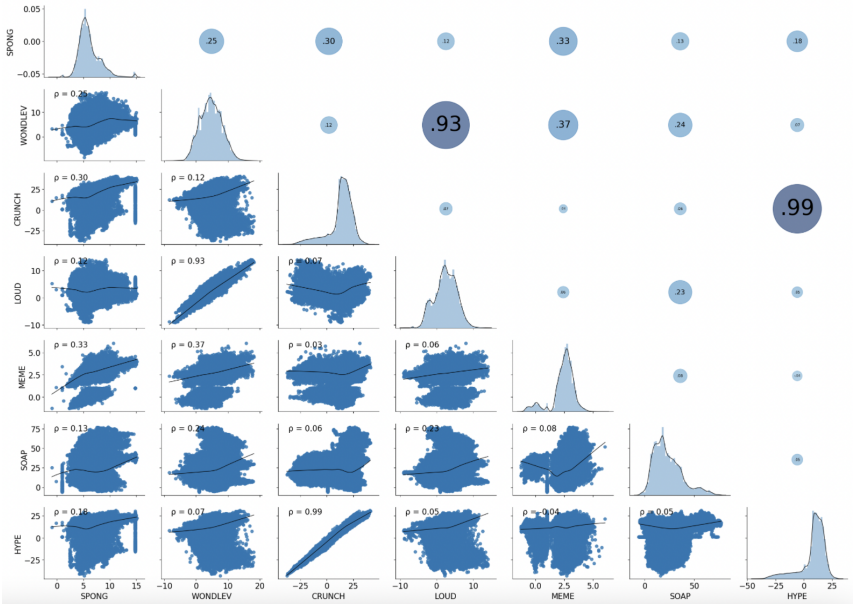
REPORT ON TIME SERIES FORECASTING

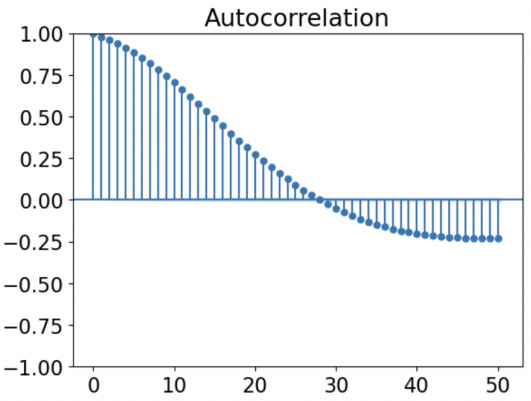
We started our second journey applying what we saw during classes. First of all we analyzed the dataset provided and we discovered that it consisted of 68528 values for each of the 7 attributes: Sponginess, Wonder level, Crunchiness, Loudness on impact, Meme creativity, Soap slipperiness and Hype root.

Data Analysis

As a first step we start analyzing the data given. Specifically, our objective was to detect possible relation between the time series and themselves: we calculated the *Correlation coefficient p* between each pair of variables.




```
plot_acf(stocks['CRUNCH'],alpha=0.5,lags=50)
plt.show()
```



We noticed an interesting correlation between Hype root - Crunchiness and Loudness - Wonder Level. In addition, we plotted the *Autocorrelation* function and the *Partial Autocorrelation* for each variable.

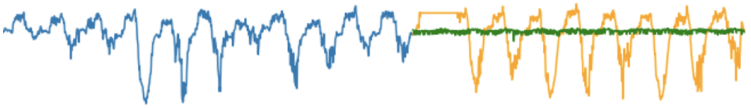
First phase (RMSE → 7.97)

Before using the *model* seen in class we inspected graphically the *dataframe* and divided the entire *dataset* in training set and test set. We noticed that the *features* were on different scales hence we normalized bringing them all into the same range [0, 1].

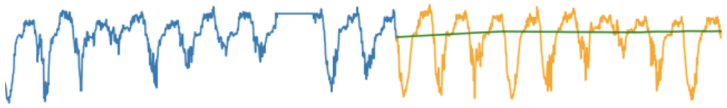
We used the same model for both *direct* and *autoregressive*, as seen in class.

When we were asked to predict the next values that would be evaluated on the hidden test set we paid attention in making the same *preprocessing* and *postprocessing* of the data used. Obviously, the window size used during training is the one we always used for making predictions.

Direct Forecasting



Autoregressive Forecasting



Results

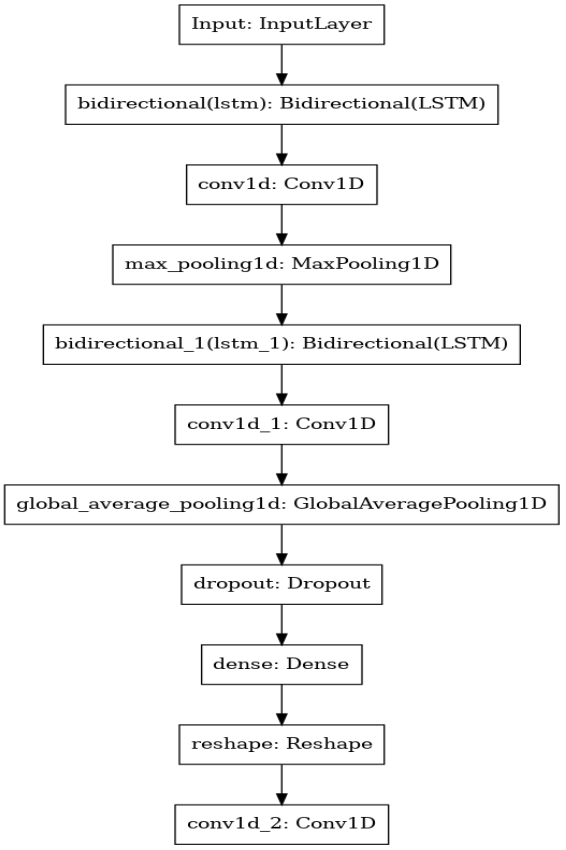
The forecast looked almost flat. There was something wrong in the way we used the data. In fact, they show seasonality that we had not yet exploited because our window size hadn't a particular logical reason, just a tentative dimension of 1700.

As soon as we changed the window size to 300, finally the prediction (green line in the previous images) started to follow the real future data in a curved manner.

Code for reproducibility:

https://www.kaggle.com/marioroca/crazyfeatureschallenge

https://www.kaggle.com/marioroca/window-300



Second phase (RMSE → 4.58)

In order to improve the forecasting, we started changing the shape of the model considering: three *bidirectional LSTM* in a row followed by a *Dropout*, a *GAP*, a *Dense Layer* and a final *Reshape* with a *Convolutional Layer.* In that phase we tried to tune the best telescope and window parameters. We changed the *metrics* and introduced parameters like the *learning rate* for the *loss function*.
The direct forecasting reached an RMSE equal to 4.58 while the autoregression seems still inefficient. We thought of using a telescope different from 1 and we started inserting different telescope measures like 8,12 and 16 (dividers of 864).

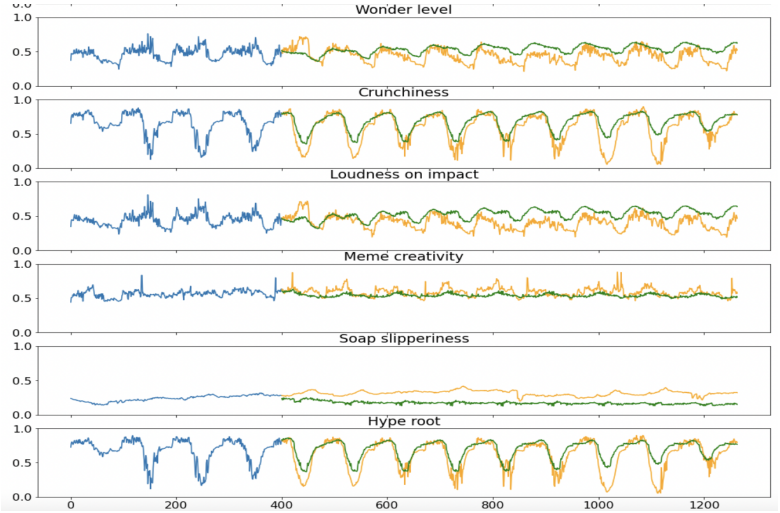
KerasTuner Implementation

KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. We introduced it both in the Direct forecasting and Autoregressive forecasting. We defined the search space by tuning the number of units in the *LSTM* layers and *Conv1D*. We chose an integer hyperparameter with *hp.Int('units', min_value=32, max_value=512, step=32)*. After defining the search space, we needed to select a tuner class to run the search: we used *Hyperband* as Tuning algorithm. Then, we started the search for the best hyperparameter configuration: here we also defined an *EarlyStopping* function preventing the Tuner from overfitting.

```
def build_CONV_LSTM_model(hp):

    # Build the neural network layer by laver
    input_layer = tfkl.Input (shape=input_shape, name='Input')

    units1 = hp.Int('units1', min_value=32, max_value=512, step=32)
    lstm=tfkl.Bidirectional(tfkl.LSTM(units=units1,return_sequences=True))(input_layer)

    units2 = hp.Int('units2', min_value=32, max_value=512, step=32)
    lstm=tfkl.Bidirectional(tfkl.LSTM(units=units2,return_sequences=True))(lstm)

    units3 = hp.Int('units3', min_value=32, max_value=512, step=32)
    lstm = tfkl.Bidirectional(tfkl.LSTM(units=units3,return_sequences=True))(lstm)

    units4= hp.Int('units4', min_value=32, max_value=512, step=32)
    lstm = tfkl.Conv1D (units4, 5, padding='same', activation='relu') (lstm)
```

All the best arguments were passed to *model.fit()*. We exploited the KerasTuner also for the probability of the *Dropout* and the *Learning rate* of the loss function.

Autoregressive Forecasting - Results

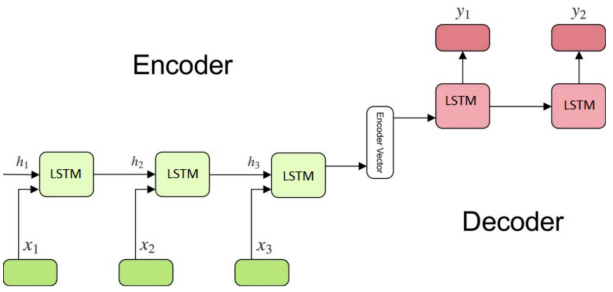


Code for reproducibility:

https://www.kaggle.com/lorispanza/autoregressiontuner

Third Phase (RMSE → 3.61)

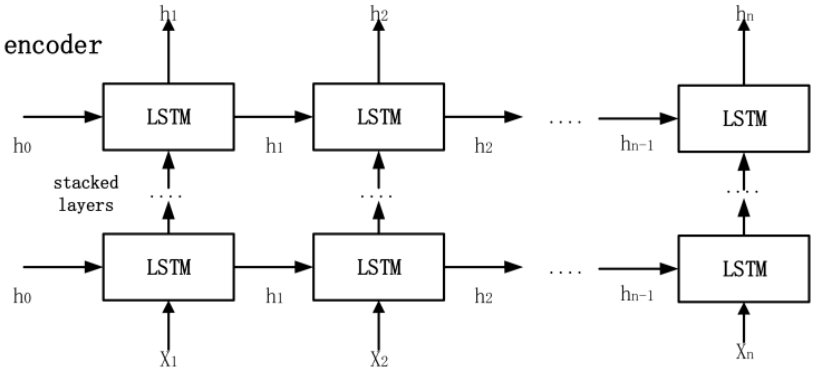
Encoder-Decoder



In the third phase we decided to use a *Seq2Sseq* model implemented via an Encoder-Decoder architecture. Both the *Encoder* and the *Decoder* are LSTM modules. The encoder reads the input sequence and produces a *hidden state* vector that summarizes all the relevant information. We discarded the outputs of the encoder and only preserved the internal states. The initial states of the Decoder were initialized to the final states of the Encoder. Using these initial states, the decoder started generating the output sequence.

In this phase we tested again different combinations of *window-stride-telescope*. We tried to exploit better the *seasonality* of the sequences starting from a window of 300 that seems to be the time in which the data form 3 complete cycles, to 2025. The most interesting seems to have been 1200 and 1500. With these values fixed we tried different telescopes. In particular we tested the telescope dimension: 8, 16, 32, 108, 144. The combination (window: 2025, telescope: 108) allowed us to reach the very first good result with a RMSE value of 3.97 and the combination (window: 1200, telescope: 144) brought us to a RSME value of 3.71, both with a stride of 10.

Our best model: stacked Seq2Seq



We improved even more the results obtained with the standard Seq2Seq using a *stacked* version of it. In particular we stacked together two LSTM both in the Encoder and in the Decoder.

This model allowed us to reach our best performances for this challenge with a RSME value of 3.61.

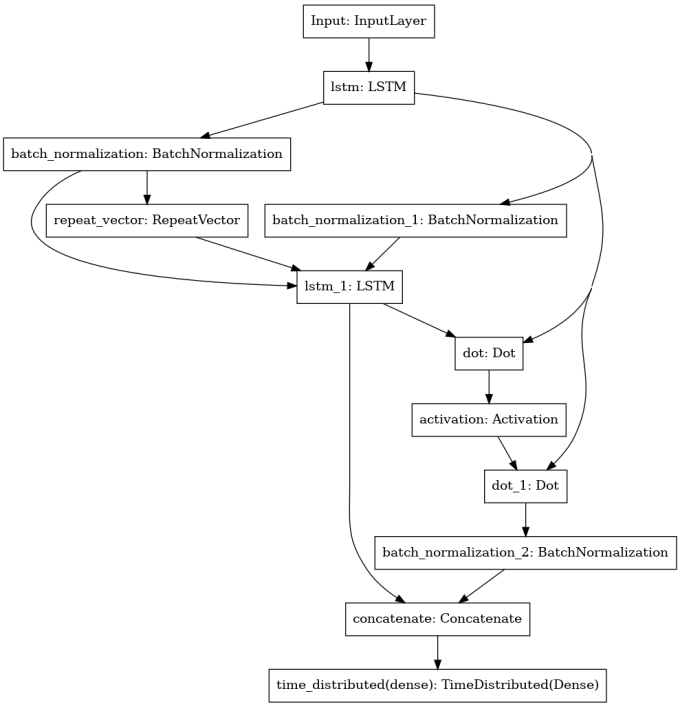
Code for reproducibility:

https://www.kaggle.com/marioroca/seq2seq

Fourth Phase - Seq2Seq with Luong Attention

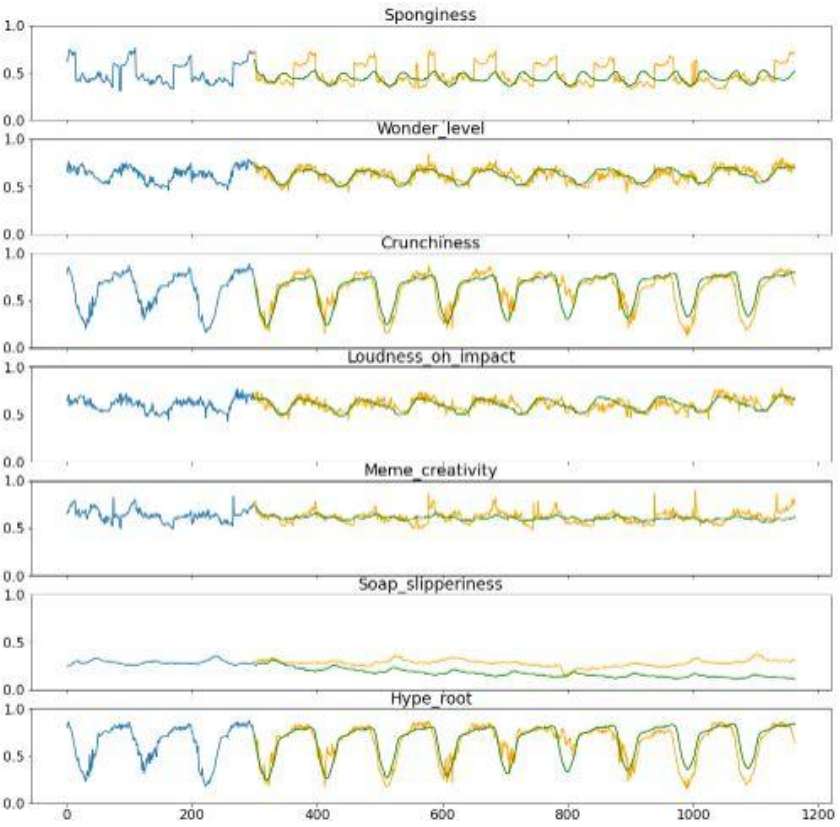
In the last days of the competition window, when we already reached satisfying results, we wanted to try to keep improving our network so we tried to implement a *Seq2Seq LSTM* Model with *Luong Attention.*

Data preprocessing didn't change because in light of all attempts done we decided to keep the one that gave us the best results in the previous phases. Big changes happened in the model, where we added several layers to implement attention.
From the encoder *LSTM* layer we derived the last hidden and the last cell states but also the stacked hidden state for the *score* calculation. *Batch normalization* layers are used to avoid *gradient explosion* problems because of the *LSTM "Elu"* activation function.
After the decoder *LSTM* we built the attention layer, where the first thing to do was compute the *attention scores* (thanks to the "Dot" layer) and apply a *softmax* activation function over it.
Then, we calculated the context vector with the other *"Dot"* layer and, at the end, we concatenated the *context vector* and the decoder *stacked hidden states* to use this result as input to the last *"Dense"* layer.



Autoregressive Forecasting - Results

We made a lot of attempts to find the best window (300), stride(5) and telescope(16) values before getting good results, at the end the best submission score on Codalab with this type of model was 5.00 of RMSE. So, in conclusion, we also experimented with this kind of model but it didn't overcome the other results obtained before.



Code for reproducibility:

https://www.kaggle.com/andrearazza/2nd-challenge

Sitography & Bibliography

- https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
- https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc
- https://towardsdatascience.com/encoder-decoder-model-for-multistep-time-series-forecasting-using-pytorch-5d54c6af6e60
- https://pradeep-dhote9.medium.com/seq2seq-encoder-decoder-lstm-model-1a1c9a43bbac
- https://levelup.gitconnected.com/building-seq2seq-lstm-with-luong-attention-in-keras-for-time-series-forecasting-1ee00958decb
- Z. Gao, W. Dong, R. Chang and C. Ai, "The Stacked Seq2seq-attention Model for Protocol Fuzzing," 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 2019, pp. 126-130, doi: 10.1109/ICCSNT47585.2019.8962499.