Time Series Forecasting - Report

Gattinoni Valentina 10583105
Manzoni Claudio 10580671
Recalcati Andrea 10578352

Artificial Neural Networks and Deep Learning

January 22, 2022

---

1 Objective

The goal is to create an architecture able to predict future samples of a multivariate time series, exploiting the learning of sequences' past observations. The implementation is done with a Neural Network.

2 Dataset and Preprocessing

We were given a training set, past observations of a multivariate time series with the following characteristics: 7 features (*Sponginess*, *Wonder level*, *Crunchiness*, *Loudness on impact*, *Meme creativity*, *Soap slipperiness*, *Hype root*) and 68528 samples (dataset's length). No automatic validation set was provided. The test set was unknown (accessible only by submitting trained models on *Codalab*).
Before sarting, we created an **overview** of the dataset (see the notebook `Dataset_Inspection.ipynb`) in order to analyze its main features. By way of example here we report the trend of some samples of the multivariate time series.

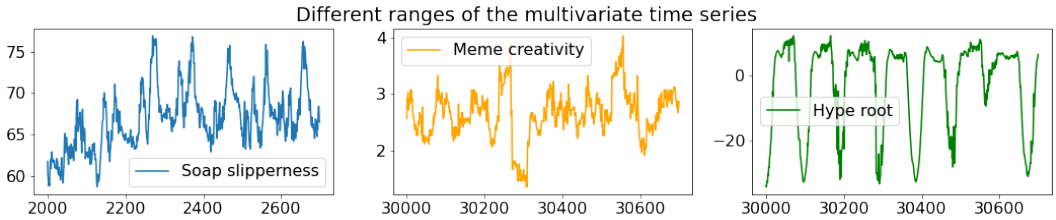


Figure 1: Normalization is needed!

By looking to *Figure 1*, some pre-processing is needed: in fact the different components of the multivariate time series have a completely different range of values. That is not good since variables that are measured at different scales do not contribute equally to the model fitting and might end up creating a bias. We therefore decided to normalize input variables using the so called the *MinMax Scaler*, transforming all features into the range [0,1].

Furthermore, we divided the entire dataset in *training* and *test* set with a 0.9/0.1 ratio. Even if an unknown test set was available on *Codalab*, we created our test set in order to have a visual representation of the *goodness of fit* of our predictions. Then, during the training phase, we took apart another 10% of training data for *validation*. Notice that data have not been shuffled before splitting. This for two main reasons: first it ensures that chopping the data into windows of consecutive samples (as will be done by the `build_sequences` function) is still possible; second because guarantees that test results are more realistic, being evaluated on the data collected after the model was trained.

Finally we noticed other two things regarding our dataset. First, when dealing with multivariate time series, it is undoubtedly interesting to study the correlation between the different features: *Crunchiness* and *Hype Root* are 99% correlated, while *Wonder level* and *Loudness on impact* display 93% correlation. In the *Developement* section we will explain the benefits that this analysis can bring. Second, in some parts of the dataset the values are suspiciously stationary and in certain points constant.

3 Development

We started by implementing a simple architecture for autoregressive prediction and we used this simple net to study some important **hyperparameters** of a time series forecasting problem: window, telescope and stride. Regarding this hyperparameter tuning, we tried to improve the performance on the *Codalab* test set with a *Trial and Error* method. After this first development stage we concluded that good hyperparameters are the following ones, which we adopt for all our future networks :

window	**telescope**	**stride**
315	9	21

Table 1: Optimal parameters: *window* (input sequences' length), *telescope* (how much to look in the future) and *stride* (determines the sequences' overlapping)

3.1 Long-Short Term Memory and Convolutional Network

A network made up only with `Conv1D` layers and one with only with `LSTM` layers did not give us reasonable and good predictions neither on *Codalab*, nor on our test set. Therefore we decided to adopt, inspired by the architecture developed during Lab classes, an **autoregressive model** that intelligently **combines** `Conv1D` and `LSTM` (detailed implementation in the Notebook `CONV_LSTM_NoCorr.ipynb`).
The resulting network is composed of 562,423 trainable parameters, and contains also `GlobalAveragePooling1D`, `MaxPool1D` and `Dropout` layers. It has been trained with the optimal parameters (tuned as explained above): we obtained quiet good predictions on our test set, as shown in the below picture.

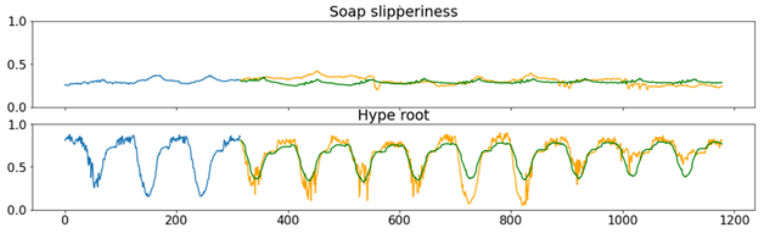


Figure 2: Predictions on our test set, for the features *Soap slipperiness* and *Hype root*

In this way we obtained a *Codalab* score of 5.20. Removing redundant features (*Crunchiness* and *Wonder Level*, which bring the same information of other features, see Notebook `Best_CONV_LSTM.ipynb`) we were able reduced the score to **5.06**. Unfortunately, deleting the constant traits in the dataset (as explained in the previous section) did not give us the desired results: we believe that this could be due to the unnatural way in which the series were subsequently linked. Finally, since the *Codalab* score is computed using the *Root Mean Squared Error*, we tried to change the loss function (using RMSE), but we did not achieve any better result.

3.2 Attention Networks

In neural networks, the *Attention* is a technique that mimics cognitive attention. The effect enhances some parts of the input data: the idea behind this is that the network should devote more focus to that small but important part of the data. Usually attention layers can solve one weakness of RNNs, who tend to forget relevant information in the previous steps of the sequence. The practical idea to keep that information is to use a weighted sum of the encoded states.We implemented this mechanism in two ways: adding it in a sequential model (see section 3.2.1), and creating a complicated architecture, using encoder and decoder (see section 3.2.2).

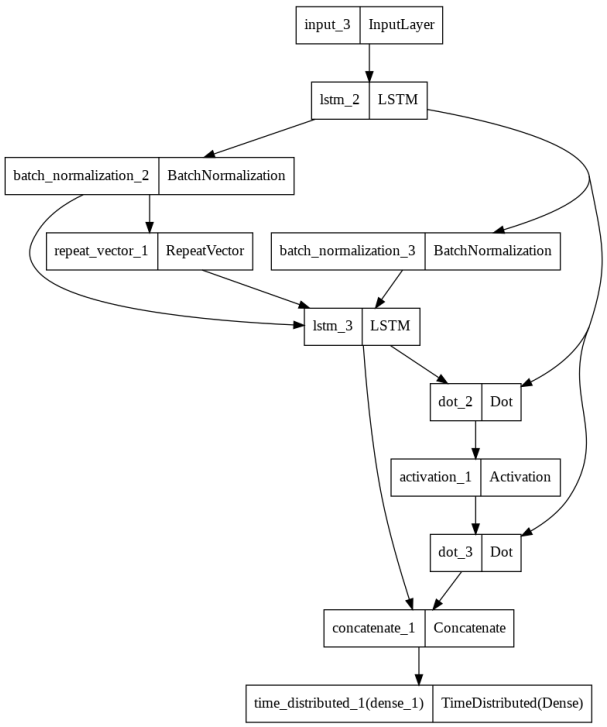
3.2.1 Attention mechanism in sequential network

We started by implementing the `Attention` layer from scratch, defining its class in the proper way (see the detailed implementation in the notebook `Sequential_Attention.ipynb`). After that, we inserted it in a neural network that we built, similar to the one of section 3.1. In particular we replaced the `Conv1D`, `GlobalAveragePooling1D` and

`Dropout` layers with the `Attention` one. This network is composed of 365,972 trainable parameters. Unfortunately, we did not obtain the desired improvement: indeed the *Codalab* score is 6.45, hence *Attention is NOT all we need*!

3.2.2 Attention mechanism in a complex architecture

On the right you can see the strage network implementing the attention mechanism, using encoder - decoder `LSTM`.
In addition to the last hidden state and cell state, we need also to return the stacked hidden states for alignment score calculation. Then, we make copies of the last hidden state of encoder and use them as input to decoder, getting also its stacked hidden state of for alignment score calculation. To build the attention layer, we calculate the alignment score and apply the `softmax` activation function over it. After the computation of the context vector, as last step, we concatenate it and the stacked hidden states of the decoder.The result is given in input to the last dense layer.
In some parts of the network batch normalization is added to avoid gradient explosion caused by the activation function in the encoder.
The implementation details are in the notebook `Complex_Attention.ipynb`
Exploiting this network, whose training required lot of time, we achieved a *Codalab* score of 7.10: this teaches us that a more complex model is not necessarily a better model.



3.3 Other attempts

Although we made several attempts using both the `Attention` and `GRU` layer instead of `LSTM` we were unable to improve our score. Hence we decided to go back and try to build a new neural network. This new architecture for auto-regressive prediction is made up with 2 `LSTM` layers followed by a series of `Dropout` and `Dense` (for implementation details see the notebook `Attempt_dense.ipynb`) . However, unlike what we desired, we got a *Codalab* score equal to 5.13.

4 Final conclusions and future developments

If in the *Docker* indicated in the Competition's *Evaluation* section we had also had some statistical libraries available for the time series analysis (such as `statsmodels`), we would have adopted also a different approach to solve the task we were assigned. Indeed, in Machine Learning (and in Model Identification in general) it is common to combine a black-box approach (like a Neural Network) with a white-box data preprocessing, to extract from them some easily observable features in order to simplify the job of the black-box model identification procedure.

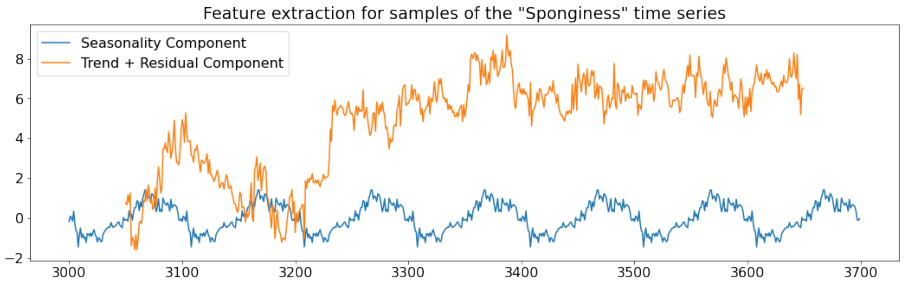


Figure 3: *Seasonality* and *Trend + Residual* decomposition of samples of input data

In our case (see `Dataset_Inspection.ipynb` for implementation details), we could have extracted a recurrent seasonality in the time series (indeed a cyclic behaviour can be noticed), trying to learn only the trend with a neural network. Nevertheless, we are really satisfied to have ventured into a real-life task and performed good results.