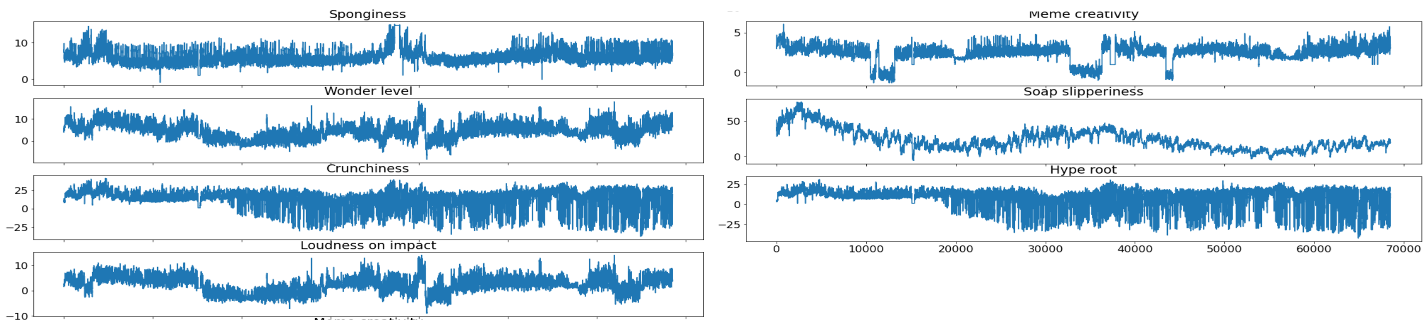
Homework #2 - Report

I. Introduction

A. Classification Problem

The problem was a *multivariate time series forecasting* problem where we were required to provide a prediction for each time step in the test prediction window.
As a first thing we can take a look at the dataset and the 7 parameters we will have to predict for the time series:



II. Training phase

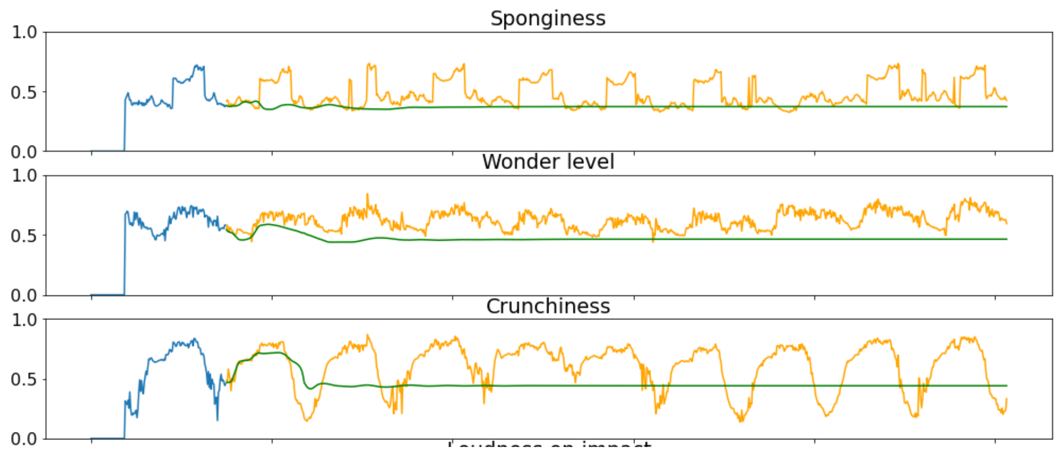
After the brief introduction to the problem we can start to take a look at the main body of the challenge that was the training of our neural network.
To do this we made use of Tensorflow and Keras for what regards the model building part, while for what regards the dataset and its management we made use of Pandas.
As it regards the data, we normalised the dataset in almost all of the different trials in order to fasten the training and to regularise the weighting , one time we tried to use the raw data but with bad results.
We also tried to standardise, with bad results, but this could be because of one error that we were not able to find in our standardisation.
Another thing we tried to do with the dataset was to omit a first part of the dataset (19000 timestamps more or less), where, visualising it, you could see a particular behaviour especially in the Hype root and Crunchiness columns. This trial was not successful, in fact it worsened our results.

A. First approach

As a first way to approach the challenge we simply used the model we viewed in class during the exercise session, both for Direct Forecasting and Autoregressive Forecasting, without changing anything but some modifications to match the dataset we were given during the data handling and obviously the telescope value. So, it was composed of two *Bidirectional LSTM* layers, the first followed by a Convolution layer and a MaxPool layer, the second one the output layer.
This model, as is, gave us a score of around 5 in the test set on Codalab, a score that we thought could be improved a lot with some adjustments to the training parameters, tunings that will be discussed in the following paragraphs.

B. Autoregressive Forecasting

As we previously mentioned we also tried to have an *autoregressive* approach that we trained in parallel with the *Direct* one, but in all the cases the autoregressive one gave out strange results.
More in depth, at first sight while training the network the validation loss given during the training of the model seemed better than the direct one, but when looking at the graphs trying to predict the test set there was a strange progressive flatness to the results of which you can see an example in the following image:



C. Direct Forecasting

In this part we will talk about the two main approaches that we carried on during the Training phase, and on which we developed in order to try to improve the performances of our model.

1. One approach that we made to try to improve it, was to change the output layer with a dense layer with a linear activation function since the problem is of regression.
   Keeping this output layer,the reshape layer before it and one dense layer, we tried to combine them with different layers.
   The models that we tried were six in total and they were: one composed by two LSTM layers, one composed by two bidirectional LSTM layers, one by two GRU layers, one by a single LSTM layer or by a single bidirectional LSTM layer or with a single GRU layer. Between these similar models, results pointed out that in general one layer models were better than the corrispective two layers models. Furthermore, the single layer LSTM model had better results than the bidirectional LSTM one and the GRU one was the one that performed worst. The best model generated from this approach reached a score of **4.06.**
2. The other approach was more relying on the model we saw in class, composed of some Bidirectional LSTM layers, followed almost always by a Pooling layer and a Convolutional Layer, followed by a Dropout Layer, and finally a Dense layer followed by a Reshape and Conv1D layer. While training we applied EarlyStopping and ReduceLROnPlateau as callbacks. The best model generated from this approach reached a score of **4.00.**

D. Tuning

a. Window and Stride

By fine tuning, we found two combinations of windows and stride that produced satisfying results. The first one was with a window of 150 and stride of 3 and the second one with a window of 500 and stride 20. While the first always had great results in both the models, the second one performed poorly in the second model.

b. Batch Size and Epochs

Same as before by fine tuning, we found the best batch size for our model to be around 32, lesser values or higher ones tended to produce spikes in the prediction that led to unnecessary losses that had a bad impact on our score.
As it regards epochs, it depended a lot on the way our model was built but didn't have much of an impact, in the way that a slower model needed more epoch, a faster one less epochs, but both the ways led to almost the same result.

E. Model

a. Layers

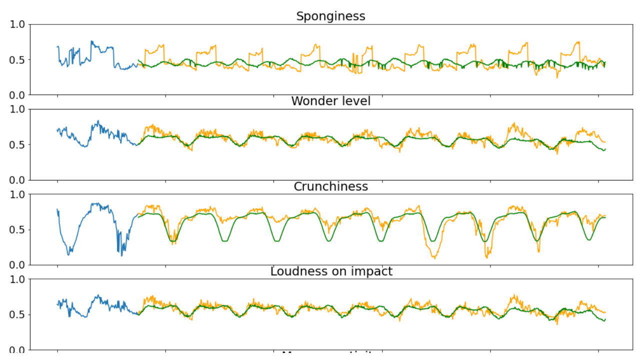
The first approach is the simplest one because we found out that on multiple occasions, by changing the layer into simpler ones, the performance was increasing. Then the same happened also by stripping the model of layers, leading to the one that we got.
The second approach instead went down many roads that more or less worsened the results of the most simple one. At first we tried to increase the number of LSTM layers or the number of neurons. In this case the training time increased and it produced a bigger neural network that in fact resulted in worse and worse performances the more we increased its size, so, more was less in this case. For this reason we also tried to implement Dropout layers and l2-norm to see if the addition of these could regularise the results, but they didn't have much effect.

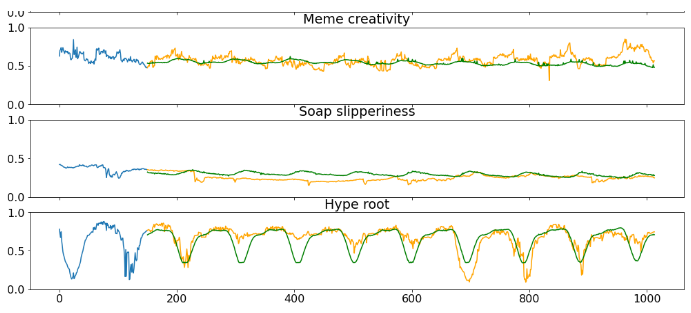
b. Attention

We also tried to implement the Attention mechanism, using Additive Attention, but in this case it didn't have much of an impact over the performances of the model, producing the same results, attesting also at 4.10 more or less.

III. Conclusions

The classes with the worst score were crunchiness and hype root. This led us to believe that maybe by doing some operations of standardisation other than normalisation could have led us to better results, but we didn't find a way to effectively do this. These could be also solvable by having more than one model, in order to learn *crunchiness* and *hype root*, giving more importance to them in one model and training the others in another one, and then combining the results.


Sponginess
Wonder level
Crunchiness
Loudness on impact


Meme creativity
Soap slipperiness
Hype root