In [1]:

```python
#Updated 3/22/2022
#Import modules
import tensorflow as tf
import time
from tensorflow import keras

from keras.models import Sequential
from keras.layers import Dense
from keras import backend as K

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import importlib

from sklearn.model_selection import train_test_split, KFold
```

In [2]:

```python
# Method that prints progress bar while running tests.
def printProgressBar (iteration, total, prefix = '', suffix = '', decimals = 1, length = 100, fill = '█', pri
ntEnd = "\r"):
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent complete (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
        printEnd    - Optional  : end character (e.g. "\r", "\r\n") (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print(f'\r{prefix} |{bar}| {percent}% {suffix}', end = printEnd)
    # Print New Line on Complete
    if iteration == total:
        print()
```

In [3]:

```
#Load and scale raw S&P 500 data.

path = "datasets/"
spy_df = pd.read_csv(path + 'spy_30min_pi_clean.csv')

spy_df["Date"] = pd.to_datetime(spy_df["Date"])

#Pivot the data such that vertical axis is date, horizontal axis is time of day.
pivot_spy_df = spy_df.copy()
pivot_spy_df = spy_df.pivot(index = "Time", columns = "Date", values = "Close")

spy_np = pivot_spy_df.to_numpy().T

"""
Different Methods for Scaling Data (in numpy arrays)
"""
#Scale by percent change since previous day's market close.
def scale_by_pcspc(np_array):
    scaled_np_array = np_array.copy()
    #Special case: Divide the first day of dataset by its opening price.
    scaled_np_array[0] = (scaled_np_array[0] - np_array[0][0])/np_array[0][0]
    for i in range(1, len(np_array)):
        #Find previous day's market close.
        prev_close = np_array[i-1][12]
        #Find percent change since previous day's market close.
        scaled_np_array[i] = (scaled_np_array[i] - prev_close)/prev_close
    return scaled_np_array

#Scale by percent change since market open.
def scale_by_pcsmo(np_array):
    scaled_np_array = np_array.copy()
    for i in range(len(np_array)):
        #Find percent change since market open.
        scaled_np_array[i] = (scaled_np_array[i] - np_array[i][0])/np_array[i][0]
    return scaled_np_array

#Standardize by each day's input prices.
def standardize_by_daily(np_array, start, end):
    scaled_np_array = np_array.copy()
    for i in range(len(np_array)):
        #Find the day's mean and standard deviation.
        daily_mean = np.mean(np_array[i][start:end])
        daily_std = np.std(np_array[i][start:end])
        #Standardize via mean and standard deviation.
        scaled_np_array[i] = (scaled_np_array[i] - daily_mean)/daily_std
    return scaled_np_array

#Normalize by historical min and max prices.
def normalize_by_hist(np_array):
    scaled_np_array = np_array.copy()
    #Find the historical min and max prices throughout the whole dataset.
    hist_min = np.min(np_array)
    hist_max = np.max(np_array)
    for i in range(len(np_array)):
        #Normalize via historical min and max prices.
        scaled_np_array[i] = (scaled_np_array[i] - hist_min)/(hist_max - hist_min)
    return scaled_np_array

#Standardize by mean and std of historical prices.
def standardize_by_hist(np_array):
    scaled_np_array = np_array.copy()
    #Find the mean and std of historical prices
    hist_mean = np.mean(np_array)
    hist_std = np.std(np_array)
    for i in range(len(np_array)):
        #Standardize via mean and std of historical prices.
        scaled_np_array[i] = (scaled_np_array[i] - hist_mean)/(hist_std)
    return scaled_np_array

scaled_spy_np = scale_by_pcsmo(spy_np)

scaled_spy_np
```

```
Out[3]: array([[ 0.        ,  0.00265128, -0.00155958, ...,  0.00467873,
                 0.00904554,  0.00483468],
               [ 0.        , -0.00837859, -0.00853375, ...,  0.00232739,
                -0.0007758 ,  0.00062064],
               [ 0.        ,  0.01818462,  0.01941748, ...,  0.02327015,
                 0.02619818,  0.028818  ],
               ...,
               [ 0.        , -0.00344097, -0.00605006, ..., -0.0082054 ,
                -0.00461317, -0.00434848],
               [ 0.        ,  0.00437362,  0.00182551, ...,  0.00079866,
                 0.00193961,  0.00258614],
               [ 0.        ,  0.00154309,  0.00255928, ...,  0.00109146,
                -0.00015055,  0.00045164]])
```

In [4]:

```
scaled_spy_df = pivot_spy_df

for col in scaled_spy_df.columns:
    start_close = scaled_spy_df[col].iloc[0]
    min_close = scaled_spy_df[col].min()
    max_close = scaled_spy_df[col].max()
    #Scale by percent change
    scaled_spy_df[col] = scaled_spy_df[col].apply(lambda x : (x-start_close)/(start_close))
    #Round to 3 decimal places.
    scaled_spy_df[col] = scaled_spy_df[col].apply(lambda x : round(x,3))

#scaled_spy_df.dropna(axis=1, inplace = True)

scaled_spy_df
```

Out[4]:

Date	**2002-12-30**	**2002-12-31**	**2003-01-02**	**2003-01-03**	**2003-01-06**	**2003-01-07**	**2003-01-08**	**2003-01-09**	**2003-01-10**	**2003-01-13**	**...**	**2019-01-11**	**2019-01-14**	**2019-01-15**	**2019-01-16**	**2019-01-17**	**2019-01-18**	**2019-01-22**
Time																		
10:00:00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10:30:00	0.003	-0.008	0.018	-0.001	0.004	-0.001	0.003	0.005	0.004	-0.007	...	-0.001	0.001	-0.001	-0.000	0.000	0.001	-0.001
11:00:00	-0.002	-0.009	0.019	-0.002	0.006	-0.005	0.001	0.007	0.008	-0.011	...	0.002	0.003	0.002	-0.001	0.000	0.004	-0.002
11:30:00	-0.002	-0.006	0.018	-0.002	0.007	-0.001	0.005	0.007	0.004	-0.009	...	0.002	0.001	0.001	0.000	0.002	0.006	-0.002
12:00:00	-0.001	-0.004	0.020	-0.002	0.008	0.001	0.002	0.007	0.007	-0.008	...	0.003	0.002	0.004	-0.002	0.002	0.007	-0.006
12:30:00	0.001	-0.002	0.020	-0.003	0.008	-0.000	-0.001	0.006	0.002	-0.007	...	0.004	0.003	0.005	-0.001	0.002	0.009	-0.006
13:00:00	0.000	-0.001	0.022	-0.002	0.010	-0.002	-0.001	0.005	0.000	-0.007	...	0.005	0.003	0.004	-0.000	0.001	0.008	-0.005
13:30:00	0.003	0.002	0.021	-0.001	0.012	0.000	-0.001	0.004	0.000	-0.009	...	0.002	0.003	0.004	-0.000	0.003	0.007	-0.008
14:00:00	0.004	-0.001	0.021	0.000	0.011	0.003	-0.003	0.002	0.002	-0.007	...	0.003	0.003	0.004	-0.000	0.003	0.007	-0.007
14:30:00	0.003	0.002	0.024	-0.003	0.014	0.001	-0.003	0.004	0.003	-0.006	...	0.004	0.002	-0.000	0.001	0.002	0.005	-0.010
15:00:00	0.005	0.002	0.023	-0.004	0.016	-0.001	-0.006	0.005	0.005	-0.008	...	0.003	0.004	0.002	0.000	0.006	0.006	-0.010
15:30:00	0.009	-0.001	0.026	-0.000	0.016	-0.002	-0.006	0.005	0.002	-0.008	...	0.003	0.003	0.003	0.001	0.006	0.006	-0.009
16:00:00	0.005	0.001	0.029	0.002	0.013	-0.003	-0.006	0.007	0.004	-0.009	...	0.005	0.001	0.005	-0.002	0.007	0.007	-0.006

13 rows × 4007 columns

In [5]:

```
#Split data into input and output.
X = scaled_spy_np[:,0:9] #Input: Prices from 10:00:00 to 14:00:00
y = scaled_spy_np[:,9:13] #Output: Prices from 14:30:00 to 16:00:00 (What we are predicting)

#Print the dimensions of the input and output data.
print("scaled_spy_np has length {}".format(len(scaled_spy_np[0])))
print("X has length {} \ny has length {}".format(len(X[0]), len(y[0])))

#Split data into training and testing data.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .1)
```

```
scaled_spy_np has length 13
X has length 9
y has length 4
```

In [6]:

```
#Create a neural network model.

def create_model(input_dim, output_dim, hidden_layers, learning_rate, activation, loss):
    #Create the model
    model = tf.keras.models.Sequential()

    #Add hidden layers
    for i in range(len(hidden_layers)):
        if i==0:
            model.add(tf.keras.layers.Dense(units=hidden_layers[i], activation=activation, input_dim = input_
dim))
        else:
            model.add(tf.keras.layers.Dense(units=hidden_layers[i], activation=activation))

    #Add output layer
    model.add(tf.keras.layers.Dense(units=output_dim))

    #Create the optimizer
    optimizer = keras.optimizers.Adam(lr=learning_rate)

    #Compile the model
    model.compile(optimizer=optimizer, loss=loss)

    return model
```

In [7]:

```
"""
Evaluate a model multiple times and displays average performance.
@params
    model: The model to be evaluated.
    X: The input data
    y: The output data
    k_folds: Number of folds to use in cross validation.
"""

def test_model(model, epochs, X, y, k_folds, **kwargs):
    test_losses = []

    #Print progress bar
    if(kwargs.get('verbose') == 1):
        printProgressBar(0, k_folds, prefix = 'Progress:', suffix = 'Complete', length = 50)

    #Shuffle X and y
    np.random.shuffle(X)
    np.random.shuffle(y)

    #Split data into k folds for cross validation.
    kf = KFold(n_splits=k_folds)
    kf.get_n_splits(X)
    i = 0
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #Fit the model
        model.fit(X_train, y_train, epochs=epochs, verbose=0)

        #Evalaute the model and store the test loss.
        test_loss = model.evaluate(X_test, y_test, verbose=0)

        #Store the performance data
        test_losses.append(test_loss)

        #Update progress bar
        if(kwargs.get('verbose') == 1):
            printProgressBar(i + 1, k_folds, prefix = 'Progress:', suffix = 'Complete', length = 50)
            i += 1
    #Average the performance data across every fold.
    avg_test_loss = np.mean(test_losses)
    return avg_test_loss
```

In [8]:

```
#Create model with 2x32 hidden layers, 0.001 learning rate, ReLu actiation function, and mean average error a
s loss function.
model = create_model(input_dim=9, hidden_layers=[32,32], output_dim=4, learning_rate=0.001, activation='relu'
,
                     loss='mae')

#Test model and store the average test loss
avg_test_loss = test_model(model, 30, X, y, k_folds=10, verbose=1)

#Print average test loss
print("Average test loss: {}".format(avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Average test loss: 0.005020525585860014
```

In [9]:

```python
#Optimize learning rate

#Array containing learning rates to test
learning_rates = np.array([0.0001,0.0005,0.001,0.005,0.01,0.05,0.1])
avg_test_losses = []

#Create and test a model with every learning rate.
for learning_rate in learning_rates:
    #Create model
    model = create_model(input_dim=9, hidden_layers=[32,32], output_dim=4, learning_rate=learning_rate, activ
ation='tanh',
                         loss='mae')

    #Find average test loss
    avg_test_loss = test_model(model, 30, X, y, k_folds=10, verbose=1)

    #Print test loss for each learning rate
    print("Learning Rate {} achieved test loss of {}".format(learning_rate, avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.0001 achieved test loss of 0.004993808083236217
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.0005 achieved test loss of 0.004987978003919124
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.001 achieved test loss of 0.00497784917242825
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.005 achieved test loss of 0.005065590282902122
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.01 achieved test loss of 0.005232418375089765
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.05 achieved test loss of 0.15582910869270564
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Learning Rate 0.1 achieved test loss of 0.2611540764570236
```

In [10]:

```python
#Optimize Epochs

#Array containing number of epochs to test
epochs = [2,4,8,16,32, 64, 128]
avg_test_losses = []

#Create and test a model with every number of epochs.
for epoch in epochs:
    #Create model
    model = create_model(input_dim=9, hidden_layers=[32,32], output_dim=4, learning_rate=0.001, activation='tanh',
                     loss='mae')

    #Find average test loss
    avg_test_loss = test_model(model, epoch, X, y, k_folds=10, verbose=1)

    #Print test loss for each epoch
    print("Epoch {} achieved test loss of {}".format(epoch, avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 2 achieved test loss of 0.005035998485982418
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 4 achieved test loss of 0.005006768787279725
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 8 achieved test loss of 0.0050131228752434255
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 16 achieved test loss of 0.0049853658769279715
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 32 achieved test loss of 0.004992210166528821
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 64 achieved test loss of 0.0050052586942911145
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Epoch 128 achieved test loss of 0.004989310633391142
```

In [11]:

```
#Optimize Number of Hidden Layers

#Array containing number of hidden layers to test
num_hidden_layers = [2,4,8,16,32,64]
avg_test_losses = []

#Create and test a model with every number of hidden layers.
for num in num_hidden_layers:
    #Create model
    model = create_model(input_dim=9, hidden_layers=[32]*num, output_dim=4, learning_rate=0.001, activation=
'tanh',
                     loss='mae')

    #Find average test loss.
    avg_test_loss = test_model(model, 30, X, y, k_folds=10, verbose=1)

    #Print test loss for each number of hidden layers
    print("Num hidden layers of {} achieved test loss of {}".format(num, avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Num hidden layers of 2 achieved test loss of 0.00502304169349372
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Num hidden layers of 4 achieved test loss of 0.005040197027847170
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Num hidden layers of 8 achieved test loss of 0.004988944577053189
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Num hidden layers of 16 achieved test loss of 0.00501821525394916
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Num hidden layers of 32 achieved test loss of 0.00500631891191005
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Num hidden layers of 64 achieved test loss of 0.00500677218660712
```

In [12]:

```
#Optimize Hidden Layer Size

#Array containing hidden layer sizes to test
hidden_layer_sizes = [2,4,8,16,32,64,128]
avg_test_losses = []

#Create and test a model with every hidden layer size.
for size in hidden_layer_sizes:
    #Create model
    model = create_model(input_dim=9, hidden_layers=[size]*2, output_dim=4, learning_rate=0.001, activation=
'tanh',
                     loss='mae')

    #Find average test loss
    avg_test_loss = test_model(model, 30, X, y, k_folds=10, verbose=1)

    #Print test loss for each hidden layer size
    print("Hidden layer size {} achieved test loss of {}".format(size, avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 2 achieved test loss of 0.005024055717512965
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 4 achieved test loss of 0.004986590845510364
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 8 achieved test loss of 0.004993353085592389
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 16 achieved test loss of 0.005004382180050016
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 32 achieved test loss of 0.005000432068482041
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 64 achieved test loss of 0.005024759937077761
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Hidden layer size 128 achieved test loss of 0.005001226114109159
```

In [13]:
```python
#Optimize Activation Function

#Array containing activation functions to test
activations = ['tanh', 'relu', 'sigmoid', 'softmax']

#Create and test a model with activation function.
for activation in activations:
    #Create model
    model = create_model(input_dim=9, hidden_layers=[32,32], output_dim=4, learning_rate=0.001, activation=activation,
                     loss='mae')

    #Find average test loss
    avg_test_loss = test_model(model, 30, X, y, k_folds=10, verbose=1)

    #Print test loss for each activation function
    print("Activation {} achieved test loss of {}".format(activation, avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Activation tanh achieved test loss of 0.004989201761782169
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Activation relu achieved test loss of 0.005007611168548465
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Activation sigmoid achieved test loss of 0.005579605698585 51
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Activation softmax achieved test loss of 0.005043719569221139
```

In [14]:
```python
#Test Optimized Model

#Create an optimized model using the hyperparameters with the lowest test losses.
model = create_model(input_dim=9, hidden_layers=[16]*8, output_dim=4, learning_rate=0.0005, activation='softmax',
                     loss='mae')

#Find average test loss of optimized model
avg_test_loss = test_model(model, 32, X, y, k_folds=10, verbose=1)

#Print test loss for optimized model
print("Final model achieved test loss of {}".format(avg_test_loss))
```

```
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Final model achieved test loss of 0.004994870815426111
```