Deep Learning Course

Assignment Four

Assignment Goals:

- Implementing Fully Connected AutoEncoders
- Implement naive generative model
- Understand VAE and GAN, then implement a classical generative model: VAE-GAN.

Please be aware that this assignment must be completed using PyTorch.

DataSet

In this Assignment, you will use the Fashion-MNIST dataset. The dataset is not given in the assignment package, please download/load by yourself. *Hint*: You can use

```
(x_train, _), (x_test, _)  = keras.datasets.fashion_mnist.load_data()
```

to load the dataset.

Requirements

1. **(20 points) Implement a Fully Connected AutoEncoder**
   - Your AutoEncoder should have a bottleneck with two neurons and use Mean Squared Error (MSE) as the objective function. Design the model structure by yourself. Notice that in an AutoEncoder, the layer with the least number of neurons is referred to as a bottleneck.
   - Train your model on Fashion-MNIST. Plot the train and test loss.
   - Randomly select 10 images from the test set, encode the selected 10 images, visualize the original images and the decoded images.

1. **(30 points) Naive generative model**

   This question is about using an AutoEncoder to generate similar but not identical Fashion-MNIST items. We use a naive approach: Try to see if a trained decoder can map randomly generated inputs (random numbers) to a recognizable Fashion-MNIST item.

   A. Start with your Fully Connected AutoEncoder from part 1. Try to generate new images by inputting some random numbers to the decoder (i.e. the bottleneck layer). Visualize 10 generated images. (10 points)

   B. Now restrict each neuron of the bottleneck layer to have a distribution with mean zeroes and variance one. Retrain the Fully Connected AutoEncoder with the normalized bottleneck. Now randomly generate inputs to the bottleneck layer that are drawn from the multi-variate standard normal distribution, and use the randomly generated inputs to generate new images. Visualize 10 generated images. (15 points)

C. Are the output images different between A) and B)? If so, why do you think this difference occurs? (5 points) Yes, they are much more clear. By constraining the bottleneck layer to follow a specific distribution and then sampling from that distribution to generate new images, we can create a controlled environment where the autoencoder can more effectively learn to generate clear and coherent images. A normalized latent space tends to be both continuous and complete, meaning that any point sampled from this space can be decoded into a coherent image, and slight movements in the latent space result in slight and predictable changes in the output. This property is beneficial for generating images that are variations of the training data but still maintain visual coherence.

1. **(50 points + 5 points BONUS(optional)) Advanced generative model**

In this part, you are asking to implement a VAE-GAN model. A VAE-GAN is a Generative Adversarial Network whose generator is an Variational Autoencoder.Here is the paper which proposed the VAE-GAN: **[PAPER]**. You may need to read this paper before implementing this model.

A. Implement a Variational Autoencoder based on your Fully Connected AutoEncoder from part 1. Use your VAE to randomly generate 10 images. Does the VAE produce a different quality of output image? (30 points) Yes, it is more clear than before.

B. Implement a VAE-GAN based on your implemented VAE. Train the VAE-GAN. (20 points)

- Then use your VAE-GAN to randomly generate 10 images from $p(z)$.
- Randomly select 10 images from the test dataset and reconstruct them using your model, then visualize the original and reconstructed images.
- **Loss-function of your VAE-GAN model:**
  - The basic assignment will choose the L_{prior} and L_{Gan} loss described in the paper (Eq. 5). So your loss function will be

    $$L = L_{prior} + L_{gan}$$

    instead of $L = L_{prior} + L^{Dis_l} + L_{gan}$ (Eq. 8).

- **BONUS(5 points)**: For those of you who want to try incorporating the L^{dis_l} term, this would be a bonus question. The basic idea here is to use a discriminator and use one of the hidden layers in the discriminator as a representation of the input. They apply this with a CNN but you can use a hidden discriminator layer with different discriminator architectures. You would be free to use your own creativity in how to select a discriminator layer.

- *Hint:* (1) For the generation and reconstruction tasks, refer to Section 4.1 in the paper.(2) the authors have posted their code in Github. You should write your own code completely from scratch, but you can look at their code as a resource for clarification.*

Submission Notes:

Please use Jupyter Notebook. The notebook should include the final code, results, and answers. You should submit your Notebook in .pdf and .ipynb format. (penalty 10 points).

Notice that your AutoEncoders should have only one bottleneck.

This assignment must be completed using PyTorch

Instructions:

The university policy on academic dishonesty and plagiarism (cheating) will be taken very seriously in this course. Everything submitted should be your writing or coding. You must not let other students copy your work. Spelling and grammar count.

Your implementation

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import random
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [2]:
```
transform = transforms.Compose([
    transforms.ToTensor()
])

# Download and load the training data
trainset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True, train=True,


# Download and load the test data
testset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True, train=False,
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3
-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3
-ubyte.gz to /root/.pytorch/F_MNIST_data/FashionMNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 26421880/26421880 [00:00<00:00, 106695996.28it/s]
Extracting /root/.pytorch/F_MNIST_data/FashionMNIST/raw/train-images-idx3-ubyte.gz to /r
oot/.pytorch/F_MNIST_data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1
-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1
-ubyte.gz to /root/.pytorch/F_MNIST_data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 29515/29515 [00:00<00:00, 28769435.87it/s]
Extracting /root/.pytorch/F_MNIST_data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to /r
oot/.pytorch/F_MNIST_data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-
ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-
ubyte.gz to /root/.pytorch/F_MNIST_data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 4422102/4422102 [00:00<00:00, 59358336.69it/s]
Extracting /root/.pytorch/F_MNIST_data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to /ro
ot/.pytorch/F_MNIST_data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-
ubyte.gz
```

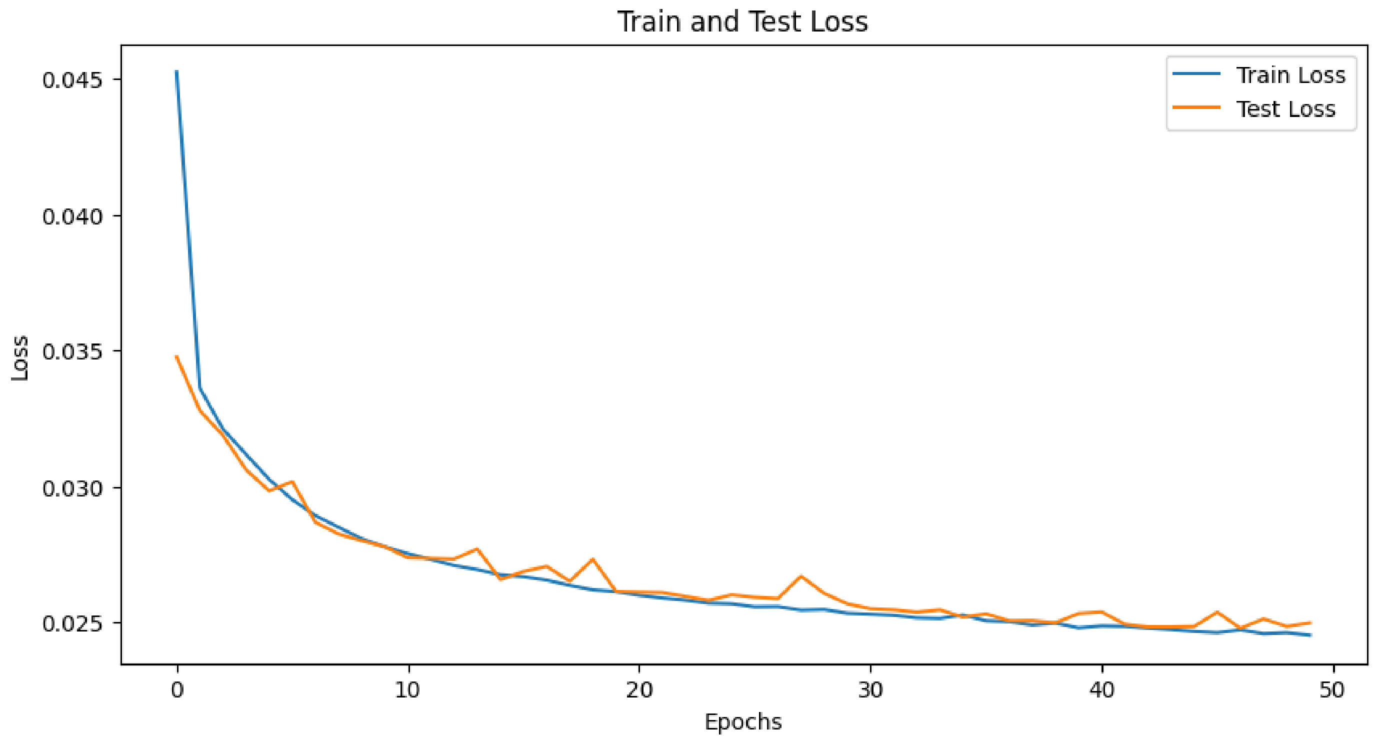
```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-
ubyte.gz to /root/.pytorch/F_MNIST_data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 5148/5148 [00:00<00:00, 18759580.36it/s]
Extracting /root/.pytorch/F_MNIST_data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to /ro
ot/.pytorch/F_MNIST_data/FashionMNIST/raw
```

In [3]:
```python
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64, shuffle=False)
```

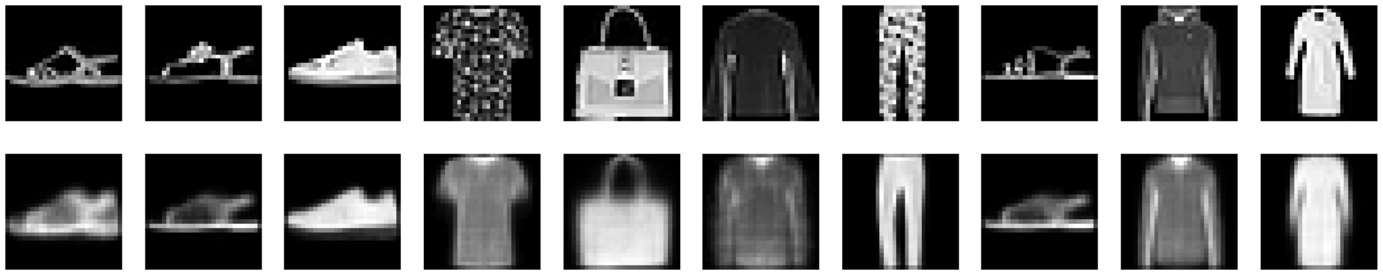
In [4]:
```python
class AutoEncoder(nn.Module):
    def __init__(self):
        super(AutoEncoder, self).__init__()
        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            # nn.Linear(64, 32),
            # nn.ReLU(),
            nn.Linear(64, 12),
            nn.ReLU(),
            nn.Linear(12, 2)
        )

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(2, 12),
            nn.ReLU(),
            nn.Linear(12, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid()
        )

    def forward(self, x):
        x_shape=x.shape
        x=torch.flatten(x, start_dim=1)
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        decoded = decoded.reshape(x_shape)
        return decoded
```

In [5]:
```python
# Initialize the model, loss function, and optimizer
model = AutoEncoder().to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# Training function
def train_model(model, criterion, optimizer, num_epochs=5):
    train_loss, test_loss = [], []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for data in trainloader:
            inputs, _ = data
            inputs = inputs.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, inputs)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
        epoch_loss = running_loss / len(trainloader)
        train_loss.append(epoch_loss)

        # Validation loss
        model.eval()
        running_loss = 0.0
        with torch.no_grad():
            for data in testloader:
                inputs, _ = data
                inputs = inputs.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, inputs)
                running_loss += loss.item()
        epoch_loss = running_loss / len(testloader)
        test_loss.append(epoch_loss)

        print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss[-1]:.4f}, Test Los

    return train_loss, test_loss

# Train the model
train_loss, test_loss = train_model(model, criterion, optimizer, num_epochs=50)
plt.figure(figsize=(10, 5))
plt.plot(train_loss, label='Train Loss')
plt.plot(test_loss, label='Test Loss')
plt.title('Train and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/50, Train Loss: 0.0452, Test Loss: 0.0348
Epoch 2/50, Train Loss: 0.0336, Test Loss: 0.0328
Epoch 3/50, Train Loss: 0.0321, Test Loss: 0.0319
Epoch 4/50, Train Loss: 0.0312, Test Loss: 0.0306
Epoch 5/50, Train Loss: 0.0303, Test Loss: 0.0298
Epoch 6/50, Train Loss: 0.0295, Test Loss: 0.0302
Epoch 7/50, Train Loss: 0.0289, Test Loss: 0.0287
Epoch 8/50, Train Loss: 0.0285, Test Loss: 0.0283
Epoch 9/50, Train Loss: 0.0281, Test Loss: 0.0280
Epoch 10/50, Train Loss: 0.0278, Test Loss: 0.0278
Epoch 11/50, Train Loss: 0.0275, Test Loss: 0.0274
Epoch 12/50, Train Loss: 0.0273, Test Loss: 0.0274
Epoch 13/50, Train Loss: 0.0271, Test Loss: 0.0273
Epoch 14/50, Train Loss: 0.0269, Test Loss: 0.0277
Epoch 15/50, Train Loss: 0.0268, Test Loss: 0.0266
Epoch 16/50, Train Loss: 0.0267, Test Loss: 0.0269
Epoch 17/50, Train Loss: 0.0266, Test Loss: 0.0271
Epoch 18/50, Train Loss: 0.0264, Test Loss: 0.0265
Epoch 19/50, Train Loss: 0.0262, Test Loss: 0.0273
Epoch 20/50, Train Loss: 0.0261, Test Loss: 0.0261
Epoch 21/50, Train Loss: 0.0260, Test Loss: 0.0261
Epoch 22/50, Train Loss: 0.0259, Test Loss: 0.0261
Epoch 23/50, Train Loss: 0.0258, Test Loss: 0.0260
Epoch 24/50, Train Loss: 0.0257, Test Loss: 0.0258
Epoch 25/50, Train Loss: 0.0257, Test Loss: 0.0260
Epoch 26/50, Train Loss: 0.0256, Test Loss: 0.0259
Epoch 27/50, Train Loss: 0.0256, Test Loss: 0.0259
Epoch 28/50, Train Loss: 0.0255, Test Loss: 0.0267
Epoch 29/50, Train Loss: 0.0255, Test Loss: 0.0261
Epoch 30/50, Train Loss: 0.0253, Test Loss: 0.0257
Epoch 31/50, Train Loss: 0.0253, Test Loss: 0.0255
Epoch 32/50, Train Loss: 0.0253, Test Loss: 0.0255
Epoch 33/50, Train Loss: 0.0252, Test Loss: 0.0254
Epoch 34/50, Train Loss: 0.0252, Test Loss: 0.0255
Epoch 35/50, Train Loss: 0.0253, Test Loss: 0.0252
Epoch 36/50, Train Loss: 0.0251, Test Loss: 0.0253
Epoch 37/50, Train Loss: 0.0250, Test Loss: 0.0251
Epoch 38/50, Train Loss: 0.0249, Test Loss: 0.0251
Epoch 39/50, Train Loss: 0.0250, Test Loss: 0.0250
Epoch 40/50, Train Loss: 0.0248, Test Loss: 0.0253
Epoch 41/50, Train Loss: 0.0249, Test Loss: 0.0254
Epoch 42/50, Train Loss: 0.0249, Test Loss: 0.0249
Epoch 43/50, Train Loss: 0.0248, Test Loss: 0.0248
Epoch 44/50, Train Loss: 0.0247, Test Loss: 0.0248
Epoch 45/50, Train Loss: 0.0247, Test Loss: 0.0249
Epoch 46/50, Train Loss: 0.0246, Test Loss: 0.0254
Epoch 47/50, Train Loss: 0.0247, Test Loss: 0.0248
Epoch 48/50, Train Loss: 0.0246, Test Loss: 0.0251
Epoch 49/50, Train Loss: 0.0246, Test Loss: 0.0249
Epoch 50/50, Train Loss: 0.0245, Test Loss: 0.0250
```

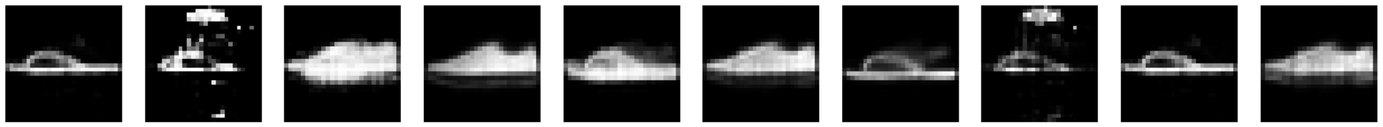


In [6]:
```python
def visualize_reconstructions(model, dataloader, num_images=10):
    device = next(model.parameters()).device
    model.eval()
    indices = random.sample(range(len(dataloader.dataset)), num_images)
    images = torch.stack([dataloader.dataset[i][0] for i in indices]).to(device)

    with torch.no_grad():
        reconstructions= model(images)
    images = images.cpu().view(-1, 28, 28).numpy()
    reconstructions = reconstructions.cpu().view(-1, 28, 28).numpy()

    fig, axs = plt.subplots(2, num_images, figsize=(20, 4))
    for i in range(num_images):
        axs[0, i].imshow(images[i], cmap='gray')
        axs[0, i].axis('off')
        axs[1, i].imshow(reconstructions[i], cmap='gray')
        axs[1, i].axis('off')
    plt.show()
visualize_reconstructions(model, testloader, num_images=10)
```



Part 2 Naive generative model

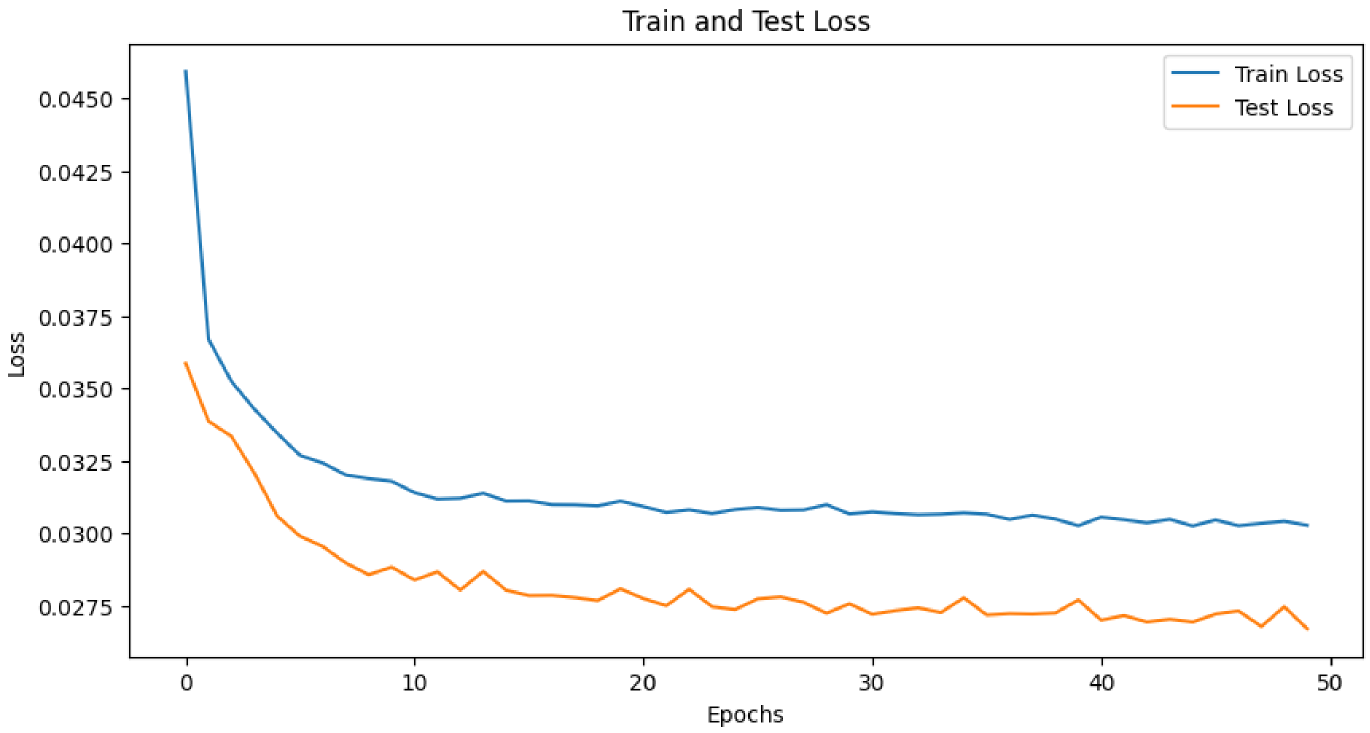
In [80]:
```
model.eval()
normalized_random_inputs = ((torch.randn(10, 2))).to(device)
with torch.no_grad():
    normalized_generated_images = model.decoder((normalized_random_inputs)).cpu().view(-
    
plt.figure(figsize=(20, 4))
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(normalized_generated_images[i], cmap='gray')
    plt.axis('off')
plt.show()
```



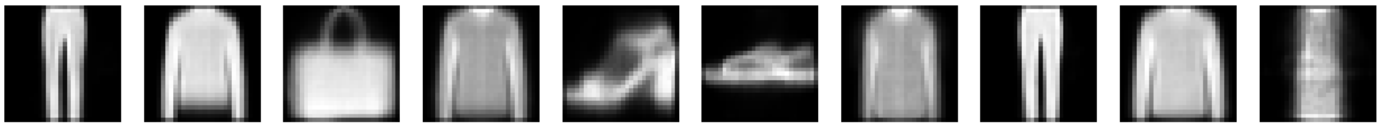
In [29]:
```
class AutoEncoder2(nn.Module):
    def __init__(self):
        super(AutoEncoder2, self).__init__()
        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            # nn.Linear(64, 32),
            # nn.ReLU(),
            nn.Linear(64, 12),
            nn.ReLU(),
            nn.Linear(12, 2)
        )

        self.bottle_neck = nn.Sequential(

            nn.BatchNorm1d(2),
            # nn.ReLU()
        )
        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(2, 12),
            nn.ReLU(),
            nn.Linear(12, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid()
        )

    def forward(self, x):
        x_shape=x.shape
        x=torch.flatten(x, start_dim=1)
        encoded = self.encoder(x)
        bot    = self.bottle_neck(encoded)
        decoded = self.decoder(bot)
        decoded = decoded.reshape(x_shape)
        return decoded
```

In [30]:
```
model1 = AutoEncoder2().to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model1.parameters(), lr=1e-3)

# Training function
def train_model1(model1, criterion, optimizer, num_epochs):
    train_loss, test_loss = [], []

    for epoch in range(num_epochs):
        model1.train()
        running_loss = 0.0
        for data in trainloader:
            inputs, _ = data
            inputs = inputs.to(device)
            outputs = model1(inputs)
            loss = criterion(outputs, inputs)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
        epoch_loss = running_loss / len(trainloader)
        train_loss.append(epoch_loss)

        # Validation loss
        model1.eval()
        running_loss = 0.0
        with torch.no_grad():
            for data in testloader:
                inputs, _ = data
                inputs = inputs.to(device)
                outputs = model1(inputs)
                loss = criterion(outputs, inputs)
                running_loss += loss.item()
        epoch_loss = running_loss / len(testloader)
        test_loss.append(epoch_loss)

        print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss[-1]:.4f}, Test Los

    return train_loss, test_loss
train_loss, test_loss = train_model1(model1, criterion, optimizer, num_epochs=50)
plt.figure(figsize=(10, 5))
plt.plot(train_loss, label='Train Loss')
plt.plot(test_loss, label='Test Loss')
plt.title('Train and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/50, Train Loss: 0.0459, Test Loss: 0.0359
Epoch 2/50, Train Loss: 0.0367, Test Loss: 0.0339
Epoch 3/50, Train Loss: 0.0352, Test Loss: 0.0333
Epoch 4/50, Train Loss: 0.0343, Test Loss: 0.0321
Epoch 5/50, Train Loss: 0.0335, Test Loss: 0.0306
Epoch 6/50, Train Loss: 0.0327, Test Loss: 0.0299
Epoch 7/50, Train Loss: 0.0324, Test Loss: 0.0295
Epoch 8/50, Train Loss: 0.0320, Test Loss: 0.0290
Epoch 9/50, Train Loss: 0.0319, Test Loss: 0.0286
Epoch 10/50, Train Loss: 0.0318, Test Loss: 0.0288
Epoch 11/50, Train Loss: 0.0314, Test Loss: 0.0284
Epoch 12/50, Train Loss: 0.0312, Test Loss: 0.0287
Epoch 13/50, Train Loss: 0.0312, Test Loss: 0.0280
Epoch 14/50, Train Loss: 0.0314, Test Loss: 0.0287
Epoch 15/50, Train Loss: 0.0311, Test Loss: 0.0280
Epoch 16/50, Train Loss: 0.0311, Test Loss: 0.0279
Epoch 17/50, Train Loss: 0.0310, Test Loss: 0.0279
Epoch 18/50, Train Loss: 0.0310, Test Loss: 0.0278
Epoch 19/50, Train Loss: 0.0309, Test Loss: 0.0277
Epoch 20/50, Train Loss: 0.0311, Test Loss: 0.0281
Epoch 21/50, Train Loss: 0.0309, Test Loss: 0.0278
Epoch 22/50, Train Loss: 0.0307, Test Loss: 0.0275
Epoch 23/50, Train Loss: 0.0308, Test Loss: 0.0281
Epoch 24/50, Train Loss: 0.0307, Test Loss: 0.0275
Epoch 25/50, Train Loss: 0.0308, Test Loss: 0.0274
Epoch 26/50, Train Loss: 0.0309, Test Loss: 0.0277
Epoch 27/50, Train Loss: 0.0308, Test Loss: 0.0278
Epoch 28/50, Train Loss: 0.0308, Test Loss: 0.0276
Epoch 29/50, Train Loss: 0.0310, Test Loss: 0.0272
Epoch 30/50, Train Loss: 0.0307, Test Loss: 0.0276
Epoch 31/50, Train Loss: 0.0307, Test Loss: 0.0272
Epoch 32/50, Train Loss: 0.0307, Test Loss: 0.0273
Epoch 33/50, Train Loss: 0.0306, Test Loss: 0.0274
Epoch 34/50, Train Loss: 0.0307, Test Loss: 0.0273
Epoch 35/50, Train Loss: 0.0307, Test Loss: 0.0278
Epoch 36/50, Train Loss: 0.0307, Test Loss: 0.0272
Epoch 37/50, Train Loss: 0.0305, Test Loss: 0.0272
Epoch 38/50, Train Loss: 0.0306, Test Loss: 0.0272
Epoch 39/50, Train Loss: 0.0305, Test Loss: 0.0273
Epoch 40/50, Train Loss: 0.0303, Test Loss: 0.0277
Epoch 41/50, Train Loss: 0.0306, Test Loss: 0.0270
Epoch 42/50, Train Loss: 0.0305, Test Loss: 0.0272
Epoch 43/50, Train Loss: 0.0304, Test Loss: 0.0269
Epoch 44/50, Train Loss: 0.0305, Test Loss: 0.0270
Epoch 45/50, Train Loss: 0.0303, Test Loss: 0.0269
Epoch 46/50, Train Loss: 0.0305, Test Loss: 0.0272
Epoch 47/50, Train Loss: 0.0303, Test Loss: 0.0273
Epoch 48/50, Train Loss: 0.0303, Test Loss: 0.0268
Epoch 49/50, Train Loss: 0.0304, Test Loss: 0.0275
Epoch 50/50, Train Loss: 0.0303, Test Loss: 0.0267
```



```
In [48]: mean = np.array([0, 0])
         covariance_matrix = np.eye(2)
         num_samples = 10
         normalized_random_inputs = torch.Tensor(np.random.multivariate_normal(mean, covariance_m
         normalized_random_inputs = normalized_random_inputs.to(device)

         model.eval()
         with torch.no_grad():
             normalized_generated_images = model1.decoder(normalized_random_inputs).cpu().view(-1

         plt.figure(figsize=(20, 4))
         for i in range(10):
             plt.subplot(1, 10, i+1)
             plt.imshow(normalized_generated_images[i], cmap='gray')
             plt.axis('off')
         plt.show()
```



Part 3

```
In [11]: 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(),
                     nn.Linear(256, 128),
                     nn.ReLU(),
                     nn.Linear(128, 64),
                     nn.ReLU(),
                     nn.Linear(64, 32),
                     nn.ReLU()

                 )

                 self.mu = nn.Linear(32, 2)
                 self.logvar = nn.Linear(32, 2)
                 # Decoder
                 self.decoder = nn.Sequential(
                     nn.Linear(2, 32),
                     nn.ReLU(),
                     nn.Linear(32, 64),
                     nn.ReLU(),
                     nn.Linear(64, 128),
                     nn.ReLU(),
                     nn.Linear(128, 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):
```

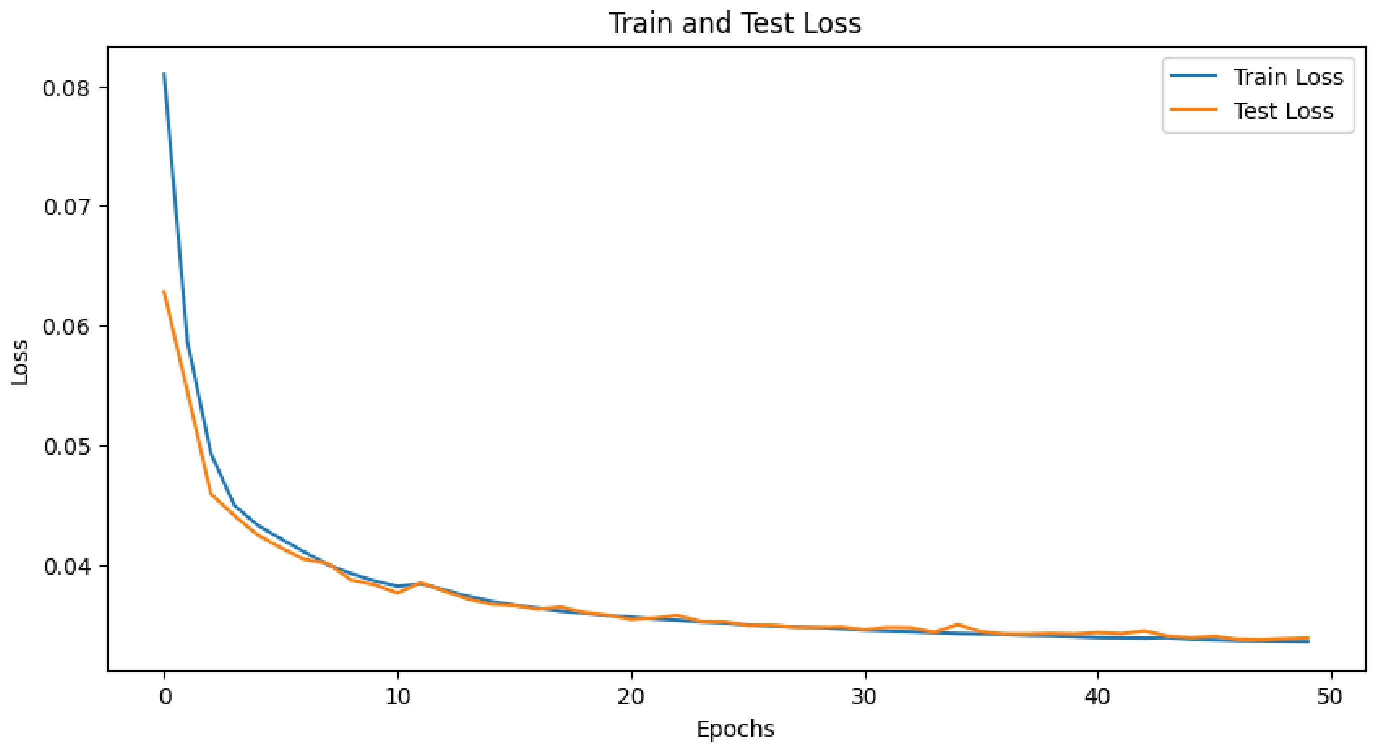
```
        x_shape=x.shape
        x=torch.flatten(x, start_dim=1)
        encoded = self.encoder(x)
        mu = self.mu(encoded)
        logvar = self.logvar(encoded)
        z = self.reparameterize(mu, logvar)
        decoded = self.decoder(z)
        decoded = torch.reshape(decoded, x_shape)
        return decoded, mu, logvar
```

In [12]:
```
VAEmodel = VAE().to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(VAEmodel.parameters(), lr=1e-4)
batch_size=64
```

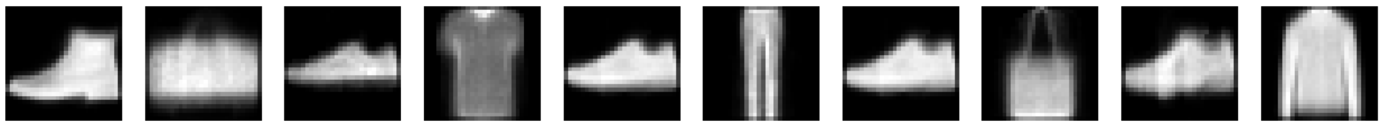
In [13]:
```
def loss_function(recon_x, x, mu, logvar):
    MSE = criterion(recon_x, x)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    # print (MSE, KLD)
    KLD/=batch_size*784
    result= MSE + KLD

    return result
```

In [14]:
```
def train_VAE(model,  optimizer, num_epochs):
    train_loss, test_loss = [], []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for data in trainloader:
            inputs, _ = data
            inputs = inputs.to(device)
            outputs,mu,logvar = model(inputs)
            loss = loss_function(outputs, inputs,mu,logvar)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
        epoch_loss = running_loss / len(trainloader)
        train_loss.append(epoch_loss)

        # Validation loss
        model.eval()
        running_loss = 0.0
        with torch.no_grad():
            for data in testloader:
                inputs, _ = data
                inputs = inputs.to(device)
                outputs,mu,logvar = model(inputs)
                loss = loss_function(outputs, inputs,mu,logvar)
                running_loss += loss.item()
        epoch_loss = running_loss / len(testloader)
        test_loss.append(epoch_loss)

        print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss[-1]:.4f}, Test Los

    return train_loss, test_loss
train_loss, test_loss = train_VAE(VAEmodel, optimizer, num_epochs=50)
plt.figure(figsize=(10, 5))
plt.plot(train_loss, label='Train Loss')
plt.plot(test_loss, label='Test Loss')
plt.title('Train and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/50, Train Loss: 0.0810, Test Loss: 0.0628
Epoch 2/50, Train Loss: 0.0586, Test Loss: 0.0545
Epoch 3/50, Train Loss: 0.0493, Test Loss: 0.0459
Epoch 4/50, Train Loss: 0.0450, Test Loss: 0.0441
Epoch 5/50, Train Loss: 0.0433, Test Loss: 0.0425
Epoch 6/50, Train Loss: 0.0422, Test Loss: 0.0414
Epoch 7/50, Train Loss: 0.0411, Test Loss: 0.0405
Epoch 8/50, Train Loss: 0.0400, Test Loss: 0.0401
Epoch 9/50, Train Loss: 0.0393, Test Loss: 0.0388
Epoch 10/50, Train Loss: 0.0387, Test Loss: 0.0384
Epoch 11/50, Train Loss: 0.0382, Test Loss: 0.0377
Epoch 12/50, Train Loss: 0.0384, Test Loss: 0.0385
Epoch 13/50, Train Loss: 0.0379, Test Loss: 0.0378
Epoch 14/50, Train Loss: 0.0374, Test Loss: 0.0372
Epoch 15/50, Train Loss: 0.0370, Test Loss: 0.0367
Epoch 16/50, Train Loss: 0.0367, Test Loss: 0.0366
Epoch 17/50, Train Loss: 0.0364, Test Loss: 0.0363
Epoch 18/50, Train Loss: 0.0361, Test Loss: 0.0365
Epoch 19/50, Train Loss: 0.0360, Test Loss: 0.0360
Epoch 20/50, Train Loss: 0.0358, Test Loss: 0.0358
Epoch 21/50, Train Loss: 0.0357, Test Loss: 0.0354
Epoch 22/50, Train Loss: 0.0355, Test Loss: 0.0356
Epoch 23/50, Train Loss: 0.0354, Test Loss: 0.0358
Epoch 24/50, Train Loss: 0.0352, Test Loss: 0.0353
Epoch 25/50, Train Loss: 0.0351, Test Loss: 0.0353
Epoch 26/50, Train Loss: 0.0350, Test Loss: 0.0350
Epoch 27/50, Train Loss: 0.0349, Test Loss: 0.0350
Epoch 28/50, Train Loss: 0.0348, Test Loss: 0.0348
Epoch 29/50, Train Loss: 0.0348, Test Loss: 0.0348
Epoch 30/50, Train Loss: 0.0347, Test Loss: 0.0348
Epoch 31/50, Train Loss: 0.0345, Test Loss: 0.0346
Epoch 32/50, Train Loss: 0.0345, Test Loss: 0.0348
Epoch 33/50, Train Loss: 0.0344, Test Loss: 0.0348
Epoch 34/50, Train Loss: 0.0344, Test Loss: 0.0344
Epoch 35/50, Train Loss: 0.0343, Test Loss: 0.0350
Epoch 36/50, Train Loss: 0.0342, Test Loss: 0.0344
Epoch 37/50, Train Loss: 0.0342, Test Loss: 0.0342
Epoch 38/50, Train Loss: 0.0341, Test Loss: 0.0342
Epoch 39/50, Train Loss: 0.0341, Test Loss: 0.0343
Epoch 40/50, Train Loss: 0.0340, Test Loss: 0.0342
Epoch 41/50, Train Loss: 0.0339, Test Loss: 0.0344
Epoch 42/50, Train Loss: 0.0339, Test Loss: 0.0343
Epoch 43/50, Train Loss: 0.0339, Test Loss: 0.0345
Epoch 44/50, Train Loss: 0.0339, Test Loss: 0.0341
Epoch 45/50, Train Loss: 0.0338, Test Loss: 0.0339
Epoch 46/50, Train Loss: 0.0337, Test Loss: 0.0340
Epoch 47/50, Train Loss: 0.0337, Test Loss: 0.0338
Epoch 48/50, Train Loss: 0.0337, Test Loss: 0.0338
Epoch 49/50, Train Loss: 0.0336, Test Loss: 0.0338
Epoch 50/50, Train Loss: 0.0336, Test Loss: 0.0339
```



In [46]:
```
normalized_random_inputs = ((torch.randn(10, 2))).to(device)
with torch.no_grad():
    normalized_generated_images = VAEmodel.decoder((normalized_random_inputs)).cpu().vie

plt.figure(figsize=(20, 4))
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(normalized_generated_images[i], cmap='gray')
    plt.axis('off')
plt.show()
```



In [16]:
```
class discriminator(nn.Module):
    def __init__(self):
        super(discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        x = self.model(x)
        return x
```

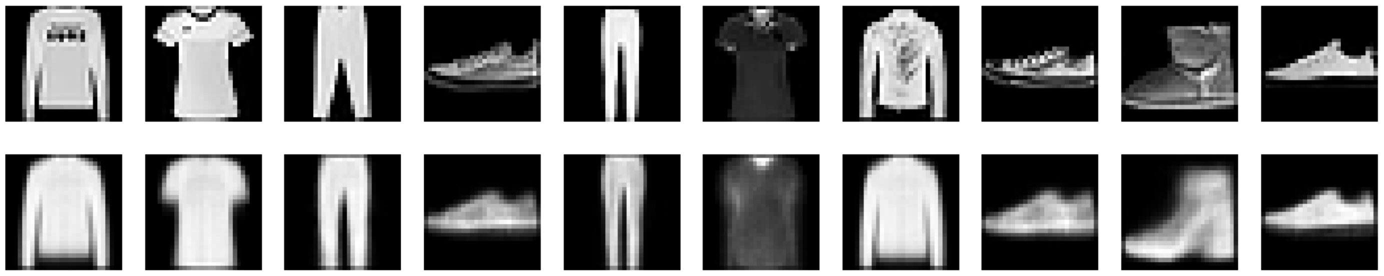
In [17]:
```
class VAE_GAN(nn.Module):
    def __init__(self):
        super(VAE_GAN, self).__init__()
        self.VAE= VAE().to(device)
        self.discriminator = discriminator().to(device)
    def forward(self, x):
        x_shape=x.shape
        x=torch.flatten(x, start_dim=1)
        z, mu, logvar= self.VAE(x)
        discriminator_output = self.discriminator(z)
        z = torch.reshape(z, x_shape)
        return discriminator_output, mu, logvar, z
```

In [18]:
```
VAEGANmodel = VAE_GAN().to(device)
criterion = nn.MSELoss()
adversarial_loss = nn.BCELoss()
VAEGANoptimizer = torch.optim.Adam(VAEGANmodel.VAE.parameters(), lr=1e-4)
Discrinimator_optimizer = torch.optim.Adam(VAEGANmodel.discriminator.parameters(), lr=1e
```

In [19]:
```
def train_VAEGAN(model,  vanganoptimizer, discriminator_opt, num_epochs):
    for epoch in range(num_epochs):
        model.train()
        vae_loss=0
        disc_loss=0
        for data in trainloader:
            inputs, _ = data
            inputs = inputs.to(device)
            discriminator_output,mu,logvar,outputs = model(inputs)
            vaeloss = loss_function(outputs, inputs,mu,logvar)
            vanganoptimizer.zero_grad()
            vaeloss.backward()
            vanganoptimizer.step()
            vae_loss += vaeloss.item()

            discriminator_opt.zero_grad()
            disc_output,_,_,_ = model(inputs)
            real_labels = torch.ones(inputs.size(0), 1).to(device)
            fake_labels = torch.zeros(inputs.size(0), 1).to(device)
            real_loss = adversarial_loss(disc_output, real_labels)
            fake_inputs = torch.randn(inputs.size(0), 2).to(device)
            fake_images = model.VAE.decoder(fake_inputs)
            disc_output,_,_,_ = model(fake_images.detach())
            fake_loss = adversarial_loss(disc_output, fake_labels)
            discloss = real_loss + fake_loss
            discloss.backward()
            disc_loss+= real_loss.item() + fake_loss.item()
            discriminator_opt.step()
        disc_loss = disc_loss / len(trainloader)
        vae_loss = vae_loss / len(trainloader)
        print(f'Epoch {epoch+1}/{num_epochs}, VAE Loss: {vae_loss:.4f}, Discriminator Lo

    return model

model2 = train_VAEGAN(VAEGANmodel, VAEGANoptimizer, Discrinimator_optimizer, num_epochs=
```

```
Epoch 1/50, VAE Loss: 0.0839, Discriminator Loss: 1.3848
Epoch 2/50, VAE Loss: 0.0602, Discriminator Loss: 1.3817
Epoch 3/50, VAE Loss: 0.0534, Discriminator Loss: 1.3768
Epoch 4/50, VAE Loss: 0.0470, Discriminator Loss: 1.3753
Epoch 5/50, VAE Loss: 0.0429, Discriminator Loss: 1.3662
Epoch 6/50, VAE Loss: 0.0404, Discriminator Loss: 1.3470
Epoch 7/50, VAE Loss: 0.0391, Discriminator Loss: 1.3349
Epoch 8/50, VAE Loss: 0.0383, Discriminator Loss: 1.3345
Epoch 9/50, VAE Loss: 0.0377, Discriminator Loss: 1.3364
Epoch 10/50, VAE Loss: 0.0373, Discriminator Loss: 1.3423
Epoch 11/50, VAE Loss: 0.0369, Discriminator Loss: 1.3446
Epoch 12/50, VAE Loss: 0.0367, Discriminator Loss: 1.3485
Epoch 13/50, VAE Loss: 0.0365, Discriminator Loss: 1.3491
Epoch 14/50, VAE Loss: 0.0363, Discriminator Loss: 1.3506
Epoch 15/50, VAE Loss: 0.0361, Discriminator Loss: 1.3513
Epoch 16/50, VAE Loss: 0.0359, Discriminator Loss: 1.3532
Epoch 17/50, VAE Loss: 0.0357, Discriminator Loss: 1.3535
Epoch 18/50, VAE Loss: 0.0357, Discriminator Loss: 1.3527
Epoch 19/50, VAE Loss: 0.0355, Discriminator Loss: 1.3553
Epoch 20/50, VAE Loss: 0.0354, Discriminator Loss: 1.3550
Epoch 21/50, VAE Loss: 0.0353, Discriminator Loss: 1.3565
Epoch 22/50, VAE Loss: 0.0351, Discriminator Loss: 1.3538
Epoch 23/50, VAE Loss: 0.0351, Discriminator Loss: 1.3568
Epoch 24/50, VAE Loss: 0.0350, Discriminator Loss: 1.3559
Epoch 25/50, VAE Loss: 0.0349, Discriminator Loss: 1.3554
Epoch 26/50, VAE Loss: 0.0348, Discriminator Loss: 1.3576
Epoch 27/50, VAE Loss: 0.0347, Discriminator Loss: 1.3575
Epoch 28/50, VAE Loss: 0.0346, Discriminator Loss: 1.3571
Epoch 29/50, VAE Loss: 0.0346, Discriminator Loss: 1.3583
Epoch 30/50, VAE Loss: 0.0345, Discriminator Loss: 1.3579
Epoch 31/50, VAE Loss: 0.0344, Discriminator Loss: 1.3580
Epoch 32/50, VAE Loss: 0.0343, Discriminator Loss: 1.3587
Epoch 33/50, VAE Loss: 0.0343, Discriminator Loss: 1.3588
Epoch 34/50, VAE Loss: 0.0342, Discriminator Loss: 1.3584
Epoch 35/50, VAE Loss: 0.0341, Discriminator Loss: 1.3594
Epoch 36/50, VAE Loss: 0.0341, Discriminator Loss: 1.3586
Epoch 37/50, VAE Loss: 0.0340, Discriminator Loss: 1.3582
Epoch 38/50, VAE Loss: 0.0340, Discriminator Loss: 1.3586
Epoch 39/50, VAE Loss: 0.0339, Discriminator Loss: 1.3586
Epoch 40/50, VAE Loss: 0.0338, Discriminator Loss: 1.3583
Epoch 41/50, VAE Loss: 0.0338, Discriminator Loss: 1.3583
Epoch 42/50, VAE Loss: 0.0338, Discriminator Loss: 1.3580
Epoch 43/50, VAE Loss: 0.0337, Discriminator Loss: 1.3587
Epoch 44/50, VAE Loss: 0.0337, Discriminator Loss: 1.3593
Epoch 45/50, VAE Loss: 0.0336, Discriminator Loss: 1.3580
Epoch 46/50, VAE Loss: 0.0336, Discriminator Loss: 1.3581
Epoch 47/50, VAE Loss: 0.0335, Discriminator Loss: 1.3587
Epoch 48/50, VAE Loss: 0.0335, Discriminator Loss: 1.3591
Epoch 49/50, VAE Loss: 0.0335, Discriminator Loss: 1.3595
Epoch 50/50, VAE Loss: 0.0334, Discriminator Loss: 1.3609
```

In [27]:
```python
def visualize_reconstructions_vae_gan(model, dataloader, num_images=10):
    device = next(model.parameters()).device
    model.eval()
    indices = random.sample(range(len(dataloader.dataset)), num_images)
    images = torch.stack([dataloader.dataset[i][0] for i in indices]).to(device)

    with torch.no_grad():
        _, _, _, reconstructions = model(images)
    images = images.cpu().view(-1, 28, 28).numpy()
    reconstructions = reconstructions.cpu().view(-1, 28, 28).numpy()

    fig, axs = plt.subplots(2, num_images, figsize=(20, 4))
    for i in range(num_images):
        axs[0, i].imshow(images[i], cmap='gray')
        axs[0, i].axis('off')
        axs[1, i].imshow(reconstructions[i], cmap='gray')
        axs[1, i].axis('off')
    plt.show()

visualize_reconstructions_vae_gan(model2, testloader, num_images=10)
```



In [23]:

```python
normalized_random_inputs = ((torch.randn(10, 2))).to(device)
with torch.no_grad():
    normalized_generated_images = model2.VAE.decoder((normalized_random_inputs)).cpu().v

plt.figure(figsize=(20, 4))
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(normalized_generated_images[i], cmap='gray')
    plt.axis('off')
plt.show()
```