```
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
import numpy as np

# Use GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

BATCH_SIZE = 64

transform = transforms.Compose([
    transforms.ToTensor()
])

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

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

class VAE(nn.Module):
    def __init__(self, latent_dim=2):
        super(VAE, self).__init__()

        # Encoder
        self.fc1 = nn.Linear(784, 400)
        self.relu = nn.ReLU()
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)

        # Decoder
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 784)
        self.sigmoid = nn.Sigmoid()

    def encode(self, x):
        h1 = self.relu(self.fc1(x))
        return self.fc_mu(h1), self.fc_logvar(h1)

    def reparameterize(self, mu, logvar):
```

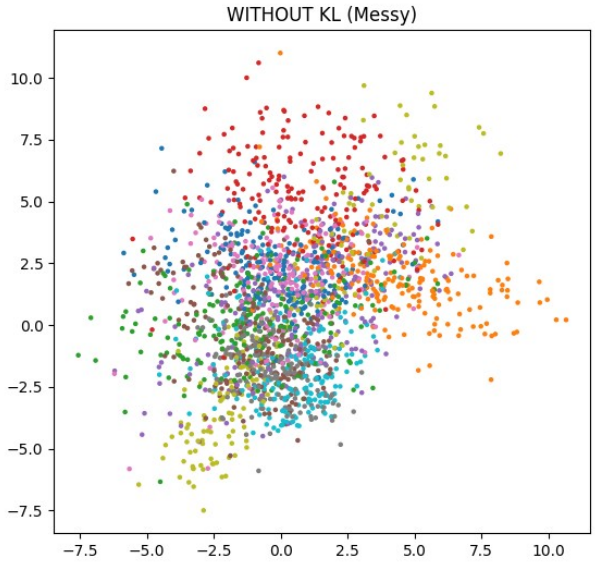
```
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h3 = self.relu(self.fc3(z))
        return self.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

def train_experiment(use_kld=True, epochs=10):
    model = VAE(latent_dim=10).to(device)
    optimizer = optim.Adam(model.parameters(), lr=1e-3)

    bce_history = []
    print(f"\nTraining started | KLD {'ON' if use_kld else 'OFF'}")

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

        for data, _ in train_loader:
            data = data.to(device)
            optimizer.zero_grad()

            recon, mu, logvar = model(data)
            BCE = nn.functional.binary_cross_entropy(
                recon, data.view(-1, 784), reduction='sum'
            )

            if use_kld:
                KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) -
logvar.exp())
            else:
                KLD = torch.tensor(0.0).to(device)

            beta = 0.1
            loss = BCE + beta * KLD

            loss.backward()
            optimizer.step()

            total_bce += BCE.item()

        avg_bce = total_bce / len(train_loader.dataset)
        bce_history.append(avg_bce)
        print(f"Epoch {epoch+1}/{epochs} | BCE: {avg_bce:.2f}")
```

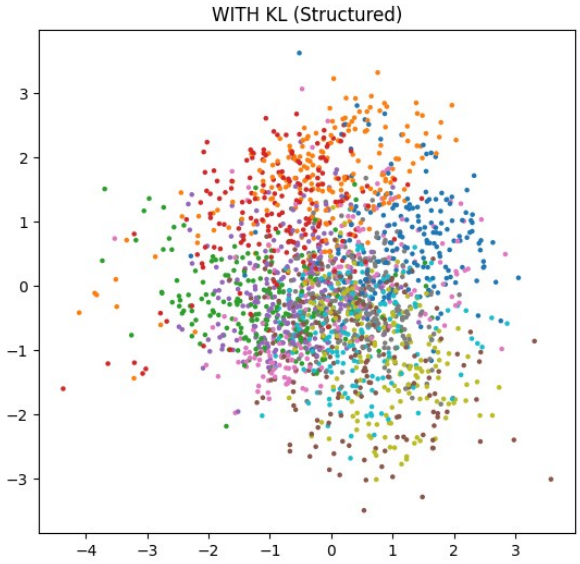
```
    return model, bce_history

model_no_kl, loss_no_kl = train_experiment(use_kld=False, epochs=20)
model_with_kl, loss_with_kl = train_experiment(use_kld=True,
epochs=20)

def compare_latent_space(model_no_kl, model_with_kl, loader):
    model_no_kl.eval()
    model_with_kl.eval()

    z_no_kl, z_with_kl, labels_list = [], [], []

    with torch.no_grad():
        for data, labels in loader:
            data = data.to(device)
            mu_no, _ = model_no_kl.encode(data.view(-1, 784))
            mu_with, _ = model_with_kl.encode(data.view(-1, 784))

            z_no_kl.append(mu_no.cpu())
            z_with_kl.append(mu_with.cpu())
            labels_list.append(labels)

            if len(z_no_kl) * BATCH_SIZE > 2000:
                break

    z_no_kl = torch.cat(z_no_kl).numpy()
    z_with_kl = torch.cat(z_with_kl).numpy()
    labels = torch.cat(labels_list).numpy()

    fig, axes = plt.subplots(1, 2, figsize=(14, 6))

    axes[0].scatter(z_no_kl[:, 0], z_no_kl[:, 1], c=labels,
cmap='tab10', s=5)
    axes[0].set_title("WITHOUT KL (Messy)")

    axes[1].scatter(z_with_kl[:, 0], z_with_kl[:, 1], c=labels,
cmap='tab10', s=5)
    axes[1].set_title("WITH KL (Structured)")

    plt.show()

def compare_generation(model_no_kl, model_with_kl, latent_dim=10):
    z = torch.randn(10, latent_dim).to(device)

    with torch.no_grad():
        gen_no_kl = model_no_kl.decode(z).cpu().view(10, 28, 28)
        gen_with_kl = model_with_kl.decode(z).cpu().view(10, 28, 28)

    fig, axes = plt.subplots(2, 10, figsize=(20, 5))
```

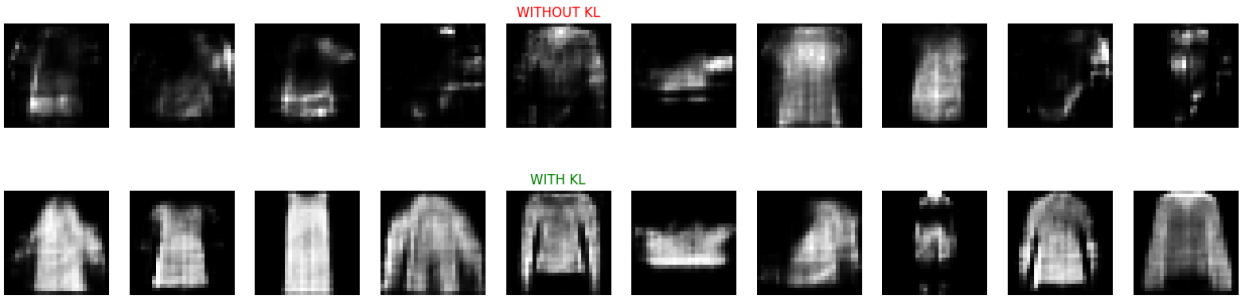
```
    for i in range(10):
        axes[0, i].imshow(gen_no_kl[i], cmap='gray')
        axes[0, i].axis('off')

        axes[1, i].imshow(gen_with_kl[i], cmap='gray')
        axes[1, i].axis('off')

    axes[0, 4].set_title("WITHOUT KL", color="red")
    axes[1, 4].set_title("WITH KL", color="green")
    plt.show()

compare_latent_space(model_no_kl, model_with_kl, test_loader)
compare_generation(model_no_kl, model_with_kl)

print(f"Final BCE (No KL):  {loss_no_kl[-1]:.2f}")
print(f"Final BCE (With KL): {loss_with_kl[-1]:.2f}")
```

```
Using device: cuda

Training started | KLD OFF
Epoch 1/20 | BCE: 247.24
Epoch 2/20 | BCE: 226.10
Epoch 3/20 | BCE: 223.25
Epoch 4/20 | BCE: 221.83
Epoch 5/20 | BCE: 220.93
Epoch 6/20 | BCE: 220.30
Epoch 7/20 | BCE: 219.79
Epoch 8/20 | BCE: 219.38
Epoch 9/20 | BCE: 219.02
Epoch 10/20 | BCE: 218.72
Epoch 11/20 | BCE: 218.48
Epoch 12/20 | BCE: 218.27
Epoch 13/20 | BCE: 218.06
Epoch 14/20 | BCE: 217.90
Epoch 15/20 | BCE: 217.74
Epoch 16/20 | BCE: 217.58
Epoch 17/20 | BCE: 217.49
Epoch 18/20 | BCE: 217.33
Epoch 19/20 | BCE: 217.23
Epoch 20/20 | BCE: 217.13

Training started | KLD ON
Epoch 1/20 | BCE: 247.77
Epoch 2/20 | BCE: 226.76
Epoch 3/20 | BCE: 224.15
Epoch 4/20 | BCE: 222.79
Epoch 5/20 | BCE: 221.91
Epoch 6/20 | BCE: 221.23
Epoch 7/20 | BCE: 220.74
Epoch 8/20 | BCE: 220.33
```

```
Epoch 9/20 | BCE: 219.95
Epoch 10/20 | BCE: 219.66
Epoch 11/20 | BCE: 219.40
Epoch 12/20 | BCE: 219.19
Epoch 13/20 | BCE: 219.02
Epoch 14/20 | BCE: 218.82
Epoch 15/20 | BCE: 218.67
Epoch 16/20 | BCE: 218.53
Epoch 17/20 | BCE: 218.40
Epoch 18/20 | BCE: 218.29
Epoch 19/20 | BCE: 218.19
Epoch 20/20 | BCE: 218.10
```








```
Final BCE (No KL):  217.13
Final BCE (With KL): 218.10
```