Sanjana Rajesh

PREDICTING MARKETS WITH DEEP LEARNING MODELS November 12 2019

In [85]:

```
from alpha_vantage.timeseries import TimeSeries
import json
from pprint import pprint
import keras
import tensorflow as tf
from keras.models import Model
from keras.layers import Dense, Dropout, LSTM, Input, Activation, concatenate
from keras import optimizers
import numpy as np
import pandas as pd
from sklearn import preprocessing
np.random.seed(4)
#from tensorflow import set_random_seed
#set_random_seed(4)
```

In [86]:

```
symbol = "LTOUF"
time_window = "daily"
```

In [87]:

```
api_key = "JDVFH3QH32AO8GEX"
print(symbol, time_window)
ts = TimeSeries(key=api_key, output_format='pandas')
```

```
LTOUF daily
```

In [88]:

```
if time_window == 'intraday':
    data, meta_data = ts.get_intraday(symbol, interval='1min', outputsize='full')
elif time_window == 'daily':
    data, meta_data = ts.get_daily(symbol, outputsize='full')
elif time_window == 'daily_adj':
    data, meta_data = ts.get_daily_adjusted(symbol, outputsize='full')
```

In [89]:

```
pprint(data.head(10))
data.to_csv(f'./{symbol}_{time_window}.csv')
```

```
            1. open  2. high  3. low  4. close  5. volume
date
2007-07-16     59.5     59.5    59.5       59.5      800.0
2007-07-17     59.5     59.5    59.5       59.5        0.0
2007-07-18     59.5     59.5    59.5       59.5        0.0
2007-07-19     59.5     59.5    59.5       59.5        0.0
2007-07-20     59.5     59.5    59.5       59.5        0.0
2007-07-23     59.5     59.5    59.5       59.5        0.0
2007-07-24     64.5     64.5    64.5       64.5     1000.0
2007-07-25     64.5     64.5    64.5       64.5        0.0
2007-07-26     64.5     64.5    64.5       64.5        0.0
2007-07-27     64.5     64.5    64.5       64.5        0.0
```

In [90]:

```
filename = symbol + "_" + time_window + ".csv"
```

In [91]:

```
history_points = 50
data = pd.read_csv(filename)
data = data.drop('date', axis=1)
data = data.drop(0, axis=0)
```

In [92]:

```
def csv_to_dataset(csv_path):
    data = pd.read_csv(csv_path)
    data = data.drop('date', axis=1)
    data = data.drop(0, axis=0)

    data = data.values

    data_normaliser = preprocessing.MinMaxScaler()
    data_normalised = data_normaliser.fit_transform(data)

    # using the last {history_points} open close high low volume data points, predict the next ope
n value
    ohlcv_histories_normalised = np.array([data_normalised[i:i + history_points].copy() for i in ra
nge(len(data_normalised) - history_points)])
    next_day_open_values_normalised = np.array([data_normalised[:, 0][i + history_points].copy() fo
r i in range(len(data_normalised) - history_points)])
    next_day_open_values_normalised = np.expand_dims(next_day_open_values_normalised, -1)

    next_day_open_values = np.array([data[:, 0][i + history_points].copy() for i in range(len(data)
- history_points)])
    next_day_open_values = np.expand_dims(next_day_open_values, -1)

    y_normaliser = preprocessing.MinMaxScaler()
    y_normaliser.fit(next_day_open_values)

    def calc_ema(values, time_period):
        # https://www.investopedia.com/ask/answers/122314/what-exponential-moving-average-ema-
formula-and-how-ema-calculated.asp
        sma = np.mean(values[:, 3])
        ema_values = [sma]
        k = 2 / (1 + time_period)
        for i in range(len(his) - time_period, len(his)):
            close = his[i][3]
            ema_values.append(close * k + ema_values[-1] * (1 - k))
        return ema_values[-1]

    technical_indicators = []
    for his in ohlcv_histories_normalised:
        # note since we are using his[3] we are taking the SMA of the closing price
        sma = np.mean(his[:, 3])
        macd = calc_ema(his, 12) - calc_ema(his, 26)
        technical_indicators.append(np.array([sma]))
        # technical_indicators.append(np.array([sma,macd,]))

    technical_indicators = np.array(technical_indicators)

    tech_ind_scaler = preprocessing.MinMaxScaler()
    technical_indicators_normalised = tech_ind_scaler.fit_transform(technical_indicators)

    assert ohlcv_histories_normalised.shape[0] == next_day_open_values_normalised.shape[0] == techn
ical_indicators_normalised.shape[0]
    return ohlcv_histories_normalised, technical_indicators_normalised,
next_day_open_values_normalised, next_day_open_values, y_normaliser
```

In [93]:

```
def multiple_csv_to_dataset(test_set_name):
    import os
    ohlcv_histories = 0
    technical_indicators = 0
    next_day_open_values = 0
    for csv_file_path in list(filter(lambda x: x.endswith('daily.csv'), os.listdir('./'))):
        if not csv_file_path == test_set_name:
            print(csv_file_path)
            if type(ohlcv_histories) == int:
                ohlcv_histories, technical_indicators, next_day_open_values, _, _ = csv_to_dataset(
```

```
csv_file_path)
            else:
                a, b, c, _, _ = csv_to_dataset(csv_file_path)
                ohlcv_histories = np.concatenate((ohlcv_histories, a), 0)
                technical_indicators = np.concatenate((technical_indicators, b), 0)
                next_day_open_values = np.concatenate((next_day_open_values, c), 0)

    ohlcv_train = ohlcv_histories
    tech_ind_train = technical_indicators
    y_train = next_day_open_values

    ohlcv_test, tech_ind_test, y_test, unscaled_y_test, y_normaliser = csv_to_dataset(test_set_name
)

    return ohlcv_train, tech_ind_train, y_train, ohlcv_test, tech_ind_test, y_test, unscaled_y_test
, y_normaliser
```

In [94]:

```
ohlcv_histories, technical_indicators, next_day_open_values, unscaled_y, y_normaliser =
csv_to_dataset('MSFT_daily.csv')

test_split = 0.9
n = int(ohlcv_histories.shape[0] * test_split)

ohlcv_train = ohlcv_histories[:n]
tech_ind_train = technical_indicators[:n]
y_train = next_day_open_values[:n]

ohlcv_test = ohlcv_histories[n:]
tech_ind_test = technical_indicators[n:]
y_test = next_day_open_values[n:]

unscaled_y_test = unscaled_y[n:]

print(ohlcv_train.shape)
print(ohlcv_test.shape)
```

```
(4482, 50, 5)
(499, 50, 5)
```

MODEL

In [95]:

```
# define two sets of inputs
lstm_input = Input(shape=(history_points, 5), name='lstm_input')
dense_input = Input(shape=(technical_indicators.shape[1],), name='tech_input')

# the first branch operates on the first input
x = LSTM(50, name='lstm_0')(lstm_input)
x = Dropout(0.2, name='lstm_dropout_0')(x)
lstm_branch = Model(inputs=lstm_input, outputs=x)

# the second branch opreates on the second input
y = Dense(20, name='tech_dense_0')(dense_input)
y = Activation("relu", name='tech_relu_0')(y)
y = Dropout(0.2, name='tech_dropout_0')(y)
technical_indicators_branch = Model(inputs=dense_input, outputs=y)

# combine the output of the two branches
combined = concatenate([lstm_branch.output, technical_indicators_branch.output],
name='concatenate')

z = Dense(64, activation="sigmoid", name='dense_pooling')(combined)
z = Dense(1, activation="linear", name='dense_out')(z)

# our model will accept the inputs of the two branches and
# then output a single value
model = Model(inputs=[lstm_branch.input, technical_indicators_branch.input], outputs=z)
adam = optimizers.Adam(lr=0.0005)
model.compile(optimizer=adam, loss='mse')
model.fit(x=[ohlcv_train, tech_ind_train], y=y_train, batch_size=32, epochs=50, shuffle=True, valid
ation_split=0.1)
```

```
ation_split=0.1)
```

```
Train on 4033 samples, validate on 449 samples
Epoch 1/50
4033/4033 [==============================] - 8s 2ms/step - loss: 0.0255 - val_loss: 8.1874e-04
Epoch 2/50
4033/4033 [==============================] - 4s 975us/step - loss: 3.6144e-04 - val_loss: 4.8707e-
04
Epoch 3/50
4033/4033 [==============================] - 3s 846us/step - loss: 2.8089e-04 - val_loss: 1.7739e-
04
Epoch 4/50
4033/4033 [==============================] - 4s 913us/step - loss: 2.5749e-04 - val_loss: 3.2015e-
04
Epoch 5/50
4033/4033 [==============================] - 4s 909us/step - loss: 2.2457e-04 - val_loss: 6.7110e-
05
Epoch 6/50
4033/4033 [==============================] - 4s 1ms/step - loss: 2.0302e-04 - val_loss: 7.1449e-05
Epoch 7/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.9304e-04 - val_loss: 9.2593e-05
Epoch 8/50
4033/4033 [==============================] - 5s 1ms/step - loss: 1.7523e-04 - val_loss: 1.2157e-04
Epoch 9/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.5852e-04 - val_loss: 1.1398e-04
Epoch 10/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.4359e-04 - val_loss: 2.2006e-04
Epoch 11/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.3621e-04 - val_loss: 7.1376e-05
Epoch 12/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.3447e-04 - val_loss: 1.4052e-04
Epoch 13/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.2929e-04 - val_loss: 1.7878e-04
Epoch 14/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.1571e-04 - val_loss: 1.1258e-04
Epoch 15/50
4033/4033 [==============================] - 5s 1ms/step - loss: 1.0367e-04 - val_loss: 3.2508e-04
Epoch 16/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.0830e-04 - val_loss: 8.6358e-05
Epoch 17/50
4033/4033 [==============================] - 4s 1ms/step - loss: 1.0129e-04 - val_loss: 1.8523e-04
Epoch 18/50
4033/4033 [==============================] - 4s 961us/step - loss: 9.6467e-05 - val_loss: 3.6866e-
04
Epoch 19/50
4033/4033 [==============================] - 4s 1ms/step - loss: 9.3313e-05 - val_loss: 7.7572e-05
Epoch 20/50
4033/4033 [==============================] - 4s 1ms/step - loss: 9.0372e-05 - val_loss: 9.7466e-05
Epoch 21/50
4033/4033 [==============================] - 4s 1ms/step - loss: 9.1084e-05 - val_loss: 1.1568e-04
Epoch 22/50
4033/4033 [==============================] - 5s 1ms/step - loss: 8.9750e-05 - val_loss: 8.4136e-05
Epoch 23/50
4033/4033 [==============================] - 5s 1ms/step - loss: 8.3688e-05 - val_loss: 2.0508e-04
Epoch 24/50
4033/4033 [==============================] - 5s 1ms/step - loss: 8.7739e-05 - val_loss: 8.3989e-05
Epoch 25/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.9414e-05 - val_loss: 1.3031e-04
Epoch 26/50
4033/4033 [==============================] - 4s 986us/step - loss: 9.0274e-05 - val_loss: 9.5653e-
05
Epoch 27/50
4033/4033 [==============================] - 5s 1ms/step - loss: 8.0015e-05 - val_loss: 9.9593e-05
Epoch 28/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.7423e-05 - val_loss: 1.4099e-04
Epoch 29/50
4033/4033 [==============================] - 5s 1ms/step - loss: 7.6404e-05 - val_loss: 1.0881e-04
Epoch 30/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.2516e-05 - val_loss: 1.6090e-04
Epoch 31/50
4033/4033 [==============================] - 4s 1ms/step - loss: 6.9800e-05 - val_loss: 8.7460e-05
Epoch 32/50
4033/4033 [==============================] - 5s 1ms/step - loss: 7.3127e-05 - val_loss: 8.1990e-05
Epoch 33/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.6048e-05 - val_loss: 1.5447e-04
Epoch 34/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.1930e-05 - val_loss: 7.0106e-05
```

```
Epoch 35/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.1053e-05 - val_loss: 1.1342e-04
Epoch 36/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.8915e-05 - val_loss: 5.2147e-05
Epoch 37/50
4033/4033 [==============================] - 5s 1ms/step - loss: 7.0300e-05 - val_loss: 7.4116e-05
Epoch 38/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.6368e-05 - val_loss: 7.1551e-05
Epoch 39/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.9339e-05 - val_loss: 7.8241e-05
Epoch 40/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.4636e-05 - val_loss: 4.1373e-05
Epoch 41/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.8827e-05 - val_loss: 7.2961e-05
Epoch 42/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.5371e-05 - val_loss: 1.2276e-04
Epoch 43/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.1156e-05 - val_loss: 2.3229e-04
Epoch 44/50
4033/4033 [==============================] - 5s 1ms/step - loss: 7.1535e-05 - val_loss: 3.4871e-04
Epoch 45/50
4033/4033 [==============================] - 4s 1ms/step - loss: 7.1682e-05 - val_loss: 1.8505e-04
Epoch 46/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.6073e-05 - val_loss: 8.0574e-05
Epoch 47/50
4033/4033 [==============================] - 4s 1ms/step - loss: 6.7226e-05 - val_loss: 1.6064e-04
Epoch 48/50
4033/4033 [==============================] - 5s 1ms/step - loss: 6.0825e-05 - val_loss: 3.5883e-05
Epoch 49/50
4033/4033 [==============================] - 4s 1ms/step - loss: 6.1162e-05 - val_loss: 9.4755e-05
Epoch 50/50
4033/4033 [==============================] - 4s 1ms/step - loss: 6.1541e-05 - val_loss: 2.5847e-05
```

Out[95]:

```
<keras.callbacks.callbacks.History at 0x25c5c6f4eb8>
```

EVALUATING MODEL

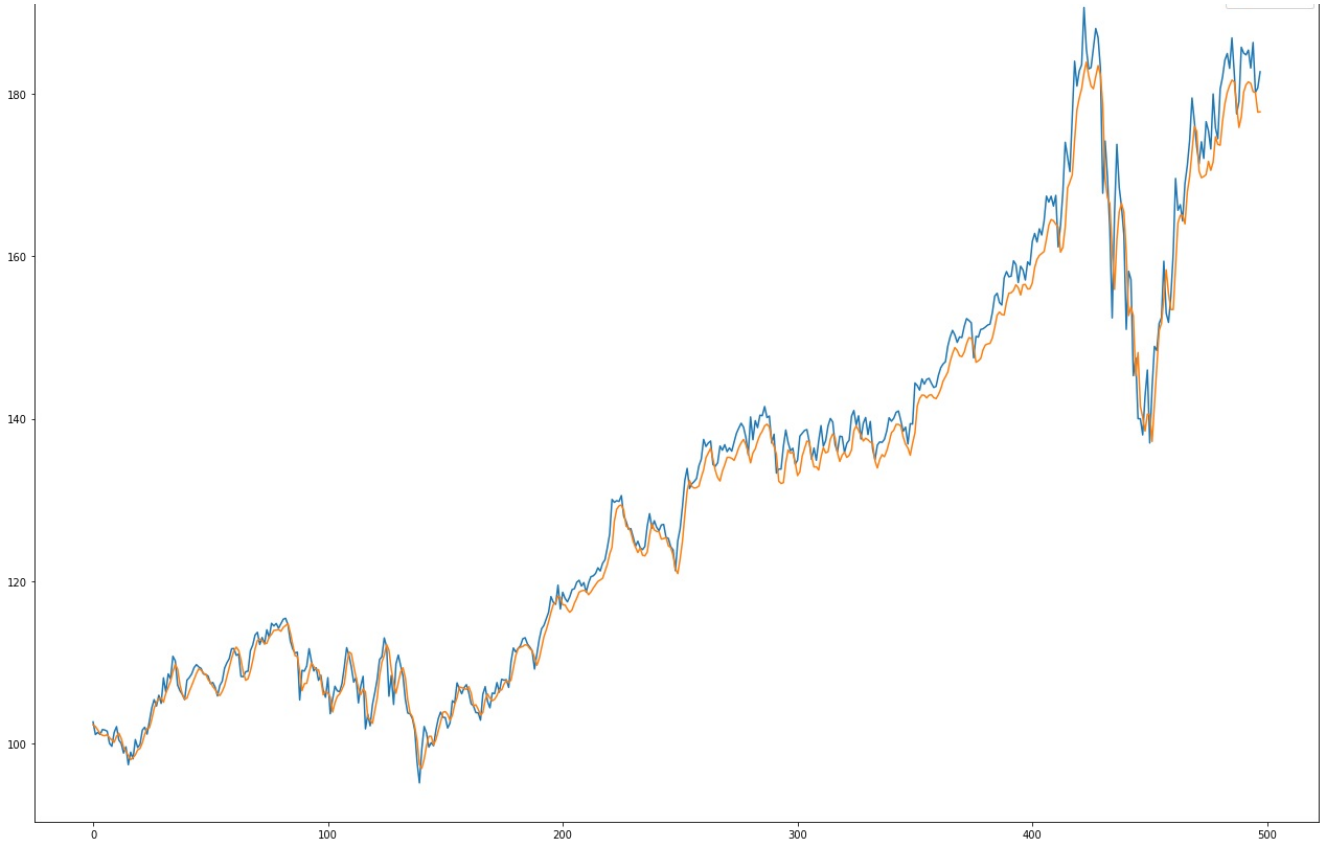
In [96]:

```
y_test_predicted = model.predict([ohlcv_test, tech_ind_test])
y_test_predicted = y_normaliser.inverse_transform(y_test_predicted)
y_predicted = model.predict([ohlcv_histories, tec +hnical_indicators])
y_predicted = y_normaliser.inverse_transform(y_predicted)
assert unscaled_y_test.shape == y_test_predicted.shape
real_mse = np.mean(np.square(unscaled_y_test - y_test_predicted))
scaled_mse = real_mse / (np.max(unscaled_y_test) - np.min(unscaled_y_test)) * 100
print(scaled_mse)
```

```
8.0439524013096
```

In [97]:

```
import matplotlib.pyplot as plt

plt.gcf().set_size_inches(22, 15, forward=True)

start = 0
end = -1

real = plt.plot(unscaled_y_test[start:end], label='real')
pred = plt.plot(y_test_predicted[start:end], label='predicted')

# real = plt.plot(unscaled_y[start:end], label='real')
# pred = plt.plot(y_predicted[start:end], label='predicted')

plt.legend(['Real', 'Predicted'])

plt.show()
```



In [98]:

```
from datetime import datetime
model.save(f'technical_model.h5')
```

In [99]:

```
ohlcv_histories, technical_indicators, next_day_open_values, unscaled_y, y_normaliser =
csv_to_dataset(filename)

test_split = 0.9
n = int(ohlcv_histories.shape[0] * test_split)

ohlcv_train = ohlcv_histories[:n]
tech_ind_train = technical_indicators[:n]
y_train = next_day_open_values[:n]

ohlcv_test = ohlcv_histories[n:]
tech_ind_test = technical_indicators[n:]
y_test = next_day_open_values[n:]

unscaled_y_test = unscaled_y[n:]

y_test_predicted = model.predict([ohlcv_test, tech_ind_test])
y_test_predicted = y_normaliser.inverse_transform(y_test_predicted)
```

ALGORITHM

In [100]:

```
buys = []
sells = []
thresh = 0.1

start = 0
end = -1

x = -1
for ohlcv, ind in zip(ohlcv_test[start: end], tech_ind_test[start: end]):
    normalised_price_today = ohlcv[-1][0]
    normalised_price_today = np.array([[normalised_price_today]])
```

```
    price_today = y_normaliser.inverse_transform(normalised_price_today)
    predicted_price_tomorrow = np.squeeze(y_normaliser.inverse_transform(model.predict([[ohlcv], [i
nd]])))
    delta = predicted_price_tomorrow - price_today
    if delta > thresh:
        buys.append((x, price_today[0][0]))
    elif delta < -thresh:
        sells.append((x, price_today[0][0]))
    x += 1
print(f"buys: {len(buys)}")
print(f"sells: {len(sells)}")
```

```
buys: 65
sells: 54
```

In [101]:

```
def compute_earnings(buys_, sells_):
    purchase_amt = 1000
    stock = 0
    balance = 0
    while len(buys_) > 0 and len(sells_) > 0:
        if buys_[0][0] < sells_[0][0]:
            # time to buy $10 worth of stock
            balance -= purchase_amt
            stock += purchase_amt / buys_[0][1]
            buys_.pop(0)
        else:
            # time to sell all of our stock
            balance += stock * sells_[0][1]
            stock = 0
            sells_.pop(0)
    print(f"earnings: ${balance}")
```

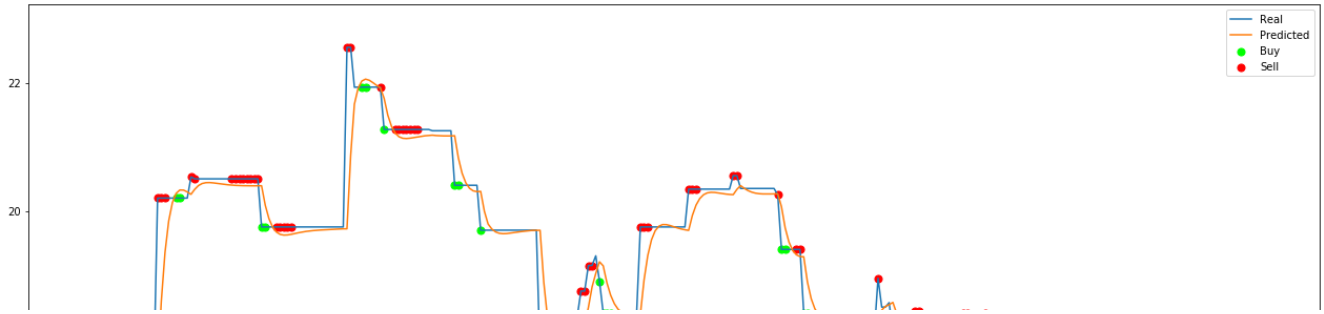
In [102]:

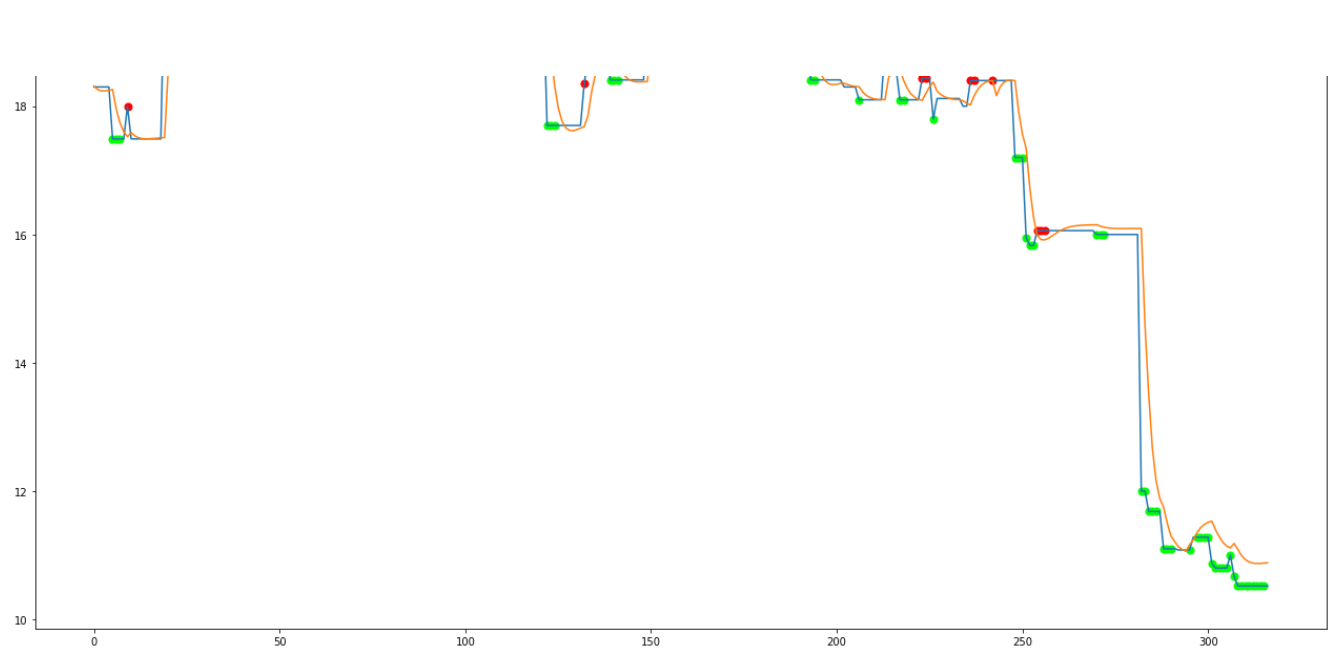
```
compute_earnings([b for b in buys], [s for s in sells])
```

```
earnings: $237.94525800095562
```

In [103]:

```
import matplotlib.pyplot as plt

plt.gcf().set_size_inches(22, 15, forward=True)

real = plt.plot(unscaled_y_test[start:end], label='real')
pred = plt.plot(y_test_predicted[start:end], label='predicted')

if len(buys) > 0:
    plt.scatter(list(list(zip(*buys))[0]), list(list(zip(*buys))[1]), c='#00ff00', s=50)
if len(sells) > 0:
    plt.scatter(list(list(zip(*sells))[0]), list(list(zip(*sells))[1]), c='#ff0000', s=50)

# real = plt.plot(unscaled_y[start:end], label='real')
# pred = plt.plot(y_predicted[start:end], label='predicted')

plt.legend(['Real', 'Predicted', 'Buy', 'Sell'])

plt.show()
```





In [ ]: