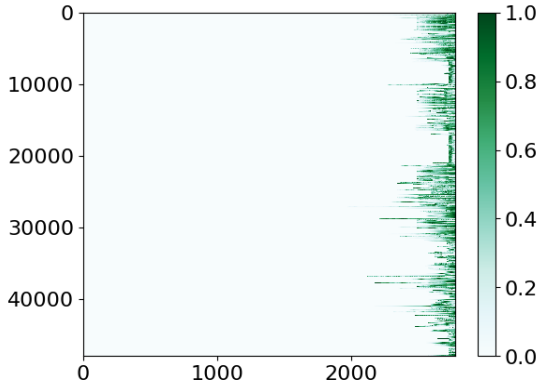
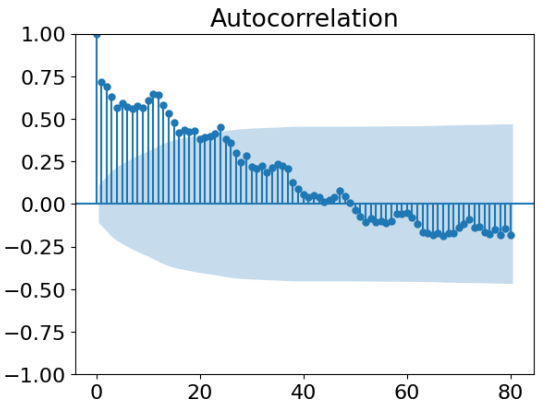
AN2DL Challenge 2

Group: Learning in the Deep *Components:* Giulia Mezzadri, Federico Angelo Mor

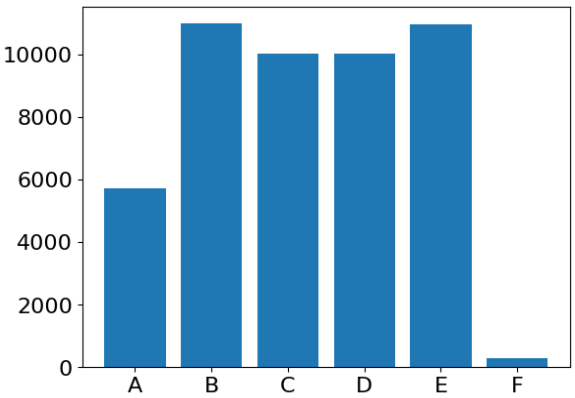
1. Explorative data analysis



We started inspecting the data and we immediately saw a first issue: the different time series had very different lengths (from less than 100 to more than 2500 values), so the padding with zeros was very significant and needed to be accounted in some way, otherwise our networks during training could get confused by that amount of non-informative values.
To solve this issue we thought about different solutions.

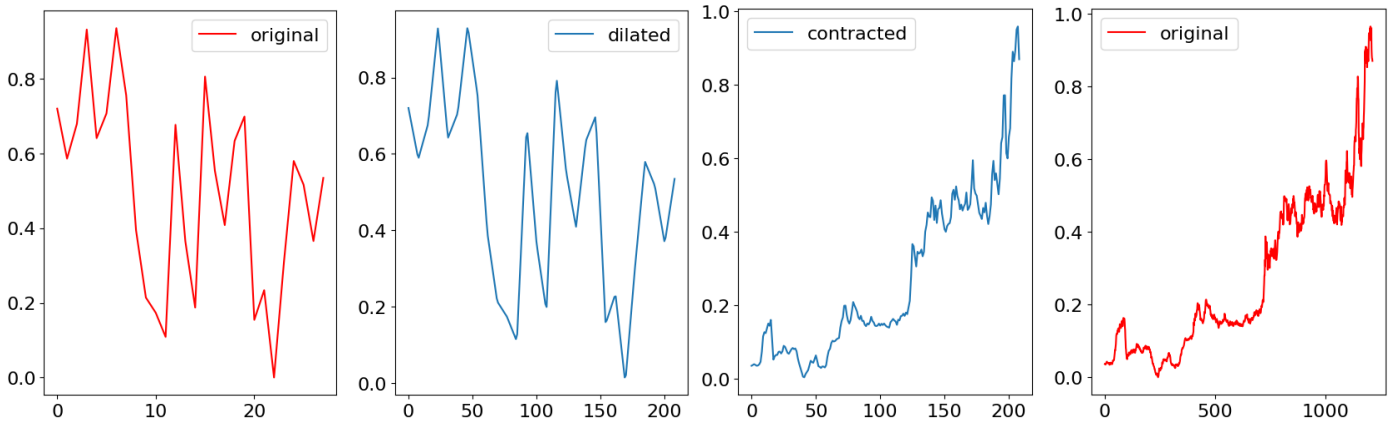


The first idea was cutting up to a certain dimension: the long sequences were just a few and the shape on the codalab test was fixed to 200 time instants. Moreover, for the time series in our dataset we didn't see a significant long-term relationship between values too far away from each other, meaning that just the last values, obtained with the crop, could carry enough information. We were convinced of this by seeing the autocorrelation plots, which for almost every time series showed the interval of meaningful correlations just limited to 24 or 30 time lags. But we considered this option too drastic. Also, a cut bigger than 200 would have just moved the padding problem to the test set input.



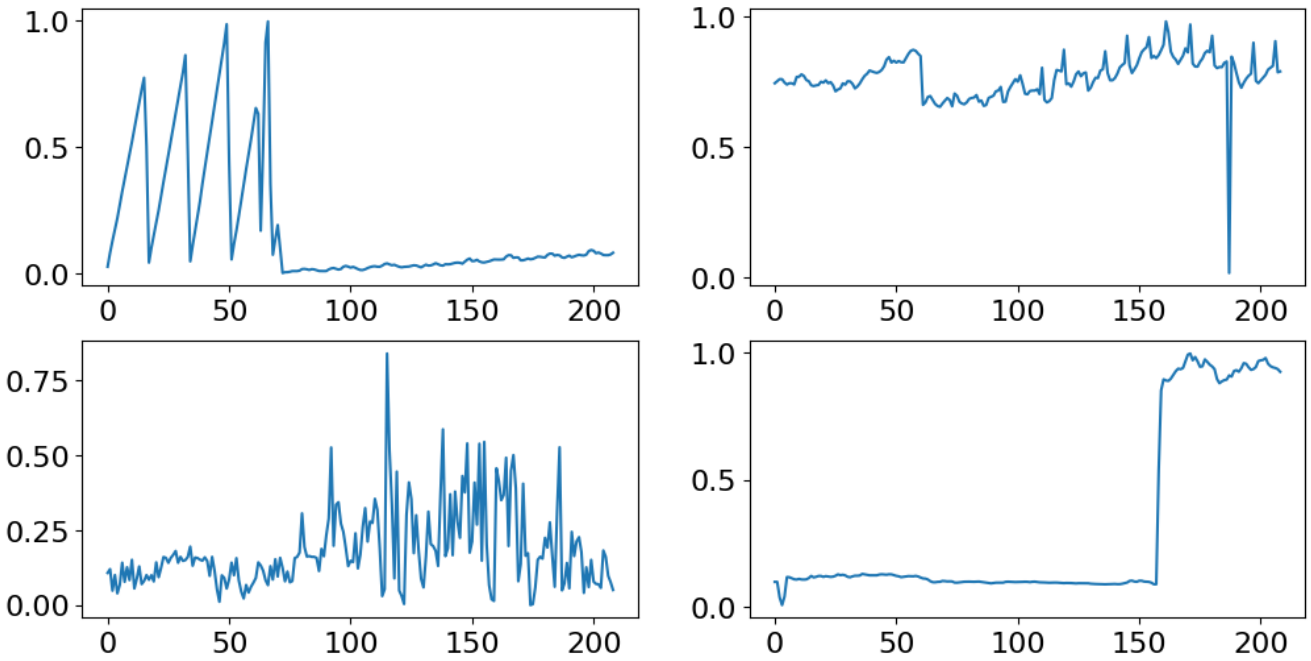
Another solution was to extract multiple time series from the longer ones. More precisely, the idea was to decompose them into multiple pieces using a certain window length (equal to 200+P with P=9 or 18 for the challenge phases) and stride computed depending on the original time series length. We came up with the function stride*(len) = (len - k) / (ceil(len/k) - 1)* where k=200+P, so that each time series would be decomposed into *ceil(len/k)* pieces. For example, during phase 1, this approach (together with the dilation of shorter sequences, see next paragraph) led us from 48000 to 53053 time series. This approach was also interesting as we could also let the stride generation function be label dependent, in order to rebalance the low frequent classes, meaning the time series labelled with A and F (setting e.g. for them a lower stride in order to produce more sequences of their label as output).

The last idea was to dilate or contract the sequences. For the dilating function we could linearly interpolate the values to the desired length k. The contracting function was instead trickier as to contract we needed somehow to skip some values, but this removal could then lead to a less precise



sequence in the end (especially in some quickly varying time series). So for that function we decided to firstly dilate the sequence into a length equal to the least common multiple of k and the original length of the input sequence, and then collect the final values, for the contracted time series, by going through this new sequence with a stride equal to k. We saw this method as maybe computationally not the best (processing all the 48000 observations required 25 seconds) but way more accurate.

For defining the cleaned training set of our models we decided to try both the last two solutions. Before actually starting to train our models we also inspected data in search of outliers. We did that by studying some statistical characteristics of the sequences, such as mean, median and variance (in the global and rolling cases); but also by looking at finite derivatives. These ones were especially useful to remove data which showed quick spikes, which indeed were not predictable unless part of a more general trend. With this filtering we removed about 6% of observations; and these are some examples of the found "outliers".



In the end we had two methods for defining the cleaned dataset. Actually we also thought about a more general approach which could have let our models be learnt and be tested on a general and varying length of time sequences (as we had with the original dataset). To be able to get such a flexibility the idea was to let a priori unfixed the length shape of the time series input of the models (setting e.g. a shape 48000, None, 1) and during the fitting provide "by hand", at each epoch, the time series batch to train on, selecting all the ones of a certain random-selected length (as inside a batch of course the time series needed to have the same length). This seemed an interesting idea and we tried to implement it using the Generator mechanism of python, but unfortunately being the topic a bit deep and technical from the syntax point of view we didn't manage to get it to be fully working.

2. Model definitions

We started defining two simple deterministic models just to establish a baseline on which compare the more sophisticated ones. We called these two models LaVARe and AVARe, since their naive logic was to take respectively the Last Value (LaV) and the Average Value (AV), and Always Return (ARe) them.

Then we moved quickly to more serious models, firstly not keeping categories into considerations in order to understand the problem better and decide which solution for the dataset cleaning problem was the most suited one. We defined a first model (named CONV_LSTM) as a sequence of four Bidirectional LSTM layers followed by four Convolutional layers, with in the end a Flattening and a final Dense layer, with P neurons, to provide the values for the forecasting.
A second attempt was to include some Attention layers, which we did in the LSTM_attention model. We tried to implement it with both "customized" methods and with the pre-existing keras function for multihead attention. Unfortunately it didn't bring any substantial improvement to the original models. More precisely, for this architecture we stacked between the Bidirectional LSTM layers of the previous net some layers of Attention, which

were then concatenated in the subsequent layers. This sequence was then followed again by some Convolutional layers, with in the end a Flattening and now more Dense layers than the just final P-shaped one, in order to make the model more complex and hopefully able to extract more information.
The last model, Transformer, was an attempt to exploit the power of transformer structures to extract long term relations in sequences of values. We therefore built a net with a transformer (a MultiHeadAttention and the related layers) followed by a Flattening and a Dense layer. But also this idea did not perform too well, maybe our architecture was too simple or not precise/tuned enough; but we decided to still keep it as a comparison to the other models.

In every model we also included some Dropout layers, and even the option of recurrent dropout for LSTM layers, and during the fitting we made use of early stopping and learning rate reduction callbacks.
So in the end, for phase 1, after experimenting on the two ideas to define the training dataset, we got these results:

Model Name	MSE	MAE
LSTM_attention	0.007713378872722387	0.05870313197374344
CONV_LSTM	0.008602024056017399	0.0611216276884079
Transformer	0.008778381161391735	0.0627778023481369
LaVARE	0.01435360214071321	0.07608981835840535
AVARE	0.11416309649871083	0.2908133047350334

Then we thought about improving our models accounting for the provided labels. To include the categories in the model a naive way could have been to separate the model into six submodels, each of which trained on the data of a certain category; or a smaller number of more general submodels which could group different categories together. Since from the graphs we couldn't see a similar pattern helping to group categories, we tried the first way and decided to create more specific models.
After finding a promising general model, we tried to apply this idea, but to our surprise it performed worse than the same model trained on the whole dataset. Maybe this was due to the fact that data in categories are not closely related to each other, so a more general and robust model could make the difference. Or due to some labels having a little amount of data the training could have resorted to overfitting.

We also tried a less naive approach in which the information of the label could be exploited directly in the net. To do this the idea was to have a second input: the first for the time series values and the second for the label information. Then we would have concatenated the one-hot-encoded vector to the subsequent layers, directly in the first ones or for example after the Flattening layer with the processing of the time series, to propagate to the Dense layers that further information. Maybe the position in which we let the concatenation happen was more important and delicate, and so should have been better tuned, but the performance on initial tests didn’t show a clear improvement compared to the other models, and so we also decided to not focus too much on this double-input strategy.

Even if these tests of specializing the most promising models to each category led to a bigger error, the general model performed with around a ten times smaller error with respect to the starting global models, so we decided to abandon the idea of splitting, and we focused on the more general approach.

Going towards phase 2 analysis, we selected the top two models of phase 1, LSTM_attention and the simpler CONV_LSTM, which also in this phase remained the best, and really close in performance.

Since for the second part of the challenge the prediction was requested for the next 18 values instead of 9, we tried a simple autoregressive implementation both to avoid training the model again from scratch with an output size of 18, and both because the autoregressive method looked more accurate than a direct full prediction.
We implemented this in the model.py by predicting the first nine values and then shifting the test data X, feeding it along with the new predicted values, to get the final nine and so a total of 18 time instants predicted. Anyway, we also tried to re-train the model with an output shape of 18, to have the whole sequence predicted in the same step, but it performed slightly worse.
Also, we tried another attempt with that autoregression technique in the model.py submitting the model divided on categories but it performed a bit worse, just as we expected from the bad performance also during the phase 1 challenge results.