Preliminary LSTM on Google Stock Data; Time Series Forecasting

Walter Hennings

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import date
df = pd.read_csv(r'\Users\whwal\Desktop\Stock Data\Data\Stocks\goog.us.txt')
df['Date'] = pd.to_datetime(df['Date'])
df = df.dropna(axis = 0, how = 'any')
df.head()
```

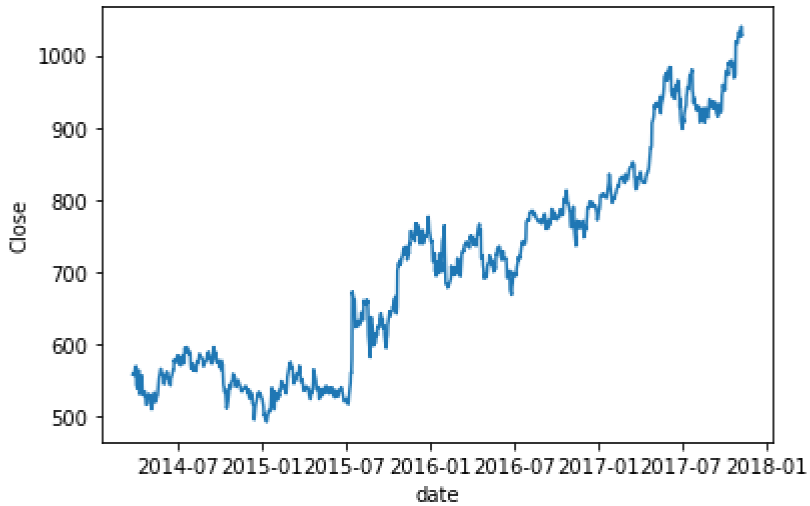
Out[2]:
```
(0, 915)
```

In [3]:
```
Open = df.Open
High = df.High
Low = df.Low
Close = df.Close
Volume = df.Volume
Date = df.Date
Date
```

Out[3]:
```
0      2014-03-27
1      2014-03-28
2      2014-03-31
3      2014-04-01
4      2014-04-02
          ...
911    2017-11-06
912    2017-11-07
913    2017-11-08
914    2017-11-09
915    2017-11-10
Name: Date, Length: 916, dtype: datetime64[ns]
```

In [4]:
```
plt.xlabel('date')
plt.ylabel('Close')
plt.plot(Date,Close)
```

Out[4]: [<matplotlib.lines.Line2D at 0x2170abc7cc8>]



In [5]:
```
start_date = pd.to_datetime(date(2014, 3, 27)) #Need to convert to datetime64 ns

# assuming your list is named my_date_list
differences = [d - start_date for d in Date] #This is our new feature
differences[1]
```

Out[5]: Timedelta('1 days 00:00:00')

In [6]:
```
#Recurrent neural network model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, BatchNormalization
from sklearn.model_selection import train_test_split
```

In [7]:
```
for i in range(len(differences)):
    differences[i] = differences[i].days
#Convert Timedelta to int
len(differences)
```

Out[7]: 916

In [8]:
```
#Need to difference the dataset:
def difference(x):
    diff = []
    for i in range(1,len(x)):
        value = x[i] - x[i-1]
        diff.append(value)
    return diff
Diffed_Close = difference(Close)
len(Diffed_Close)
```

Out[8]: 915

In [9]:
```
#Need to create a supervised learning problem by creating feature x-1 and target
def lag_transform(x):
    lagged = x[:len(x)-1]
    x = x[-(len(x)-1):]
    d = {'x-1': lagged,'x': x}
    df = pd.DataFrame(d)
    return df
Supervised_Close = lag_transform(Diffed_Close)
```

In [13]:
```
#Scale the data between [-1,1]
def scale_data(train, test,a,b):
    X = train
    minimum = []
    maximum = []
    for j in range(len(X.columns)):
        minimum.append(min(X.iloc[:,j]))
        maximum.append(max(X.iloc[:,j]))
    scaled_train = pd.DataFrame()
    scaled_train['x-1'] = np.zeros(len(X))
    scaled_train['x'] = np.zeros(len(X))
    scaled_test = pd.DataFrame()
    scaled_test['x-1'] = np.zeros(len(test))
    scaled_test['x'] = np.zeros(len(test))
    for i in range(len(X)):
        for j in range(len(X.columns)):
            scaled_train.iloc[i,j] = (b-a)*((X.iloc[i,j] - minimum[j])/(maximum[j
    for i in range(len(test)):
        for j in range(len(test.columns)):
            scaled_test.iloc[i,j] = (b-a)*((test.iloc[i,j] - minimum[j])/(maximum

    return scaled_train, scaled_test, minimum, maximum


from numpy import array
train = Supervised_Close[:round((.7)*len(Supervised_Close))]
test = Supervised_Close[round((.7)*len(Supervised_Close)):]
scaled_data = scale_data(train,test,-1,1)
scaled_X_train = array(scaled_data[0].iloc[:,0]).reshape(len(train),1,1)
scaled_X_test = array(scaled_data[1].iloc[:,0]).reshape(len(test),1,1)
scaled_y_train = array(scaled_data[0].iloc[:,1]).reshape(len(train),1,1)
scaled_y_test = array(scaled_data[1].iloc[:,1]).reshape(len(test),1,1)
```

In [ ]:

Define a LSTM Model

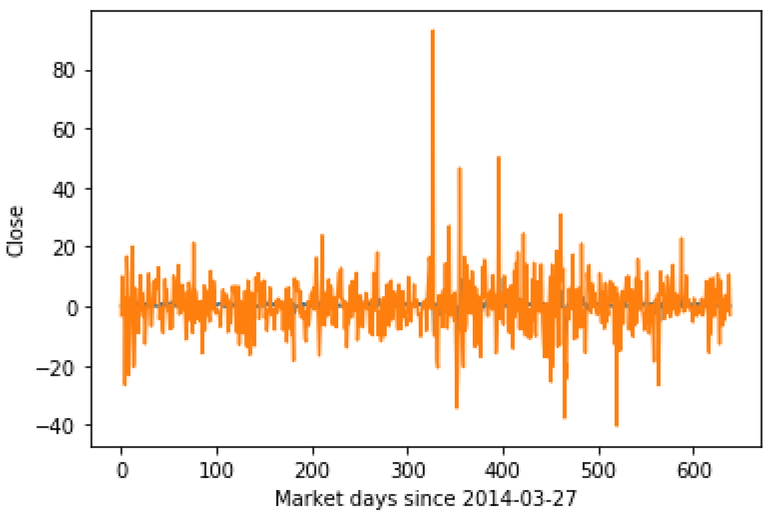
In [100]:
```
model = Sequential()
model.add(LSTM(128, input_shape = (scaled_X_train.shape[1:]),
               activation = 'relu',
               return_sequences = True))
model.add(Dropout(0.2))


model.add(LSTM(128, input_shape = (scaled_X_train.shape[1:]),
               activation = 'relu',
               return_sequences = True))
model.add(Dropout(0.2))


model.add(LSTM(128, input_shape = (scaled_X_train.shape[1:])))
model.add(Dropout(0.2))


model.add(Dense(32))

model.add(Dense(1, activation = 'linear'))

opt = tf.keras.optimizers.Adam(lr = .001, decay = 1e-6)

model.compile(loss = 'mse',
              optimizer = opt,
              metrics = ['mean_absolute_error'])




history = model.fit(
    scaled_X_train, scaled_y_train,
    batch_size = 685,
    epochs = 3000,
    validation_data = (scaled_X_test,scaled_y_test))
```

```
Epoch 2986/3000
640/640 [==============================] - 0s 73us/sample - loss: 0.0204 - me
an_absolute_error: 0.0965 - val_loss: 0.0185 - val_mean_absolute_error: 0.096
6
Epoch 2987/3000
640/640 [==============================] - 0s 76us/sample - loss: 0.0202 - me
an_absolute_error: 0.0964 - val_loss: 0.0185 - val_mean_absolute_error: 0.096
6
Epoch 2988/3000
640/640 [==============================] - 0s 76us/sample - loss: 0.0203 - me
an_absolute_error: 0.0966 - val_loss: 0.0185 - val_mean_absolute_error: 0.096
5
Epoch 2989/3000
640/640 [==============================] - 0s 70us/sample - loss: 0.0201 - me
an_absolute_error: 0.0962 - val_loss: 0.0185 - val_mean_absolute_error: 0.096
4
Epoch 2990/3000
640/640 [==============================] - 0s 75us/sample - loss: 0.0201 - me
an_absolute_error: 0.0961 - val_loss: 0.0185 - val_mean_absolute_error: 0.096
4
```

In [111]:
```
y_predict_train = model.predict(scaled_X_train)
y_predict_test = model.predict(scaled_X_test)
#predicted = y_predict_train + y_predict_test
def unscale(y,featurerange):
    minim = min(scaled_data[2])
    maxim = max(scaled_data[3])
    featuremin = featurerange[0]
    featuremax = featurerange[1]
    inverted_scaled = np.zeros(len(y))
    for i in range(len(y)):
        X = (y[i] - featuremin)/(featuremax - featuremin)
        inverted_scaled[i] = X*(maxim - minim) + minim
    return inverted_scaled
```

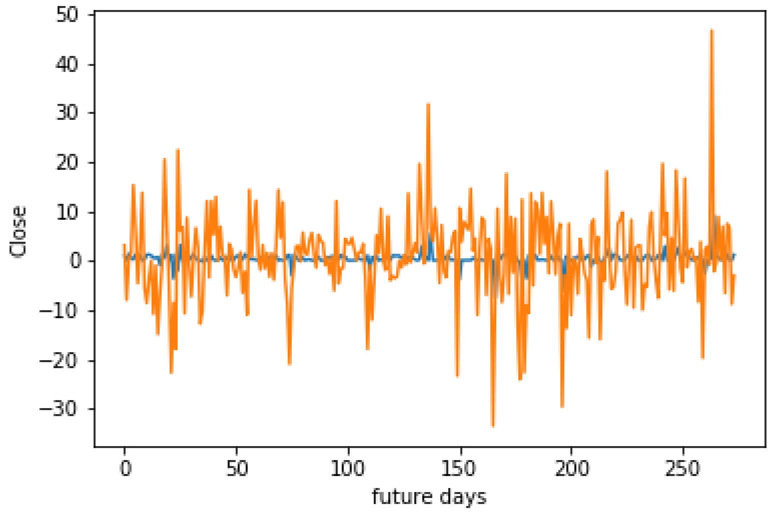
In [109]:
```
plt.xlabel('Market days since 2014-03-27')
plt.ylabel('Close')
plt.plot(unscale(y_predict_train,[-1,1]))
plt.plot(unscale(scaled_y_train.reshape(len(scaled_y_train)),[-1,1]))
plt.show
```

Out[109]: <function matplotlib.pyplot.show(*args, **kw)>



In [110]:
```
plt.xlabel('future days')
plt.ylabel('Close')
plt.plot(unscale(y_predict_test,[-1,1]))
plt.plot(unscale(scaled_y_test.reshape(len(scaled_y_test)),[-1,1]))
plt.show
```

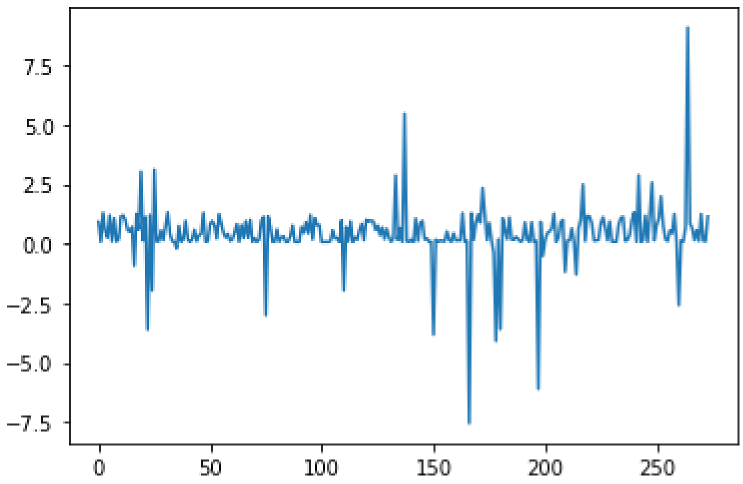
Out[110]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



Training on 3000 epochs gives a pretty good model considering there is only one feature. The model (blue) and is able to predict spikes on the test set (orance), indicating that there is some level of predictability on Google's stock given the training data. A better model can be made with multiple features for each day, such as supply/demand, etc.

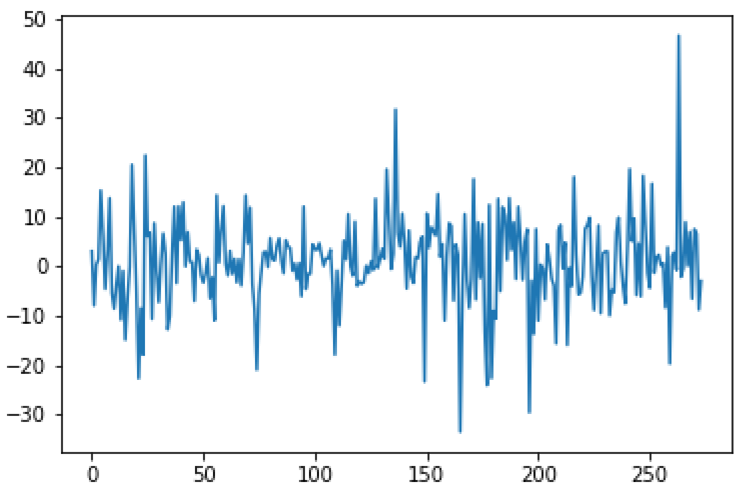
In [112]:
```
plt.plot(unscale(y_predict_test,[-1,1])) #The RNN Model:
```

Out[112]:
```
[<matplotlib.lines.Line2D at 0x21756f9f3c8>]
```



In [113]:
```
plt.plot(unscale(scaled_y_test.reshape(len(scaled_y_test)),[-1,1])) #The actual D
```

Out[113]:
```
[<matplotlib.lines.Line2D at 0x21756fedb88>]
```



Looking at these two models separately shows the discrepancy in the scale of the model results compared to the actual data, but spikes in the change in closing price are picked up anyways.

In [ ]:

In [ ]: