Experiment No: 04

Aim: To implement Autoencoder for anomaly detection.

Problem Statement:

Use Autoencoder to implement anomaly detection. Build the model by using:

a. Import required libraries

b. Upload / access the dataset

c. Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

e. Compile the models with Optimizer, Loss, and Evaluation Metrics.

Objectives:

a) Apply Autoencoder deep learning architecture to determine anomalies in input dataset

b) Evaluate Model.

Theory:

Autoencoders are very useful in the field of unsupervised machine learning. You can use them to compress the data and reduce its dimensionality. If anyone needs the original data can reconstruct it from the compressed data using an Autoencoder. Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip. So we can't expect an Autoencoder trained on handwritten digits to compress landscape photos.
- Lossy: The output of the Autoencoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression they are not the way to go.
- Unsupervised: To train an Autoencoder we don't need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an unsupervised learning technique since

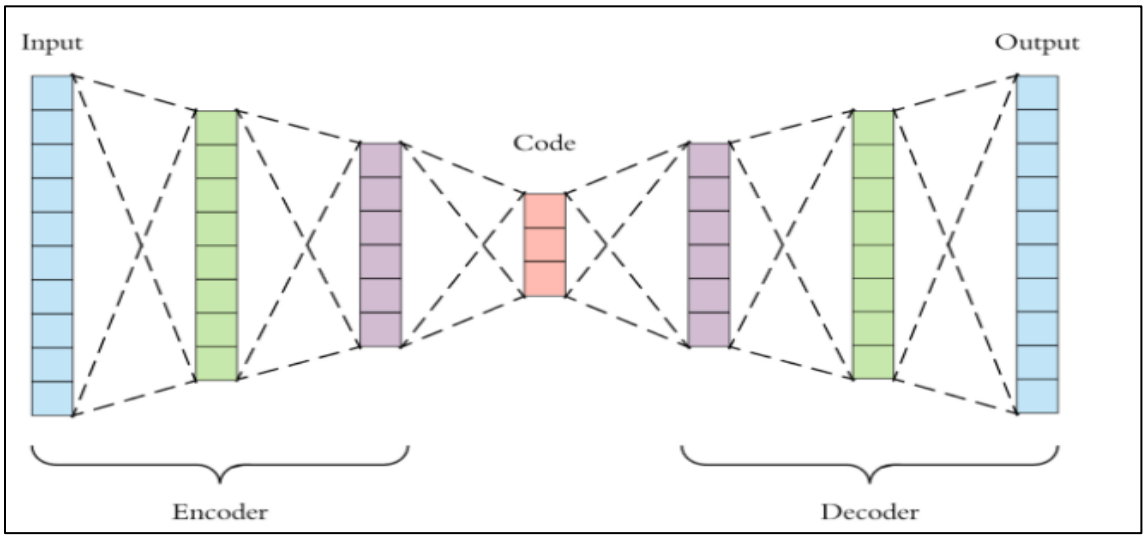
they don’t need explicit labels to train on. But to be more precise they are self-supervised because they generate their own labels from the training data.

Architecture

An Autoencoder is a type of neural network that can learn to reconstruct images, text, and other data from compressed versions of themselves. An Autoencoder consists of three layers:

1. Encoder
2. Code
3. Decoder

The Encoder layer compresses the input image into a latent space representation. It encodes the input image as a compressed representation in a reduced dimension. The compressed image is a distorted version of the original image.



The Code layer represents the compressed input fed to the decoder layer. The decoder layer decodes the encoded image back to the original dimension. The decoded image is reconstructed from latent space representation, and it is reconstructed from the latent space representation and is a lossy reconstruction of the original image.

There are 4 hyperparameters that we need to set before training an Autoencoder:

- Code size: number of nodes in the middle layer. Smaller size results in more compression.
- Number of layers: the Autoencoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.

- Number of nodes per layer: the Autoencoder architecture is called a stacked Autoencoder since the layers are stacked one after another. The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.
- Loss function: we either use mean squared error (MSE) or binary cross entropy. If the input values are in the range [0, 1] then we typically use cross entropy, otherwise we use the mean squared error.

Training of Autoencoder

Training of an Auto-encoder for data compression: For a data compression procedure, the most important aspect of the compression is the reliability of the reconstruction of the compressed data. This requirement dictates the structure of the Auto-encoder as a bottleneck.

Step 1: Encoding the input data The Auto-encode first tries to encode the data using the initialized weights and biases.

Step 2: Decoding the input data The Auto-encoder tries to reconstruct the original input from the encoded data to test the reliability of the encoding.

Step 3: Backpropagating the error after the reconstruction, the loss function is computed to determine the reliability of the encoding. The error generated is backpropagated.

The above-described training process is reiterated several times until an acceptable level of reconstruction is reached.

After the training process, only the encoder part of the Auto-encoder is retained to encode a similar type of data used in the training process. The different ways to constrain the network are:-

- Keep small Hidden Layers: If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.
- Regularization: In this method, a loss term is added to the cost function which encourages the network to train in ways other than copying the input.
- Denoising: Another way of constraining the network is to add noise to the input and teach the network how to remove the noise from the data.

- Tuning the Activation Functions: This method involves changing the activation functions of various nodes so that a majority of the nodes are dormant thus effectively reducing the size of the hidden layers.

The different variations of Auto-encoders are:-

- Denoising Auto-encoder: This type of auto-encoder works on a partially corrupted input and trains to recover the original undistorted image. As mentioned above, this method is an effective way to constrain the network from simply copying the input.
- Sparse Auto-encoder: This type of auto-encoder typically contains more hidden units than the input but only a few are allowed to be active at once. This property is called the sparsity of the network. The sparsity of the network can be controlled by either manually zeroing the required hidden units, tuning the activation functions or by adding a loss term to the cost function
- Variational Auto-encoder: This type of auto-encoder makes strong assumptions about the distribution of latent variables and uses the Stochastic Gradient Variational Bayes estimator in the training process.

Conclusion:

Autoencoders can be used as an anomaly detection algorithm when we have an unbalanced dataset where we have a lot of good examples and only a few anomalies. Autoencoders are trained to minimize reconstruction error. When we train the Autoencoders on normal data or good data, we can hypothesis that the anomalies will have higher reconstruction errors than the good or normal data.