Stock price prediction

December 9, 2019

1 Recurrent Neural Network to Predict Amazon Stock Market Price

Data: Historical daily share price chart and data for Amazon since 1997 adjusted for splits. The latest closing stock price for Amazon as of November 13, 2019 is 1753.11.

This data was accessed from macrotrends throught he following link: https://www.macrotrends.net/stocks/charts/AMZN/amazon/stock-price-history

[1]:
```
import warnings
warnings.filterwarnings("ignore")

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
import sys

np.random.seed(42)
```

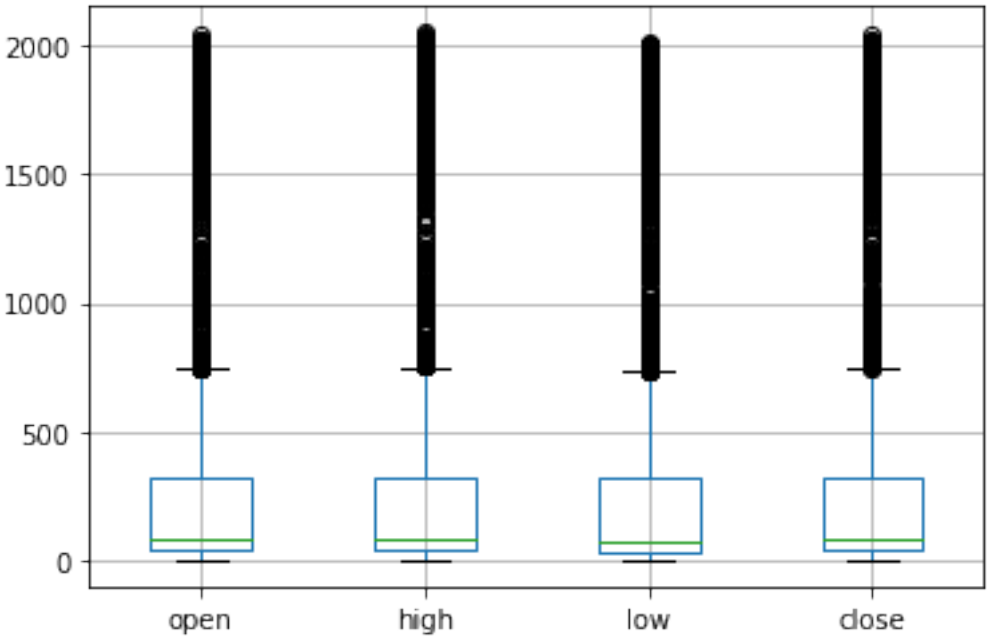
[2]:
```
dataset = pd.read_csv('MacroTrend_AMZN_Stock.csv', sep=',')
df_stock = dataset.copy()
df_stock.head()
```

[2]:

	date	open	high	low	close	volume
0	1997-05-16	1.8650	1.9792	1.7083	1.7292	14700000
1	1997-05-19	1.7083	1.7708	1.6250	1.7083	6106800
2	1997-05-20	1.7292	1.7500	1.6358	1.6358	5467200
3	1997-05-21	1.6042	1.6458	1.3750	1.4275	18853200
4	1997-05-22	1.4375	1.4483	1.3125	1.3958	11776800

[3]:
```
df_stock['DateTime'] = pd.to_datetime(df_stock['date'])
df_stock = df_stock[['DateTime','open','high','low','close','volume']]
```

[4]:
```
df_prices = df_stock[['open','high','low','close']]
boxplot = df_prices.boxplot(column=['open','high','low','close'])
```



[5]:
```
# PLOTTING ALL INDICATORS IN ONE PLOT
plt.figure(figsize=(20,15))
plt.plot(df_stock['DateTime'], df_stock['open'], 'r', label = 'open')
plt.plot(df_stock['DateTime'], df_stock['high'], 'g', label = 'high')
plt.plot(df_stock['DateTime'], df_stock['low'], 'b', label = 'low')
plt.plot(df_stock['DateTime'], df_stock['close'], 'black', label = 'close')
plt.legend(loc = 'upper right')
plt.show()
```

This plot shows that there appears to be very little deviation between the 4 measures of stock price on any given day. As the close price is the meaasure that would be of most interest, this is the price that will be the target for prediction.

[6]:
```
#df_stock = df_stock[['open','high','low','close']]
df_close = df_stock[['close']].values
```

[7]:
```
#scaling data
min_max_scaler = sklearn.preprocessing.MinMaxScaler(feature_range=(0,1))
df_close = min_max_scaler.fit_transform(df_close)
#def normalise_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
#df_close_norm = df_close.copy()
#df_close_norm = normalise_data(df_close_norm)
```

[46]:
```
#split dataset into train, validation and test sets
val_set_size_percent = 10
test_set_size_percent = 10
seq_len = 20

def load_data(stock, seq_len):
    #data_raw = stock.as_matrix()
    data_raw = stock
    data = []
    for index in range(len(data_raw) - seq_len):
        data.append(data_raw[index: index + seq_len])
    data = np.array(data);
    val_set_size = int(np.round(val_set_size_percent/100*data.shape[0]));
    test_set_size = int(np.round(test_set_size_percent/100*data.shape[0]));
    train_set_size = data.shape[0] - (val_set_size + test_set_size);
    x_train = data[:train_set_size,:-1:]
    y_train = data[:train_set_size,-1,:]
    x_val = data[train_set_size:train_set_size+val_set_size,:-1,:]
    y_val = data[train_set_size:train_set_size+val_set_size,-1,:]
    x_test = data[train_set_size+val_set_size:,:-1,:]
    y_test = data[train_set_size+val_set_size:,-1,:]
    return [x_train, y_train, x_val, y_val, x_test, y_test]

x_train, y_train, x_val, y_val, x_test, y_test = load_data(df_close, seq_len)
print('x_train shape: ', x_train.shape)
print('y_train shape: ', y_train.shape)
print('x_val shape: ', x_val.shape)
print('y_val shape: ', y_val.shape)
print('x_test shape: ', x_test.shape)
print('y_test shape: ', y_test.shape)
```

```
x_train shape:  (4514, 19, 1)
y_train shape:  (4514, 1)
x_val shape:  (564, 19, 1)
y_val shape:  (564, 1)
x_test shape:  (564, 19, 1)
y_test shape:  (564, 1)
```

1.1 Building the model

1.2 In Tensorflow

[47]:
```
#parameters and placeholders
n_steps = seq_len-1
n_inputs = 1
n_neurons = 200
n_outputs = 1
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])
```

[48]:
```
#function to get next batch
index_in_epoch = 0;
perm_array = np.arange(x_train.shape[0])
np.random.shuffle(perm_array)

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(perm_array) #shuffle permutation array
        start = 0 #start next epoch
        index_in_epoch = batch_size
    end = index_in_epoch
    return x_train[perm_array[start:end]], y_train[perm_array[start:end]]
```

1.2.1 RNN

[11]:
```
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
```

```
WARNING: The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-11-d4d8488671e3>:2: BasicRNNCell.__init__
(from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed in
```

```
a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced
by that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-11-d4d8488671e3>:4: MultiRNNCell.__init__
(from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed in
a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.StackedRNNCells, and will be
replaced by that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-11-d4d8488671e3>:5: dynamic_rnn (from
tensorflow.python.ops.rnn) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
WARNING:tensorflow:From /opt/anaconda3/envs/py36/lib/python3.6/site-
packages/tensorflow/python/ops/tensor_array_ops.py:162: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From <ipython-input-11-d4d8488671e3>:7: dense (from
tensorflow.python.layers.core) is deprecated and will be removed in a future
version.
Instructions for updating:
Use keras.layers.dense instead.
```

[12]:
```
#cost function
loss = tf.reduce_mean(tf.square(outputs - Y))
```

[13]:
```
#optimiser
optimiser = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimiser.minimize(loss)
```

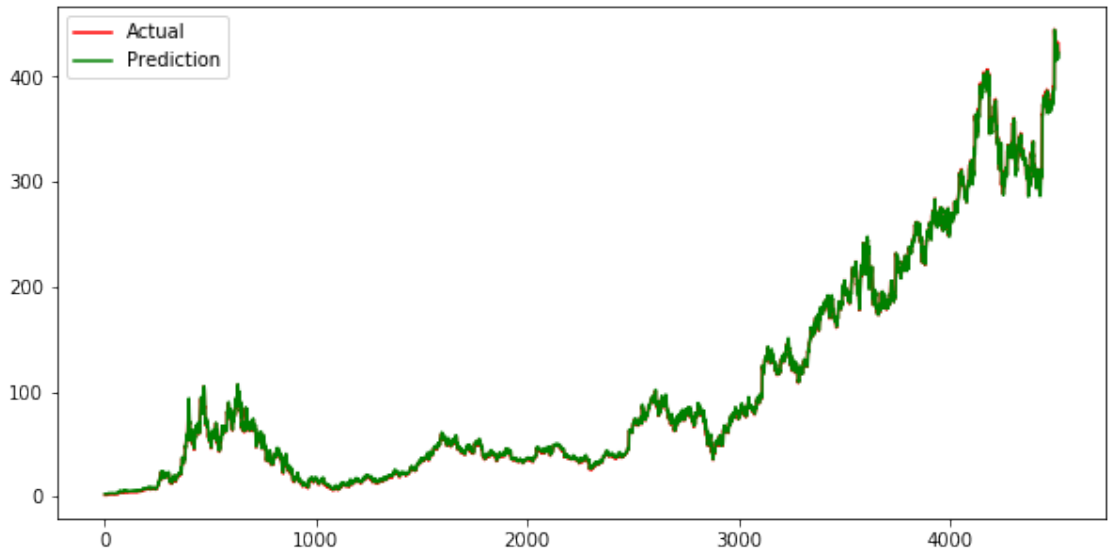
[14]:
```
#fitting the model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) #get next training batch
        sess.run(training_op, feed_dict={X: x_batch, 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_val, Y: y_val})
            print('%.0f epochs: MSE train/valid = %.6f/%.
 ↪6f'%(iteration*batch_size/train_set_size, mse_train, mse_valid))

#predictions
```

```
    y_train_pred_rnn = sess.run(outputs, feed_dict={X: x_train})
    y_test_pred_rnn = sess.run(outputs, feed_dict={X: x_test})
```

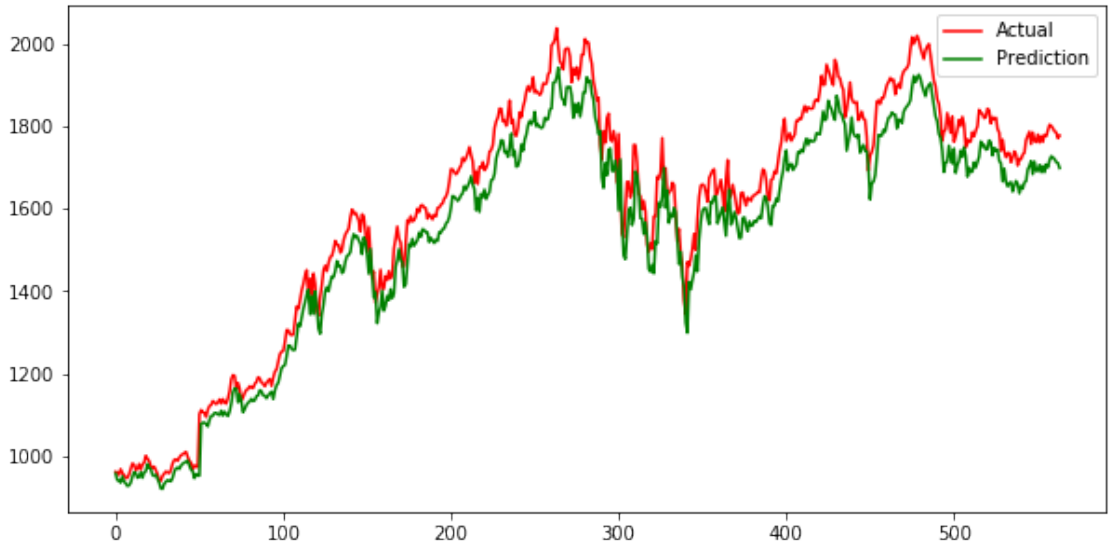
```
0 epochs: MSE train/valid = 0.000814/0.002936
5 epochs: MSE train/valid = 0.000011/0.000814
10 epochs: MSE train/valid = 0.000011/0.000133
15 epochs: MSE train/valid = 0.000006/0.000142
20 epochs: MSE train/valid = 0.000005/0.000081
25 epochs: MSE train/valid = 0.000004/0.000095
30 epochs: MSE train/valid = 0.000004/0.000080
35 epochs: MSE train/valid = 0.000006/0.000059
40 epochs: MSE train/valid = 0.000004/0.000105
45 epochs: MSE train/valid = 0.000003/0.000082
50 epochs: MSE train/valid = 0.000003/0.000033
55 epochs: MSE train/valid = 0.000003/0.000095
60 epochs: MSE train/valid = 0.000005/0.000049
65 epochs: MSE train/valid = 0.000004/0.000119
70 epochs: MSE train/valid = 0.000004/0.000043
75 epochs: MSE train/valid = 0.000003/0.000071
80 epochs: MSE train/valid = 0.000007/0.000140
85 epochs: MSE train/valid = 0.000004/0.000035
90 epochs: MSE train/valid = 0.000003/0.000116
95 epochs: MSE train/valid = 0.000004/0.000060
100 epochs: MSE train/valid = 0.000003/0.000107
```

[15]:
```
#invert predictions
y_train_pred_rnn = min_max_scaler.inverse_transform(y_train_pred_rnn)
y_train_inv = min_max_scaler.inverse_transform(y_train)

y_test_pred_rnn = min_max_scaler.inverse_transform(y_test_pred_rnn)
y_test_inv = min_max_scaler.inverse_transform(y_test)
```

[17]:
```
#plotting
actual_pred_train_rnn = pd.DataFrame({'Actual':y_train_inv[:,0], 'Predicted':
 ↪y_train_pred_rnn[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_train_rnn['Actual'], color='r', label='Actual')
plt.plot(actual_pred_train_rnn['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[18]:
```
#plotting
actual_pred_test_rnn = pd.DataFrame({'Actual':y_test_inv[:,0], 'Predicted':
 ↪y_test_pred_rnn[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_test_rnn['Actual'], color='r', label='Actual')
plt.plot(actual_pred_test_rnn['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[19]:
```
# calculate root mean squared error
from sklearn.metrics import mean_squared_error
train_score_rnn = np.sqrt(mean_squared_error(y_train_inv[:,0],␣
 ↪y_train_pred_rnn[:,0]))
print('Test Score: %.2f RMSE' % (train_score_rnn))

test_score_rnn = np.sqrt(mean_squared_error(y_test_inv[:,0], y_test_pred_rnn[:
 ↪,0]))
print('Test Score: %.2f RMSE' % (test_score_rnn))
```

```
Test Score: 3.50 RMSE
Test Score: 72.04 RMSE
```

1.2.2 LSTM

[20]:
```
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.
 ↪elu)
        for layer in range(n_layers)]
```

```
WARNING:tensorflow:From <ipython-input-20-680c4573e2ca>:2:
BasicLSTMCell.__init__ (from tensorflow.python.ops.rnn_cell_impl) is deprecated
and will be removed in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.LSTMCell, and will be replaced by
that in Tensorflow 2.0.
```

[21]:
```
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
```

[22]:
```
#cost function
loss = tf.reduce_mean(tf.square(outputs - Y))
```

[23]:
```
#optimiser
optimiser = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimiser.minimize(loss)
```

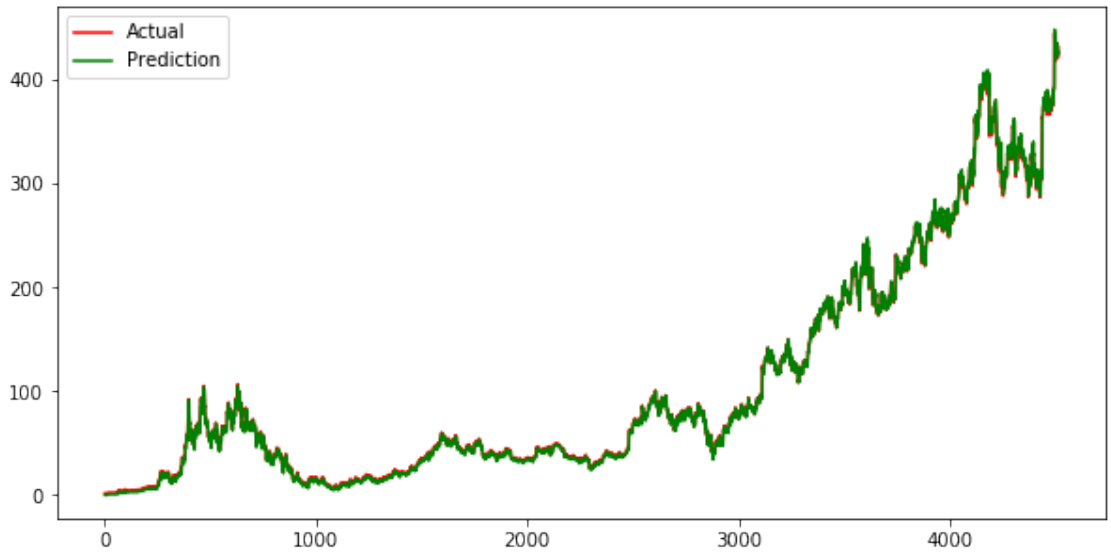
[25]:
```
#fitting the model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) #get next training batch
        sess.run(training_op, feed_dict={X: x_batch, 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_val, Y: y_val})
            print('%.0f epochs: MSE train/valid = %.6f/%.
 ↪6f'%(iteration*batch_size/train_set_size, mse_train, mse_valid))

#predictions
    y_train_pred_lstm = sess.run(outputs, feed_dict={X: x_train})
    y_test_pred_lstm = sess.run(outputs, feed_dict={X: x_test})
```

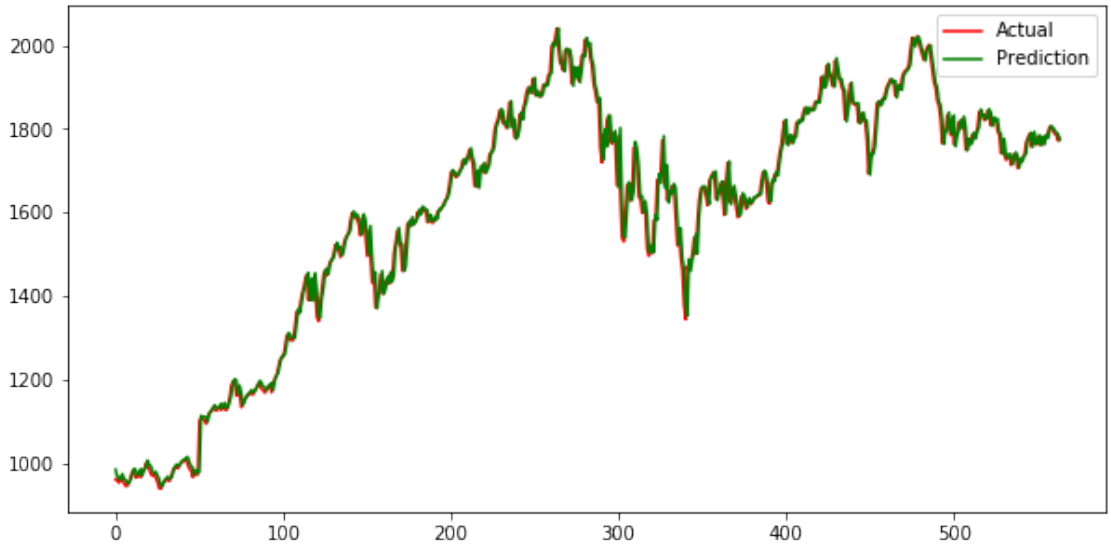
```
0 epochs: MSE train/valid = 0.001883/0.068391
5 epochs: MSE train/valid = 0.000011/0.000102
10 epochs: MSE train/valid = 0.000007/0.000103
15 epochs: MSE train/valid = 0.000006/0.000061
20 epochs: MSE train/valid = 0.000005/0.000058
25 epochs: MSE train/valid = 0.000007/0.000118
30 epochs: MSE train/valid = 0.000004/0.000040
35 epochs: MSE train/valid = 0.000005/0.000073
40 epochs: MSE train/valid = 0.000004/0.000045
45 epochs: MSE train/valid = 0.000008/0.000042
50 epochs: MSE train/valid = 0.000004/0.000030
55 epochs: MSE train/valid = 0.000005/0.000041
60 epochs: MSE train/valid = 0.000003/0.000033
65 epochs: MSE train/valid = 0.000003/0.000035
70 epochs: MSE train/valid = 0.000004/0.000037
75 epochs: MSE train/valid = 0.000003/0.000030
80 epochs: MSE train/valid = 0.000005/0.000029
85 epochs: MSE train/valid = 0.000003/0.000030
90 epochs: MSE train/valid = 0.000003/0.000032
95 epochs: MSE train/valid = 0.000003/0.000056
100 epochs: MSE train/valid = 0.000004/0.000052
```

[26]:
```
#invert predictions
y_train_pred_lstm = min_max_scaler.inverse_transform(y_train_pred_lstm)
y_train_inv = min_max_scaler.inverse_transform(y_train)

y_test_pred_lstm = min_max_scaler.inverse_transform(y_test_pred_lstm)
y_test_inv = min_max_scaler.inverse_transform(y_test)
```

[27]:
```
#plotting train lstm
actual_pred_lstm_train = pd.DataFrame({'Actual':y_train_inv[:,0], 'Predicted':
 ↪y_train_pred_lstm[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_lstm_train['Actual'], color='r', label='Actual')
plt.plot(actual_pred_lstm_train['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[28]:
```
#plotting test lstm
actual_pred_lstm_test = pd.DataFrame({'Actual':y_test_inv[:,0], 'Predicted':
 ↪y_test_pred_lstm[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_lstm_test['Actual'], color='r', label='Actual')
plt.plot(actual_pred_lstm_test['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[29]:
```
train_score_lstm = np.sqrt(mean_squared_error(y_train_inv[:,0],␣
↪y_train_pred_lstm[:,0]))
print('Test Score: %.2f RMSE' % (train_score_lstm))

test_score_lstm = np.sqrt(mean_squared_error(y_test_inv[:,0], y_test_pred_lstm[:
↪,0]))
print('Test Score: %.2f RMSE' % (test_score_lstm))
```

```
Test Score: 3.65 RMSE
Test Score: 30.18 RMSE
```

1.2.3 LSTM with Peephole

[49]:
```
layers = [tf.contrib.rnn.LSTMCell(num_units=n_neurons, activation=tf.nn.
↪leaky_relu, use_peepholes=True)
        for layer in range(n_layers)]
```

[50]:
```
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
```

[51]:
```
#cost function
loss = tf.reduce_mean(tf.square(outputs - Y))
```

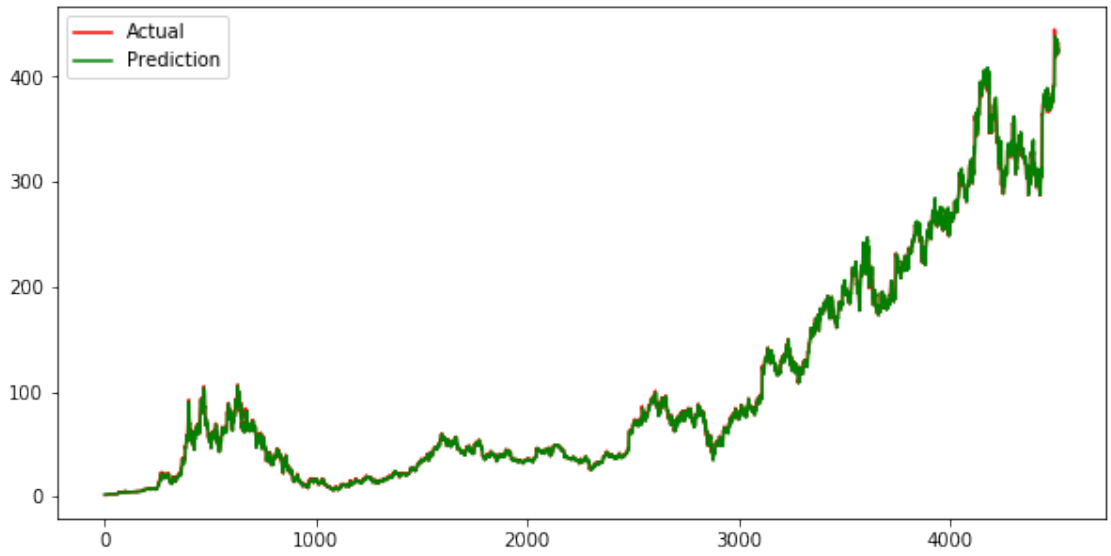
[52]:
```
#optimiser
optimiser = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimiser.minimize(loss)
```

[53]:
```
#fitting the 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) #get 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_val, Y: y_val})
            print('%.0f epochs: MSE train/valid = %.6f/%.
↪6f'%(iteration*batch_size/train_set_size, mse_train, mse_valid))

#predictions
    y_train_pred_peep = sess.run(outputs, feed_dict={X: x_train})
    y_test_pred_peep = sess.run(outputs, feed_dict={X: x_test})
```

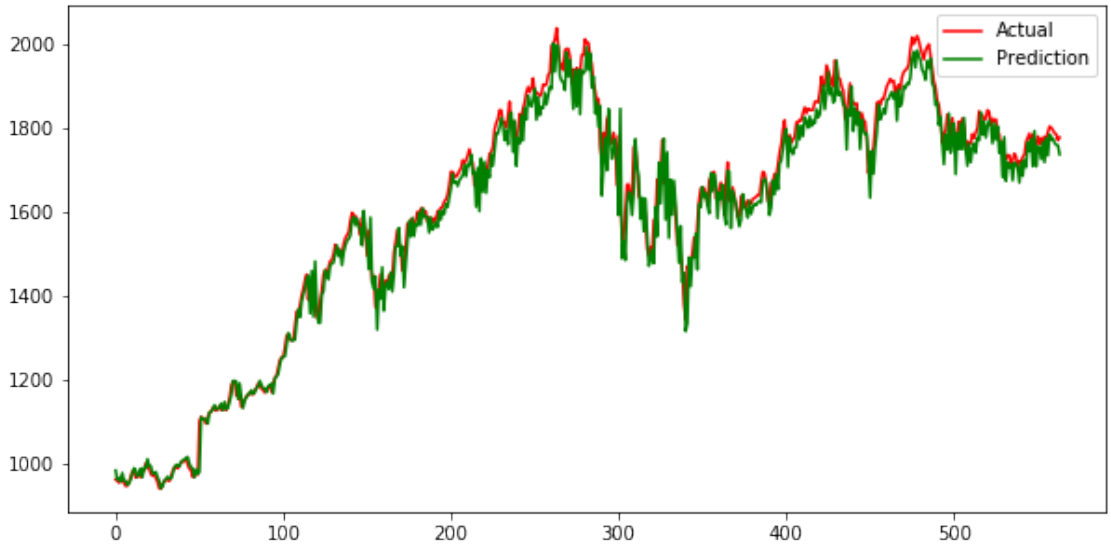
```
0 epochs: MSE train/valid = 0.003283/0.098770
5 epochs: MSE train/valid = 0.000014/0.000161
10 epochs: MSE train/valid = 0.000009/0.000094
15 epochs: MSE train/valid = 0.000007/0.000076
20 epochs: MSE train/valid = 0.000007/0.000073
25 epochs: MSE train/valid = 0.000009/0.000090
30 epochs: MSE train/valid = 0.000005/0.000085
35 epochs: MSE train/valid = 0.000005/0.000088
40 epochs: MSE train/valid = 0.000004/0.000123
45 epochs: MSE train/valid = 0.000003/0.000090
50 epochs: MSE train/valid = 0.000005/0.000051
55 epochs: MSE train/valid = 0.000004/0.000093
60 epochs: MSE train/valid = 0.000004/0.000033
65 epochs: MSE train/valid = 0.000003/0.000031
70 epochs: MSE train/valid = 0.000003/0.000056
75 epochs: MSE train/valid = 0.000003/0.000034
80 epochs: MSE train/valid = 0.000003/0.000050
85 epochs: MSE train/valid = 0.000003/0.000031
90 epochs: MSE train/valid = 0.000005/0.000059
95 epochs: MSE train/valid = 0.000003/0.000034
100 epochs: MSE train/valid = 0.000004/0.000035
```

[54]:
```
#invert
y_train_pred_peep = min_max_scaler.inverse_transform(y_train_pred_peep)
y_train_inv = min_max_scaler.inverse_transform(y_train)

y_test_pred_peep = min_max_scaler.inverse_transform(y_test_pred_peep)
y_test_inv = min_max_scaler.inverse_transform(y_test)
```

[55]:
```
#plotting train peephole
actual_pred_train_peep = pd.DataFrame({'Actual':y_train_inv[:,0], 'Predicted':
 ↪y_train_pred_peep[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_train_peep['Actual'], color='r', label='Actual')
plt.plot(actual_pred_train_peep['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[56]:
```
#plotting test peephole
actual_pred_test_peep = pd.DataFrame({'Actual':y_test_inv[:,0], 'Predicted':
 ↪y_test_pred_peep[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_test_peep['Actual'], color='r', label='Actual')
plt.plot(actual_pred_test_peep['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[57]:
```
train_score_peep = np.sqrt(mean_squared_error(y_train_inv[:,0],␣
↪y_train_pred_peep[:,0]))
print('Test Score: %.2f RMSE' % (train_score_peep))

test_score_peep = np.sqrt(mean_squared_error(y_test_inv[:,0], y_test_pred_peep[:
↪,0]))
print('Test Score: %.2f RMSE' % (test_score_peep))
```

```
Test Score: 3.46 RMSE
Test Score: 42.58 RMSE
```

1.2.4 GRU

[58]:
```
layers = [tf.contrib.rnn.GRUCell(num_units=n_neurons, activation=tf.nn.
↪leaky_relu)
        for layer in range(n_layers)]
```

```
WARNING:tensorflow:From <ipython-input-58-63d1b18b5fdb>:2: GRUCell.__init__
(from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed in
a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.GRUCell, and will be replaced by
that in Tensorflow 2.0.
```

[59]:
```
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
```

[60]:
```
#cost function
loss = tf.reduce_mean(tf.square(outputs - Y))
```

[61]:
```
#optimiser
optimiser = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimiser.minimize(loss)
```

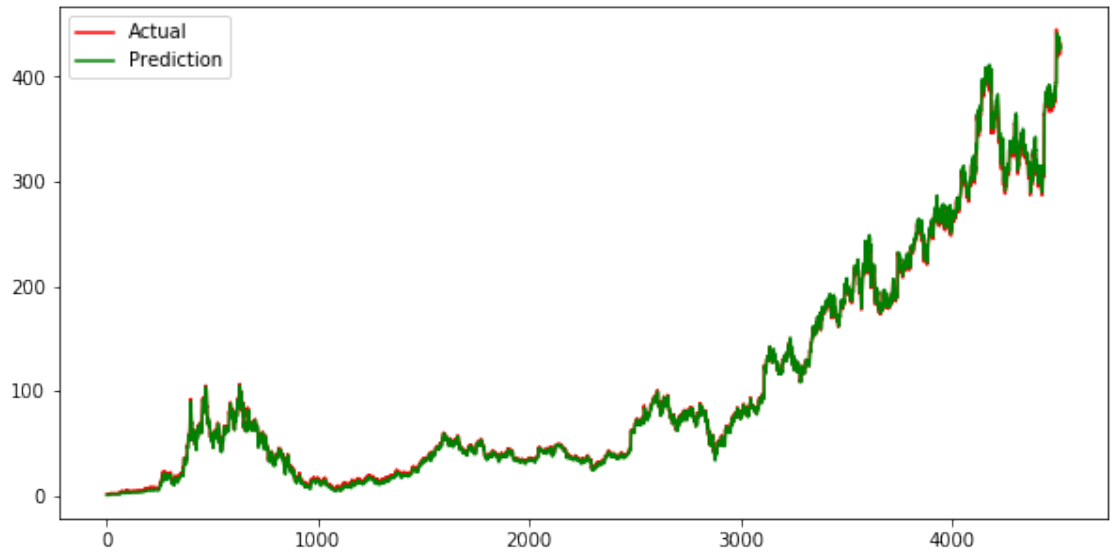
[62]:
```
#fitting the model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) #get next training batch
        sess.run(training_op, feed_dict={X: x_batch, 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_val, Y: y_val})
```

```
            print('%.0f epochs: MSE train/valid = %.6f/%.
 ↪6f'%(iteration*batch_size/train_set_size, mse_train, mse_valid))

#predictions
    y_train_pred_gru = sess.run(outputs, feed_dict={X: x_train})
    y_test_pred_gru = sess.run(outputs, feed_dict={X: x_test})
```

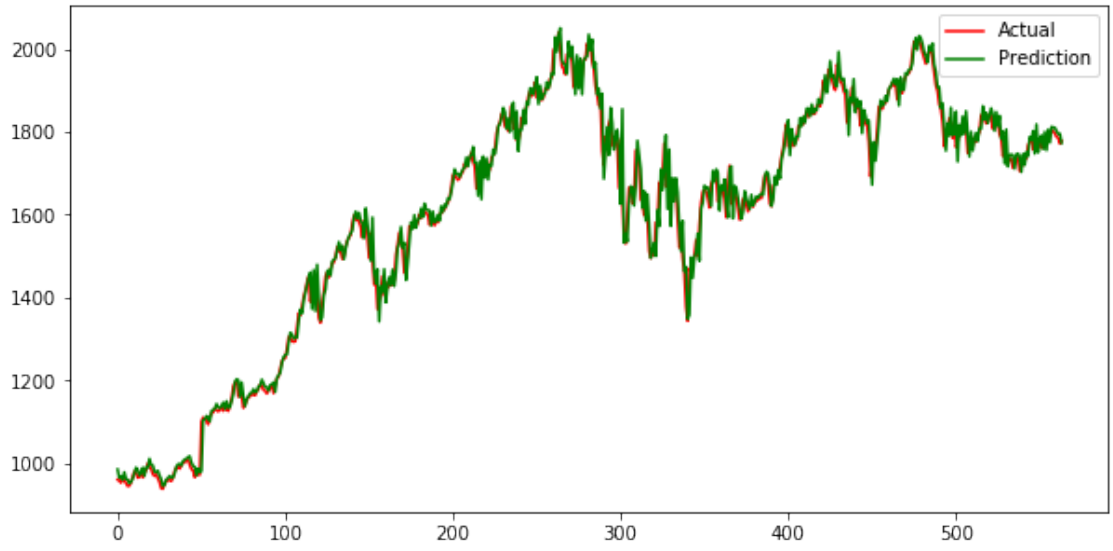
```
0 epochs: MSE train/valid = 0.003082/0.093872
5 epochs: MSE train/valid = 0.000013/0.000197
10 epochs: MSE train/valid = 0.000011/0.000147
15 epochs: MSE train/valid = 0.000007/0.000167
20 epochs: MSE train/valid = 0.000006/0.000069
25 epochs: MSE train/valid = 0.000006/0.000057
30 epochs: MSE train/valid = 0.000003/0.000041
35 epochs: MSE train/valid = 0.000007/0.000116
40 epochs: MSE train/valid = 0.000003/0.000074
45 epochs: MSE train/valid = 0.000003/0.000029
50 epochs: MSE train/valid = 0.000005/0.000041
55 epochs: MSE train/valid = 0.000003/0.000066
60 epochs: MSE train/valid = 0.000003/0.000030
65 epochs: MSE train/valid = 0.000005/0.000072
70 epochs: MSE train/valid = 0.000004/0.000040
75 epochs: MSE train/valid = 0.000003/0.000040
80 epochs: MSE train/valid = 0.000003/0.000052
85 epochs: MSE train/valid = 0.000003/0.000041
90 epochs: MSE train/valid = 0.000006/0.000042
95 epochs: MSE train/valid = 0.000003/0.000036
100 epochs: MSE train/valid = 0.000004/0.000055
```

[63]:
```
y_train_pred_gru = min_max_scaler.inverse_transform(y_train_pred_gru)
y_train_inv = min_max_scaler.inverse_transform(y_train)

y_test_pred_gru = min_max_scaler.inverse_transform(y_test_pred_gru)
y_test_inv = min_max_scaler.inverse_transform(y_test)
```

[64]:
```
#plotting train gru
actual_pred_train_gru = pd.DataFrame({'Actual':y_train_inv[:,0], 'Predicted':
 ↪y_train_pred_gru[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_train_gru['Actual'], color='r', label='Actual')
plt.plot(actual_pred_train_gru['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[65]:
```
#plotting
actual_pred_test_gru = pd.DataFrame({'Actual':y_test_inv[:,0], 'Predicted':
 ↪y_test_pred_gru[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_test_gru['Actual'], color='r', label='Actual')
plt.plot(actual_pred_test_gru['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[66]:
```
train_score_gru = np.sqrt(mean_squared_error(y_train_inv[:,0],␣
↪y_train_pred_gru[:,0]))
print('Test Score: %.2f RMSE' % (train_score_gru))

test_score_gru = np.sqrt(mean_squared_error(y_test_inv[:,0], y_test_pred_gru[:
↪,0]))
print('Test Score: %.2f RMSE' % (test_score_gru))
```

```
Test Score: 3.86 RMSE
Test Score: 34.26 RMSE
```

1.2.5 Summary of performance on test set

[67]:
```
print('RNN test Score: %.2f RMSE' % (test_score_rnn))
print('LSTM test Score: %.2f RMSE' % (test_score_lstm))
print('LSTM with peephole test Score: %.2f RMSE' % (test_score_peep))
print('GRU test Score: %.2f RMSE' % (test_score_gru))
```

```
RNN test Score: 72.04 RMSE
LSTM test Score: 30.18 RMSE
LSTM with peephole test Score: 42.58 RMSE
GRU test Score: 34.26 RMSE
```

The best-performing version (lowest RMSE) is LSTM with an RMSE score of 30.18.

A forecasting model using LSTM networks in Keras will now be built to forecast stocks.

2 In Keras

[167]:
```
import pandas as pd
import numpy as np
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasRegressor
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras import optimizers
```

[186]:
```
model_k = Sequential()
model_k.add(LSTM(100,activation='relu',input_shape=(19,1)))
model_k.add(Dense(20, activation='relu'))
model_k.add(Dense(1))

optimizer = optimizers.RMSprop(lr=0.001)
model_k.compile(optimizer=optimizer, loss='mse')
```

```
num_epochs = 25
model_k.fit(x_train,y_train, epochs=num_epochs, validation_data=(x_val,y_val),␣
 ↪verbose=1)
```

```
Train on 4514 samples, validate on 564 samples
Epoch 1/25
4514/4514 [==============================] - 3s 735us/step - loss: 2.8116e-04 -
val_loss: 0.0194
Epoch 2/25
4514/4514 [==============================] - 2s 375us/step - loss: 7.4659e-05 -
val_loss: 3.8102e-04
Epoch 3/25
4514/4514 [==============================] - 2s 379us/step - loss: 5.6497e-05 -
val_loss: 3.2370e-04
Epoch 4/25
4514/4514 [==============================] - 2s 395us/step - loss: 4.9694e-05 -
val_loss: 0.0021
Epoch 5/25
4514/4514 [==============================] - 2s 391us/step - loss: 4.4652e-05 -
val_loss: 4.4645e-04
Epoch 6/25
4514/4514 [==============================] - 2s 378us/step - loss: 4.2164e-05 -
val_loss: 6.0300e-04
Epoch 7/25
4514/4514 [==============================] - 2s 371us/step - loss: 3.6827e-05 -
val_loss: 0.0061
Epoch 8/25
4514/4514 [==============================] - 2s 376us/step - loss: 3.3188e-05 -
val_loss: 2.0073e-04
Epoch 9/25
4514/4514 [==============================] - 2s 377us/step - loss: 3.2119e-05 -
val_loss: 1.8068e-04
Epoch 10/25
4514/4514 [==============================] - 2s 392us/step - loss: 2.8263e-05 -
val_loss: 1.7148e-04
Epoch 11/25
4514/4514 [==============================] - 2s 389us/step - loss: 2.6918e-05 -
val_loss: 3.2547e-04
Epoch 12/25
4514/4514 [==============================] - 2s 371us/step - loss: 2.4202e-05 -
val_loss: 9.4611e-05
Epoch 13/25
4514/4514 [==============================] - 2s 377us/step - loss: 2.2939e-05 -
val_loss: 3.1974e-04
Epoch 14/25
4514/4514 [==============================] - 2s 386us/step - loss: 2.1958e-05 -
val_loss: 1.8307e-04
```

```
Epoch 15/25
4514/4514 [==============================] - 2s 383us/step - loss: 2.0638e-05 -
val_loss: 9.8612e-05
Epoch 16/25
4514/4514 [==============================] - 2s 379us/step - loss: 1.8795e-05 -
val_loss: 1.7550e-04
Epoch 17/25
4514/4514 [==============================] - 2s 374us/step - loss: 1.9180e-05 -
val_loss: 1.7042e-04
Epoch 18/25
4514/4514 [==============================] - 2s 379us/step - loss: 1.7188e-05 -
val_loss: 0.0017
Epoch 19/25
4514/4514 [==============================] - 2s 377us/step - loss: 1.7551e-05 -
val_loss: 3.7990e-04
Epoch 20/25
4514/4514 [==============================] - 2s 385us/step - loss: 1.7742e-05 -
val_loss: 6.1827e-05
Epoch 21/25
4514/4514 [==============================] - 2s 375us/step - loss: 1.6159e-05 -
val_loss: 4.0958e-04
Epoch 22/25
4514/4514 [==============================] - 2s 384us/step - loss: 1.5697e-05 -
val_loss: 2.0467e-04
Epoch 23/25
4514/4514 [==============================] - 2s 373us/step - loss: 1.4288e-05 -
val_loss: 1.8881e-04
Epoch 24/25
4514/4514 [==============================] - 2s 378us/step - loss: 1.4261e-05 -
val_loss: 8.6992e-05
Epoch 25/25
4514/4514 [==============================] - 2s 378us/step - loss: 1.4277e-05 -
val_loss: 9.6567e-05
```

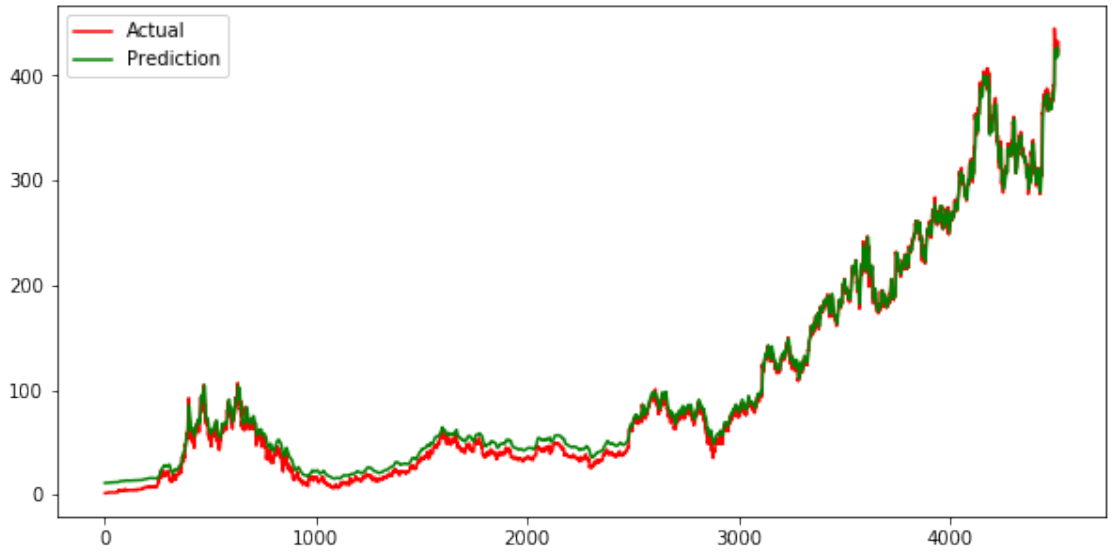
[186]:
```
<keras.callbacks.callbacks.History at 0x7fdfb3fdeef0>
```

[187]:
```
y_train_pred_keras = model_k.predict(x_train)
y_test_pred_keras = model_k.predict(x_test)
```

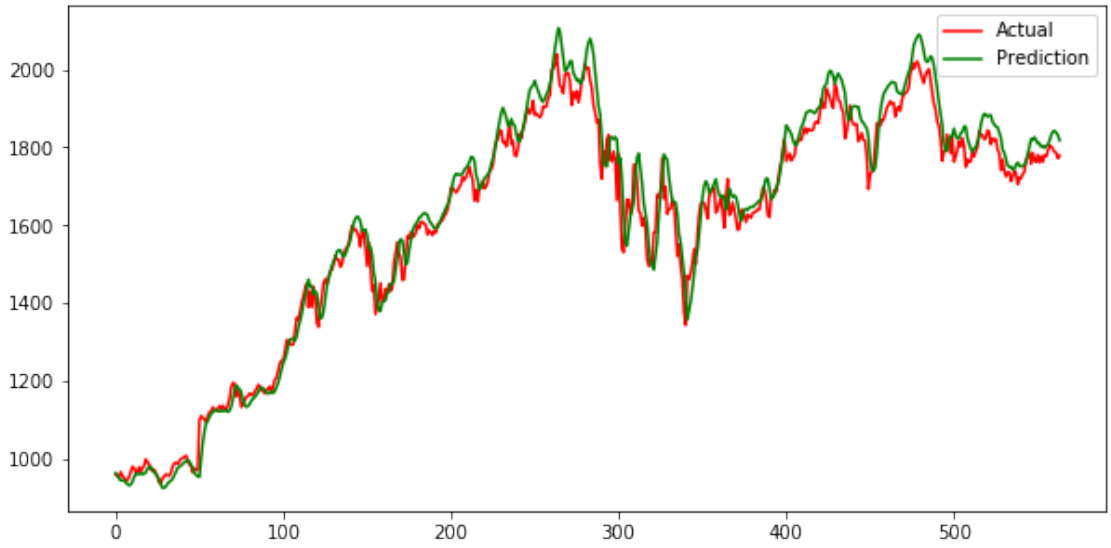
[188]:
```
y_train_pred_keras = min_max_scaler.inverse_transform(y_train_pred_keras)
y_train_inv = min_max_scaler.inverse_transform(y_train)

y_test_pred_keras = min_max_scaler.inverse_transform(y_test_pred_keras)
y_test_inv = min_max_scaler.inverse_transform(y_test)
```

[189]:
```
#plotting train keras
```

```
actual_pred_train_keras = pd.DataFrame({'Actual':y_train_inv[:,0], 'Predicted':
 ↪y_train_pred_keras[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_train_keras['Actual'], color='r', label='Actual')
plt.plot(actual_pred_train_keras['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[190]:
```
#plotting train keras
actual_pred_test_keras = pd.DataFrame({'Actual':y_test_inv[:,0], 'Predicted':
 ↪y_test_pred_keras[:,0]})
plt.figure(figsize=(10,5))
plt.plot(actual_pred_test_keras['Actual'], color='r', label='Actual')
plt.plot(actual_pred_test_keras['Predicted'], color='g', label='Prediction')
plt.legend()
plt.show()
```



[191]:
```
train_score_keras = np.sqrt(mean_squared_error(y_train_inv[:,0],␣
 ↪y_train_pred_keras[:,0]))
print('Test Score: %.2f RMSE' % (train_score_keras))

test_score_keras = np.sqrt(mean_squared_error(y_test_inv[:,0],␣
 ↪y_test_pred_keras[:,0]))
print('Test Score: %.2f RMSE' % (test_score_keras))
```

```
Test Score: 7.51 RMSE
Test Score: 48.47 RMSE
```

Improvement of the Keras model may be achieved with hyperparameter tuning.