ACADEMIC YEAR 2023/2024

POLITECNICO
MILANO 1863

Artificial Neural Networks and Deep Learning

Challenge 2 - Time series forecasting

Stefano Bruni - 10660827
Matteo Pisani - 10921568
Flaminia Telese - 10931641

Prof. Matteo Matteucci

Introduction

In this homework, we had to deal with a time series forecasting problem. The goal was to develop a model to predict future observations from past ones for different types of univariate time series. A training dataset was given, consisting of 48000 time series of different length and characteristics. Assessment was made over a distinct, hidden dataset of sixty 200-samples-long time series, using the MSE metric. The test forecast horizon was of 9 samples in the first phase, of 18 in the second.

Length analysis

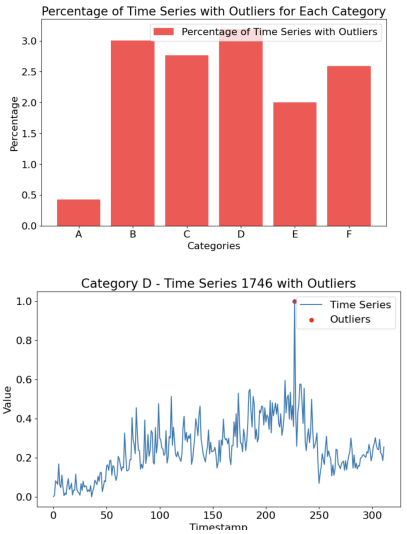
As anticipated, we were given a dataset of 48000 time series, each with different length and belonging to six possible categories. After further inspection, we noticed that about half of the time series consisted of no more than 200 samples and that the dataset was not balanced, with categories ‘A’ and ‘F’ showing considerably less time series compared to the others.

```
Category A:
  Number of Time Series: 5728
  Average Valid Timestamps: 278.18034217877096
  Maximum Valid Timestamps: 1943
  Minimum Valid Timestamps: 46

Category B:
  Number of Time Series: 10987
  Average Valid Timestamps: 165.9428415400018
  Maximum Valid Timestamps: 1484
  Minimum Valid Timestamps: 42

Category C:
  Number of Time Series: 10017
  Average Valid Timestamps: 208.14625137266646
  Maximum Valid Timestamps: 2708
  Minimum Valid Timestamps: 42

Category D:
  Number of Time Series: 10016
  Average Valid Timestamps: 216.9909145367412
  Maximum Valid Timestamps: 2641
  Minimum Valid Timestamps: 42

Category E:
  Number of Time Series: 10975
  Average Valid Timestamps: 163.04601366742597
  Maximum Valid Timestamps: 2776
  Minimum Valid Timestamps: 42

Category F:
  Number of Time Series: 277
  Average Valid Timestamps: 194.8303249097473
  Maximum Valid Timestamps: 1068
  Minimum Valid Timestamps: 24
```

Results of the length analysis

Outlier analysis

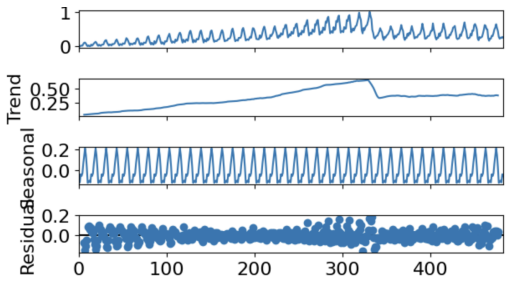
To check if any time series had some noticeable outliers, we computed the z-score over the samples. To correctly identify outliers, we looked for a z-score [5] of 4 or 5; any value lower than that would find wrong outliers most of the time. We tried two strategies to reduce the effect of outliers on training. The first was robust scaling [1]: it works by considering the quantile range calculated over the same timestamp in different time series. This often modified the time series in undesired ways, completely changing their trends and without really removing the outliers, so it was not used and instead we kept the original 0-1 normalization of the dataset. The other strategy was to directly drop the timestamps presenting the outliers: this was slightly more effective, but required to re-normalize the specific time series, and overall didn’t prove to be very helpful to increase the model performance.



Percentage of time series with outliers across categories and an example of time series with an outlier.

Autocorrelation analysis of time series

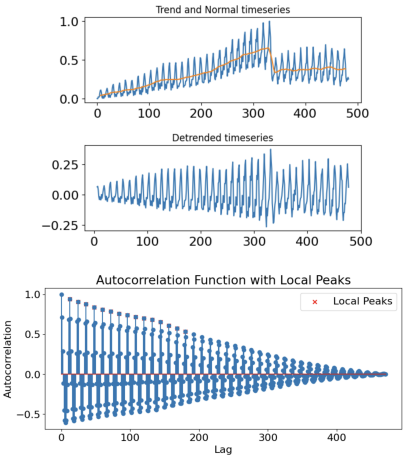
Each time series was split into trend, seasonality and residual using additive decomposition [4]; a good network should model all of those three aspects; moreover knowing the decomposition, together with other tools such as the autocorrelation spectrum, can help design a better architecture. Indeed, the autocorrelation function can be used to tune the window and stride sizes used to create the training set. If the autocorrelation presents spikes after some lag, it means the signal is very similar to itself after that interval, indicating a potential periodicity. We looked for repeating spikes with value above 0.5 to detect potential seasonality in our dataset.



Example of detrending

Moreover, we calculated, for each time series, the number of timestamps required for the autocorrelation to go to 0 for the first time: this shows the presence of correlation, which means that past observations within that time window are informative about the future. However, relying too much on autocorrelation can lead to overfitting, since the model won't learn to predict based on previous, less correlated samples. It resulted that the average first zero lag over the entire dataset was 38, indicating that a time window greater than 40 can help our model grasp the relation between past and future samples. We then proceeded to further analyze the seasonality of the signals. As the trend component could prevent us from correctly determining the periodicity of our signals, we proceeded to study the autocorrelation of the signals deprived of their trending. Overall, almost one third of our time series showed some periodicity, and the most recurrent lag was 12 timestamps (possibly indicating hours or months). Hence, a time window of 12 or bigger can be suitable to help our model understand this kind of behavior.

After this analysis, we chose to use a window size of 200, according to the test task, and a basic stride of 20. We asked ourselves if the presence of padded data needed to fill each window up to 200 samples could have altered the training process: we experimented with masking layers, but they dramatically slowed the training time, making them unfeasible to use. In the end, to deal with windowing and seasonality, we introduced attention layers, so that the model would independently learn which are the relevant timestamps to look during prediction, thus reducing the need do handle the padded values and it would also be able to catch the seasonality of our time series even if the stride and windows were not optimal.



Example of detrended time series and autocorrelation function

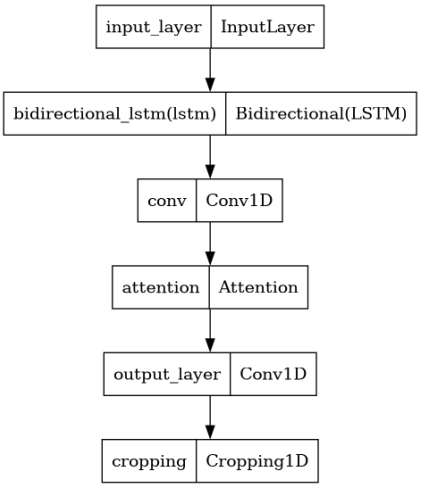
Training, test and validation split

For what concerns splitting, we decided to dedicate some series exclusively and completely for training, some others for testing, and some others for validation. Moreover, test and validation sets were built to contain the same number of time series for each category. All of this was made in order to better estimate the performance over the test set, which consisted of 10 unseen time series of each category. As previously mentioned, the ‘F’ time series set is considerably smaller than the rest, thus we were limited in how big the validation and tests set could be: we wanted to keep the most possible ‘F’ data for training, to avoid creating a model not able to properly forecast for ‘F’ time series. So we decided that each category group in the validation and test sets was to be as big as 10% of the total amount of ‘F’ time series, so as to have 80% of ‘F’ data for training and respecting the constraints we put above.

Best model: ConvLSTMAttentionNet

The best model we built is a very standard one: it consists of a Bidirectional LSTM

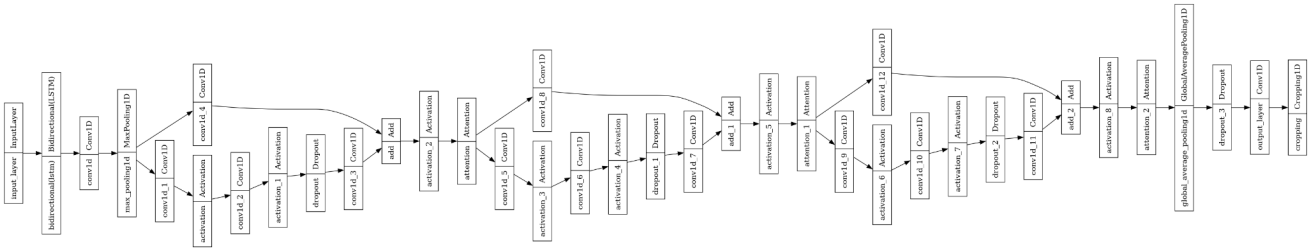
layer right after the input, a Convolutional1D layer, an Attention layer and finally a last convolution. Despite its simple structure, this model performed significantly better than any other architecture we tried in both the first and second phases: both adding a Dropout layer and a Ridge regularizer worsened the performance; similarly, using a Multihead Attention layer instead of a simple Attention, as well as changing the score mode from dot product to concatenation, gave worse results. Generally speaking, we observed that adding more complexity would only drop our local validation and test MSE; many other experiments were made that confirmed this intuition. The model is also autoregressive, taking as input a window of 200 time samples and predicting one step in the future; to predict more samples, we add the previous predictions to the input and repeat the process. This choice was made to follow the prerequisites of the competition, which asked for a model "not limited to predicting in a predefined time context"; moreover, as the forecasting horizon grew, we noticed that direct models had a lower performance on our local test set and on the hidden one. In the end, this architecture obtained a hidden test MSE of 0.00527 when predicting 9 samples and of 0.00864 when predicting 18 samples. We thought that the model's simplicity could have been its winning factor, allowing it to stay more general and not overfit. This could be the reason behind an average performance in the first phase and an even better one (rank speaking) in the second phase.



Model architecture schema

Experiment: Attention ResNet

Among the many experiments made, the most promising result was an Attention ResNet based on Wang's article [2] regarding the ResNet structure and Wang's article regarding the use of Attention modules in ResNets [3]. We adapted those architectures to our necessities, using Conv1D layers instead of the 2D ones and changing the output block to meet the requisites of the challenge. We also decided to put a LSTM layer on top of the model since we found out that it improved the performance regardless of the architecture. The results on the hidden test set of this model were: MSE of 0.00582 in the first phase and one of 0.0115 in the second phase.



Attention ResNet architecture schema

Individual contributions

On the premise that all teammates contributed to every step mentioned in this report, the members main focus was on:

- Stefano Bruni: general preprocessing, autocorrelation and decomposition analysis;
- Matteo Pisani: general preprocessing, best model design, report design;
- Flaminia Telese: best model design, Attention ResNet design.

References

[1] Compare the effect of different scalers on data with outliers — scikit-learn 1.3.2 documentation

[2] [1611.06455] Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline, GitHub repository: https://github.com/cauchyturing/UCR_Time_Series_Classification_Deep_Learning_Baseline/tree/master

[3] [1704.06904] Residual Attention Network for Image Classification

[4] Chapter 6 Time series decomposition | Forecasting: Principles and Practice (2nd ed)

[5] Standard score