

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
```

```
transform = transforms.ToTensor()

train_dataset = datasets.MNIST(
    root='./data',
    train=True,
    transform=transform,
    download=True
)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 19.5MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 485kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.51MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 9.68MB/s]
```

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

        # Encoder
        self.fc1 = nn.Linear(784, 400)
        self.fc_mu = nn.Linear(400, 20)
        self.fc_logvar = nn.Linear(400, 20)

        # Decoder
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 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.fc3(z))
        return torch.sigmoid(self.fc4(h))

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

def loss_function(recon_x, x, mu, logvar):
```

```

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

```

```

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

model = VAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 10

for epoch in range(epochs):
    model.train()
    train_loss = 0

    for data, _ in train_loader:
        data = data.view(-1, 784).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()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {train_loss/len(train_loader.dataset):.4f}")

```

```

Epoch [1/10], Loss: 164.5122
Epoch [2/10], Loss: 121.0364
Epoch [3/10], Loss: 114.4375
Epoch [4/10], Loss: 111.5981
Epoch [5/10], Loss: 109.8181
Epoch [6/10], Loss: 108.6625
Epoch [7/10], Loss: 107.8387
Epoch [8/10], Loss: 107.1762
Epoch [9/10], Loss: 106.6723
Epoch [10/10], Loss: 106.2307

```

```

with torch.no_grad():
    z = torch.randn(64, 20).to(device)
    samples = model.decode(z)

```

