Ai Coders

THE TEAM

- Alessio Parato
- Jan Reichle
- Matteo Deò (Not participating)

THE PROBLEM

The assigned problem is about forecasting of time series: given different time series characterized by specific patterns, it is necessary to be able to identify them and predict what trend the time series could have in the future; the further the forecast goes into the future, the more the uncertainity increases.
As we will see, useful metrics in this type of exercise are MSE (mean square error) and MAE (mean absolute error).

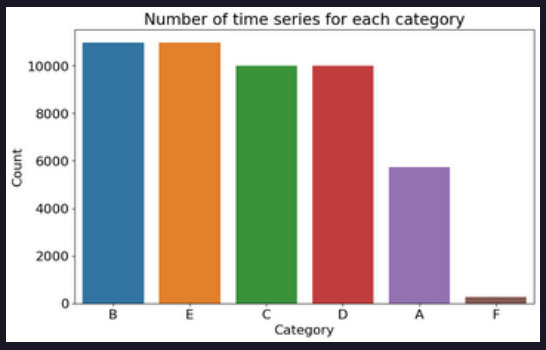
INSPECTION AND PROCESS DATA

We are given three files:

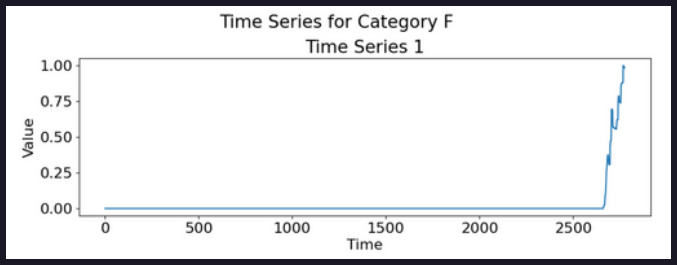
- training_data.npy -> (48000, 2776) 48000 time series of length 2776
- valid_periods.npy -> (48000, 2) containing for each of the time series the start and end index of the current series, i.e. the part without padding
- categories.npy -> (48000,) containing for each of the time series the code of its category. Categories are {'A', 'B', 'C', 'D', 'E', 'F'}

After uploading the files, we combined the information into a pandas dataframe containing the time series, category, start and end indexes.
We then created a graph to count and display the time series for each category.
As you can see, categories F and A are those with fewer time series.

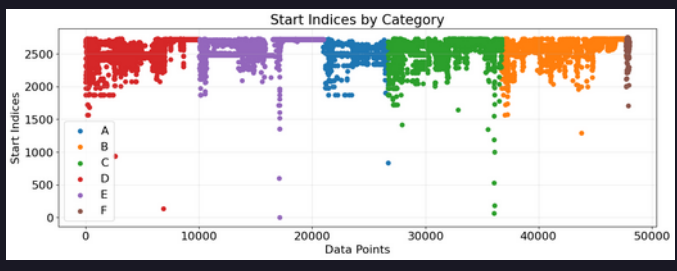
```
Category B: 10987 values
Category E: 10975 values
Category C: 10017 values
Category D: 10016 values
Category A: 5728 values
Category F: 277 values
```



Visualizing some time series of the categories (like the one in the image) we notice some things:
Time series have variable padding before the useful values and the values are already normalized to [0,1].
We will ignore the padding later in the bild_sequence() function.



We also created a visualization of the starting indices (the final ones were all equal to 2776)
As you can see, there are some time series with lower start indices and therefore they have much more values.
Given that there are not many of the latter, to keep time series of more homogeneous length we have removed those with start index < 2000. If desired, this step could be omitted.



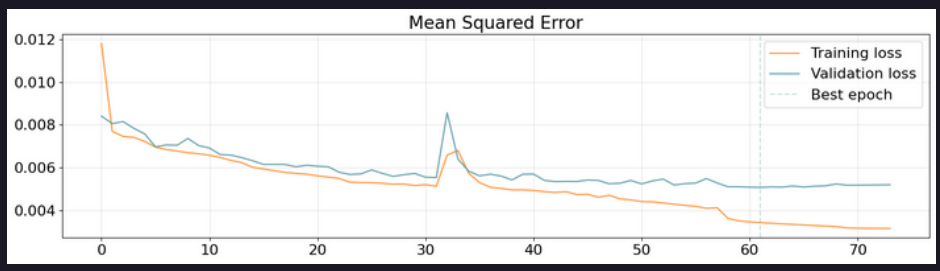
Having the data normalized to [0,1] we wanted to create a function to see if there were any "outliers" or values that deviate significantly from the average. To do this we used trashold = 3 * standard deviation as the value.
We found that for each category the outlier values were few: between 0.1% and 0.4%

The split between train and test was then performed using 10% of the time series for each category.

The build_sequence() function is used to create the sequence correctly from the time series. In practice, the time series is divided into smaller parts with the window parameter, telescope indicates how many points we want to predict after the window, the stride parameter is used to space the windows. With the start and end indices, we were able to select only the real values of the time series ignoring the padding.
By doing this we obtain sequences in the right format to use for the fit

LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to work with sequential data, such as sequences of text, audio, or time series.
We have tried different variants with different layers of LSTM with different quantities of units (32, 64, 128, 256).
At the end, there is a Dense layer as the output layer of the network.
As loss function we used Mean Squared Error (MSE) and the Adam optimizer.
The best results between the various categories were obtained with a single LSTM layer composed of 256 units.



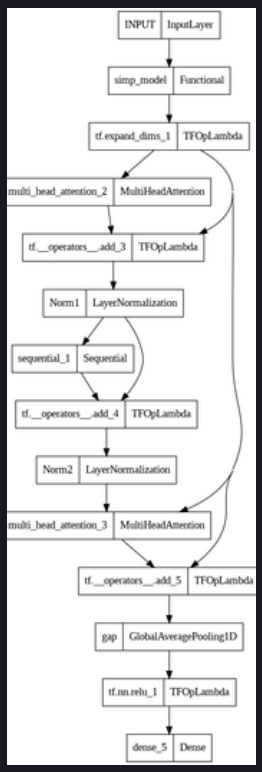
BIDIRECTIONAL LSTM

As a second step we wanted to test a variant of the LSTM, the Bidirectional LSTM which in practice reads the input sequence both forwards and backwards. The architecture of the model remains very similar to the previous one and this time too we tried with various layers using different values of units, however the results were not better than the previous LSTM. We also tried adding a dropout layer which sometimes improved the performance a bit but still remained below that of the simple LSTM

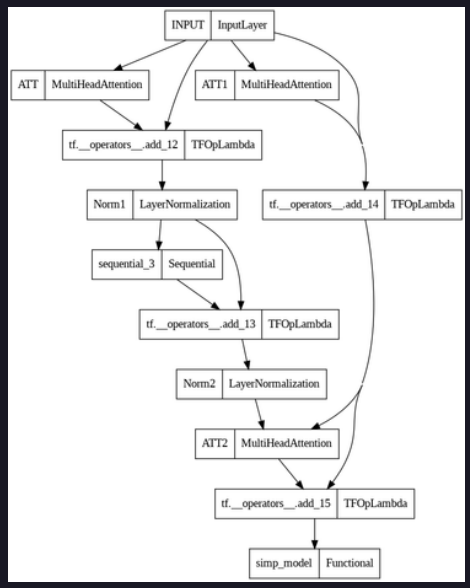
BIDIRECTIONAL + CONV1D

We then tried to add the use of Conv1D layers to the Bidirectional LSTM model, they work in a similar way to Conv2D for images but by searching for patterns within the temporal sequence through filters.
This time too, different layers with different filters and kernel sizes were tested, however it was not possible to obtain better results than the simple LSTM

ATTENTION MODEL



The plan was to recreate the attention model from the class. Now the question was: What should we put as an input or where should we add our trained model?
There are two options here. First, we could add our trained model at the front of the Encoder and Decoder or second at the end. We did both.
First we added our model in the front of the Encoder and Decoder, as you can see on the left. Then the encoder can concentrate on the 18 points and decide which are important (Attention/Homework2_Attention_Plan1.ipynb).
For the training, we decided again to train with all categories simultaneously first and then with each category individually. On the test set we got good results but not at the online submission.
As next we added our trained model at the end of the Decoder (Attention/Homework2_Attention_Plan2.ipynb). In this case the attention model would decide which input points would be important.
The idea was good, but the training took so much time that Colab was always cancelled.



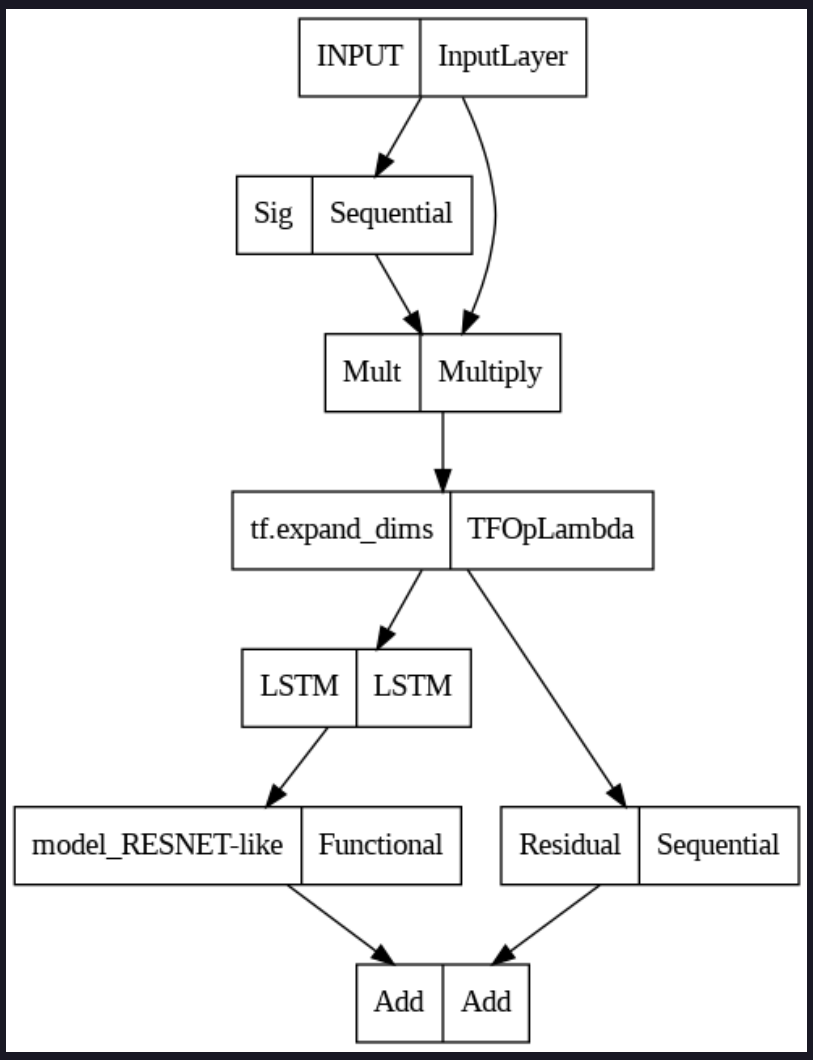
FROM CLASSICAL CNN TO A COMBINATION

We also want to know, if it is possible to get good results with the classical CNNs as we got in the case of picture classification. Since the Forcasting problem is a regression problem we decided not to use BatchNormalization layers. The idea was to look at the data not from time to time as we did in a recurrent model but to look at some datapoints at once.
We decided to use only a simple model, so that we have short training times. First we trained the network with all categories simultaneously and then with each category individually to specialise the network for each category. With this training we were able to get smaller mean squared errors in each category as you can see in file 'Homework2_Cate.ipynb'.

As a modification we added some skip connections to get a ResNet-like CNN (Homework2_Cate_RESNET.ipynb). In this way, we were able to reduce the MSE again slightly.

As the next step we added two dense layers in the beginning, one with the relu activation function and the other with the sigmoid activation function. We multiplied the output of the last layer with the original input layer to get a weighted input for the network.
We wanted to recreate the attention layer in a very simple way. We also added an extra new 'path' by adding from the original input a Conv1D layer and one dense layer. At the end we calculated the average of both outputs.
After multiple trainings we calculated the MSEs of our test sets and got worse results, we think that we got overfitting on the training sets.

Finally we created one last model which combines all of the other ideas (Homework2_Main1.ipynb).
In the beginning of the model we would like to weight the input with a small model called 'Sig' which stands for 'Signal'. It is a very simple model consisting of two dense layers. The last one must have 200 neurons and we decided to use the sigmoid function as activation function. We multiply the output of this model with the original input to get the weighted input.
As the main part we used a ResNet-like model as we created in the other file 'Homework2_RESNET.ipynb', eight Conv1D layers and one dense layer. In front of this model, we added a LSTM layer for the recurrent part of the hole model.
At least, we added a model called 'Residual'. As the input for this model, we took the weighted input. The output of the 'Residual' model we added to the output of the ResNet-like model.
The main idea for this part was to create a model which calculates the error of the main-model. To achieve this we used BatchNormalization in the end of the this model.
The complete model can be seen on the right side.



First we trained the model with all categories simultaneously and then with each category individually to specialise the models as we did in the other files. But we got worse results at the online submission.
So we trained the model again with all categories. Now we have also received a smaller MSE on our test set, but this was not the case at the online submission. We got the MSE 0.0071 but with the first training we reach MSE 0.0063
To improve this, we increased the number of training points of each category. Unfortunately, this did not help either. Again we got a worse MSE of 0.0066 at the online submission.

In this process, the model that was only trained for a relatively short period of epochs and which was trained with all categories at the same time was the best.
More precisely, the model was trained for a total of 95 epochs in four training sessions. If we have trained all epochs at once, the programme would have been cancelled by Colab again due to too much time and the training would probably have ended earlier due to EarlyStopping. After the training we got a MSE of 0,0078 and a MAE of 0,056 on the test set. On the online submission we got a MSE of 0,0063 and a MAE of 0,060.