Homework 2 - Data Series Forecasting

Laura Amabili - 10598083

Marcello Bavaro - 10597421

Luca Maniscalchi - 10583576

Simone Sarti - 10580595

**DATA ANALYSIS**

To better understand the data series we were working with, we performed some analysis on them. The results can be found in the relative notebook, however the most interesting points are:

* There are two couples of highly correlated series: (Wonder lever - Loudness on impact) and (Crunchiness - Hype root)
* All the series have high autocorrelation, meaning that they can be predicted from the past
* All the series show small p-value for the Augmented Dickey–Fuller test, so we can reject the hypothesis that the series is a unit root
* Autocorrelation and Partial-Autocorrelation plots were “similar” to those of ARs, as the partial-autocorrelation rapidly shrank towards the interval of confidence while autocorrelation remained high (however periodicity was visible in the plots)

**PREPROCESSING**

Neural networks do not work well when data are not on the same scale, so before starting the training of our models we normalized our data. We rescaled them from the original range so that all values were within the range of 0 and 1, we trained all models on these normalized data and finally predictions were converted back into their original scale.

Time series datasets may contain trends and seasonality, which need to be removed prior to modeling in order to obtain a stationary process (if it is not already) to analyze. So we checked whether there was any evidence of a trend or seasonal effects and, if there were, remove them. In our dataset, some seasonality was noticeable, however, for a more precise evaluation of the stationarity of the data, we analyzed the rolling mean and standard deviation. Even if the standard deviation was more or less stable, on the other side the mean was varying.

We modeled trend and seasonality components, removed them from observations and then trained models on the residuals. After some tests we decided not to go ahead with these procedures because it wasn't [worth the effort](https://context.reverso.net/traduzione/inglese-italiano/wasn%27t+worth+the+effort) for the following motivations:

* we noticed only a little improvement in models performance
* we have faced some difficulties in restoring trend and seasonality in predictions in order to convert them back into their original shape

We also attempted to augment our data by adding to the dataset, for each series, the difference between successive values and the values themselves but shifted. There were slight improvements in performance but also in this case they were marginal and not worth the effort of managing more complex networks.

**SINGLE BRANCH NETWORKS & KERAS TUNER**

In our first attempts to find a good network we used the simpler and more “typical” single branch architectures from input to output. During this first phase, we tested networks with LSTMs, bidirectional LSTMs, GRUs, a combination of dense layers with the previous ones, dropouts and of course different numbers of layers and units. We also used in an extensive way Keras Tuner, which allowed us to test different combinations of hyperparameters in a single run. The best score we obtained is 3,98.

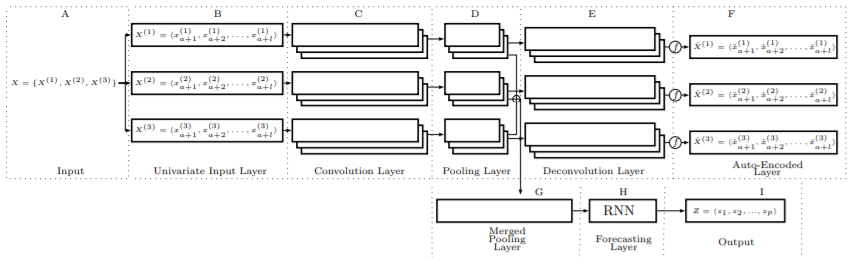
**CRNN & AECRNN**

The *Convolutional Recursive Neural Network* (CRNN) and later the *AutoEncoder Convolutional Recursive Neural Network* (AECRNN) were our first attempt at a complex network for this challenge. We recreated these networks from a paper titled *“Correlated Time Series Forecasting using Deep Neural Networks: A Summary of Results*'' that can be found [here](https://arxiv.org/abs/1808.09794).

We implemented it using 3 convolutional (and later deconvolutional) layers, and also 3 LSTM layers for the RNN segment. Unlike in the article, our network had to predict 7 series at the same time and not just one. To correctly implement the network we also had to understand how to feed the features taken singularly from the training and testing sets to the multiple input layers of the network. We did that using dictionaries to pass various components of the sets at the same time as “x” to the fit() (and predict()) methods. In this case the best window to be used couldn’t be too small (100) nor too large (4000), a value of 720 worked pretty well.

The next step was the implementation of the Autoencoder part to create the AECRNN, the idea being that if we also consider how well we can reconstruct the series, we should get a better model. One decoder had to be added for each series in Input, meaning that we not only had to manage a network that now had 8 outputs, but they also had different shapes (7xAutoencoders: (window,1), 1xRNN: (telescope,7)). We used dictionaries again, passing y\_train to the RNN and the X\_train components to the autoencoders as “y” of fit(). Multiple outputs also mean multiple losses (with their weights), dictionaries can be used to define them, in particular we assigned a weight of 1.0 to the RNN branch and of 0.14 (=1/7) to the autoencoders. We didn’t care about the output of the autoencoders, so we just discarded that part after the model was fitted.

Considering that some of our series were highly correlated (and the title of the paper), we expected great results from these networks, but in reality it scored around 4.6. Still, the autoencoders did a very good job at reconstructing the series (in particular when larger windows were used).



**MULTI-OUTPUT NETWORKS**

The experience with the (AE)CRNN taught us how to handle a network with multiple inputs and outputs, so why not try to use one of the best performing base networks we had and use it many times in parallel? The starting point for the Multi-Output series of networks had 7 parallel branches, each taking in a single series and trying to predict just that one, independently from the others. Once again we had to employ dictionaries to manage multiple inputs and outputs. Some post-processing had to be done to re-transform an output with shape (7,x,864,1) into (x,864,7), to do that we used the np.swapaxes() method. We tried many versions of this network with different amounts of LSTMs and dense layers (with and without dropouts). In general the performance was better than the one we obtained with basic networks and (AE)CRNNs.

We later experimented with differentiating the structure of some of the branches. After many tests we concluded that some series were predicted well using a sequence of two LSTMs while others obtained better results adding dense layers after the LSTMs. At this point branches still received as input only the correspondent data series to predict, the exception being branch 5 (Soap slipperiness, which always obtained bad results taking in input only its data serie), because after some attempts we discovered that good way to improve its performance was to pass as input to its branch all the data series together. This solution surprised us because there was no big correlation between soap slipperiness and the other data series. Another peculiar result of our tests was the absence of improvements when we tried to exploit the correlation between the two couples of correlated data series; in fact giving as input to the single branches of correlated features the combination of the two data series the model gave a worse score. The best network “hydra” obtained 3,7.

**TRANSFORMERS**

The transformer architecture we chose to use came from an article titled *“Attention Is All You Need”*, but in our case we used a version specifically designed for time series instead of natural language.

Due to the limited computational power at our disposal and the heaviness of the network, we were forced to reduce the number of transformer blocks (from 4 to 2) in addition to the window (to 1000) and batch size (typically 128, here 64). Small windows just didn’t work. Unlike other models for which the optimal stride seemed to be around 100/200, the transformer got worse with bigger strides, increasing it to more than 50 just flattened the prediction to become the mean. Like many of our other networks, the transformer also had big problems in predicting the shape of the sponginess series. It also had problems with flatter series, in this case the forecasted series appeared to be shifted up/down with respect to the real one, increasing significantly the total error of our submissions. The results we obtained were never better than those obtained with other models we worked on, maybe because of the need to reduce the size of the network.