Homework 2 – Multivariate time series prediction

The aim of this project is to design and implement forecasting models to learn how to exploit past observations in the input sequence to correctly predict future samples of a multivariate time series.

1. Exploration Data Analysis and Data Pre-processing

We were provided with a dataset multivariate time series consisting in 7 features with length of 68528 samples, all with uniform sampling rate. After importing libraries and fixing the seed for assuring reproducibility, we imported the dataset and, as required, converted the values of the sensor into float32.

Then, after inspecting the dataset to assess that no null values nor missing values were present, we split it into train and test sets. Clearly, the best estimator for the future is the last part of the dataset: thus, we chose it as test set (the last 5945 samples), paying attention to avoid shuffling in order not to destroy information; the first 62583 samples were chosen to belong to the training set.

Data that goes into neural networks should usually be normalized in some way to make it more amenable to be processed by the network. In our specific case, we computed the maximum and minimum values for each feature among the samples in the training set and used them for train and test data normalization.

Afterwards, we created an ad hoc function 'build_sequences': this function takes as input the original long time series and divides each of them into small subsequences of a predefined length (defined by the 'window' variable), based on the 'stride' parameter which defines the number of steps between two following subsequences. Another parameter is fed to the function, the 'telescope', indicating how much far away the model will look in the future in terms of points: this parameter is used for defining the labels of each subsequence which consist in its final samples, because the labels are the future itself.

1.1 Correlation analysis

Due to the continuous and chronologically ordered nature of time series data, we investigated if there could be some degrees of correlation between the series observations, thus correlation analysis is a fundamental step when dealing with time series.

We started by investigating the cross-correlation between the multiple time series provided, as shown in Figure 1, observing that for some of them, like Loudness on impact-Wonder level and Cruchness-Hype root, correlation was significative. This has also guided us in interpreting performances of the model we tested, from the observation that the RMSE measured on highly correlated time series was almost the same for most of the models.

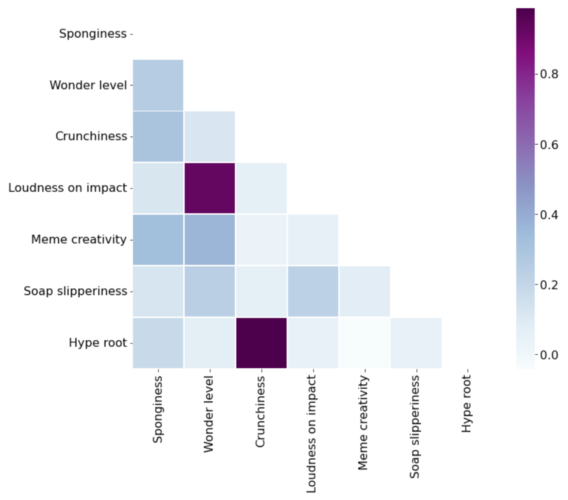

Figure 1

The successive and most relevant step has been focusing on the measure of the level of correlation between each time series and its own lagged version through the autocorrelation function, to eventually detect patterns and seasonality. As a reference, Figure 2 shows the autocorrelation for Sponginess time series. This has guided us in choosing meaningful values for some parameters like window length.

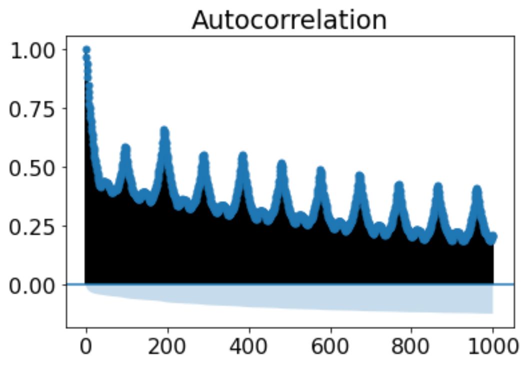

Figure 2

2. Model Implementation

In order to achieve better performances, we implemented many different types of models and architectures. Here are enumerated all the trials:

- Multi-head LSTM model
- RNN
- GRU
- Convolutional LSTM, with 1D convolutional layers for resuming past information, progressively and with an increasing degree of abstraction in time, combined with LSTMs to support sequence prediction.

Both for LSTM and GRU we investigated *bidirectional architectures* to take advantage of the sequences' analysis in both directions to then build a more informative internal representation. We also explored the addition of *skip connections* to some of these models in order to bypass convolutional layers and give raw data the ability to directly influence the formulation of predictions. Finally, we implemented *attention* and added it to the models which achieved better RMSE score.

Among the tested models, we decided to report the one which achieved the best score on the test set, which is the Convolutional LSTM with an attention layer. The architecture is made of a convolutional layer with 'ReLu' nonlinear activation function, kernel size 1×1, followed by a Dropout layer (0.3 of probability). Then, the Bidirectional LSTM layer with 256 units is added followed by another Dropout layer (0.3 of probability). At this point the attention is implemented: attention is a mechanism combined in the Neural Networks with recurrencies, allowing them to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and better quality. We implemented attention by means of an ad hoc function ('attention_3d_block') which takes as input the output of the previous layer, applies a dense layer which computes the SoftMax for each value of the sequences in input and finally multiplies each value for the corresponding score of the SoftMax, weighting them. Finally, a Flatten layer followed by a Dense layer are added, and input and output are connected through the Model class.

Next step is model compiling: being a multi-series forecasting prediction problem, we chose the Mean Square Error as loss function and Adam as optimizer algorithm with learning rate 0.001, which is able to adjust the learning rate based on the history of the gradient to both speed up the learning and increase the performance at the same time.

At the end of the training performed by splitting the training data into training set and validation set (0.1 proportion), we re-trained the model on the whole dataset, without holding out data for test or validation.

2.1 Hyperparameter Tuning

First of all, we investigated the two extreme cases:

- Direct forecasting, which consists in trying to predict the whole future in one single shot, so setting the telescope to 864. Performances in this case are very poor (RMSE $= 19.36$) and, indeed, using an internal built test set, we noticed that the error behaviour was almost uniform.
- Autoregression, selecting as value for the telescope 1. The performances of the model are very poor also in this case (RMSE $= 11.1007$), but at least much trustable with respect to the other extreme case: indeed, plotting the error in time, it is shown to grow.

Since both strategies appeared to be sub-optimal, we tried to optimize the telescope value in order to achieve the best performances.

Moreover, we tried many different combinations of hyperparameters, such as: number of filters in convolutional layers, kernel size, learning rate, dropout percentage, number of units in BiLSTM layer, window size, stride (paying attention to choosing values which guaranteed to have the ratio between window and stride have null residual to avoid having to pad or discard the last window), batch size.

In the end, we found that the best performing algorithms were the ones characterized by:

- Learning rate $= 0.001$
- Window $= 300$
- Stride $= 10$
- Telescope $= 24$
- Batch size $= 128$

3. Results

The final model chosen as the best one is, as described above, the Convolutional LSTM with attention, which allowed us to obtain, on the hidden test set, RMSE $= 3.49$.

A summary of the results obtained on the hidden test set with the above mentioned models is illustrated in Table 1.

Table 1 – Models summary

Model	**RMSE**
RNN	13.33
GRU	4.91
Multi-head LSTM	4.06
ConvLSTM	4.56
ConvLSTM with attention	**3.49**