Assignment9

December 11, 2017

1 Autoencoders

In this assignment, we will look at autoencoders and compare their performance with an autoencoder-based PCA model. For this, we will continue to use the MNIST dataset that we have seen in the previous assignments.

As usual, the structure of the code is given to you, and you need to fill in the parts corresponding to the questions below.

PART A: In this part we will build three models: an autoencoder-based PCA, a 2-layer and a 3-layer autoencoder. We will then train and test these models on the MNIST dataset and compare their performance visually as well as by ccomputing the test set recontruction error. - Question 1 Autoencoder-based PCA: implement an autoencoder-based PCA model formed of two linear layers, using a structure similar to the 2-layer autoencoder model. - Question 2 2-layer autoencoder: Use different number of nodes (e.g.[128,16]), in hidden layers and compare the results. - Question 3 3-layer: after reading the basic autoencoder provided as an example, build a 3-layer autoencoder. Use different number of nodes (e.g.[128,64,16]), in hidden layers and compare the results.

PART B: In this part we will interpolate between two images in the latent space. This is done by first obtaining the encoded images:

$$latent\_1 = encoder(image\_1) \tag{1}$$

$$latent\_2 = encoder(image\_2) \tag{2}$$

and then generating the new interpolated image using the following equation:

$$interpolated\_image\_\lambda = decoder[\lambda * latent\_1 + (1 - \lambda) * latent\_2] \tag{3}$$

We will do this using the PCA model and then the autoencoder model and look at the differences between the two interpolations. Then, we need to get the image space representation and to verify that the interpolation generated using the 3-layer autoencoder is better. - Question 1 Fill interpolation() function in orde to 1.1. Get the latent space representation. 1.2. Interpolate between two images. 1.3. Get the image space represantion. - Question 2 Interpolate between an image corresponding to digit 1 and an image corresponding to digit 7 using the latent space representation obtained using the autoencoder-based PCA. Interpolate between an image corresponding to digit 1 and an image corresponding to digit 7 using the latent space representation obtained using the 3-layer autoencoder. Repeat the same procedure interpolating between images corresponding to digits 4 and 8.

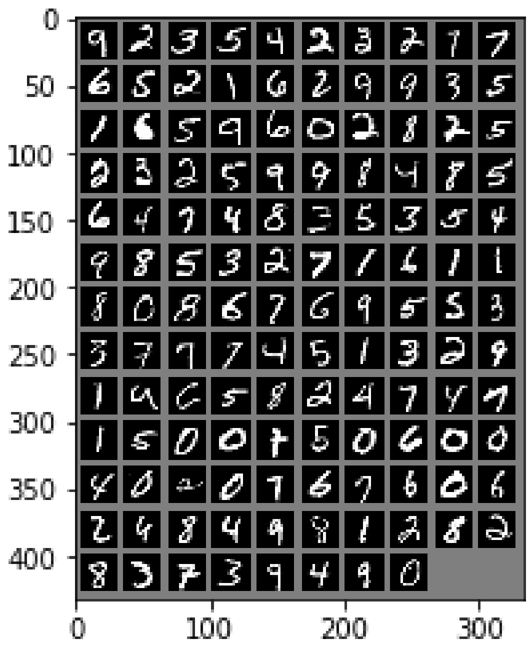
```
In [27]: # Import libraries

         import os
         import torch
         import torchvision
         from torch import nn
         from torch.autograd import Variable
         from torch.utils.data import DataLoader
         from torchvision import transforms
         import torchvision.datasets as dset
         from torchvision.utils import save_image
```

```
In [28]: # Import dataset
         batch_size = 128

         # Convert the data to Tensor and normalise by the mean and std  of the data set
         transform = transforms.Compose([transforms.ToTensor(),
                                         transforms.Normalize((0.5,), (0.5,))])

         # Import and normalize the train set in mini-batches
         train_set = torchvision.datasets.MNIST(root='./data', train=True,
                                                download=True, transform=transform)
         train_set_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
                                                      shuffle=True)

         # Import and normalize the test set in mini-batches
         test_set = torchvision.datasets.MNIST(root='./data', train=False,
                                                download=True, transform=transform)
         test_set_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size,
                                                      shuffle=True)
```

Visualise Visualise examples of the images from the dataset.

```
In [29]: # Import libraries for visualising
         import matplotlib.pyplot as plt
         import numpy as np

         # Load a batch of training images for visualising
         data_iterator = iter(train_set_loader)
         images, labels = data_iterator.next()

         # Create function for visualisation
         def show_image(img):
             # revert the normalisation for displaying the images in their original form
```

```
        img = img * 0.5 + 0.5
        # Convert to numpy for visualisation
        npimg = img.numpy()
        # Plot each image
        plt.imshow(np.transpose(npimg, (1, 2, 0)))


# Plot images as a grid using the 'make_grid' function
show_image(torchvision.utils.make_grid(images,10,5))
plt.show()
```



1.1 PART A:

1.2 Models

In this section we will implements three models as three different classes:

1. An autoencoder-based PCA.
2. A 2-layer autoencoder.
3. A 3-layer autoencoder.

1. Autoencoder-based PCA model Build an autoencoder-based PCA model. This will be made of two linear layers. The first linear layer is the encoder part. It receives as an input the image and has an output of shape num_hidden_1, where num_hidden_1 the number of hidden units in layer *L1* . The second linear layer is the decoder and has an output of the same shape as the images in the dataset, since its role is to reconstruct the image.

```
In [30]: class autoencoder_PCA(nn.Module):
             def __init__(self,num_in,num_hidden_1,num_out):
                 super(autoencoder_PCA, self).__init__()
                 ####### Question 1 #######
                 # The layers of the encoder

                 # The layers of the decoder

                 ####### End Question 1

             def forward(self, x):
                 x = self.encoder(x)
                 x = self.decoder(x)
                 return x
```

2. 2-layer autoencoder model This model is made of two parts:

a. **the encoder**: a layer that takes as an input the image and has an output of shape num_hidden_1 and a **ReLU** activation function. A second layer that has an output of shape num_hidden_2 and no activation function.

b. **the decoder**: this takes as an input the output of the encoder and has the same structure as a decoder, but in inverse order (the mirror image of the decoder). Additionally, the decoder applies a **tanh** activation function to the final layer.

This function is already given to you as an example of a basic autoencoder.

```
In [31]: class layer_2_autoencoder(nn.Module):
             def __init__(self,num_in,num_hidden_1,num_hidden_2,num_out):
                 super(layer_2_autoencoder, self).__init__()
                 # The layers of the encoder
                 self.encoder = nn.Sequential(
                     nn.Linear(num_in, num_hidden_1),
                     nn.ReLU(True),
                     nn.Linear(num_hidden_1, num_hidden_2))
                 # The layers of the decoder
                 self.decoder = nn.Sequential(
                     nn.Linear(num_hidden_2, num_hidden_1),
                     nn.ReLU(True),
                     nn.Linear(num_hidden_1, num_out),
                     nn.Tanh())

             # The forward pass
             def forward(self, x):
                 x = self.encoder(x)
                 x = self.decoder(x)
                 return x
```

2. 3-layer model This model is similar to the previous one, but with a deeper structure:

a. **the encoder**: a layer that takes as an input the image and has an output of shape num_hidden_1 and a **ReLU** activation function. A second layer that has an output of shape num_hidden_2 and a **ReLU** activation function. A final linear layer with output of shape num_hidden_3 and no activation function

b. **the decoder**: this takes as an input the output of the encoder and has the same structure as a decoder, but in inverse order (the mirror image of the decoder). Additionally, the decoder applies a **tanh** activation function to the final layer.

```
In [32]: class layer_3_autoencoder(nn.Module):
             def __init__(self,num_in,num_hidden_1,num_hidden_2,num_hidden_3,num_out):
                 super(layer_3_autoencoder, self).__init__()
                 ####### Question 3 #######
                 # The layers of the encoder

                 # The layers of the dencoder

                 ####### End Question 3 #######

             def forward(self, x):
                 x = self.encoder(x)
                 x = self.decoder(x)
                 return x
```

1.3 Train and test the models

Train function Now that we have build the models, let's define a function which performs the training. this function has the following parameters: the train set, the batch size, the number of epochs for which to perform the trainig, the learning rate, and the model to train.

```
In [33]: def train(train_set, batch_size, num_epochs, learning_rate, model):

             no_batches = int(np.round((len(train_set) / batch_size)))
             criterion = nn.MSELoss()
             optimizer = torch.optim.Adam(
             model.parameters(), lr=learning_rate, weight_decay=1e-5)

             losses = []
             for epoch in range(num_epochs):
                 train_loss = 0
                 for data in train_set_loader:
                     img, _ = data
                     img = img.view(img.size(0), -1)
                     img = Variable(img)

                     # =================== forward pass =====================
                     output = model(img)
```

```
                loss = criterion(output, img)

                # =================== backward pass ====================
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                train_loss += loss.data[0]

                # =================== log ========================
            print('====> Epoch: {} Average loss: {:.4f}'.format(
                   epoch, train_loss / no_batches))
            losses.append(train_loss / no_batches)
        return losses
```

Test function Similarly to the train function, build a test function that evaluates the performance of the models on the test set

```
In [34]: def evaluate_test(test_set, test_set_loader, batch_size, model):

             test_loss = 0
             criterion = nn.MSELoss()
             test_no_batches = np.round(int(len(test_set)/batch_size))

             for data in test_set_loader:
                     img, _ = data
                     img = img.view(img.size(0), -1)
                     img = Variable(img)

                     # =================== forward pass ====================
                     output = model(img)
                     loss = criterion(output, img)
                     test_loss += loss.data[0]

                 # =================== log ========================
             test_loss = test_loss/test_no_batches
             print('====> Test loss: {:.4f}'.format(test_loss))
             return test_loss
```

Now that we have written the train and test functions, let's train and then compare the reconstruction error on the train and test set of the three models. For each model, plot the training error.

1.3.1 1. Autoencoder-based PCA model

Train and test an autoencoder-based PCA with num_hidden_1=16 hidden units in the hidden layer.

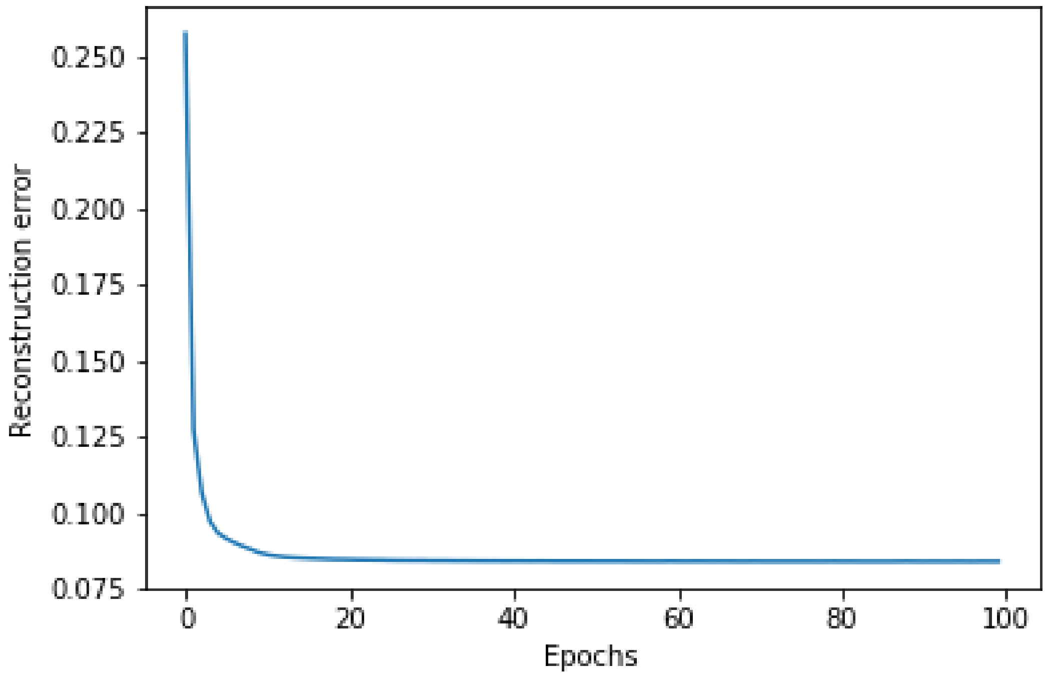
```
In [35]: #define the model
         num_input=28*28
```

```
num_hidden_1=16
num_output=28*28
model1 = autoencoder_PCA(num_input,num_hidden_1,num_output)

#train the model
model1_loss_train = train(train_set= train_set, batch_size = 128, num_epochs = 100, lea

# Plot the training error
fig = plt.figure()
plt.plot(model1_loss_train)
plt.xlabel('Epochs')
plt.ylabel('Reconstruction error')
plt.show()

# Calculate the loss in test set
model1_loss_test= evaluate_test(test_set = test_set, test_set_loader=  test_set_loader,
```

```
====> Epoch: 0 Average loss: 0.2573
====> Epoch: 1 Average loss: 0.1273
====> Epoch: 2 Average loss: 0.1066
====> Epoch: 3 Average loss: 0.0972
====> Epoch: 4 Average loss: 0.0933
====> Epoch: 5 Average loss: 0.0915
====> Epoch: 6 Average loss: 0.0902
====> Epoch: 7 Average loss: 0.0890
====> Epoch: 8 Average loss: 0.0880
====> Epoch: 9 Average loss: 0.0869
====> Epoch: 10 Average loss: 0.0862
====> Epoch: 11 Average loss: 0.0858
====> Epoch: 12 Average loss: 0.0855
====> Epoch: 13 Average loss: 0.0853
====> Epoch: 14 Average loss: 0.0852
====> Epoch: 15 Average loss: 0.0850
====> Epoch: 16 Average loss: 0.0849
====> Epoch: 17 Average loss: 0.0848
====> Epoch: 18 Average loss: 0.0847
====> Epoch: 19 Average loss: 0.0847
====> Epoch: 20 Average loss: 0.0846
====> Epoch: 21 Average loss: 0.0846
====> Epoch: 22 Average loss: 0.0845
====> Epoch: 23 Average loss: 0.0845
====> Epoch: 24 Average loss: 0.0845
====> Epoch: 25 Average loss: 0.0844
====> Epoch: 26 Average loss: 0.0844
====> Epoch: 27 Average loss: 0.0844
====> Epoch: 28 Average loss: 0.0843
====> Epoch: 29 Average loss: 0.0844
====> Epoch: 30 Average loss: 0.0843
```

```
====> Epoch: 31 Average loss: 0.0843
====> Epoch: 32 Average loss: 0.0843
====> Epoch: 33 Average loss: 0.0842
====> Epoch: 34 Average loss: 0.0842
====> Epoch: 35 Average loss: 0.0842
====> Epoch: 36 Average loss: 0.0842
====> Epoch: 37 Average loss: 0.0842
====> Epoch: 38 Average loss: 0.0842
====> Epoch: 39 Average loss: 0.0842
====> Epoch: 40 Average loss: 0.0842
====> Epoch: 41 Average loss: 0.0842
====> Epoch: 42 Average loss: 0.0841
====> Epoch: 43 Average loss: 0.0842
====> Epoch: 44 Average loss: 0.0841
====> Epoch: 45 Average loss: 0.0841
====> Epoch: 46 Average loss: 0.0841
====> Epoch: 47 Average loss: 0.0841
====> Epoch: 48 Average loss: 0.0841
====> Epoch: 49 Average loss: 0.0841
====> Epoch: 50 Average loss: 0.0841
====> Epoch: 51 Average loss: 0.0841
====> Epoch: 52 Average loss: 0.0841
====> Epoch: 53 Average loss: 0.0841
====> Epoch: 54 Average loss: 0.0841
====> Epoch: 55 Average loss: 0.0841
====> Epoch: 56 Average loss: 0.0841
====> Epoch: 57 Average loss: 0.0840
====> Epoch: 58 Average loss: 0.0840
====> Epoch: 59 Average loss: 0.0841
====> Epoch: 60 Average loss: 0.0841
====> Epoch: 61 Average loss: 0.0840
====> Epoch: 62 Average loss: 0.0841
====> Epoch: 63 Average loss: 0.0841
====> Epoch: 64 Average loss: 0.0840
====> Epoch: 65 Average loss: 0.0841
====> Epoch: 66 Average loss: 0.0840
====> Epoch: 67 Average loss: 0.0840
====> Epoch: 68 Average loss: 0.0840
====> Epoch: 69 Average loss: 0.0840
====> Epoch: 70 Average loss: 0.0840
====> Epoch: 71 Average loss: 0.0840
====> Epoch: 72 Average loss: 0.0840
====> Epoch: 73 Average loss: 0.0840
====> Epoch: 74 Average loss: 0.0840
====> Epoch: 75 Average loss: 0.0840
====> Epoch: 76 Average loss: 0.0840
====> Epoch: 77 Average loss: 0.0840
====> Epoch: 78 Average loss: 0.0840
```

```
====> Epoch: 79 Average loss: 0.0840
====> Epoch: 80 Average loss: 0.0840
====> Epoch: 81 Average loss: 0.0840
====> Epoch: 82 Average loss: 0.0840
====> Epoch: 83 Average loss: 0.0840
====> Epoch: 84 Average loss: 0.0840
====> Epoch: 85 Average loss: 0.0840
====> Epoch: 86 Average loss: 0.0840
====> Epoch: 87 Average loss: 0.0840
====> Epoch: 88 Average loss: 0.0840
====> Epoch: 89 Average loss: 0.0840
====> Epoch: 90 Average loss: 0.0840
====> Epoch: 91 Average loss: 0.0840
====> Epoch: 92 Average loss: 0.0840
====> Epoch: 93 Average loss: 0.0840
====> Epoch: 94 Average loss: 0.0840
====> Epoch: 95 Average loss: 0.0840
====> Epoch: 96 Average loss: 0.0840
====> Epoch: 97 Average loss: 0.0840
====> Epoch: 98 Average loss: 0.0840
====> Epoch: 99 Average loss: 0.0840
```



```
====> Test loss: 0.0828
```

1.3.2 2. 2-layer autoencoder

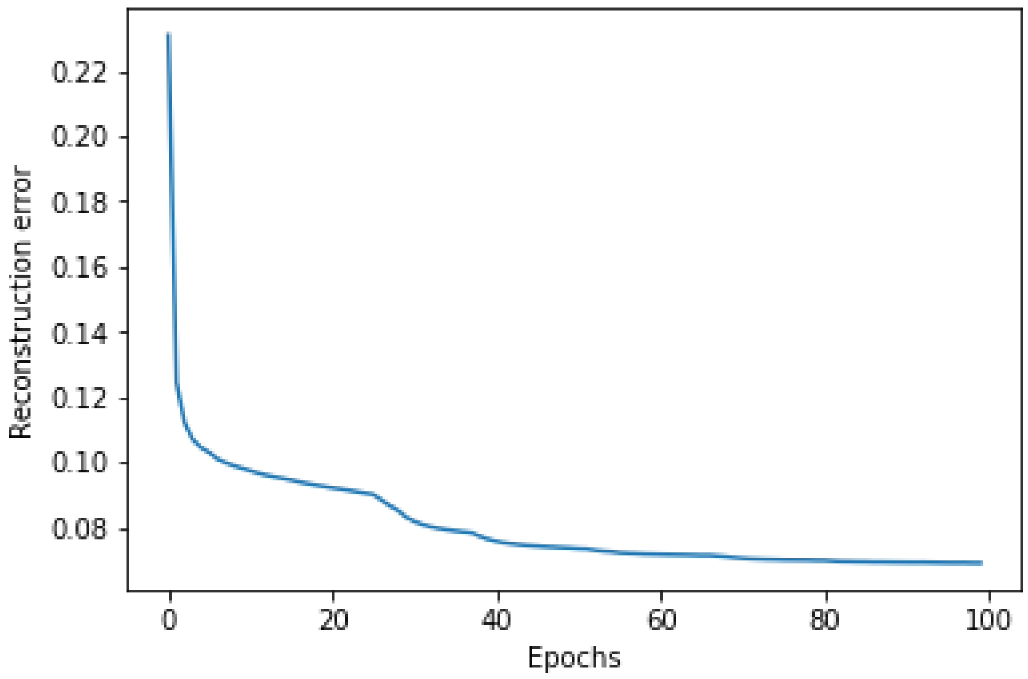
Train and test an autoencoder-based PCA with num_hidden_1=32 and num_hidden_2=16 hidden units in the hidden layers.

```
In [36]: #define the model
         num_input=28*28
         num_hidden_1=32
         num_hidden_2=16
         num_output=28*28

         model2 = layer_2_autoencoder(num_input,num_hidden_1,num_hidden_2,num_output)

         #train the model
         model2_loss_train = train(train_set= train_set, batch_size = 128, num_epochs = 100, lea

         # Plot the error
         fig = plt.figure()
         plt.plot(model2_loss_train)
         plt.xlabel('Epochs')
         plt.ylabel('Reconstruction error')
         plt.show()

         #calculate the loss in test set
         model2_loss_test= evaluate_test(test_set = test_set, test_set_loader=  test_set_loader,
```

```
====> Epoch: 0 Average loss: 0.2310
====> Epoch: 1 Average loss: 0.1244
====> Epoch: 2 Average loss: 0.1120
====> Epoch: 3 Average loss: 0.1069
====> Epoch: 4 Average loss: 0.1044
====> Epoch: 5 Average loss: 0.1029
====> Epoch: 6 Average loss: 0.1010
====> Epoch: 7 Average loss: 0.0998
====> Epoch: 8 Average loss: 0.0989
====> Epoch: 9 Average loss: 0.0981
====> Epoch: 10 Average loss: 0.0973
====> Epoch: 11 Average loss: 0.0966
====> Epoch: 12 Average loss: 0.0959
====> Epoch: 13 Average loss: 0.0954
====> Epoch: 14 Average loss: 0.0950
====> Epoch: 15 Average loss: 0.0945
====> Epoch: 16 Average loss: 0.0939
====> Epoch: 17 Average loss: 0.0934
====> Epoch: 18 Average loss: 0.0929
====> Epoch: 19 Average loss: 0.0925
====> Epoch: 20 Average loss: 0.0921
====> Epoch: 21 Average loss: 0.0917
====> Epoch: 22 Average loss: 0.0913
```

```
====> Epoch: 23 Average loss: 0.0909
====> Epoch: 24 Average loss: 0.0905
====> Epoch: 25 Average loss: 0.0901
====> Epoch: 26 Average loss: 0.0884
====> Epoch: 27 Average loss: 0.0866
====> Epoch: 28 Average loss: 0.0852
====> Epoch: 29 Average loss: 0.0831
====> Epoch: 30 Average loss: 0.0818
====> Epoch: 31 Average loss: 0.0808
====> Epoch: 32 Average loss: 0.0801
====> Epoch: 33 Average loss: 0.0796
====> Epoch: 34 Average loss: 0.0792
====> Epoch: 35 Average loss: 0.0789
====> Epoch: 36 Average loss: 0.0785
====> Epoch: 37 Average loss: 0.0783
====> Epoch: 38 Average loss: 0.0772
====> Epoch: 39 Average loss: 0.0763
====> Epoch: 40 Average loss: 0.0757
====> Epoch: 41 Average loss: 0.0752
====> Epoch: 42 Average loss: 0.0749
====> Epoch: 43 Average loss: 0.0746
====> Epoch: 44 Average loss: 0.0744
====> Epoch: 45 Average loss: 0.0742
====> Epoch: 46 Average loss: 0.0740
====> Epoch: 47 Average loss: 0.0739
====> Epoch: 48 Average loss: 0.0737
====> Epoch: 49 Average loss: 0.0736
====> Epoch: 50 Average loss: 0.0734
====> Epoch: 51 Average loss: 0.0733
====> Epoch: 52 Average loss: 0.0729
====> Epoch: 53 Average loss: 0.0727
====> Epoch: 54 Average loss: 0.0725
====> Epoch: 55 Average loss: 0.0721
====> Epoch: 56 Average loss: 0.0720
====> Epoch: 57 Average loss: 0.0719
====> Epoch: 58 Average loss: 0.0718
====> Epoch: 59 Average loss: 0.0717
====> Epoch: 60 Average loss: 0.0717
====> Epoch: 61 Average loss: 0.0716
====> Epoch: 62 Average loss: 0.0715
====> Epoch: 63 Average loss: 0.0715
====> Epoch: 64 Average loss: 0.0714
====> Epoch: 65 Average loss: 0.0714
====> Epoch: 66 Average loss: 0.0714
====> Epoch: 67 Average loss: 0.0712
====> Epoch: 68 Average loss: 0.0709
====> Epoch: 69 Average loss: 0.0707
====> Epoch: 70 Average loss: 0.0705
```

```
====> Epoch: 71 Average loss: 0.0703
====> Epoch: 72 Average loss: 0.0702
====> Epoch: 73 Average loss: 0.0701
====> Epoch: 74 Average loss: 0.0701
====> Epoch: 75 Average loss: 0.0700
====> Epoch: 76 Average loss: 0.0700
====> Epoch: 77 Average loss: 0.0699
====> Epoch: 78 Average loss: 0.0699
====> Epoch: 79 Average loss: 0.0698
====> Epoch: 80 Average loss: 0.0698
====> Epoch: 81 Average loss: 0.0696
====> Epoch: 82 Average loss: 0.0695
====> Epoch: 83 Average loss: 0.0695
====> Epoch: 84 Average loss: 0.0694
====> Epoch: 85 Average loss: 0.0694
====> Epoch: 86 Average loss: 0.0693
====> Epoch: 87 Average loss: 0.0693
====> Epoch: 88 Average loss: 0.0693
====> Epoch: 89 Average loss: 0.0693
====> Epoch: 90 Average loss: 0.0692
====> Epoch: 91 Average loss: 0.0692
====> Epoch: 92 Average loss: 0.0692
====> Epoch: 93 Average loss: 0.0692
====> Epoch: 94 Average loss: 0.0691
====> Epoch: 95 Average loss: 0.0691
====> Epoch: 96 Average loss: 0.0691
====> Epoch: 97 Average loss: 0.0691
====> Epoch: 98 Average loss: 0.0690
====> Epoch: 99 Average loss: 0.0690
```



```
====> Test loss: 0.0680
```

Use different number of nodes (e.g.[128,16]), in hidden layers and compare the results.

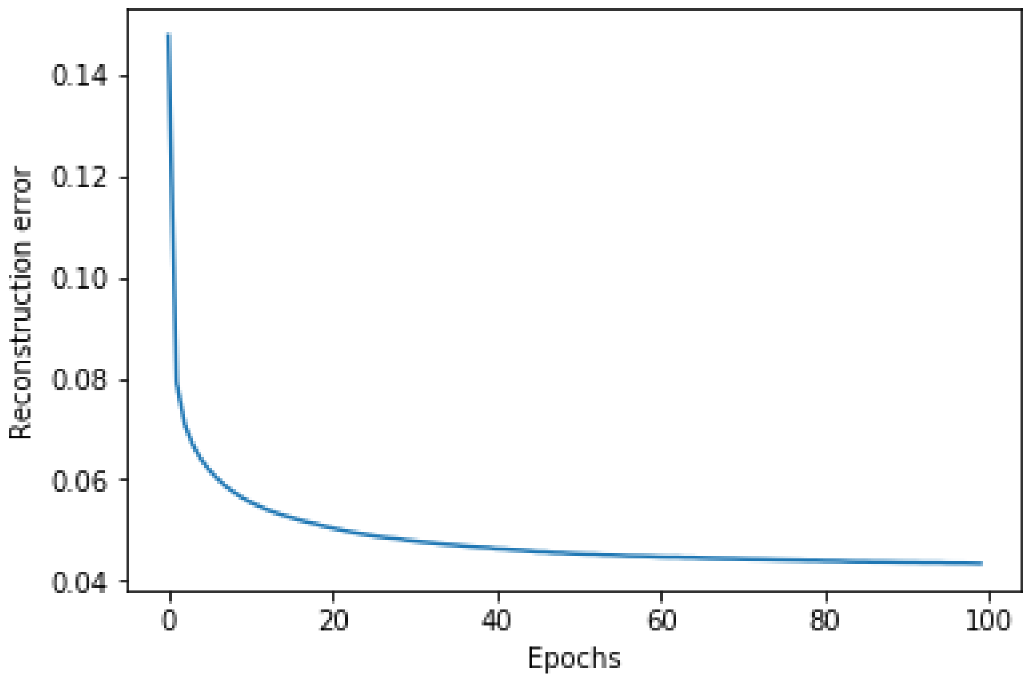
```
In [37]: #define the model
         ####### Question 2 #######
         num_input=28*28
         num_hidden_1=
         num_hidden_2=
         num_output=28*28

         model2 = layer_2_autoencoder(num_input,num_hidden_1,num_hidden_2,num_output)

         #train the model
         model2_loss_train = train(train_set= train_set, batch_size = 128, num_epochs = 100, lea

         # Plot the error
         fig = plt.figure()
         plt.plot(model2_loss_train)
         plt.xlabel('Epochs')
         plt.ylabel('Reconstruction error')
         plt.show()

         #calculate the loss in test set
```

```
        model2_loss_test= evaluate_test(test_set = test_set, test_set_loader=  test_set_loader,

        ####### End of Question 2 #######
```

```
====> Epoch: 0 Average loss: 0.1477
====> Epoch: 1 Average loss: 0.0792
====> Epoch: 2 Average loss: 0.0709
====> Epoch: 3 Average loss: 0.0668
====> Epoch: 4 Average loss: 0.0640
====> Epoch: 5 Average loss: 0.0619
====> Epoch: 6 Average loss: 0.0602
====> Epoch: 7 Average loss: 0.0587
====> Epoch: 8 Average loss: 0.0575
====> Epoch: 9 Average loss: 0.0564
====> Epoch: 10 Average loss: 0.0555
====> Epoch: 11 Average loss: 0.0548
====> Epoch: 12 Average loss: 0.0541
====> Epoch: 13 Average loss: 0.0535
====> Epoch: 14 Average loss: 0.0529
====> Epoch: 15 Average loss: 0.0525
====> Epoch: 16 Average loss: 0.0520
====> Epoch: 17 Average loss: 0.0516
====> Epoch: 18 Average loss: 0.0512
====> Epoch: 19 Average loss: 0.0508
====> Epoch: 20 Average loss: 0.0504
====> Epoch: 21 Average loss: 0.0501
====> Epoch: 22 Average loss: 0.0497
====> Epoch: 23 Average loss: 0.0494
====> Epoch: 24 Average loss: 0.0491
====> Epoch: 25 Average loss: 0.0489
====> Epoch: 26 Average loss: 0.0486
====> Epoch: 27 Average loss: 0.0484
====> Epoch: 28 Average loss: 0.0483
====> Epoch: 29 Average loss: 0.0480
====> Epoch: 30 Average loss: 0.0478
====> Epoch: 31 Average loss: 0.0476
====> Epoch: 32 Average loss: 0.0475
====> Epoch: 33 Average loss: 0.0473
====> Epoch: 34 Average loss: 0.0472
====> Epoch: 35 Average loss: 0.0470
====> Epoch: 36 Average loss: 0.0469
====> Epoch: 37 Average loss: 0.0467
====> Epoch: 38 Average loss: 0.0466
====> Epoch: 39 Average loss: 0.0465
====> Epoch: 40 Average loss: 0.0463
====> Epoch: 41 Average loss: 0.0462
====> Epoch: 42 Average loss: 0.0461
====> Epoch: 43 Average loss: 0.0460
```

```
====> Epoch: 44 Average loss: 0.0459
====> Epoch: 45 Average loss: 0.0457
====> Epoch: 46 Average loss: 0.0457
====> Epoch: 47 Average loss: 0.0455
====> Epoch: 48 Average loss: 0.0455
====> Epoch: 49 Average loss: 0.0454
====> Epoch: 50 Average loss: 0.0453
====> Epoch: 51 Average loss: 0.0452
====> Epoch: 52 Average loss: 0.0452
====> Epoch: 53 Average loss: 0.0451
====> Epoch: 54 Average loss: 0.0451
====> Epoch: 55 Average loss: 0.0450
====> Epoch: 56 Average loss: 0.0449
====> Epoch: 57 Average loss: 0.0448
====> Epoch: 58 Average loss: 0.0448
====> Epoch: 59 Average loss: 0.0448
====> Epoch: 60 Average loss: 0.0447
====> Epoch: 61 Average loss: 0.0447
====> Epoch: 62 Average loss: 0.0447
====> Epoch: 63 Average loss: 0.0446
====> Epoch: 64 Average loss: 0.0446
====> Epoch: 65 Average loss: 0.0445
====> Epoch: 66 Average loss: 0.0444
====> Epoch: 67 Average loss: 0.0444
====> Epoch: 68 Average loss: 0.0444
====> Epoch: 69 Average loss: 0.0443
====> Epoch: 70 Average loss: 0.0443
====> Epoch: 71 Average loss: 0.0442
====> Epoch: 72 Average loss: 0.0442
====> Epoch: 73 Average loss: 0.0442
====> Epoch: 74 Average loss: 0.0441
====> Epoch: 75 Average loss: 0.0441
====> Epoch: 76 Average loss: 0.0441
====> Epoch: 77 Average loss: 0.0441
====> Epoch: 78 Average loss: 0.0440
====> Epoch: 79 Average loss: 0.0440
====> Epoch: 80 Average loss: 0.0439
====> Epoch: 81 Average loss: 0.0439
====> Epoch: 82 Average loss: 0.0439
====> Epoch: 83 Average loss: 0.0438
====> Epoch: 84 Average loss: 0.0438
====> Epoch: 85 Average loss: 0.0438
====> Epoch: 86 Average loss: 0.0437
====> Epoch: 87 Average loss: 0.0437
====> Epoch: 88 Average loss: 0.0437
====> Epoch: 89 Average loss: 0.0436
====> Epoch: 90 Average loss: 0.0436
====> Epoch: 91 Average loss: 0.0436
```

```
====> Epoch: 92 Average loss: 0.0436
====> Epoch: 93 Average loss: 0.0436
====> Epoch: 94 Average loss: 0.0436
====> Epoch: 95 Average loss: 0.0435
====> Epoch: 96 Average loss: 0.0435
====> Epoch: 97 Average loss: 0.0435
====> Epoch: 98 Average loss: 0.0434
====> Epoch: 99 Average loss: 0.0434
```



```
====> Test loss: 0.0435
```

1.3.3 3. 3-layer autoencoder

```
In [38]: #define the model
         num_input=28*28
         num_hidden_1=64
         num_hidden_2=32
         num_hidden_3=16
         num_output=28*28

         model3 = layer_3_autoencoder(num_input,num_hidden_1,num_hidden_2,num_hidden_3,num_outpu

         #train the model
```

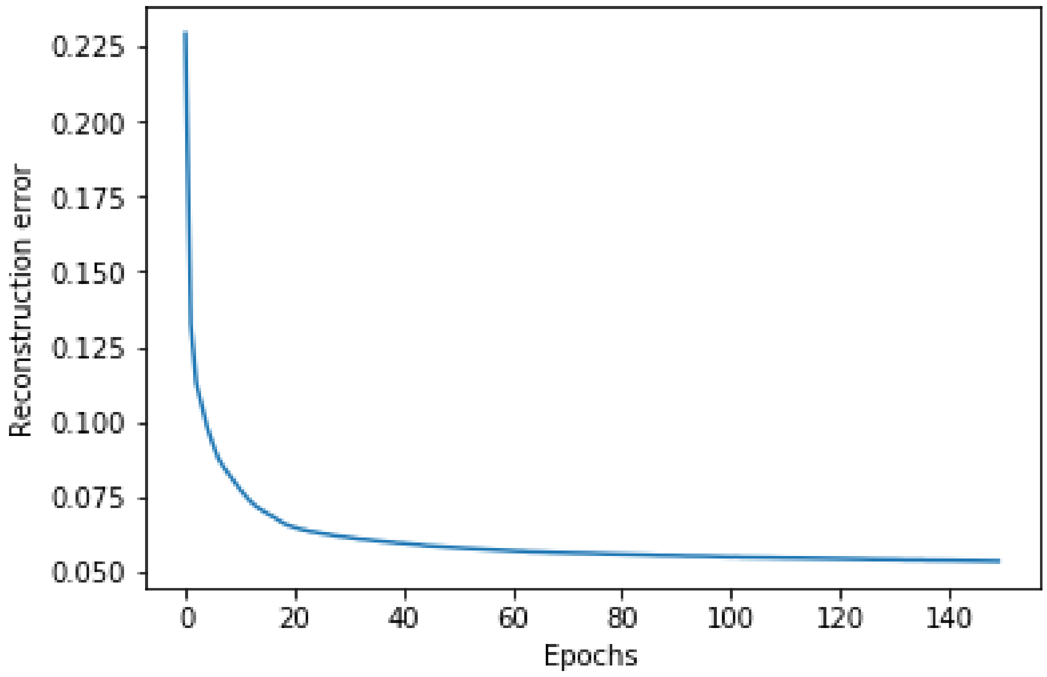
```
    model3_loss_train = train(train_set= train_set, batch_size = 128, num_epochs = 150, lea

    # Plot the error
    fig = plt.figure()
    plt.plot(model3_loss_train)
    plt.xlabel('Epochs')
    plt.ylabel('Reconstruction error')
    plt.show()

    #calculate the loss in test set
    model3_loss_test= evaluate_test(test_set = test_set, test_set_loader=  test_set_loader,
```

```
====> Epoch: 0 Average loss: 0.2293
====> Epoch: 1 Average loss: 0.1324
====> Epoch: 2 Average loss: 0.1128
====> Epoch: 3 Average loss: 0.1056
====> Epoch: 4 Average loss: 0.0982
====> Epoch: 5 Average loss: 0.0928
====> Epoch: 6 Average loss: 0.0879
====> Epoch: 7 Average loss: 0.0849
====> Epoch: 8 Average loss: 0.0824
====> Epoch: 9 Average loss: 0.0798
====> Epoch: 10 Average loss: 0.0775
====> Epoch: 11 Average loss: 0.0754
====> Epoch: 12 Average loss: 0.0734
====> Epoch: 13 Average loss: 0.0718
====> Epoch: 14 Average loss: 0.0705
====> Epoch: 15 Average loss: 0.0694
====> Epoch: 16 Average loss: 0.0684
====> Epoch: 17 Average loss: 0.0672
====> Epoch: 18 Average loss: 0.0661
====> Epoch: 19 Average loss: 0.0653
====> Epoch: 20 Average loss: 0.0648
====> Epoch: 21 Average loss: 0.0642
====> Epoch: 22 Average loss: 0.0638
====> Epoch: 23 Average loss: 0.0634
====> Epoch: 24 Average loss: 0.0631
====> Epoch: 25 Average loss: 0.0627
====> Epoch: 26 Average loss: 0.0625
====> Epoch: 27 Average loss: 0.0621
====> Epoch: 28 Average loss: 0.0619
====> Epoch: 29 Average loss: 0.0617
====> Epoch: 30 Average loss: 0.0614
====> Epoch: 31 Average loss: 0.0611
====> Epoch: 32 Average loss: 0.0609
====> Epoch: 33 Average loss: 0.0607
====> Epoch: 34 Average loss: 0.0605
====> Epoch: 35 Average loss: 0.0603
```

```
====> Epoch: 36 Average loss: 0.0601
====> Epoch: 37 Average loss: 0.0599
====> Epoch: 38 Average loss: 0.0598
====> Epoch: 39 Average loss: 0.0595
====> Epoch: 40 Average loss: 0.0595
====> Epoch: 41 Average loss: 0.0593
====> Epoch: 42 Average loss: 0.0592
====> Epoch: 43 Average loss: 0.0590
====> Epoch: 44 Average loss: 0.0587
====> Epoch: 45 Average loss: 0.0586
====> Epoch: 46 Average loss: 0.0584
====> Epoch: 47 Average loss: 0.0583
====> Epoch: 48 Average loss: 0.0582
====> Epoch: 49 Average loss: 0.0580
====> Epoch: 50 Average loss: 0.0579
====> Epoch: 51 Average loss: 0.0578
====> Epoch: 52 Average loss: 0.0577
====> Epoch: 53 Average loss: 0.0575
====> Epoch: 54 Average loss: 0.0574
====> Epoch: 55 Average loss: 0.0574
====> Epoch: 56 Average loss: 0.0572
====> Epoch: 57 Average loss: 0.0571
====> Epoch: 58 Average loss: 0.0570
====> Epoch: 59 Average loss: 0.0570
====> Epoch: 60 Average loss: 0.0569
====> Epoch: 61 Average loss: 0.0568
====> Epoch: 62 Average loss: 0.0568
====> Epoch: 63 Average loss: 0.0566
====> Epoch: 64 Average loss: 0.0566
====> Epoch: 65 Average loss: 0.0566
====> Epoch: 66 Average loss: 0.0565
====> Epoch: 67 Average loss: 0.0564
====> Epoch: 68 Average loss: 0.0564
====> Epoch: 69 Average loss: 0.0563
====> Epoch: 70 Average loss: 0.0563
====> Epoch: 71 Average loss: 0.0562
====> Epoch: 72 Average loss: 0.0562
====> Epoch: 73 Average loss: 0.0562
====> Epoch: 74 Average loss: 0.0561
====> Epoch: 75 Average loss: 0.0560
====> Epoch: 76 Average loss: 0.0560
====> Epoch: 77 Average loss: 0.0560
====> Epoch: 78 Average loss: 0.0559
====> Epoch: 79 Average loss: 0.0558
====> Epoch: 80 Average loss: 0.0559
====> Epoch: 81 Average loss: 0.0558
====> Epoch: 82 Average loss: 0.0557
====> Epoch: 83 Average loss: 0.0557
```

```
====> Epoch: 84 Average loss: 0.0556
====> Epoch: 85 Average loss: 0.0556
====> Epoch: 86 Average loss: 0.0555
====> Epoch: 87 Average loss: 0.0554
====> Epoch: 88 Average loss: 0.0554
====> Epoch: 89 Average loss: 0.0554
====> Epoch: 90 Average loss: 0.0554
====> Epoch: 91 Average loss: 0.0553
====> Epoch: 92 Average loss: 0.0552
====> Epoch: 93 Average loss: 0.0552
====> Epoch: 94 Average loss: 0.0551
====> Epoch: 95 Average loss: 0.0552
====> Epoch: 96 Average loss: 0.0551
====> Epoch: 97 Average loss: 0.0550
====> Epoch: 98 Average loss: 0.0550
====> Epoch: 99 Average loss: 0.0550
====> Epoch: 100 Average loss: 0.0549
====> Epoch: 101 Average loss: 0.0549
====> Epoch: 102 Average loss: 0.0548
====> Epoch: 103 Average loss: 0.0548
====> Epoch: 104 Average loss: 0.0548
====> Epoch: 105 Average loss: 0.0548
====> Epoch: 106 Average loss: 0.0547
====> Epoch: 107 Average loss: 0.0547
====> Epoch: 108 Average loss: 0.0547
====> Epoch: 109 Average loss: 0.0547
====> Epoch: 110 Average loss: 0.0546
====> Epoch: 111 Average loss: 0.0546
====> Epoch: 112 Average loss: 0.0546
====> Epoch: 113 Average loss: 0.0545
====> Epoch: 114 Average loss: 0.0545
====> Epoch: 115 Average loss: 0.0545
====> Epoch: 116 Average loss: 0.0545
====> Epoch: 117 Average loss: 0.0545
====> Epoch: 118 Average loss: 0.0544
====> Epoch: 119 Average loss: 0.0544
====> Epoch: 120 Average loss: 0.0544
====> Epoch: 121 Average loss: 0.0543
====> Epoch: 122 Average loss: 0.0543
====> Epoch: 123 Average loss: 0.0542
====> Epoch: 124 Average loss: 0.0542
====> Epoch: 125 Average loss: 0.0541
====> Epoch: 126 Average loss: 0.0541
====> Epoch: 127 Average loss: 0.0541
====> Epoch: 128 Average loss: 0.0540
====> Epoch: 129 Average loss: 0.0540
====> Epoch: 130 Average loss: 0.0541
====> Epoch: 131 Average loss: 0.0540
```

```
====> Epoch: 132 Average loss: 0.0539
====> Epoch: 133 Average loss: 0.0539
====> Epoch: 134 Average loss: 0.0539
====> Epoch: 135 Average loss: 0.0539
====> Epoch: 136 Average loss: 0.0538
====> Epoch: 137 Average loss: 0.0538
====> Epoch: 138 Average loss: 0.0539
====> Epoch: 139 Average loss: 0.0538
====> Epoch: 140 Average loss: 0.0538
====> Epoch: 141 Average loss: 0.0537
====> Epoch: 142 Average loss: 0.0537
====> Epoch: 143 Average loss: 0.0537
====> Epoch: 144 Average loss: 0.0537
====> Epoch: 145 Average loss: 0.0536
====> Epoch: 146 Average loss: 0.0536
====> Epoch: 147 Average loss: 0.0536
====> Epoch: 148 Average loss: 0.0536
====> Epoch: 149 Average loss: 0.0535
```



```
====> Test loss: 0.0529
```

```
In [ ]: ####### Question 3 #######
        #define the model
```

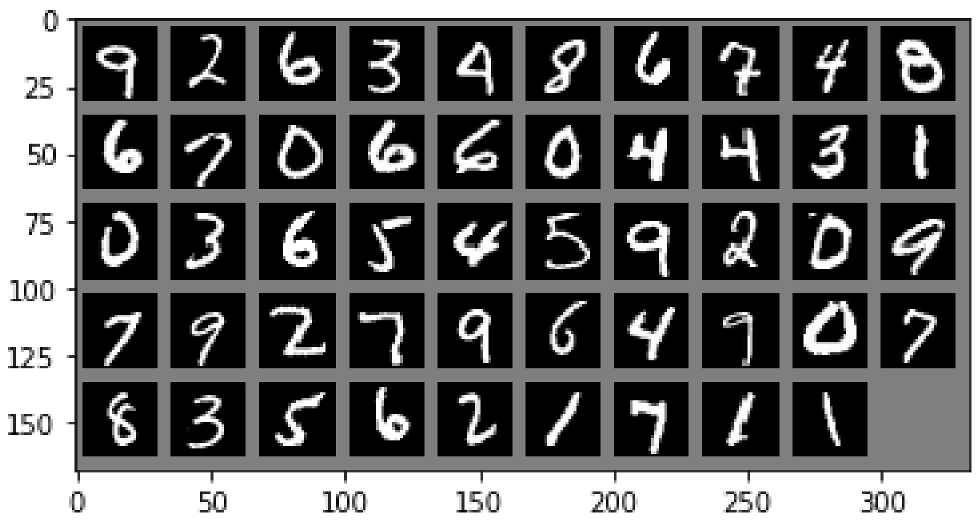
```
num_input=28*28
num_hidden_1=
num_hidden_2=
num_hidden_3=
num_output=28*28

model3 = layer_3_autoencoder(num_input,num_hidden_1,num_hidden_2,num_hidden_3,num_output

#train the model
model3_loss_train = train(train_set= train_set, batch_size = 128, num_epochs = 150, lear

# Plot the error
fig = plt.figure()
plt.plot(model3_loss_train)
plt.xlabel('Epochs')
plt.ylabel('Reconstruction error')
plt.show()

#calculate the loss in test set
model3_loss_test= evaluate_test(test_set = test_set, test_set_loader=  test_set_loader,b
####### End of Question 3 #######
```

1.3.4 Visualise the reconstructions obtained from the three models

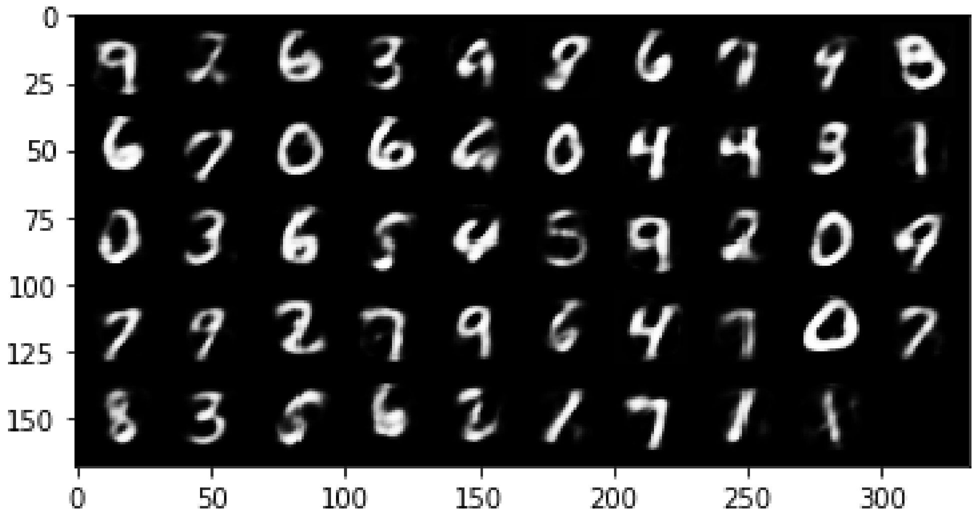
```
In [40]: data_iterator = iter(test_set_loader)
         images, labels = data_iterator.next()

         # This function takes as an input the images to reconstruct
         # and the name of the model with which the reconstructions
         # are performed
         def to_img(x):
             x = 0.5 * (x + 1)
             x = x.clamp(0, 1)
             x = x.view(x.size(0), 1, 28, 28)
             return x

         def visualise_output(images, model):
             data =images
             img= data
             img = img.view(img.size(0), -1)
             img = Variable(img)
             out = model(img)
             pic = out.data.view(out.data.size(0), 1, 28, 28)
             pic = pic * 0.5 + 0.5
             np_pic = torchvision.utils.make_grid(pic[1:50], 10, 5).numpy()
             plt.imshow(np.transpose(np_pic, (1, 2, 0)))
             plt.show()
```

```
In [41]: # First visualise the original images
         print('Original images')
         show_image(torchvision.utils.make_grid(images[1:50],10,5))
         plt.show()

         # Reconstruct and visualise the images using the first model
         print('PCA based autoencoder reconstruction:')
         visualise_output(images, model1)

         # Reconstruct and visualise the images using the second model
         print('2-layer autoencoder reconstruction:')
         visualise_output(images, model2)

         # Reconstruct and visualise the images using the third model
         print('3-layer autoencoder reconstruction:')
         visualise_output(images, model3)
```

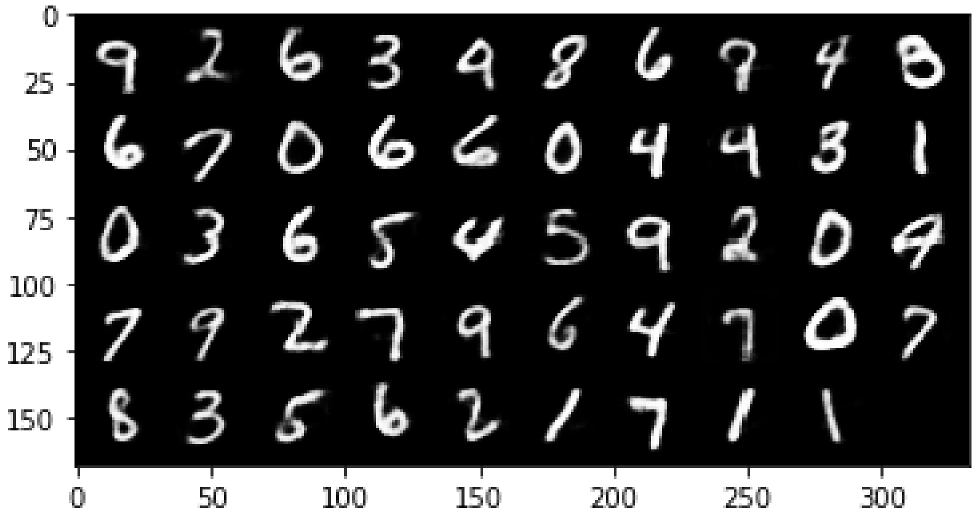
```
Original images
```



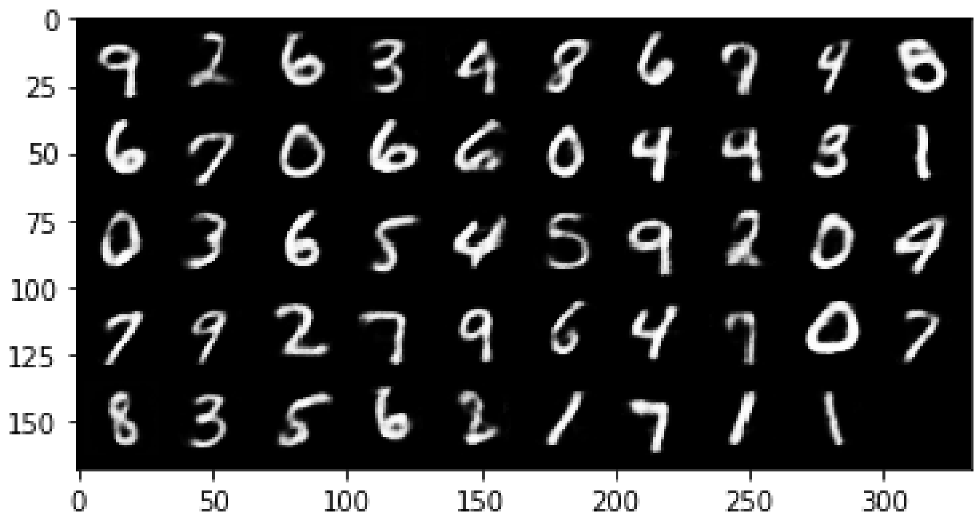
```
PCA based autoencoder reconstruction:
```



```
2-layer autoencoder reconstruction:
```



```
3-layer autoencoder reconstruction:
```



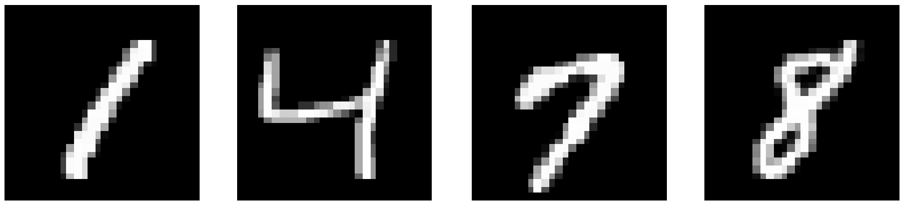
1.4 Part B

In this part of the assignment we need to interpolate between two images using the latent space representation obtained by the autoencoder-based PCA and the 3-layer autoencoder and subsequently to verify that the interpolation generated using the 3-layer autoencoder is better.

```
In [43]: #Extract images to apply interpolation
         one = []
         four = []
         seven = []
         eight = []

         for i in range(len(train_set)):
             if (train_set[i][1]==1):
                 one.append(train_set[i])

             if (train_set[i][1]==4):
                 four.append(train_set[i])

             if (train_set[i][1]==7):
                 seven.append(train_set[i])

             if (train_set[i][1]==8):
                 eight.append(train_set[i])

         img1= seven[0][0]
         npimg = img1.numpy()
         im7 = npimg.reshape(28, 28)
```

```
        img2 = one[0][0]
        npimg = img2.numpy()
        im1 = npimg.reshape(28, 28)

        img3 = four[0][0]
        npimg = img3.numpy()
        im4 = npimg.reshape(28, 28)

        img4 = eight[0][0]
        npimg = img4.numpy()
        im8 = npimg.reshape(28, 28)
```

```
In [44]: f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4)
         ax1.imshow(im1, cmap='gray')
         ax1.axis('off')
         ax2.imshow(im4, cmap='gray')
         ax2.axis('off')
         ax3.imshow(im7, cmap='gray')
         ax3.axis('off')
         ax4.imshow(im8, cmap='gray')
         ax4.axis('off')
         plt.show()
```



```
In [45]: sevens = torch.Tensor(seven[0][0])
         ones = torch.Tensor(one[0][0])
         fours = torch.Tensor(four[0][0])
         eights = torch.Tensor(eight[0][0])
```

Fill interpolation() function in orde to 1.1. Get the latent space representation. 1.2. Interpolate between two images using the following equation

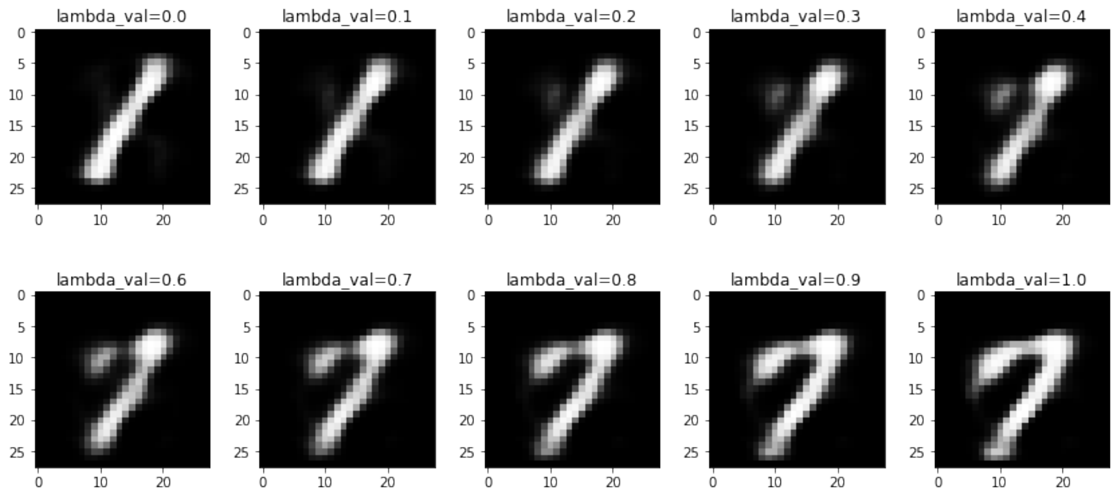
$$interpolated\_image\_\lambda = decoder[\lambda * latent\_1 + (1-\lambda) * latent\_2]. \tag{4}$$

1.3. Get the image space represantion.

```
In [48]: def interpolation(lambda1, model, image1, image2):

             img1= image1
             img1 = img1.view(img1.size(0), -1)
             img1 = Variable(img1)
             ############ Question 1 ############
             #Latent space interpretation of image1
             latent_1 = model.encoder(img1)

             img2= image2
             img2 = img2.view(img1.size(0), -1)
             img2 = Variable(img2)
             #Latent space interpretation of image2
             latent_2 =

             #Interpolation of the two images
             inter_image_lat =


             inter_image =


             return inter_image
             ############ END of question 1 ############
```

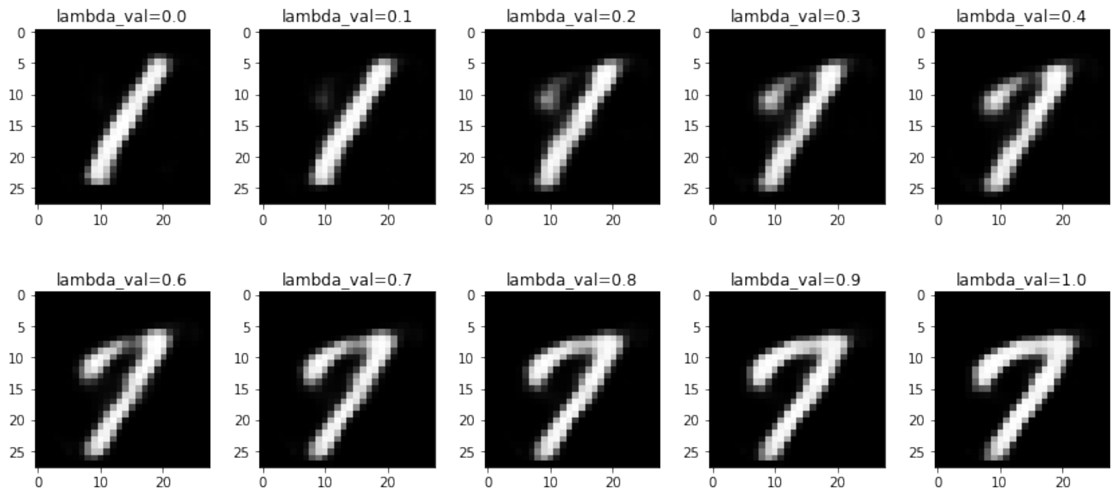
Interpolation between digits *seven* and *one* using the autoencoder-based PCA.

```
In [50]: ############ Question 2 ############

         lambda_range=np.linspace(0,1,10)


         fig, axs = plt.subplots(2,5, figsize=(15, 6))
         fig.subplots_adjust(hspace = .5, wspace=.001)

         axs = axs.ravel()

         for ind,l in enumerate(lambda_range):
             image_lambda=

             pic = to_img(image_lambda)
             image = pic.data.numpy()
             image = image.reshape(28,28)

             axs[ind].imshow(image, cmap='gray')
             axs[ind].set_title('lambda_val='+str(round(l,1)))
         plt.show()
```



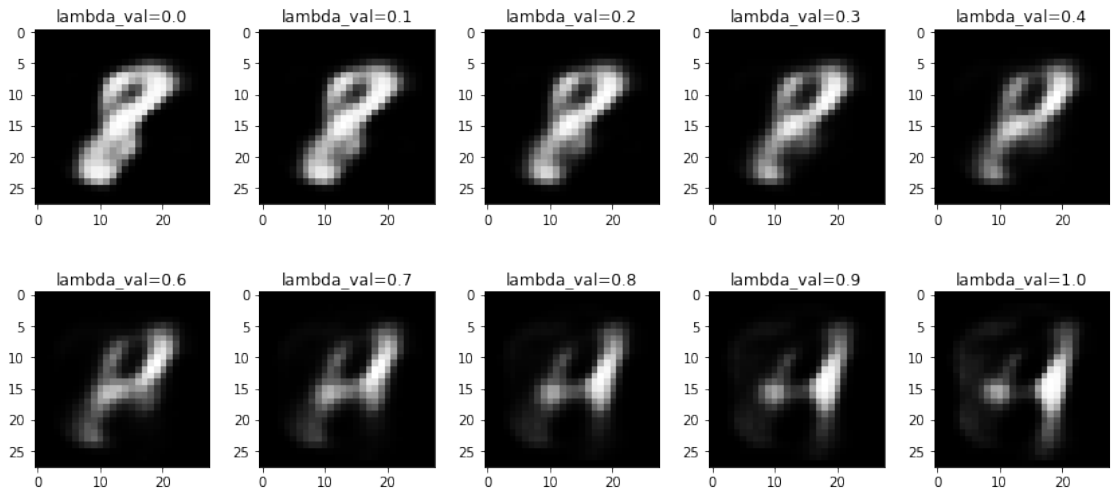
Interpolation between digits *seven* and *one* using the 3-layer autoencoder.

```
In [51]: ############ Question 2 ############

         lambda_range=np.linspace(0,1,10)


         fig, axs = plt.subplots(2,5, figsize=(15, 6))
         fig.subplots_adjust(hspace = .5, wspace=.001)

         axs = axs.ravel()

         for ind,l in enumerate(lambda_range):
             image_lambda=

             pic = to_img(image_lambda)
             image = pic.data.numpy()
             image = image.reshape(28,28)

             axs[ind].imshow(image, cmap='gray')
             axs[ind].set_title('lambda_val='+str(round(l,1)))
         plt.show()
```



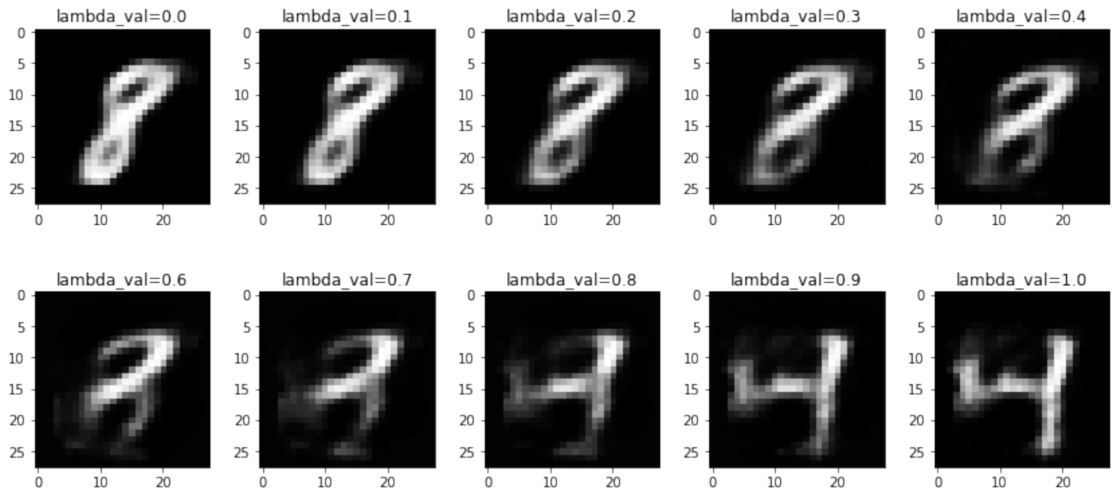
Interpolation between digits *four* and *eight* using the autoencoder-based PCA.

```
In [52]: ############ Question 2 ############

         lambda_range=np.linspace(0,1,10)


         fig, axs = plt.subplots(2,5, figsize=(15, 6))
         fig.subplots_adjust(hspace = .5, wspace=.001)

         axs = axs.ravel()

         for ind,l in enumerate(lambda_range):
             image_lambda=

             pic = to_img(image_lambda)
             image = pic.data.numpy()
             image = image.reshape(28,28)

             axs[ind].imshow(image, cmap='gray')
             axs[ind].set_title('lambda_val='+str(round(l,1)))
         plt.show()
```



Interpolation between digits *four* and *eight* using the 3-layer autoencoder.

```
In [53]: ############ Question 2 ############

         lambda_range=np.linspace(0,1,10)


         fig, axs = plt.subplots(2,5, figsize=(15, 6))
         fig.subplots_adjust(hspace = .5, wspace=.001)

         axs = axs.ravel()

         for ind,l in enumerate(lambda_range):
             image_lambda=

             pic = to_img(image_lambda)
             image = pic.data.numpy()
             image = image.reshape(28,28)

             axs[ind].imshow(image, cmap='gray')
             axs[ind].set_title('lambda_val='+str(round(l,1)))
         plt.show()
```


lambda_val=0.0
lambda_val=0.1
lambda_val=0.2
lambda_val=0.3
lambda_val=0.4
lambda_val=0.6
lambda_val=0.7
lambda_val=0.8
lambda_val=0.9
lambda_val=1.0