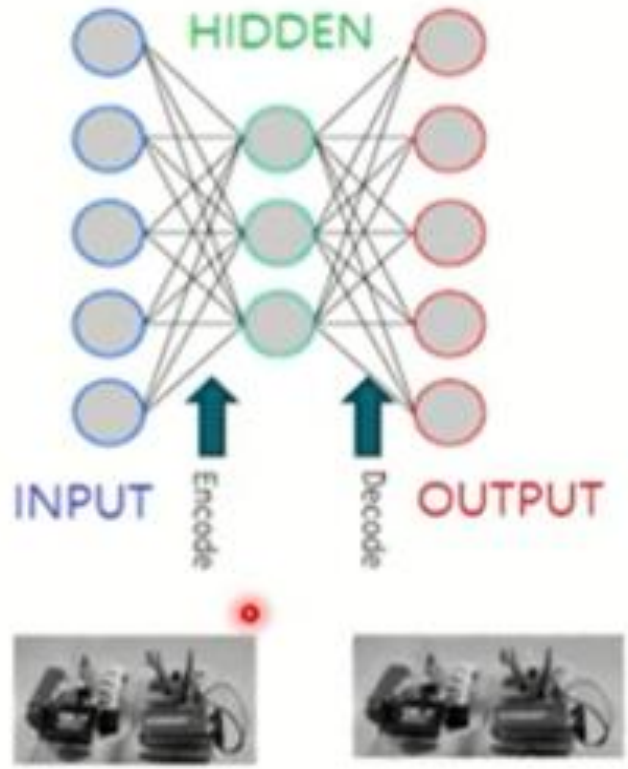


The Features / Characteristics of Auto Encoders

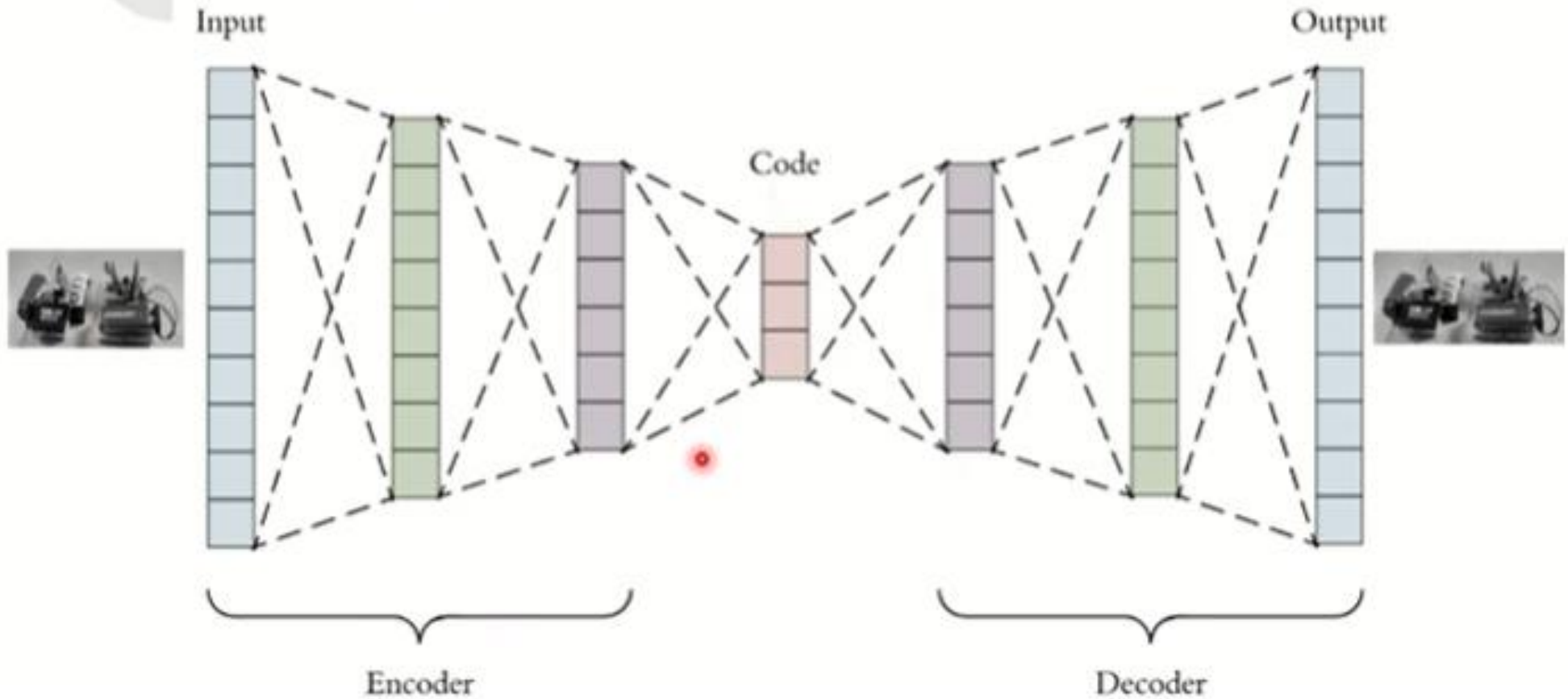
- Data-specific: This can work only on the data which are similar to what the system is already trained on. (Also, this is not like GZIP or WINRAR, where the compression happens and packaging is done) For an instance, if an autoencoder is trained on compressing cat images may not work fine with donkey images.

- Lossy: Well, the expectation may not always happen. Same is the case with Auto encoders. The output may not be exact as input. But, it will be a very closer ones. It will certainly be a degraded version. No doubt. This is lossy version. And if you want lossless compression, well, find out different methods.

- Unsupervised / Self supervised – We can call this unsupervised as we need not do anything other than feeding the raw input. No explicit labeling required. To be precise – They are Self Supervised. They generate own labels from training.

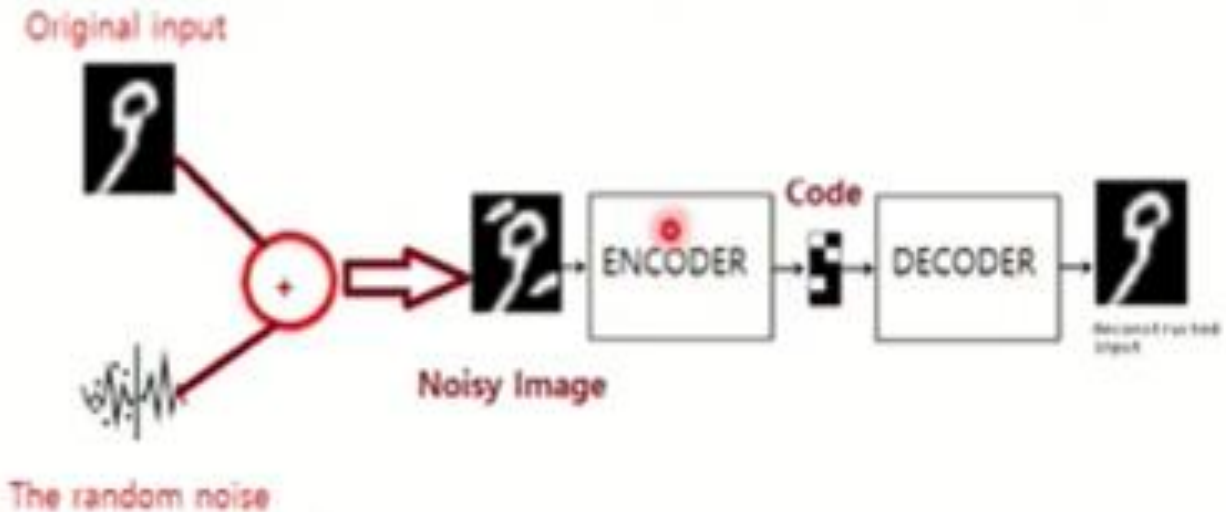


A better pic



DENOISING AUTOENCODERS

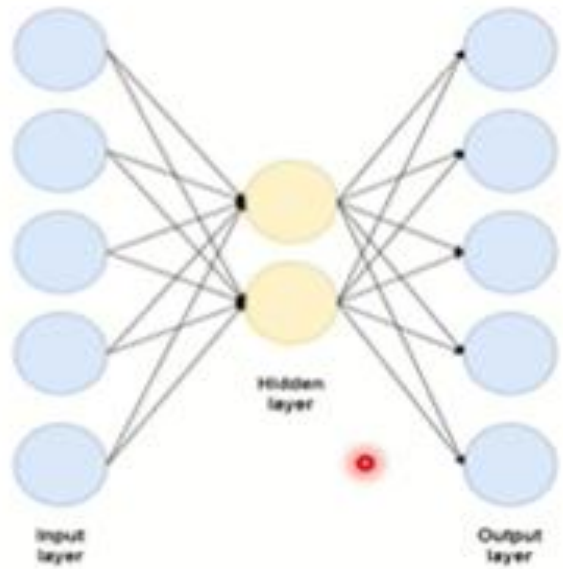
- Denoising Autoencoders produce a Corrupted (noised) copy of the input through the introduction of some noise. i.e. noise added to corrupt the input!
- Why to do this? **Autoencoders with more hidden layers than inputs run the risk of learning the identity function – where the output simply equals the input – thereby becoming useless. i.e. the no learning the features happen.**
- **So, this random noise inclusion avoids that. Here, we force the autoencoder to learn the original data after the noise being removed. Here, the autoencoder identifies the noise, removes the noise, learns the important features from the input data.**



Vanilla Autoencoder

- It is the simplest version of autoencoders. This is pretty ordinary too. It has two layers (ignoring the input layer). Input, hidden and output layers are there.
- Just to note:
  - The hidden layer is smaller < input and output layer.
  - Input and Output layer are of same size.

- One can understand the same from the diagram.
- The hidden layer is compressed representation, and with two sets of weights (and biases) we encode our input data into the compressed representation and decode our compressed representation back into input space.



CONVOLUTIONAL AUTO ENCODERS

A **convolutional autoencoder** is a type of autoencoder that uses convolutional layers in both the encoder and decoder parts of the network. This architecture is particularly effective for processing image data, as it leverages the spatial hierarchies in images.

How It Works

1. **Encoder**:
    - The encoder consists of multiple convolutional layers followed by activation functions (usually ReLU).
    - Each convolutional layer applies a set of filters (kernels) to the input image, capturing local patterns and features.
    - Typically, pooling layers (like max pooling) are used after some convolutional layers to downsample the feature maps, reducing the spatial dimensions while retaining important features. This also helps to make the representation more abstract.

2. **Bottleneck**:
    - The output of the encoder is a compressed representation of the input image, often called the latent space or bottleneck. This representation captures the most salient features learned by the encoder.
3. **Decoder**:
    - The decoder mirrors the encoder structure but uses transposed convolutional layers (also known as deconvolutional layers) to upsample the compressed representation back to the original input size.
    - The transposed convolutions effectively reverse the pooling operations, reconstructing the image by learning to "decode" the features.

4. **Loss Function:**

   - The network is trained to minimize the difference between the input image and the reconstructed image produced by the decoder. A common loss function for this purpose is the mean squared error (MSE).

SPARSE AUTOENCODERS

A sparse auto-encoder is a type of neural network that has more hidden units than input units but only allows a few hidden units to be "active" (or turned on) at any given time. This limited activation is known as **sparsity**.

Sparsity Control: You can control how sparse the network is by:

- Manually setting some hidden units to zero.
- Adjusting the activation functions (the rules that determine whether a unit turns on).
- Adding a special term to the loss function (the equation that the model tries to minimize during training) that encourages sparsity.

A sparse auto-encoder is a powerful tool that learns useful patterns by activating only a few hidden units at a time, helping to filter out noise and focus on important features. However, it requires careful tuning and can be more computationally demanding.

Advantages

1.Noise Reduction: The sparsity helps filter out noise and irrelevant information when the model learns from the data.

2.Meaningful Features: By focusing on a few active units, sparse autoencoders can learn important features from the data, leading to more useful representations.

Disadvantages

1. Hyper parameter Sensitivity: The performance of a sparse autoencoder greatly depends on choosing the right settings (hyperparameters). Different inputs should activate different hidden units, which can be tricky to balance.

2. Increased Complexity: Enforcing sparsity adds more computational demands, making the training process more complex and potentially slower.

MANIFOLD

A **Manifold** is a mathematical structure that generalizes the notion of curves and surfaces to higher dimensions. Actually, a manifold is a space that, when viewed locally (i.e., within a small neighborhood around any point), resembles Euclidean space (like lines, planes, etc.), but globally, it may have a more complex structure.

MANIFOLD LEARNING

Manifold learning is a type of machine learning technique used for **dimensionality reduction**, which means simplifying complex data while keeping the important information intact. It is based on the idea that real-world data (like images, sounds, or even stock market data) often lies on a lower-dimensional **manifold** inside a much higher-dimensional space.

Popular Manifold Learning Algorithms:

- **t-SNE** (t-Distributed Stochastic Neighbor Embedding): Often used to visualize high-dimensional data in 2D or 3D.
- **Isomap**: Preserves the distances between data points as it reduces dimensions.
- **LLE** (Locally Linear Embedding): Focuses on preserving the local relationships between nearby data points.

Multidimensional Scaling (MDS)

- It is a statistical technique used for visualizing the similarities or dissimilarities between a set of objects in a lower-dimensional space, typically in two or three dimensions.
- The main objective of MDS is to represent the data in such a way that the distances between the points in the reduced space reflect the similarities or dissimilarities of the original data.

Types of MDS

- **Classical MDS**: Preserves the actual distances between objects. It assumes that the dissimilarities are measured on an interval or ratio scale.
  - Assumes the input dissimilarities are **Euclidean distances**, and it finds a configuration of points in lower dimensions that preserves those distances as closely as possible.
  - In this case, the dimensionality reduction technique is **linear** since the method essentially uses linear algebra (eigenvalue decomposition) to achieve the reduction.
- **Non-metric MDS**: Focuses on preserving the rank order of dissimilarities rather than the exact distances. It's useful when the dissimilarities are ordinal or when the exact values are not as important as their relative ranking.
  - Focuses on preserving the **rank ordering** of dissimilarities rather than the exact distances.
  - This version of MDS is **non-linear**, as it doesn't assume a simple linear relationship between high-dimensional and low-dimensional spaces.

EXAMPLE

Suppose you have 4 objects (A, B, C, D), and you know the dissimilarities between each pair of objects, as shown in the matrix below. The goal is to use MDS to place these 4 objects on a 2D plane such that the distances between the objects on the plane correspond as closely as possible to the dissimilarities.

Step 1: Dissimilarity Matrix

This is a symmetric matrix where the entry at row i and column j represents the dissimilarity between objects i and j.

	A	B	C	D
A	0	5	4	7
B	5	0	6	3
C	4	6	0	2
D	7	3	2	0

In this matrix, for example, the dissimilarity between A and B is 5, between A and C is 4, etc. The diagonal entries are 0, indicating no dissimilarity between an object and itself.

Step 2: Objective of MDS

MDS tries to map these objects into a 2D or 3D space such that the pairwise distances between them reflect the dissimilarities in the matrix.

1.First, we aim to compute a matrix of squared distances based on the dissimilarity matrix.

2.Then, we will apply **Classical MDS** to map the objects to a 2D space.

Step 3: Applying Classical MDS

1. Centering the Dissimilarity Matrix:

MDS works by performing **eigenvalue decomposition** on a matrix derived from the dissimilarity matrix. Start by converting the dissimilarity matrix into a matrix of **squared distances** D^2

$$D^2 = \begin{bmatrix} 0^2 & 5^2 & 4^2 & 7^2 \\ 5^2 & 0^2 & 6^2 & 3^2 \\ 4^2 & 6^2 & 0^2 & 2^2 \\ 7^2 & 3^2 & 2^2 & 0^2 \end{bmatrix} = \begin{bmatrix} 0 & 25 & 16 & 49 \\ 25 & 0 & 36 & 9 \\ 16 & 36 & 0 & 4 \\ 49 & 9 & 4 & 0 \end{bmatrix}$$

Step 4: Center the Distance Matrix (Double Centering)

Convert this distance matrix into a matrix that can be decomposed using eigenvalue decomposition. This involves double-centering the matrix. The centered matrix B is given by:

$$B = -\frac{1}{2}(D^2 - \text{row means} - \text{column means} + \text{overall mean})$$

This process shifts the data so that the points have a mean of zero in each dimension, which is a necessary step before applying eigenvalue decomposition.

Step 5: Eigenvalue Decomposition of the Centered Matrix

Once you have the centered matrix B, the next step is to perform **eigenvalue decomposition**. This means breaking down matrix B into its eigenvalues and eigenvectors. The decomposition of B is expressed as:

$B = Q\lambda Q^{-1}$ Where:

- Q is a matrix of **eigenvectors**.
- λ is a diagonal matrix of **eigenvalues**.

Eigenvalue decomposition essentially transforms the matrix B into a set of eigenvectors and corresponding eigenvalues. The eigenvalues provide information about the variance in the data along the eigenvectors (directions in the new coordinate space).

6. Select the Largest Eigenvalues

The eigenvalues in λ represent how much variance is explained in the data along the corresponding eigenvectors. In classical MDS, we usually want to project the data into a lower-dimensional space (e.g., 2D or 3D), so select the largest k eigenvalues and their corresponding eigenvectors.

For example:

- If we want to reduce the data to 2 dimensions, we select the two largest eigenvalues and the two corresponding eigenvectors.

7. Construct the Low-Dimensional Representation

Once we have the eigenvectors corresponding to the largest eigenvalues, we use these to construct the coordinates of the objects in the lower-dimensional space. The coordinates are obtained by:

$$X = Q_k \Lambda_k^{1/2}$$

Where:

- Q_k is the matrix of the eigenvectors corresponding to the largest k eigenvalues.
- Λ_k is the diagonal matrix of the largest k eigenvalues.
- X is the matrix of the coordinates of the objects in the new k-dimensional space.

The resulting matrix X gives the coordinates of each object in the low-dimensional space, where the pairwise Euclidean distances between the objects in this space closely approximate the original dissimilarities.

NON METRIC MDS

Step 1: Define the Dissimilarity Matrix

Consider three objects: A, B, and C. Their pairwise dissimilarities are represented in the following matrix:

$$\text{Dissimilarity Matrix (D)} = \begin{bmatrix} 0 & 4 & 7 \\ 4 & 0 & 3 \\ 7 & 3 & 0 \end{bmatrix}$$

- The diagonal elements are zero because they represent the dissimilarity of an object to itself.
- $D_{AB} = 4$ indicates that the dissimilarity between objects A and B is 4.
- $D_{AC} = 7$ indicates the dissimilarity between A and C is 7.
- $D_{BC} = 3$ indicates the dissimilarity between B and C is 3.

Step 2: Rank the Dissimilarities

Now we need to rank the dissimilarities. We will look at the non-diagonal elements (the actual dissimilarities):

1. $D_{AB} = 4$
2. $D_{AC} = 7$
3. $D_{BC} = 3$

The ranks of the dissimilarities (1 for the least dissimilarity, 3 for the most) are as follows:

- $\text{Rank}(D_{AB}) = 2$
- $\text{Rank}(D_{AC}) = 3$
- $\text{Rank}(D_{BC}) = 1$

$$\text{Rank Matrix (R)} = \begin{bmatrix} - & 2 & 3 \\ 2 & - & 1 \\ 3 & 1 & - \end{bmatrix}$$

Step 3: Initialize Random Configuration

For NMDS, we will start with a random configuration of points in 2D space. Let's assign random initial coordinates to our objects:

- Object A: $(1, 1)$
- Object B: $(3, 1)$
- Object C: $(2, 4)$

Step 4: Calculate Distances in the Low-Dimensional Space

Now, we will calculate the Euclidean distances between the objects in their initial positions:

- Distance between A and B:

$$d_{AB} = \sqrt{(3-1)^2 + (1-1)^2} = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

- Distance between A and C:

$$d_{AC} = \sqrt{(2-1)^2 + (4-1)^2} = \sqrt{1^2 + 3^2} = \sqrt{1+9} = \sqrt{10} \approx 3.16$$

- Distance between B and C:

$$d_{BC} = \sqrt{(2-3)^2 + (4-1)^2} = \sqrt{(-1)^2 + 3^2} = \sqrt{1+9} = \sqrt{10} \approx 3.16$$

The distance matrix based on the initial configuration is:

$$\text{Distance Matrix (D\_2D)} = \begin{bmatrix} 0 & 2 & 3.16 \\ 2 & 0 & 3.16 \\ 3.16 & 3.16 & 0 \end{bmatrix}$$

Step 5: Rank the Distances

Now, we will rank the distances we just calculated in the low-dimensional space:

1. $d_{AB} = 2$ (Rank 1)
2. $d_{AC} = 3.16$ (Rank 2)
3. $d_{BC} = 3.16$ (Rank 2)

The rank matrix for distances becomes:

$$\text{Rank Distances} = \begin{bmatrix} - & 1 & 2 \\ 1 & - & 2 \\ 2 & 2 & - \end{bmatrix}$$

Step 6: Compare Ranks and Calculate Stress

The next step is to calculate the **stress**. The stress function measures the discrepancy between the ranks of the original dissimilarities and the ranks of the distances in the reduced space. The formula for stress is:

$$\text{Stress} = \sqrt{\frac{\sum_{i<j}(rank_{ij} - rank_{d_{ij}})^2}{\sum_{i<j}(rank_{ij})^2}}$$

Where:

- $rank_{ij}$ is the rank from the original dissimilarity matrix.
- $rank_{d_{ij}}$ is the rank from the distance matrix.

Calculating Stress:

1. For D_{AB} (original rank 2) and d_{AB} (calculated rank 1):
    - Contribution to stress: $(2 - 1)^2 = 1$
2. For D_{AC} (original rank 3) and d_{AC} (calculated rank 2):

1. For D_{AB} (original rank 2) and d_{AB} (calculated rank 1):
   - Contribution to stress: $(2-1)^2 = 1$
2. For D_{AC} (original rank 3) and d_{AC} (calculated rank 2):
   - Contribution to stress: $(3-2)^2 = 1$
3. For D_{BC} (original rank 1) and d_{BC} (calculated rank 2):
   - Contribution to stress: $(1-2)^2 = 1$

Sum of contributions:

- Numerator: $1 + 1 + 1 = 3$

Sum of ranks squared:

- $(1^2 + 2^2 + 3^2) = 1 + 4 + 9 = 14$

Putting it together:

$$\text{Stress} = \sqrt{\frac{3}{14}} \approx 0.65$$

Step 7: Iterative Adjustment

To minimize stress, we would typically iterate ' adjusting the positions of points A, B, and C based on the current stress value. Adjustments would made to the coordinates of A, B, and C, and the process of calculating distances, ranking, and stress would repeat until the stress is minimized.