REPORT CHALLENGE2 – ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

Group “Pinguino”: Pietro Caforio, Federico G. Ciliberto, Luca Civitavecchia

- 1. INTRODUCTION

This report serves as a comprehensive documentation of our journey through the challenge, aiming to present the most crucial steps undertaken, the challenges encountered, and the wealth of knowledge gained along the way.

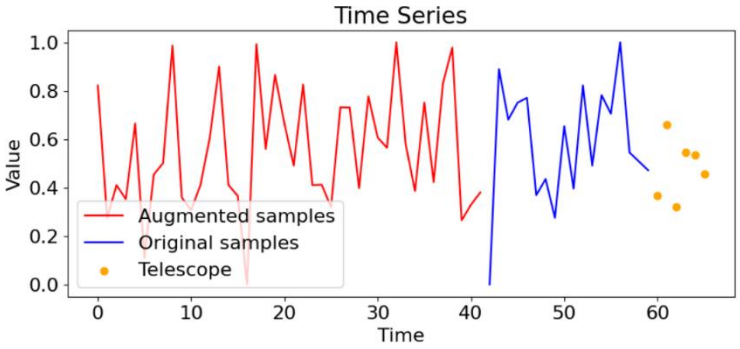
The different sections are not to be intended as a chronological explanation of the development of our model, but instead as a detailed analysis of the most important aspects of the developing process.

- 2. DATA PREPROCESSING

Our first approach was to thoroughly analyze the time series provided. We immediatly noticed a great imbalance in the time series classes, in particular, class F was heavily underrepresented. Multiple strategies on how to handle this were tried, but empirically we noticed that they didn’t produce better results. Thus, in our final models we didn't address the problem.

Another issue we analyzed, is how to handle the different classes. A first approach we tried was to train six different models, one per class, each one trained only on its respective class samples. Then, at inference time, based on the class of the time series to be analyzed, we would route it towards the model trained specifically for that class (more on this can be found in folder separate_models). However, we immediatly noticed a great increase in performances, by training a single model on the whole dataset, ignoring the classes information, and using that same single model, at inference time. This was partially justified when, analyzing the various time series, we noticed that in the same class there wasn’t much similarity among the samples.
Regarding the normalization, the time series provided were normalized between 0 and 1. We tried re-normalizing the time series with different techniques, like Robust Scaler. However, since it didn’t show any promising result, we abandoned the re-normalization idea, to avoid useless intricacies.

Another issue we addressed was how to treat the different time series lengths. In a first moment, we just selected a window size appropriate for every time series (meaning shorter than the shorter sequence). This was however a big constraint that was preventing our models to express their full potential. Then, we tried to drop the shorter sequences (for example the 1% shorter sequences). However the best results were achieved when we started using a bigger window size, copying multiple times the shorter sequences (also zero padding was tried). In the process of copy-pasting the sequences to make them at least as long as the window size, we also performed some random augmentation on them, with a gaussian noise [1]. More details on the window can be found in the next paragraph.



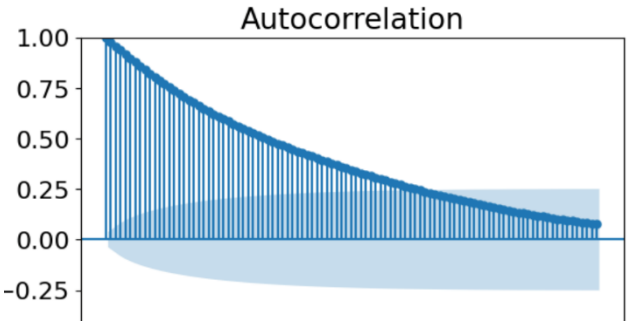
The image on the left shows the sequence built performing copy and random augmentation from a timeseries of length 24, using a window of size 60 and a telescope of size 6 (these numbers have been chosen only for representation purposes)

- 3. WINDOW SIZE

The time series were split in multiple sequences, which were then fed to the model. An hyperparameter of our model that immediatly showed a great impact on the performances, was the size of these smaller sequences, called “window-size”. As already briefly described in the previous paragraph, firstly we tried using a window size that was longer or equal than all the time series. However this was preventing our models to show their full potential, so we increased the window size, dropping the series which were shorter than it. This improved the performances, however we have carefully sought a way to avoid data loss. Our solution was to copy and paste the time series until the window size was reached. To avoid recopying

exactly, we also seized the opportunity to augment the data a bit, by introducing some random noise when copying. The same strategy was adopted instead of the standard zero-padding on longer timeseries, to make their length a multiple of the window size.

Up to this point, the window size choice was only guided by trial&error. Trying to refine our models, we then switched to a more mathematically correct technique. The idea was that the sequences fed to the model should contain enough data for it to forecast efficiently, and autocorrelation can be a useful tool in detecting the presence of seasonality and other patterns in the time series [2]. Basically, our approach was to compute the autocorrelation for each time series, and select the lag such that the autocorrelation value was lower than a certain threshold (for example, 0.2). On these values, we then computed the mean and used that as a window size. As already explained, if a certain time series was shorter than the window size, we padded it using augmentation. The threshold value was an hyperparameter that we set using trial&error.



The image on the left shows the autocorrelation function plot of a time series selected randomly from the dataset, showing the statistical significance of the autocorrelation value at different time lags (blue area).

- 4. ATTENTION IS (ALMOST) ALL YOU NEED

In later stages of the development, we tried including an attention mechanism in our models. The rationale behind was that by conducting some researches online, we discovered that attention mechanisms can be useful also in time series forecasting because they allow models to focus on specific parts of the input sequence when making predictions [3]. In timeseries data, certain time points or patterns may be more relevant than others for predicting some future samples. In principle, attention mechanisms enable the model to dynamically weigh the importance of the different parts of the embedding vector obtained from the LSTM, capturing long-term dependencies and improving the model's ability to make accurate predictions based on relevant temporal information.

This can also help our models adapt to the varying and, in some cases, very different patterns and relationships within the timeseries of the dataset, leading to more effective and accurate forecasting. Inspired by some applications we have carefully researched online [4], we added some layers in the ending part of our model that was exploiting attention. This showed immediatly some promising results, so we fixed the hyperparameters needed in this part with some trial&error, and deployed this technique on our best models so far.

- 5. ARCHITECTURES

The first very simple model we implemented was a trivial persistence model that simply returns the last nine (more in general, same as the required telescope) observations. This was done to obtain a baseline that could provide a comparison to our more sophisticated models [5].

From this point on, we tried approaching the problem with some actual networks. In every model tested, it was necessary to decide whether to let the model predict all the samples directly or to use an autoregressive-based approach, where the model predicts a smaller number of samples and then uses its own predictions to extend one side of the sequence, simultaneously removing the same number of samples from the other side to maintain a fixed size, repeating these steps until achieving the desired number of predicted samples. In all models, the autoregressive approach demonstrated better performance, therefore we decided to use it. The second architecture we tried was a very simple refactoring of one of the models seen during lectures. It was composed of a bidirectional LSTM layer, and some one-dimensional convolution layers. Some cropping was also used, to achieve the desired telescope length. This model was already performing considerably better than the baseline. We tweaked its hyperparameters a bit, trying to push performances, but we noticed that in most cases, adding complexity was introducing some overfitting and decreasing the score on the test set. We hypotesized that the model was too simple to extract meaningful embeddings, so we were curious

to try some more powerful architectures to do so. At the same time, we wanted an architecture capable of propagating the embeddings from the previous layers, in ordet to not lose the most important information extracted from these. These requirements guided us towards a ResNet-like architecture. Obviously, the classical ResNet architecture needed to be adjusted to our case, using one dimensional convolutions [6]. After the resnet-like architecture, some bidirectional LSTM layers and some more 1D convolutions were placed. This architecture further improved the performances on the test set.

On top of this model, we tried adding some final layers to implement an attention mechanism, as described in section 4. This was done trying different approaches, for example, one thing that has shown a significant impact on performance is whether to include 1D convolutional layers after the attention mechanism. Eventually, the best solution turned out to be the one composed of: ResNet-like part, bidirectional LSTM, attention mechanism.

Based on the tips provided during the lectures, we attempted to implement another model by eliminating the bidirectional LSTM layer, opting instead for a straightforward fully convolutional neural network with nothing else [7]. Initially, we believed that incorporating layers of one dimensional convolution along with maxpooling, normalization, and dropout would suffice. However, the validation set revealed less-than-promising results, prompting us to consider the possibility of repurposing the ResNet architecture that we had previously adapted for time series, but this time, with nothing else on top. Surprisingly, this adjustment yielded improved results on the validation set; however, these gains did not translate to the test set. More details on this approach can be found in folder fully_conv.

- 6. ENSEMBLE MODELS

To enhance our overall performance, we attempted to create an ensemble of our most effective models. We explored two approaches:

1. Initially, we computed predictions for each model independently using auto-regression and then performed a weighted average of their predictions. This approach yielded significantly improved results compared to using the individual models alone.
2. Subsequently, we considered the idea of averaging the predictions of individual models at each step of the autoregression and then using this average when expanding the window during the next autoregression step. This is because such predictions, being themselves a partial result of the ensemble, are presumably more reliable than predictions from individual models. This strategy resulted in a modest improvement over the previous method.

The models incorporated in our ensemble strategy were as follows:

1. An architecture comprising a ResNet-like part, one bidirectional LSTM layer, and an attention mechanism. Sequences were constructed and padded using the augmentation method explained in previous section 2, and we utilized a window of size 150 obtained from the autocorrelation mean with a threshold of 0,1.
2. An architecture including a ResNet-like part, one bidirectional LSTM layer, and three one dimensional convolutional layers. Similar to the first model, sequences were constructed and padded using the augmentation method of section 2 and a window of size 123 obtained from the autocorrelation mean with a threshold of 0,2.
3. An architecture featuring a ResNet-like part, one bidirectional LSTM layer, and three one dimensional convolutional layers. The sequences this time were constructed with zero-padding and a window of size 24, such that it is equal to the shorter sequence.
4. A simple architecture with a bidirectional LSTM layer, and two one dimension convolutional layers. Sequences were constructed and padded using the augmentation method explained in previous section 2, and we utilized a window of size 24, which is the length of the shortest sequence.

For the ensemble, we used weights hand-tuned, based on the performance of each of the composing models. More details on each of these models can be found in the folder models_ensembled.

By combining these models in an ensemble, we aimed to harness their complementary strengths and achieve superior predictive performance.

REFERENCES

[1] Data Augmentation: https://towardsdatascience.com/time-series-augmentations-16237134b29b

[2] Autocorrelation: A Gentle Introduction to Autocorrelation and Partial Autocorrelation - MachineLearningMastery.com

[3] Trasnformers for timeseries forecasting: Time-Series Forecasting Using Attention Mechanism (analyticsvidhya.com)

[4] Forecasting with LSTM and attention: https://www.youtube.com/watch?v=DKQerT3ybls

[5] Persistence model as a baseline: How to Make Baseline Predictions for Time Series Forecasting with Python - MachineLearningMastery.com

[6] Resnet 1d: dl-4-tsc/classifiers/resnet.py at master · hfawaz/dl-4-tsc (github.com)

[7] Fully Convolutional: https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/#:~:text=Convolutional%20Neural%20Network%20models%2C%20or,of%20time%20series%20forecasting%20problem.

TEAM CONTRIBUTIONS

All the team members worked together throughout the challenge. In the following we just state some of the focuses of each member, but please note that we all equally contributed to the tasks and most of the times we worked side by side.

Pietro Caforio:
Attention mechanisms research and implementation, sequence building approaches, architectures research and implementation, timeseries augmentation techniques, models ensembling, autocorrelation research and implementation.

Federico Giuseppe Ciliberto:
Sequence building approaches, architectures research and implementation, timeseries augmentation techniques, data preprocessing, autocorrelation research and implementation.

Luca Civitavecchia:
Architectures research and implementation, data preprocessing, models ensembling, fully convolutional trials, hyperparameters fine tuning, autocorrelation research and implementation.