In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np

# 使用 transforms 來將圖像轉換為張量並將像素值歸一化到 [0, 1]：
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
)

#為了更方便地處理數據集，可以使用 DataLoader，這樣可以在訓練時分批讀取數據：
train_dataset = datasets.FashionMNIST(root='data', train = True, transfor
train_loader = DataLoader(train_dataset, batch_size = 64, shuffle = True)

test_dataset = datasets.FashionMNIST(root = 'data', train = False, transf
test_loader = DataLoader(test_dataset, batch_size = 64, shuffle = False)
```

In [ ]:
```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(784, 512)
        self.fc2_mu = nn.Linear(512, 2)
        self.fc2_logvar = nn.Linear(512, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        mu = self.fc2_mu(x)
        logvar = self.fc2_logvar(x)
        return mu, logvar
```

In [ ]:
```
class Decoder(nn.Module):

    def __init__(self):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(2, 512)
        self.fc2 = nn.Linear(512, 784)

    def forward(self, z):
        z = F.relu(self.fc1(z))
        z = torch.sigmoid(self.fc2(z))
        return z
```

In [ ]:
```
def reparameterize(mu, logvar):
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(std)
    return mu + eps * std
```

In [ ]:
```
class VAE(nn.Module):

    def __init__(self):
        super(VAE, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, x):
        mu, logvar = self.encoder(x.view(-1, 784))
        z = reparameterize(mu, logvar)
        x_reconstructed = self.decoder(z)
        return x_reconstructed, mu, logvar

    # loss = reconstruction + KL Divergence
    def lossfunction(self, recon_x, x, mu, logvar):
        # recon_x is the output of decoder, x.view is the flattened versi
        BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction
        KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
        return BCE + KLD
```

In [ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = VAE().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

In [ ]:
```
def train(epoch):
    model.train()
    train_loss = 0
    # batch_idx is the index of the current batch, data is the input data
    # but we don't need labels here
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        # celaring the cumulative batch
        optimizer.zero_grad()
        # throw data into model, model outputs three values 1.reconstruct
        recon_batch, mu, logvar = model(data)
        # calculate loss
        loss = model.lossfunction(recon_batch, data, mu, logvar)
        # do back propogation
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(f'Train Epoch: {epoch} [{batch_idx * len(data)}/{len(tr
    print(f'====> Epoch: {epoch} Average loss: {train_loss / len(train_lo
```

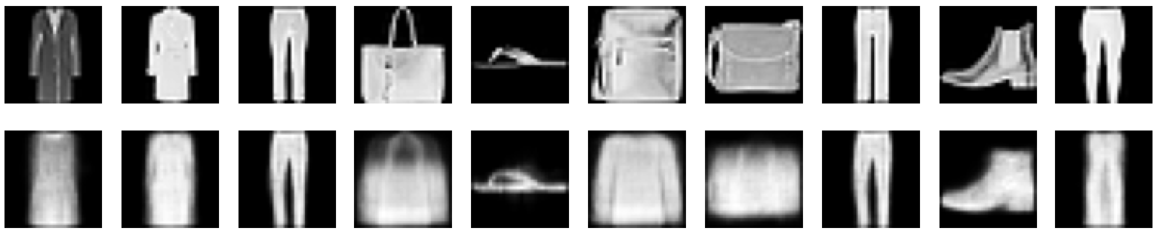
In [ ]:
```
def test(epoch):
    model.eval()
    test_loss  = 0
    with torch.no_grad():
        for data, _ in test_loader:
            data = data.to(device)
            recon_batch, mu, logvar = model(data)
            loss = model.lossfunction(recon_batch, data, mu, logvar)
            test_loss += loss.item()
    test_loss /= len(test_loader.dataset)
    print(f'====> Test set loss: {test_loss:.6f}')
```

In [ ]:
```
for epoch in range(1, 11):
    train(epoch)
    test(epoch)
```

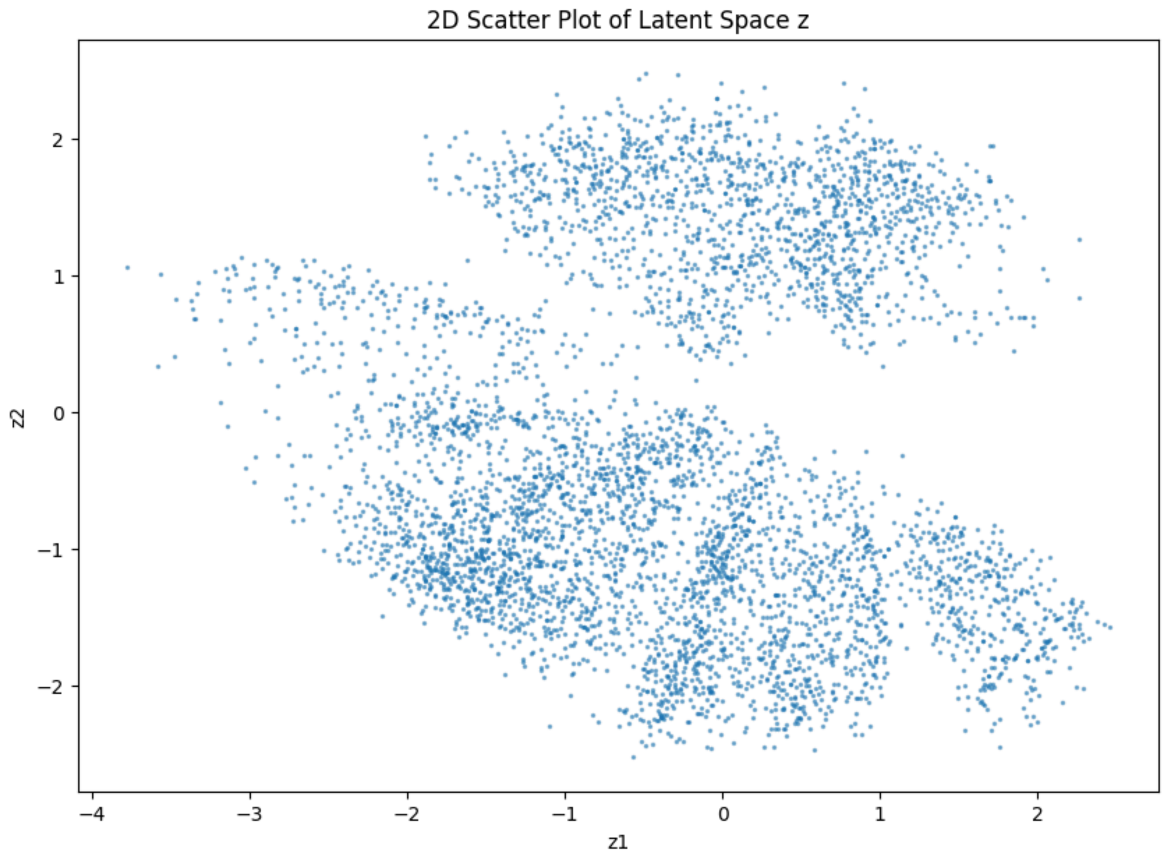
```
Train Epoch: 1 [0/60000 (0%)]   Loss: 551.485168
Train Epoch: 1 [6400/60000 (11%)]       Loss: 304.652649
Train Epoch: 1 [12800/60000 (21%)]      Loss: 276.518097
Train Epoch: 1 [19200/60000 (32%)]      Loss: 295.007690
Train Epoch: 1 [25600/60000 (43%)]      Loss: 272.580292
Train Epoch: 1 [32000/60000 (53%)]      Loss: 279.573547
Train Epoch: 1 [38400/60000 (64%)]      Loss: 282.425842
Train Epoch: 1 [44800/60000 (75%)]      Loss: 277.789398
Train Epoch: 1 [51200/60000 (85%)]      Loss: 281.809296
Train Epoch: 1 [57600/60000 (96%)]      Loss: 280.971313
====> Epoch: 1 Average loss: 282.977427
====> Test set loss: 273.302869
Train Epoch: 2 [0/60000 (0%)]   Loss: 259.749542
Train Epoch: 2 [6400/60000 (11%)]       Loss: 271.975769
Train Epoch: 2 [12800/60000 (21%)]      Loss: 279.351776
Train Epoch: 2 [19200/60000 (32%)]      Loss: 260.492371
Train Epoch: 2 [25600/60000 (43%)]      Loss: 299.672394
Train Epoch: 2 [32000/60000 (53%)]      Loss: 287.496521
Train Epoch: 2 [38400/60000 (64%)]      Loss: 281.339539
Train Epoch: 2 [44800/60000 (75%)]      Loss: 244.735031
Train Epoch: 2 [51200/60000 (85%)]      Loss: 276.669922
Train Epoch: 2 [57600/60000 (96%)]      Loss: 277.979095
====> Epoch: 2 Average loss: 269.923825
====> Test set loss: 270.597531
Train Epoch: 3 [0/60000 (0%)]   Loss: 283.002930
Train Epoch: 3 [6400/60000 (11%)]       Loss: 268.268768
Train Epoch: 3 [12800/60000 (21%)]      Loss: 281.840363
Train Epoch: 3 [19200/60000 (32%)]      Loss: 260.123138
Train Epoch: 3 [25600/60000 (43%)]      Loss: 267.386963
Train Epoch: 3 [32000/60000 (53%)]      Loss: 273.343170
Train Epoch: 3 [38400/60000 (64%)]      Loss: 269.832214
Train Epoch: 3 [44800/60000 (75%)]      Loss: 280.742096
Train Epoch: 3 [51200/60000 (85%)]      Loss: 258.704346
Train Epoch: 3 [57600/60000 (96%)]      Loss: 266.035400
====> Epoch: 3 Average loss: 267.230344
====> Test set loss: 267.861895
Train Epoch: 4 [0/60000 (0%)]   Loss: 267.802155
Train Epoch: 4 [6400/60000 (11%)]       Loss: 266.173340
Train Epoch: 4 [12800/60000 (21%)]      Loss: 255.159454
Train Epoch: 4 [19200/60000 (32%)]      Loss: 269.316772
Train Epoch: 4 [25600/60000 (43%)]      Loss: 276.357361
Train Epoch: 4 [32000/60000 (53%)]      Loss: 271.125641
Train Epoch: 4 [38400/60000 (64%)]      Loss: 236.604355
Train Epoch: 4 [44800/60000 (75%)]      Loss: 257.136292
Train Epoch: 4 [51200/60000 (85%)]      Loss: 263.118164
Train Epoch: 4 [57600/60000 (96%)]      Loss: 250.034744
====> Epoch: 4 Average loss: 265.698615
====> Test set loss: 266.728986
Train Epoch: 5 [0/60000 (0%)]   Loss: 247.459183
Train Epoch: 5 [6400/60000 (11%)]       Loss: 265.274628
Train Epoch: 5 [12800/60000 (21%)]      Loss: 278.338806
Train Epoch: 5 [19200/60000 (32%)]      Loss: 270.580719
Train Epoch: 5 [25600/60000 (43%)]      Loss: 275.645081
Train Epoch: 5 [32000/60000 (53%)]      Loss: 246.324661
Train Epoch: 5 [38400/60000 (64%)]      Loss: 256.769409
Train Epoch: 5 [44800/60000 (75%)]      Loss: 277.949097
Train Epoch: 5 [51200/60000 (85%)]      Loss: 275.103027
Train Epoch: 5 [57600/60000 (96%)]      Loss: 280.816925
====> Epoch: 5 Average loss: 264.648919
====> Test set loss: 265.914043
```

```
Train Epoch: 6 [0/60000 (0%)]   Loss: 274.601624
Train Epoch: 6 [6400/60000 (11%)]       Loss: 257.355957
Train Epoch: 6 [12800/60000 (21%)]      Loss: 263.498993
Train Epoch: 6 [19200/60000 (32%)]      Loss: 266.299347
Train Epoch: 6 [25600/60000 (43%)]      Loss: 253.853439
Train Epoch: 6 [32000/60000 (53%)]      Loss: 268.926117
Train Epoch: 6 [38400/60000 (64%)]      Loss: 268.264313
Train Epoch: 6 [44800/60000 (75%)]      Loss: 270.147614
Train Epoch: 6 [51200/60000 (85%)]      Loss: 254.557724
Train Epoch: 6 [57600/60000 (96%)]      Loss: 281.139954
====> Epoch: 6 Average loss: 263.970317
====> Test set loss: 265.252249
Train Epoch: 7 [0/60000 (0%)]   Loss: 253.533966
Train Epoch: 7 [6400/60000 (11%)]       Loss: 261.496033
Train Epoch: 7 [12800/60000 (21%)]      Loss: 272.251495
Train Epoch: 7 [19200/60000 (32%)]      Loss: 263.223846
Train Epoch: 7 [25600/60000 (43%)]      Loss: 269.971008
Train Epoch: 7 [32000/60000 (53%)]      Loss: 261.991699
Train Epoch: 7 [38400/60000 (64%)]      Loss: 270.766327
Train Epoch: 7 [44800/60000 (75%)]      Loss: 280.789368
Train Epoch: 7 [51200/60000 (85%)]      Loss: 268.426178
Train Epoch: 7 [57600/60000 (96%)]      Loss: 270.372955
====> Epoch: 7 Average loss: 263.417405
====> Test set loss: 264.927165
Train Epoch: 8 [0/60000 (0%)]   Loss: 253.651917
Train Epoch: 8 [6400/60000 (11%)]       Loss: 265.290710
Train Epoch: 8 [12800/60000 (21%)]      Loss: 277.592072
Train Epoch: 8 [19200/60000 (32%)]      Loss: 276.377319
Train Epoch: 8 [25600/60000 (43%)]      Loss: 256.200104
Train Epoch: 8 [32000/60000 (53%)]      Loss: 264.769623
Train Epoch: 8 [38400/60000 (64%)]      Loss: 267.516754
Train Epoch: 8 [44800/60000 (75%)]      Loss: 253.145065
Train Epoch: 8 [51200/60000 (85%)]      Loss: 262.675903
Train Epoch: 8 [57600/60000 (96%)]      Loss: 261.338135
====> Epoch: 8 Average loss: 262.952601
====> Test set loss: 264.130661
Train Epoch: 9 [0/60000 (0%)]   Loss: 243.290100
Train Epoch: 9 [6400/60000 (11%)]       Loss: 256.626404
Train Epoch: 9 [12800/60000 (21%)]      Loss: 247.310242
Train Epoch: 9 [19200/60000 (32%)]      Loss: 262.353394
Train Epoch: 9 [25600/60000 (43%)]      Loss: 261.102997
Train Epoch: 9 [32000/60000 (53%)]      Loss: 290.598175
Train Epoch: 9 [38400/60000 (64%)]      Loss: 272.735870
Train Epoch: 9 [44800/60000 (75%)]      Loss: 265.815765
Train Epoch: 9 [51200/60000 (85%)]      Loss: 267.550049
Train Epoch: 9 [57600/60000 (96%)]      Loss: 249.776703
====> Epoch: 9 Average loss: 262.573781
====> Test set loss: 264.467745
Train Epoch: 10 [0/60000 (0%)]  Loss: 263.751343
Train Epoch: 10 [6400/60000 (11%)]      Loss: 258.394714
Train Epoch: 10 [12800/60000 (21%)]     Loss: 245.976196
Train Epoch: 10 [19200/60000 (32%)]     Loss: 267.838196
Train Epoch: 10 [25600/60000 (43%)]     Loss: 246.154114
Train Epoch: 10 [32000/60000 (53%)]     Loss: 266.675140
Train Epoch: 10 [38400/60000 (64%)]     Loss: 257.705078
Train Epoch: 10 [44800/60000 (75%)]     Loss: 248.659302
Train Epoch: 10 [51200/60000 (85%)]     Loss: 244.768768
Train Epoch: 10 [57600/60000 (96%)]     Loss: 265.811584
====> Epoch: 10 Average loss: 262.296453
====> Test set loss: 263.557671
```

In [ ]:
```
def show_reconstructed_images(model, num_images = 10):
    model.eval()
    with torch.no_grad():
        for i, (data, _) in enumerate(train_loader):
            data = data.to(device)
            recon_batch, _, _ = model(data)
            if i == 0:
                comparison = torch.cat([data[:num_images], recon_batch.vi
                break
    fig, axes = plt.subplots(2, num_images, figsize = (10, 2))
    for k in range(num_images):
        axes[0, k].imshow(comparison[k].cpu().numpy().squeeze(), cmap='gr
        axes[1, k].imshow(comparison[num_images + k].cpu().numpy().squeez
        axes[0, k].axis('off')
        axes[1, k].axis('off')
    plt.show()

show_reconstructed_images(model)
```



In [ ]:
```
def plot_latent_space(model, num_points = 5000):
    model.eval()
    z_list = []
    with torch.no_grad():
        for i, (data, _) in enumerate(train_loader):
            data = data.to(device)
            mu, logvar = model.encoder(data.view(-1, 784))
            z = reparameterize(mu, logvar)
            z_list.append(z.cpu().numpy())
            if len(z_list) * train_loader.batch_size >= num_points:
                break

    z_array = np.concatenate(z_list)[:num_points]
    plt.figure(figsize=(10, 7))
    plt.scatter(z_array[:, 0], z_array[:, 1], s=2, alpha=0.5)
    plt.title('2D Scatter Plot of Latent Space z')
    plt.xlabel('z1')
    plt.ylabel('z2')
    plt.show()

plot_latent_space(model)
```



In [ ]:
```
def generate_grid_images(model, grid_size=15):
    model.eval()
    figure = np.zeros((28 * grid_size, 28 * grid_size))
    grid_x = np.linspace(-3, 3, grid_size)
    grid_y = np.linspace(-3, 3, grid_size)
    with torch.no_grad():
        for i, yi in enumerate(grid_x):
            for j, xi in enumerate(grid_y):
                z = torch.tensor([[xi, yi]], dtype=torch.float).to(device
                recon_x = model.decoder(z)
                digit = recon_x.view(28, 28).cpu().numpy()
                figure[i * 28: (i + 1) * 28, j * 28: (j + 1) * 28] = digi
    plt.figure(figsize=(10, 10))
    plt.imshow(figure, cmap='gray')
    plt.title('Generated Images in Latent Space Grid')
    plt.axis('off')
    plt.show()

generate_grid_images(model)
```

Generated Images in Latent Space Grid