# Homework 4 Part 2 - Solutions

**In order to solve this assignment, you must work with a TensorFlow kernel in HiPerGator. If you are working locally, you must [install TensorFlow 2 (https://www.tensorflow.org/install)](https://www.tensorflow.org/install).**

# Problem 1 (25 points)

In this problem you will be experimenting with a special MLP architecture, known as *autoencoder* (or AE).

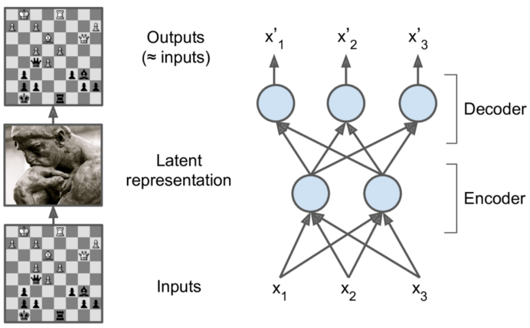
An autoencoder attempts to find efficient latent representations of the inputs, it then spits out something that (hopefully) looks very close to the inputs. An autoencoder is always composed of two parts:

1. an encoder (or recognition network) that converts the inputs to a latent representation, followed by
2. a decoder (or generative network) that converts the internal representation to the outputs.

In [1]:

```
from IPython.display import Image
Image('figures/autoencoder.png', width=400)
```

Out[1]:




As you can see, an autoencoder typically has the same architecture as a Multi-Layer Perceptron (MLP), except that the number of neurons in the output layer must be equal to the number of inputs. In this example, there is just one hidden layer composed of two neurons (the encoder), and one output layer composed of three neurons (the decoder). The outputs are often called the reconstructions because the autoencoder tries to reconstruct the inputs, and the cost function contains a reconstruction loss that penalizes the model when the reconstructions are different from the inputs.

Because the internal representation has a lower dimensionality than the input data (it is 2D instead of 3D), the autoencoder is said to be undercomplete. An undercomplete autoencoder cannot trivially copy its inputs to the codings, yet it must find a way to output a copy of its inputs. It is forced to learn the most important features in the input data (and drop the unimportant ones).

Just like other neural networks we have discussed, autoencoders can have multiple hidden layers. In this case they are called stacked autoencoders (or deep autoencoders). Adding more layers helps the autoencoder learn more complex codings.

The architecture of a stacked autoencoder is typically symmetrical with regard to the central hidden layer (the coding layer). For example, an autoencoder for the MNIST may have 784 inputs, followed by a hidden layer with 100 neurons, then a central hidden layer of 30 neurons, then another hidden layer with 100 neurons, and an output layer with 784 neurons:

In [29]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
import tensorflow as tf
from tensorflow import keras
from time import time
import warnings
warnings.filterwarnings('ignore')
```

In [22]:

```
mnist = keras.datasets.mnist

(X_train_full, t_train_full), (X_test, t_test) = mnist.load_data()

X_train_full.shape, t_train_full.shape, X_test.shape, t_test.shape
```

Out[22]:

```
((60000, 28, 28), (60000,), (10000, 28, 28), (10000,))
```

In [23]:

```
# Training and Validation sets
# First 5,000 samples as validation and the remaining ones as training samples
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
t_valid, t_train = t_train_full[:5000], t_train_full[5000:]
X_test = X_test / 255.0
```

In [24]:

```
plt.figure(figsize=(10,5))
plot_idx=1
for i in range(10):
    labels = np.where(t_train==i)[0]
    idx = np.random.permutation(range(len(labels)))
    for j in range(1,16):
        plt.subplot(10,15,plot_idx)
        plt.imshow(X_train[labels[j],:,:], cmap='binary')
        plt.axis('off')
        plot_idx+=1
```

In [25]:

```
# Reproducible results - fix the random seed generator (doesn't account for GPU-induced randomness)
tf.random.set_seed(2)

# Stacked Auto-Encoder
def stacked_autoencoder(X_train, X_valid, embedding_size=30, input_shape=[28,28], epochs=10):
    stacked_encoder = keras.models.Sequential([
        keras.layers.Flatten(input_shape=input_shape),
        keras.layers.Dense(100, activation='relu'),
        keras.layers.Dense(embedding_size, activation='relu')])

    stacked_decoder = keras.models.Sequential([
        keras.layers.Dense(100, activation='relu', input_shape=[embedding_size]),
        keras.layers.Dense(input_shape[0] * input_shape[1], activation='sigmoid'),
        keras.layers.Reshape(input_shape)])

    stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])

    stacked_ae.compile(loss=keras.losses.BinaryCrossentropy(),
                       optimizer=keras.optimizers.Adam(),
                      )

    start = time()
    history = stacked_ae.fit(X_train, X_train, epochs=epochs, batch_size=32,
                             validation_data=[X_valid, X_valid])
    print('Elapsed Time: ',time()-start, ' seconds')

    return stacked_ae, history


# Embedding dimensionality
embedding_size = 30

# Training Stacked Autoencoder
stacked_ae, history = stacked_autoencoder(X_train, X_valid, embedding_size)
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1450 - val_loss: 0.1068
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1023 - val_loss: 0.0976
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0961 - val_loss: 0.0937
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0928 - val_loss: 0.0912
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0909 - val_loss: 0.0901
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0896 - val_loss: 0.0890
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0887 - val_loss: 0.0885
Epoch 8/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0880 - val_loss: 0.0879
Epoch 9/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0874 - val_loss: 0.0876
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0869 - val_loss: 0.0866
Elapsed Time:  52.06462097167969  seconds
```

When compiling the stacked autoencoder, we use the binary cross-entropy loss instead of the mean squared error. We are treating the reconstruction task as a multilabel binary classification problem: each pixel intensity represents the probability that the pixel should be black. Framing it this way (rather than as a regression problem) tends to make the model converge faster.

Let's visualize some example reconstructions:

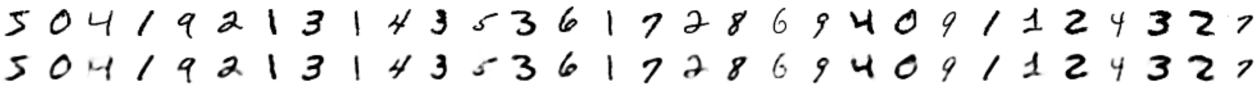
In [26]:

```
def plot_image(image):
    plt.imshow(image, cmap='binary')
    plt.axis('off')

def show_reconstructions(model, X_valid=X_valid, input_shape=[28,28], compute_error=True, n_images=30):
    reconstructions = model.predict(X_valid[:n_images])
    if compute_error:
        error=X_valid[:n_images]-reconstructions
        avg_MSE=np.mean(np.mean(error.reshape((n_images,input_shape[0]*input_shape[1]))**2,axis=1))
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(X_valid[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])
    if compute_error:
        print('Average MSE of reconstruction: ', avg_MSE)
```

In [27]:

```python
# Show Reconstructions

show_reconstructions(stacked_ae)
```

```
Average MSE of reconstruction:  0.008211262876900032
```



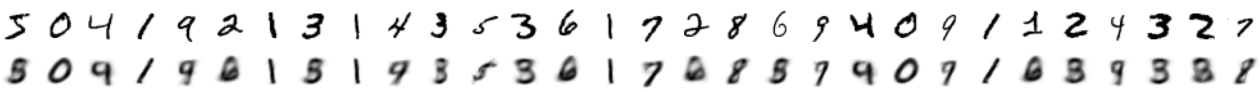
### Accessing Outputs at the Bottleneck Layer

To demonstrate this, let's consider a stacked autoencoder with a 2-dimensional bottleneck layer (embedding dimension):

In [12]:

```python
# Embedding dimensionality
embedding_size = 2

# Training Stacked Autoencoder
stacked_ae = stacked_autoencoder(X_train, X_valid, embedding_size)

# Show Reconstructions
show_reconstructions(stacked_ae)
```

```
Epoch 1/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.2446 - val_loss: 0.2145
Epoch 2/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.2103 - val_loss: 0.2059
Epoch 3/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.2054 - val_loss: 0.2027
Epoch 4/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.2023 - val_loss: 0.1996
Epoch 5/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.1997 - val_loss: 0.1972
Epoch 6/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.1976 - val_loss: 0.1951
Epoch 7/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.1962 - val_loss: 0.1939
Epoch 8/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.1951 - val_loss: 0.1928
Epoch 9/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.1941 - val_loss: 0.1923
Epoch 10/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.1934 - val_loss: 0.1915
Elapsed Time:  29.715887308120728  seconds
1/1 [==============================] - 0s 48ms/step
Average MSE of reconstruction:  0.039719154049662087
```



If you want to access the embedding output produced at the bottleneck layer, you can do the following:

In [13]:

```python
stacked_ae.layers

# The "1st layer" corresponds ot the stacked encoder
# The "2nd layer" corresponds ot the stacked decoder
```

Out[13]:

```
[<keras.engine.sequential.Sequential at 0x1efab29feb0>,
 <keras.engine.sequential.Sequential at 0x1efad3b8250>]
```

In [14]:

```python
enc = stacked_ae.layers[0]
enc.layers

# As you can see, the stacked encoder contains a reshaping layer (flatten), 1st hidden layer and the bottleneck layer.
# Let's obtain the output at each layer and pass it to the next
```

Out[14]:

```
[<keras.layers.reshaping.flatten.Flatten at 0x1efad3b9f70>,
 <keras.layers.core.dense.Dense at 0x1efab9bf9d0>,
 <keras.layers.core.dense.Dense at 0x1efab2bbe20>]
```

In [15]:

```
flatten = enc.layers[0](X_train)
hidden1 = enc.layers[1](flatten)
bottleneck = enc.layers[2](hidden1)
```

In [16]:

```
bottleneck.shape
```

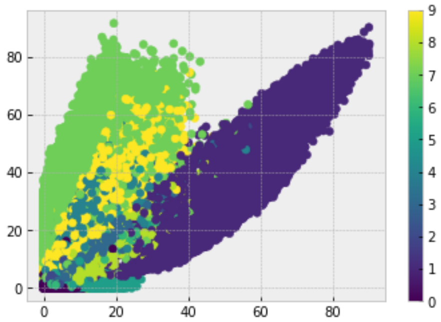
Out[16]:

```
TensorShape([55000, 2])
```

As expected, the bottleneck layer mapped all 55,000 training images to a 2-dimensional space. Since its 2-D, we can visualize it:

In [17]:

```
plt.scatter(bottleneck[:,0], bottleneck[:,1], c=t_train)
plt.colorbar();
```




Answer the following questions:

1. (7 points) Experiment with different embedding dimensions (at least 3 values). At least a "very small" embedding space (like 2), "very large" embedding space (like 90), and another in between. Discuss your observations regarding the speed of training, quality of reconstruction images, and the reconstruction average MSE.

In [ ]:

```

```

In [28]:

```
# Embedding dimensionality
embedding_size = 10

# Training Stacked Autoencoder
stacked_ae = stacked_autoencoder(X_train, X_valid, embedding_size)

# Show Reconstructions
show_reconstructions(stacked_ae)
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1790 - val_loss: 0.1473
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1433 - val_loss: 0.1380
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1368 - val_loss: 0.1337
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1334 - val_loss: 0.1313
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1313 - val_loss: 0.1297
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1299 - val_loss: 0.1280
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1289 - val_loss: 0.1273
Epoch 8/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1281 - val_loss: 0.1266
Epoch 9/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1274 - val_loss: 0.1261
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1268 - val_loss: 0.1256
Elapsed Time:  51.89879751205444  seconds
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-28-df5107599615> in <module>
      6
      7 # Show Reconstructions
----> 8 show_reconstructions(stacked_ae)

<ipython-input-26-1e306f1adc0b> in show_reconstructions(model, X_valid, input_shape, compute_error, n_images)
      4
      5 def show_reconstructions(model, X_valid=X_valid, input_shape=[28,28], compute_error=True, n_images=30):
----> 6     reconstructions = model.predict(X_valid[:n_images])
      7     if compute_error:
      8         error=X_valid[:n_images]-reconstructions

AttributeError: 'tuple' object has no attribute 'predict'
```

In [31]:

```
from time import time


def create_auto_encoder(X, X_valid, X_test, embedding_size, input_shape = (28, 28), tracking_metrics = ['accuracy'], batch_size = 32, epo

    encoder = keras.models.Sequential([
        keras.layers.Flatten(input_shape=input_shape),
        keras.layers.Dense(100, activation='relu'),
        keras.layers.Dense(embedding_size, activation='relu')
    ])

    decoder = keras.models.Sequential([
        keras.layers.Dense(100, input_shape=(embedding_size,), activation='relu'),
        keras.layers.Dense(input_shape[0] * input_shape[1], activation="sigmoid"),
        keras.layers.Reshape(input_shape)
    ])

    ae_model = keras.models.Sequential([
        encoder,
        decoder
    ])

    ae_model.compile(
        loss = keras.losses.BinaryCrossentropy(),
        optimizer = keras.optimizers.Adam(),
        metrics = tracking_metrics
    )

    print(f"Training Auto Encoder on Input [ samples = {X.shape[0]}, size = {input_shape} ] for max epochs [ {epochs} ] on embedding size

    start_time = time()
    history = ae_model.fit(X, X, validation_data = (X_valid, X_valid), batch_size = batch_size, epochs = epochs)

    duration_of_fit = time() - start_time

    X_test_reconstructed = ae_model.predict(X_test)

    avg_mse = calculate_avg_of_mse(X_test, X_test_reconstructed)

    print(f"Duration of fit [{duration_of_fit}] and [ {avg_mse} ] ")

    return duration_of_fit, avg_mse, ae_model, X_test_reconstructed


EMBEDDING_SPACES = [3, 40, 90, 200]

metrics = {
    "Embedding Size": [],
    "Time (s)": [],
    "Average MSE": []
}

n_samples = 25

X_test_reconstructeds = {}

for embedding_size in EMBEDDING_SPACES:

    N_train_samples, height, width = X_train.shape

    duration, avg_mse, ae_model, X_test_reconstructed = create_auto_encoder(X_train, X_valid, X_test, embedding_size, input_shape=(height,

    metrics["Embedding Size"].append(embedding_size)
    metrics['Time (s)'].append(duration)
    metrics['Average MSE'].append(avg_mse)

    X_test_reconstructeds[embedding_size] = X_test_reconstructed

df = pd.DataFrame(data=metrics)
print(df)

plt.figure(figsize=(n_samples * 1.5, 1.5 + (1.5 * len(EMBEDDING_SPACES))))
plt.suptitle(f"Reconstruction for different Embedding sizes [ {EMBEDDING_SPACES} ]", fontsize=15)

plots = 1

for i in range(n_samples):
    plt.subplot(1 + len(EMBEDDING_SPACES), n_samples, plots)
    plt.imshow(X_test[i], cmap='binary')
    plt.axis('off')
    plots += 1

for i, embedding_size in enumerate(EMBEDDING_SPACES):
    for j in range(n_samples):
        plt.subplot(1 + len(EMBEDDING_SPACES), n_samples, plots)
        plt.imshow(X_test_reconstructeds[embedding_size][j], cmap='binary')
        plt.axis('off')
        plots += 1
```

```
plt.show()
racy: 0.1860
Duration of fit [58.35504961013794] and [ 0.03727047906969878 ]
Training Auto Encoder on Input [ samples = 55000, size = (28, 28) ] for max epochs [ 10 ] on embedding size [ 40 ]
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1475 - accuracy: 0.2143 - val_loss: 0.1098 - val_accu
racy: 0.2544
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1021 - accuracy: 0.2634 - val_loss: 0.0966 - val_accu
racy: 0.2735
Epoch 3/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0948 - accuracy: 0.2764 - val_loss: 0.0924 - val_accu
racy: 0.2794
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0913 - accuracy: 0.2832 - val_loss: 0.0899 - val_accu
racy: 0.2871
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0892 - accuracy: 0.2885 - val_loss: 0.0889 - val_accu
racy: 0.2903
Epoch 6/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0878 - accuracy: 0.2940 - val_loss: 0.0872 - val_accu
```

In [ ]:

2. (11 points) Compare the stacked AE reconstructions with those produced with PCA (for the same embedding dimensionality). Discuss your observations based on reconstruction visualization and average MSE.

In [35]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [36]:

```
X_train.shape
```

Out[36]:

```
(55000, 28, 28)
```

In [41]:

```
x,y,z =X_train.shape
x,y,z
```

Out[41]:

```
(55000, 28, 28)
```

In [43]:

```
xv, yv, zv = X_valid.shape
xv,yv,zv
```

Out[43]:

```
(5000, 28, 28)
```

In [44]:

```
X_train_pc = X_train.reshape(x,y*z)
X_valid_pc = X_valid.reshape(xv,yv*zv)
```

In [47]:

```
X_train_pc.shape
```

Out[47]:

```
(55000, 784)
```

In [ ]:

In [48]:

```
scaler = StandardScaler()
X_train_pc_sc = scaler.fit_transform(X_train.pc)
X_valid_pc_sc = scaler.transform(X_valid_pc)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-48-26aba01455c6> in <module>
      1 scaler = StandardScaler()
----> 2 X_train_pc_sc = scaler.fit_transform(X_train.pc)
      3 X_valid_pc_sc = scaler.transform(X_valid_pc)

AttributeError: 'numpy.ndarray' object has no attribute 'pc'
```

In [49]:

```
def create_and_train_pca(t_train, t_valid, t_test, embedding_size):

    # Find dimensions of the figures
    num_samples, height, width = t_train.shape

    # Flatten the figures
    t_flattened = t_train.reshape((num_samples, height * width))
    t_test_flattend = t_test.reshape((t_test.shape[0], height * width))

    # Create a standard scaler
    scalar = StandardScaler()
    t_scaled = scalar.fit_transform(t_flattened)

    # Create PCA Object
    pca = PCA(num_components=embedding_size)
    t_embedded = pca.fit_transform(t_scaled)

    # Recreate Training Data
    t_train_reconstructed_flattened = scalar.inverse_transform(
        pca.inverse_transform(t_embedded)
    )

    t_train_recnstd = t_train_reconstructed_flattened.reshape((num_samples, height, width))

    avg_MSE_train = calculate_avg_of_mse(t_train, t_train_reconstructed)

    t_test_scaled = scalar.transform(t_test_flattend)
    t_test_embedded = pca.transform(t_test_scaled)

    t_test_reconstructed_flattened = scalar.inverse_transform(
        pca.inverse_transform(t_test_embedded)
    )

    t_test_reconstructed = t_test_reconstructed_flattened.reshape((t_test.shape[0], height, width))

    avg_MSE_test = calculate_avg_of_mse(t_test, t_test_reconstructed)

    return avg_MSE_test, t_test_reconstructed, avg_MSE_train, t_train_reconstructed
```

In [50]:

```
## All possible embedding sizes to be tested with Stacked AE and PCA
EMBEDDING_SPACES = [3, 40, 90, 200, 400]

## Collect metrics
metrics = {
    "Embedding Size": [],
    "Avrge MSE (Stacked AE)": [],
    "Avrge MSE (PCA)": []
}

## Number of pictures to plot
Num_representation_samples = 25

(num_train_samples, height, width) = t_train.shape

ae_reconstructeds = {}
pca_reconstructeds = {}

for embedding_size in EMBEDDING_SPACES:

    duration_stacked_ae, avg_mse_stacked_ae, ae_model, t_test_reconstructed_stacked_ae = create_auto_encoder(t_train, t_valid, t_test, emb

    avg_MSE_test_pca, t_test_reconstructed_pca, avg_MSE_train_pca, t_train_reconstructed_pca = create_and_train_pca(t_train, t_valid, t_te

    metrics["Embedding Size"].append(embedding_size)
    metrics["Avrge MSE (Stacked AE)"].append(avg_mse_stacked_ae)
    metrics["Avrge MSE (PCA)"].append(avg_MSE_test_pca)

    ae_reconstructeds[embedding_size] = t_test_reconstructed_stacked_ae
    pca_reconstructeds[embedding_size] = t_test_reconstructed_pca


for embedding_size in EMBEDDING_SPACES:

    plt.figure(figsize=(Num_representation_samples * 1.5, 3 * 1.5))
    plt.suptitle(f"Original vs Stacked AE vs PCA for Embedding Size [ {embedding_size} ]", fontsize = 16)

    plots = 1

    for i in range(Num_representation_samples):
        plt.subplot(3, Num_representation_samples, plots)
        plt.imshow(X_test[i], cmap='binary')
        plt.axis('off')
        plots += 1

    for i in range(Num_representation_samples):
        plt.subplot(3, Num_representation_samples, plots)
        plt.imshow(ae_reconstructeds[embedding_size][i], cmap='binary')
        plt.axis('off')
        plots += 1

    for i in range(Num_representation_samples):
        plt.subplot(3, Num_representation_samples, plots)
        plt.imshow(pca_reconstructeds[embedding_size][i], cmap='binary')
        plt.axis('off')
        plots += 1

    plt.show()

df = pd.DataFrame(data=metrics)
print(df)
```

```
Training Auto Encoder on Input [ samples = 55000, size = (28, 28) ] for max epochs [ 10 ] on embedding size [ 3 ]
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2187 - accuracy: 0.1262 - val_loss: 0.1943 - val_accu
racy: 0.1397
Epoch 2/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1915 - accuracy: 0.1485 - val_loss: 0.1860 - val_accu
racy: 0.1570
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1851 - accuracy: 0.1636 - val_loss: 0.1808 - val_accu
racy: 0.1733
Epoch 4/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1811 - accuracy: 0.1736 - val_loss: 0.1776 - val_accu
racy: 0.1804
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1786 - accuracy: 0.1791 - val_loss: 0.1755 - val_accu
racy: 0.1820
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1766 - accuracy: 0.1818 - val_loss: 0.1737 - val_accu
racy: 0.1884
```

In [ ]:

3. (7 points) For what autoencoder design (architecture, activation functions and objective function), will an autoencoder be producing the same results as PCA? Justify your answer.

The highest variance is retained throughout a linear transformation by PCA. The eigen vectors will be organized in decreasing order of eigen values to produce the linear transformer. Therefore, the hidden layer of an auto encoder that mimics PCA must undergo the same linear transformation. The number of units in the output layer will match those in the input layer.

Architecture There will be one hidden layer to resemble a PCA. The number of units (neurons) in the hidden layer will be equal to the anticipated number of PCA components. The input layer and output layer will both have the same number of units (For reconstruction).

Function of Activation The activation function will be linear at all layers since PCA performs linear transformation using a linear transformation matrix with eigen vectors ordered in descending order of eigen values. Additionally, the PCA's linear transformation matrix will be reflected in the wrights from the input layer to the hidden layer. The weights from the hidden layer to the output layer will be the inverse of the linear transformation matrix related to PCA (will be similar to, but not necessarily precise).

Objective Purpose The auto encoder could be trained using an objecting function that penalizes divergence from actual value in either direction because it is anticipated that the reconstruction of the AE will provide the same result as that of the input. It is recommended to utilize either mean squared error or mean absolute error.

In [ ]:

---

# Problem 2 (7.5 points)

For this problem, consider the final project training data. Feel free to discuss with your team, but this is an individual assignment.

In [18]:

```
X_train = np.load('data_train.npy').T
t_train = np.load('t_train.npy') # or np.load('t_train_corrected.npy')

X_train.shape, t_train.shape
```

Out[18]:

```
((9032, 90000), (9032,))
```

You can convert `numpy` arrays to `tensorflow` tensors with:

In [19]:

```
X_train_tf = tf.constant(X_train.reshape(X_train.shape[0], 300, 300))

X_train_tf.shape
```

Out[19]:

```
TensorShape([9032, 300, 300])
```

Answer the following questions:

1. (1 point) Split your data into training and validation sets. Use a stratified 80/20 partition with a fixed `random_state` (in order to avoid data leakage).

In [ ]:

```
RANDOM_State= 42

X_train_set, X_valid_set, T_train_set, T_valid_set = train_test_split(X_train, t_train, test_size=0.2, stratify=t_train ,random_state=RAND(
```

In [ ]:

2. (3.5 points) Train a stacked autoencoder with an embedding dimension of at least 100-dimensional.

In [134]:

```
# Embedding dimensionality
embedding_size = 100

# Training Stacked Autoencoder
stacked_ae = stacked_autoencoder(X_train, X_valid, embedding_size)

# Show Reconstructions
show_reconstructions(stacked_ae)
```

```
Epoch 1/10
1719/1719 [==============================] - 7s 4ms/step - loss: 0.1362 - val_loss: 0.0978
Epoch 2/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0920 - val_loss: 0.0875
Epoch 3/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0852 - val_loss: 0.0830
Epoch 4/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0818 - val_loss: 0.0805
Epoch 5/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0796 - val_loss: 0.0792
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0780 - val_loss: 0.0779
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0769 - val_loss: 0.0765
Epoch 8/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0759 - val_loss: 0.0761
Epoch 9/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0752 - val_loss: 0.0753
Epoch 10/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0747 - val_loss: 0.0746
Elapsed Time:  59.37343430519104  seconds

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-134-42959faa56c5> in <module>
      6 
      7 # Show Reconstructions
----> 8 show_reconstructions(stacked_ae)

<ipython-input-26-1e306f1adc0b> in show_reconstructions(model, X_valid, input_shape, compute_error, n_images)
      4 
      5 def show_reconstructions(model, X_valid=X_valid, input_shape=[28,28], compute_error=True, n_images=30):
----> 6     reconstructions = model.predict(X_valid[:n_images])
      7     if compute_error:
      8         error=X_valid[:n_images]-reconstructions

AttributeError: 'tuple' object has no attribute 'predict'
```

3. (3 points) Visualize the embedding projections for training and validation sets.

In [130]:

```
N_Samples = 25

X_train_set_reconstructed = stacked_ae.predict(X_train_set)
X_valid_set_reconstructed = stacked_ae.predict(X_valid_set)

N_sample_indices = np.random.choice(np.arange(0, X_train_set.shape[0]), N_Samples)
X_train_orig = X_train_set[N_sample_indices]
X_train_reconstructed = X_train_set_reconstructed[N_sample_indices]

plt.figure(figsize=(N_Samples * 1.5 , 2 * 1.5))
plt.suptitle("Train Set Reconstruction Visualization", fontsize=15)

for i in range(N_Samples):

    plt.subplot(2, N_Samples, i + 1)
    plt.imshow(X_train_orig[i].reshape(300, 300), cmap='gray')
    plt.axis('off')

    plt.subplot(2, N_Samples, N_Samples + i + 1)
    plt.imshow(X_train_reconstructed[i].reshape(300, 300), cmap='gray')
    plt.axis('off')

plt.show()

plt.figure(figsize=(N_Samples * 1.5 , 2 * 1.5))
plt.suptitle("Validation Set Reconstruction Visualization", fontsize=15)

N_sample_indices = np.random.choice(np.arange(0, X_valid_set.shape[0]), N_Samples)
X_valid_orig = X_valid_set[N_sample_indices]
X_valid_reconstructed = X_valid_set_reconstructed[N_sample_indices]

for i in range(N_Samples):

    plt.subplot(2, N_Samples, i + 1)
    plt.imshow(X_valid_orig[i].reshape(300, 300), cmap='gray')
    plt.axis('off')

    plt.subplot(2, N_Samples, N_Samples + i + 1)
    plt.imshow(X_valid_reconstructed[i].reshape(300, 300), cmap='gray')
    plt.axis('off')

plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-130-57ecf2e60d4b> in <module>
      1 N_Samples = 25
      2 
----> 3 X_train_set_reconstructed = stacked_ae.predict(X_train_set)
      4 X_valid_set_reconstructed = stacked_ae.predict(X_valid_set)
      5 

AttributeError: 'tuple' object has no attribute 'predict'
```

In [ ]:

```

```

## Problem 3 (15 points)

In this problem, you will be working with the [California Housing dataset (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html)](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html). The California Housing dataset consists of 20,640 samples, each described with 8 features. Let's import it:

In [108]:

```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/ (http://lib.stat.cmu.edu/datasets/)

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

In [109]:

```
X = housing.data # feature matrix (attributes/features are described above)

t = housing.target # target vector (median house value expressed in $100,000)

X.shape, t.shape
```

Out[109]:

```
((20640, 8), (20640,))
```

Answer the following questions:

1. (1 point) Partition the data into a *full training set* and a test set. Use a 80/20 stratified split with a fixed `random_state` . Then partition the *full training set* into a train set and a validation set. For this last partition, use a 70/30 stratified split with a fixed `random_state` .

In [112]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

In [ ]:

```
random_state = 42

X_full_train, X_test, T_full_train, T_test = train_test_split(X, t, test_size=0.2, stratify= t , random_state=random_state)
X_train, X_valid, T_train, T_valid = train_test_split(X_full_train, T_full_train, stratify=t, test_size=0.3, random_state=random_state)
```

2. (1 point) Apply the standardization scaling to the train, validation and test sets. Use the train set to find the mean and standard deviation.

In [ ]:

```
scalar = StandardScaler()
X_train_scaled= scalar.fit_transform(X_train)
X_valid_scaled = scalar.transform(X_valid)
X_test_scaled = scalar.transform(X_test)
```

3.(5 points) Use the Sequential API to build an MLP with 2 hidden layers with the Leaky ReLU activation function and associated alpha=0.2. The first hidden layer should have 50 neurons and the second 10 neurons. How many neurons should you include in the input and output layers? what should be the activation function in the output layer?

In [99]:

```
from tensorflow.keras import Model,Sequential,layers,utils,Input
from tensorflow.keras.layers import Dense,LeakyReLU
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

In [101]:

```
model = keras.models.Sequential(
    [

        keras.layers.Dense(50, input_shape=(8,)),
        keras.layers.LeakyReLU(alpha = 0.2),
        keras.layers.Dense(10),
        keras.layers.LeakyReLU(alpha = 0.2),

        keras.layers.Dense(1, activation="relu")
    ]
)
```

4. (3 points) Compile the model with the Mean Squared Error [loss function (https://keras.io/api/losses/)](https://keras.io/api/losses/), the Adam [optimizer (https://keras.io/api/optimizers/)](https://keras.io/api/optimizers/) with learning rate of 0.001, and the `MeanSquaredError` [performance metric (https://keras.io/api/metrics/)](https://keras.io/api/metrics/).

In [115]:

```
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
loss = tf.keras.losses.MeanAbsoluteError()
accuracy = tf.keras.metrics.MeanAbsoluteError()
```

In [116]:

```
model.compile(optimizer = optimizer, loss = loss, metrics = accuracy)
```

In [119]:

```
model.compile(
    loss = keras.losses.MeanSquaredError(),
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
    metrics = [keras.metrics.MeanSquaredError()]
)
```

In [ ]:

```

```

5. (2 points) Train the model using the train and validation sets with online learning, 200 epochs and early stopping callback with a patience of 10 (on the loss value for the validation set). Plot the learning curves. Discuss your observations.

In [124]:

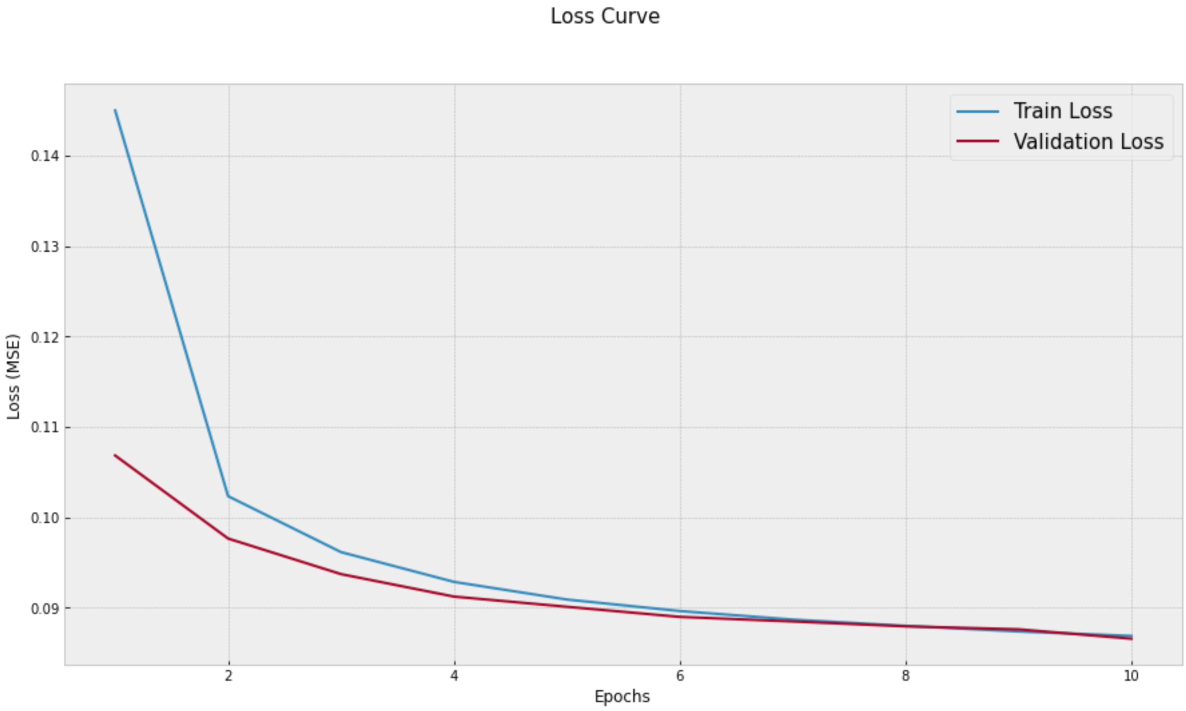
```
def self_plot(train,test,l1,l2,sav):
    fig = plt.figure(figsize=(18,9))
    ax1 = fig.add_subplot(1,1,1)
    ax1.plot(train,label = l1)
    ax1.plot(test,label = l2)
    ax1.legend()
    plt.savefig(sav+'.png')
```

In [77]:

```
b_size = 1 #batch_size
n_epoch = 200
call_back = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                 min_delta=0,
                                                 patience=10,
                                                 verbose=0,
                                                 mode='auto',
                                                 baseline=None,
                                                 restore_best_weights=True)
call_back_2 = tf.keras.callbacks.ModelCheckpoint(filepath = 'model_hw4_pt_2',
                                                  monitor='val_loss',
                                                  mode='max',
                                                  save_weights_only= True)
```

In [78]:

```
epochs = len(history.history['val_loss'])

plt.figure(figsize=(15, 8))
plt.suptitle("Loss Curve", fontsize=15)
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.plot(np.arange(1, epochs + 1), history.history['loss'], label='Train Loss')
plt.plot(np.arange(1, epochs + 1), history.history['val_loss'], label = 'Validation Loss')
plt.legend(fontsize=15)
plt.show()
```




6. (2 points) Evaluate the mean squared error performance in the train and test sets.

In [131]:

```
train_loss, train_mse = model.evaluate(X_train_scaled, T_train)
test_loss, test_mse = model.evaluate(X_test_scaled, T_test)


print(f"Mean Squared Error for Train set is [ {train_mse} ] and for Test set is [ {test_mse} ]")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-131-82b3667e10b9> in <module>
----> 1 train_loss, train_mse = model.evaluate(X_train_scaled, T_train)
      2 test_loss, test_mse = model.evaluate(X_test_scaled, T_test)
      3 
      4 
      5 print(f"Mean Squared Error for Train set is [ {train_mse} ] and for Test set is [ {test_mse} ]")

NameError: name 'X_train_scaled' is not defined
```

7. (2 points) Predict the housing prices for the train and test sets. Use these predictions to calculate the [$r^2$ score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html).

In [132]:

```
from sklearn.metrics import r2_score as r2
```

In [129]:

```
Y_train = model.predict(X_train_scaled)
Y_test = model.predict(X_test_scaled)

r2_train = r2_score(T_train, Y_train)
r2_test = r2_score(T_test, Y_test)

print(f"R2 square for Train is [ {r2_train} ] and for Test is [ {r2_test} ]")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-129-99d13141a505> in <module>
----> 1 Y_train = model.predict(X_train_scaled)
      2 Y_test = model.predict(X_test_scaled)
      3 
      4 r2_train = r2_score(T_train, Y_train)
      5 r2_test = r2_score(T_test, Y_test)

NameError: name 'X_train_scaled' is not defined
```

In [ ]:

```

```

## Submit Your Solution

Confirm that you've successfully completed the assignment.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.