# Artificial Neural Networks and Deep Learning – Challenge 2

Giada Silvestrini 10659711
Ludovica Tassini 10663137
Giulia Venturini 10680310

Group: AI Queens

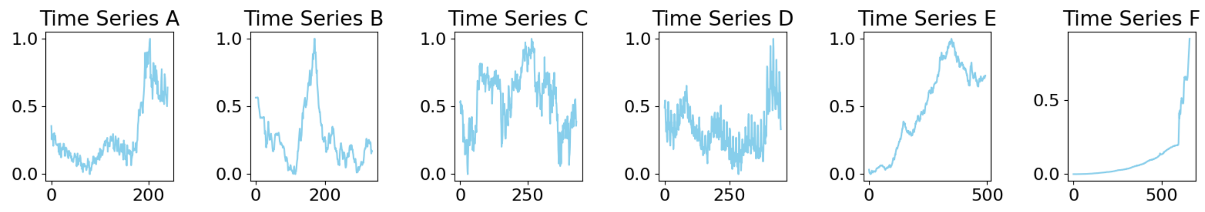
## THE PROBLEM: *Time Series Forecasting*

The assigned challenge is about univariate time series forecasting: having a dataset of 48000 uncorrelated time series belonging to six different categories, the goal is to build a model that at inference time is able to predict a fixed number of samples from new unseen time series.
In particular, during the first phase we have to predict for 60 different series (test set) the last 9 samples, while in the second phase the last 18. For this reason, the model should show generalization capabilities among all the different domains.
The evaluation metric used to assess the model performance is the *mean squared error*.

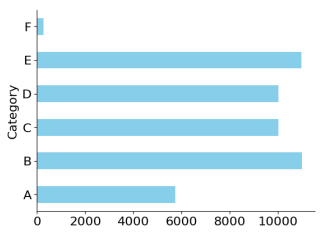
## FIRST PHASE

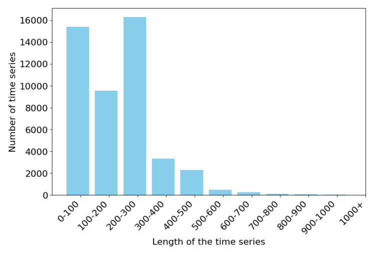
### *Preprocessing*

During the first days of the competition, we started by inspecting the datasets that were given. We started by matching each time series with its category in a .csv file and we plotted some examples and their statistics.
The following are examples of distribution of samples in time series of different categories:



We noticed that the *training_data* file contained the time series padded with zeros, so that they all have the same length. We decided to create a new dataset containing the time series without padding stored in a .npy file named *training_no_padding*: this helped us in better framing the problem.
The following are diagrams representing some interesting features of our dataset: the two histograms represent respectively the number of time series for each category and the lengths distribution.





As we can see from the first image, the “F” category is severely underrepresented, while from the second image we can observe that the length distribution is heavily shifted towards certain values (there are very few long series and a huge number of small and medium-sized ones, roughly between 0 and 300 samples long).

## *Network development*

The first approach we tried was a simple re-adaptation of what we have seen during the laboratory session, thus converting the *training_data* file into a data frame of 2776 rows and 48000 columns, as if they were a single multivariate series. As expected this method was not working, because the workload was excessive and because it was not properly fitting the assigned problem.

The second approach to the problem was inputting one time series at a time as a univariate data frame, training a basic model with that, saving the weights and iterating this procedure using the saved weights as initialization parameters of the next iteration. Since we used the *training_no_padding* file, we needed to reshape the input and resize the “window” at each step.
This method conceptually worked fine and, in fact, the model was able to train for almost 300 iterations (thus it processed 300 series). However, we were not able to conclude all the 48000 time series due to resource scarcity.

We understood that we needed to look for other strategies: in particular, we started reasoning on the shapes to be given to the model. The model we needed to train would accept as input all the 48000 time series with an input shape that reflected the required window size of 200, and an output shape that would match the required prediction length of 9 or 18. We decided to train with a prediction length of 18 even for the first phase and to crop the prediction output on *model.py*, so that we could reuse the weights for the second phase.

Following these main ideas, we modified the very first code:

- We loaded the *training_no_padding.npy* file and shuffled it: shuffling the dataset could be helpful to the training, since we noticed that the time series were clustered according to their categories. For instance, we tried using as test set the last 3800 rows and we found out that this would mean excluding from the training the entire “F” category.
- In order to save memory space, we converted all the samples of each series from type *float64* to *float32*.
- We built the sequences of each series to be passed to the model, with a window of 200, a stride of 5 and a telescope of 18. This means that each time series is scanned in groups of 200 samples at a time, every time shifting by 5 samples until the end of the series (in case the length of the series is not a multiple of the window, we padded it with zeros). For each group we selected the next 18 samples (telescope) as forecasting samples.
- We built the model.

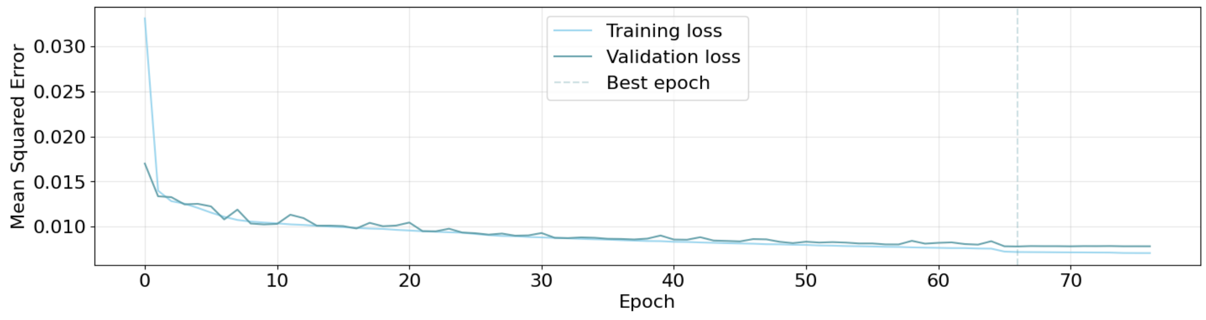
The first model we built only included one bidirectional LSTM layer (with 64 units) followed by a 1D convolutional one (with 128 units). This gave us a baseline performance that once submitted on Codalab resulted in a 0.00611 MSE.
Starting from this basic network, we experimented with some changes: we added one more 1D convolutional layer (with 64 units) just after the bidirectional one. This improved the performance of the model, since deeper architectures, possibly with multiple convolutional layers, can help capture hierarchical/local features. In fact, it ended up being one of our best models in this first phase: the MSE on Codalab was 0.00599.

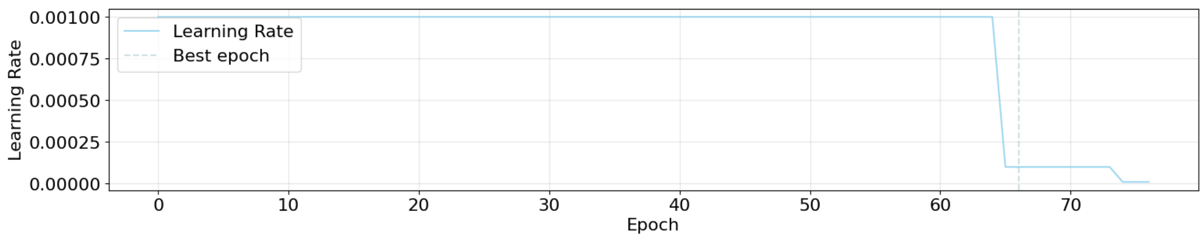
After that we made some trials by changing some parameters and layers, that unfortunately did not improve remarkably the performance (although locally they seemed promising w.r.t. the aforementioned ones):

- We augmented the time series of category “F”, both by duplicating the existing ones and by generating new series, similar to the already available ones.
- We dropped the first samples of each time series, based on their lengths, to focus on the patterns of the very last part of each series.
- We modified the number of units of the different layers, experimenting with different levels of complexity (both incrementing and decrementing).
- We added different types of layers such as: multiple consecutive bidirectional LSTM layers, two dense layers to replace the bidirectional LSTM counterparts.
- We added an attention layer following the bidirectional and convolutional ones. This model ended up being one of the most performing ones during this first phase, even though we expected the attention layer to improve much more significantly the forecasting performance.
- In order to limit overfitting, we also inserted a batch normalization layer.

We also considered trying to perform classification first, in order to develop 6 different models (one for each category): each time series in the test set is assigned its label and, according to it, the specific model is chosen. We decided not to further explore this path, because this would have decreased the generalization capabilities, and would have added the classification error to the forecasting one.
We tried to implement the forecasting procedure in an autoencoder fashion, but after a lot of trials we noticed that the performance was not so good. The issue could be the fact that autoencoders are typically applied to fixed-size input data, and they may not perform well with longer time sequences.

The following plots represent a typical training trend for our models:





## SECOND PHASE

In this second phase, we decided to submit the models that obtained the best 6 scores in the previous phase. In particular, the two most significant ones are:

- *Attention.ipynb*: 1 bidirectional layer, 2 convolutional layers (increasing the number of units), 1 attention layer. First phase: 0.0060. Second phase: 0.0115.
- *Baseline.ipynb*: 1 bidirectional layer, 2 convolutional layers (increasing the number of units). First phase: 0.0059. Second phase: 0.0123.

The other models that we submitted in this second phase had very similar performances to the aforementioned ones.
The results we obtained in this phase, as expected, almost perfectly doubled the error obtained in the first phase on Codalab. In particular, the final test errors of our models were nearly identical to the ones that we obtained locally: this could mean that they were not remarkably overfitting the given data.

## CONTRIBUTIONS

For this challenge we collaborated all together since the beginning, as for the first challenge. In the first days we focused on deeply understanding the problem we needed to solve: processing the dataset and setting up a properly working network.

Once we obtained a basic working model, each of us proceeded in testing some idea: for instance, some of us focused on improving the dataset that was given (data augmentation, shuffling and cropping), while some others focused on experimenting with the layers of the model. Doing this, we maximized the number of training experiments.
These attempts contributed in obtaining satisfying results and a positive outcome in the competition.

## LINKS

This is the link to the file .npy containing the time series without padding: training_no_padding.npy