Artificial Neural Networks and Deep Learning

Second Homework – Time Series Forecasting

This second challenge consisted in a time series forecasting problem: given a dataset of several numpy arrays whose values are sequential in time (time series) and already normalized in the range [0, 1]. The goal was to train a network able to extract information from the available data and predict the future time samples.

An initial (faulty) approach

Starting with the approach seen in the lab sessions we tried to define a dataframe from the given dataset, treating each time series as an independent column covariate. This method had a short life: the resulting dataframe dimensions were huge and incompatible with the size of the RAM available, thus this path was not feasible.

Data Inspection

Guided by this conceptual error we proceeded to inspect the available data. First of all we checked the category distribution, noticing it's widely unbalanced *[figure 1]*. Moreover, the dataset is sparse, meaning that most of the values are null due to padding, carrying no information and wasting memory. We also noticed that the absolute majority of time series were way shorter than the maximum length of 2776 samples (around 90% of the time series consist of 400/500 samples or less).

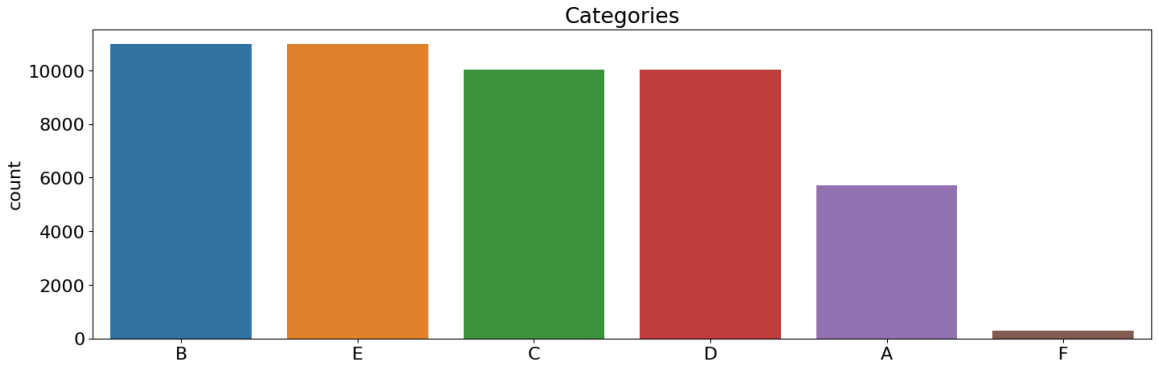

Figure 1

To deal with class imbalance we thought initially to build a classifier and then 6 different predictors (one per class) to fully exploit the categorization of the series. This idea was quickly abandoned since, according to informations given on the Codalab page and forum:

- data may be mislabeled hence requiring unsupervised classification, this was confirmed by bad results achieved in brief experiments with a supervised classifier
- data is mono variate, meaning that there shouldn't be any significant difference in the information content coming from different classes, this fact was enforced by training the model exclusively on a single class and later with a visual inspection. To be more specific: given the information that the 60 time series in the test set were equally distributed with respect to category, despite training our model with only B-category series, results were overall decent in predicting samples from series of other categories.

Development Phase

First model approaches followed the architectures seen in the lab sessions using only one category for training (‘B’).
We used the function ***build_sequences*** to create training data and associated labels, giving to the model consistent data shapes to train on. From the time series in the dataset the function extracts a 200 samples window which is going to be the test set, input for our model, then the subsequent 9/18 (based on phase) samples of telescope i.e. the values we want to predict, the test set, also saving the class from which this values were extracted; this is repeated along all the time series moving the window by a *stride* parameter. We chose the window length according to the data shape of the test set to grasp all the information available. This may seem a mistake because we give to the model information about the test set, however it’s not the case because the pipeline works regardless of input size (as we see on the dataset), moreover if the information content in 200 samples wasn’t enough for the forecasting the problem would have been malposed, or if fewer samples would have been sufficient the model will perform correctly anyway.
As usual, techniques of Early Stopping and Learning Rate Reduction on Plateau were included in every model to improve generalization and avoid overfitting.
The experiments involved using: sequences of LSTM layers, bidirectional layers, sequences of Conv1D and GRU,trying also to add dropout and attention layers, unfortunately without positive feedbacks.
Since we discovered that a strict division between categories wasn’t necessary, we started training our model experiments with all the series available, to deal with with class imbalance we adjusted the stride for each class in order to have less unbalance in the number of sequences/labels for each category’s series i.e. applying a very small stride to ‘F’ category series with respect to ‘B’ ones.

Another small data inspection

Based on the unexpectedly good results in training the model with only a single category, and after experimenting with the whole set of series, before proceeding further we tried to visualize the series ourselves, to detect if and how the categories were meaningful since results, length of series and valid periods suggested quite the opposite; in order to do this, we simply plotted some series for each category. *[figure 2]*
Without noticing patterns of some sort, we lately decided to adopt a risky but in the end compensating choice: create new categories for our time series.

Figure 2

A new approach

This new approach consisted in creating new categories based on the length of the de-padded time series, we chose to have 5 new categories, from XS to XL.
(len < 101, 100 < len < 201, 200 < len < 501, 500 < len < 1001, 1000 < len)
Also, we fixed a few small errors in the previous models, such as:

- basing strides on old category division was not a wise choice since as we have seen, length and (original) category of a series are not correlated, while now it makes sense to have more sequences generated by short series thus less stride inside the function.
- extracting the “real” series considering only the valid period caused, in the ***build_sequences*** function to generate only few data to train with. Taking the series before the actual first valid value allowed the creation of more sequences/labels, especially again for short series.

These modifications came at the price of having a lot more padding in the training data, anyway, the final model was able to perform better with them. *[figure 3]*

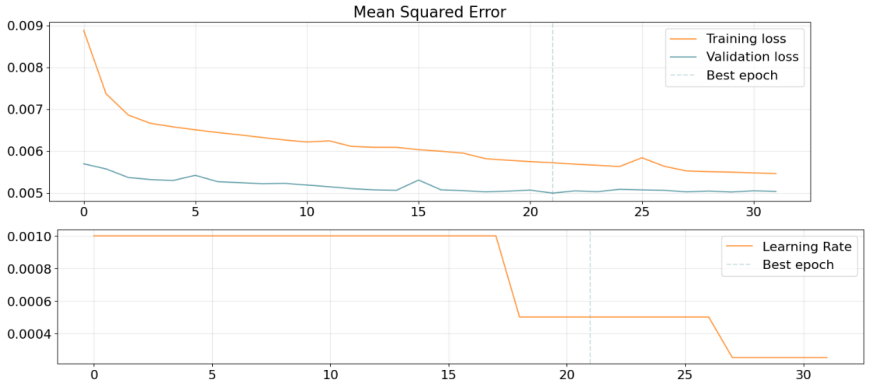

Figure 3: Best submitted model in Development Phase

Final Phase

In the final phase, we submitted our best model so far, which consists of a sequence of 3 LSTM layers, each doubling in the number of units starting from 32, followed by a Dense one as output layer. This was re-trained to predict this time 18 future samples, but once submitted, we noticed a way worse performance than expected. To solve this, we gave a “second chance” to autoregressive approach and this time, predicting only the value on t+1 and shifting the input series one by one in an autoregressive fashion until getting 18 future samples gave us a massive improvement in performance, ending the competition with:

- MSE = 0.00933014
- MAE = 0.06992757

Increasing the telescope (2-3) didn’t improve performances.

Contributions:

Overall we didn't decide to split tasks a priori, so everyone participated in every step of the task. In any case, these below are the aspects that each one of us feel like he focused on more during this first competition homework.

- *Luca Di Stefano*: Data Inspection and preprocessing, Experiment Execution, Model Architecture Design
- *Gabriele Romano*: Experiment Execution, Model Architecture Design, Hyperparameter Tuning, Report Tweaking
- *Mattia Pezzano*: Preprocessing and Data Parsing Supervision, Model Architecture Design, Report Drafting
- *Riccardo Villa*: Data Inspection/Visualization, New Category Division, Report Drafting

References:

1. https://karpathy.github.io/2019/04/25/recipe/ Tips for Work Organization
2. https://keras.io/api/ Keras Documentation