EX. NO: 11
17.10.25

EXPERIMENTS USING VARIATIONAL AUTO-ENCODERS

Aim

To implement a variational autoencoder and study its generative ability to reconstruct and generate new data samples using the MNIST dataset.

Objectives

To understand the concept and working of a VAE.

To generate new data points by sampling from a latent distribution

To visualize reconstructed and newly generated images.

To compare the performance of VAE with a basic autoencoder.

pseudocode

Start

Import Libraries (tensorflow / keras / numpy, matplotlib).

Load Mnist dataset and normalize images (0-1)

Define encoder

Input layer (784)

Dense (256, activation = relu)

Dense (128, activation = relu)

Output : mean (μ) and log variance (σ2)

Sample latent vector z using reparameterization.

$z = \mu + \sigma * \epsilon$, where $\varepsilon \sim N(0,1)$

Define Decoder

Input : Latent vector z

Dense (128, activation = relu)

Dense (256, activation = relu)

Dense (784, activation = sigmoid)

reconstructed output

Define total loss

Total Loss = Reconstruction Loss + KL divergence Loss

compile and train model using MNIST dataset

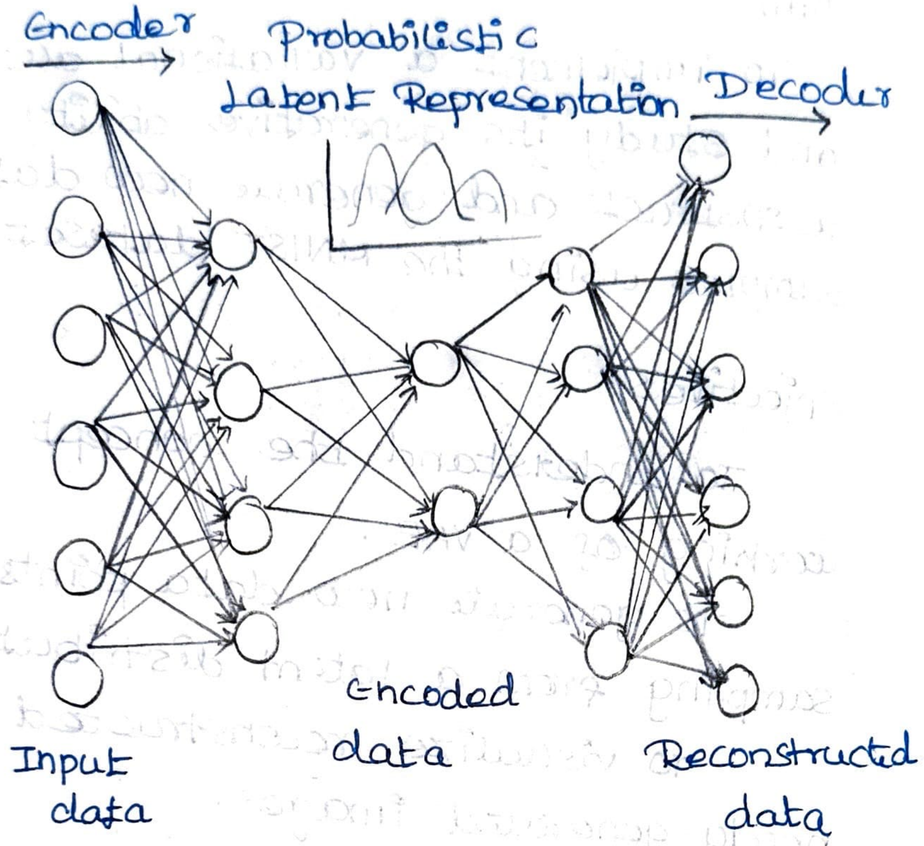
visualize

Original vs Reconstructed images

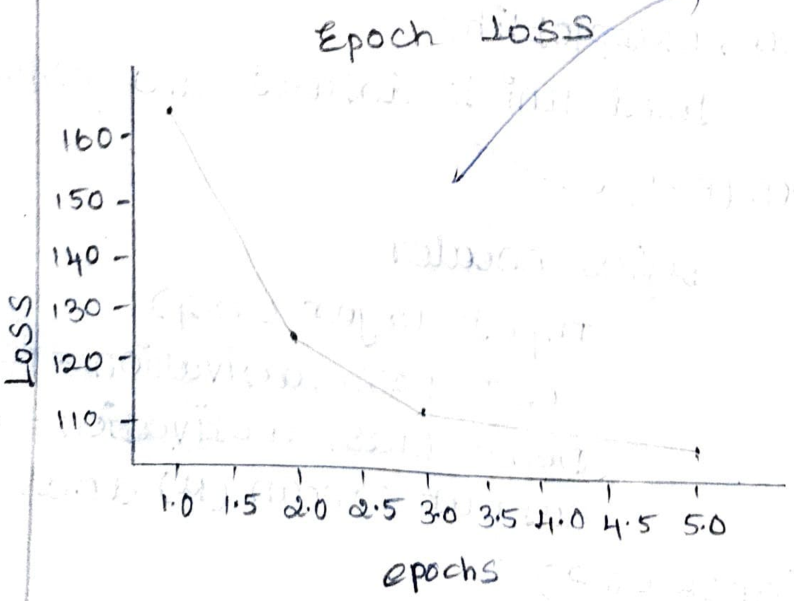
Observation

Training Dynamics

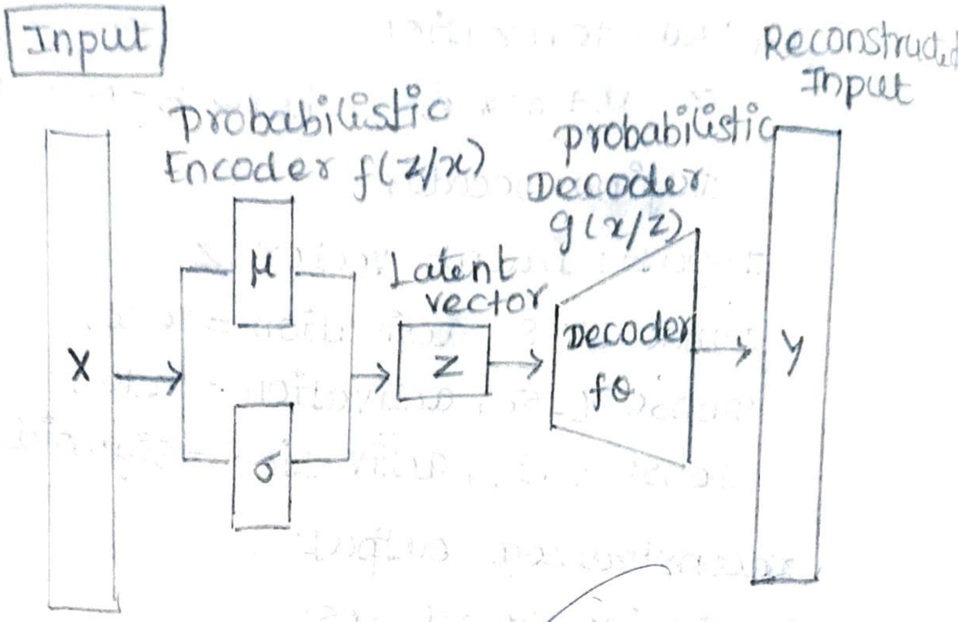
The VAE balanced accurate reconstructed with maintaining a continuous latent space. reconstruction loss decreased gradually while KL divergence regularized the model.



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



VAE ARCHITECTURE



Latent Space Representation

The Latent space followed a gaussian distribution, where similar digits clustered together, allowing smooth transitions between types

Reconstruction Quality

Reconstructed images were slightly blurred due to sampling but still recognizable, showing that key digit structures were learned.

Generative Ability

Random samples from the latent space produced realistic new digits, proving the VAE's generative capability.

Comparison with autoencoder

unlike basic autoencoder, the VAE learned a probabilistic model and could generate new samples, not just reconstruct inputs.

Result

To successfully implement using VAE

Untitled2.ipynb ☆ Changes will not be saved

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Copy to Drive

```
#lab 11
import torch, torch.nn as nn, torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# 1 Data
train = datasets.MNIST('data', train=True, transform=transforms.ToTensor(), download=True)
train_loader = DataLoader(train, batch_size=128, shuffle=True)

# 2 VAE Model
class VAE(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc_mu = nn.Linear(256, 64)
        self.fc_logvar = nn.Linear(256, 64)
        self.fc_dec1 = nn.Linear(64, 256)
        self.fc_out = nn.Linear(256, 784)

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

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h = torch.relu(self.fc_dec1(z))
        return torch.sigmoid(self.fc_out(h))

    def forward(self, x):
        x = x.view(-1, 784)
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

model = VAE()
opt = optim.Adam(model.parameters(), lr=0.001)
```

Untitled2.ipynb ☆ Changes will not be saved

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all Copy to Drive

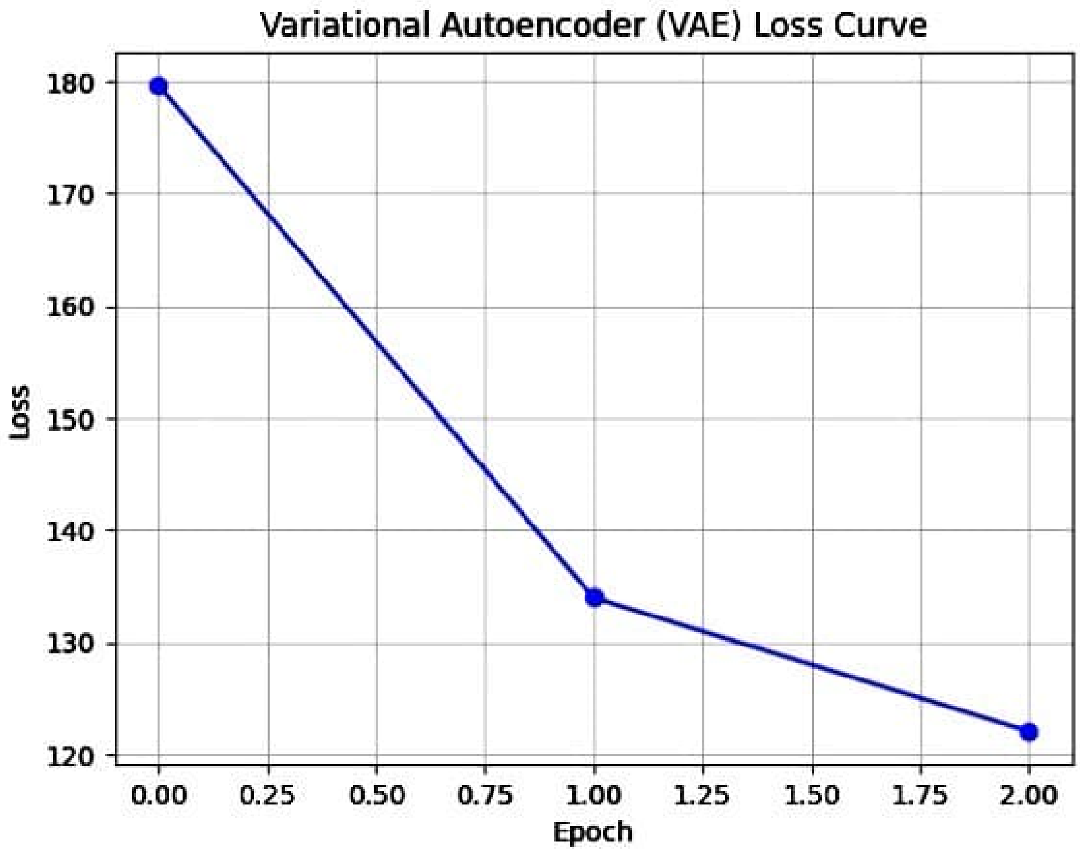
```
# 3 VAE Loss Function
def vae_loss(recon_x, x, mu, logvar):
    BCE = nn.functional.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return (BCE + KLD) / x.size(0)

# 4 Training
epochs, losses = 3, []
for epoch in range(epochs):
    run = 0
    for img, _ in train_loader:
        recon, mu, logvar = model(img)
        loss = vae_loss(recon, img, mu, logvar)
        opt.zero_grad()
        loss.backward()
        opt.step()
        run += loss.item()
    avg = run / len(train_loader)
    losses.append(avg)
    print(f"Epoch {epoch+1}/{epochs} | Loss: {avg:.4f}")

# 5 Plot Loss Curve
plt.plot(losses, 'o-', color='blue')
plt.title("Variational Autoencoder (VAE) Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()

# 6 Reconstructed Images
with torch.no_grad():
    sample = next(iter(train_loader))[0][:5]
    recon, _, _ = model(sample)
    for i in range(5):
        plt.subplot(2,5,i+1); plt.imshow(sample[i][0], cmap='gray'); plt.axis('off')
        plt.subplot(2,5,5+i+1); plt.imshow(recon[i].view(28,28), cmap='gray'); plt.axis('off')
plt.suptitle("Original (Top) vs Reconstructed (Bottom)")
plt.show()
```

```
Epoch 1/3 | Loss: 179.6595
Epoch 2/3 | Loss: 133.9765
Epoch 3/3 | Loss: 122.0765
```

Untitled2.ipynb Changes will not be saved

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Copy to Drive



Original (Top) vs Reconstructed (Bottom)