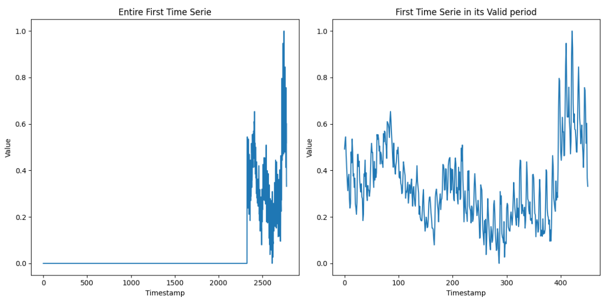
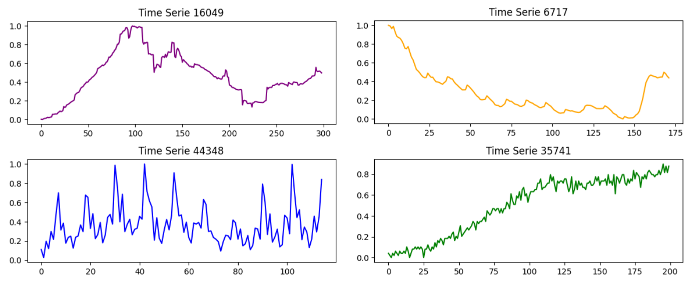
In this homework we were asked to predict future samples of 60 input time series of length 200, by designing and implementing a forecasting model able to learn how to exploit past observations in the input sequences to correctly predict the future (9 samples for the “development” phase and 18 for the “final”).



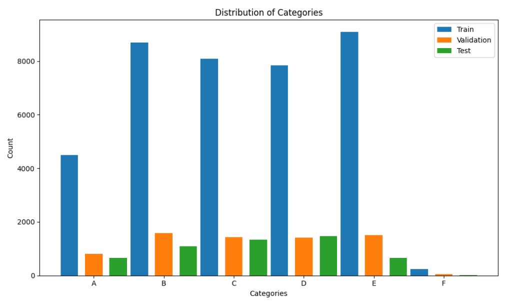


The data provided for this homework consists of 48000 time series of length 2776, with their relative category (6 in total) and their valid time periods, since time series are padded with zeros to reach the length of the longest one.

Data Processing

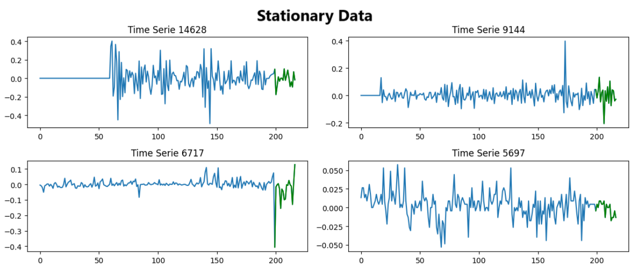
At first we ignored the categories and focused on preparing data for training, removing the zero paddings (but not for time series having valid periods shorter than 218) and splitting the provided data into training, test and validation set, with a 73,12,15 split respectively.
We also augmented the number of time series by using different stride lengths (5,10,50,100,218) and this made it clear that the more flexible and powerful training sessions were carried out using a stride of 218.
Our opinion is that using a **stride of 218** avoids increasing the number of time series too much, thus **speeding up** the **training** process, and also using a stride as long as the data fed into our architecture and the predictions (200 for inputs, 18 for predictions) **avoids** having **correlations** between data, since each time series has its unique timestamps and so there is no repetition.

Next we introduced the **use of categories** in the training of our data, so we provided our models with different time series for each category using a numpy array for indexing the categories. While doing this we also needed to **augment** the number of time series for **category F** because, as the plot shows (last 3 columns), the samples provided were less than the others. This solution is revealed to be **pointless** and not performant since time series of different categories are uncorrelated.



We then retraced our steps and brought some improvements. Since we noticed a big difference between local results and those obtained on CodaLab, we tried to reproduce a more **balanced** and similar **test set** by using a uniform distribution of test samples per each category(100 per category, except for category F with only 39 due to limited data provided). This resulted in a more loyal performance comparison.

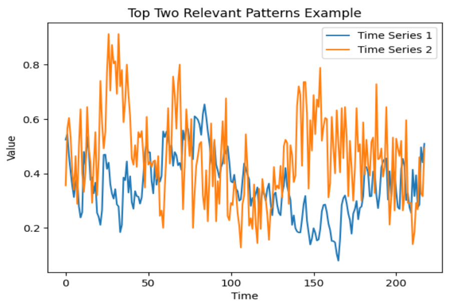
After difficulties in improving, we decided to change **data normalisation** using **logarithmic scaling** and **robust scaling**. The first did not show any gain, while for the second it is a more complex matter. By performing column-wise, on each time series it applies a scaling calculated on the timestamps and not by singular time series, and the reverse transformation (de-normalisation) cannot be done because we don't know the interquartile ranges and medians of the actual values. We could probably have applied robust scaling row by row to improve performance, but we decided to look at other alternatives instead.



A quite important step for the development of the project was the **stationary data**, we made data stationary and changed the task in some sort of "deltas predictions". Models trained on this data performed quite well in identifying prominent temporal patterns but lacked flexibility in unusual time series and in particular it is strongly dependent on the first prediction since all other deltas predicted depend on it.

This last point gave us an interesting idea to improve the performance of our models that will be further discussed.

At last, thanks to the concept of **magnitude** and frequency, we extracted the **patterns** that are considered more **relevant** for an eventual training (1 for each time series). Note that while doing this, for the first time, we also removed the time series that have valid periods shorter than 218.

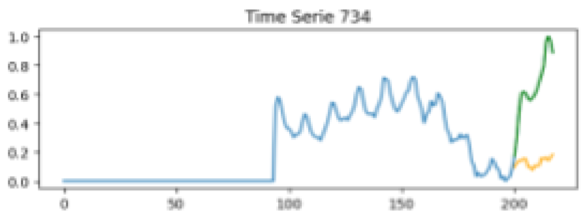


Models

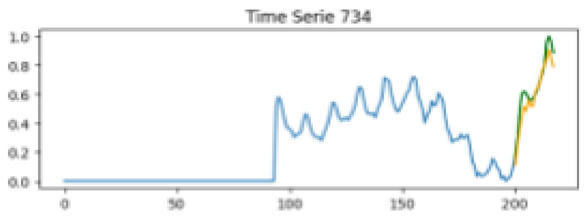
The first decision we made was to make our models predict 18 samples even for the first phase of the competition because in this way they would have been able to gain a greater ability to generalise well (the only changes applied to the *model.py* was to *return out[:, :9]* instead of *return out during the first phase*).

Baseline model

The first approach to the task consisted in trying with simple models. The first of them to show good results consisted of a simple Bidirectional LSTM layer. Since it wasn't good at predicting irregular series, we tried to increase its complexity by stacking 2 consecutive Bidirectional LSTM layers (*MODELS → Baseline Model*). Increasing the model complexity with a second layer, the prediction mse resulted to be of 0.00477 on CodaLab's development phase.



Prediction made using only one Bidirectional LSTM layer

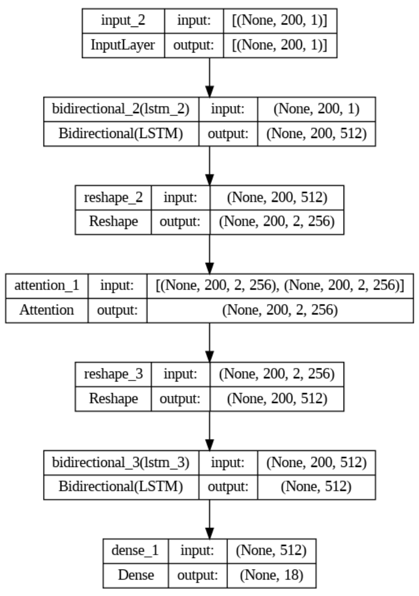


Prediction made using a stack of 2 Bidirectional LSTM layer

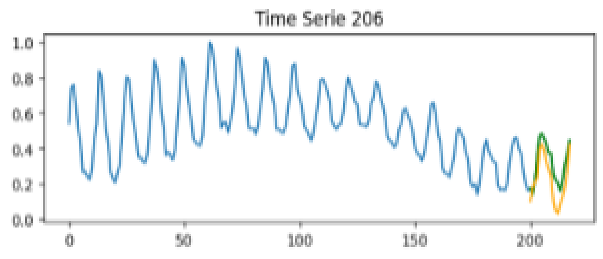
To improve the performance of the Baseline Model we tried to tune the hyperparameters changing the number of LSTM cells, learning rate, batch size and tuning the callbacks. The best results were obtained with both 256 and 128 units for each of the Bidirectional LSTM layers. We also tried L2 regularisation and dropout to prevent overfitting, anyway it didn't lead to any improvement. *(see MODELS → other models: Regularized baseline model)*

Best Model

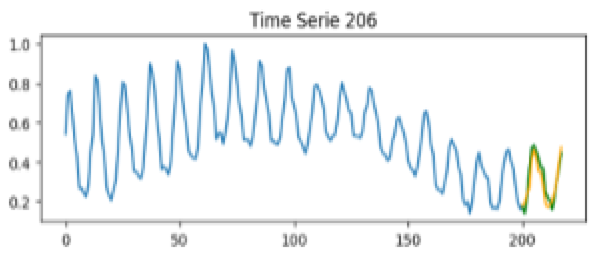
After the laboratory session on the Attention mechanism, we decided to try it.
The model we created consisted of a **Bidirectional LSTM** layer capturing dependencies and patterns in both directions of the input sequence, followed by a **Reshape** layer, which was used to provide appropriate input to the **Attention** layer.
This layer is used to give a **higher importance to the last samples** of the series and to **ignore** the **zero paddings**.
The output of the Attention layer is **reshaped back** to its original shape and is fed to another **Bidirectional LSTM** layer that extracts its dependencies.
The 18 values of each prediction are then returned by the **Dense** layer with a **linear activation function**.


input_2 | input: | [(None, 200, 1)]
InputLayer | output: | [(None, 200, 1)]
bidirectional_2(lstm_2) | input: | (None, 200, 1)
Bidirectional(LSTM) | output: | (None, 200, 512)
reshape_2 | input: | (None, 200, 512)
Reshape | output: | (None, 200, 2, 256)
attention_1 | input: | [(None, 200, 2, 256), (None, 200, 2, 256)]
Attention | output: | (None, 200, 2, 256)
reshape_3 | input: | (None, 200, 2, 256)
Reshape | output: | (None, 200, 512)
bidirectional_3(lstm_3) | input: | (None, 200, 512)
Bidirectional(LSTM) | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 18)

For the training process we used a **ReduceLROnPlateau** with an initial learning rate of 1e-3 reduced down to 1e-5, an **EarlyStopping** with patience 5 and a **batch size** of 32.



Baseline model prediction



Best model prediction

Enhance Best Model

In order to improve the performances of our model we trained a model with 2 Bidirectional LSTM layers to predict just 1 sample (*see MODELS →)* (this idea originates from the stationary data problem with the first sample mentioned above).
This model has been used to replace the predictions of the first timestamp of the *Best Model* since it performed better.
We found out that exploiting this model to predict the following timestamps was useless since it would propagate the errors produced at the previous predictions. *(see MODELS → other models: 2 predictions, 3 predictions).* This gave us an mse of 0.00468272 in the development phase and 0.00863581 in the final phase.

Conclusions

In conclusion, the data processing part had a major role in obtaining good results.
This was possible also thanks to the data normalisation and the dimensioning of the window.
The best model we obtained used techniques such as bidirectional layers, attention and callbacks for its training, which were fundamental to have a model that generalised the time series well and made predictions based on past trends and regularities.

Team contributions:

- **Simone Callegarin:** Data preprocessing part and normalisation (stationary, robust scaling and others), test with ResNet like architectures, test with multiple different architecture and combinations (GRUs, LSTMs, BidirectionalLSTMs, CNNs, FCNs and others), use of Attention and MultiHeadAttention layers.
- **Sabrina Azzi:** Data preprocessing, different types of LSTM and BiDirectional LSTM models (including model shown in class), tried different dimensions of strides of data, tested GRUs, CNNs, attention mechanism.
- **Gianvito Caleca:** Tried different strides of data, hyperparameters tuning, model selection (tried different layers and hyperparameters, MultiHeadAttention, masking..), regularisation.
- **Nicolò Caruso:** Various experiments on the LSTM units cell number, on the number of LSTM layers and on the usage of convolutional layers in many configurations (start, middle, end position) . Also trials about the use of LSTM with more unit cells compared with Bidirectional LSTM and different window/strides from data.