Branch: master

Find file Copy path

wsu / methods / project / ANN.ipynb

borodark conclusion

1a104b1 11 hours ago

1 contributor

Raw Blame History

546 lines (545 sloc) 143 KB

Finansial Time Series Forecasting using Artificial Neural Netwo

Importing the dataset

The same daily data for the stock of American Airlines Group Inc (NASDAQ: AAL) is used in this forecast
dataset also contains the Date, Adjusted Close and Volume data.

In [1]:
```
#import all libraries
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
from IPython.display import Image
import ml_metrics as metric
import forecasting_metrics as fmetric
```

In [16]:
```
# import dataset
dataset = pd.read_csv('data/stock_market_data-AAL.csv')
df_stock = dataset.copy()
df_stock = df_stock.dropna().sort_values(by=['Date'])
print(df_stock[:10])
df_stock = df_stock[['Open', 'High', 'Low', 'Close']]
print(df_stock[:10])
print('Dataset shape = ',df_stock.shape)
```

```
      index        Date    Low   High  Close   Open
3392      0  2005-09-27  19.10  21.40  19.30  21.05
3391      1  2005-09-28  19.20  20.53  20.50  19.30
3390      2  2005-09-29  20.10  20.58  20.21  20.40
3389      3  2005-09-30  20.18  21.05  21.01  20.26
3388      4  2005-10-03  20.90  21.75  21.50  20.90
3387      5  2005-10-04  21.44  22.50  22.16  21.44
3386      6  2005-10-05  21.75  22.31  22.20  22.10
3385      7  2005-10-06  22.40  23.00  22.58  22.60
3384      8  2005-10-07  21.80  22.60  22.15  22.25
3383      9  2005-10-10  22.10  22.29  22.21  22.28
       Open   High    Low  Close
3392  21.05  21.40  19.10  19.30
3391  19.30  20.53  19.20  20.50
3390  20.40  20.58  20.10  20.21
3389  20.26  21.05  20.18  21.01
3388  20.90  21.75  20.90  21.50
3387  21.44  22.50  21.44  22.16
3386  22.10  22.31  21.75  22.20
3385  22.60  23.00  22.40  22.58
3384  22.25  22.60  21.80  22.15
3383  22.28  22.29  22.10  22.21
Dataset shape =  (3393, 4)
```

```
Dataset shape = (3393, 4)
```

Standardizing the dataset

The process makes the **mean** of all the input features equal to 0 and **variance** to 1. This way there is **no bi**

Without scaling the neural network get confused and may give a higher weight to the features having hig as `tanh` or `sigmoid` are defined on the [-1, 1] or [0, 1] interval respectively.

The **rectified linear unit**, also known as `ReLU`, activations are commonly used activations which are un performed using sklearn's `MinMaxScaler`.

In [3]:
```
def normalize_data(df):
    min_max_scaler = sklearn.preprocessing.MinMaxScaler()
    df['Open'] = min_max_scaler.fit_transform(df.Open.values.reshape
    df['High'] = min_max_scaler.fit_transform(df.High.values.reshape
    df['Low'] = min_max_scaler.fit_transform(df.Low.values.reshape(-
    df['Close'] = min_max_scaler.fit_transform(df['Close'].values.re
    return df
df_stock_norm = df_stock.copy()
df_stock_norm = normalize_data(df_stock_norm)
print(df_stock_norm[:10])
```

```
          Open      High       Low     Close
3392  0.315980  0.316297  0.291495  0.286648
3391  0.287239  0.302090  0.293146  0.306259
3390  0.305305  0.302907  0.308010  0.301520
3389  0.303005  0.310581  0.309331  0.314594
3388  0.313516  0.322012  0.321222  0.322602
3387  0.322385  0.334259  0.330140  0.333388
3386  0.333224  0.331156  0.335260  0.334042
3385  0.341435  0.342423  0.345995  0.340252
3384  0.335687  0.335892  0.336086  0.333224
3383  0.336180  0.330830  0.341040  0.334205
```

Splitting the dataset into Training and Testing: building X & Y

The whole dataset is split into train, valid and test data. The result is: `x_train`, `y_train`, `x_valid`, `y_v`

In [4]:
```
# Splitting the dataset into Train, Valid & test data
valid_set_size_percentage = 10
test_set_size_percentage = 10
seq_len = 20 # taken sequence length as 20
def load_data(stock, seq_len):
    data_raw = stock.values
    data = []
    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
    test_set_size = int(np.round(test_set_size_percentage/100*data.s
    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]

x_train, y_train, x_valid, y_valid, x_test, y_test = load_data(df_st
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)
```

```
x_train.shape =  (2699, 19, 4)
y_train.shape =  (2699, 4)
x_valid.shape =  (337, 19, 4)
y_valid.shape =  (337, 4)
x_test.shape =  (337, 19, 4)
y_test.shape =  (337, 4)
```

Our total data set is 3393.

So the first 19 data points are x_train.

The next 2699 data points are y_train out of which last 19 data points are x_valid.

The next 337 data points are y_valid out of which last 19 data are x_test.

Finally, the next and last 337 data points are y_test.

Building the Model

Parameters, Placeholders & Variables

We will first fix the Parameters, Placeholders & Variables to building any model. The Artificial Neural N (features of the stock (OHLC) at time **T = t**) and Y the network's output: **Price of the stock at T+1**. The sha and the outputs are a 1-dimensional vector. The crucial part is to properly define the input and outpu observations per training batch. The training is stopped when epoch reaches 100.

In [5]:
```
## Building the Model
# parameters & Placeholders
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]
tf.reset_default_graph()
X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])
```

Designing the network architecture

The function `get_next_batch` runs the next batch for any model . Then we will write the layers for each

In [6]:
```
# function to get the 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:
```

Let's run the model using GRU cell: https://en.wikipedia.org/wiki/Gated_recurrent_unit (https://en.wikiped

In [7]:
```
#GRU
layers = [tf.contrib.rnn.GRUCell(num_units=n_neurons, activation=tf.
          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=t
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 Tenso
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/2018090
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issu

WARNING:tensorflow:From <ipython-input-7-04dc9adfbf9c>:3: GRUCell.__
future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.GRUCell, and will be rep
WARNING:tensorflow:From <ipython-input-7-04dc9adfbf9c>:5: MultiRNNCe
in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.StackedRNNCells, and wil
WARNING:tensorflow:From <ipython-input-7-04dc9adfbf9c>:6: dynamic_rn
n.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
WARNING:tensorflow:From /Users/iostaptchenko/projects/secret/wsu/ds/
(from tensorflow.python.framework.ops) is deprecated and will be rem
Instructions for updating:
```

```
Colocations handled automatically by placer.
WARNING:tensorflow:From <ipython-input-7-04dc9adfbf9c>:8: dense (fro
Instructions for updating:
Use keras.layers.dense instead.
```

Cost function

This is the cost function to optimize the model. The cost function is used to generate a measure of de **squared error** (`MSE`) function is commonly used. `MSE` computes the average squared deviation between p

In [8]:
```
# Cost function
loss = tf.reduce_mean(tf.square(outputs - y))
```

Optimizer

The optimizer takes care of the necessary computations that are used to adapt the network's weight a which the **weights and biases have to be changed** during training in order to minimize the network's c **research**.

In [9]:
```
#optimizer
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

In this model we use Adam (Adaptive Moment Estimation) Optimizer, which is an extension of the stocha

Fitting the neural network model & prediction

After having defined the placeholders, variables, initializers, cost functions and optimizers of the network n = batch_size are drawn from the training data and fed into the network.

The training dataset gets divided into n / batch_size batches that are sequentially fed into the network. targets.

A sampled data batch of X flows through the network until it reaches the output layer. There, TensorFlo conducts an optimization step and updates the network parameters, corresponding to the selected learn

The procedure continues until all batches have been presented to the network. One full sweep over all ba

The training of the network stops once the maximum number of epochs is reached or another stopping c

In [10]:
```
# 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) # fetch the ne
        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: RMSE train/valid = %.6f/%.6f'%( # pr
```

```
                iteration*batch_size/train_set_size, math.sqrt(mse_t
# Predictions
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})
```

```
0.00 epochs: RMSE train/valid = 0.350956/0.563553
4.98 epochs: RMSE train/valid = 0.022483/0.022241
9.97 epochs: RMSE train/valid = 0.014662/0.014732
14.95 epochs: RMSE train/valid = 0.014084/0.016229
19.93 epochs: RMSE train/valid = 0.011915/0.012340
24.92 epochs: RMSE train/valid = 0.012240/0.014829
29.90 epochs: RMSE train/valid = 0.013865/0.017448
34.88 epochs: RMSE train/valid = 0.011011/0.012309
39.87 epochs: RMSE train/valid = 0.011189/0.012480
44.85 epochs: RMSE train/valid = 0.012009/0.014371
49.83 epochs: RMSE train/valid = 0.011302/0.012482
54.82 epochs: RMSE train/valid = 0.011273/0.012543
59.80 epochs: RMSE train/valid = 0.013086/0.016700
64.78 epochs: RMSE train/valid = 0.010705/0.011430
69.77 epochs: RMSE train/valid = 0.010558/0.011242
74.75 epochs: RMSE train/valid = 0.010424/0.011204
79.73 epochs: RMSE train/valid = 0.011985/0.013605
84.72 epochs: RMSE train/valid = 0.011263/0.012922
89.70 epochs: RMSE train/valid = 0.011087/0.012915
94.68 epochs: RMSE train/valid = 0.011079/0.012470
99.67 epochs: RMSE train/valid = 0.010580/0.012179
```

Now we have predicted the scaled stock prices and saved as y_test_pred. We can compare these predic

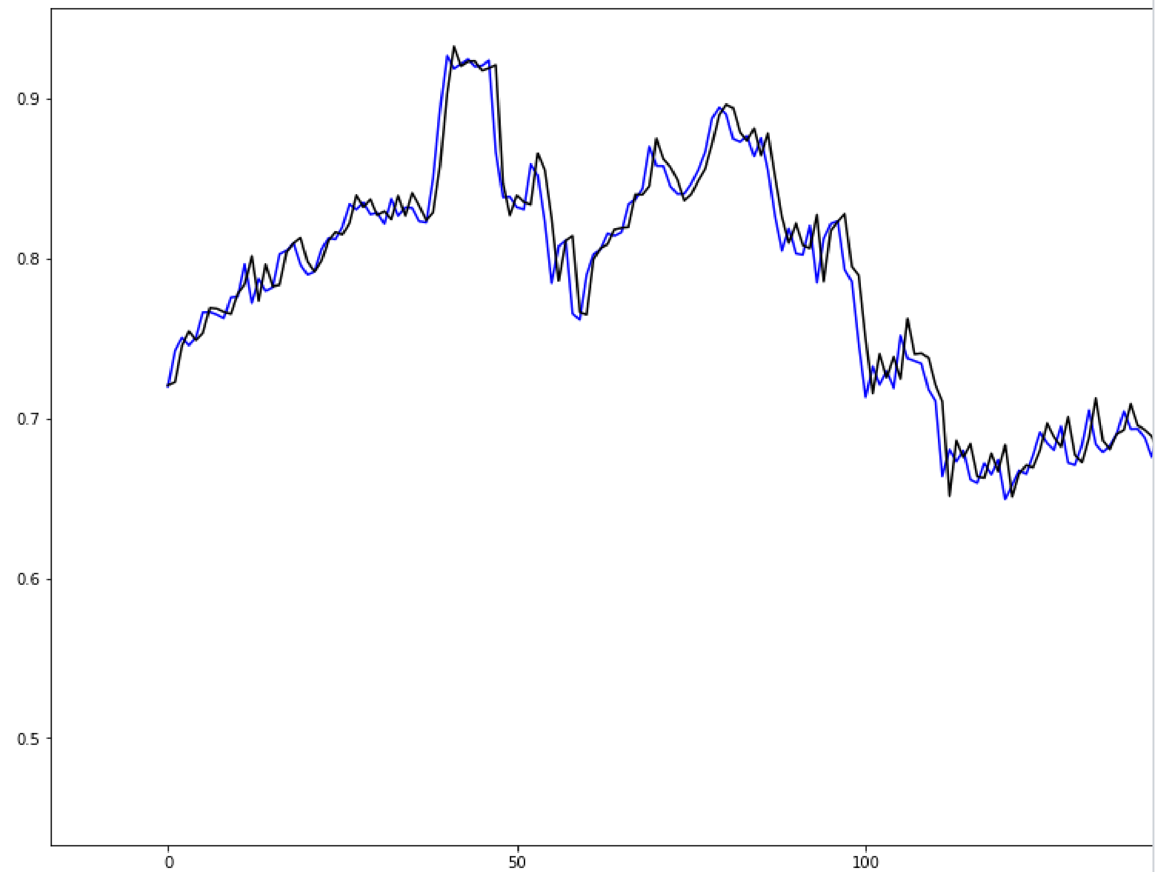
In [13]:
```
#checking prediction output nos
print(y_test_pred.shape)
```

```
(337, 4)
```

Let's compare between our target and prediction.

In [19]:
```
# ploting the graph
comp = pd.DataFrame({'test':y_test[:,3],'pred':y_test_pred[:,3]})
plt.figure(figsize=(30,10))
plt.plot(comp['test'], color='blue', label='Target')
plt.plot(comp['pred'], color='black', label='Prediction')
plt.legend()
plt.show()
# Print errors
def mape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

errors = {
'ME':  fmetric.me(actual=comp['test'], predicted=comp['pred']),
'RMSE': fmetric.rmse(actual=comp['test'], predicted=comp['pred']),
'MAE':  fmetric.mae(actual=comp['test'], predicted=comp['pred']),
'MPE':  100*fmetric.mpe(actual=comp['test'], predicted=comp['pred'])
'MAPE': mape(comp['test'], comp['pred']),
'MASE': fmetric.mase(actual=comp['test'], predicted=comp['pred']),
}
print(errors)
```




```
{'ME': -0.003913519198776012, 'RMSE': 0.01641030210773595, 'MAE': 0.
```