UNIT-II

Questions:

1) Principal component Analysis / Diff b/w PCA & Autoencoders
2) Autoencoder Architechure.
3) Denoising of Autoencoder for
4) Sparsity in Autoencoder
5) Lower-Dimensional Representation
6) Diff b/w AE & PCA.
7)
8)
9)
10)
11)
12)
13)
14)
15)

Chapter 2:-

1) Learning Lower Dimensional Representation

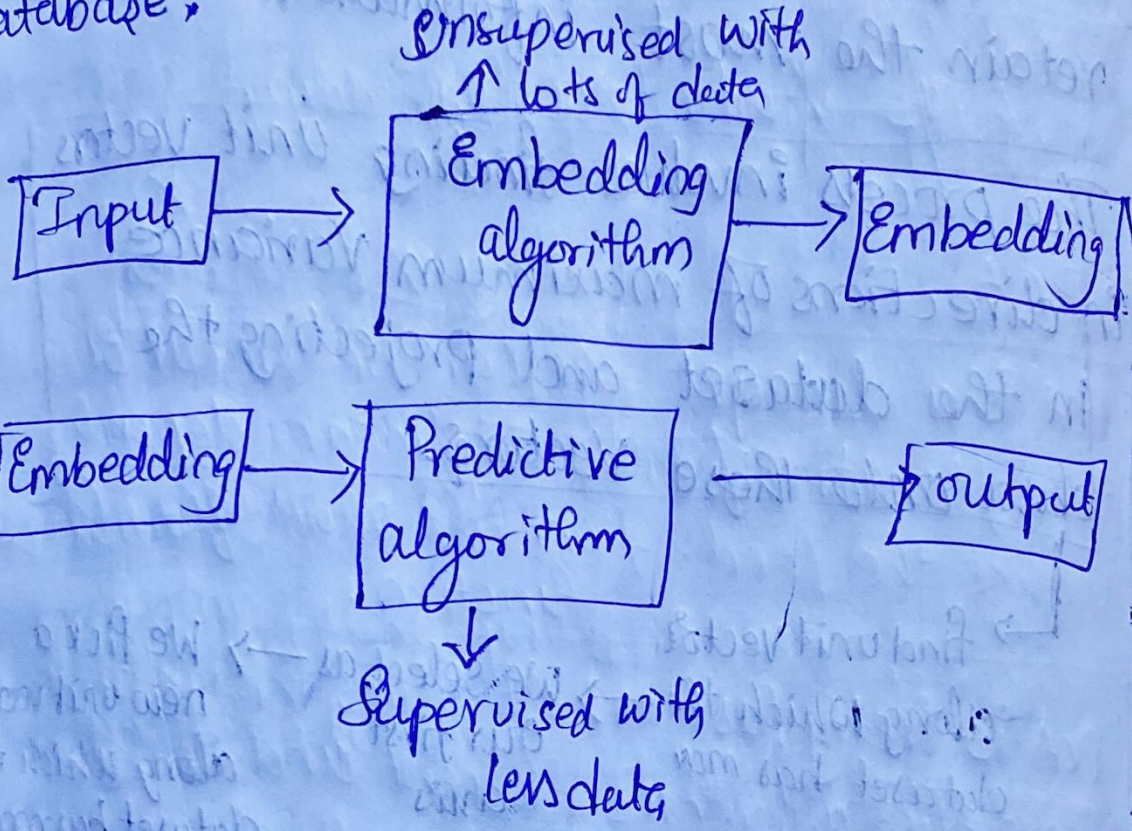
Large input vectors leads to larger models with many parameters, which can be Expressive but are also data-hungry and Prone to Overfitting.

The convolutional architectures address the curses of dimensionality by Reducing the number of Parameters, maintaining Expressiveness without need for Extensive training data.

There is often a scarcity of labeled data, which is necessary for training Convolutional networks, leading to the needs for methods that can work effectively with less labeled data.

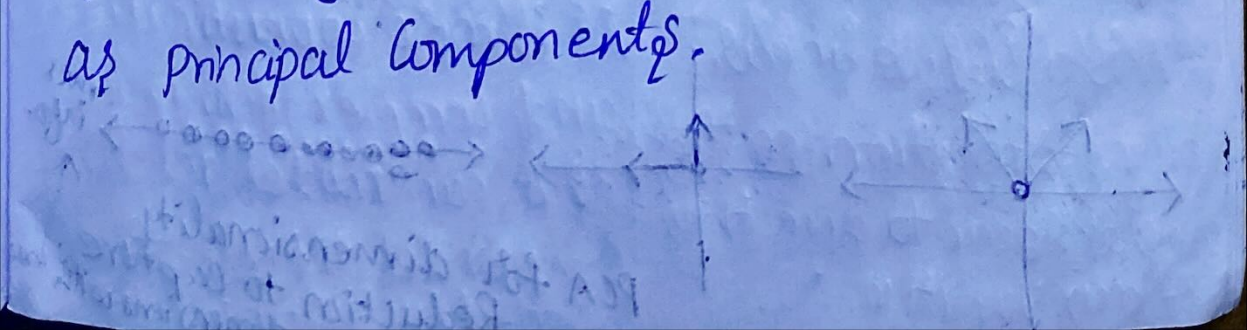
One Solution is to develop models that learn from unlabeled data by creating lower-dimensional embeddings in an unsupervised manner.

These Embeddings capture essential features and can be used with smaller models, reducing the requirments for large labeled database.



Principal Component Analysis:

Principal component Analysis is a Statistical Procedure that aims to transform a set of Possibly correlated variables into a set of linearly uncorrelated variables known as principal components.



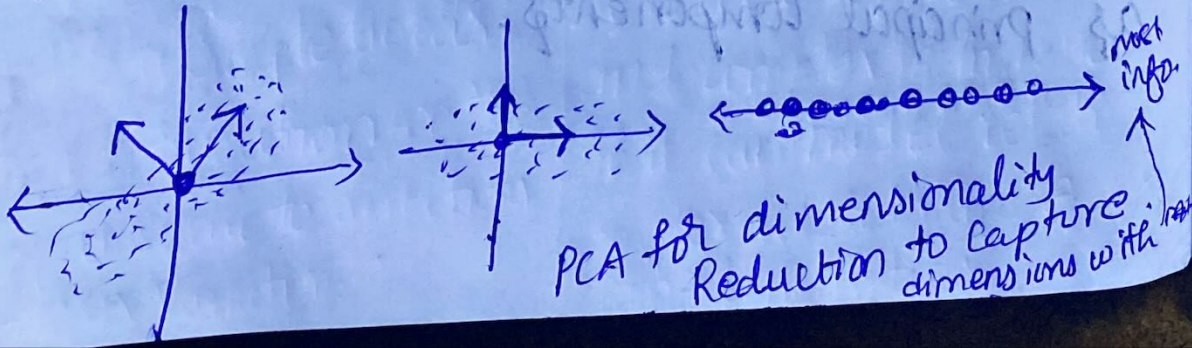
The goal of PCA ~~is to retain as much~~ seeks to reduce the dimensionality of data by finding the new axes that retain the most information.

The process involves finding unit vectors in directions of maximum variance in the dataset and projecting the data onto these vectors.

→ find unit vector along which the dataset has max variance → We select as our first axis → We pick a new unit vector along which the dataset has max variance. ← This is our second axis

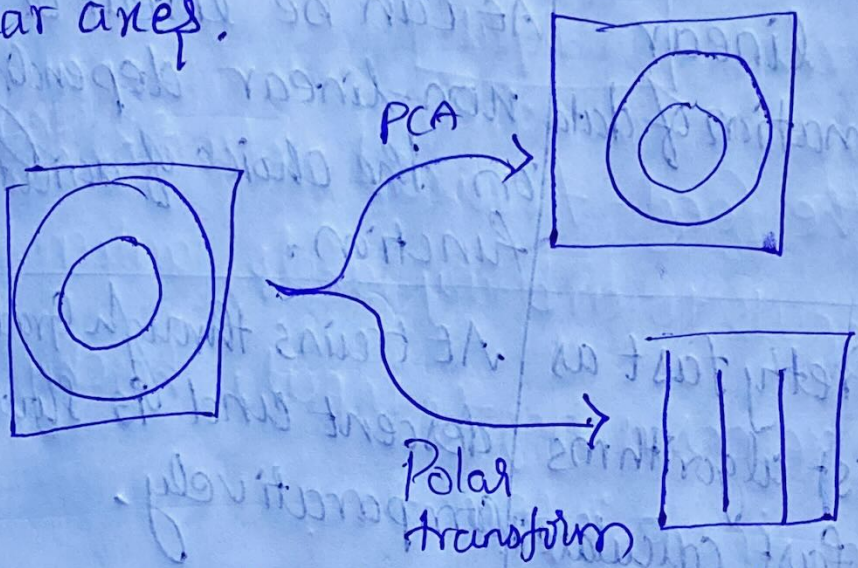
Continue this process until we have found a total of d new vectors. ← Project our data onto this new set of axis.

This is result is shown in below fig.



The dataset is represented as a matrix, and an Embedding matrix is created that Reduces dimensions while preserving as much info as possible.

PCA can fail With complex datasets, Particularly Where important patterns are nonlinear, as it primarily capture Variance along linear axes.



A Situation in which PCA fails to optimally transform the data for dimensionality Reducing

The nonlinear Relationships in data, such as the above concentric circles, may Require diff approches, like polar transformation, to Separate features Effectively.

The limitations of PCA in dealing with nonlinear relationships highlight the needs

for non-linear dimensionality Reduction techniques, which are addressed by Neural models in deep learning.

PCA v/s Auto Encoders

PCA	AE
PCA is a linear transformation of data	AE can be linear or non-linear depending on the choice of activation function.
PCA is Pretty fast as there Exist algorithms that can fast calculate it.	AE trains through Gradient descent and is slower comparatively.
PCA Projects data into dimensions that are orthogonal to each other resulting in very low or close to zero correlation in the Projected data.	AE transformed data doesn't guarantee that because the way it's trained is merely to minimize the reconstruction loss.
PCA is a simple linear transformation on the input space to directions of max variations.	AE is more sophisticated and complex technique that can model relatively complex relationships & non-linearities.

one rule to thumb could be the size of Data. PCA for small datasets	AE for larger dataset.
PCA hyperparameter is "K"	AE it is the architecture of the neural network
AE with Single layer & linear activation has similar performance as PCA.	AE with multiple layers and non-activation function termed as Deep Autoencoder Prone to Overfitting.

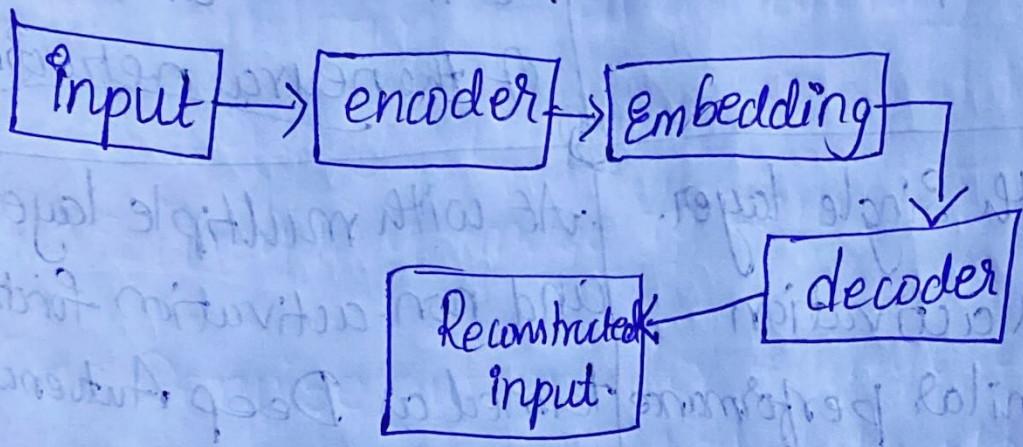
Autoencoder Architecture:

The feed-forward networks learn increasingly refined representation of input data, with the final convolutional layer output serving as a compressed representation.

Autoencoders are introduced as an architecture designed to address the loss of information by capturing and compressing input data into a lower-dimensional space.

An autoencoder consist of two main components:

an Encoder the compress the input into a code, and a decoder the reconstructs the original input from this code.

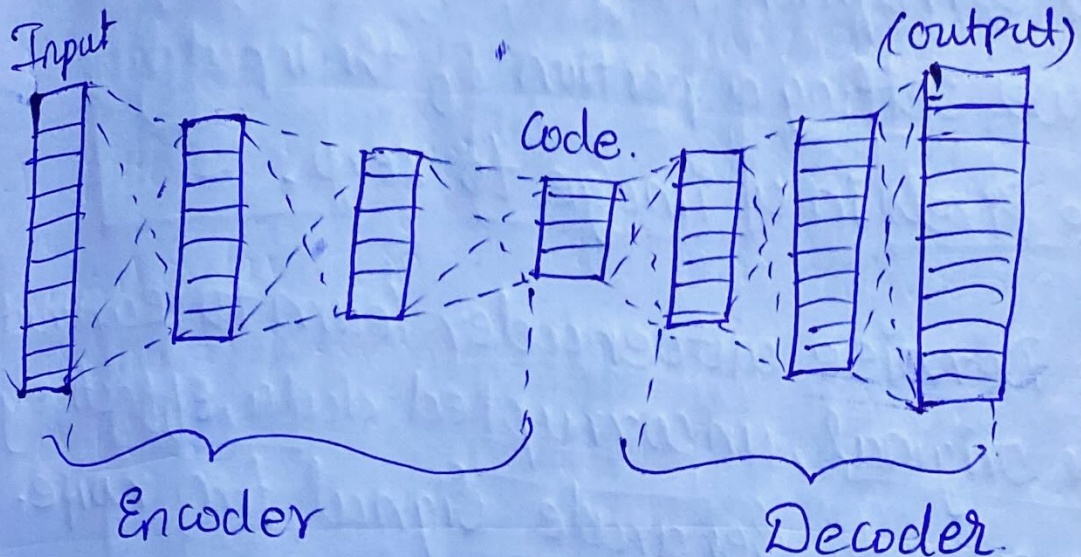


Unlike the feed-forward networks that map inputs to arbitary labels, autoencoder focus on reconstructing the input, thereby retaining more information.

The encoder part of the autoencoder learns to identify and encode the most critical features of the input data.

The decoder part learns to Revere the encoding process to reconstruct the input data from the encoded representation

Autoencoders can be used to generate lower dimensional data Representation without the supervision required in traditional methods.



Denoising of autoencoder

Denoising:

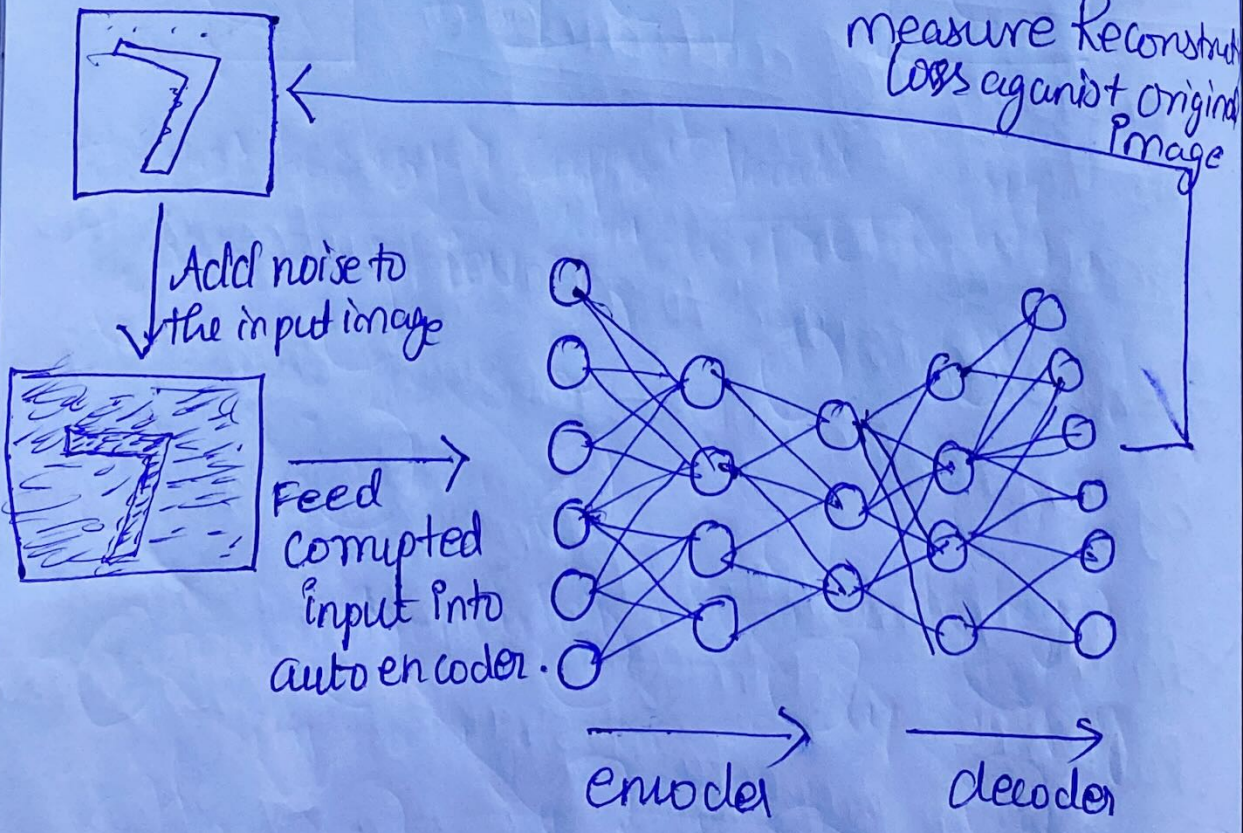
Denoising is used to enhance autoencoders by generating noise-resistant Embedding.

In the human Perception we can recognize Patterns and images despite significant noise, a property desirable for ML models we can even distinguish 8 and 7 if all the pixels got corrupted easily.

The Denoising encoders are trained using corrupted inputs, forcing them to learn robust features.

The corruption process in denoising involves setting a portion of the input data, like Pixels in a image to zero.

The Denoise autoencoder learn to reconstruct the original, uncorrupted data, effectively learning to separate signal from noise.

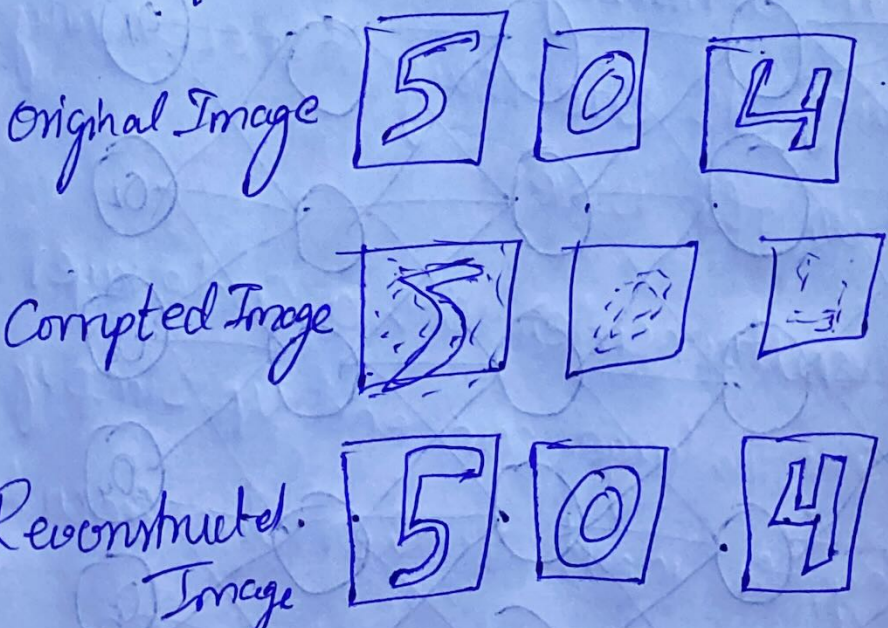


The geometric interpretation involves the concept of a manifold, the shape within the data that the autoencoder learns to reconstruct.

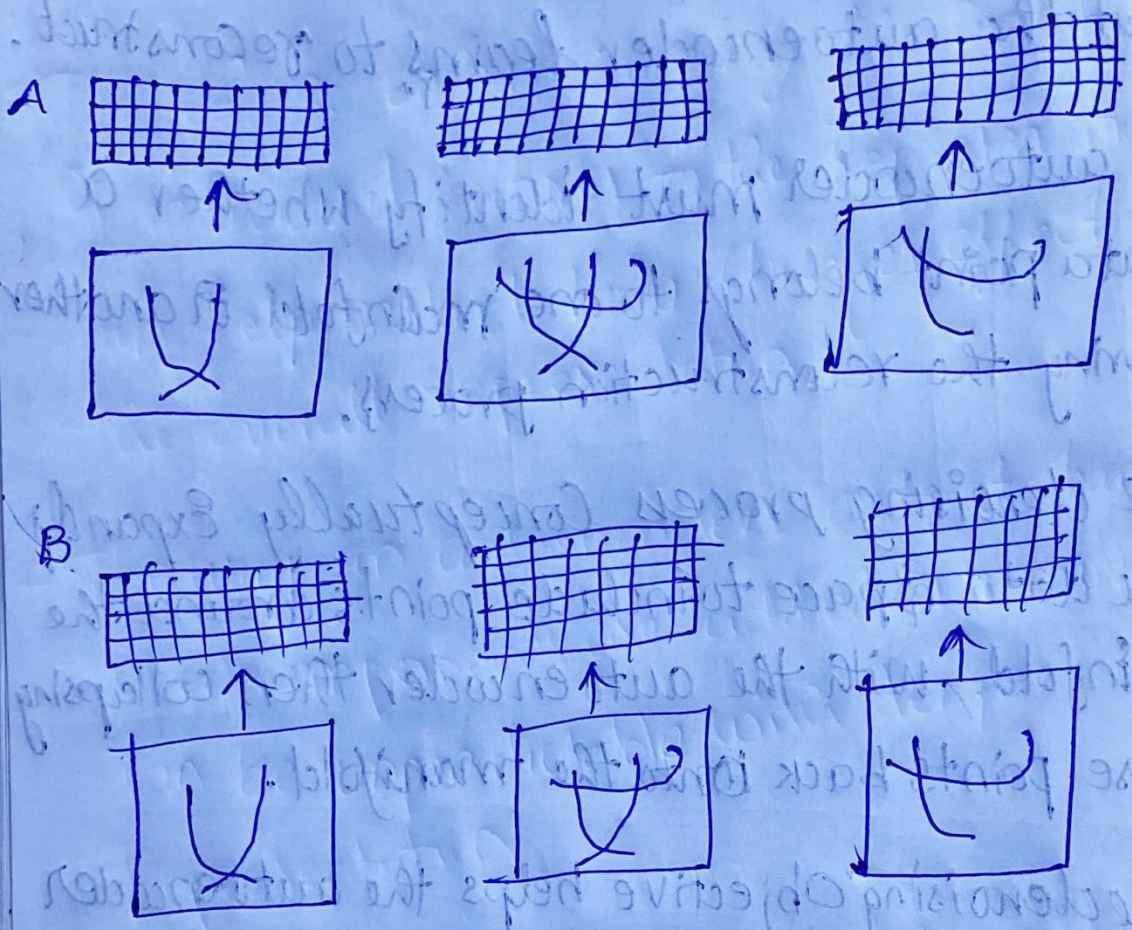
An autoencoder must identify whether a data point belongs to one manifold or another during the reconstruction process.

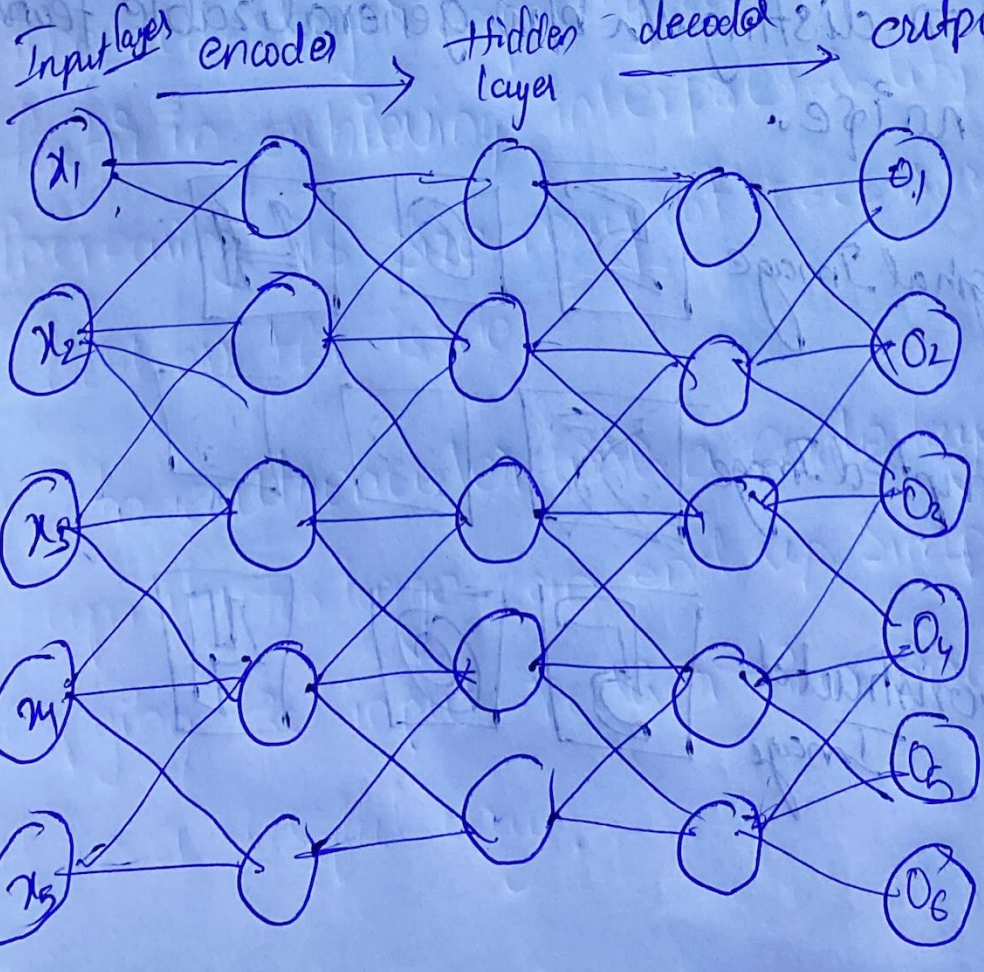
The denoising process conceptually expands the data space to include points around the manifold, with the autoencoder then collapsing these points back onto the manifold.

The denoising objective helps the autoencoder learn to distinguish b/w generalizable features and noise.



Sparsity in Autoencoders:





Deep learning models struggle with interpretability due to their complexity and non linearity. Deep learning models are generally very difficult to interpert because of the non linearities and massive number of paleameters that make up a model.

The Interpereta bility can be solved by auto-encoder. In general, an autoencoder Representations are dense, which means that many features are combined in ways that are diffcult to separate or interpret.

In figA. The activation of a dense representation combine and overlay information from multiple features in ways that are diffcult to interpret.

Introducing Sparsity into autoencoder representation can help make the model output more interpretable.

The goal is to achieve a one-to-one corresponding b/w the high level features in the input

data and the components in the auto-encoder code.

→ The Fig B. colour-coding the contributions of individual features to the Representation can help understand how changes effect the overall representation.

→ The way an autoencoders output changes when components are added or removed can be unpredictable with dense Representation.

→ The code's layer capacity can be limiting factor in achieving interpretability, increasing its size may not necessarily address the issue.

→ An autoencoder with a code layer capacity too large can end up simply copying the input rather than encoding it in a useful way.

→ The Autoencoder can effectively summarize data points when all relevant info is present in the individual data points themselves.