IMPORTANT NOTE

Hello my friends,

In the following tutorials I will be implementing the RNN model on Spyder, which is a user-friendly Python IDE within [Anaconda](https://www.anaconda.com/products/individual).

Spyder is a great IDE but it requires you to install the TensorFlow and Keras libraries first on your machine. On Google Colab you don't have to do that because these libraries (and many others) are already pre-installed. Therefore I'm giving you right here below the Google Colab link of this exact same RNN implementation for this section:

<https://colab.research.google.com/drive/1shvi6Km0x6pqajKCLq39D0TgKO27Iind?usp=sharing>

**My recommendation is the following:**

You should follow the implementation on the video lectures of this section, and in order to avoid the hassle of installing the libraries, I highly recommend to code at the same time on the Google Colab notebook given above (the codes are exactly the same). Just don't forget to create a copy of this Colab notebook because the latter is in read-only mode, by clicking "File" and then clicking "Save a copy in Drive". Then you're good to go!

Also, I am giving you the zip folder of this Part 3 - RNNs, containing the datasets and the codes in both .py and .ipynb formats for this practical activity.

Enjoy this section, and enjoy Deep Learning!

Hadelin

PS: Don't forget to upload the dataset (**both Google\_Stock\_Price\_Train.csv and Google\_Stock\_Price\_Test.csv**) into your Colab notebook by clicking the "Files" button on the left (the button looking like a folder) and then the "Upload" button.

Resources for this lecture

* Part 3 - Recurrent Neural Networks.zip

Evaluating the RNN

Hi guys,

as seen in the practical lectures, the RNN we built was a regressor. Indeed, we were dealing with Regression because we were trying to predict a continuous outcome (the Google Stock Price). For Regression, the way to evaluate the model performance is with a metric called RMSE (Root Mean Squared Error). It is calculated as the root of the mean of the squared differences between the predictions and the real values.

However for our specific Stock Price Prediction problem, evaluating the model with the RMSE does not make much sense, since we are more interested in the directions taken by our predictions, rather than the closeness of their values to the real stock price. We want to check if our predictions follow the same directions as the real stock price and we don’t really care whether our predictions are close the real stock price. The predictions could indeed be close but often taking the opposite direction from the real stock price.

Nevertheless if you are interested in the code that computes the RMSE for our Stock Price Prediction problem, please find it just below:

* import math
* from sklearn.metrics import mean\_squared\_error
* rmse = math.sqrt(mean\_squared\_error(real\_stock\_price, predicted\_stock\_price))

Then consider dividing this RMSE by the range of the Google Stock Price values of January 2017 (that is around 800) to get a relative error, as opposed to an absolute error. It is more relevant since for example if you get an RMSE of 50, then this error would be very big if the stock price values ranged around 100, but it would be very small if the stock price values ranged around 10000.

Enjoy Deep Learning!