SVM

Question 1:

1. A soft-margin linear SVM with C = 0.02.

Corresponds to Fig. 1.4. The decision boundary of linear SVM is linear. In comparison with Fig. 1.3(problem 2), the line does not separate the two classes strictly, which corresponds to the case C is small and more errors are allowed.

2. A soft-margin linear SVM with C = 20.

Corresponds to Fig. 1.3. The decision boundary of linear SVM is linear. In comparison with Fig. 1.4(problem 1), the line separates two classes strictly, which corresponds to the case C is big.

3. A hard-margin kernel SVM with k(u, v) = uTv + (uTv)2

Corresponds to Fig. 1.5. The decision function of quadratic kernel is given by f(x) = ∑ i αi(xi · x + (xi · x) 2) + b. Hence the decision boundary is f(x) = 0. Since f(x) is second order function of x, the curve can be ellipse or hyperbolic curve. Fig. 5 is hyperbolic curve.

4. A hard-margin kernel SVM with k(u, v) = exp(−5||u − v||22)

Corresponds to Fig. 1.6. We can write out the decision function as f(x) = i αi exp (−γ‖xi − x‖ 2) +b. If γ is small, then the kernel value is large if the distance 1 between the x and xi is large. This makes the classification good with many supporting vectors. If Fig. 1.6 corresponds to the case γ is large (-5), then it is easy to classify many circle point in the middle in Fig. 1.6. Hence, Fig. 1.6 corresponds to γ = -5 .

5. A hard-margin kernel SVM with k(u, v) = exp(−1/5 ||u − v||22)

Corresponds to Fig. 1.1. We can write out the decision function as f(x) = i αi exp (−γ‖xi − x‖ 2) +b. If γ is large, then the kernel value is quite small even if the distance 1 between the x and xi is small. This makes the classification hard with few supporting vectors. If Fig. 1.1 corresponds to the case γ is large (-5), then it is hard to classify many circle point in the middle in Fig. 1.1. Hence, Fig. 1.1 corresponds to γ = -1/5 .

Autoencoders

Question 2 Part 1:

Network: an autoencoder where the encoder and the decoder are linear models with bottleneck size of 2 dimensions and sigmoid activation function

With a LEARNING_RATE = 1e-3 and EPOCH_NUMBER= 15, the loss after training and validation is 0.7623(average training loss) and 0.7596(reconstruction error)

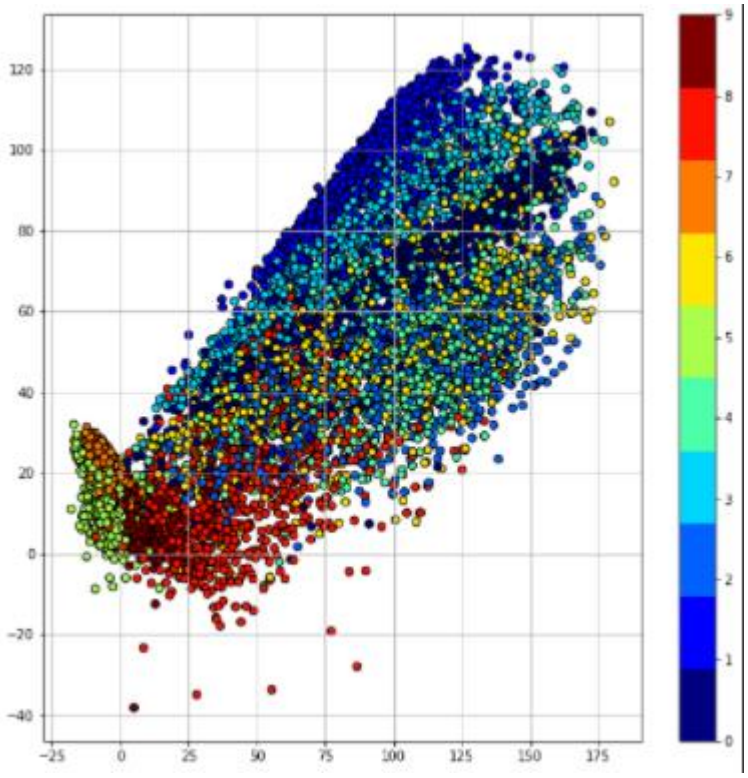
Code(architecture):

```python
class Autoencoder(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(Autoencoder,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                self.encoder = nn.Linear(28*28, output_size)

            def forward(self, x):
                x = x.flatten(1)
                encoded = self.encoder(x)
                return encoded

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 28*28),
                  nn.Sigmoid()
                )

            def forward(self, z):
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    def forward(self,x):
        # This function should be modified for the DAE and VAE
        x = self.encoder(x)
        x = self.decoder(x)
        # for the VAE forward function should also return mu and logvar
        return x
```

Training and validation losses:

```
100%|██████████| 1875/1875 [00:21<00:00, 85.87it/s]====> Epoch: 1 Average loss: 0.9148
100%|██████████| 313/313 [00:03<00:00, 101.19it/s]====> Val set loss (reconstruction error) : 0.8315
100%|██████████| 1875/1875 [00:22<00:00, 84.97it/s]====> Epoch: 2 Average loss: 0.8113
100%|██████████| 313/313 [00:03<00:00, 101.81it/s]====> Val set loss (reconstruction error) : 0.7924
100%|██████████| 1875/1875 [00:22<00:00, 85.04it/s]====> Epoch: 3 Average loss: 0.7886
100%|██████████| 313/313 [00:03<00:00, 98.17it/s] ====> Val set loss (reconstruction error) : 0.7782
100%|██████████| 1875/1875 [00:22<00:00, 84.94it/s]====> Epoch: 4 Average loss: 0.7776
100%|██████████| 313/313 [00:03<00:00, 99.41it/s] ====> Val set loss (reconstruction error) : 0.7708
100%|██████████| 1875/1875 [00:22<00:00, 84.85it/s]====> Epoch: 5 Average loss: 0.7714
100%|██████████| 313/313 [00:03<00:00, 99.55it/s] ====> Val set loss (reconstruction error) : 0.7661
100%|██████████| 1875/1875 [00:22<00:00, 84.94it/s]====> Epoch: 6 Average loss: 0.7678
100%|██████████| 313/313 [00:03<00:00, 99.61it/s]====> Val set loss (reconstruction error) : 0.7637
100%|██████████| 1875/1875 [00:22<00:00, 83.61it/s]====> Epoch: 7 Average loss: 0.7660
100%|██████████| 313/313 [00:03<00:00, 100.30it/s]====> Val set loss (reconstruction error) : 0.7625
100%|██████████| 1875/1875 [00:22<00:00, 84.52it/s]====> Epoch: 8 Average loss: 0.7650
100%|██████████| 313/313 [00:03<00:00, 100.11it/s]====> Val set loss (reconstruction error) : 0.7618
100%|██████████| 1875/1875 [00:22<00:00, 84.88it/s]====> Epoch: 9 Average loss: 0.7645
100%|██████████| 313/313 [00:03<00:00, 100.02it/s]====> Val set loss (reconstruction error) : 0.7614
100%|██████████| 1875/1875 [00:22<00:00, 84.14it/s]====> Epoch: 10 Average loss: 0.7641
100%|██████████| 313/313 [00:03<00:00, 99.76it/s]====> Val set loss (reconstruction error) : 0.7611
100%|██████████| 1875/1875 [00:22<00:00, 84.33it/s]====> Epoch: 11 Average loss: 0.7636
100%|██████████| 313/313 [00:03<00:00, 100.43it/s]====> Val set loss (reconstruction error) : 0.7601
100%|██████████| 1875/1875 [00:22<00:00, 83.79it/s]====> Epoch: 12 Average loss: 0.7628
100%|██████████| 313/313 [00:03<00:00, 99.87it/s] ====> Val set loss (reconstruction error) : 0.7599
100%|██████████| 1875/1875 [00:22<00:00, 84.37it/s]====> Epoch: 13 Average loss: 0.7626
100%|██████████| 313/313 [00:03<00:00, 99.19it/s]====> Val set loss (reconstruction error) : 0.7598
100%|██████████| 1875/1875 [00:22<00:00, 83.39it/s]====> Epoch: 14 Average loss: 0.7624
100%|██████████| 313/313 [00:03<00:00, 98.98it/s]====> Val set loss (reconstruction error) : 0.7597
100%|██████████| 1875/1875 [00:22<00:00, 84.15it/s]====> Epoch: 15 Average loss: 0.7623
100%|██████████| 313/313 [00:03<00:00, 98.34it/s] ====> Val set loss (reconstruction error) : 0.7596
```

Scatter Plots:



Question 2 Part 2:

Network: an autoencoder with bottleneck size of 2 dimensions and Sigmoid activation function

With a LEARNING_RATE = 1e-3 and EPOCH_NUMBER= 15, the loss after training and validation is 0.6548(average training loss) and 0.6553(reconstruction error)

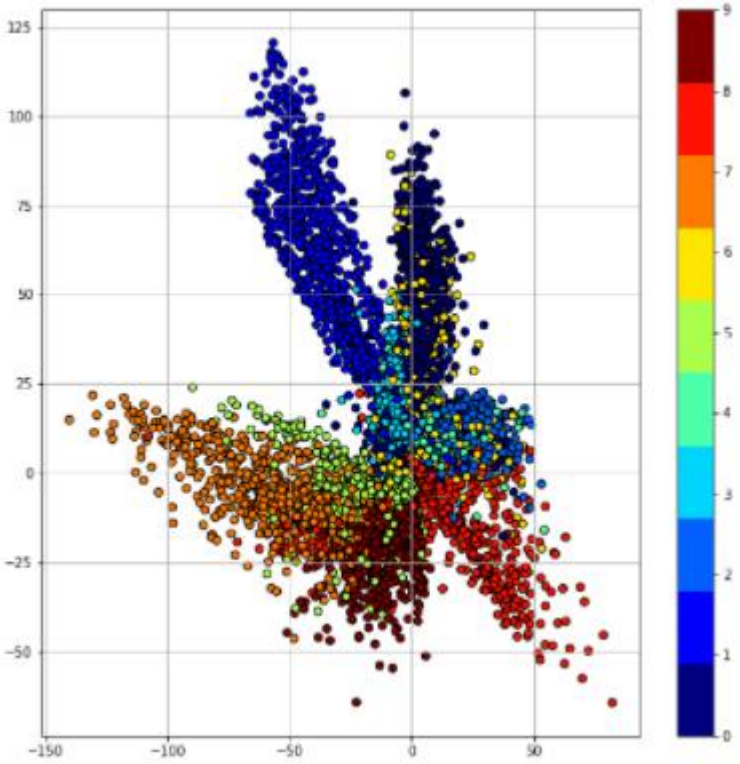
Code(architecture):

```python
class Autoencoder(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(Autoencoder,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder = nn.Sequential(
                  nn.Linear(28*28, 1024),
                  nn.ReLU(),
                  nn.Linear(1024, output_size)
                )

            def forward(self, x):
                # needs your implementation
                x = x.flatten(1)
                encoded = self.encoder(x)
                return encoded

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 1024),
                  nn.ReLU(),
                  nn.Linear(1024, 28*28),
                  nn.Sigmoid()
                )

            def forward(self, z):
                # needs your implementation
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    def forward(self,x):
        # This function should be modified for the DAE and VAE
        x = self.encoder(x)
        x = self.decoder(x)
        # for the VAE forward function should also return mu and logvar
        return x
```

Training and validation losses:

```
100%|██████████| 1875/1875 [00:24<00:00, 75.16it/s]====> Epoch: 1 Average loss: 0.6882
100%|██████████| 313/313 [00:03<00:00, 97.99it/s]====> Val set loss (reconstruction error) : 0.6683
100%|██████████| 1875/1875 [00:25<00:00, 74.03it/s]====> Epoch: 2 Average loss: 0.6672
100%|██████████| 313/313 [00:03<00:00, 98.09it/s]====> Val set loss (reconstruction error) : 0.6638
100%|██████████| 1875/1875 [00:25<00:00, 73.46it/s]====> Epoch: 3 Average loss: 0.6637
100%|██████████| 313/313 [00:03<00:00, 95.87it/s]====> Val set loss (reconstruction error) : 0.6617
100%|██████████| 1875/1875 [00:25<00:00, 73.35it/s]====> Epoch: 4 Average loss: 0.6619
100%|██████████| 313/313 [00:03<00:00, 97.77it/s]====> Val set loss (reconstruction error) : 0.6597
100%|██████████| 1875/1875 [00:25<00:00, 73.23it/s]====> Epoch: 5 Average loss: 0.6603
100%|██████████| 313/313 [00:03<00:00, 98.32it/s] ====> Val set loss (reconstruction error) : 0.6583
100%|██████████| 1875/1875 [00:25<00:00, 72.66it/s]====> Epoch: 6 Average loss: 0.6591
100%|██████████| 313/313 [00:03<00:00, 95.38it/s]====> Val set loss (reconstruction error) : 0.6578
100%|██████████| 1875/1875 [00:25<00:00, 72.88it/s]====> Epoch: 7 Average loss: 0.6585
100%|██████████| 313/313 [00:03<00:00, 94.89it/s]====> Val set loss (reconstruction error) : 0.6573
100%|██████████| 1875/1875 [00:25<00:00, 73.47it/s]====> Epoch: 8 Average loss: 0.6576
100%|██████████| 313/313 [00:03<00:00, 96.21it/s]====> Val set loss (reconstruction error) : 0.6569
100%|██████████| 1875/1875 [00:25<00:00, 72.93it/s]====> Epoch: 9 Average loss: 0.6568
100%|██████████| 313/313 [00:03<00:00, 95.65it/s]====> Val set loss (reconstruction error) : 0.6561
100%|██████████| 1875/1875 [00:25<00:00, 73.02it/s]====> Epoch: 10 Average loss: 0.6564
100%|██████████| 313/313 [00:03<00:00, 96.68it/s] ====> Val set loss (reconstruction error) : 0.6563
100%|██████████| 1875/1875 [00:25<00:00, 73.06it/s]====> Epoch: 11 Average loss: 0.6560
100%|██████████| 313/313 [00:03<00:00, 98.90it/s] ====> Val set loss (reconstruction error) : 0.6554
100%|██████████| 1875/1875 [00:25<00:00, 72.66it/s]====> Epoch: 12 Average loss: 0.6556
100%|██████████| 313/313 [00:03<00:00, 96.06it/s]====> Val set loss (reconstruction error) : 0.6557
100%|██████████| 1875/1875 [00:26<00:00, 72.00it/s]====> Epoch: 13 Average loss: 0.6553
100%|██████████| 313/313 [00:03<00:00, 94.82it/s] ====> Val set loss (reconstruction error) : 0.6545
100%|██████████| 1875/1875 [00:25<00:00, 72.69it/s]====> Epoch: 14 Average loss: 0.6552
100%|██████████| 313/313 [00:03<00:00, 95.62it/s] ====> Val set loss (reconstruction error) : 0.6542
100%|██████████| 1875/1875 [00:25<00:00, 73.08it/s]====> Epoch: 15 Average loss: 0.6548
100%|██████████| 313/313 [00:03<00:00, 96.24it/s] ====> Val set loss (reconstruction error) : 0.6553
```

Scatter plots:



Comparison:

The plot in this network (Question 2) is more grouped with colors binded together when compared with the scatter plot in network (Question 1). This is because of the reason that the network in Question 1 is just a linear layer with no hidden layers and in Question 2, the architecture is Linear layer with a hidden layer (simply more layers) to better categorize and encode the data. If we have more layers, each with an activation function, the data will be encoded with more information(information extraction will be better), so that the decoded outputs will be less blurred and less noisy.

Question 2 Part 3:

Network 1: an autoencoder with bottleneck size of 30 dimensions and tanh activation function

With a LEARNING_RATE = 1e-2 and EPOCH_NUMBER= 15, the loss after training and validation is 0.5348(average training loss) and 0.5343(reconstruction error)

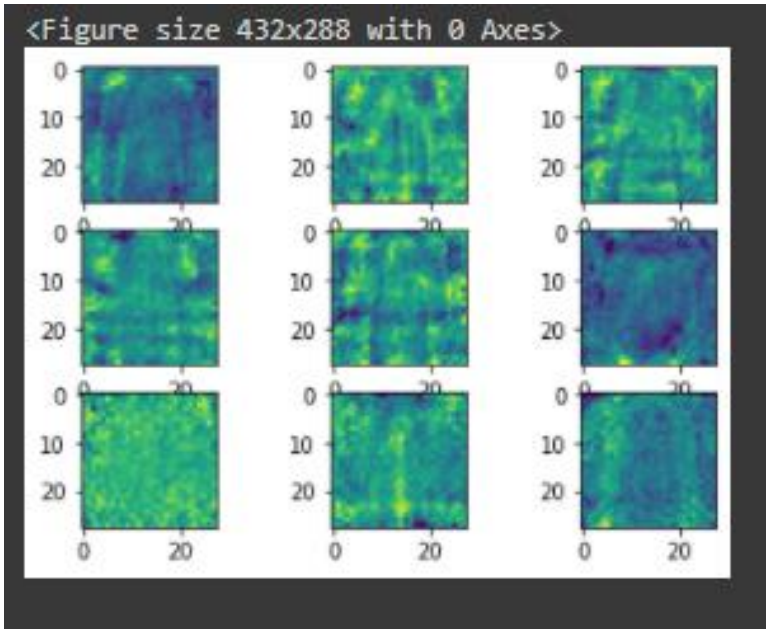
Code(architecture):

```python
class Autoencoder(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(Autoencoder,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder = nn.Sequential(nn.Linear(28*28, output_size))

            def forward(self, x):
                x = x.flatten(1)
                encoded = self.encoder(x)
                return encoded

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 28*28),
                  nn.Tanh()
              )

            def forward(self, z):
                # needs your implementation
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    def forward(self,x):
        # This function should be modified for the DAE and VAE
        x = self.encoder(x)
        x = self.decoder(x)
        # for the VAE forward function should also return mu and logvar
        return x
```

Training and validation losses:

```
100%|██████████| 1875/1875 [00:17<00:00, 107.22it/s]====> Epoch: 1 Average loss: 0.5624
100%|██████████| 313/313 [00:02<00:00, 129.17it/s]====> Val set loss (reconstruction error) : 0.5328
100%|██████████| 1875/1875 [00:17<00:00, 108.35it/s]====> Epoch: 2 Average loss: 0.5327
100%|██████████| 313/313 [00:02<00:00, 126.66it/s]====> Val set loss (reconstruction error) : 0.5322
100%|██████████| 1875/1875 [00:17<00:00, 108.51it/s]====> Epoch: 3 Average loss: 0.5345
100%|██████████| 313/313 [00:02<00:00, 125.62it/s]====> Val set loss (reconstruction error) : 0.5380
100%|██████████| 1875/1875 [00:17<00:00, 108.36it/s]====> Epoch: 4 Average loss: 0.5345
100%|██████████| 313/313 [00:02<00:00, 124.46it/s]====> Val set loss (reconstruction error) : 0.5315
100%|██████████| 1875/1875 [00:17<00:00, 109.66it/s]====> Epoch: 5 Average loss: 0.5346
100%|██████████| 313/313 [00:02<00:00, 127.71it/s]====> Val set loss (reconstruction error) : 0.5320
100%|██████████| 1875/1875 [00:17<00:00, 109.00it/s]
====> Epoch: 6 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 129.38it/s]====> Val set loss (reconstruction error) : 0.5292
100%|██████████| 1875/1875 [00:17<00:00, 109.51it/s]====> Epoch: 7 Average loss: 0.5347
100%|██████████| 313/313 [00:02<00:00, 125.28it/s]====> Val set loss (reconstruction error) : 0.5337
100%|██████████| 1875/1875 [00:17<00:00, 109.55it/s]====> Epoch: 8 Average loss: 0.5350
100%|██████████| 313/313 [00:02<00:00, 129.78it/s]====> Val set loss (reconstruction error) : 0.5322
100%|██████████| 1875/1875 [00:17<00:00, 107.95it/s]====> Epoch: 9 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 127.85it/s]====> Val set loss (reconstruction error) : 0.5355
100%|██████████| 1875/1875 [00:17<00:00, 108.26it/s]====> Epoch: 10 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 129.92it/s]====> Val set loss (reconstruction error) : 0.5308
100%|██████████| 1875/1875 [00:17<00:00, 108.91it/s]====> Epoch: 11 Average loss: 0.5347
100%|██████████| 313/313 [00:02<00:00, 129.93it/s]====> Val set loss (reconstruction error) : 0.5319
100%|██████████| 1875/1875 [00:17<00:00, 108.20it/s]====> Epoch: 12 Average loss: 0.5349
100%|██████████| 313/313 [00:02<00:00, 126.62it/s]====> Val set loss (reconstruction error) : 0.5293
100%|██████████| 1875/1875 [00:17<00:00, 107.34it/s]====> Epoch: 13 Average loss: 0.5349
100%|██████████| 313/313 [00:02<00:00, 128.61it/s]====> Val set loss (reconstruction error) : 0.5339
100%|██████████| 1875/1875 [00:17<00:00, 109.09it/s]====> Epoch: 14 Average loss: 0.5346
100%|██████████| 313/313 [00:02<00:00, 130.36it/s]====> Val set loss (reconstruction error) : 0.5302
100%|██████████| 1875/1875 [00:16<00:00, 111.85it/s]====> Epoch: 15 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 133.74it/s]====> Val set loss (reconstruction error) : 0.5343
```

Kernel plots:



Network 2: an autoencoder with bottleneck size of 30 dimensions and tanh activation function and adding isotropic gaussian noise to the inputs values.

With a LEARNING_RATE = 1e-2 and EPOCH_NUMBER= 15, the loss after training and validation is 0.5509(average training loss) and 0.5499(reconstruction error)

Code(architecture):

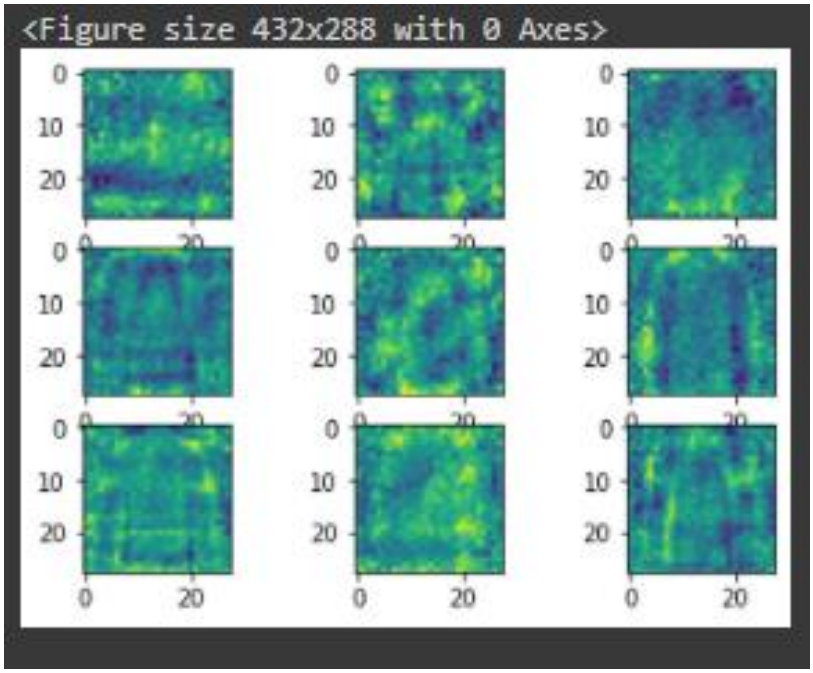
```python
class DAE(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(DAE,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder = nn.Sequential(nn.Linear(28*28, output_size))

            def forward(self, x):
                x = x.flatten(1)
                encoded = self.encoder(x)
                return encoded

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 28*28),
                  nn.Tanh()
              )

            def forward(self, z):
                # needs your implementation
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    # Implement this function for the DAE model
    def add_noise(self, x, noise_type):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        if noise_type=='Gaussian':
            return x + torch.normal(0.1, 0.5,x.size()).to(device)
        elif noise_type=='Dropout':
            m = nn.Dropout(p=0.2)
            return x + m(x)
```

```python
def forward(self,x):
    # This function should be modified for the DAE and VAE
    x = self.add_noise(x, 'Gaussian')
    x = self.encoder(x)
    x = self.decoder(x)
    # for the VAE forward function should also return mu and logvar
    return x
```

Training and validation losses:

```
100%|██████████| 1875/1875 [00:17<00:00, 109.38it/s]====> Epoch: 1 Average loss: 0.5839
100%|██████████| 313/313 [00:02<00:00, 125.78it/s]====> Val set loss (reconstruction error) : 0.5437
100%|██████████| 1875/1875 [00:17<00:00, 107.84it/s]====> Epoch: 2 Average loss: 0.5491
100%|██████████| 313/313 [00:02<00:00, 129.83it/s]====> Val set loss (reconstruction error) : 0.5535
100%|██████████| 1875/1875 [00:17<00:00, 109.82it/s]====> Epoch: 3 Average loss: 0.5509
100%|██████████| 313/313 [00:02<00:00, 126.16it/s]====> Val set loss (reconstruction error) : 0.5487
100%|██████████| 1875/1875 [00:16<00:00, 110.40it/s]====> Epoch: 4 Average loss: 0.5513
100%|██████████| 313/313 [00:02<00:00, 130.69it/s]====> Val set loss (reconstruction error) : 0.5488
100%|██████████| 1875/1875 [00:17<00:00, 107.71it/s]====> Epoch: 5 Average loss: 0.5509
100%|██████████| 313/313 [00:02<00:00, 128.06it/s]====> Val set loss (reconstruction error) : 0.5460
100%|██████████| 1875/1875 [00:17<00:00, 108.93it/s]====> Epoch: 6 Average loss: 0.5510
100%|██████████| 313/313 [00:02<00:00, 124.89it/s]====> Val set loss (reconstruction error) : 0.5502
100%|██████████| 1875/1875 [00:17<00:00, 108.05it/s]====> Epoch: 7 Average loss: 0.5511
100%|██████████| 313/313 [00:02<00:00, 128.73it/s]====> Val set loss (reconstruction error) : 0.5507
100%|██████████| 1875/1875 [00:16<00:00, 110.46it/s]====> Epoch: 8 Average loss: 0.5511
100%|██████████| 313/313 [00:02<00:00, 126.64it/s]====> Val set loss (reconstruction error) : 0.5546
100%|██████████| 1875/1875 [00:17<00:00, 109.72it/s]====> Epoch: 9 Average loss: 0.5508
100%|██████████| 313/313 [00:02<00:00, 128.47it/s]====> Val set loss (reconstruction error) : 0.5528
100%|██████████| 1875/1875 [00:17<00:00, 107.44it/s]====> Epoch: 10 Average loss: 0.5509
100%|██████████| 313/313 [00:02<00:00, 126.91it/s]====> Val set loss (reconstruction error) : 0.5439
100%|██████████| 1875/1875 [00:17<00:00, 108.31it/s]====> Epoch: 11 Average loss: 0.5512
100%|██████████| 313/313 [00:02<00:00, 123.86it/s]====> Val set loss (reconstruction error) : 0.5490
100%|██████████| 1875/1875 [00:17<00:00, 109.81it/s]====> Epoch: 12 Average loss: 0.5504
100%|██████████| 313/313 [00:02<00:00, 124.17it/s]====> Val set loss (reconstruction error) : 0.5507
100%|██████████| 1875/1875 [00:17<00:00, 108.97it/s]====> Epoch: 13 Average loss: 0.5513
100%|██████████| 313/313 [00:02<00:00, 127.91it/s]====> Val set loss (reconstruction error) : 0.5469
100%|██████████| 1875/1875 [00:16<00:00, 110.55it/s]====> Epoch: 14 Average loss: 0.5507
100%|██████████| 313/313 [00:02<00:00, 129.29it/s]====> Val set loss (reconstruction error) : 0.5511
100%|██████████| 1875/1875 [00:17<00:00, 108.98it/s]====> Epoch: 15 Average loss: 0.5509
100%|██████████| 313/313 [00:02<00:00, 126.20it/s]====> Val set loss (reconstruction error) : 0.5499
```

Kernel plots:



Network 3: autoencoder with bottleneck size of 30 dimensions and tanh activation function and turning some of the inputs values to zero.

With a LEARNING_RATE = 1e-2 and EPOCH_NUMBER= 15, the loss after training and validation is 0.7821(average training loss) and 0.7807(reconstruction error)

Code(architecture):

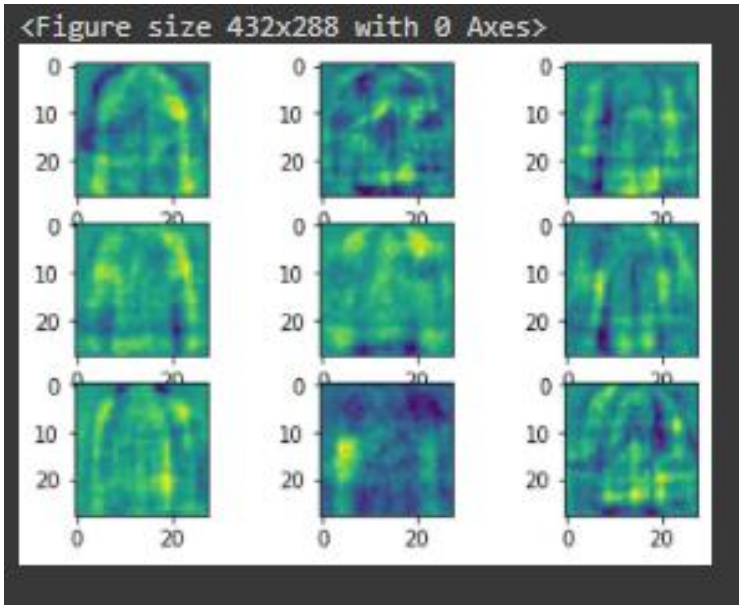
```python
class DAE(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(DAE,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder = nn.Sequential(nn.Linear(28*28, output_size))

            def forward(self, x):
                x = x.flatten(1)
                encoded = self.encoder(x)
                return encoded

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 28*28),
                  nn.Tanh()
                )

            def forward(self, z):
                # needs your implementation
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    # Implement this function for the DAE model
    def add_noise(self, x, noise_type):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        if noise_type=='Gaussian':
            return x + torch.normal(0.1, 0.5,x.size()).to(device)
        elif noise_type=='Dropout':
            m = nn.Dropout(p=0.2)
            return x + m(x)
```

```python
def forward(self,x):
    # This function should be modified for the DAE and VAE
    x = self.add_noise(x, 'Dropout')
    x = self.encoder(x)
    x = self.decoder(x)
    # for the VAE forward function should also return mu and logvar
    return x
```

Training and validation losses:

```
100%|██████████| 1875/1875 [00:17<00:00, 109.82it/s]====> Epoch: 1 Average loss: 0.8316
100%|██████████| 313/313 [00:02<00:00, 129.10it/s]====> Val set loss (reconstruction error) : 0.7890
100%|██████████| 1875/1875 [00:17<00:00, 108.53it/s]====> Epoch: 2 Average loss: 0.7954
100%|██████████| 313/313 [00:02<00:00, 125.62it/s]====> Val set loss (reconstruction error) : 0.7937
100%|██████████| 1875/1875 [00:17<00:00, 108.86it/s]====> Epoch: 3 Average loss: 0.7938
100%|██████████| 313/313 [00:02<00:00, 127.80it/s]====> Val set loss (reconstruction error) : 0.7944
100%|██████████| 1875/1875 [00:17<00:00, 108.95it/s]====> Epoch: 4 Average loss: 0.7914
100%|██████████| 313/313 [00:02<00:00, 127.49it/s]====> Val set loss (reconstruction error) : 0.7853
100%|██████████| 1875/1875 [00:16<00:00, 110.38it/s]====> Epoch: 5 Average loss: 0.7899
100%|██████████| 313/313 [00:02<00:00, 126.85it/s]====> Val set loss (reconstruction error) : 0.7826
100%|██████████| 1875/1875 [00:17<00:00, 108.13it/s]====> Epoch: 6 Average loss: 0.7874
100%|██████████| 313/313 [00:02<00:00, 125.73it/s]====> Val set loss (reconstruction error) : 0.7844
100%|██████████| 1875/1875 [00:17<00:00, 106.27it/s]====> Epoch: 7 Average loss: 0.7866
100%|██████████| 313/313 [00:02<00:00, 127.69it/s]====> Val set loss (reconstruction error) : 0.7839
100%|██████████| 1875/1875 [00:17<00:00, 109.46it/s]====> Epoch: 8 Average loss: 0.7854
100%|██████████| 313/313 [00:02<00:00, 129.25it/s]====> Val set loss (reconstruction error) : 0.7835
100%|██████████| 1875/1875 [00:17<00:00, 108.65it/s]====> Epoch: 9 Average loss: 0.7836
100%|██████████| 313/313 [00:02<00:00, 133.66it/s]====> Val set loss (reconstruction error) : 0.7870
100%|██████████| 1875/1875 [00:16<00:00, 111.29it/s]====> Epoch: 10 Average loss: 0.7849
100%|██████████| 313/313 [00:02<00:00, 128.52it/s]====> Val set loss (reconstruction error) : 0.7842
100%|██████████| 1875/1875 [00:16<00:00, 110.83it/s]====> Epoch: 11 Average loss: 0.7847
100%|██████████| 313/313 [00:02<00:00, 130.32it/s]====> Val set loss (reconstruction error) : 0.7856
100%|██████████| 1875/1875 [00:16<00:00, 111.64it/s]====> Epoch: 12 Average loss: 0.7831
100%|██████████| 313/313 [00:02<00:00, 134.24it/s]====> Val set loss (reconstruction error) : 0.7804
100%|██████████| 1875/1875 [00:16<00:00, 110.63it/s]====> Epoch: 13 Average loss: 0.7821
100%|██████████| 313/313 [00:02<00:00, 130.50it/s]====> Val set loss (reconstruction error) : 0.7838
100%|██████████| 1875/1875 [00:16<00:00, 110.34it/s]====> Epoch: 14 Average loss: 0.7832
100%|██████████| 313/313 [00:02<00:00, 129.16it/s]====> Val set loss (reconstruction error) : 0.7849
100%|██████████| 1875/1875 [00:17<00:00, 109.46it/s]====> Epoch: 15 Average loss: 0.7821
100%|██████████| 313/313 [00:02<00:00, 134.22it/s]====> Val set loss (reconstruction error) : 0.7807
```

Kernel plots:



Comparison:

The kernels differs as follows:

The kernel with normal autoencoder (without any noise) has the weights somewhat better than the ones with added noise, but this may lead to overfitting leading to easy decoding.

The kernel with gaussian noise has some proper weights with noise taken from the gaussian distribution making the denoising autoencoder better by reducing the problem of identity function.

The kernel with dropout makes the input drop some random values and this makes it not good because it may drop some important features of the object being encoded.

That way by purposefully adding noise, the risk of learning the identity function instead of extracting features is eliminated. Then, the model is trained to predict the original, uncorrupted data point as its output.

Question 2 Part 4:

Network 1: an autoencoder with bottleneck size of 30 dimensions and Tanh activation function

With a LEARNING_RATE = 1e-2 and EPOCH_NUMBER= 15, the loss after training and validation is 0.5348(average training loss) and 0.5324(reconstruction error)

Code(architecture):

```python
class Autoencoder(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(Autoencoder,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder = nn.Sequential(
                    nn.Linear(28*28, output_size)
              )

            def forward(self, x):
                x = x.flatten(1)
                encoded = self.encoder(x)
                return encoded

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 28*28),
                  nn.Tanh()
              )

            def forward(self, z):
                # needs your implementation
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    def forward(self,x):
        # This function should be modified for the DAE and VAE
        x = self.encoder(x)
        x = self.decoder(x)
        # for the VAE forward function should also return mu and logvar
        return x
```

Code for plotting 30D vectors in 2D scatter

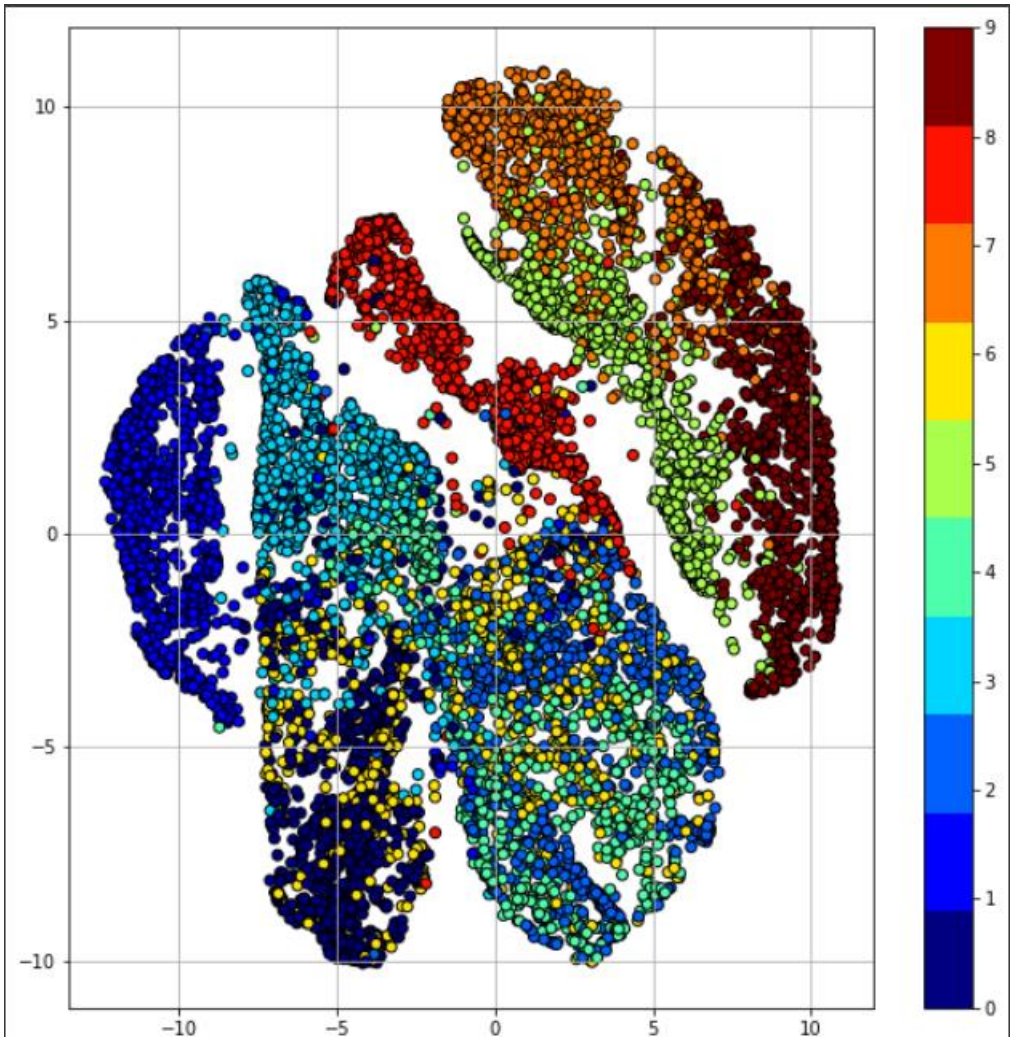
```python
with torch.no_grad():
    model = trainer.model
    model.eval()
    z=[];label=[]
    for x,y in trainer.val_loader:
        x = x.to(trainer.device)
        x = x.view(x.size(0), -1)
        z_ = model.encoder(x.to(trainer.device))
        z += z_.cpu().tolist()
        label += y.cpu().tolist()
    z = np.asarray(z)
    label = np.asarray(label)
    from sklearn.manifold import TSNE
    tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300, learning_rate='auto')
    transformed_z = tsne.fit_transform(z)


from autoencoder_starter import scatter_plot
scatter_plot(latent_representations=transformed_z,labels=label)
```

Training and validation losses:

```
100%|██████████| 1875/1875 [00:18<00:00, 102.44it/s]====> Epoch: 1 Average loss: 0.5646
100%|██████████| 313/313 [00:02<00:00, 122.45it/s]====> Val set loss (reconstruction error) : 0.5337
100%|██████████| 1875/1875 [00:18<00:00, 103.57it/s]====> Epoch: 2 Average loss: 0.5325
100%|██████████| 313/313 [00:02<00:00, 120.39it/s]====> Val set loss (reconstruction error) : 0.5302
100%|██████████| 1875/1875 [00:18<00:00, 100.90it/s]====> Epoch: 3 Average loss: 0.5344
100%|██████████| 313/313 [00:02<00:00, 122.29it/s]====> Val set loss (reconstruction error) : 0.5356
100%|██████████| 1875/1875 [00:18<00:00, 102.34it/s]====> Epoch: 4 Average loss: 0.5349
100%|██████████| 313/313 [00:02<00:00, 122.99it/s]====> Val set loss (reconstruction error) : 0.5371
100%|██████████| 1875/1875 [00:18<00:00, 104.09it/s]====> Epoch: 5 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 124.53it/s]====> Val set loss (reconstruction error) : 0.5303
100%|██████████| 1875/1875 [00:17<00:00, 104.31it/s]====> Epoch: 6 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 123.30it/s]====> Val set loss (reconstruction error) : 0.5325
100%|██████████| 1875/1875 [00:18<00:00, 104.09it/s]====> Epoch: 7 Average loss: 0.5346
100%|██████████| 313/313 [00:02<00:00, 122.06it/s]====> Val set loss (reconstruction error) : 0.5312
100%|██████████| 1875/1875 [00:18<00:00, 102.06it/s]====> Epoch: 8 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 120.60it/s]====> Val set loss (reconstruction error) : 0.5323
100%|██████████| 1875/1875 [00:18<00:00, 102.10it/s]====> Epoch: 9 Average loss: 0.5349
100%|██████████| 313/313 [00:02<00:00, 122.20it/s]====> Val set loss (reconstruction error) : 0.5334
100%|██████████| 1875/1875 [00:18<00:00, 102.40it/s]
====> Epoch: 10 Average loss: 0.5346
100%|██████████| 313/313 [00:02<00:00, 121.52it/s]====> Val set loss (reconstruction error) : 0.5299
100%|██████████| 1875/1875 [00:18<00:00, 101.29it/s]====> Epoch: 11 Average loss: 0.5347
100%|██████████| 313/313 [00:02<00:00, 121.76it/s]====> Val set loss (reconstruction error) : 0.5327
```

```
100%|██████████| 1875/1875 [00:18<00:00, 103.71it/s]====> Epoch: 12 Average loss: 0.5350
100%|██████████| 313/313 [00:02<00:00, 123.22it/s]====> Val set loss (reconstruction error) : 0.5308
100%|██████████| 1875/1875 [00:18<00:00, 102.14it/s]====> Epoch: 13 Average loss: 0.5349
100%|██████████| 313/313 [00:02<00:00, 119.74it/s]====> Val set loss (reconstruction error) : 0.5333
100%|██████████| 1875/1875 [00:18<00:00, 101.68it/s]====> Epoch: 14 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 121.36it/s]====> Val set loss (reconstruction error) : 0.5309
100%|██████████| 1875/1875 [00:18<00:00, 103.26it/s]====> Epoch: 15 Average loss: 0.5348
100%|██████████| 313/313 [00:02<00:00, 122.14it/s]====> Val set loss (reconstruction error) : 0.5324
```

latent space plots:



Network 2: an autoencoder with bottleneck size of 30 dimensions and Tanh activation function

With a LEARNING_RATE = 1e-2 and EPOCH_NUMBER= 15, the loss after training and validation is 0.9513(average training loss) and 0.9414(reconstruction error)

Code(architecture):

```python
class VAE(nn.Module):

    def __init__(self,dim_latent_representation=2):

        super(VAE,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder = nn.Sequential(
                    nn.Linear(28*28, output_size)
                )
                self.x1 = nn.Linear(output_size, output_size)
                self.x2 = nn.Linear(output_size, output_size)

            def forward(self, x):
                x = x.flatten(1)
                encoded = self.encoder(x)
                mu = self.x1(encoded)
                logvar = self.x2(encoded)
                return mu, logvar

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder = nn.Sequential(
                  nn.Linear(input_size, 28*28),
                  nn.Tanh()
                )

            def forward(self, z):
                # needs your implementation
                decoded = self.decoder(z)
                decoded = decoded.reshape(-1, 1, 28, 28)
                return decoded

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)
```

```python
# Implement this function for the VAE model
def reparameterise(self, mu, logvar):
    if self.training:
          # var = logvar.exp()
          # std = var.sqrt()
          # eps = Variable(torch.cuda.FloatTensor(std.size()).normal_())
          # return eps.mul(std).add(mu)
          std = torch.exp(logvar/2)
          eps = torch.randn_like(std)
          return mu + std * eps
    else:
        return mu

def forward(self,x):
    # This function should be modified for the DAE and VAE
    mu, logvar = self.encoder(x)
    reparameterised_data = self.reparameterise(mu, logvar)
    x = self.decoder(reparameterised_data)
    # for the VAE forward function should also return mu and logvar
    return x, mu, logvar
```

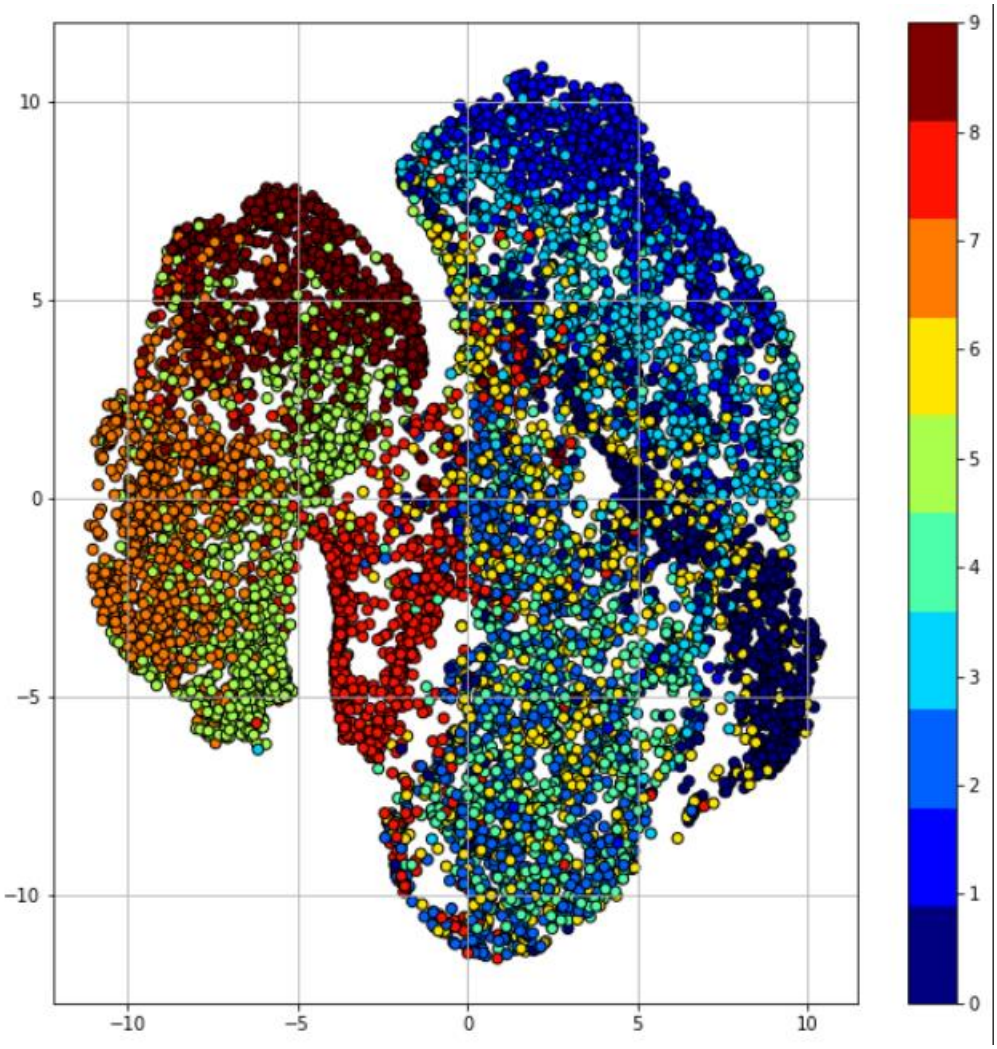
Code for plotting 30D vectors in 2D scatter

```
with torch.no_grad():
    model = trainer.model
    model.eval()
    z=[];label=[];z1=[];z2=[];z3=[]
    for x,y in trainer.val_loader:
        x = x.to(trainer.device)
        x = x.view(x.size(0), -1)
        z_ = model.encoder(x.to(trainer.device))
        z_ = model.reparameterise(z_[0], z_[1])
        z += z_.cpu().tolist()
        label += y.cpu().tolist()

    z = np.asarray(z)
    label = np.asarray(label)
    from sklearn.manifold import TSNE
    tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300, learning_rate='auto')
    transformed_z = tsne.fit_transform(z)

from autoencoder_starter import scatter_plot
scatter_plot(latent_representations=transformed_z, labels=label)
```

Training and validation losses:

```
  cpuset_checked))
100%|██████████| 1875/1875 [00:16<00:00, 112.39it/s]====> Epoch: 1 Average loss: 2.5389

100%|██████████| 313/313 [00:02<00:00, 139.20it/s]====> Val set loss (reconstruction error) : 1.0171

100%|██████████| 1875/1875 [00:16<00:00, 112.91it/s]====> Epoch: 2 Average loss: 0.9843

100%|██████████| 313/313 [00:02<00:00, 137.72it/s]====> Val set loss (reconstruction error) : 0.9635

100%|██████████| 1875/1875 [00:16<00:00, 114.50it/s]====> Epoch: 3 Average loss: 0.9597

100%|██████████| 313/313 [00:02<00:00, 138.53it/s]====> Val set loss (reconstruction error) : 0.9513

100%|██████████| 1875/1875 [00:16<00:00, 111.03it/s]====> Epoch: 4 Average loss: 0.9533

100%|██████████| 313/313 [00:02<00:00, 136.69it/s]====> Val set loss (reconstruction error) : 0.9483

100%|██████████| 1875/1875 [00:16<00:00, 111.49it/s]====> Epoch: 5 Average loss: 0.9517

100%|██████████| 313/313 [00:02<00:00, 134.14it/s]====> Val set loss (reconstruction error) : 0.9475

100%|██████████| 1875/1875 [00:16<00:00, 110.70it/s]====> Epoch: 6 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 137.36it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:16<00:00, 110.36it/s]====> Epoch: 7 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 137.33it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:16<00:00, 111.39it/s]====> Epoch: 8 Average loss: 0.9876

100%|██████████| 313/313 [00:02<00:00, 136.32it/s]====> Val set loss (reconstruction error) : 0.9480

100%|██████████| 1875/1875 [00:16<00:00, 111.58it/s]
====> Epoch: 9 Average loss: 0.9515
100%|██████████| 313/313 [00:02<00:00, 135.38it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:16<00:00, 111.11it/s]====> Epoch: 10 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 131.88it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:17<00:00, 109.96it/s]====> Epoch: 11 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 135.69it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:17<00:00, 109.54it/s]====> Epoch: 12 Average loss: 0.9514

100%|██████████| 313/313 [00:02<00:00, 137.67it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:17<00:00, 110.10it/s]====> Epoch: 13 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 135.30it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:17<00:00, 109.19it/s]====> Epoch: 14 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 132.70it/s]====> Val set loss (reconstruction error) : 0.9474

100%|██████████| 1875/1875 [00:16<00:00, 110.49it/s]====> Epoch: 15 Average loss: 0.9513

100%|██████████| 313/313 [00:02<00:00, 137.41it/s]====> Val set loss (reconstruction error) : 0.9474
```

latent space plots:



Comparison:

The latent features as shown in the TSNE graphs of two networks are different in the way that in Network1, the points are well distinguished from each other than the one from Network2(VAE). In Network2(VAE), the points are not well separated from each other.

VAE are autoencoders where we add some noise to the latent variable(z) by the way of adding a normal distribution to the encoded output. This is one of the reasons why there seems to be a normal distribution of points in the latent space for VAE Network.

VAE gives significant control over how we want to model our latent distribution. This kind of control does not exist in the usual AE framework. This is actually the power of Bayesian models themselves, VAEs are simply making it more practical and feasible for large-scale datasets. So, to conclude, if we want precise control over our latent representations and what we would like them to represent, then we choose VAE.