POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Time Series Forecasting

AN2DL 2ND CHALLENGE

TEAM NAME: POLLIKUNGFU

Alessia Marino: 10933660, Antonino Fabio Coppola: 10926052, Lorenzo Manoni: 10740351

---

1. Introduction

This document is intended to explain the process behind the building of the final model for the second challenge of the Artificial Neural Network and Deep Learning course. The objective of the assignment was to build a model that correctly forecasts signals with the minimum MSE (Mean Squared Error) we could reach by uploading the models to the CodaLab platform which will evaluate the model onto a test set that is unknown to us.

2. Dataset Specification

The Data provided to us was divided in three files .npy that are NumPy arrays, the "training_data" which contains the Values of the signals, the "valid_periods" which contains the start and end time of each signal, and finally, the "categories" which contains the category to which the signal belongs. The data set provided consists of a total of 48000 time series of length 2777 belonging to one of the 6 categories (A, B, C, D, E, F) (Figure 1).

Each sample is divided into sequences of length 200 and the amount of sequences depends on the length of the signal. No further details were provided on the origin of the dataset, therefore a more detailed analysis was requested. It was found that the samples were divided by class and the sequences were continuous, as shown in the images below. The dataset is unbalanced because we have a different number of signals for each category and these were also of different lengths. For this reason we had to preprocess the signals by adding padding at the beginning of the sequences.

Category	**A**	**B**	**C**	**D**	**E**	**F**
Samples	5728	10987	10017	10016	10975	277

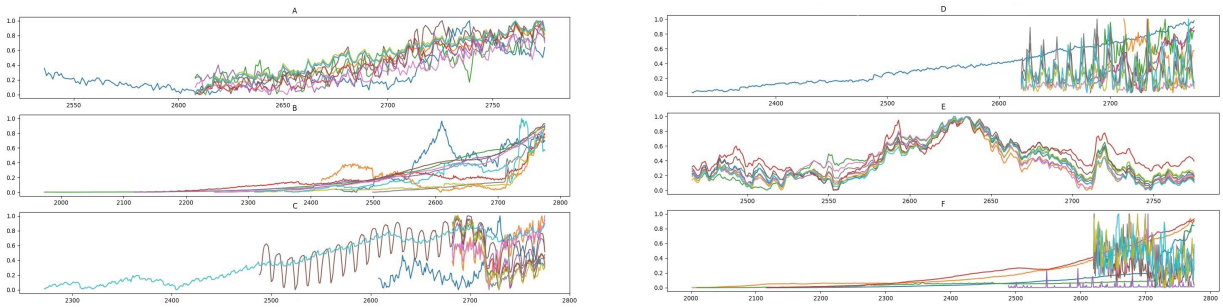

Figure 1: Plot of 10 signals for each category

3. Dataset split and prepossessing

3.1. Dataset split

We decided to split the dataset into train and validation sets. We kept the 5% of the data for validation and the remaining 95% for the training. The data we were given had already been normalized so there was no need to make any changes to the values. For the construction of the sequences several techniques were adopted, we carried out several tests to verify which algorithm managed to divide the signal most effectively. In the end, we adopted a system that receives only the valid signal, eliminating the initial padding and adding it again only in case the signal was not long enough to respect the length of the window.

3.2. Preprocessing

We tried different techniques for preprocessing the dataset:

- We tried to detrend the signal (Figure 2) in order to let the model learn simpler signals and generalize more on the data presented;
- We tried to remove outliers from each category, that are too different from the z-score ($z = \frac{x-\mu}{\sigma}$).

Both techniques were unsuccessful since they performed poorly with respect to the unprocessed dataset. We even tried to remove seasonality from the signal and during the training seemed to perform slightly better, but the machine for test on CodaLab did not have installed the library used to perform such operation so this option was abandoned and never actually tested properly.

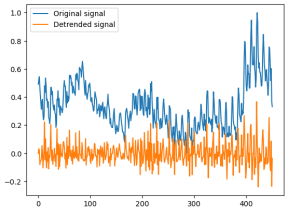

Figure 2: Signal detrend

4. Experiments

Several attempts were made during the build of the models and training phase. In the beginning, we trained different models one for each category (We call this model the "Categorical Model"), because we noticed that the category of each sample was provided during the test phase on CodaLab. Since the previous attempt was a lot worse concerning the results present in the "Results" section of CodaLab, we decided to train a unique model with the entire dataset, no more taking into account the category. This outperformed the previous submission and was more in line with the submissions of the other participants, so we continued to use this technique for each trial.

Since we had to make a prediction of 9 steps in the future with a window size of 200, we used different techniques in order to achieve the best performance. In the beginning, we simply trained with a telescope of 9 (for the first phase of the challenge). We then implemented an autoregressive model that makes a prediction of 1 step in the future and uses this as input 9 times, but this ended up increasing the final error since the errors produced at each prediction were summed up. This technique wasn't abandoned for good, since in the second phase the steps to predict were doubled. So we reused this to make the second 9 predictions from the previous ones. We also tried to retrain the models used in the first phase to obtain the 18 future values in a single prediction but this solution performed worse than the previous one.

Regarding the choice between LSTM and GRU, the first architecture was the one with the best results. We made a lot of tests with different models with Long Short-Term Memory (LSTM) and Bidirectional LSTM (BLSTM). At the end of each model, we had a Convolutional Neural Network (CNN) where we tried different combinations of layers such as BatchNormalization, Dropout, GlobalAveragePooling, and Dense, also trying to set different parameter values for each of them. In order to use LSTM layers we applied a reshape of the input signal because the LSTM wants a tridimensional shape as input, therefore the shape from $(BS, 200)$ was modified into $(BS, 200, 1)$.

We also tried to use the Masking Layer to hide the irrelevant parts of the sequences and to mask these padding values so that the model does not consider them during training but we realized that the computational time increased significantly and the performance decreased.

5. Final model

5.1. Hyperparameters

- BATCH_SIZE: we used a batch size of 256 samples;
- EARLY_STOPPING_PATIENCE: we used a patience of 12 epochs;
- REDUCE_LR_ON_PLATEAU_PATIENCE: we used a patience of 10 epochs;
- MAX_EPOCHS: we used a maximum of 200 epochs, but due to early stopping this maximum was never reached;
- OPTIMIZER: we opted for Adam Optimization due to its speed, reduced memory requirements, and ability to deliver strong performance;
- LOSS: we used Mean Squared Error, which significantly penalizes larger errors to reduce the influence of outliers during model training;
- LEARNING_RATE: we used a learning rate of 1e-3 in order to avoid fast overfitting, so we gave the model the time to adapt itself to the dataset in small steps.

5.2. Model Layers

The first layer is the Input layer to instantiate a Keras tensor. Then we added two Bidirectional LSTM layers by wrapping the LSTM layers into the Bidirectional ones with 128 units for each LSTM. After that, we add an LSTM layer with 64 units. Then the Convolutional Neural Network (CNN) begins, it is composed of a Convolutional layer with 64 filters, a kernel size of 3 and the "Relu" function as an activation function. After that, there is a Dropout layer with a rate of 0.3 to reduce overfitting and a Global Average Pooling (GAP). At the end of the model, there is a Dense layer with 9 units because we want to predict the future 9 steps.

6. Comparison between models

To conclude, we compare the results obtained by the best models, for each technique used. The loss and the validation loss are based on the split of the dataset into the training and the validation set. With the final model during the second phase, we achieved 0.0097 of MSE.

Models	Train (9)	Validation (9)	Test (9)	Test (18)
Categorical Model	0.0073	0.0062	0.0142	-
Final Model with detrend	0.0031	0.0029	0.0058	0.0116
Final Model	0.0047	0.0043	0.0049	0.0097

Table 1: MSE Values for Train, Validation, and Test for the number of telescope samples.

7. Contributions

Each of the three team members significantly contributed to the project by delving into and experimenting with various types of neural networks. Each individual played a pivotal role in exploring and testing diverse functions, thereby making a substantial contribution to the research and development of the ultimate solution.
We thoroughly enjoyed the process of discovering and implementing new techniques, enabling us to enhance our models by leveraging the wealth of knowledge gained during our coursework, while also actively seeking out new insights and information.

References

[1] Time Series Forecasting: https://www.tableau.com/learn/articles/time-series-forecasting

[2] how-to-improve-deep-learning-forecasts-for-time: https://towardsdatascience.com/how-to-improve-deep-learning-forecasts-for-time-series-1799e3975d7c

[3] Seaborn library: https://seaborn.pydata.org

[4] Time series forecasting TensorFlow: https://www.tensorflow.org/tutorials/structured_data/time_series

[5] Forecastegy: https://forecastegy.com/posts/deseasonalizing-time-series-data-python/

[6] Bidirectional layer: https://keras.io/api/layers/recurrent_layers/bidirectional/