Experiments Using Variational
17/10/2025. Lab 11 : Autoencoder (VAE)

Aim : To implement a Variational Autoencoder (VAE) and study its generative ability to reconstruct

Objectives

1) To understand the concept and working of a VAE

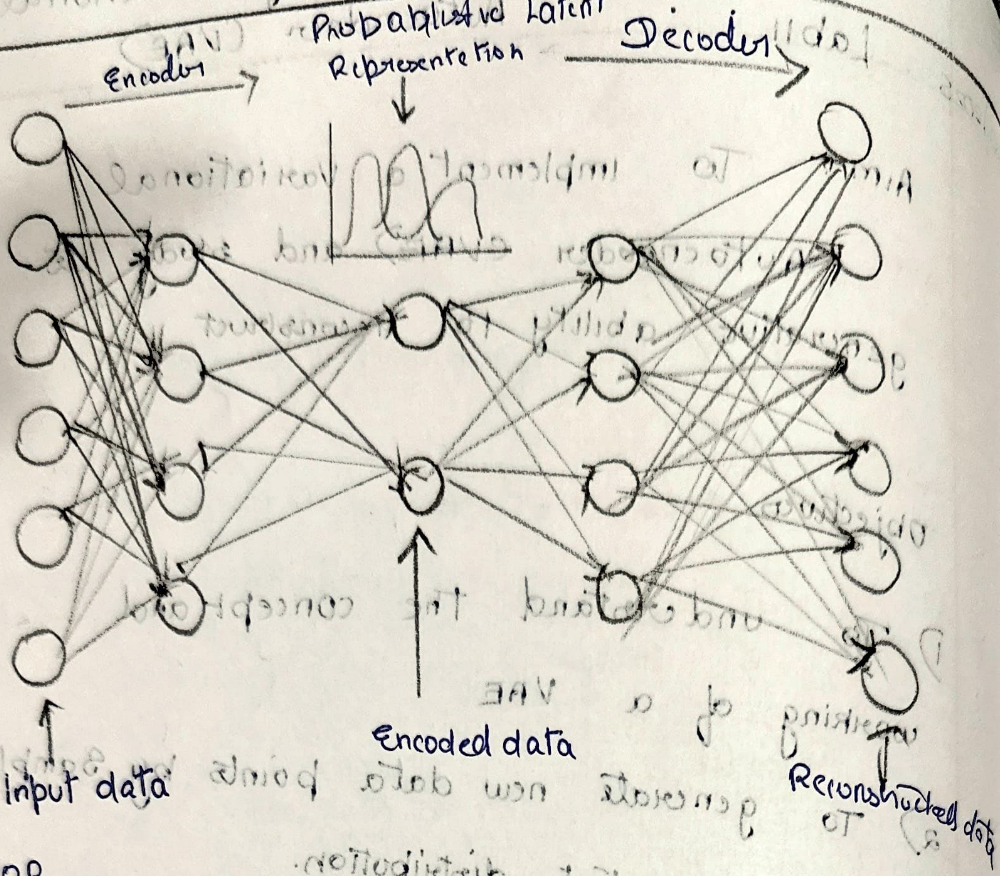
2) To generate new data points by sampling from a latent distribution.

3) To visualize reconstructed & newly generated images

4) To compare the performance of VAE

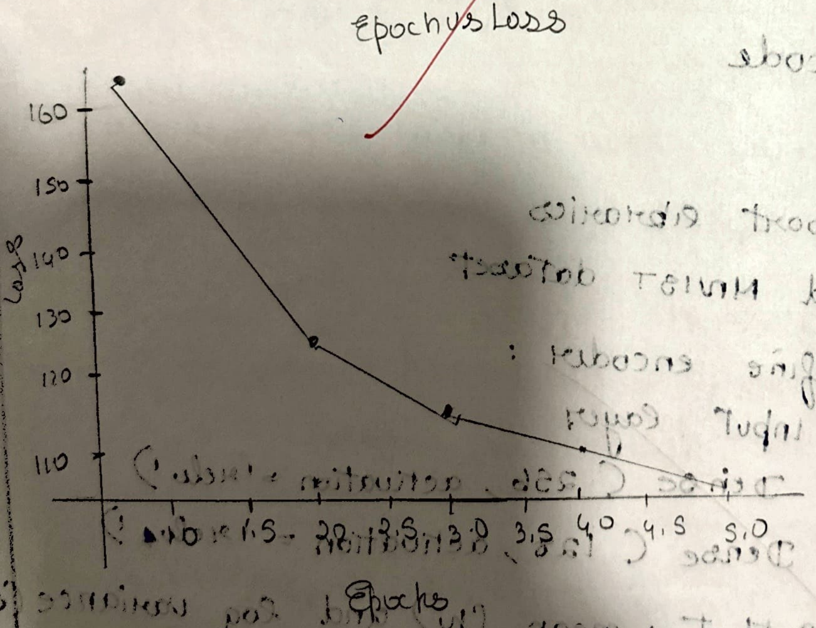
Pseudo code

Start
1) Import libraries
2) Load MNIST dataset
3) Define encoder :
   - Input layer
   - Dense (256, activation = 'relu')
   - Dense (128, activation = 'relu')
   - output : mean (μ) and log variance (σ^2)
4) Sample latent
   $z = \mu + \sigma * \varepsilon$, where $\varepsilon \sim N(0, 1)$
5) Define Decoder :
   - Input



OP

Epoch [1/5], Loss : 164.0216
Epoch [2/5], Loss : 121.5716
Epoch [3,5], Loss : 114.6072
Epoch [4,5], Loss : 111.6099
Epoch [5,5], Loss : 109.8843



6) Define Total loss :

Total loss = Reconstruction loss + KL Divergence loss

7) Compile and train model using MNIST dataset

8) Visualize

End

Observation - ~~Fast~~

1) The autoencoder effectively reduced 784 dimensional input images into a 32-dimensional latent representation

2) The training loss decreased steadily with each epoch

3) Reconstructed images were visually similar to originals

4) The encoder captured essential patterns like stroke and shapes of handwritten digits

5) The model achieved efficient image compression.

6) The latent space representation could be used for feature extraction / or dimensionality reduction in other ML Tasks.

The VAE demonstrated better generative capability than traditional autoencoder.

Result

Successfully implemented VAE.

```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
```

```python
batch_size = 128
learning_rate = 1e-3
num_epochs = 5
latent_dim = 20
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
transform = transforms.ToTensor()
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```python
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        # Encoder
        self.fc1 = nn.Linear(28*28, 400)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)
        # Decoder
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 28*28)

    def encode(self, x):
        h1 = torch.relu(self.fc1(x))
        return self.fc_mu(h1), self.fc_logvar(h1)

    def reparameterize(self, mu, logvar):
```

[14]
✓ 0s

```
        return self.decode(z), mu, logvar
```

[15]
✓ 0s

```
def loss_function(recon_x, x, mu, logvar):
    BCE = nn.functional.binary_cross_entropy(recon_x, x.view(-1, 28*28), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD
```

[16]
✓ 0s

```
model = VAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

[17]
✓ 1m

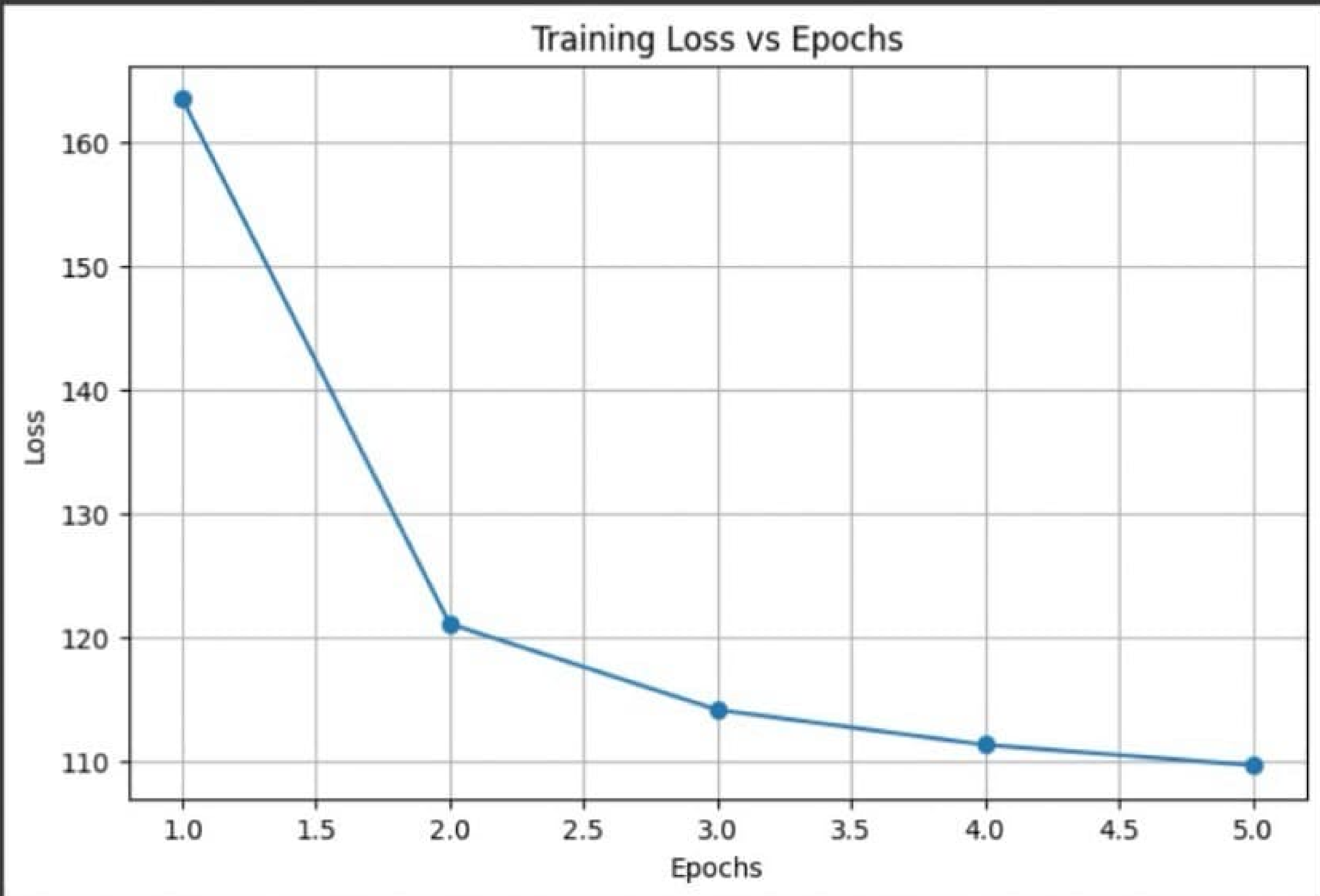
```
train_losses = []

for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for data, _ in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    avg_loss = train_loss / len(train_loader.dataset)
    train_losses.append(avg_loss)
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}')
```

```
Epoch [1/5], Loss: 163.4689
Epoch [2/5], Loss: 121.1113
Epoch [3/5], Loss: 114.1747
Epoch [4/5], Loss: 111.3527
Epoch [5/5], Loss: 109.7027
```

```
plt.figure(figsize=(8,5))
plt.plot(range(1, num_epochs+1), train_losses, marker='o')
plt.title("Training Loss vs Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```



```
n = 10
plt.figure(figsize=(20,4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i+1)
    plt.imshow(data[i].cpu().squeeze(), cmap='gray')
    plt.axis('off')
    # Reconstructed
    ax = plt.subplot(2, n, i+1+n)
    plt.imshow(recon[i].cpu().view(28,28), cmap='gray')
    plt.axis('off')
plt.suptitle("Top: Original | Bottom: Reconstructed", fontsize=14)
plt.show()
```