Stock Price Predictions in NY

By: Dhruv and David

Our Project

- We utilized recurrent neural networks in tensorflow to demonstrate the future price prediction for different stocks in the New York Stock Exchange.
- 5 Steps:
  - Imported Libraries and Settings
  - Analyzed the data
  - Manipulated the Data
  - Modeled and Validated the Data
  - Predicted the Outcomes

Importing Libraries and Settings

- Before proceeding with the project, we made sure to mount the google drive to the Google Colaboratory Document.
- We downloaded a stock price data source, imported the stock prices into a header, then printed out the information from the data.

```
# import all stock prices
df = pd.read_csv("../input/prices-split-adjusted.csv", index_col = 0)
df.info()
df.head()

# 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
['AAPL', 'CLX', 'ETR', 'MCK', 'WMT', 'HCN', 'CTSH', 'NVDA', 'AIV', 'EFX']
```

Data Visualization

df.describe():

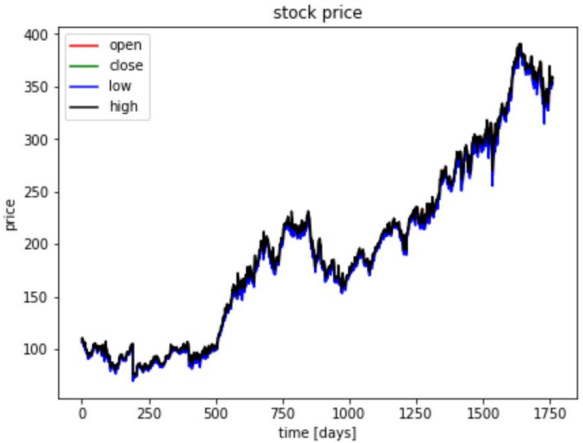
	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	64.993618	65.011913	64.336541	65.639748	5.415113e+06
std	75.203893	75.201216	74.459518	75.906861	1.249468e+07
min	1.660000	1.590000	1.500000	1.810000	0.000000e+00
25%	31.270000	31.292776	30.940001	31.620001	1.221500e+06
50%	48.459999	48.480000	47.970001	48.959999	2.476250e+06
75%	75.120003	75.139999	74.400002	75.849998	5.222500e+06
max	1584.439941	1578.130005	1549.939941	1600.930054	8.596434e+08

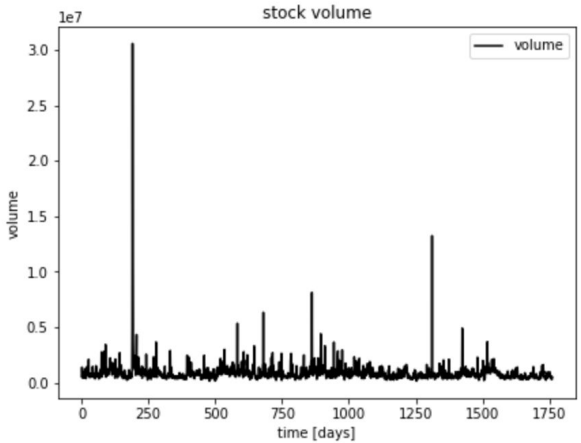
df.tail():

	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

Analyzing the Data

We used pandas to analyze the data, and our initial plot of the data looked like this:





Data Manipulation

- After analyzing our data, we had a few next steps to manipulate the data in what we wanted to see.
  - We first wanted to choose a specific stock to use our model on
  - Next, we use a feature (volume) for our prediction
  - Then, we had to normalize our stock data
  - Finally, create the train, validation, and test sets

```
# 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
```

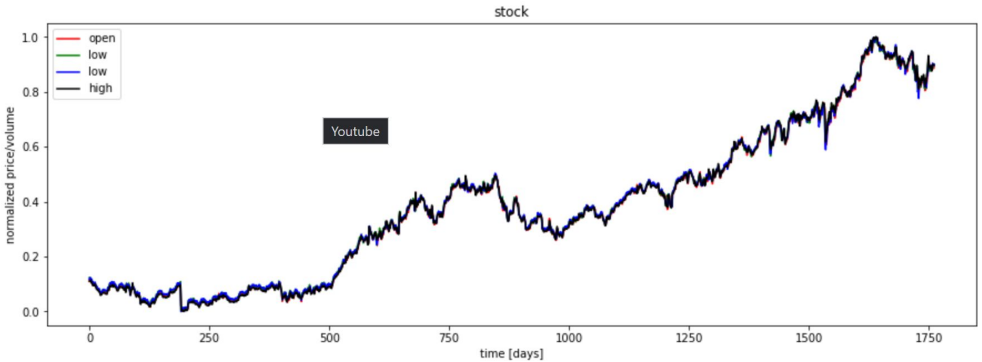
```
# function to create train, validation, test data given stock data and sequence length
def load_data(stock, seq_len):
    data_raw = stock.as_matrix() # convert to numpy array
    data = []

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

Data Manipulation (cont.)

```
# choose one stock
df_stock = df[df.symbol == 'EQIX'].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)
```

Here, we chose a stock, in this case “EQIX” and normalized it with the code to the left. Then, we created the training and testing data.

```
# 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)
```



This is what the plot turned out to be after manipulating the data. We can see a trend with the number of days increasing, the stock volume also increases.

Modeling and Validating Data

Now, we started to model and validate our data that we normalized. We set our parameters with the number of input, hidden, and output layers, the batch size, and the number of epochs used.

```
# 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]
```

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

We used our loss function as mean squared error for this project:

```
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)
```

Modeling and Validating Data (cont.)

With the model complete, we can start to run it and see what our accuracies are.

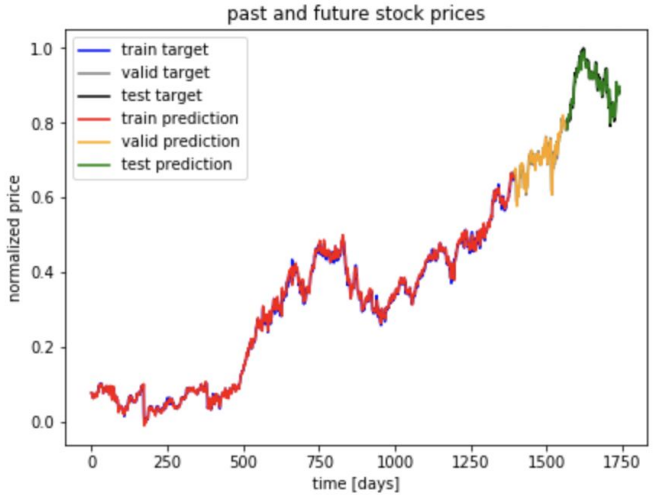
```
# run graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next training batch
        sess.run(training_op, feed_dict={X: x_batch, y: 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})
```

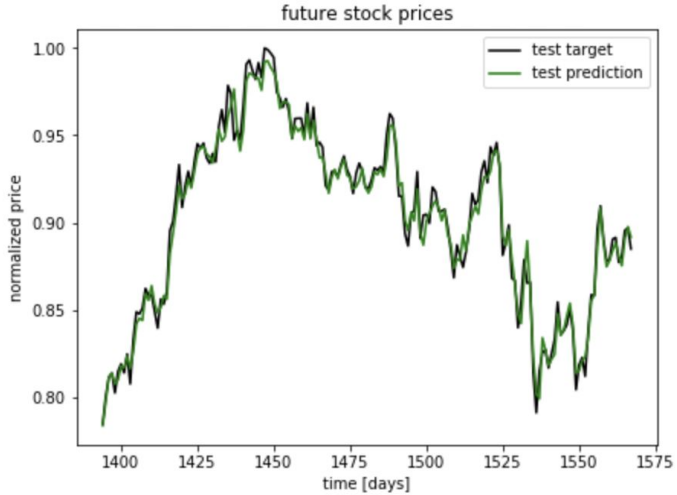
```
0.00 epochs: MSE train/valid = 0.125422/0.300375
4.99 epochs: MSE train/valid = 0.000243/0.001049
9.97 epochs: MSE train/valid = 0.000150/0.000744
14.96 epochs: MSE train/valid = 0.000136/0.000558
19.94 epochs: MSE train/valid = 0.000129/0.000432
24.93 epochs: MSE train/valid = 0.000114/0.000543
29.91 epochs: MSE train/valid = 0.000097/0.000431
34.90 epochs: MSE train/valid = 0.000100/0.000294
39.89 epochs: MSE train/valid = 0.000085/0.000304
44.87 epochs: MSE train/valid = 0.000079/0.000278
49.86 epochs: MSE train/valid = 0.000090/0.000395
54.84 epochs: MSE train/valid = 0.000077/0.000322
59.83 epochs: MSE train/valid = 0.000073/0.000234
64.81 epochs: MSE train/valid = 0.000090/0.000426
69.80 epochs: MSE train/valid = 0.000065/0.000251
74.78 epochs: MSE train/valid = 0.000081/0.000363
79.77 epochs: MSE train/valid = 0.000122/0.000311
84.76 epochs: MSE train/valid = 0.000076/0.000278
89.74 epochs: MSE train/valid = 0.000087/0.000380
94.73 epochs: MSE train/valid = 0.000072/0.000275
99.71 epochs: MSE train/valid = 0.000060/0.000211
```

Predictions

Finally, we can predict future stock prices with our model and plot the graphs to check out the results!

```
correct sign prediction for close - open price for train/valid/test: 0.73/0.47/0.45
```





Thanks for listening!