INF385T_ML_Project_Prototype_updated

April 17, 2018

```
In [1]: # import required packages

        import numpy as np
        import pandas as pd
        import math
        import sklearn
        import sklearn.preprocessing
        import datetime
        import os
        import matplotlib.pyplot as plt
        import tensorflow as tf

In [2]: # display parent directory and working directory

        print(os.path.dirname(os.getcwd())+':', os.listdir(os.path.dirname(os.getcwd())));
        print(os.getcwd()+':', os.listdir(os.getcwd()));

/home/nbuser: ['.bash_logout', '.profile', '.bashrc', '.local', '.cache', '.nb.setup.log', 'anac
/home/nbuser/library: ['lab5.ipynb', 'Lab Assignment 3.ipynb', 'testing.ipynb', 'prices split ad


In [3]: # split data in 80%/10%/10% train/validation/test sets

        valid_set_size_percentage = 10
        test_set_size_percentage = 10

In [4]: # import stock prices data

        df = pd.read_csv("prices-split-adjusted.csv", index_col = 0)
        df.info()

        # number of different stocks

        print('\nnumber of different stocks: ', len(list(set(df.symbol))))
        print(list(set(df.symbol))[:10])

<class 'pandas.core.frame.DataFrame'>
Index: 851264 entries, 2016-01-05 to 2016-12-30
```

```
Data columns (total 6 columns):
symbol    851264 non-null object
open      851264 non-null float64
close     851264 non-null float64
low       851264 non-null float64
high      851264 non-null float64
volume    851264 non-null float64
dtypes: float64(5), object(1)
memory usage: 45.5+ MB

number of different stocks:  501
['LLY', 'SCG', 'PRU', 'PAYX', 'HD', 'RAI', 'MAR', 'XRX', 'GWW', 'SYK']
```

In [5]: `df.head()`

Out[5]:

```
           symbol        open       close         low        high     volume
date
2016-01-05   WLTW  123.430000  125.839996  122.309998  126.250000  2163600.0
2016-01-06   WLTW  125.239998  119.980003  119.940002  125.540001  2386400.0
2016-01-07   WLTW  116.379997  114.949997  114.930000  119.739998  2489500.0
2016-01-08   WLTW  115.480003  116.620003  113.500000  117.440002  2006300.0
2016-01-11   WLTW  117.010002  114.970001  114.089996  117.330002  1408600.0
```

In [6]: `df.tail()`

Out[6]:

```
           symbol        open       close         low        high     volume
date
2016-12-30    ZBH  103.309998  103.199997  102.849998  103.930000   973800.0
2016-12-30   ZION   43.070000   43.040001   42.689999   43.310001  1938100.0
2016-12-30    ZTS   53.639999   53.529999   53.270000   53.740002  1701200.0
2016-12-30    AIV   44.730000   45.450001   44.410000   45.590000  1380900.0
2016-12-30    FTV   54.200001   53.630001   53.389999   54.480000   705100.0
```

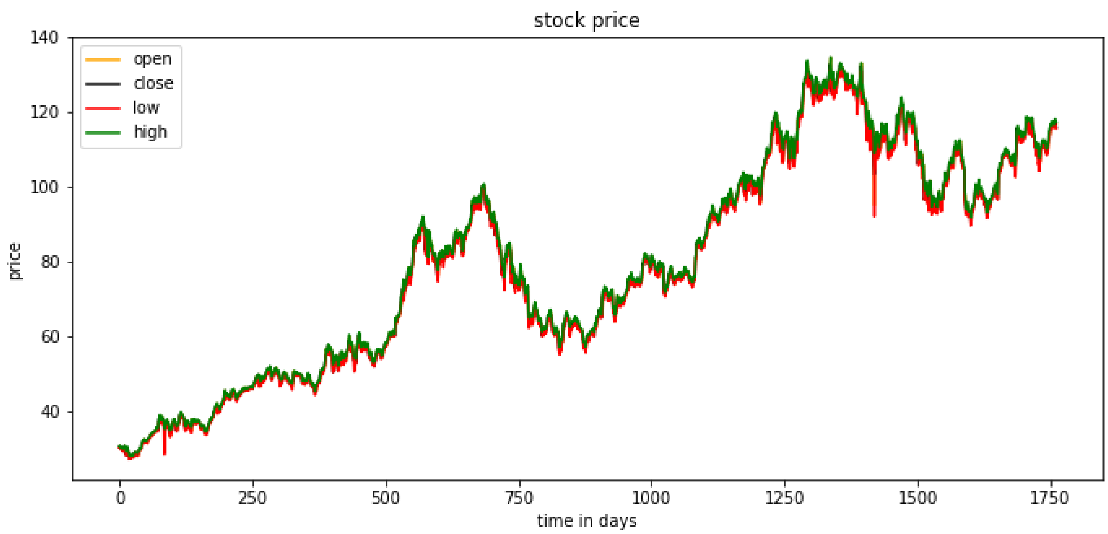
In [7]: `df.describe()`

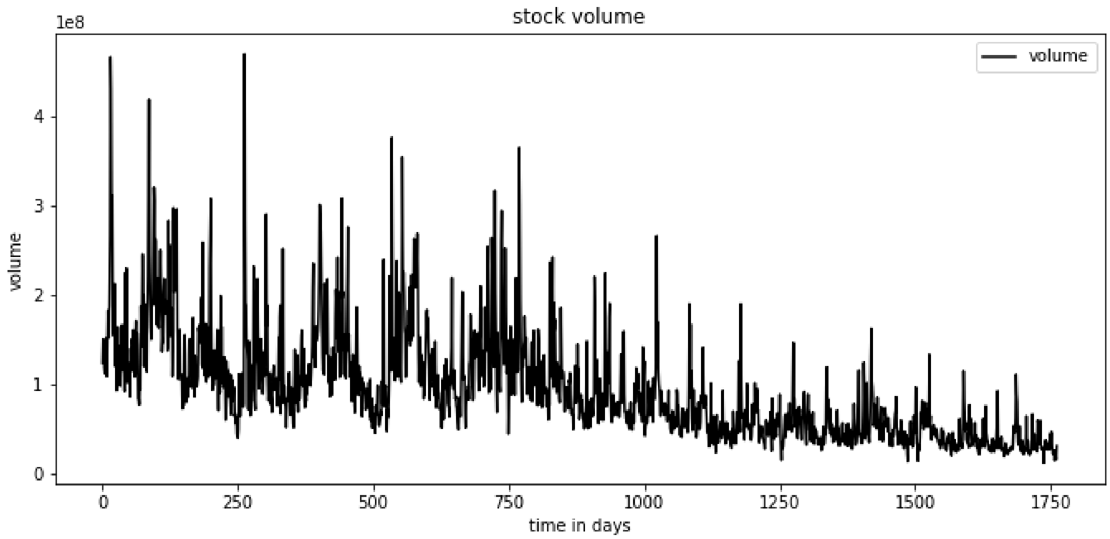
Out[7]:

```
                open          close            low           high  \
count  851264.000000  851264.000000  851264.000000  851264.000000
mean       64.993618      65.011913      64.336541      65.639748
std        75.203893      75.201216      74.459518      75.906861
min         1.660000       1.590000       1.500000       1.810000
25%        31.270000      31.292776      30.940001      31.620001
50%        48.459999      48.480000      47.970001      48.959999
75%        75.120003      75.139999      74.400002      75.849998
max      1584.439941    1578.130005    1549.939941    1600.930054

             volume
count  8.512640e+05
mean   5.415113e+06
```

```
std      1.249468e+07
min      0.000000e+00
25%      1.221500e+06
50%      2.476250e+06
75%      5.222500e+06
max      8.596434e+08
```

```
In [8]: #Plotting volume and price of a specific stock versus time

        plt.figure(figsize=(25, 5));
        plt.subplot(1,2,1);
        plt.plot(df[df.symbol == 'AAPL'].open.values, color='orange', label='open')
        plt.plot(df[df.symbol == 'AAPL'].close.values, color='black', label='close')
        plt.plot(df[df.symbol == 'AAPL'].low.values, color='red', label='low')
        plt.plot(df[df.symbol == 'AAPL'].high.values, color='green', label='high')
        plt.title('stock price')
        plt.xlabel('time in days')
        plt.ylabel('price')
        plt.legend(loc='best')
        # plt.show()

        plt.figure(figsize=(25, 5));
        plt.subplot(1,2,2);
        plt.plot(df[df.symbol == 'AAPL'].volume.values, color='black', label='volume')
        plt.title('stock volume')
        plt.xlabel('time in days')
        plt.ylabel('volume')
        plt.legend(loc='best');
        plt.show()
```



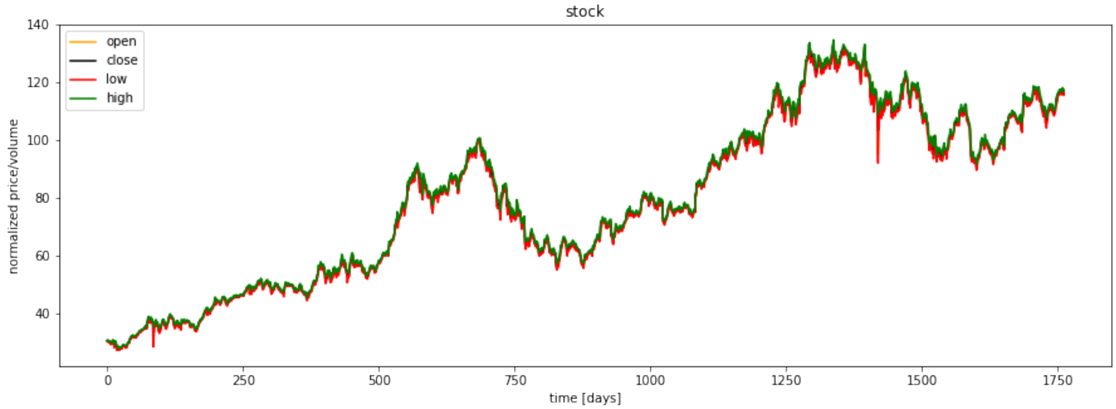


In [9]:
```
# function for min-max normalization of stock
def normalize_data(df):
    min_max_scaler = sklearn.preprocessing.MinMaxScaler()
    df['open'] = min_max_scaler.fit_transform(df.open.values.reshape(-1,1))
    df['high'] = min_max_scaler.fit_transform(df.high.values.reshape(-1,1))
    df['low'] = min_max_scaler.fit_transform(df.low.values.reshape(-1,1))
    df['close'] = min_max_scaler.fit_transform(df['close'].values.reshape(-1,1))
    return df

# function to create train, validation, test data given stock data and sequence length
def load_data(stock, seq_len):
    raw_data = stock.as_matrix() # convert to numpy array
    data = []

    # create all possible sequences of length seq_len
    for index in range(len(raw_data) - seq_len):
        data.append(raw_data[index: index + seq_len])

    data = np.array(data);
    valid_set_size = int(np.round(valid_set_size_percentage/100*data.shape[0]));
    test_set_size = int(np.round(test_set_size_percentage/100*data.shape[0]));
    train_set_size = data.shape[0] - (valid_set_size + test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = data[:train_set_size,-1,:]

    x_valid = data[train_set_size:train_set_size+valid_set_size,:-1,:]
    y_valid = data[train_set_size:train_set_size+valid_set_size,-1,:]
```

```
    x_test = data[train_set_size+valid_set_size:,:-1,:]
    y_test = data[train_set_size+valid_set_size:,-1,:]

    return [x_train, y_train, x_valid, y_valid, x_test, y_test]

# choose one stock
df_stock = df[df.symbol == 'AAPL'].copy()
df_stock.drop(['symbol'],1,inplace=True)
df_stock.drop(['volume'],1,inplace=True)

cols = list(df_stock.columns.values)
print('df_stock.columns.values = ', cols)

# normalize stock
df_stock_norm = df_stock.copy()
df_stock_norm = normalize_data(df_stock_norm)

# create train, test data
seq_len = 20 # choose sequence length
x_train, y_train, x_valid, y_valid, x_test, y_test = load_data(df_stock_norm, seq_len)
print('x_train.shape = ',x_train.shape)
print('y_train.shape = ', y_train.shape)
print('x_valid.shape = ',x_valid.shape)
print('y_valid.shape = ', y_valid.shape)
print('x_test.shape = ', x_test.shape)
print('y_test.shape = ',y_test.shape)
```

```
df_stock.columns.values =  ['open', 'close', 'low', 'high']
x_train.shape =  (1394, 19, 4)
y_train.shape =  (1394, 4)
x_valid.shape =  (174, 19, 4)
y_valid.shape =  (174, 4)
x_test.shape =  (174, 19, 4)
y_test.shape =  (174, 4)
```

In [10]:
```
plt.figure(figsize=(15, 5));
plt.plot(df[df.symbol == 'AAPL'].open.values, color='orange', label='open')
plt.plot(df[df.symbol == 'AAPL'].close.values, color='black', label='close')
plt.plot(df[df.symbol == 'AAPL'].low.values, color='red', label='low')
plt.plot(df[df.symbol == 'AAPL'].high.values, color='green', label='high')

#plt.plot(df_stock_norm.volume.values, color='gray', label='volume')

plt.title('stock')
plt.xlabel('time [days]')
plt.ylabel('normalized price/volume')
plt.legend(loc='best')
plt.show()
```



In [11]:
```
# prepare batches

index_in_epoch = 0;
permutation_array  = np.arange(x_train.shape[0])
np.random.shuffle(permutation_array)

# function to get the next batch
def get_next_batch(batch_size):
    global index_in_epoch, x_train, perm_array
    start = index_in_epoch
    index_in_epoch += batch_size

    if index_in_epoch > x_train.shape[0]:
        np.random.shuffle(permutation_array) # shuffle permutation array
        start = 0 # start next epoch
        index_in_epoch = batch_size

    end = index_in_epoch
    return x_train[permutation_array[start:end]], y_train[permutation_array[start:end]]
```

In [12]:
```
## Model Run w/ Basic RNN

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 200
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
```

```python
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use Basic RNN Cell
layers = [tf.contrib.rnn.BasicRNNCell(num_units=n_neurons, activation=tf.nn.elu)
          for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next training batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})
            print('%.2f epochs: MSE train/valid = %.6f/%.6f'%(
                iteration*batch_size/train_set_size, mse_train, mse_valid))

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})
```

```
0.00 epochs: MSE train/valid = 0.077708/0.139151
4.99 epochs: MSE train/valid = 0.000588/0.001505
9.97 epochs: MSE train/valid = 0.000238/0.000627
14.96 epochs: MSE train/valid = 0.000234/0.000552
19.94 epochs: MSE train/valid = 0.000230/0.000630
24.93 epochs: MSE train/valid = 0.000212/0.000523
29.91 epochs: MSE train/valid = 0.000142/0.000408
34.90 epochs: MSE train/valid = 0.000144/0.000402
39.89 epochs: MSE train/valid = 0.000162/0.000431
```

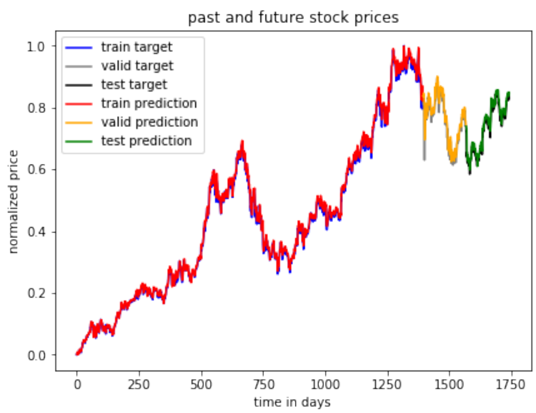
```
44.87 epochs: MSE train/valid = 0.000122/0.000353
49.86 epochs: MSE train/valid = 0.000137/0.000386
54.84 epochs: MSE train/valid = 0.000120/0.000346
59.83 epochs: MSE train/valid = 0.000156/0.000432
64.81 epochs: MSE train/valid = 0.000149/0.000398
69.80 epochs: MSE train/valid = 0.000169/0.000464
74.78 epochs: MSE train/valid = 0.000158/0.000441
79.77 epochs: MSE train/valid = 0.000113/0.000311
84.76 epochs: MSE train/valid = 0.000107/0.000295
89.74 epochs: MSE train/valid = 0.000101/0.000289
94.73 epochs: MSE train/valid = 0.000108/0.000294
99.71 epochs: MSE train/valid = 0.000117/0.000336
```

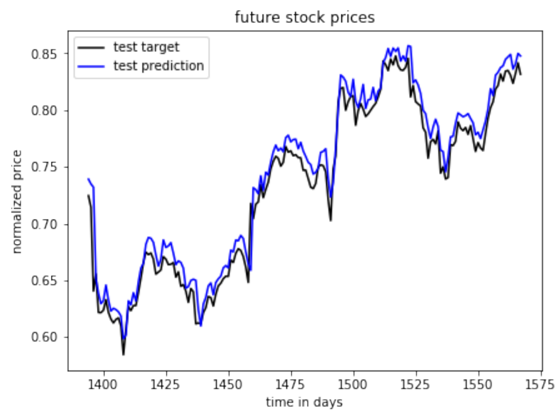
In [13]:
```python
print(y_train.shape[0])
print(y_test.shape[0])
```

```
1394
174
```

In [14]:
```python
var = 0 # 0 = open, 1 = close, 2 = highest, 3 = lowest

## show predictions
plt.figure(figsize=(15, 5));
plt.subplot(1,2,1);

plt.plot(np.arange(y_train.shape[0]), y_train[:,var], color='blue', label='train target

plt.plot(np.arange(y_train.shape[0], y_train.shape[0]+y_valid.shape[0]), y_valid[:,var]
         color='gray', label='valid target')

plt.plot(np.arange(y_train.shape[0]+y_valid.shape[0],
                   y_train.shape[0]+y_test.shape[0]+y_test.shape[0]),
         y_test[:,var], color='black', label='test target')

plt.plot(np.arange(y_train_pred.shape[0]),y_train_pred[:,var], color='red',
         label='train prediction')

plt.plot(np.arange(y_train_pred.shape[0], y_train_pred.shape[0]+y_valid_pred.shape[0]),
         y_valid_pred[:,var], color='orange', label='valid prediction')

plt.plot(np.arange(y_train_pred.shape[0]+y_valid_pred.shape[0],
                   y_train_pred.shape[0]+y_valid_pred.shape[0]+y_test_pred.shape[0]),
         y_test_pred[:,var], color='green', label='test prediction')

plt.title('past and future stock prices')
plt.xlabel('time in days')
```

```
plt.ylabel('normalized price')
plt.legend(loc='best');

plt.subplot(1,2,2);

plt.plot(np.arange(y_train.shape[0], y_train.shape[0]+y_test.shape[0]),
         y_test[:,var], color='black', label='test target')

plt.plot(np.arange(y_train_pred.shape[0], y_train_pred.shape[0]+y_test_pred.shape[0]),
         y_test_pred[:,var], color='blue', label='test prediction')

plt.title('future stock prices')
plt.xlabel('time in days')
plt.ylabel('normalized price')
plt.legend(loc='best');

plt.show()
```





In [15]:
```
# classify into binary classes based on Close-Open predictions (gainers= +, lossers= -)

corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_train.shape[

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_valid.shape[

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.shape[0]


print('correct classifiation of close - open price for train/valid/test: %.2f/%.2f/%.2f
    corr_price_development_train, corr_price_development_valid, corr_price_development_
```

```
correct classifiation of close - open price for train/valid/test: 0.74/0.77/0.89
```

```
In [16]: ## Model Run w/ Gated RNN, LSTM

         # parameters
         n_steps = seq_len-1
         n_inputs = 4
         n_neurons = 200
         n_outputs = 4
         n_layers = 2
         learning_rate = 0.001
         batch_size = 50
         n_epochs = 100
         train_set_size = x_train.shape[0]
         test_set_size = x_test.shape[0]

         tf.reset_default_graph()

         X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
         y = tf.placeholder(tf.float32, [None, n_outputs])

         # use LSTM Cell
         layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.elu)
                   for layer in range(n_layers)]

         multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
         rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

         stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
         stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
         outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
         outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

         loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared error
         optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
         training_op = optimizer.minimize(loss)

         # run model
         with tf.Session() as sess:
             sess.run(tf.global_variables_initializer())
             for iteration in range(int(n_epochs*train_set_size/batch_size)):
                 x_batch, y_batch = get_next_batch(batch_size) # fetch the next training batch
                 sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
                 if iteration % int(5*train_set_size/batch_size) == 0:
                     mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
                     mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})
                     print('%.2f epochs: MSE train/valid = %.6f/%.6f'%(
```

```
                        iteration*batch_size/train_set_size, mse_train, mse_valid))

            y_train_pred = sess.run(outputs, feed_dict={X: x_train})
            y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
            y_test_pred = sess.run(outputs, feed_dict={X: x_test})

        # classify into binary classes based on Close-Open predictions (gainers= +, lossers= -)

        corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train[:,0]),
                    np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_train.shape[

        corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid[:,0]),
                    np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_valid.shape[

        corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0]),
                    np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.shape[0]


        print('correct classifiation of close - open price for train/valid/test: %.2f/%.2f/%.2f
            corr_price_development_train, corr_price_development_valid, corr_price_development_
```

```
0.00 epochs: MSE train/valid = 0.134898/0.319776
4.99 epochs: MSE train/valid = 0.000780/0.001395
9.97 epochs: MSE train/valid = 0.000527/0.001028
14.96 epochs: MSE train/valid = 0.000496/0.001112
19.94 epochs: MSE train/valid = 0.000697/0.001356
24.93 epochs: MSE train/valid = 0.000489/0.000936
29.91 epochs: MSE train/valid = 0.000302/0.000622
34.90 epochs: MSE train/valid = 0.000273/0.000660
39.89 epochs: MSE train/valid = 0.000238/0.000553
44.87 epochs: MSE train/valid = 0.000233/0.000518
49.86 epochs: MSE train/valid = 0.000197/0.000475
54.84 epochs: MSE train/valid = 0.000174/0.000410
59.83 epochs: MSE train/valid = 0.000173/0.000388
64.81 epochs: MSE train/valid = 0.000157/0.000388
69.80 epochs: MSE train/valid = 0.000138/0.000333
74.78 epochs: MSE train/valid = 0.000197/0.000498
79.77 epochs: MSE train/valid = 0.000128/0.000328
84.76 epochs: MSE train/valid = 0.000127/0.000313
89.74 epochs: MSE train/valid = 0.000112/0.000285
94.73 epochs: MSE train/valid = 0.000111/0.000292
99.71 epochs: MSE train/valid = 0.000142/0.000358
correct classifiation of close - open price for train/valid/test: 0.75/0.78/0.89
```

In [17]:
```
# Optimize Hyper-paramters of Model
```

In [ ]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Neurons
```

```
number_neurons=[]
best_neurons=[]

for x in range(1,501,100):
    # parameters
    n_steps = seq_len-1
    n_inputs = 4
    n_neurons = x
    n_outputs = 4
    n_layers = 2
    learning_rate = 0.001
    batch_size = 50
    n_epochs = 100
    train_set_size = x_train.shape[0]
    test_set_size = x_test.shape[0]

    tf.reset_default_graph()

    X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
    y = tf.placeholder(tf.float32, [None, n_outputs])

    # use LSTM Cell
    layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.elu)
             for layer in range(n_layers)]

    multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
    rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

    stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
    stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
    outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
    outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

    loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared error
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    training_op = optimizer.minimize(loss)

    # run model
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for iteration in range(int(n_epochs*train_set_size/batch_size)):
            x_batch, y_batch = get_next_batch(batch_size) # fetch the next training batc
            sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
            if iteration % int(5*train_set_size/batch_size) == 0:
                mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
                mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

        y_train_pred = sess.run(outputs, feed_dict={X: x_train})
```

```
            y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
            y_test_pred = sess.run(outputs, feed_dict={X: x_test})

        # classify into binary classes based on Close-Open predictions (gainers= +, lossers=

        corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0]),
                    np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.shape[

        number_neurons.append(x)
        best_neurons.append(corr_price_development_test)

    # evaluate results

    plt.plot(x=number_neurons,y=best_neurons,color='black')
    plt.title('Optimizing Number of Neurons')
    plt.xlabel('Number of Neurons')
    plt.ylabel('Correct Classifications')

    plt.show()

    selected_value_neurons = max(best_neurons)
    selected_index = best_neurons.index(selected_value_neurons)
    selected_neurons = number_neurons[selected_index]

    print('Optimal Number of Neurons: %1f'%(selected_neurons))
    print('Highest correct classifiation of close - open price for test set: %.15f'%(selecte
```

In [20]:

```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Layers

number_layers=[]
best_layers=[]

for x in range(1,6,1):
    # parameters
    n_steps = seq_len-1
    n_inputs = 4
    n_neurons = selected_value_neurons
    n_outputs = 4
    n_layers = x
    learning_rate = 0.001
    batch_size = 50
    n_epochs = 100
    train_set_size = x_train.shape[0]
    test_set_size = x_test.shape[0]

    tf.reset_default_graph()

    X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
```

```
    y = tf.placeholder(tf.float32, [None, n_outputs])

    # use LSTM Cell
    layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.elu)
             for layer in range(n_layers)]

    multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
    rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

    stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
    stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
    outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
    outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

    loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared error
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    training_op = optimizer.minimize(loss)

    # run model
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for iteration in range(int(n_epochs*train_set_size/batch_size)):
            x_batch, y_batch = get_next_batch(batch_size) # fetch the next training bat
            sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
            if iteration % int(5*train_set_size/batch_size) == 0:
                mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
                mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

        y_train_pred = sess.run(outputs, feed_dict={X: x_train})
        y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
        y_test_pred = sess.run(outputs, feed_dict={X: x_test})

    # classify into binary classes based on Close-Open predictions (gainers= +, lossers

    corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0]),
                np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.shape

    number_layers.append(x)
    best_layers.append(corr_price_development_test)

# evaluate results

plt.plot(x=number_layers,y=best_layers,color='black')
plt.title('Optimizing Number of Layers')
plt.xlabel('Number of Layers')
plt.ylabel('Correct Classifications')

plt.show()
```

```
selected_value = max(best_layers)
selected_index = best_layers.index(selected_value)
selected_layers = number_layers[selected_index]

print('Optimal Number of Layers: %1f'%(selected_layers))
print('Highest correct classifiation of close - open price for test set: %.15f'%(select
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

~\Anaconda3\lib\site-packages\tensorflow\python\framework\tensor_shape.py in __init__(se
    427       try:
--> 428         dims_iter = iter(dims)
    429       except TypeError:


TypeError: 'numpy.float64' object is not iterable


During handling of the above exception, another exception occurred:


ValueError                                Traceback (most recent call last)

<ipython-input-20-07813d3618f8> in <module>()
     27
     28     multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
---> 29     rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)
     30
     31     stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])


~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn.py in dynamic_rnn(cell, inputs,
    546       if not dtype:
    547         raise ValueError("If there is no initial_state, you must give a dtype.")
--> 548       state = cell.zero_state(batch_size, dtype)
    549
    550     def _assert_has_shape(x, shape):


~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in zero_state(self,
    891     with ops.name_scope(type(self).__name__ + "ZeroState", values=[batch_size]):
    892       if self._state_is_tuple:
--> 893         return tuple(cell.zero_state(batch_size, dtype) for cell in self._cells)
    894       else:
```

```
    895         # We know here that state_size of each cell is not a tuple and


    ~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in <genexpr>(.0)
    891     with ops.name_scope(type(self).__name__ + "ZeroState", values=[batch_size]):
    892       if self._state_is_tuple:
--> 893         return tuple(cell.zero_state(batch_size, dtype) for cell in self._cells)
    894       else:
    895         # We know here that state_size of each cell is not a tuple and


    ~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in zero_state(self,
    227     with ops.name_scope(type(self).__name__ + "ZeroState", values=[batch_size]):
    228       state_size = self.state_size
--> 229       return _zero_state_tensors(state_size, batch_size, dtype)
    230
    231


    ~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in _zero_state_tens
    128     size.set_shape(c_static)
    129     return size
--> 130   return nest.map_structure(get_state_shape, state_size)
    131
    132


    ~\Anaconda3\lib\site-packages\tensorflow\python\util\nest.py in map_structure(func, *str
    323
    324   return pack_sequence_as(
--> 325       structure[0], [func(*x) for x in entries])
    326
    327


    ~\Anaconda3\lib\site-packages\tensorflow\python\util\nest.py in <listcomp>(.0)
    323
    324   return pack_sequence_as(
--> 325       structure[0], [func(*x) for x in entries])
    326
    327


    ~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in get_state_shape(
    123   def get_state_shape(s):
    124     """Combine s with batch_size to get a proper tensor shape."""
--> 125     c = _concat(batch_size, s)
    126     c_static = _concat(batch_size, s, static=True)
```

```
    127     size = array_ops.zeros(c, dtype=dtype)


    ~\Anaconda3\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in _concat(prefix,
    103                           "but saw tensor: %s" % s)
    104   else:
--> 105     s = tensor_shape.as_shape(suffix)
    106     s_static = s.as_list() if s.ndims is not None else None
    107     s = (constant_op.constant(s.as_list(), dtype=dtypes.int32)


    ~\Anaconda3\lib\site-packages\tensorflow\python\framework\tensor_shape.py in as_shape(sh
    796     return shape
    797   else:
--> 798     return TensorShape(shape)
    799
    800


    ~\Anaconda3\lib\site-packages\tensorflow\python\framework\tensor_shape.py in __init__(se
    429       except TypeError:
    430         # Treat as a singleton dimension
--> 431         self._dims = [as_dimension(dims)]
    432       else:
    433         # Got a list of dimensions


    ~\Anaconda3\lib\site-packages\tensorflow\python\framework\tensor_shape.py in as_dimensio
    374     return value
    375   else:
--> 376     return Dimension(value)
    377
    378


    ~\Anaconda3\lib\site-packages\tensorflow\python\framework\tensor_shape.py in __init__(se
     33       if (not isinstance(value, compat.bytes_or_text_types) and
     34           self._value != value):
---> 35         raise ValueError("Ambiguous dimension: %s" % value)
     36       if self._value < 0:
     37         raise ValueError("Dimension %d must be >= 0" % self._value)


    ValueError: Ambiguous dimension: 0.896551724138
```

In [ ]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Batch Sizes
```

```
number_batch=[]
best_batch=[]

for x in range(1,101,10):
    # parameters
    n_steps = seq_len-1
    n_inputs = 4
    n_neurons = selected_value_neurons
    n_outputs = 4
    n_layers = selected_layers
    learning_rate = 0.001
    batch_size = x
    n_epochs = 100
    train_set_size = x_train.shape[0]
    test_set_size = x_test.shape[0]

    tf.reset_default_graph()

    X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
    y = tf.placeholder(tf.float32, [None, n_outputs])

    # use LSTM Cell
    layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.elu)
              for layer in range(n_layers)]

    multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
    rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

    stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
    stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
    outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
    outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

    loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared error
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    training_op = optimizer.minimize(loss)

    # run model
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for iteration in range(int(n_epochs*train_set_size/batch_size)):
            x_batch, y_batch = get_next_batch(batch_size) # fetch the next training batc
            sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
            if iteration % int(5*train_set_size/batch_size) == 0:
                mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
                mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

        y_train_pred = sess.run(outputs, feed_dict={X: x_train})
```

```
            y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
            y_test_pred = sess.run(outputs, feed_dict={X: x_test})

        # classify into binary classes based on Close-Open predictions (gainers= +, lossers=

        corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0]),
                    np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.shape[

        number_batch.append(x)
        best_batch.append(corr_price_development_test)

    # evaluate results

    plt.plot(x=number_batch,y=best_batch,color='black')
    plt.title('Optimizing Number of Batches')
    plt.xlabel('Number of Batches')
    plt.ylabel('Correct Classifications')

    plt.show()

    selected_value = max(best_batch)
    selected_index = best_batch.index(selected_value)
    selected_batch = number_batch[selected_index]

    print('Optimal Number of Batches: %1f'%(selected_batch))
    print('Highest correct classifiation of close - open price for test set: %.15f'%(selecte
```