In [1]:
```
import datetime
import pandas as pd
import pandas_datareader.data as web
```

In [3]:
```
start = datetime.datetime(2004, 8, 19) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('GOOG', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                Open       High        Low      Close  Adj Close    Volume
Date
2004-08-19  49.676899  51.693783  47.669952  49.845802  49.845802  44994500
2004-08-20  50.178635  54.187561  49.925285  53.805050  53.805050  23005800
2004-08-23  55.017166  56.373344  54.172661  54.346527  54.346527  18393200
2004-08-24  55.260582  55.439419  51.450363  52.096165  52.096165  15361800
2004-08-25  52.140873  53.651051  51.604362  52.657513  52.657513   9257400
```

In [4]:
```
import time
import math
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
import numpy as np
import pandas as pd
import tensorflow as tf
import sklearn.preprocessing as prep
from keras import backend
```

```
Using TensorFlow backend.
```

In [5]:
```
import os
s=os.getcwd()
s
```

Out[5]:
```
'/Users/Yuffie/USA/SCU/COEN281DataMining/TermProject/GOOG'
```

In [6]:
```
df = pd.read_csv('/Users/Yuffie/USA/SCU/COEN281DataMining/TermProject/data/GOOG.csv')
df.head()
```

Out[6]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	49.676899	51.693783	47.669952	49.845802	49.845802	44994500
1	2004-08-20	50.178635	54.187561	49.925285	53.805050	53.805050	23005800
2	2004-08-23	55.017166	56.373344	54.172661	54.346527	54.346527	18393200
3	2004-08-24	55.260582	55.439419	51.450363	52.096165	52.096165	15361800
4	2004-08-25	52.140873	53.651051	51.604362	52.657513	52.657513	9257400

In [7]:
```
# Data preparation
col_list = df.columns.tolist()
col_list
```

Out[7]:
```
['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
```

In [8]:
```
col_list.remove('Close')
col_list.append('Close')
col_list.remove('Date')
col_list.remove('Volume')
col_list
```

Out[8]:
```
['Open', 'High', 'Low', 'Adj Close', 'Close']
```

In [9]:
```
df = df[col_list]
df.head()
```

Out[9]:

	Open	High	Low	Adj Close	Close
0	49.676899	51.693783	47.669952	49.845802	49.845802
1	50.178635	54.187561	49.925285	53.805050	53.805050
2	55.017166	56.373344	54.172661	54.346527	54.346527
3	55.260582	55.439419	51.450363	52.096165	52.096165
4	52.140873	53.651051	51.604362	52.657513	52.657513

In [10]:
```
# Save data
df.to_csv('GOOG-adjust.csv', index=False)
validate_df = pd.read_csv('GOOG-adjust.csv')
validate_df.head()
```

Out[10]:

	Open	High	Low	Adj Close	Close
0	49.676899	51.693783	47.669952	49.845802	49.845802
1	50.178635	54.187561	49.925285	53.805050	53.805050
2	55.017166	56.373344	54.172661	54.346527	54.346527
3	55.260582	55.439419	51.450363	52.096165	52.096165
4	52.140873	53.651051	51.604362	52.657513	52.657513

In [11]:
```
# Standardization the dataset
def standard_scaler(X_train, X_test):
    train_samples, train_nx, train_ny = X_train.shape
    test_samples, test_nx, test_ny = X_test.shape

    X_train = X_train.reshape((train_samples, train_nx * train_ny))
    X_test = X_test.reshape((test_samples, test_nx * test_ny))

    preprocessor = prep.StandardScaler(with_mean=True, with_std=True).fit(X_train)
    X_train = preprocessor.transform(X_train)
    X_test = preprocessor.transform(X_test)

    X_train = X_train.reshape((train_samples, train_nx, train_ny))
    X_test = X_test.reshape((test_samples, test_nx, test_ny))

    return X_train, X_test
```

In [12]:
```
# Split the data to X_train, y_train, X_test, y_test
def preprocess_data(stock, seq_len):
    amount_of_features = len(stock.columns)
    data = stock.as_matrix()

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index : index + sequence_length])

    result = np.array(result)
    row = round(0.99 * result.shape[0])
    train = result[: int(row), :]
    y_test_org = result[int(row) :, -1][ : ,-1]

    train, result = standard_scaler(train, result)

    X_train = train[:, : -1]
    y_train = train[:, -1][: ,-1]
    #train_temp = train[:, -2][: ,-1]
    #y_train = (train_temp - y_train)/y_train

    X_test = result[int(row) :, : -1]
    y_test = result[int(row) :, -1][ : ,-1]
    #test_temp = result[int(row) :, -2][ : ,-1]
    #y_test = (test_temp - y_test)/y_test

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], amount_of_features))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], amount_of_features))

    return [X_train, y_train, X_test, y_test, y_test_org]
```

In [13]:
```
# Build LSTM Neural Network
# LSTM --> Dropout --> LSTM --> Dropout --> Fully-Conneted(Dense)
def build_model(layers):
    model = Sequential()

    # By setting return_sequences to True we are able to stack another LSTM layer
    model.add(LSTM(
        return_sequences=True,
        input_shape=(None, 5), units=20))
    model.add(Dropout(0.1))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.1))

    model.add(Dense(
        units=1))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop", metrics=['accuracy'])
    print("Compilation Time : ", time.time() - start)
    return model
```

In [14]:
```
window = 20
X_train, y_train, X_test, y_test, y_test_org = preprocess_data(df[:: 1], window)
print("X_train", X_train.shape)
print("y_train", y_train)
print("X_test", X_test)
print("y_test", y_test)
```

```
X_train (3293, 20, 5)
y_train [-1.45135311 -1.44718798 -1.45057355 ...,  2.72242353  2.7160568
  2.72435148]
X_test [[[ 2.46954437  2.45159511  2.45858682  2.44750992  2.44750992]
  [ 2.45512241  2.45052344  2.46487619  2.47585245  2.47585245]
  [ 2.47774833  2.49967889  2.50539877  2.51752494  2.51752494]
  ...,
  [ 2.66279221  2.71041412  2.68897457  2.7328161   2.7328161 ]
  [ 2.72122383  2.7222555   2.74326066  2.72296143  2.72296143]
  [ 2.73820845  2.73261236  2.75780456  2.72781721  2.72781721]]

 [[ 2.45777118  2.45313265  2.46750584  2.47846711  2.47846711]
  [ 2.48038096  2.5023231   2.50807385  2.52023996  2.52023996]
  [ 2.52324199  2.51165013  2.51375917  2.52147515  2.52147515]
  ...,
  [ 2.72450896  2.72566405  2.7466342   2.726434    2.726434  ]
  [ 2.74166086  2.73606373  2.76133604  2.73126942  2.73126942]
  [ 2.73865715  2.71786401  2.73522879  2.73822773  2.73822773]]

 [[ 2.48304463  2.5049622   2.51072858  2.52288009  2.52288009]
  [ 2.52590099  2.5143013   2.51643918  2.52419256  2.52419256]
  [ 2.49940554  2.50351858  2.52586548  2.50367647  2.50367647]
  ...,
  [ 2.74496123  2.73948312  2.76472349  2.7347487   2.7347487 ]
  [ 2.74210992  2.72130362  2.73874181  2.74168826  2.74168826]
  [ 2.73053371  2.72917416  2.75595347  2.73903541  2.73903541]]

 ...,
 [[ 2.89678399  2.91692927  2.901791    2.91168596  2.91168596]
  [ 2.89941878  2.90966833  2.91225711  2.90668773  2.90668773]
  [ 2.90557877  2.93240146  2.93917611  2.94396868  2.94396868]
  ...,
  [ 2.93824691  2.92953195  2.95660592  2.9427536   2.9427536 ]
  [ 2.93871027  2.94147815  2.96920295  2.95998919  2.95998919]
  [ 2.95363692  2.99270316  2.9810335   3.01738114  3.01738114]]

 [[ 2.90232948  2.91254244  2.91514651  2.90954955  2.90954955]
  [ 2.9084594   2.93529711  2.94210702  2.94694075  2.94694075]
  [ 2.92623698  2.92526361  2.92117628  2.94433172  2.94433172]
  ...,
  [ 2.94215755  2.94505893  2.97275037  2.96365309  2.96365309]
  [ 2.95726087  2.99636196  2.98474751  3.02106486  3.02106486]
```

```
  [ 3.02136209  3.02357938  2.98807741  2.98686797  2.98686797]]

 [[ 2.91137543  2.938186    2.94501374  2.94982566  2.94982566]
  [ 2.92912957  2.92815511  2.92409658  2.947304    2.947304  ]
  [ 2.92782596  2.94586324  2.95452614  2.97566128  2.97566128]
  ...,
  [ 2.96072199  2.99998585  2.98830689  3.02477806  3.02477806]
  [ 3.02503998  3.0272628   2.99179718  2.9905273   2.9905273 ]
  [ 2.96566525  2.9418905   2.87813352  2.87132028  2.87132028]]]
y_test [ 2.73475362  2.73556065  2.73838538  2.70090198  2.71771574  2.62916331
  2.63853402  2.65104363  2.64759112  2.85702331  2.84733843  2.84523124
  2.88495646  2.88531496  2.91625237  2.88675004  2.92006338  2.94929698
  2.91078243  2.89647925  2.88607738  2.8871983   2.86437631  2.91634213
  2.85621628  2.85303278  2.92526458  2.93185545  2.9527046   3.01368237
  2.98319376  2.86773906  2.86661814]
```

In [15]:
```
model = build_model([X_train.shape[2], window, 50, 1])
```

```
Compilation Time :  0.02867722511291504
```

In [16]:
```
# Training the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=300,
    validation_split=0.1,
    verbose=0)
```

Out[16]:
```
<keras.callbacks.History at 0x11c18c400>
```

In [17]:
```
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore[0], math.sqrt(trainScore[0])))
print(model.metrics_names)
testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore[0], math.sqrt(testScore[0])))
```

```
Train Score: 0.01 MSE (0.08 RMSE)
['loss', 'acc']
Test Score: 0.26 MSE (0.51 RMSE)
```

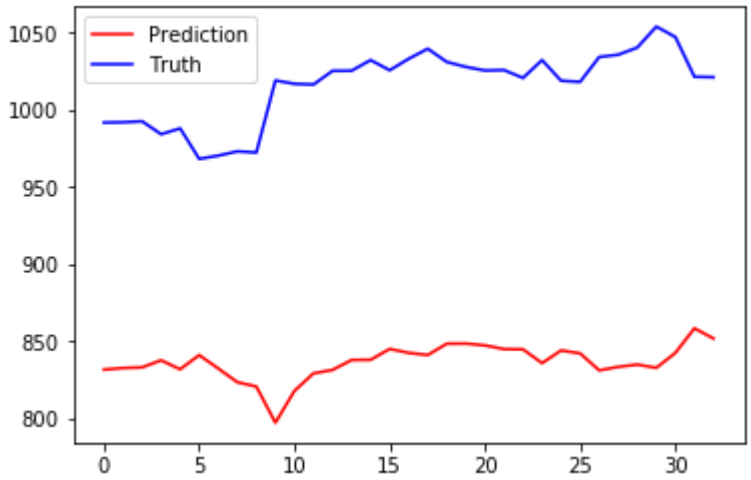
In [18]:
```
#Visualize the Prediction
diff = []
ratio = []
pred = model.predict(X_test)
for u in range(len(y_test)):
    pr = pred[u][0]
    ratio.append((y_test[u] / pr) - 1)
    diff.append(abs(y_test[u] - pr))
#print('error_ratio', ratio)
#print('error_abs', diff)
#print(pred)
```

In [19]:
```
# Scale the representation back
y_test = y_test_org
pred_org = []
for u in range(len(y_test)):
    pred_org.append(y_test[u]/(ratio[u] + 1))
    #print(ratio[u])
print(pred_org)
print(y_test)
```

```
[831.85985782969169, 832.88837382811664, 833.31085589661939, 837.89680754055848, 832.07770200117966, 841.270
82413042069, 832.57173677133801, 823.72104226471265, 820.88636285855978, 797.46651377643786, 818.07284012999
901, 829.43904685356495, 831.65255181366433, 838.09275924517408, 838.23784538737414, 845.25862220246347, 84
2.71372575600492, 841.30775186548158, 848.65448338577187, 848.74842595062591, 847.56769398768381, 845.170775
4019817, 845.02319306805771, 836.05804367217115, 844.28963913310474, 842.38956900494509, 831.36150226620236,
833.6509016045128, 835.12514464045103, 833.02394493227257, 842.73294114004477, 858.62991820628474, 852.03335
816727144]
[  992.         992.179993   992.809998   984.450012   988.200012
   968.450012   970.539978   973.330017   972.559998  1019.27002
  1017.109985  1016.640015  1025.5       1025.579956  1032.47998
  1025.900024  1033.329956  1039.849976  1031.26001   1028.069946  1025.75
  1026.        1020.909973  1032.5       1019.090027  1018.380005
  1034.48999   1035.959961  1040.609985  1054.209961  1047.410034
  1021.659973  1021.409973]
```

In [20]:
```
import matplotlib.pyplot as plt2

plt2.plot(pred_org, color='red', label='Prediction')
plt2.plot(y_test, color='blue', label='Truth')
plt2.legend(loc='upper left')
plt2.show()
```



In [ ]:
```
```