AUTOENCODER

Lab 10 : Perform Compression on MNIST DATASET using

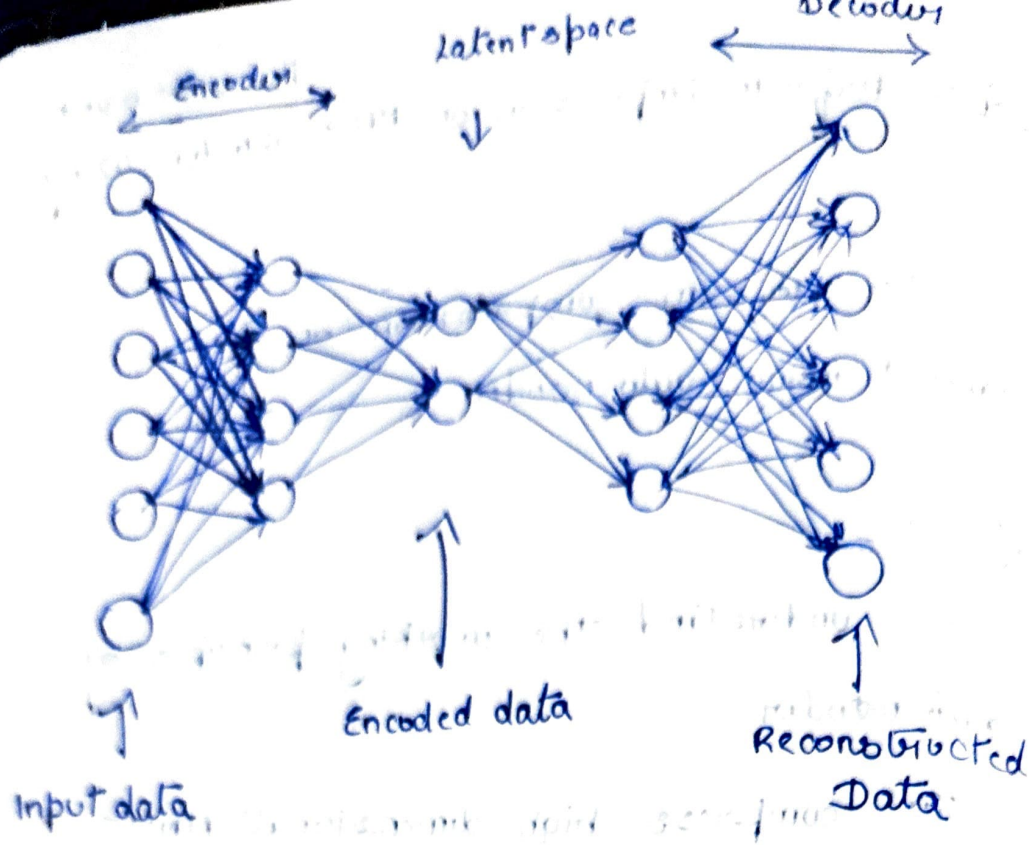
Aim : To perform compression on MNIST dataset using autoencoder

Objective

1) To understand the working principle of autoencoder.
2) To compress high dimensional MNIST images into lower dimensional latent space
3) To reconstruct the images from compressed data using decoder network.
4) To visualize and evaluate the reconstruction quality

Pseudocode

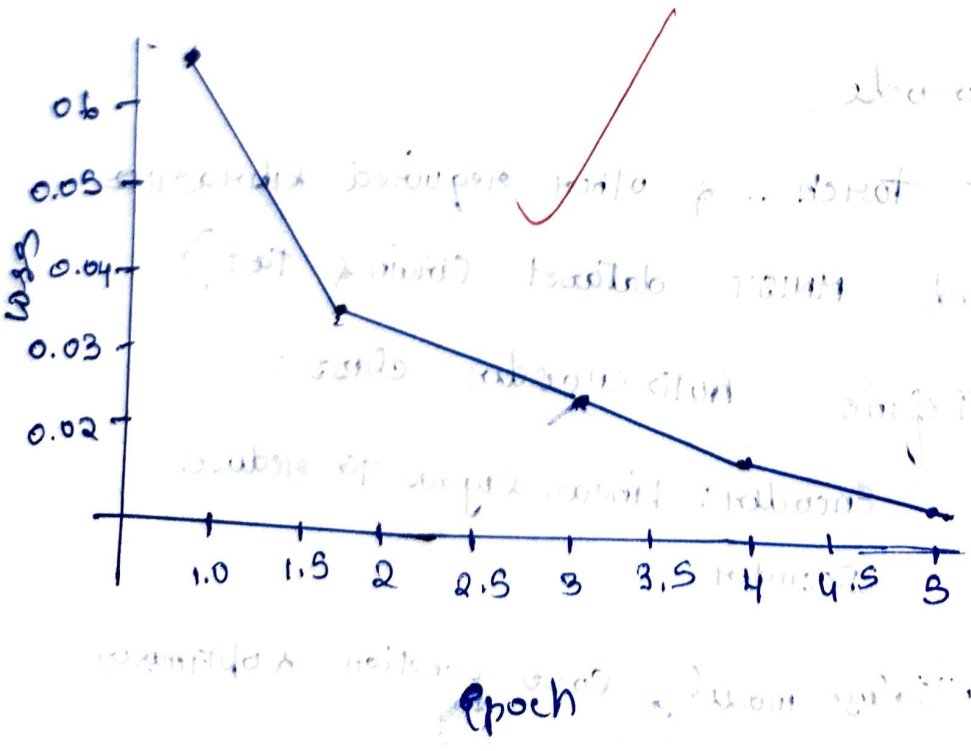
1) Import torch .. & other required libraries
2) Load MNIST dataset (train & test)
3) Define Auto encoder class :
   - Encoder : ~~linear layers to reduce~~
   - Decoder
4) Initialize model, loss function & optimizer
5) Train for 8-10 epochs.



Output

Epoch [1/5], Loss : 0.0621
Epoch [2/5], Loss : 0.0321
Epoch [3/5], Loss : 0.0249
Epoch [4/5], Loss : 0.0219
Epoch [5/5], Loss : 0.0197

Accuracy : 98.17%



For each batch :

- Forward pass through encoder & decoder
- Compute reconstruction loss
- Back propagate & update weights

6) Test :

- Pass test image through model
- Compare original vs reconstructed image

7) Visualize

Observation

1) Training Behavior :

- The autoencoder successfully learned to reconstruct digit images after several epochs.
- Training loss steadily decreased, indicating model captured key feature of digits

2) Compression effect.

- Encoder reduced 784 dimensional input into a 32 dimensional latent
- Minor loss of sharpness and fine features was observed due to dimensionality reduction

3) Visualization :

Plots showed strong similarity between original & reconstructed digit

Result.

Successfully Implemented Autoencoder

```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
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)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 38.0MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.08MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 9.76MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 8.63MB/s]
```

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

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        # Encoder: reduce 784 → 128 → 64 → 32
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32)
        )
        # Decoder: reconstruct 32 → 64 → 128 → 784
        self.decoder = nn.Sequential(
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid()
        )
```

```
        )

    def forward(self, x):
        x = x.view(-1, 28*28)        # Flatten the image
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
epochs = 5
losses = []
```

```
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, _ in train_loader:
        images = images.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, images.view(-1, 28*28))
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_loss = running_loss / len(train_loader)
    losses.append(avg_loss)
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}")
```

```
Epoch [1/5], Loss: 0.0621
Epoch [2/5], Loss: 0.0321
Epoch [3/5], Loss: 0.0249
Epoch [4/5], Loss: 0.0219
Epoch [5/5], Loss: 0.0197
```

How can I install Python libraries? | Load data from Google Drive | Show an example of training a

What can I help you build?

```
model.eval()
with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device)
```

+ Code + Text ▷ Run all

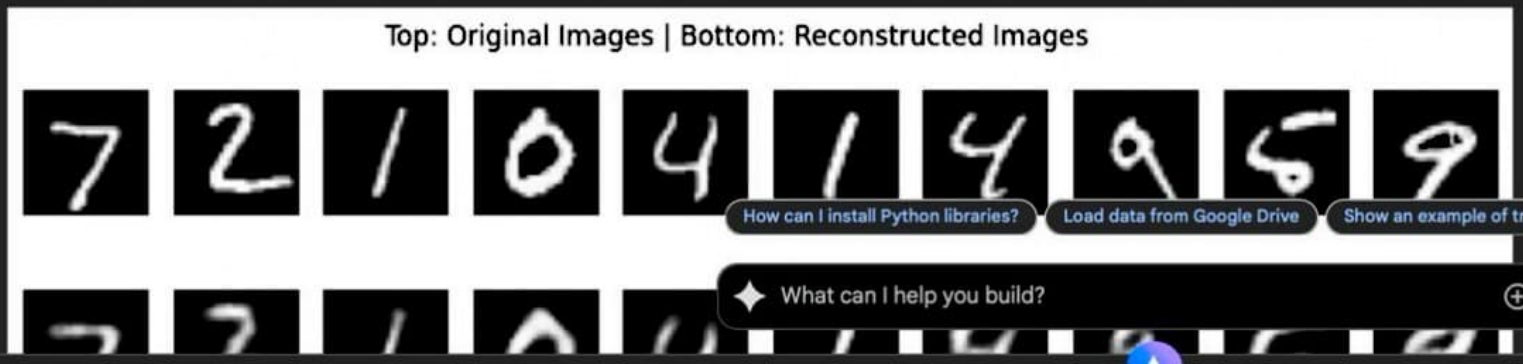
```
model.eval()
with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device)
        outputs = model(images)
        break
```

```
model.eval()
total_loss = 0
with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device)
        outputs = model(images)
        loss = criterion(outputs, images.view(-1, 28*28))
        total_loss += loss.item()

avg_mse = total_loss / len(test_loader)
accuracy = (1 - avg_mse) * 100
print(f"Approximate Reconstruction Accuracy: {accuracy:.2f}%")
```

```
Approximate Reconstruction Accuracy: 98.17%
```

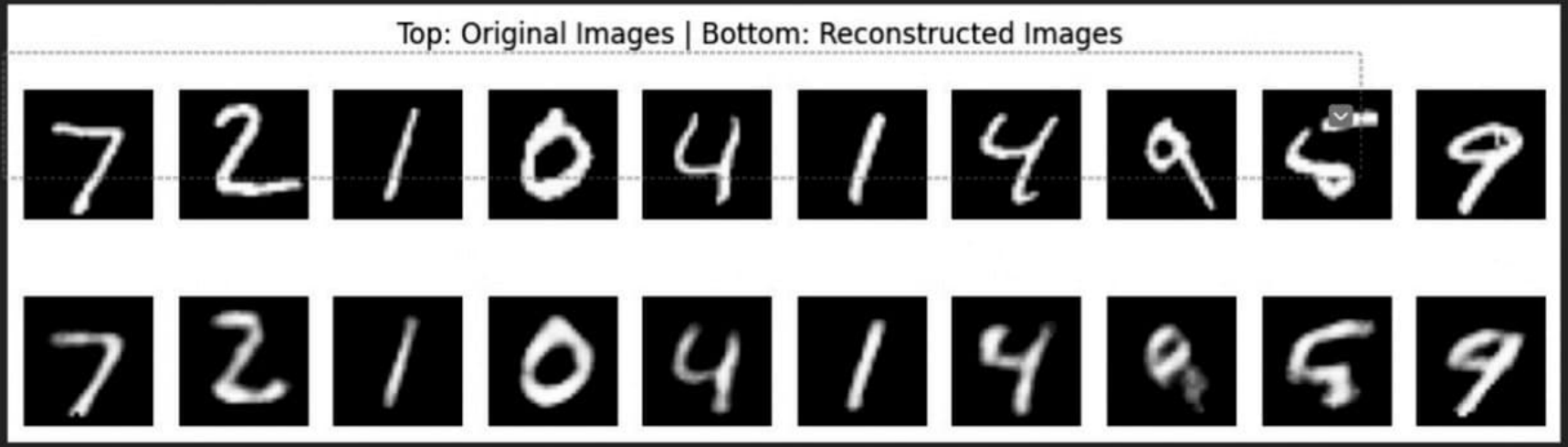
```
f, axarr = plt.subplots(2, 10, figsize=(12, 3))
for i in range(10):
    axarr[0][i].imshow(images[i].cpu().squeeze(), cmap="gray")
    axarr[0][i].axis("off")
    axarr[1][i].imshow(outputs[i].cpu().view(28, 28), cmap="gray")
    axarr[1][i].axis("off")
plt.suptitle("Top: Original Images | Bottom: Reconstructed Images")
plt.show()
```



How can I install Python libraries? Load data from Google Drive Show an example of training

What can I help you build?

```
f, axarr = plt.subplots(2, 10, figsize=(12, 3))
for i in range(10):
    axarr[0][i].imshow(images[i].cpu().squeeze(), cmap="gray")
    axarr[0][i].axis("off")
    axarr[1][i].imshow(outputs[i].cpu().view(28, 28), cmap="gray")
    axarr[1][i].axis("off")
plt.suptitle("Top: Original Images | Bottom: Reconstructed Images")
plt.show()
```




```
plt.plot(range(1, epochs + 1), losses, marker='o')
plt.title("Training Loss per Epoch (Autoencoder)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

[10]
✓ 0s

```
plt.plot(range(1, epochs + 1), losses, marker='o')
plt.title("Training Loss per Epoch (Autoencoder)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```