A2NDL - Homework 2 - NeuroNauti

Objective

The objective of the second Homework is to forecast future instances of the input time series, leveraging their historical data; the task requires building a forecasting model able to forecast multiple, unrelated, time series.

This requirement is declined into the necessity of building a model that is able to generalize between the different categories and must be unrestricted in the length of the forecasting.

Dataset Analysis and Preprocess

The dataset analysis requires padding each dataset to the same length and applying a windowing technique to extract as many sub-timeseries of the desired length as possible.
Furthermore, the dataset is split into three files:

- *training_data.npy*, containing the time series;
- *categories.npy*, containing the the category of each time series;
- *valid_periods.npy*, containing the start and end index of the time series;

Each file contains 48000 rows.

Since each time series has been padded to have the same length in the provided dataset the corresponding valid periods must be extracted by referencing the *valid_periods* file. In the same way, the category of each time series is extracted from the *categories* file.

To extract and preprocess the data, the following operations were performed for each category in the dataset:

- The indices corresponding to all series in a category are extracted and shuffled
- For each index, the correct start and end time are gathered
- The time series is fully extracted and padded, if necessary, to make it divisible in windows of the same size
- If the index of the time series is less than the split point, defined as the number of time series needed in the test set (10% of the size of the dataset), then it is resized to be 219 samples long (200 for the model input and 19 for the prediction test)
- Otherwise, the time series is windowed by creating sub-sequences, each of them 209 samples long (200 for the model input and 9 for validation), with a stride of 10

After this preprocessing step, 245280 windows have been extracted, divided into 6 categories (A, B, C, D, E, F). The size of these categories was respectively 47442, 37677, 56607, 61295, 61295, 1182.
To perform training and validation, a split of 90%/10% has been used, resulting in a total of 242610 and 2670 time series have been respectively placed in the training and validations sets.

Approach

To tackle the problem, we swiftly started by implementing a Conv1D model with a layer of bidirectional LSTM, followed by a cropping layer to provide the required number predicted values; this however proved to be an insufficient solution, as the training clearly showed quick improvements over the first couple of epochs but better values could not be achieved over the subsequent epochs, despite implementing techniques such as reducing the learning rate on plateau.

We initially tried to augment the dataset, but the only possible data augmentation technique available on time series is adding gaussian or white noise to samples in each time serie; while this later proved to be a crucial solution to reduce overfitting in the training phase, did not help us in improving the accuracy.
We finally settled on a system with:

1. A batch normalization level;
2. A Gaussian noise level for the pre-processing with a standard deviation of 0.01;
3. A 1-dimensional convolutional layer with 256 filters and a kernel size of 3x3;
4. A 1-dimensional convolutional layer with 1024 filters and a kernel size of 3x3;
5. A max-pooling layer;
6. A LSTM layer with 512 units;
7. An attention layer with softmax activation;
8. A dense layer with a number of units equal to the number of samples to be forecasted (9).

The hyperparameters have been defined in the tuning phase *(refer to the next subsection)*.

To provide an estimation of the next 18 points (as per the request of the phase 2 of the problem) we implemented an auto regressive prediction technique:

1. The first 9 data points are predicted from the first 200 samples of the test serie;
2. The last 9 data points are predicted from the last 191 samples of the test serie concatenated to the prediction made in the previous step
3. The two predictions are concatenated to make a vector of 18 predictions.

Tuning

In order to find the optimal values for the hyper parameters in the model, we used Python's *keras-tuner* library to define:

1. The number of filters in the first convolutional layer;
2. The number of filters in the second convolutional layer;
3. The number of LSTM units.

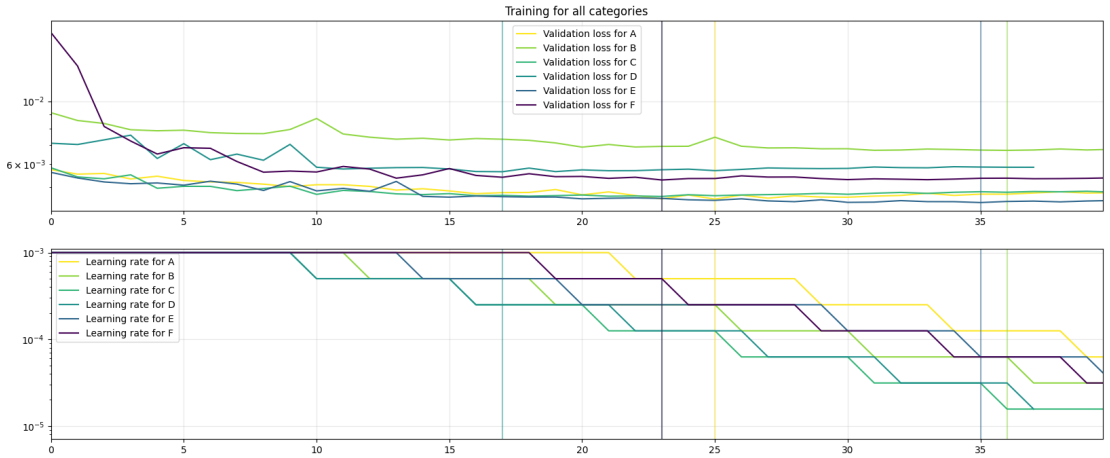
We used the built-in Bayesian Tuner over a reduced number epochs (15) and a randomly sampled subset of the dataset (25%) containing all the time series, without discriminating by category.

Training

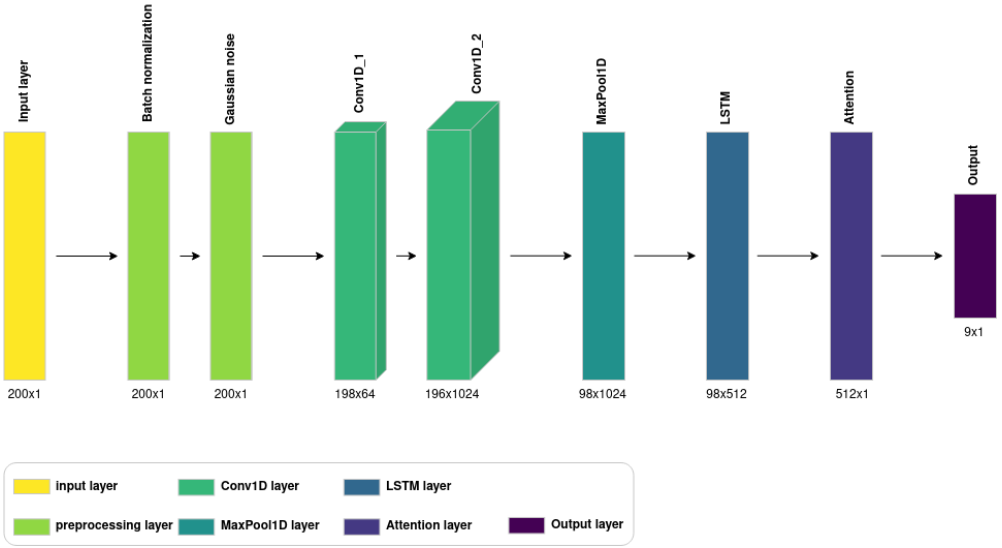
7 different models have been trained:

- One on the full dataset, containing all the time series of every category;
- The remaining six on a single category each;

The training has been performed with a 90%/10% split on the dataset and the considered loss function is the mean square error (MSE). The plots emphasize the amount of overfitting in the validation set that, although limited by the Gaussian noise preprocessing, has not been completely removed.



System Architecture



Code

All the code used to extract, clean, and analyze the data, as well as define, tune, train, and evaluate the model is in the model.ipynb file in the form of a jupyter notebook.
Each section is titled and commented as much as possible to describe what is going on in the code cell.

Other relevant attempts

A few other attempts were made in building a better model, but failed:

1. **Robust Scaling**: we tried adding Python's sklearn robust scalers to identify and remove outliers in the dataset; however, it failed improving our results;
2. **Autocorrelation** in windowing: we tried leveraging autocorrelation in time series to center the windowing process; despite finding the peaks in the autocorrelation, we were not able to translate this idea into any kind of improvement in the model;
3. **ARIMA model**: to remove outliers, we tried using an AutoRegressive Integrated Moving Average (ARIMA) model; however, we were not able to tune the parameters correctly and get any noticeable improvements
4. **Differentiated evaluation**: since some categories have a better evaluation with the model trained on the whole dataset, evaluate them with it and use the models trained on the single category for all the others; while this worked against the local dataset, did not perform as good in submission
5. **Models trained on a single category only**: since the dataset is provided with the division of the time series in classes, we trained 6 different models (each on a different category) and predicted the next points according to the class of the test time serie; this once again worked on the local test set but not on the remote one, probably due to overfitting in the data

Who did what

- Matteo Beltrante: data analysis, model definition and tuning, report writing
- Lorenzo Rossi: data analysis, model definition and tuning, report writing