Artificial Neural Networks and Deep Learning 2023-Homework 2

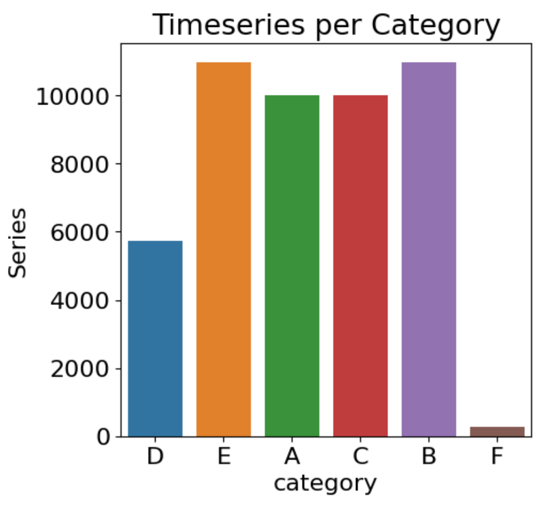
Team name: NCSP

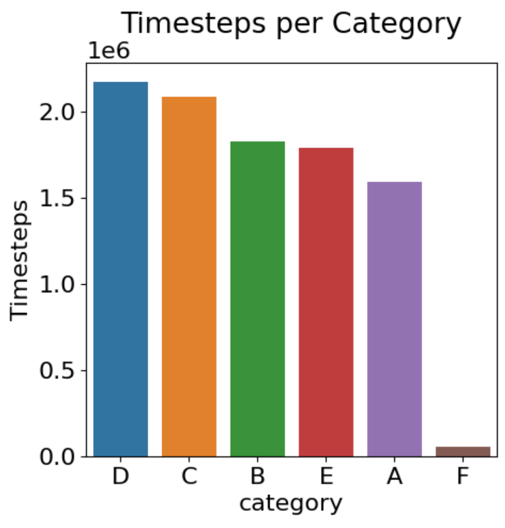
Data Inspection and Visualization

We started by converting the input data into a pandas data frame to perform some analysis and gather insight on the provided dataset. As first thing we printed some info about the data frame using df.describe(), in particular we notice that the data has already been normalized.

	Series ID	**Value**
Count	9.51 e06	9.51 e06
Mean	2.25 e04	4.25 e-01
Std	1.36 e04	2.69 e-01
Min	0	0
25%	1.08 e04	1.09 e-01
50%	2.39 e04	4.03 e-01
75%	7.35 e04	6.40 e-01
Max	4.80 e04	1

Then, we printed the number of timeseries and timestamps per class and we discovered that they are quite unbalanced, in particular for class F that has a lot of data less than the others.





We also looked at the valid periods of the time series and we have understood that a huge number of time series has a length lower than 50. Eventually we looked at different time series from the same class and noticed no apparent correlation between them, but we wanted to investigate this point further (see Class Analysis).

Dataset Split

To split the dataset in the three usual sets (training, validation and test) we have decided to use a particular approach. Instead of splitting each time series in parts, with the test coming after the training, we have decided to use some time series for the training, others for the validation and the remaining for the test for mainly two reasons:

1) There are a lot of short time series, so we thought it wouldn't be optimal to reduce even more the number of timestamps used for training on those series.

2) Using the new split, we still achieve the general rule of having training, validation and test set one independent from the others.

Class-based Models

One thing we noticed was that together with the timeseries we were also given their categories both at training and at evaluation time. Wondering whether this information could be useful for timeseries forecasting, we tried building a model that also used the category information to make its predictions.
This was accomplished by concatenating the one-hot encoding of the category information to the timeseries itself.

The performance we reached was close to the previous models but didn't really achieve anything better, probably meaning that the information was not that useful and/or that the model learnt to discard it during training not being able to understand how to fully exploit it.

Autocorrelation Analysis

Another analysis we performed was on the autocorrelation values of the timeseries. We started by calculating (using the pandas library, applied to the data frame version of the dataset) the autocorrelation of each considered window for different values of lag. One thing that we observed was that, in general, the highest autocorrelation (even among different categories) was usually reached with a lag that was a multiple of 12. Because of this we tried by training models using an input size that was itself a multiple of 12; still, this didn't lead to any considerably better result than that of a model trained with an input window of about 200.

Another kind of autocorrelation analysis we performed was directly on the original timeseries contained in dataset.npy. We thought that since series with low values of autocorrelation aren't usually predictable, that they could also not be useful to learn to predict a class of timeseries. By calculating the maximum autocorrelation considering a lag between 1 and 60, we saw that only about 5300 in 48000 series had a maximum autocorrelation of less than 0.8, and just 9500 in 48000 less than 0.9.

We then tried training different models by keeping only samples with a higher maximum autocorrelation than a certain value (0.80, 0.85 or 0.90) and what we saw was that in general these models behaved well locally, but usually not so well on the online evaluation, only coming close to our previous best model but without an improvement, at least in the prediction of 9 future timesteps.

We think that this loss in performance may be caused by the fact that by only keeping samples with the highest autocorrelation, our models had an "optimistic view" that may not have allowed them to fully generalize.

Still, in the final phase of the challenge which involved the prediction of 18 timestamps by combining it with the average-based autoregressive we managed to perform the best result.

Other Analysis

-During the inspection of the data, we have immediately noticed that we have a lot of timeseries with a valid period lower than 50. Thus, using a larger window size (like 200) we will have a lot of 0 values, we thought that this may complicate the learning process. (Similarly, we have removed the windows, that due the addition of the padding, have a lot of 0 values before the valid periods). Locally the model was improving the performance but submitting it we obtained worse results. Hence, we decided that it's better to maintain all the data without removing them.

-Even if the data are already normalized (values between 0 and 1) we noticed that the data aren't well distributed and the presence of outliers could still mislead the training process. Thus, we have tried to apply the Robust scaling to the data, but this hasn't improved the performance of our models.

Autoregressive

For all the models we used we tried to perform autoregressive prediction using different autoregressive telescopes, mainly of 1 or 3. One peculiar thing we noticed was that, when using autoregression, simpler models, that gave worse results in the prediction of the entire telescope, gave better results than more complex and better model. In fact, the result of simpler model became significantly better when using autoregression but, unfortunately, this only made them comparable to the result of the complex models on the 9 or 18 values but not better.

As an experiment, we also tried use the more complex models to predict all values while also using autoregression and then averaging the values obtained with the corresponding results. This meant that the i-th value to predict was actually predicted i times and then their average was computed. This gave slightly better result than using the best model trained to predict 18 values (0.01129692) but gave way better result when using the best model trained to predict 9 values during the development phase (0.00999891).

Attention

We have tried to implement a model with an attention layer from the keras library by simply adding it between the Bidirectional and Conv1D layers. We considered this an essential addition to the model since we thought it would enable our model to understand sequential dependencies much better. In reality this model didn't perform as well as we had hoped and one of the possible reasons for why this didn't work could be related to the dataset: maybe it doesn't exhibit strong temporal dependencies or maybe the number of samples is too low.

First Model

As first model we have tried using a model very similar to the one seen during the Laboratory session on Time Series Forecasting to test out our first ideas. The model was very simple since it was composed mainly of a Bidirectional LSTM layer of 64 units and a 1D Convolutional layer with 128 filters (Kernel size = 3). This model gave us a reasonable performance with a 0.0063 MSE on the online test set. This made us realize that trying to develop more complex models might not be the correct way to approach the problem, so we tried to keep our solutions as simple as possible.

Final Model

We achieved the best performance using a model composed of the following layers:

- Bidirectional LSTM with 64 units
- Conv1D with 32 filters, 2x2 kernel size, activation='relu'
- MaxPooling1D with pool size = 2
- Conv1D with 64 filters, 2x2 kernel size, activation='relu'
- MaxPooling1D with pool size = 2
- Flatten layer
- Dense with 32 units
- Dense with 9 units as output

In the first phase we predicted 9 samples, same as the output layer while in the second phase we used an average as described in the autoregressive section. The final performance in the second phase was **MSE = 0.00983344 MAE = 0.06838870**

Conclusions

During this challenge we experimented with many different techniques to try and solve this Time Series Forecasting problem and after many iterations we settled on our final model described above. Our fist intuition was to keep the model as simple as possible since we quickly realized that increasing its complexity wasn't giving us any benefit. We encountered some difficulties in data analysis since there were many time series belonging to different categories. In fact, we didn't manage to use the information about the categories in any meaningful way, one area where we could improve our approach to the problem. Attention also didn't give us satisfying results, but it might have been because we used the Keras default layer and didn't try any custom implementation.

We then tried with standard autoregression, using autoregression telescopes of 1 or 3, it didn't give significantly better results, especially with complex model that tended to overfit. So, we came up with the idea to mix autoregression with the full models, predicting the entire telescope, by averaging the values that were predicted multiple times. This gave significantly better results on the 18 values prediction when using model predicting 9 values and autoregressive telescope of 1.

Contributions

Niccolò Maria Rizzi: Convolutional-based models, Category-based models, Autocorrelation analysis and training

Marco Baratto: Data Inspection, Attention models training

Stefano Bonfanti: Data Inspection and Visualization, GAP Models, Attention, Robust Scaler implementation, Dataset split, Short timeseries analysis

Giacomo Savazzi: Autoregression, Autoregession with Average.

References

Pandas DataFrame - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

Attention is all you need - https://arxiv.org/abs/1706.03762

Keras attention layer - https://keras.io/api/layers/attention_layers/attention/

Medium: time series forecasting a complete guide - https://medium.com/analytics-vidhya/time-series-forecasting-a-complete-guide-d963142da33f