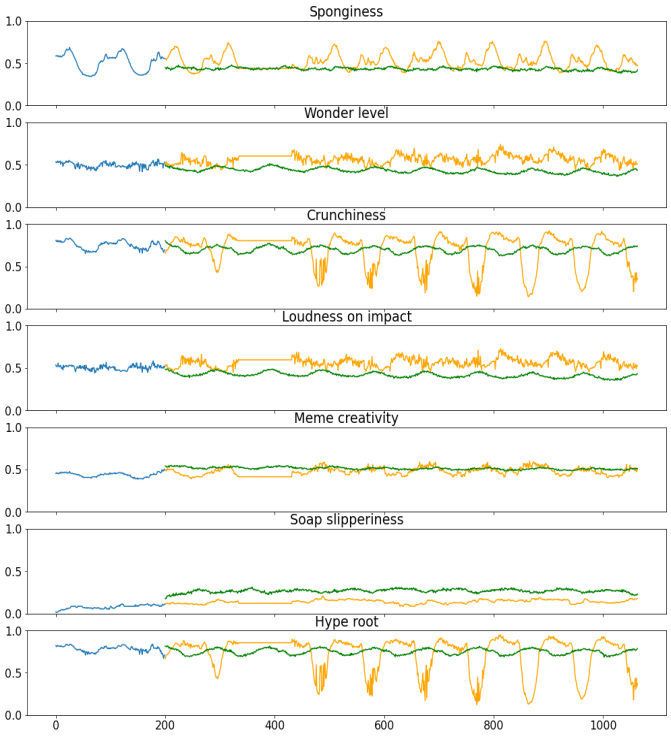
Second challenge: Time Series Forecasting

Marco Bonalumi
Pasquale Dazzeo
Riccardo Ambrosini Barzaghi

0. Introduction

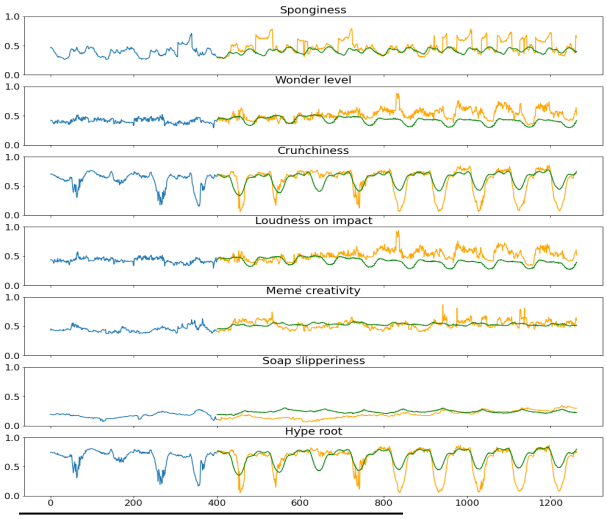
The second challenge required to forecast the future values of a multivariate time series, minimizing its Root Mean Squared Error and its Mean Absolute Error.

1. First steps



We began the challenge by submitting the model seen during the exercise sessions in order to have a starting point: however, it scored poorly, so we began experimenting with various models and techniques to achieve a better submission score.
We started from stacked LSTMs, which had better performance locally with respect to the exercise session model but scored poorly in submission. Below there is an example of prediction with a stacked LSTM with two layers (both 512 units, 0.3 dropout). Later we tried 2 stacked GRU 128 units each with 0.2 dropout and it scored 0.0128 locally on the test set, 19 on CodaLab.
Since we had a submission RMSE around 14, we decided to try bidirectional LSTMs and achieved better results locally : in parallel, we also started trying an AutoRegressive approach (training the model with a telescope of 1 and forecasting one sample at a time, concatenating each new prediction in the values to predict up to the requested telescope). The best results were a biLSTM with a single layer and 256 units, with the AR approach, which scored on the submission 14.11 and a biLSTM, single layer with 600 units, without the AR approach, which scored 14.82. The last picture shows the prediction of the first biLSTM, which had better local accuracy.

2. The turning point



Later on, we found that our submission file (model.py) had some errors which worsened the score, so we tried resubmitting all the models developed until then. Unsurprisingly, the best biLSTM scored around 5.34 : however, we were puzzled by the fact that also simple LSTMs with a small number of units had good scores locally, so we also tried submitting them. We found that the simpler models performed better than the more complex ones, so we started tuning the hyperparameters of those models. Without tuning, we submitted a LSTM with 200 units, scoring 4.41 : this submission also confirmed that the models without AR were outperforming the others, so we abandoned the AR approach.[1]

[1] as suggested in A Comparative Analysis of the ARIMA and LSTM Predictive Models and Their Effectiveness for Predicting Wind Speed, Elsaraiti, Merabet, 2021 and in Time Series Analysis and Modeling to Forecast: a Survey, Dama, Sinoquet, 2021

3. Keras Tuner usage and best models

```
Results summary
Results in tuning/neurons-dropout-activation
Showing 10 best trials
Objective(name='val_mse', direction='min')
Trial summary
Hyperparameters:
neurons: 128
dropout_rate: 0.2
stacked: 1
act: selu
Score: 0.011057048104703426
Trial summary
Hyperparameters:
neurons: 64
dropout_rate: 0.1
stacked: 1
act: selu
Score: 0.011119414120912552
Trial summary
Hyperparameters:
neurons: 64
dropout_rate: 0.3
stacked: 1
act: selu
Score: 0.011261108331382275
```

We wrote several tuners trying to find the best combination of hyperparameters for our models.
Our procedure consisted in finding the best hyperparameters produced by the tuner, decreasing the interval of values around the numbers resulting from the previous run of the tuner and running it again to find the best possible values; these values were then fed to the actual model in order to train the parameters for more epochs and if the result was satisfying we submitted it to Codalab
Firstly, we tried to look for the correct number of units for 2-layers LSTM and 2-layers GRU, also having the dropout rate as a variable. Then we tried with bidirectional models and with a variable number of layers, up to 5 but we obtained the best performances with 2 or 3 level deep networks.
We also tuned the window size and the stride, starting from the best models out of the Keras tuner and even the batch size, but this path revealed to be inconclusive, because the actual models that were produced did not have a particularly better performance once fully trained. Other attempts consisted in trying other activation functions ('selu' was the best one) and a different optimizer (the best we could find was Adam with Nesterov momentum, however it didn't change much so we stuck with Adam). Two of the best LSTMs deriving from the optimization with Keras Tuner scored around 3.93, resulting in our best submissions.

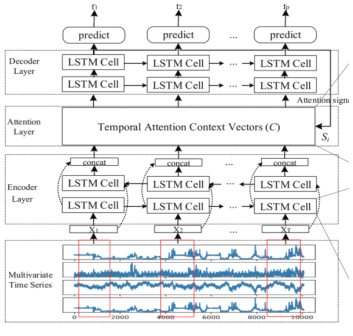
Example of the output from a run to find the best parameters for GRU layers: number of GRUs, number of units per GRU, dropout value and activation function.

4. Advanced architectures (Attention Mechanism, Transformers)

While putting our effort into optimizing our best models using LSTMs, we also wrote a couple notebooks that make use of Attention Mechanism.

The first working model obtained 5.9 RMSE on codalab. The model was simply made by two **Multi Head Attention blocks**, each block composed by: Multi Head (self)Attention with 8 heads and key_size of 100 (same as query and value size) + Dropout + LayerNormalization, followed by a GlobalAveragePooling (in hindsight a bad modeling decision) and the output section. It had a big window of 6912 (864*8) and a telescope of 864. Also in all the attention models we used a stride equal to the telescope, since the results were better.

The second attempt was on a **DA-RNN**[2] model, that should have helped in catching the long-term temporal dependencies within the time series, as stated in the paper. Unfortunately we obtained no significant results.

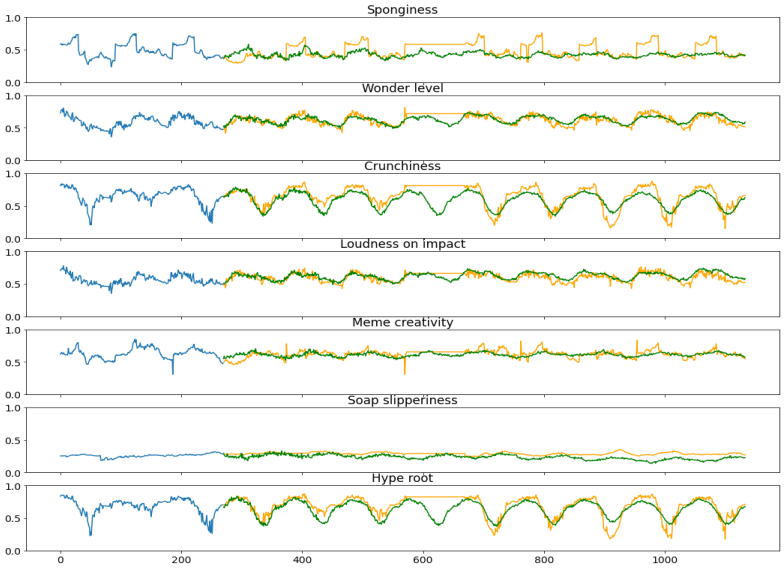


The third attempt was an **LSTM Encoder-Decoder** with Multi Head Attention blocks applied on the latent representation of the time series[3]. The best model of this kind, obtaining 4.1 RMSE on codalab, had as encoder and decoder one LSTM with 200 units, and in between two "Transformer Encoder blocks"[4], i.e. a MultiHeadAttention block (with 4 heads and key_size of 200) with skip connection (Add and LayerNormalization) and a position-wise

[2] as in A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction, 2017, Qin et al.
[3] as in Multivariate time series forecasting via attention-based encoder–decoder framework, 2020, Du et al.; Note also the representation of the architecture in the image.
[4] as in Attention Is All You Need, 2017, Vaswani et al.

Feed Forward block (Dense, ReLU activation, Dense) with skip connection (AddNorm as the previous).
It was also the first model (not considering the autoregressive ones) to predict less than the full telescope in one go. It had a window of 1296 (864*2) and a window of 432 (864/2), two predictions concatenated gave the result.
In the image is shown a graphic representation of the model, taken from the paper.

The fourth and last attempt was an actual **Transformer Encoder-Decoder** network[5]. Since the discrepancies with text analysis, for which transformer networks are most used, we just built the network's encoder blocks and output the forecast after those. The blocks are the same as in the LSTM Encoder-Decoder, but using Conv1D instead of Dense layers in the position-wise Feed Forward block.
The current best model has 1 encoder block, 4 heads and key_size of 32 for the Multi Head (self)Attention, it has a window of 216 (864/4) and a telescope of 54 (864/16).
Another parameter tuned in this phase is the dimension of the first Conv1D layer in the position-wise FF block: while the second Conv1D must have the same dimension as the output of the Multi Head Attention (since they must be summed together), the first one is tunable. The best value, calling X the dimension of the second Conv1D layer, turned out to be X/3, while also X/2 and X/4 were tested in different configurations.



The best transformer obtained 4.2 RMSE on Codalab (on jan, 15th), despite wonderful local results, and after investigating a bit on the possible causes we've come to the conclusion that the test set was too little.

Finally, we also tried adding a **Time2Vec** layer, concatenating a few values learned from the time series to help with the recognition of characteristic sinusoids, obtaining a slight improvement, down to 4.1 RMSE on Codalab.
Unfortunately we had no time left to try to further improve the model.

One final remark: during this phase, the loss function used was the Huber loss, but the best transformer model used the MSE, as all the models before working with Transformers.

[5] as in Attention Is All You Need, 2017, Vaswani et al.