Artificial Neural Networks and Deep Learning Homework 2: Time Series Forecasting

Ricci Alessandro - 10931743 | Scroccarello Francesco - 10723028
Taccucci Jacopo - 10827846 | Viganò Martina - 10671356

1 Introduction

The challenge addressed the problem of forecasting 18 samples of 6 independent univariate time series using a single model capable of generalizing over all of them. To address the problem, we implemented different models, we reasoned about advantages and disadvantages of each single one of them and we had a huge session of trial and errors to match our theoretical speculation. Once established the best model, we tried to squeeze as much performances as we could with a session of hyper-parameter search, addressed both empirically and using an algorithmic tuner.
We monitored the performance of our models using both the Mean Absolute Error and the Mean Squared Error indices, to check respectively the percentage of prediction errors and their amplitude. Our best result achieved on the given test set, consisted in a model with `MSE` of 0.0095 and a `MAE` of 0.066, which is a rather solid performance.

2 Data preparation

The dataset at our disposal was a collection of 48000 labelled time series, ordered by category and zero-padded for a total length of 2776 observations per-series. After an initial study of the dataset, we discovered an imbalance in the distribution of the categories: while class A and F were composed respectively of 5728 and 277 samples, for the remaining categories (i.e. `B, C, D, E`) we could exploit more then 10000 examples. Rebalance the dataset was not so trivial as the discrepancy was very significant, especially for class `F`, so we made use of both `SMOTE` [1] in order to increase the number of elements in the underrepresented classes and of `RandomUnderSampler` with the aim to reduce the samples for the others, as shown in Figure 2.
It should be considered that the adoption of these techniques is only a mitigation of the problem, but it was still preferred to a more effective balancing, which could have aligned the cardinality of the samples, because this last solution would have implied adding too much bias as it would have introduced a lot of correlated samples.
It is also worth noting that zero padding did not adversely affect the performance of sampling techniques. Before sampling, mean and variance were calculated on the length of the validity periods for each class, these statistics, recalculated after the sampling operations deviated very little from the initial values. The class that showed the highest variance was, as expected, class F.

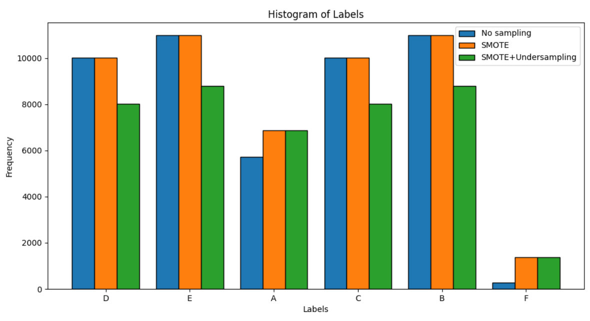

Figure 1: balance level in the dataset before and after the rebalancing.

3 The model

We explored a wide range of possibilities exploiting lots of Keras [2] layers and most of our models were trained considering a fixed sample size of 218, with the last 18 reserved for prediction error. We also made a feeble attempt to use sliding windows, but this approach was soon discarded due to its computational complexity and poor performance on our local test set. We were aware that proceeding by considering only the last 218 values of the input would have led to cut some series reducing the amount of training data available, but their number was negligible compared to the gain in training performances.
Furthermore, we found that the duration of some series was shorter than the selected number of samples, so we applied a masking mechanism to consider only the actual data-points and discard those added by padding.

3.1 Classification and forecasting

To tackle the complexity of the task we decided to split the work of the network into two phases, corresponding to two subnetworks:

- classification phase: sub-network responsible for recognizing the specific class of the signal and predicting the class probability, implemented using bidirectional `LSTM` layers and a dense layer with `softmax` as activation function;
- forecasting phase: the network, given the concatenation of the (masked) initial input and the output of the classifier, had to predict the next samples using some dense layers with `ReLu` activation and a final dense layers with `linear` activation.

We tried to push the performances of the network at its limits by playing with the hyper-parameters such as the number of units or the number of layers, but we did not achieve the expected results. This can be explained by the fact that the forecaster was directly linked to the classifier, thus the classification errors of the first subnetwork were propagated also to the forecaster, impacting on its predicting capability. As a result, the model performance was not satisfing and we decided to pursue another solution.

3.2 One model for category

Since the provided test set was a collection of 60 labelled time series, our idea was to exploit the given labels by training 6 different model (i.e. one for each possible category) and invoking the correct one at runtime to predict the future samples. The obtained results were worse than before and we were very disappointed about that because, theoretically speaking, it should have performed better due to the focus of the models on just a single category. This poor performance was probably due to the fact that either models were too simple and frail to the noise in the dataset or that they were trained on a small portions of the whole training set to feed them with different data. So we discarded also this idea.

3.3 Models ensemble

After one week of hard working we started being suspicious about the capability of our model to generalize over all the categories, so aiming at a possible reduction in overfitting we implemented an ensemble of a Classification and Forecasting network (3.1) and a beta version of our final model, which details will be addressed in subsection 3.4; the enseble returned as output the average of the single predictions. As expected, the overall variance of the regressor decreased and we managed to slightly improve our performances, however in this process we reached a plateau where none of our ideas could further improve the `MSE` and `MAE` metrics, so we decided to discard this idea. Nevertheless, this trial enlighten the urge to adopt some regularization techniques to increase the generalization ability of the network.

3.4 Convolutional LSTM

We tried a slightly different approach: instead of performing a direct classification on the series, we tried to make the network learn how to extract the meaningful features through convolutions against filters to identify the recurring patterns that could then be used to recognize the right series and forecast them. So the network performed a convolution 1D against 120 filters with a kernel size of 6 and padding option same, at the end of which we added a maxpooling layer for dimensionality reduction with a pool size of 3. Then the features were passed to a bidirectional LSTM layer with 128 units and then, through a flattening layer, connected to the output for a prediction with a linear activation. This model was extremely fast to train and obtained also rather good results. Despite of this, the first beta model was not enough to satisfy our team, thus we tried to improve its performances in several ways: we added more convolutional layers in series to learn hierarchically features, we tried parallel convolutions with different kernel sizes to capture different bandwidths, or we even tried to implement some sorts of residual learning with a cascade of convolutional layers and skip links to make sure to learn the complex mapping function as a whole; none of these ideas brought better results tho. The impact of all of them had a common cause: they were indeed all complications of the model, thus increasing its variance. So, instead of keeping increasing the complexity of the model we started increasing the regularization we performed during the training, exploring lots of different combinations and also automatic tuning tools. After a long session of experiments, we ended up with our best model whose code can be found in the notebook. The key point is the high dropout rate before the flattening, and the idea came from the fact that the dataset was ordered by category. A shuffle on the data resulted in a not so good learning effect, because essentially we wanted the network to learn consequent examples of the same series to enforce the memory effect. This on the other hand was biasing too much the capability of the network to generalize due to an high focus on the first examples received. So a possible regularization of this poisonous effect was to randomly disable half of the results obtained by feature extraction and memory parsing to allow the network to learn at each step a subset of the model istelf, ending up with an implicit combination of submodels general enough to address the task we were facing.

4 Conclusion

This report addresses the problem of forecasting independent and univariate time series. Throughout the duration of this project, we developed various techniques, whose reasoning and implementation were grounded on different theoretical concepts in the time-series analytic world. The achieved results suggest that, considering the available data and with limited knowledge of the dataset sources, the adoption of simpler and plain models is preferred with respect to more complex ones. We also experienced that not always what we consider a key winning idea leads to perfect results and that sometimes it is acceptable to make a step back and start exploring other possibilities. In fact, although not all the implemented models kept up our expectations, we reached acceptable and rather satisfying results in the end thanks to the approaches described in the previous sections as well as the combined teamwork which proceeded sometimes in parallel on separate paths to maximize the efforts, and then rejoining the learnt aspects to improve our work.

5 Contributions

Team Member	Reasoning	Programming	Testing	Total Time
Ricci Alessandro	25h	25h	25h	75h
Scroccarello Francesco	25h	25h	25h	75h
Taccucci Jacopo	25h	25h	25h	75h
Viganò Martina	25h	25h	25h	75h

Table 1: Contributions

References

[1] Synthetic Minority Oversampling Technique, reference guide

[2] Keras API doc