```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
import os
import time
import torch
import torch.nn as nn
import torchvision
from torchvision import transforms

import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
```

```
transform = transforms.ToTensor()
```

```
train_dataset = torchvision.datasets.MNIST(root = "./data" , train = True , download = True ,  transform = transform)
valid_dataset = torchvision.datasets.MNIST(root = "./data" , train = False , download = True ,  transform = transform)
```

```
    Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
    Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
    100%|██████████| 9912422/9912422 [00:00<00:00, 143944767.85it/s]
    Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

    Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
    Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
    100%|██████████| 28881/28881 [00:00<00:00, 47541481.09it/s]
    Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

    Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
    Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
    100%|██████████| 1648877/1648877 [00:00<00:00, 50785300.20it/s]Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./da


    Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
    Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
    100%|██████████| 4542/4542 [00:00<00:00, 7186167.02it/s]
    Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

```
train_dl = torch.utils.data.DataLoader(train_dataset , batch_size = 100)
```

```
class Encoder(nn.Module):
  def __init__(self , input_size = 28*28 , hidden_size1 = 500, hidden_size2 = 250 , hidden_size3 = 100, hidden_size4 = 50, z_dim
    super().__init__()
    self.fc1 = nn.Linear(input_size , hidden_size1)
    self.fc2 = nn.Linear(hidden_size1 , hidden_size2)
    self.fc3 = nn.Linear(hidden_size2 , hidden_size3)
    self.fc4 = nn.Linear(hidden_size3 , hidden_size4)
    self.fc5 = nn.Linear(hidden_size4 , z_dim)
    self.relu = nn.ReLU()
  def forward(self , x):
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    x = self.relu(self.fc3(x))
    x = self.relu(self.fc4(x))
    x = self.fc5(x)
    return x
```

```
class Decoder(nn.Module):
  def __init__(self , output_size = 28*28 , hidden_size1 = 500 , hidden_size2 = 250 , hidden_size3 = 100, hidden_size4 = 50, z_di
    super().__init__()
    self.fc1 = nn.Linear(z_dim , hidden_size4)
    self.fc2 = nn.Linear(hidden_size4 , hidden_size3)
    self.fc3 = nn.Linear(hidden_size3 , hidden_size2)
    self.fc4 = nn.Linear(hidden_size2 , hidden_size1)
    self.fc5 = nn.Linear(hidden_size1 , output_size)
    self.relu = nn.ReLU()
  def forward(self , x):
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    x = self.relu(self.fc3(x))
    x = self.relu(self.fc4(x))
    x = torch.sigmoid(self.fc5(x))
    return x
```

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

```
device(type='cuda')
```

```
enc = Encoder().to(device)
dec = Decoder().to(device)
```

```
loss_fn = nn.MSELoss()
optimizer_enc = torch.optim.Adam(enc.parameters())
optimizer_dec = torch.optim.Adam(dec.parameters())
```

```
train_loss = []
num_epochs = 400
checkpoint_path = '/content/drive/MyDrive/model/Autoencoder/checkpoint_30z_5h_200e.pth'
```

```
# Check if a checkpoint exists to resume training
if os.path.exists("/content/drive/MyDrive/model/Autoencoder/checkpoint_30z_5h_200e.pth"):
  checkpoint = torch.load("/content/drive/MyDrive/model/Autoencoder/checkpoint_30z_5h_200e.pth")
  enc.load_state_dict(checkpoint["enc_state_dict"])
  dec.load_state_dict(checkpoint["dec_state_dict"])
  optimizer_enc.load_state_dict(checkpoint["optimizer_enc_state_dict"])
  optimizer_dec.load_state_dict(checkpoint["optimizer_dec_state_dict"])
  train_loss = checkpoint["loss"]
  start_epoch = checkpoint["epoch"] + 1  # Start from the next epoch after the loaded checkpoint
  print("Resume training from epoch", start_epoch)
else:
  start_epoch = 1
```

```
Resume training from epoch 401
```

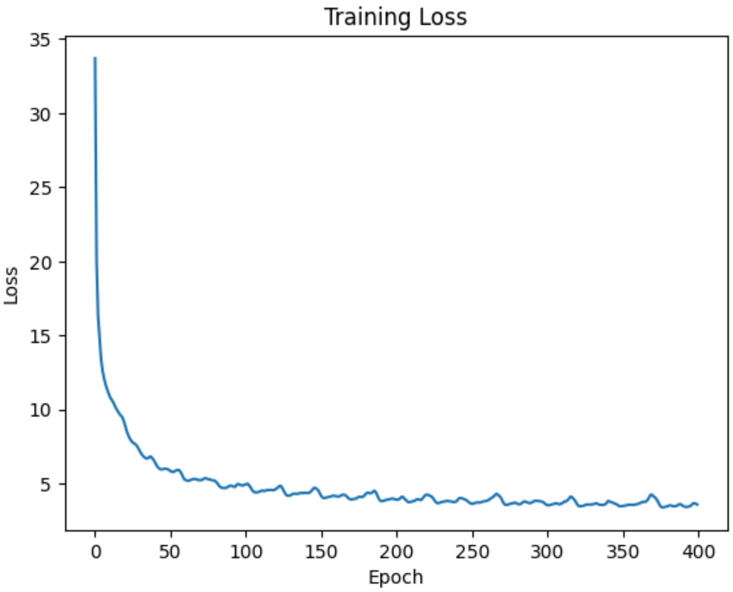
```
total_batches = len(train_dl)
for epoch in range(start_epoch,num_epochs+1):
    train_epoch_loss = 0
    start_time = time.time()
    # Create a tqdm progress bar for the epoch
    epoch_progress = tqdm(enumerate(train_dl, 1), total=total_batches, desc=f'Epoch {epoch}/{num_epochs}', leave=False)
    for step, (imgs, _) in epoch_progress:
        imgs = imgs.to(device)
        imgs = imgs.flatten(1)
        latents = enc(imgs)
        output = dec(latents)
        loss = loss_fn(output, imgs)
        train_epoch_loss += loss.item()
        optimizer_enc.zero_grad()
        optimizer_dec.zero_grad()
        loss.backward()
        optimizer_enc.step()
        optimizer_dec.step()
        # Update the progress bar description with current step and loss
        epoch_progress.set_description(f'Epoch {epoch}/{num_epochs}, Step {step}/{total_batches}, Loss: {loss.item():.4f}')
    train_loss.append(train_epoch_loss)
    # Close the tqdm progress bar for the epoch
    epoch_progress.close()

    # Print the epoch loss after each epoch
    print('\n')
    print(f'Epoch {epoch}/{num_epochs}, Loss: {train_epoch_loss:.4f}, Time taken: [{time.time() - start_time:.2f}s]')

    # Save the model checkpoint along with training-related information
    checkpoint = {
        'epoch': epoch,
        'enc_state_dict': enc.state_dict(),  # Save the encoder model's state dictionary
        'dec_state_dict':dec.state_dict(),
        'optimizer_enc_state_dict': optimizer_enc.state_dict(),  # Save the optimizer state
        'optimizer_dec_state_dict': optimizer_dec.state_dict(),
        'loss': train_loss,  # Save the loss
    }
    torch.save(checkpoint, checkpoint_path)
```

```
checkpoint = torch.load("/content/drive/MyDrive/model/Autoencoder/checkpoint_30z_5h_400e.pth")
saved_losses = checkpoint['loss']

# Plot the loss values
plt.plot(saved_losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```



```
values = None
all_labels = []

with torch.no_grad():
  for (imgs , labels) in train_dl:
    imgs = imgs.to(device)
    imgs = imgs.flatten(1)
    all_labels.extend(list(labels.numpy()))
    latents = enc(imgs)
    if values is None:
      values = latents.cpu()
    else:
      values = torch.vstack([values , latents.cpu()])
```

```
values.shape
```

```
torch.Size([60000, 30])
```

```
len(all_labels)
```

```
60000
```

```
# cmap = plt.get_cmap('viridis', 10)
# cmap
all_labels = np.array(all_labels)
values = values.numpy()
# pc = plt.scatter(values[: , 0] , values[: , 1] , c = all_labels , cmap = cmap)
# plt.colorbar(pc)
```

```
all_means = {}
for i in range(10):
  inds = np.argwhere(all_labels == i)
  print(inds.shape)
  num_latents = values[inds].squeeze()
  mean = num_latents.mean(axis = 0)
  # all_means[i] = (mean[0] , mean[1], mean[2], mean[3], mean[4])
  all_means[i] = tuple(mean[:30])
```

```
    (5923, 1)
    (6742, 1)
    (5958, 1)
    (6131, 1)
    (5842, 1)
    (5421, 1)
    (5918, 1)
    (6265, 1)
    (5851, 1)
    (5949, 1)
```

```
all_means
```

```
    {0: (-0.019664943,
      -0.04140795,
      -0.093276344,
      -0.034774188,
      0.06456965,
      0.6099748,
      4.649796,
      0.28887,
      2.0323434,
      -0.006129101,
      0.44975147,
      -0.09560164,
      0.14724708,
      0.70593846,
      1.0216453,
      1.5119395,
      -1.3156971,
      0.11654582,
      0.5168116,
      -0.5505509,
      -0.6365081,
      0.010190381,
      1.6171646,
      0.0033239129,
      -0.15580587,
      1.4233271,
      -0.023702528,
      -0.044816338,
      -0.12257131,
      0.010026846),
     1: (0.041038856,
      -0.31787196,
      -0.022012569,
      0.045954015,
      0.10092047,
      0.48969728,
      -1.9134811,
      -1.8123839,
      -1.8941734,
      -0.02852272,
      0.029468417,
      -0.21950857,
      0.07557216,
      -0.9434995,
      3.6907852,
      0.31471026,
      0.10005181,
      -0.37161803,
      0.36695603,
      -0.2910862,
      2.2215939,
      0.0038460789,
      -2.867131,
      0.038801245,
      -0.11711168,
      -1.6802964,
      1.0380716,
      0.011429321,
```

```
with torch.no_grad():
  pred = dec(torch.Tensor(all_means[5])[None , ...].to(device)).cpu()
transforms.ToPILImage()(pred.reshape(1 , 28 , 28))
```