variationalautoencoder-mnist

February 2, 2026

[1]:
```python
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# Transform: normalize images to [0,1]
transform = transforms.Compose([
    transforms.ToTensor()
])

# Load MNIST dataset (auto-downloads)
train_dataset = datasets.MNIST(
    root="./data",
    train=True,
    download=True,
    transform=transform
)

test_dataset = datasets.MNIST(
    root="./data",
    train=False,
    download=True,
    transform=transform
)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
100%|      | 9.91M/9.91M [00:00<00:00, 17.9MB/s]
100%|      | 28.9k/28.9k [00:00<00:00, 481kB/s]
100%|      | 1.65M/1.65M [00:00<00:00, 4.47MB/s]
100%|      | 4.54k/4.54k [00:00<00:00, 3.85MB/s]
```

```
Using device: cuda
```

[2]:
```
import torch.nn as nn
import torch.nn.functional as F

latent_dim = 20

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

        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU()
        )

        self.fc_mu = nn.Linear(256, latent_dim)
        self.fc_logvar = nn.Linear(256, latent_dim)

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 28*28),
            nn.Sigmoid()
        )

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

    def forward(self, x):
        x = x.view(-1, 28*28)
        h = self.encoder(x)
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        z = self.reparameterize(mu, logvar)
        out = self.decoder(z)
        return out.view(-1, 1, 28, 28), mu, logvar
```

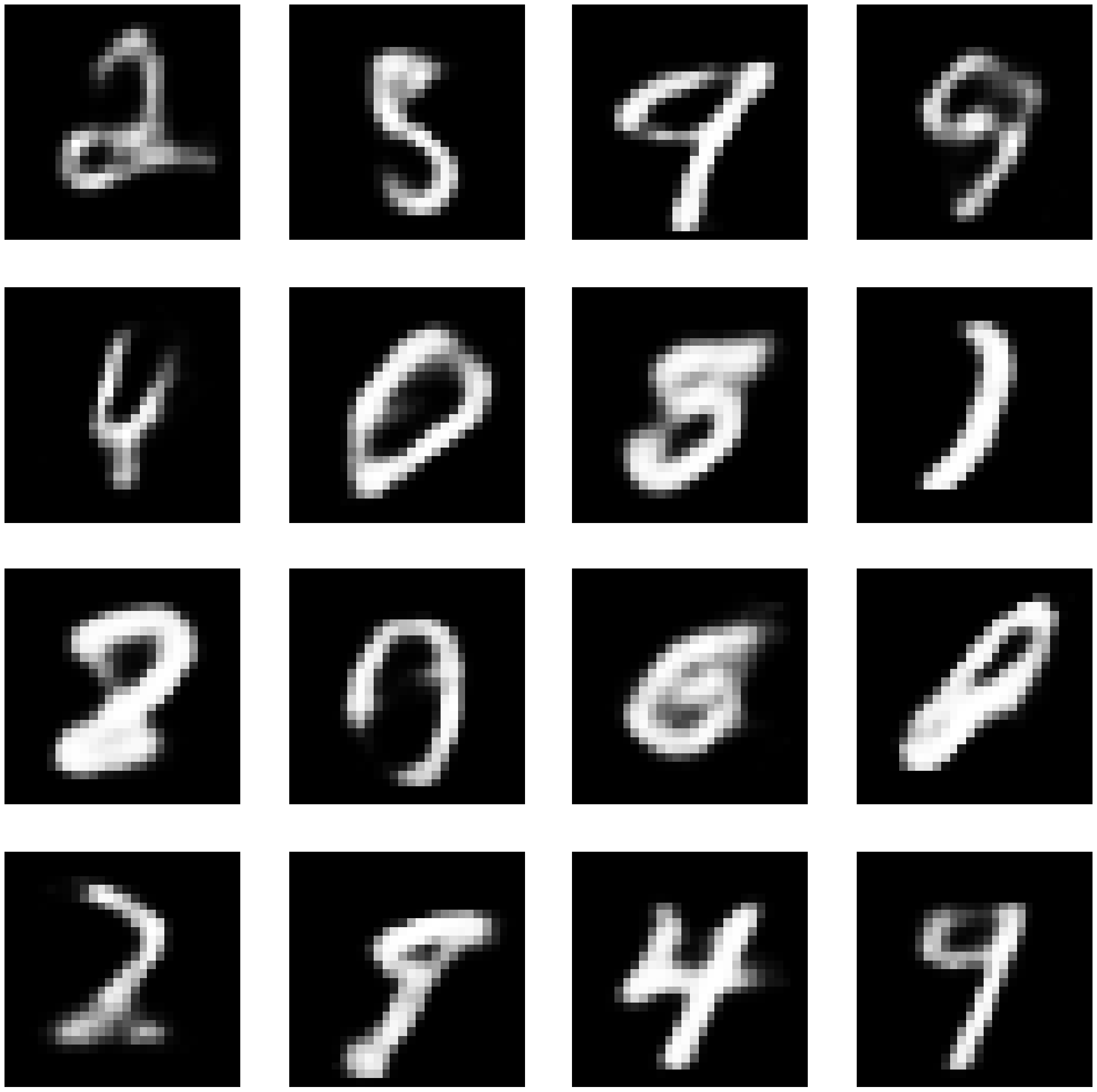
[3]:
```
def vae_loss(recon_x, x, mu, logvar):
    recon_loss = F.binary_cross_entropy(
        recon_x, x, reduction="sum"
    )
    kl_loss = -0.5 * torch.sum(
        1 + logvar - mu.pow(2) - logvar.exp()
    )
    return recon_loss + kl_loss
```

[4]:

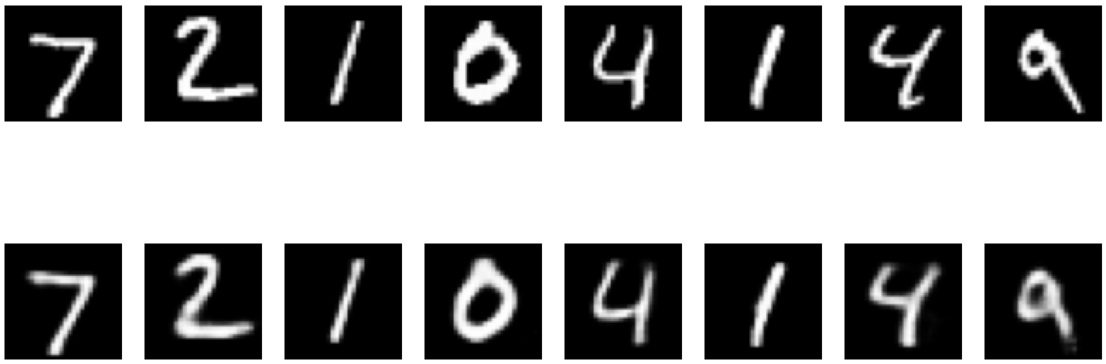
```
model = VAE().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

epochs = 50
losses = []

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

    for imgs, _ in train_loader:
        imgs = imgs.to(device)

        recon, mu, logvar = model(imgs)
        loss = vae_loss(recon, imgs, mu, logvar)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(train_dataset)
    losses.append(avg_loss)

    print(f"Epoch [{epoch+1}/{epochs}] Loss: {avg_loss:.2f}")
```

```
Epoch [1/50] Loss: 175.85
Epoch [2/50] Loss: 128.79
Epoch [3/50] Loss: 116.94
Epoch [4/50] Loss: 111.63
Epoch [5/50] Loss: 108.82
Epoch [6/50] Loss: 107.06
Epoch [7/50] Loss: 105.68
Epoch [8/50] Loss: 104.63
Epoch [9/50] Loss: 103.85
Epoch [10/50] Loss: 103.15
Epoch [11/50] Loss: 102.54
Epoch [12/50] Loss: 102.07
Epoch [13/50] Loss: 101.64
Epoch [14/50] Loss: 101.25
Epoch [15/50] Loss: 100.89
Epoch [16/50] Loss: 100.57
Epoch [17/50] Loss: 100.27
Epoch [18/50] Loss: 100.03
Epoch [19/50] Loss: 99.78
Epoch [20/50] Loss: 99.52
Epoch [21/50] Loss: 99.39
Epoch [22/50] Loss: 99.22
Epoch [23/50] Loss: 99.00
Epoch [24/50] Loss: 98.88
Epoch [25/50] Loss: 98.71
Epoch [26/50] Loss: 98.55
Epoch [27/50] Loss: 98.39
Epoch [28/50] Loss: 98.28
Epoch [29/50] Loss: 98.19
Epoch [30/50] Loss: 98.08
Epoch [31/50] Loss: 97.98
Epoch [32/50] Loss: 97.89
Epoch [33/50] Loss: 97.72
Epoch [34/50] Loss: 97.67
Epoch [35/50] Loss: 97.60
Epoch [36/50] Loss: 97.48
Epoch [37/50] Loss: 97.40
Epoch [38/50] Loss: 97.31
Epoch [39/50] Loss: 97.27
Epoch [40/50] Loss: 97.15
Epoch [41/50] Loss: 97.13
Epoch [42/50] Loss: 97.02
Epoch [43/50] Loss: 96.93
Epoch [44/50] Loss: 96.92
Epoch [45/50] Loss: 96.85
Epoch [46/50] Loss: 96.80
Epoch [47/50] Loss: 96.68
Epoch [48/50] Loss: 96.61
Epoch [49/50] Loss: 96.64
Epoch [50/50] Loss: 96.56
```

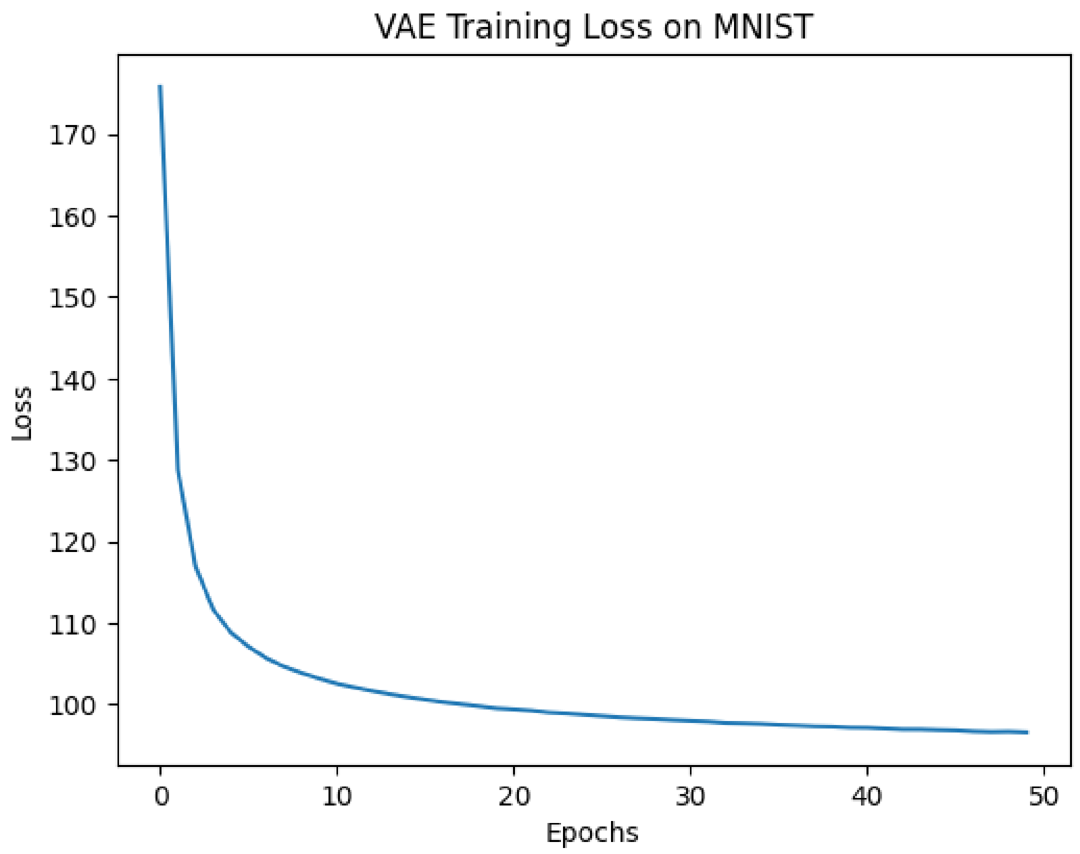
[5]:
```
import matplotlib.pyplot as plt

model.eval()
with torch.no_grad():
    z = torch.randn(16, latent_dim).to(device)
    generated = model.decoder(z).view(-1, 1, 28, 28)

plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(generated[i][0].cpu(), cmap="gray")
    plt.axis("off")
plt.show()
```



[6]:
```
imgs, _ = next(iter(test_loader))
imgs = imgs.to(device)

with torch.no_grad():
    recon, _, _ = model(imgs)

plt.figure(figsize=(10,4))
for i in range(8):
    plt.subplot(2,8,i+1)
    plt.imshow(imgs[i][0].cpu(), cmap="gray")
    plt.axis("off")

    plt.subplot(2,8,i+9)
    plt.imshow(recon[i][0].cpu(), cmap="gray")
    plt.axis("off")

plt.show()
```



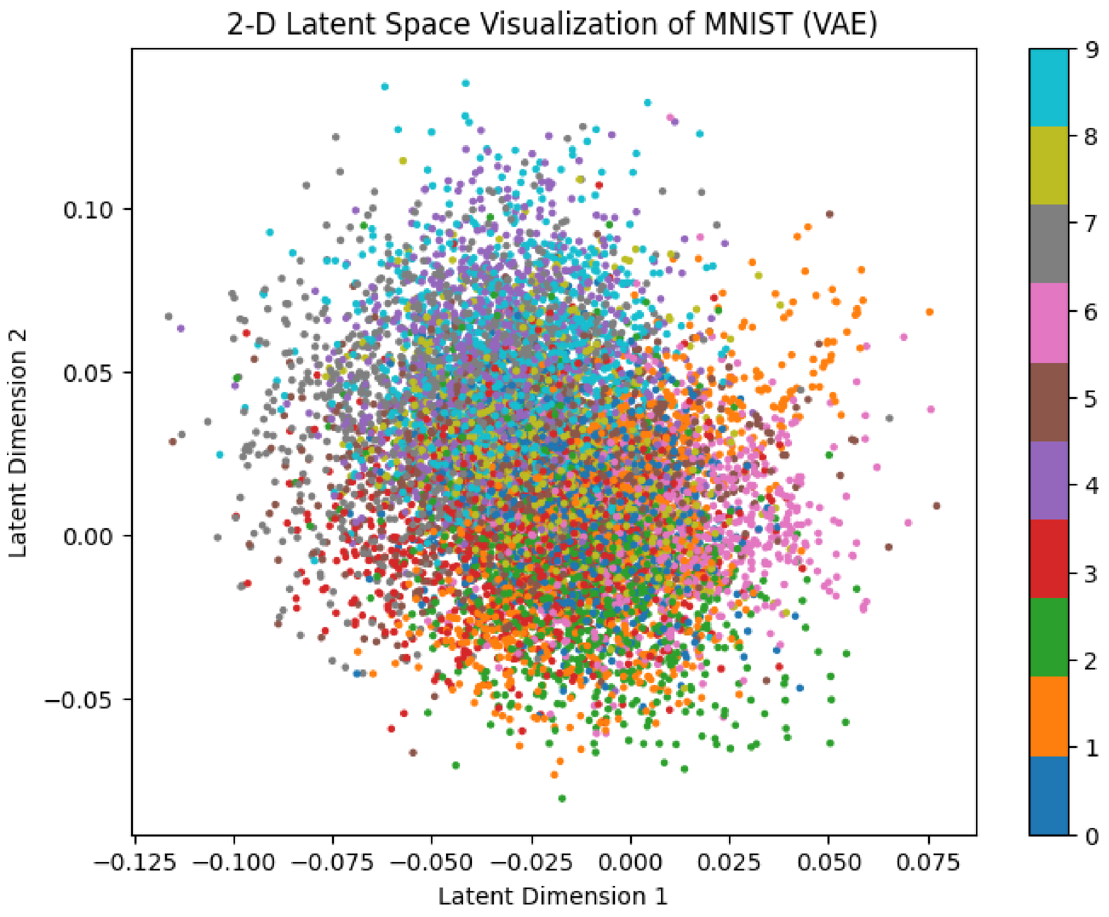
[7]:
```
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("VAE Training Loss on MNIST")
plt.show()
```



[8]:
```
latent_dim = 2    # for 2-D visualization
```

[9]:
```
import numpy as np

model.eval()

latent_vectors = []
labels = []

with torch.no_grad():
    for imgs, lbls in test_loader:
        imgs = imgs.to(device)
        _, mu, _ = model(imgs)    # use mean ( ) for visualization
        latent_vectors.append(mu.cpu())
        labels.append(lbls)

latent_vectors = torch.cat(latent_vectors).numpy()
labels = torch.cat(labels).numpy()
```

[10]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
scatter = plt.scatter(
    latent_vectors[:, 0],
    latent_vectors[:, 1],
    c=labels,
    cmap="tab10",
    s=5
)

plt.colorbar(scatter)
plt.xlabel("Latent Dimension 1")
plt.ylabel("Latent Dimension 2")
plt.title("2-D Latent Space Visualization of MNIST (VAE)")
plt.show()
```


2-D Latent Space Visualization of MNIST (VAE)
Latent Dimension 2
Latent Dimension 1