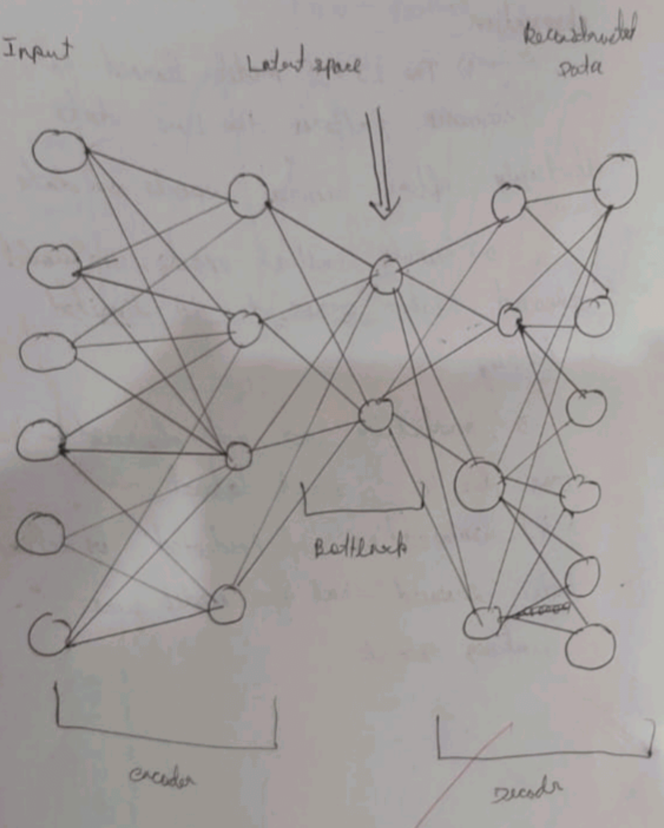
Architecture of Autoencoder



Ex:10 Perform compression on MNIST Dataset using Autoencoder

Aim:

To implement an autoencoder using Pytorch for compressing and reconstructing images from the MNIST dataset, demonstrating unsupervised feature learning and dimensionality reduction.

objectives:

1. To understand the working Principle of an Autoencoder and its encoder - decoder structure,

2. To build and train a fully connected Autoencoder using Pytorch on the MNIST dataset

3. To evaluate the model ability to compress and reconstruct images.

4. To visualize the original and reconstructed image to analyze reconstruction quality

Output :

Epoch [1/10], Loss : 0.0605

Epoch [2/10], loss : 0.0322

Epoch [3/10], loss : 0.0262

Epoch [4/10], loss : 0.0212

Epoch [5/10], loss : 0.0196

⋮

Epoch [10/10], loss : 0.0157

Pseudocode :

BEGIN

Import torch, torch.nn, torch.optim,

Load MNIST dataset with transfer

create data Loader for train test

Define autoencoder class :

Encoder :

Decoder :

Forward Pass :

Initialize model, MSE loss, adam optimizer

For each epoch :

For each Batch in train set

output = model (img)

loss = MSE (output, imgs)

Print epoch loss

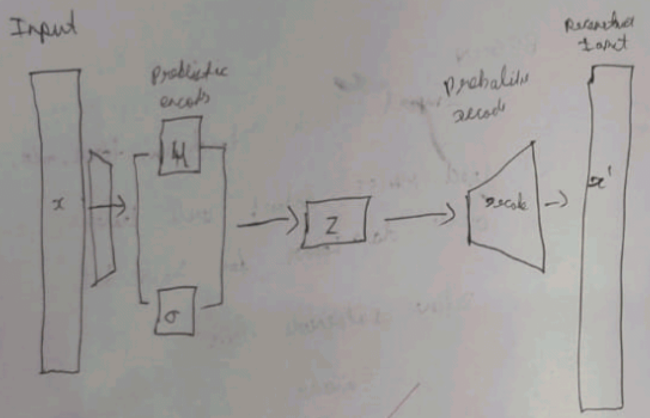
TEST model on sample img

END

RESULT :

successfully performed compression on MNIST using Autoencoder

Architecture of VAE



Experiment using variational Autoencoder (VAE)

AIM:

To implement a variation Auto encoder (VAE) using Pytorch for learning probablistic latent representation and generating new handwritten digit images from MNIST datasets.

objectives:

1. To understand the concept of VAE and how they differs from standard autoencoder.

2. To implement VAE model using Pytorch that learn a latest distribution

3. To reconstruct and genrate new images using the learned latent space.

4. To evaluate the genration capability of the trained VAE

output:

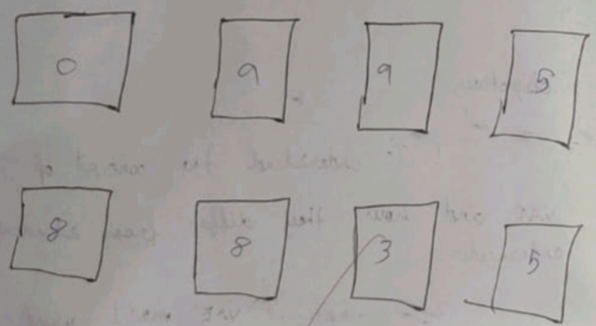
Epoch [1/10]; loss : 104.4406

Epoch [2/10]; loss . 121.4266

Epoch [3/10], loss : 111.4229

Epoch [4/10], loss : 119.7214

Epoch [5/10], loss : 108.5538



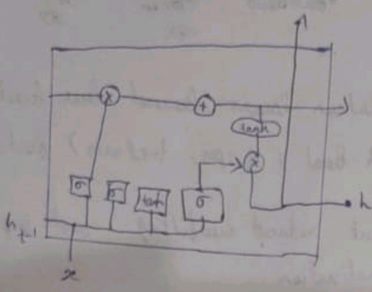
Pseudocode:

```
BEGIN
    Import torch, torch.nn, torch.optim
    Load MNIST data
    DEFINE VAE class:
        Encode:
        Reparameterization
        Decoder:
        Forward:
    Define loss Function:
        Reconstruct loss
    Initialize model, optimizer (Adam)
    For each epoch
        For each builds:
            recon, M, logvar = model (img)
            loss ~ BCE + KLD
        Print average epoch loss
    Test model
        display generated sample
END
```

result

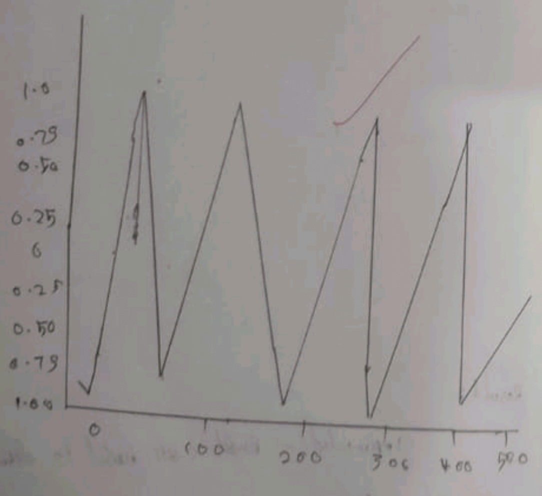
successfully experimented using variational Autoencoder.

LSTM Architecture:



Out put

LSTM Prediction vs Actual value



Ex: 8 using LSTM algorithm

Aim:

To implement the LSTM algorithm

Algorithm:

* LSTM is a type of recurrent neural network
* capable of learning long term dependencies, especially in sequential data
* It uses gets to control the flow of information

Pseudo code:

1) Load the dataset
2) preprocess the dataset
   a) Normalize the value
   b) convert the data into sequences
3) split the dataset into training and testing sets
4) Define the LSTM model

[1]

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# ==============================
# 1. Load and Prepare MNIST Data
# ==============================
transform = transforms.Compose([
    transforms.ToTensor(),
])

train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)

# ==============================
# 2. Define Autoencoder Model
# ==============================
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Encoder: compress input (28x28 → 64 features)
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 256),
            nn.ReLU(True),
            nn.Linear(256, 64),
            nn.ReLU(True)
        )
```

```python
    'features': compressed_features,
    'labels': labels
}, 'mnist_compressed.pt')

print("Compressed features saved to mnist_compressed.pt")
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 62.2MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.65MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 14.3MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 6.73MB/s]
Epoch [1/10], Loss: 0.0425
Epoch [2/10], Loss: 0.0165
Epoch [3/10], Loss: 0.0120
Epoch [4/10], Loss: 0.0100
Epoch [5/10], Loss: 0.0088
Epoch [6/10], Loss: 0.0079
Epoch [7/10], Loss: 0.0072
Epoch [8/10], Loss: 0.0067
Epoch [9/10], Loss: 0.0063
Epoch [10/10], Loss: 0.0060
```


```
Compressed features saved to mnist_compressed.pt
```

Commands + Code + Text Run all

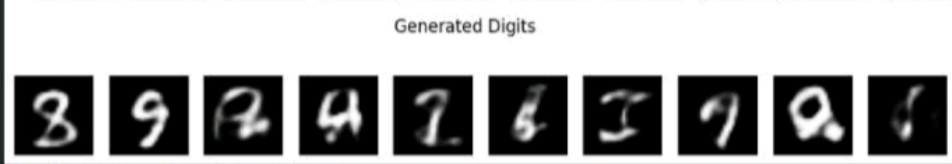
RAM Disk

[?] 2m

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# ======================================
# 1. Load and Prepare the MNIST Dataset
# ======================================
transform = transforms.Compose([
    transforms.ToTensor(),
])

train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ======================================
# 2. Define Variational Autoencoder (VAE)
# ======================================
class VAE(nn.Module):
    def __init__(self, latent_dim=20):
        super(VAE, self).__init__()
        self.latent_dim = latent_dim

        # Encoder: 784 -> 400 -> μ, log(σ²)
        self.fc1 = nn.Linear(28*28, 400)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)
```

```
}, 'mnist_vae_latent.pt')

print("Latent features saved to mnist_vae_latent.pt")
```

```
Epoch [1/10]  Loss: 162.8189
Epoch [2/10]  Loss: 120.8224
Epoch [3/10]  Loss: 114.1165
Epoch [4/10]  Loss: 111.2614
Epoch [5/10]  Loss: 109.5371
Epoch [6/10]  Loss: 108.3486
Epoch [7/10]  Loss: 107.5457
Epoch [8/10]  Loss: 106.9098
Epoch [9/10]  Loss: 106.4470
Epoch [10/10]  Loss: 106.0076
```




```
Latent features saved to mnist_vae_latent.pt
```