AN2DL - Second Challenge

Andrea Deretti
Francesco Rettore
Michele Russo

POLITECNICO
MILANO 1863

Politecnico di Milano
Italy
21 January 2022

1 Introduction

1.1 Goal of the Competition and Available Data

This competition consists in a time series forecasting task. In particular, we are required to forecast seven different signals over a time horizon of 864 time instants. Moreover, the training set is composed of 68528 samples for each of the seven signals. The data at our disposal is visualized in Figure 1.

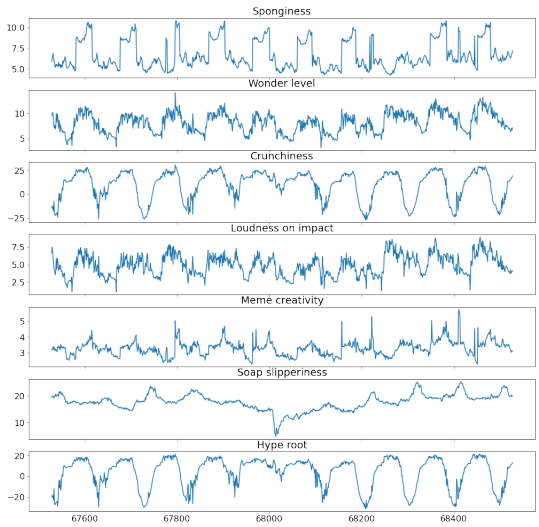

Figure 1: Traceplot of the training data.

1.2 Data Pre-Processing

In order to be able to validate the results we achieved, we decided to remove the last 864 samples from the training set and use them as a test set. Obviously, once we had selected the best model using this test set we trained it again on the whole data to use all the available information to obtain a more accurate forecast.

Moreover, we had to extract from the time series a supervised dataset to train a Neural Network. In particular, we adopted a sliding window approach where:

- The training feature is given by a sequence of ***window*** consecutive samples.
- The target is given by a sequence of ***telescope*** consecutive samples which follow the former ones.
- Different observations are obtained by iteratively sliding the two sequences of ***stride*** samples.

The choice for the three parameters *window*, *telescope* and *stride* will be discussed in details in Section 2.2.

Since, as shown in Figure 1, many of the signals we are dealing with are evidently periodic, we first tried to perform a *Seasonality and Trend decomposition* in order to separate the seasonal component of the series. We did this in an attempt to make the task easier by training the Neural Network to predict only the trend and the residuals of the series and adding the seasonal component to the final forecasts. However, the performances obtained via this procedure turned out to be

worse than we expected, so we later decided to drop this idea. This may indicate that the seasonal component of the signals is not to be assumed as constant in time.

Another idea we tested consists in considering the series of the differences between two consecutive time steps rather than the original series. Even though this is a well-established practice when dealing with non-stationary time series, again the obtained results were not satisfying so we decided to discard this approach as well.

All in all, we decided to normalize the data in the range $[0, 1]$ as the original signals take values in quite different ranges.

2 Model for the Neural Network

2.1 Architecture of the Neural Network

Given the fact that we are dealing with sequential data, we decided to resort to a Recurrent Neural Network. In particular, inspired by the *LSTNet* architecture proposed in [1], we first tried to combine convolutional, recurrent and recurrent-skip layers to extract short-term local dependency patterns among the variables and long-term patterns for the time series trends. However, the tests we performed showed that simple architectures were able to achieve better performances. Therefore, the final architecture for the Network is based on three *Gated Recurrent Units*, as shown in Figure 2.

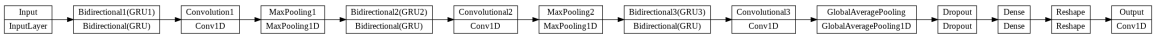

Figure 2: Architecture of the Neural Network

2.2 Choice of the Parameters for the Sliding Window Approach

As we mentioned before, a good choice for the three parameters *window*, *telescope* and *stride* is crucial to obtain a good forecast. Our first approach to fix them consisted in setting a grid of possible values for each parameter and performing an extensive search by trying every possible combination and comparing the resulting performances on the test set. We soon realized, however, that the optimal values for the parameters were highly dependent on the specific architecture we were testing. Therefore, it would have been necessary to repeat this process for any model we considered in order to assess its optimal performances. This was, of course, not feasible as this brute force search is computationally expensive. Consequently, we decided to resort to black box optimization in order to tackle this problem. In particular, we performed a random search of the parameter space according to Algorithm 1, implemented in *RandomSearch.ipynb*.

Such approach is less expensive than the extensive search as it randomly tests different combinations of the parameter values and selects the best one over a fixed number of iterations. In particular, for each architecture we tested we ran the algorithm for 20 iterations and for each iteration we trained the model for 20 epochs. In the end, the optimal values we obtained for the final architecture are the following:

- *window*: 100;
- *telescope*: 48;
- *stride*: 10.

[1] Guokun Lai, Wei-Cheng Chang, Yiming Yang, Hanxiao Liu: Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks

Algorithm 1 Random Search of the Parameter Space

Input: n, $windowList$, $telescopeList$, $strideList$
$bestPerformance \leftarrow \emptyset$;
$bestParameters \leftarrow ()$;
for $t \leftarrow 1$ **to** n **do**
  $window \leftarrow windowList.random()$;
  $telescope \leftarrow telescopeList.random()$;
  $stride \leftarrow strideList.random()$;
  create the training data using the parameters $window$, $telescope$ and $stride$;
  train the model on the training data;
  $performance \leftarrow$ assess the model on the test data;
  if $performance > bestPerformance$ **then**
    $bestPerformance \leftarrow performance$;
    $bestParameters \leftarrow (window, telescope, stride)$;
  end
end
return $bestParameters$

3 Training

During the training phase, we used the Mean Squared Error as the loss function and the Adam algorithm as the optimizer.

In order to avoid overfitting, we decided to use 10% of the training samples, built as described in Section 1.2, as a validation set. For what regards the training of the Neural Network, we observed that the loss function on the validation set reaches an optimal value pretty fast and then remains steady. For this reason we decided not to resort to the *Early Stopping* technique as it could have limited the performances. Instead, we trained the model for 35 epochs and we monitored the validation loss "by hand" to make sure we did not overfit the training data and tried to achieve the lowest loss on the training set. In addition to this, we adopted a decreasing learning rate rather than a fixed one. In particular, we set the learning rate to decrease after 10 epochs without an improvement in the validation loss. Specifically, such technique is called *ReduceLROnPlateau*.

Initially, we decided to train a multivariate model that takes as input all the signals at once. The performances we achieved were satisfying and so we used this model as a baseline for the following steps. We then trained seven univariate models, each regarding just one of the seven signals and with the same architecture of the previous multivariate model. We did this as we believed that assuming a priori that all the series are correlated would be a mistake. Therefore, we expected that the forecast for some of the signals would be more accurate by building a model that takes as input only the past of that specific signal. Indeed, we observed that the predictions for the signals *Sponginess*, *Loudness on Impact* and *Meme Creativity* were more accurate than in the case of the multivariate model. For this reason, we decided to build an embedded model by replacing the multivariate predictions for such signals with the corresponding univariate predictions.

4 Assessment of the Results

After the training of the model, we compared the forecast of the seven time series against their actual values for the last 864 samples of the original dataset (which we kept as a test set as specified in Section 1.2). In Figure 3 we plot the forecasts (in orange) against the actual data (in green). As we can see, the results obtained are satisfying for many of the signals.

All in all, we trained the model again on the whole dataset and we tested its performance on the

hidden test set (which consists of 864 future samples). The *Mean Squared Error* is equal to 3.88.

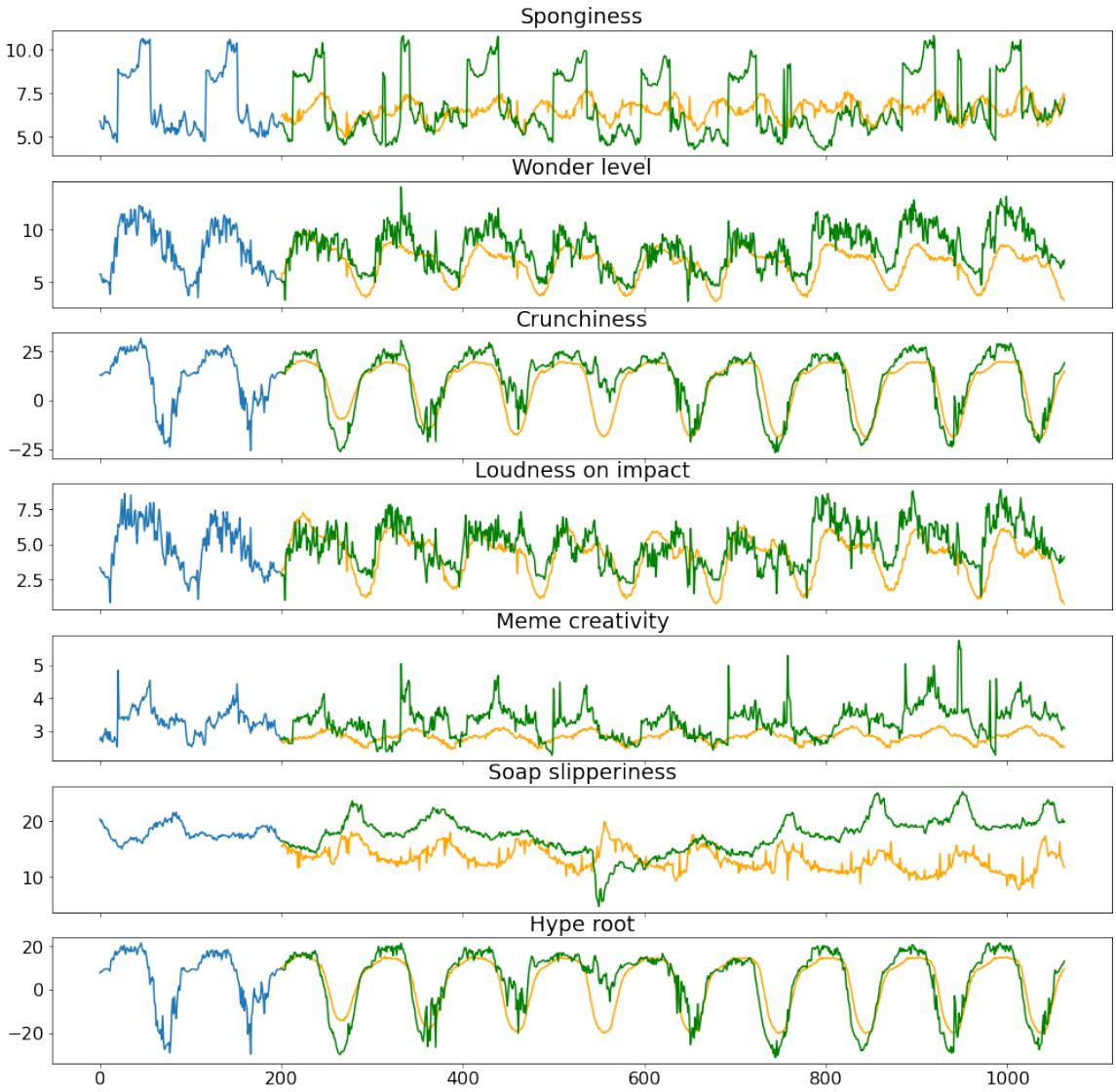

Figure 3: Forecasts vs Actual Values