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

In [2]:
```
start = datetime.datetime(1990, 2, 16) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('CSCO', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                Open      High       Low     Close  Adj Close     Volume
Date
1990-02-16  0.073785  0.079861  0.073785  0.077257   0.063830  940636800
1990-02-20  0.077257  0.079861  0.074653  0.079861   0.065982  151862400
1990-02-21  0.078993  0.078993  0.075521  0.078125   0.064547   70531200
1990-02-22  0.078993  0.081597  0.078993  0.078993   0.065265   45216000
1990-02-23  0.078993  0.079861  0.078125  0.078559   0.064906   44697600
```

In [3]:
```
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 [4]:
```
#import os
#s=os.getcwd()
#s
```

In [5]:
```
#df = pd.read_csv('/Users/Yuffie/USA/SCU/COEN281DataMining/TermProject/data/GOOG.csv')
#df.head()
```

In [6]:
```
# Data preparation
col_list = df.columns.tolist()
col_list
```

Out[6]: ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']

In [7]:
```
col_list.remove('Close')
col_list.append('Close')
#col_list.remove('Date')
col_list.remove('Volume')
col_list
```

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

In [8]:
```
df = df[col_list]
df.head()
```

Out[8]:

	Open	High	Low	Adj Close	Close
Date					
1990-02-16	0.073785	0.079861	0.073785	0.063830	0.077257
1990-02-20	0.077257	0.079861	0.074653	0.065982	0.079861
1990-02-21	0.078993	0.078993	0.075521	0.064547	0.078125
1990-02-22	0.078993	0.081597	0.078993	0.065265	0.078993
1990-02-23	0.078993	0.079861	0.078125	0.064906	0.078559

In [9]:
```
# Save data
df.to_csv('CSCO-adjust.csv', index=False)
validate_df = pd.read_csv('CSCO-adjust.csv')
validate_df.head()
```

Out[9]:

	Open	High	Low	Adj Close	Close
0	0.073785	0.079861	0.073785	0.063830	0.077257
1	0.077257	0.079861	0.074653	0.065982	0.079861
2	0.078993	0.078993	0.075521	0.064547	0.078125
3	0.078993	0.081597	0.078993	0.065265	0.078993
4	0.078993	0.079861	0.078125	0.064906	0.078559

In [10]:
```
# 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 [11]:
```
# 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 [12]:
```
# 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 [13]:
```
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)
```

In [14]:
```
model = build_model([X_train.shape[2], window, 50, 1])
```

```
Compilation Time :  0.027719974517822266
```

In [15]:
```
# Training the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=300,
    validation_split=0.1,
    verbose=0)
```

Out[15]:
```
<keras.callbacks.History at 0x11eb5ed30>
```

In [16]:
```
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: 0.02 MSE (0.16 RMSE)
```

In [17]:
```
#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)
```

```
error_ratio [-0.10002125594084632, -0.097734102457580629, -0.10880591335525391, -0.12126242661110309, -0.085
356287641541573, -0.091674665551494061, -0.10838080424605545, -0.1602378164385353, -0.1169279885690947, -0.1
1600702240266958, -0.13264197696991387, -0.074182685535347459, -0.084743981633116139, -0.12580514179465385,
-0.12364521548859586, -0.10293768979756757, -0.10723468006329018, -0.11800221129632127, -0.1140761369750166
2, -0.11114102743923149, -0.075593571238189883, -0.083159202284251332, -0.10978717230378143, -0.139298139539
59169, -0.13440215187826021, -0.10385527692715668, -0.10218218046023808, -0.10762094107453279, -0.1393658307
406066, -0.12169688066066064, -0.11296466948274819, -0.1199033059899578, -0.13645633690240555, -0.1268289735
3964784, -0.13948371390444747, -0.11540684311620475, -0.11447617616219874, -0.11736235546028895, -0.12434185
646863405, -0.11224481870303271, -0.090077628755886045, -0.10965335922893926, -0.11230088083134304, -0.136057
85562634942, -0.129563477815692, -0.11244865630081835, -0.13726389527017813, -0.12009439422931567, -0.093082
976971779252, -0.13548541317031282, -0.11642172250107752, -0.12015253541269777, -0.12264684737777676, -0.117
15223326661217, -0.1434106979358275, -0.13170452839323388, -0.12178966592246609, -0.11492582931871742, -0.11
463111426346073, -0.021568233972119155, -0.10170890274548006, -0.099007428940544928, -0.11265773575328564, -
0.1362617841644802, -0.12761972988545955, -0.10513937640066573, -0.077594809088669336, -0.12354734594080141,
-0.13762220656426083, -0.11600186381967625]
```

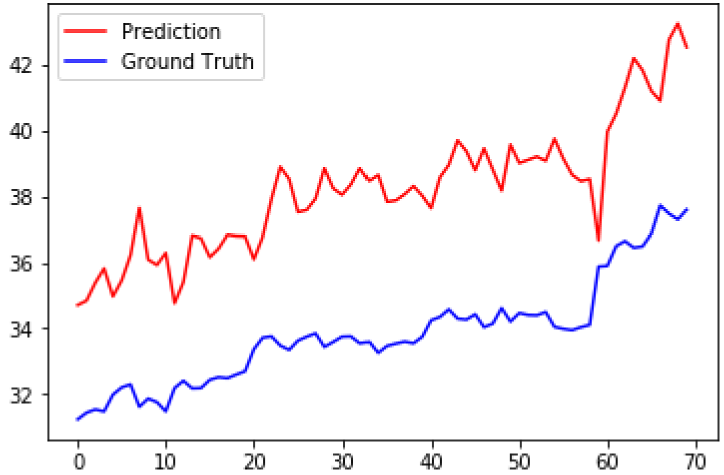
In [18]:

```
# 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)
```

```
[34.711930927500504, 34.845604921604469, 35.390720688851125, 35.824119684100623, 34.975367531376833, 35.4608
61630107949, 36.226226570511905, 37.653518602014593, 36.089923117774667, 35.927887217297638, 36.294124414766
671, 34.769277369384312, 35.41085701663021, 36.811014956165529, 36.731697674186783, 36.162481280346668, 36.4
26146125731478, 36.836829316491112, 36.797742289825493, 36.788739281994935, 36.098839170447157, 36.778469156
271484, 37.923513287679171, 38.898486848966272, 38.528282010365594, 37.527421781477564, 37.59114518054551, 3
7.932308766590182, 38.855067802823029, 38.244199821658881, 38.048089900003824, 38.359419174928505, 38.851537
488739936, 38.468981427575137, 38.651212693383904, 37.836604024732239, 37.875887804621534, 38.06771465941966
4, 38.314037558880123, 38.017237985243732, 37.64057361637331, 38.580476891845301, 38.954642686121701, 39.701
731445069342, 39.371050187555895, 38.792121993191849, 39.455866994995084, 38.810983560093341, 38.17328170156
3916, 39.571338090955201, 39.011824846545117, 39.109051721982532, 39.208842980942919, 39.078084920181595, 3
9.750670383050263, 39.145663096805144, 38.658166139277839, 38.460054679708747, 38.526315470894211, 36.670928
15849724, 39.964775460563388, 40.51087786115766, 41.303117722126139, 42.200287461798624, 41.828091773796075,
41.201945898234321, 40.903932861352381, 42.763291121791127, 43.25250404627846, 42.534024067591588]
[ 31.24      31.440001  31.540001  31.48      31.99      32.209999
  32.299999  31.620001  31.870001  31.76      31.48      32.189999  32.41
  32.18      32.189999  32.439999  32.52      32.490002  32.599998
  32.700001  33.369999  33.720001  33.759998  33.48      33.349998
  33.630001  33.75      33.849998  33.439999  33.59      33.75      33.759998
  33.549999  33.59      33.259998  33.470001  33.540001  33.599998
  33.549999  33.75      34.25      34.349998  34.580002  34.299999  34.27
  34.43      34.040001  34.150002  34.619999  34.209999  34.470001  34.41
  34.400002  34.5       34.049999  33.990002  33.950001  34.040001
  34.110001  35.880001  35.900002  36.5      36.650002  36.450001
  36.490002  36.869999  37.73      37.48      37.299999  37.599998]
```

In [19]:
```
import matplotlib.pyplot as plt2

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



In [ ]: