SANT DNYANESHWAR SHIKSHAN SANSTHA'S
ANNASAHEB DANGE COLLEGE OF ENGINEERING & TECHNOLOGY, ASHTA

DEPARTMENT OF ARTIFICIAL INTELLIENCE AND DATA SCIENCE

LABORATORY MANUAL

Deep Learning

Mr. Vikas Honmane

THIRD YEAR B.TECH

Academic Year

2024-25

Experiments List

Experiments :	
1.	Designing and developing a model for Autoencoder for Dimensionality reduction
2.	Implement character and Digit Recognition using ANN.
3.	Implement the analysis of X-ray images using autoencoders
4.	Implement Speech Recognition using NLP, For Home automation
5.	Develop a code to design object detection and classification for traffic analysis using CNN
6.	Designing and developing a model for Text generation using LSTM
7.	Implement image augmentation using deep RBM.
8.	Designing and developing a model for Autoencoder for classification
9.	Micro Project: Number plate recognition of traffic video analysis.

DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.
2. All the students should sit according to their roll numbers starting from their left to right.
3. All the students are supposed to enter the Presence in the log book.
4. Do not change the Computer on which you are working.
5. All the students are expected to get at least the algorithm of the program/concept to be implemented.
6. Strictly observe the instructions given by the teacher/Lab Instructor.
7. Do not disturb machine Hardware / Software Setup.

Instruction for Laboratory Teachers

1. Submission related to whatever lab work has been completed should be done during the next lab session along with signing the index.

2. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

3. Continuous assessment in the prescribed format must be followed.

Expt. No. 1	Designing and developing a model for Autoencoder for Dimensionality reduction
Date :	

Aim: To study and demonstrate basics of Autoencoder for Dimensionality reduction on MNIST dataset

Purpose:

 Reduce the number of features in a dataset while retaining its significant structures and patterns.

Objectives:

1. Explain working of Autoencoders.
2. Design autoencoder for dimensionality reduction on MNIST dataset.
3. Implement denoising autoencoder.

Equipment's /Components / Software:

S.No	Name
1	Google colab

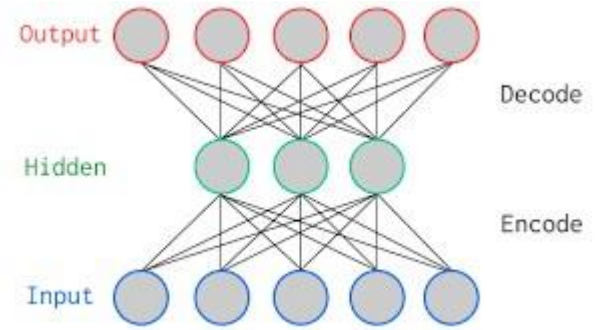
Theory:

Autoencoder

an autoencoder is an unsupervised machine learning algorithm that takes an image as input and tries to reconstruct it using fewer number of bits from the bottleneck also known as latent space. The image is majorly compressed at the bottleneck. The compression in autoencoders is achieved by training the network for a period of time and as it learns it tries to best represent the input image at the bottleneck. The general image compression algorithms like JPEG and JPEG lossless compression techniques compress the images without the need for any kind of training and do fairly well in compressing the images.

Autoencoders are similar to dimensionality reduction techniques like Principal Component Analysis (PCA). They project the data from a higher dimension to a lower dimension using linear transformation and try to preserve the important features of the data while removing the non-essential parts.

However, the major difference between autoencoders and PCA lies in the transformation part: as you already read, PCA uses linear transformation whereas autoencoders use non-linear transformations.

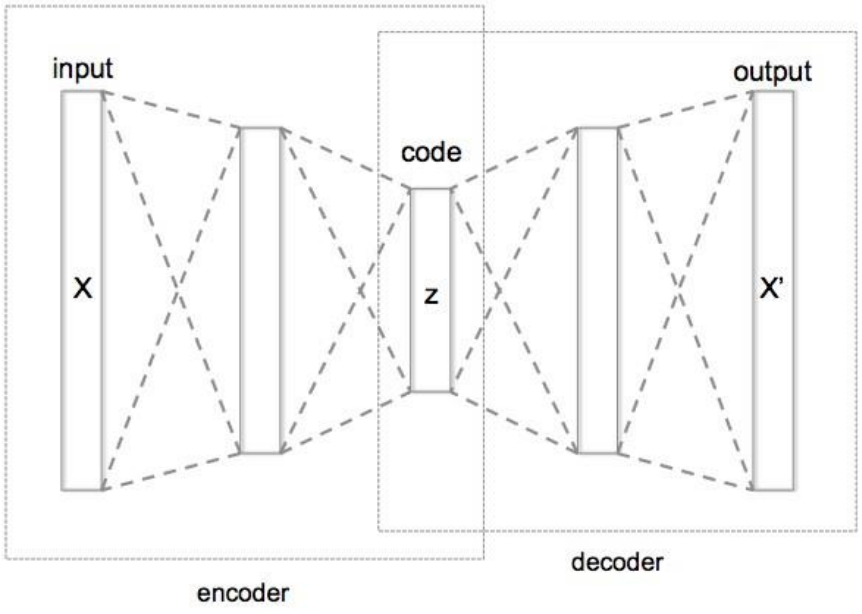


The above figure is a two-layer vanilla autoencoder with one hidden layer. In deep learning terminology, you will often notice that the input layer is never taken into account while counting the total number of layers in an architecture. The total layers in an architecture only comprises of the number of hidden layers and the ouput layer.

As shown in the image above, the input and output layers have the same number of neurons.

Let's take an example. You feed an image with just five pixel values into the autoencoder which is compressed by the encoder into three pixel values at the bottleneck (middle layer) or latent space. Using these three values, the decoder tries to reconstruct the five pixel values or rather the input image which you fed as an input to the network.

In reality, there are more number of hidden layers in between the input and the output.



Autoencoder can be broken in to three parts

- Encoder: this part of the network compresses or downsamples the input into a fewer number of bits. The space represented by these fewer number of bits is often called the *latentspace* or *bottleneck*. The bottleneck is also called the "maximum point of compression" since at this point the input is compressed the maximum. These compressed bits that represent the

  original input are together called an “encoding” of the input.

- Decoder: this part of the network tries to reconstruct the input using only the encoding of the input. When the decoder is able to reconstruct the input exactly as it was fed to the encoder, you can say that the encoder is able to produce the best encodings for the input with which the decoder is able to reconstruct well!

There are variety of autoencoders, such as the convolutional autoencoder, denoising autoencoder, variational autoencoder and sparse autoencoder. However, as you read in the introduction, you'll only focus on the convolutional and denoising ones in this tutorial

Code and Result:

```
# import libraries
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F

import matplotlib.pyplot as plt
import matplotlib_inline.backend_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
```

Import and process the data

```
# import dataset (comes with colab!)
data = np.loadtxt(open('sample_data/mnist_train_small.csv','rb'),delimiter=',')

# don't need labels!
data = data[:,1:]

# normalize the data to a range of [0 1]
dataNorm = data / np.max(data)

# convert to tensor
```

```
dataT = torch.tensor( dataNorm ).float()
```

Create the DL model

```
# create a class for the model
def createTheMNISTAE():

  class aenet(nn.Module):
    def __init__(self):
      super().__init__()

      ### input layer
      self.input = nn.Linear(784,250)

      ### encoder layer
      self.enc = nn.Linear(250,50)

      ### latent layer
      self.lat = nn.Linear(50,250)

      ### decoder layer
      self.dec = nn.Linear(250,784)

    # forward pass
    def forward(self,x):
      x = F.relu( self.input(x) )
      x = F.relu( self.enc(x) )
      x = F.relu( self.lat(x) )
      y = torch.sigmoid( self.dec(x) )
      return y

  # create the model instance
  net = aenet()

  # loss function
  lossfun = nn.MSELoss()

  # optimizer
  optimizer = torch.optim.Adam(net.parameters(),lr=.001)

  return net,lossfun,optimizer
# test the model with a bit of data
net,lossfun,optimizer = createTheMNISTAE()

X = dataT[5:10,:]
yHat = net(X)

print(X.shape)
print(yHat.shape)
# let's see what the model did!
```

```
fig,axs = plt.subplots(2,5,figsize=(10,3))

for i in range(5):
  axs[0,i].imshow(X[i,:].view(28,28).detach() ,cmap='gray')
  axs[1,i].imshow(yHat[i,:].view(28,28).detach() ,cmap='gray')
  axs[0,i].set_xticks([]), axs[0,i].set_yticks([])
  axs[1,i].set_xticks([]), axs[1,i].set_yticks([])

plt.suptitle('Yikes!!!')
plt.show()
```

Create a function that trains the model

```
def function2trainTheModel():

  # number of epochs
  numepochs = 10000

  # create a new model
  net,lossfun,optimizer = createTheMNISTAE()

  # initialize losses
  losses = torch.zeros(numepochs)

  # loop over epochs
  for epochi in range(numepochs):

    # select a random set of images
    randomidx = np.random.choice(dataT.shape[0],size=32)
    X = dataT[randomidx,:]

    # forward pass and loss
    yHat = net(X)
    loss = lossfun(yHat,X)

    # backprop
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # losses in this epoch
    losses[epochi] = loss.item()
  # end epochs

  # function output
  return losses,net
```

Run the model and show the results!

```
# train the model (90s)
losses,net = function2trainTheModel()
print(f'Final loss: {losses[-1]:.4f}')

# visualize the losses
```

```
plt.plot(losses,'.-')
plt.xlabel('Epochs')
plt.ylabel('Model loss')
plt.title('OK, but what did it actually learn??')
plt.show()
# Repeat the visualization when testing the model
X = dataT[:5,:]
yHat = net(X)

# let's see what the model did!
fig,axs = plt.subplots(2,5,figsize=(10,3))

for i in range(5):
  axs[0,i].imshow(X[i,:].view(28,28).detach() ,cmap='gray')
  axs[1,i].imshow(yHat[i,:].view(28,28).detach() ,cmap='gray')
  axs[0,i].set_xticks([]), axs[0,i].set_yticks([])
  axs[1,i].set_xticks([]), axs[1,i].set_yticks([])

plt.suptitle('Disregard the yikes!!!')
plt.show()
Add noise to see a use case of an autoencoder
# grab a small set of images
X = dataT[:10,:]

# add noise
Xnoise = X + torch.rand_like(X)/4

# clip at 1
Xnoise[Xnoise>1] = 1

# show the noisy images
fig,axs = plt.subplots(2,5,figsize=(10,3))

for i in range(5):
  axs[0,i].imshow(X[i,:].view(28,28).detach() ,cmap='gray')
  axs[1,i].imshow(Xnoise[i,:].view(28,28).detach() ,cmap='gray')
  axs[0,i].set_xticks([]), axs[0,i].set_yticks([])
  axs[1,i].set_xticks([]), axs[1,i].set_yticks([])

plt.show()

# show the noisy images
fig,axs = plt.subplots(3,10,figsize=(12,5))

for i in range(10):
  axs[0,i].imshow(X[i,:].view(28,28).detach() ,cmap='gray')
  axs[1,i].imshow(Xnoise[i,:].view(28,28).detach() ,cmap='gray')
  axs[2,i].imshow(Y[i,:].view(28,28).detach() ,cmap='gray')
  axs[0,i].set_xticks([]), axs[0,i].set_yticks([])
  axs[1,i].set_xticks([]), axs[1,i].set_yticks([])
```

```
  axs[2,i].set_xticks([]), axs[2,i].set_yticks([])

plt.suptitle('Neato.')
plt.show()
```

Conclusion:

The autoencoder successfully captured the underlying structure of the input data, enabling effective reconstruction with minimal loss.

Sr. No.	Question	CO Mapped
1	What are the primary components of an autoencoder, and how do they contribute to the model's functionality?	1ADPC314_1
2	How does an autoencoder differ from other dimensionality reduction techniques such as PCA (Principal Component Analysis)?	1ADPC314_4
3	What are some common applications of autoencoders in various fields, and how do they benefit from the use of autoencoders?	1ADPC314_4
4	How do different types of autoencoders, such as variational autoencoders (VAEs) and convolutional autoencoders, enhance the capabilities of standard autoencoders?	1ADPC314_4
5	What are the main challenges associated with training autoencoders, and how can they be addressed to improve model performance and generalization?	1ADPC314_4

Student Activity:

Implement the code, write theory ,come with Printout of that	

Other Links:

https://www.geeksforgeeks.org/implementing-an-autoencoder-in-pytorch/