9/10/25

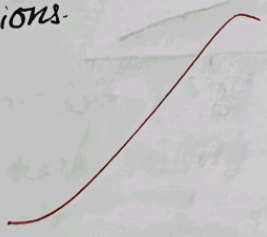
10. MNIST Compression using Autoencoders

Aim:

To perform image compression on MNIST dataset using an Autoencoder

Algorithm:

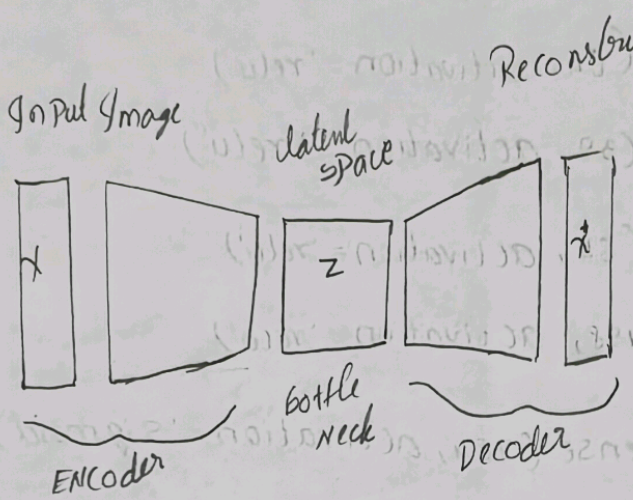
* Load and PreProcess data:
  Load MNIST dataset of handwritten digits
  Normalize Pixel values to range [0,1]
* Build the Autoencoder architecture
  Define the Encoder : reduce input using Dense layers
  Define the Decoder : reconstruct the input image from compressed latent vector
* Compile the Autoencoder:
  Use Adam optimizer and binary cross entropy loss
* Train on MNIST training data using input = output
* Encode and Decode
  Encoder to generator compressed representation
  Decoder to reconstruct images from these compressed representations.



Pseudocode:

```
BEGIN
    LOAD MNIST dataset
    NORMALIZE all pixel values to [0,1]
    FLATTEN each image into 784-dimensional vector

    SET encoding_dim = 32
    INPUT_LAYER = Input(shape = (784))
    ENCODE1 = Dense(128, activation = "relu")(INPUT_LAYER)
    ENCODE2 = Dense(64, activation = 'relu')(ENCODE1)
    ENCODE3 = Dense(encoding_dim, activation = 'relu')(ENCODE2)
    DECODE1 = Dense(64, activation = 'relu')(ENCODE3)
    DECODE2 = Dense(128, activation = 'relu')(DECODE1)
    CREATE autoencoder MODEL(INPUT LAYER → OUTPUT LAYER)
COMPILE MODEL Using Adam Optimizer and
    binary_crossentropy loss
TRAIN autoencoder on (X_train, X_train)
        for 50 epochs with batch_size = 256

ENCODE test images using encoder Part
DECODE them back using autoencoder
DISPLAY original and reconstructed images
COMPUTE compression_ratio = 784/encoding_dim

END
```

Autoencoder Architecture



Epoch(1) : Loss : 0.0378

Epoch(2) : loss : 0.0290

Epoch(3) : loss : 0.0229

Epoch(4) : Loss : 0.149

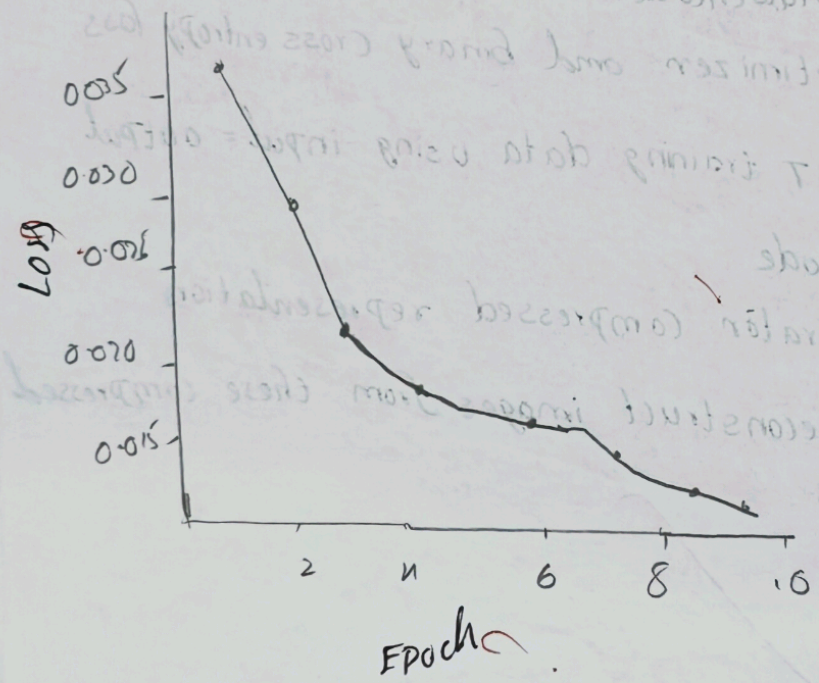
Epoch(5) : loss : 0.143

Epoch (8) : loss :

Epoch (9) : Loss : 0.0137

Epoch (10) : loss : 0.0133

Training loss over epochs:



Result:

Implementation of compression using autoencoders was done successfully

11. Variational Auto Encoders

Aim:

To implement and analyze the Performance of a variational Autoencoder for learning compressed latent representations of data.

Algorithm:

* Input: Training dataset 'x'.
* Initialize: Encoder and Decoder neural networks
* For each epochs

Pass input 'x' through Encoder → get μ, σ

sample $z = \mu + \sigma * \epsilon$

Pass z through Decoder → get reconstruction x'

compute:

Reconstruction loss : $\|x - x'\|^2$

KL divergence $= -0.5 \Sigma (1 + \log(\sigma^2) - \mu^2 - \sigma^2)$

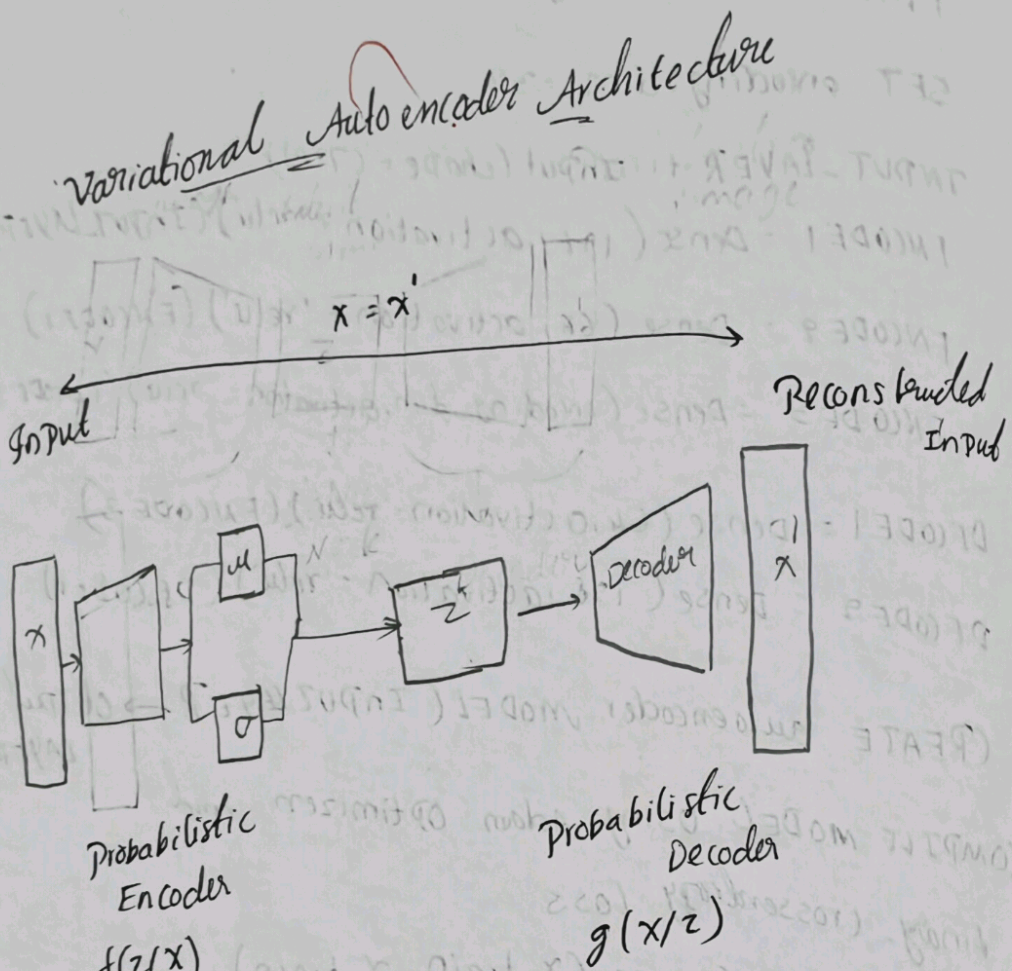
Total loss = Reconstruction + KL divergence

After training, Generate new samples

Pseudocode.

```
Input: Dataset x
set learning_rate = 0.001
set latent_dimension = L
Initialize Encoder network Parameters (w_enc, b_enc)
Intialize Decoder network Parameters (w_dec, b_dec)
FUNCTION ENCODER (x):
    h = ReLU (w1*x + b1)
    μ = wμ * h + bμ
    log σ^2 = wσ * h + bσ
    return μ, log σ^2

FUNCTION DECODER(z)
    h = ReLU( w3 * z + b3)
    return x
For each epoch in range ( num_epochs):
    For each batch ( x_batch) in Dataset x:
        μ, log σ^2 = ENCODER (x_batch)
        z = SAMPLE( μ, log σ^2)

        x' = DECODER (z)
        loss = loss (x_batch, x', μ, log σ^2)

        Back propogate loss

        Update encoder and decoder weights

    FUNCTION Generate ():
```

Variational Auto encoder Architecture



output:

Epoch 1 : Loss : 164.2524

Epoch 2 : Loss : 121.3644

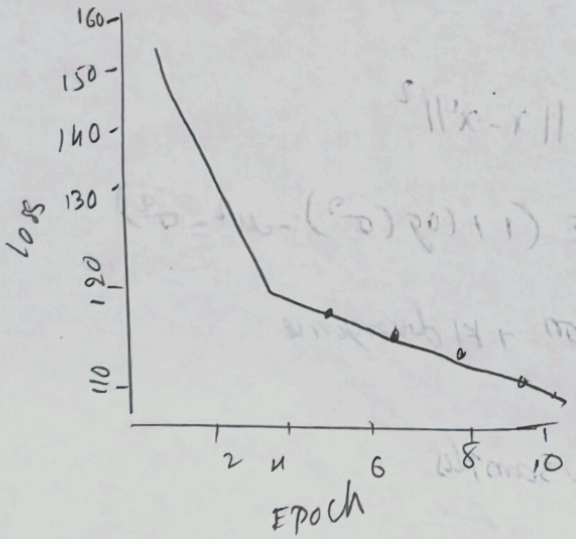
Epoch 3 : Loss : 114.3897

Epoch 4 : Loss : 111.3462

Epoch 5 : Loss : 109.5547

Avg Loss : 111.1155

Training over loss vs epochs

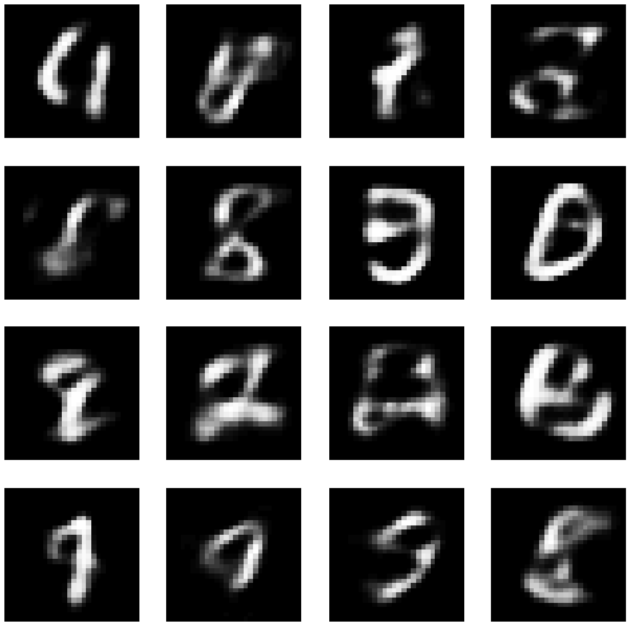


Result: Implementation of variation Autoencoders was done successfully

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

# Encoder
class Encoder(nn.Module):
    def __init__(self, input_dim=784, hidden_dim=400, latent_dim=20):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc_mu = nn.Linear(hidden_dim, latent_dim)
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)

    def forward(self, x):
        h = F.relu(self.fc1(x))
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        return mu, logvar

# Decoder
class Decoder(nn.Module):
    def __init__(self, latent_dim=20, hidden_dim=400, output_dim=784):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(latent_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, z):
        h = F.relu(self.fc1(z))
        x_recon = torch.sigmoid(self.fc2(h))
        return x_recon

# VAE
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

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

    def forward(self, x):
        mu, logvar = self.encoder(x)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decoder(z)
        return x_recon, mu, logvar

# Loss Function
def loss_function(x_recon, x, mu, logvar):
    BCE = F.binary_cross_entropy(x_recon, x, reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

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

# Train
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
vae = VAE().to(device)
optimizer = torch.optim.Adam(vae.parameters(), lr=1e-3)

for epoch in range(5):
    total_loss = 0
    for x, _ in train_loader:
        x = x.view(-1, 784).to(device)
        x_recon, mu, logvar = vae(x)
        loss = loss_function(x_recon, x, mu, logvar)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {total_loss/len(train_loader.dataset):.4f}")

# Generate new images
with torch.no_grad():
```

```
    z = torch.randn(16, 20).to(device)
    samples = vae.decoder(z).cpu().view(-1, 1, 28, 28)

plt.figure(figsize=(6,6))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(samples[i].squeeze(), cmap='gray')
    plt.axis('off')
plt.show()
```

```
100%|██████████| 9.91M/9.91M [00:01<00:00, 6.09MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 160kB/s]
100%|██████████| 1.65M/1.65M [00:01<00:00, 1.52MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 12.0MB/s]
Epoch 1, Loss: 165.0683
Epoch 2, Loss: 122.3628
Epoch 3, Loss: 114.9047
Epoch 4, Loss: 111.8477
Epoch 5, Loss: 110.0657
```



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

# Encoder
class Encoder(nn.Module):
    def __init__(self, input_dim=784, hidden_dim=400, latent_dim=20):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc_mu = nn.Linear(hidden_dim, latent_dim)
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)

    def forward(self, x):
        h = F.relu(self.fc1(x))
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        return mu, logvar

# Decoder
class Decoder(nn.Module):
    def __init__(self, latent_dim=20, hidden_dim=400, output_dim=784):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(latent_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, z):
        h = F.relu(self.fc1(z))
        x_recon = torch.sigmoid(self.fc2(h))
        return x_recon

# VAE
class VAE(nn.Module):
    def __init__(self):
```

```
        super(VAE, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

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

    def forward(self, x):
        mu, logvar = self.encoder(x)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decoder(z)
        return x_recon, mu, logvar

# Loss Function
def loss_function(x_recon, x, mu, logvar):
    BCE = F.binary_cross_entropy(x_recon, x, reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

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

# Train
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
vae = VAE().to(device)
optimizer = torch.optim.Adam(vae.parameters(), lr=1e-3)

train_losses = []

for epoch in range(5):
    total_loss = 0
    for x, _ in train_loader:
        x = x.view(-1, 784).to(device)
        x_recon, mu, logvar = vae(x)
        loss = loss_function(x_recon, x, mu, logvar)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    avg_loss = total_loss / len(train_loader.dataset)
    train_losses.append(avg_loss)
    print(f"Epoch {epoch+1}, Loss: {avg_loss:.4f}")

# --- Plot training loss ---
plt.figure(figsize=(6,4))
plt.plot(train_losses, marker='o')
plt.title("VAE Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Average Loss")
plt.grid(True)
plt.show()

# --- Reconstruction visualization ---
vae.eval()
with torch.no_grad():
    x, _ = next(iter(train_loader))
    x = x.view(-1, 784).to(device)
    x_recon, _, _ = vae(x)
    x = x.view(-1, 1, 28, 28)
    x_recon = x_recon.view(-1, 1, 28, 28)

# Show first 10 original and reconstructed images
n = 10
plt.figure(figsize=(15,4))
for i in range(n):
    # Original
    plt.subplot(2, n, i + 1)
    plt.imshow(x[i].cpu().squeeze(), cmap='gray')
    plt.title("Original")
    plt.axis('off')
    # Reconstructed
    plt.subplot(2, n, i + 1 + n)
    plt.imshow(x_recon[i].cpu().squeeze(), cmap='gray')
    plt.title("Reconstructed")
    plt.axis('off')
plt.show()

# --- Generate new samples ---
```

```
with torch.no_grad():
    z = torch.randn(16, 20).to(device)
    samples = vae.decoder(z).cpu().view(-1, 1, 28, 28)

plt.figure(figsize=(6,6))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(samples[i].squeeze(), cmap='gray')
    plt.axis('off')
plt.suptitle("Generated New Digits")
plt.show()
```

```
Epoch 1, Loss: 165.2717
Epoch 2, Loss: 122.1188
Epoch 3, Loss: 115.0454
Epoch 4, Loss: 112.0027
Epoch 5, Loss: 110.1681
```