POLITECNICO
MILANO 1863

AN2DL – Challenge 2 – Time Series Forecasting

Guadagno Antonio – 10561018
Emanuele Claudio Alfredo – 10682738

The goal of the project

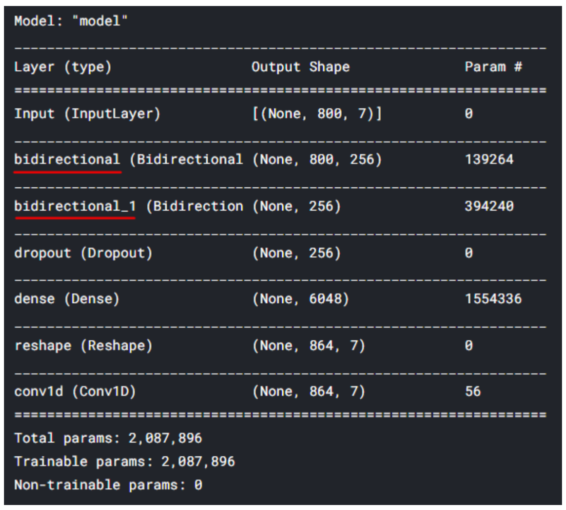
The goal of the project is to design and develop a forecasting model able to predict future samples of a multivariate time series starting from past observations provided in input.

Preliminary note: during the course of the challenge, we've noticed that running the same notebook several times we obtained slightly different results. Searching on the internet, we've discovered that this is due to an intrinsic non-deterministic behavior of the GPUs. For this reason, we have run the notebooks several times trying to get the best results possible.

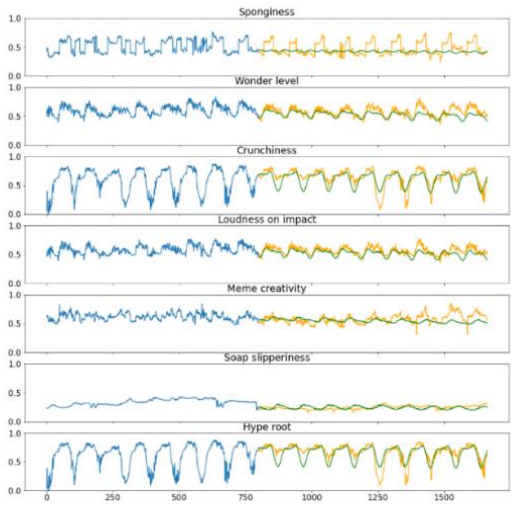
1 – LSTM (No Convolutional) – Direct method

The first model we've tried is characterized by the presence of two **bidirectional LSTM layers** having 128 memory elements each (no convolutional layer). Using this model, we've obtained at the best epoch a val_loss of 0.0110, a val_root_mean_sqaured_error of 0.1048 and a RMSE on Codalab test set equal to 4.312.

In the figures below you can see the structure of the model and a forecast produced by it.



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 800, 7)]          0
_________________________________________________________________
bidirectional (Bidirectional (None, 800, 256)          139264
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               394240
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 6048)              1554336
_________________________________________________________________
reshape (Reshape)            (None, 864, 7)            0
_________________________________________________________________
conv1d (Conv1D)              (None, 864, 7)            56
=================================================================
Total params: 2,087,896
Trainable params: 2,087,896
Non-trainable params: 0
```



Before arriving to the final model, we took advantage of the days in which Codalab was down to do some hyperparameters tuning in order to understand how to improve the model. We've changed a parameter at a time (to perform AB tests) and these are the most relevant results we have obtained:

- Removing bidirectionality: val_loss = 0.0118, val_root_mean_squared_error = 0.1085.
- Reducing the window size from 800 to 400: val_loss = 0.0115, val_root_mean_squared_error = 0.1070.
- Increasing the window size from 800 to 1000: val_loss = 0.0114, val_root_mean_squared_error = 0.1068.
- Adding a 0.2 dropout to LSTM layers: val_loss = 0.0114, val_root_mean_squared_error = 0.1066.
- Decreasing the number of memory elements to 80/80: val_loss = 0.0117, val_root_mean_squared_error = 0.1084.
- Decreasing the number of memory elements to 100/100: val_loss = 0.0112, val_root_mean_squared_error = 0.1059.

Of course, we've not submitted that models for which we have experienced a loss in the performances (higher val_loss and higher val_root_mean_sqaured_error).

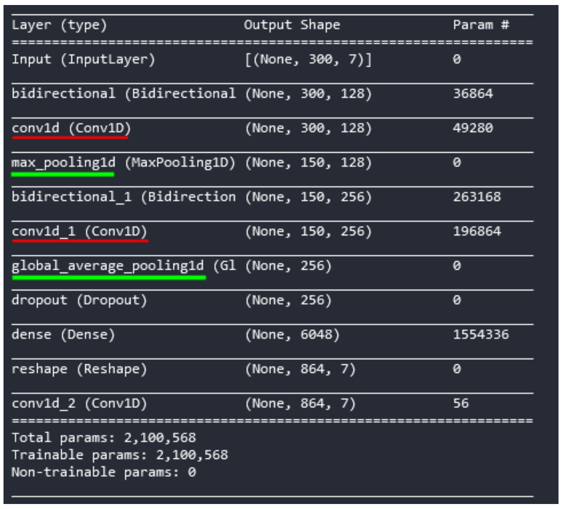
At this point it was clear that to improve performances we needed a different model and so we switched to convolutional LSTM models.

2 – Convolutional LSTM – Direct Method

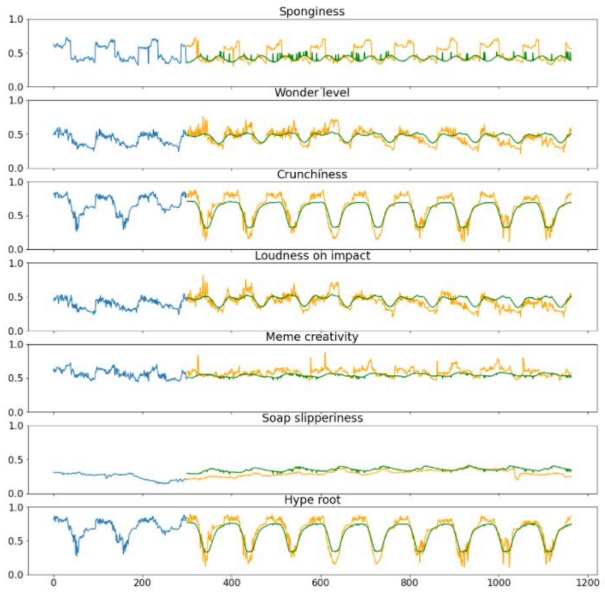
To improve our model, we have introduced

- two 1D-convolutional layers
- one MaxPool1D layer
- one GlobalAveragePooling1D layer

More details are reported in the picture below, anyway the model is very similar to the one seen during the exercise lectures. These changes led to an improvement in results: we obtained a RMSE on the Codalab's test set equal to 3.92 (3.80 considering the final trained on all the data).

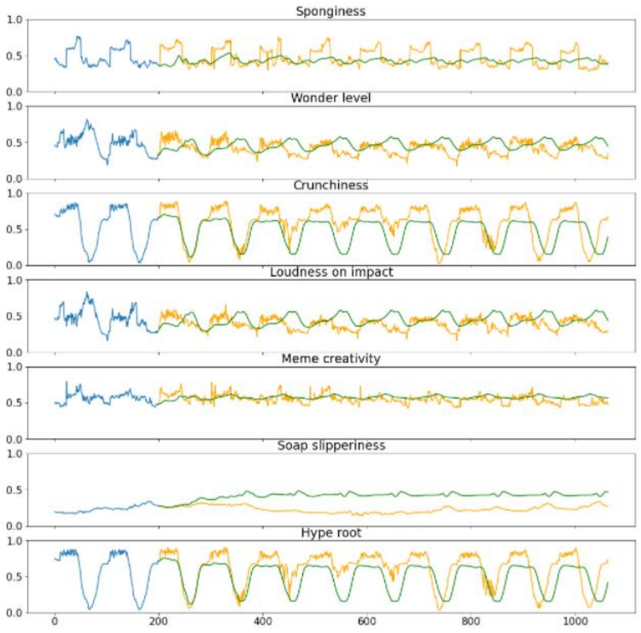


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 300, 7)]          0
_________________________________________________________________
bidirectional (Bidirectional (None, 300, 128)          36864
_________________________________________________________________
conv1d (Conv1D)              (None, 300, 128)          49280
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 150, 128)          0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 150, 256)          263168
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 150, 256)          196864
_________________________________________________________________
global_average_pooling1d (Gl (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 6048)              1554336
_________________________________________________________________
reshape (Reshape)            (None, 864, 7)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 864, 7)            56
=================================================================
Total params: 2,100,568
Trainable params: 2,100,568
Non-trainable params: 0
_________________________________________________________________
```



This time we have obtained better performances by introducing **recurrent dropout** in LSTM layers, which however has also led to a considerable increase in the time required for the training.

3 – Convolutional LSTM – Autoregression method



In addition to the direct method presented above, we have also tried the autoregressive approach.

The results that we have obtained with this technique are, on average, worse than those obtained with the direct method.

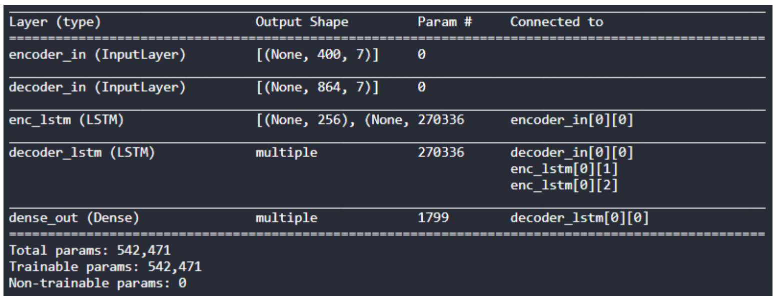
The reason probably is that the error made on a single prediction is then amplified when we use it for the following forecasting.

To make a comparison, using the Convolutional LSTM model presented above and the autoregressive approach (instead of the direct method) we have obtained a val_loss equal to 0.0011 on the single prediction which resulted in a RMSE of 5.257 on Codalab's test set (instead of 3.92, result obtained with the direct method).

Here on the left you can see the plot of the predictions that we have obtained.

4 – Seq2Seq Encoder/Decoder

After performing some hyperparameters tuning also on the Convolutional LSTM model presented above, we were unable to obtain consistent improvements, so we have decided to try completely different models. We have developed a Seq2Seq model with a LSTM encoder and a LSTM decoder. Below you can see the graph of the components that we obtain calling the function *"tfk.utils.plot_model"*.



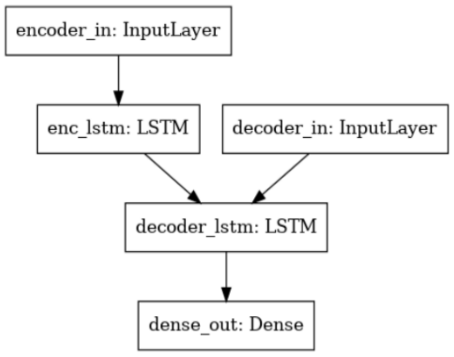
```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
encoder_in (InputLayer)         [(None, 400, 7)]     0

decoder_in (InputLayer)         [(None, 864, 7)]     0

enc_lstm (LSTM)                 [(None, 256), (None, 270336      encoder_in[0][0]

decoder_lstm (LSTM)             multiple             270336      decoder_in[0][0]
                                                                 enc_lstm[0][1]
                                                                 enc_lstm[0][2]

dense_out (Dense)               multiple             1799        decoder_lstm[0][0]
==================================================================================================
Total params: 542,471
Trainable params: 542,471
Non-trainable params: 0
```



We have a connection between *enc_lstm* and *decoder_lstm because* the decoder takes in input the encoder state.

To perform the training, we needed to slightly modify the function "*build_sequences*":

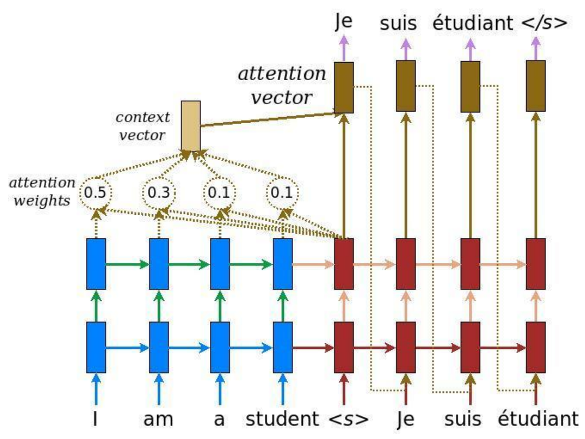
- the **encoder** takes in input data inside the window [idx , idx+window]
- the **decoder** takes in input data inside the window [idx + window **- 1** , idx + window + telescope **- 1**] so that the last point read by the encoder is also the first point read by the decoder
- the **desired output** is contained in the window [idx + window , idx + window + telescope]

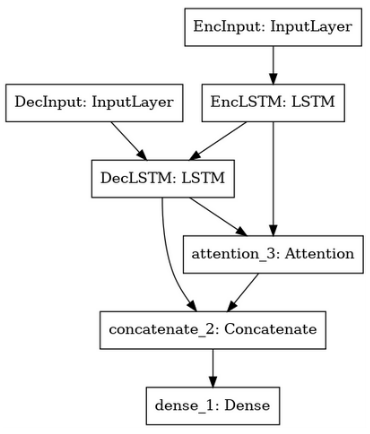
These changes are due to the fact that, working with time series, we cannot use tokens <START> and <END>. To make the final predictions we needed to create a new decoder modifying the structure of the first one. The new decoder takes in input:

- **for the first prediction**: the encoder state and the last data read by the encoder itself
- **for predictions after the first one**: the previous prediction and its own state computed during the previous iteration

5 – Seq2Seq Encoder/Decoder + Attention

Starting from our Seq2Seq model we have also tried to implement the Attention mechanism. Below you can see a drawing of the model we referred to (presented during the last theoretical lesson) and the model we have obtained. The decoder takes in input the encoder state, the attention takes in input both the decoder and the encoder outputs, the result of the attention is then concatenated with the result of the decoder. We needed to use an autoregressive approach and a **RepeatVector** for the output of the decoder, anyway we have encountered an error in Kaggle that didn't allow us to perform a good training. Unfortunately, we have been not able to solve it.


Je
suis
étudiant
</s>
attention vector
context vector
attention weights
0.5
0.3
0.1
0.1
I
am
a
student
<s>
Je
suis
étudiant


EncInput: InputLayer
DecInput: InputLayer
EncLSTM: LSTM
DecLSTM: LSTM
attention_3: Attention
concatenate_2: Concatenate
dense_1: Dense