Learn by example RNN/LSTM/GRU time series

I know I cannot predict stock prices based on historic data, but still the Recurring Neural network examples (RNN or LSTM or GRU, etc) to predict stock prices are appealing, who knows I might discover something:-)

Welcome to my second notebook on Kaggle. I did record my notes so it might help others in their journey to understand Neural Networks by examples (in this case using Recurring Networks stock predictions.) After seeing many youtube video's and various courses on Neural Networks found the Kaggle Keras course and examples helping me a lot to move from powerpoint understanding to run my own Neural Networks using Keras. Many thanks to this community! The least I could do is to contribute back, hence this notebook.

Table of Contents

1. Introduction
2. Sinus wave proof of concept
    1. Results
3. IBM stock prediction
    1. Everybody on Kaggle rich!
    2. you can't predict future stock prices on historic data

Introduction

Please watch this video about RNN/LSTM/GRU time series prediction, it gives you a good overview, it inspired me to reproduce the steps taken in this notebook:
https://www.youtube.com/watch?v=2np77NOdnwk
The code base:
https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction
I would like to acknowledge this tutorial for providing ideas and code, learning by example:
https://www.kaggle.com/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru/notebook
If you are new to Neural Networks you might want to have a look at my first notebook:
https://www.kaggle.com/charel/learn-neural-networks-by-example-mnist-digits

```
import numpy as np # linear algebra
```

```
from numpy import newaxis
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
from keras import optimizers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

print ('import completed')

from google.colab import drive

drive.mount('/content/drive/')
```

```
     import completed
     Mounted at /content/drive/
```

```
# Enter in how much steps we will enroll the network.
# RNN/LSTM/GRU can be taught patterns over times series as big as the number of times you enr
# So by design these networks are deep/long to catch recurrent patterns.
Enrol_window = 100

print ('enrol window set to',Enrol_window )
```

```
     enrol window set to 100
```

```
# Support functions
sc = MinMaxScaler(feature_range=(0,1))
def load_data(datasetname, column, seq_len, normalise_window):
    # A support function to help prepare datasets for an RNN/LSTM/GRU
    data = datasetname.loc[:,column]

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

    if normalise_window:
        #result = sc.fit_transform(result)
        result = normalise_windows(result)

    result = np.array(result)

    #Last 10% is used for validation test, first 90% for training
    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return [x_train, y_train, x_test, y_test]

def normalise_windows(window_data):
    # A support function to normalize a dataset
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data

def predict_sequence_full(model, data, window_size):
    #Shift the window by 1 new prediction each time, re-run predictions on new window
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
    return predicted

def predict_sequences_multiple(model, data, window_size, prediction_len):
    #Predict sequence of <prediction_len> steps before shifting prediction run forward by <pr
    prediction_seqs = []
    for i in range(int(len(data)/prediction_len)):
        curr_frame = data[i*prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
        prediction_seqs.append(predicted)
    return prediction_seqs

def plot_results(predicted_data, true_data):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    plt.plot(predicted_data, label='Prediction')
    plt.legend()
    plt.show()

def plot_results_multiple(predicted_data, true_data, prediction_len):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    #Pad the list of predictions to shift it in the graph to it's correct start
    for i, data in enumerate(predicted_data):
        padding = [None for p in range(i * prediction_len)]
        plt.plot(padding + data, label='Prediction')
        plt.legend()
    plt.show()

print ('Support functions defined')
```

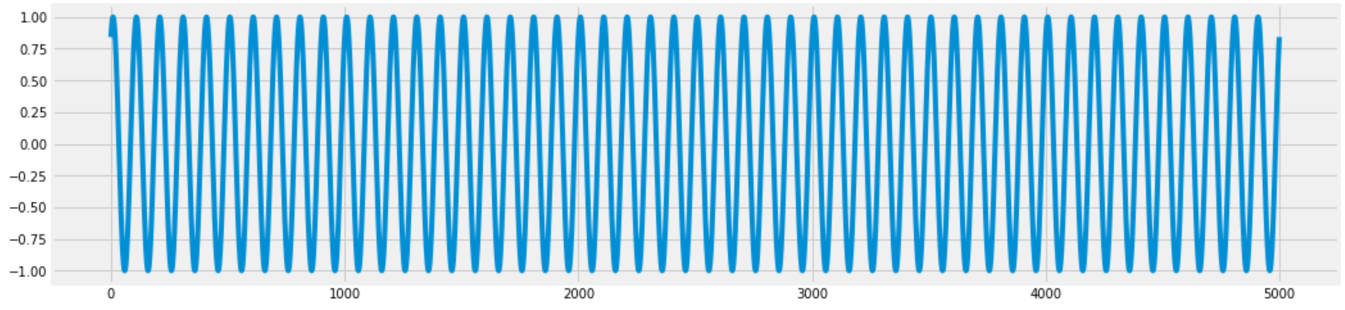
```
     Support functions defined
```

Sinus wave proof of concept

First let's run some tests on a plain sinus wave and see of the Neural Network can predict it right, kind of proof of concept

```
# Load the data
dataset = pd.read_csv('/content/drive/MyDrive/DTS/sinwave/Sin Wave Data Generator.csv')
dataset["Wave"][:].plot(figsize=(16,4),legend=False)
```

```
     <matplotlib.axes._subplots.AxesSubplot at 0x7f1098ac1990>
```



```
# Prepare the dataset, note that all data foer the sinus wave is already normalized between 0
# A label is the thing we're predicting
# A feature is an input variable, in this case a stock price

feature_train, label_train, feature_test, label_test = load_data(dataset, 'Wave', Enrol_windo

print ('Datasets generated')
```

```
     Datasets generated
```

```
# The LSTM model I would like to test
# Note: replace LSTM with GRU or RNN if you want to try those
```

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(feature_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1, activation = "linear"))

model.compile(loss='mse', optimizer='adam')

print ('model compiled')

print (model.summary())
```

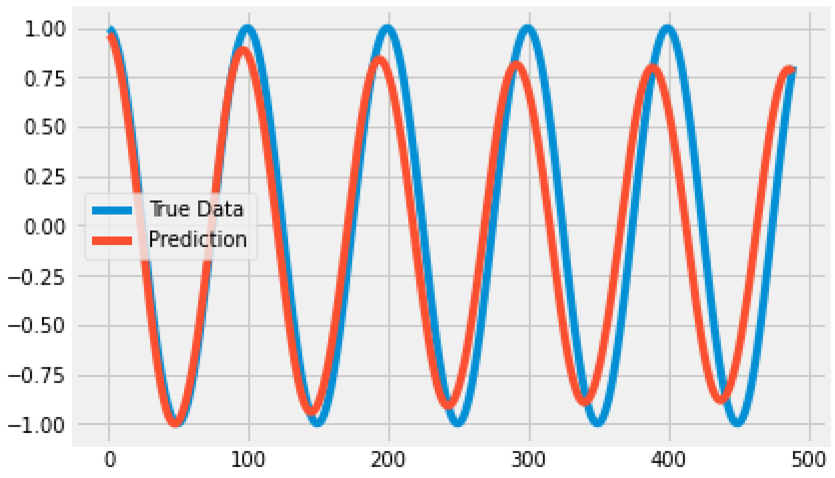
```
     model compiled
     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      lstm (LSTM)                 (None, 100, 50)           10400

      dropout (Dropout)           (None, 100, 50)           0

      lstm_1 (LSTM)               (None, 100)               60400

      dropout_1 (Dropout)         (None, 100)               0

      dense (Dense)               (None, 1)                 101

     =================================================================
     Total params: 70,901
     Trainable params: 70,901
     Non-trainable params: 0
     _________________________________________________________________
     None
```

```
#Train the model
model.fit(feature_train, label_train, batch_size=512, epochs=10, validation_data = (feature_t
```

```
     Epoch 1/10
     9/9 [==============================] - 20s 2s/step - loss: 0.2792 - val_loss: 0.1298
     Epoch 2/10
     9/9 [==============================] - 12s 1s/step - loss: 0.0816 - val_loss: 0.0381
     Epoch 3/10
     9/9 [==============================] - 12s 1s/step - loss: 0.0192 - val_loss: 0.0039
     Epoch 4/10
     9/9 [==============================] - 19s 2s/step - loss: 0.0095 - val_loss: 0.0018
     Epoch 5/10
     9/9 [==============================] - 12s 1s/step - loss: 0.0055 - val_loss: 0.0019
     Epoch 6/10
     9/9 [==============================] - 12s 1s/step - loss: 0.0051 - val_loss: 6.2230e-0
```

```
Epoch 7/10
9/9 [==============================] - 12s 1s/step - loss: 0.0041 - val_loss: 3.0877e-04
Epoch 8/10
9/9 [==============================] - 12s 1s/step - loss: 0.0040 - val_loss: 2.5582e-04
Epoch 9/10
9/9 [==============================] - 12s 1s/step - loss: 0.0037 - val_loss: 2.2033e-04
Epoch 10/10
9/9 [==============================] - 12s 1s/step - loss: 0.0036 - val_loss: 1.7325e-04
<keras.callbacks.History at 0x7f10963fa390>
```

```
#Let's use the model and predict the wave
predictions= predict_sequence_full(model, feature_test, Enrol_window)
plot_results(predictions,label_test)
```



Results

Actually not a bad result because remember the models predicts 500 steps in the future and moreover after the enrol_window length (eg 100 setps) the predictions are being made on predictions, so eny error quickly multiplies ba magnitudes.
Having this confidence let's try to predicts some stock prices

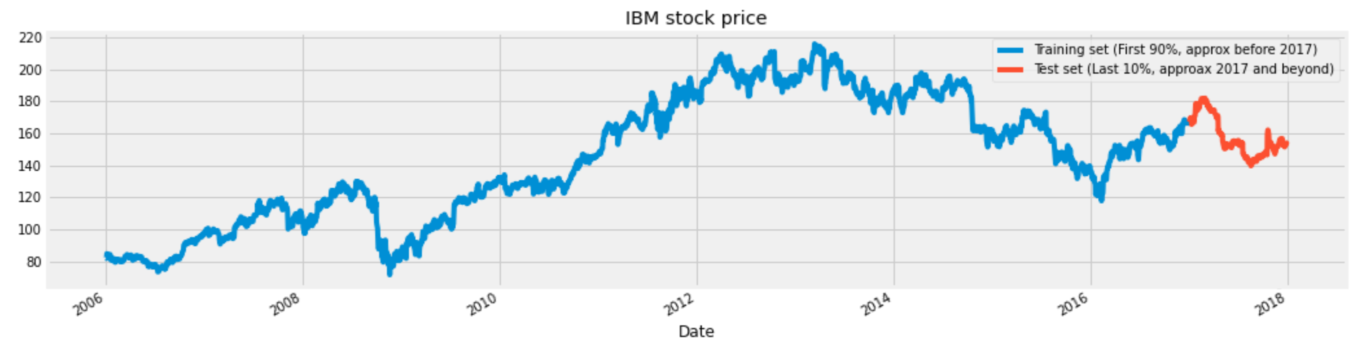
IBM stock prediction

Could we predict stock prices with the neural network? Let's try it on some actual data. The daily stock prices of IBM stock are available for 2006-2017. Let's try to predict that last 10% of the data (approx the 2017 data) based on all the data before.

```
# Let's get the stock data
dataset = pd.read_csv('/content/drive/MyDrive/DTS/DIJA/IBM_2006-01-01_to_2018-01-01.csv', ind
dataset.head()
```

Date	Open	High	Low	Close	Volume	Name
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM

```
# Prepare the dataset, note that the stock price data will be normalized between 0 and 1
# A label is the thing we're predicting
# A feature is an input variable, in this case a stock price
# Selected 'Close' (stock pric at closing) attribute for prices. Let's see what it looks like

feature_train, label_train, feature_test, label_test = load_data(dataset, 'Close', Enrol_wind

dataset["Close"][:'2016'].plot(figsize=(16,4),legend=True)
dataset["Close"]['2017':].plot(figsize=(16,4),legend=True) # 10% is used for thraining data w
plt.legend(['Training set (First 90%, approx before 2017)','Test set (Last 10%, approax 2017
plt.title('IBM stock price')
plt.show()
```



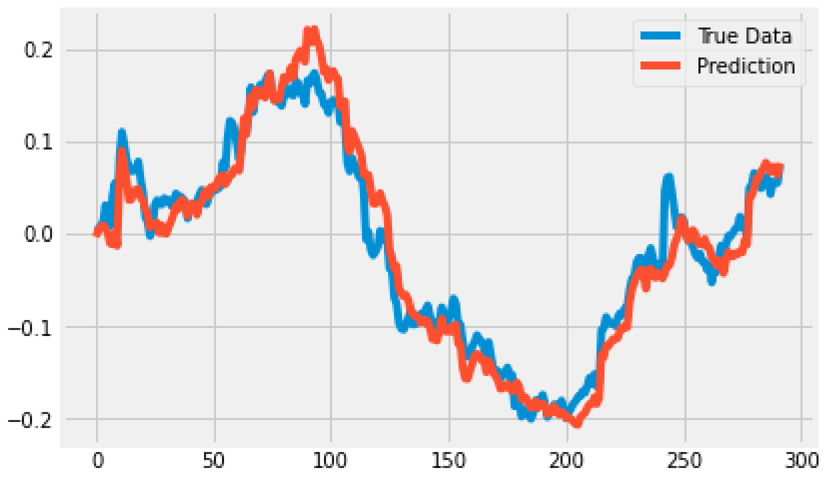
```
# The same LSTM model I would like to test, lets see if the sinus prediction results can be m
# Note: replace LSTM with GRU or RNN if you want to try those

model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(feature_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1, activation = "linear"))

model.compile(loss='mse', optimizer='adam')

print ('model compiled')

print (model.summary())
```

```
     model compiled
     Model: "sequential_1"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      lstm_2 (LSTM)               (None, 100, 50)           10400

      dropout_2 (Dropout)         (None, 100, 50)           0

      lstm_3 (LSTM)               (None, 100)               60400

      dropout_3 (Dropout)         (None, 100)               0

      dense_1 (Dense)             (None, 1)                 101

     =================================================================
     Total params: 70,901
     Trainable params: 70,901
     Non-trainable params: 0
     _________________________________________________________________
     None
```

```
#Train the model
model.fit(feature_train, label_train, batch_size=512, epochs=5, validation_data = (feature_te
```

```
     Epoch 1/5
     6/6 [==============================] - 14s 1s/step - loss: 0.0134 - val_loss: 0.0040
     Epoch 2/5
     6/6 [==============================] - 7s 1s/step - loss: 0.0027 - val_loss: 0.0016
     Epoch 3/5
     6/6 [==============================] - 7s 1s/step - loss: 0.0014 - val_loss: 9.8106e-04
     Epoch 4/5
     6/6 [==============================] - 7s 1s/step - loss: 0.0015 - val_loss: 6.6642e-04
     Epoch 5/5
     6/6 [==============================] - 7s 1s/step - loss: 0.0010 - val_loss: 6.2472e-04
     <keras.callbacks.History at 0x7f10916e7550>
```

```
#Let's use the model and predict the stock
predicted_stock_price = model.predict(feature_test)
plot_results(predicted_stock_price,label_test)
```

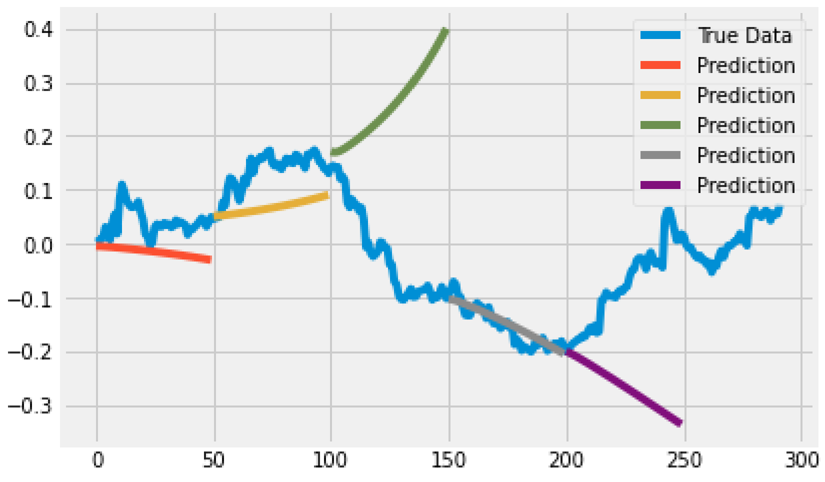


Everybody on Kaggle rich!

This looks incredible correct, but "if it is to good to be true, it is probably not true".
Let's step back and actually see what we did. We created a testset of 100 (dependend on how you set the enrol_window) actual datapoints and ask to predict nr 101 (which is probably anhow close to nr 100). And we did so for each point in this graph. Hence the fantastic result, it wasn't that hard (Remember that you are looking at normalised data)

Like the sinewave example we need to predict a new point based on the actual last 100 points, the next point on 99 actual points and 1 prediction, the next point on 98 actuals and 2 predictions, and so forth.
Lets make some 50 predictions ahead in the future and do this every 50 times to get a bearing how the model predicts

```
predictions = predict_sequences_multiple(model, feature_test, Enrol_window, 50)
plot_results_multiple(predictions, label_test, 50)
```



You can't predict future stock prices on historic data

Ouch, can't use that to put some real money in the stock market. We basically knew already that you cant predict future stock prices on historic data. Pick for example the grey or purple line, it probably learned the stock went down last 100 sequence so it predicts it will go down, what would be a recognizable pattern to predict the trend will break and would go back up again after point 200. So it is nog recognisable in the historic data, else the algorithem would have found it. Maybe with a richer data set with correleated stocks? Other (News?) items? Etc Anyhow, still a nice learning example which helped me to practice with LSTM (but could also picked GRU or RNN, fw simple code changes in the model)