Homework 2

Bistacchia, Erica*
Gurrieri, Davide†
Negroni, Sabrina‡

2023/2024

POLITECNICO
MILANO 1863

CONTENTS

1 INTRODUCTION

The goal of this project was to design and implement a time series forecasting model able to learn how to exploit past observations in the input sequences to correctly predict future samples. We wanted to develop a model which exhibits sufficient generalisation capabilities in the forecasting domain, transcending the constraints of specific time domains.

We were provided with a dataset consisting of univariate uncorrelated time series of variable length. To simplify the portability of the dataset the sequences are pre-padded with zeros (to the maximum length of 2776) and saved as a Numpy array in a compact form of 48000×2776. In addition, we had further information: the valid periods, crucial for recovering the original time series without padding, and labels indicating the time series macro-category between six classes A, B, C, D, E and F.

2 DATA PRE-PROCESSING

Given the nature of the problem, we deemed data pre-processing as particularly crucial since a higher model complexity did not prove to directly reflect into an improvement in performance. In the following sections we will explain the most important aspects we have focused on, going beyond the mere selection of the model architecture.

*erica.bistacchia@mail.polimi.it
†davide.gurrieri@mail.polimi.it
‡sabrina.negroni@mail.polimi.it

2.1 Data description: problems and solution

Analyzing the distribution of the time series across the six categories, we observed a noticeable imbalance, particularly with category F being significantly underrepresented. This did not generate any problems since we discarded labels in the developed models, as we will explain in detail later. It should however be noted that it would be important in a classification problem.

The dataset was already normalised using the min_max normalization method, resulting in time series values within the [0,1] range. As a result, further normalization seemed to be unnecessary. Despite this, we experimented with the robust scaler which lead the considered models to perform slightly worse. This was contrary to our expectations as the robust scaler should theoretically help the network to become less sensitive to outliers during training.

Additionally, we chose to exclude short time series from the dataset as they lack sufficient information about patterns or behaviors for the network to learn effectively. After a deeper examination of models performance, we decided to discard time series with less than 36 timestamps. This decision was influenced by two key considerations: firstly, we assumed that in order to accurately predict 18 timestamps it was essential to have at least an equivalent number of samples; secondly, we aimed to avoid the risk of discarding crucial information for achieving generalisation.

2.2 Sequence building

We decided to use the rolling window technique as, by focusing on shorter segments, models can discern and learn from short-term trends, seasonality and patterns potentially not apparent when considering the entire time series. In addition, the use of smaller windows facilitates a more responsive adaptation to dynamic patterns over time.

We defined a custom function named `build_sequences`. This function iterates over all input time series and extracts only the valid interval without zero padding. It then calculates the actual length and skips to the next time series if it is less than `min_length`. Padding is then added at the beginning to reach the desired length if the time series is shorter than `window + telescope`. Otherwise, if the time series is sufficiently long, the function slides the window over the time series from the bottom moving with a specified stride. In the last iteration, if the window exceeds the length of the time series the function can proceed in two ways depending on the `repeat_first` parameter. If it is set to `False`, padding of appropriate length is added. On the contrary, if it set to `True` the stride is modified so that the beginning of the last window coincides with the beginning of the time series. This way some timestamps are repeated more than expected,nevertheless preventing the addition of padding or loss of initial information.

When dealing with a time series forecasting problem, there are some relevant parameters to set, in particular the window size and the stride step. These choices are not straightforward and are problem-related.

This challenge had two phases: in the first one we had a `telescope=9`, in the second `telescope=18`. We decided to experiment with different values in order to find the most appropriate

ones. It is important to take into consideration what these variables are related to: the `window` determines how many timestamps are important to correctly predict future samples, while the `stride` is linked to the dataset size (since if stride << window it will lead to repeating samples in different sequences). For this reason, we tried different combinations of values in order to find the most suitable ones. Initially, `window=100` and `stride=10` result to be the most appropriate since we needed to predict 9 future samples. In the second phase instead, we switched to bigger values as the number of future samples to predict increased, in particular we set `window=200` doubling its previous value and `stride=70`. Different tested out values resulted in a worse performance.

2.3 Padding

In order to build the sequences, padding is necessary as the model can only take data of constant length in input. To do so, we used the function `numpy.pad`, which offers a high level of flexibility. Several types of padding can be selected, including the more common ones with a constant, mean or median value, as well as more sophisticated ones such as padding with the reflection of the vector mirrored on the first and last values of the vector along each axis.

Among all the padding techniques explored, the more advanced ones, namely `varp` (variable padding) and `reflect` (time series reflection), were tested. Despite testing all options, we did not observe any noticeable improvements. In addition, we experimented with a masking approach using the Masking layer implemented with `keras.layers.Masking`, which filtered out padding for further processing. However, also this strategy did not produce substantial improvements leading us to retain zero-padding because of its simplicity.

2.4 Jittering

We designed a layer that takes a batch of time series as input and applies jittering to them. We chose to define it in such a way that no jitter is applied to the last 7 time instants so as to not distort excessively the input and to have a prediction of the future as accurate as possible[1]. The layer activates only with probability `prob` at each epoch and it randomly adds a noise with sigma standard deviation which is sampled in an interval specified in the jittering function. The most remarkable aspect is that using this layer (i.e. applying jittering in the training phase) leads to better results than applying jittering directly on the dataset (as an augmentation pre-processing technique). The disadvantage is that, if we want to add noise only in the valid periods of the time series, this implementation only works with constant padding. (Figure 1)

The idea behind this is that a proper jittering does not change significantly the time series, however it allows to train a more robust model with respect to noisy samples, improving its predictive performance. The choice of the noise's hyperparameters is a difficult question to address and mostly depends on the dataset characteristics. It is often worth experimenting and analyzing how these affect the performance of the target model.

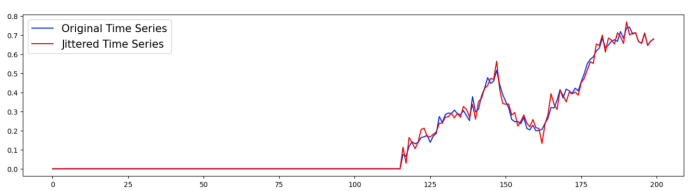

Figure 1. Example of Jittering applied to a short timeseries window.

3 TESTED MODELS

3.1 Nine Timestamp forecasting

The initial architecture we implemented was the one provided in the Lab, consisting of a Bidirectional LSTM, two Conv1D layers, a Flattening layer, and lastly a Cropping layer to achieve the desired output size of (9,1). However, this model faced limitations in addressing our forecasting problem. Consequently, we tried to refine our approach by tuning hyperparameters, adding layers and exploring alternative neural network architectures to improve prediction capabilities.

Our primary focus was to increase the ability of the model to identify potential patterns, periodicity or seasonality within the time series. To achieve this, we incorporated an Attention layer implemented with `keras.layers.Attention` sequential to the bidirectional LSTM. We experimented with both `dot` and `concat` modes for computing the attention scores with the second one achieving better results.

We then observed that the Cropping layer, originally placed at the end of the network, was probably removing significant information. To address this issue, we opted to replace it with a Flatten layer followed by a Dense layer. This demonstrated substantial improvements in the model ability to capture relevant features.

The most notable advancement was given by the integration of a custom `DataAugmentation` layer at the beginning of the neural network. This introduced a jittering into the time series during training with a probability initially set at 30%.

This final architecture, trained with a window size of 100, a stride step of 10 and a telescope of 9 timestamps, proved to be the most successful during the initial phase of the challenge, achieving a Mean Squared Error (MSE) of 0.0050.

We also tried to take into consideration the labels by training 6 different models, however with no significant improvements. We therefore decided to remove the label information, also because after a deep analysis we did not notice any particular characteristics which allows to distinguish one class from the others.

Finally, we experimented also with a Transformer architecture that included a 4-block encoder. Within each block, a combination of multi-headed self-attention and a feed-forward neural network was employed. To improve model stability, the input sequence was subjected to layer normalization and attention mechanisms, preserving crucial information through residual connections.

During training, data augmentation introduces random noise, contributing to model adaptability. The final layers incorporate Global Average Pooling to reduce dimensionality and multilayer perceptron for further feature refinement.

3.2 Eighteen Timestamp forecasting

In the second phase, our preliminary approach was to re-evaluate the best-performing model described above, expecting about twice the mean square error (MSE). However, the performance of the model was slightly lower than expected, registering an MSE of 0.0124. Consequently, our attention shifted to building alternative models that could show better generalization and more accurate predictions for the 18 timestamps.

We initially introduced in our model additional layers of bidirectional LSTM as well as some convolutional ones. To fine-tune the parameters of these layers, we employed Keras-Tuner.

As this did not exhibit satisfactory results, we then tried different architectures also based on literature. The first model we tried was Xgboost, which according to literature seemed to be particularly well suited for time series forecasting problems [2]. The model performed very well in validation but unfortunately could not be used as it was not supported by the Codalab platform.

As a result, we switched to other architectures starting with a very simple structure consisting of only two Dense layers with 512 and 256

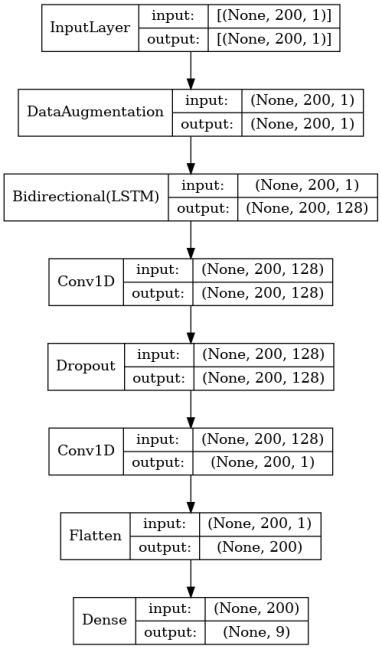

Figure 2. Final model architecture.

units respectively in addition to the output layer. To this basic structure we then added a Bidirectional LSTM at the beginning of the network, followed by a Conv1D layer both with 64 units in order to best extract temporal dependencies as well as local and global spatial patterns.

We included also the Data Augmentation layer to incorporate some noise in training with a 40% probability. As the model overfitted, we then added 3 Dropout layers before each Dense layer with rate of 0.2 , 0.3 and 0.4 respectively.

The constructed model was trained with a window of 200 and a stride of 10. Despite the excellent performance in validation with an MSE around 0.0065, our expectations were not satisfied during testing where the MSE dropped to 0.0114. This inconsistency could be attributed to the test set, which may include significantly different time series from those used for training. Hence, the above mentioned results speak for the fact that our model may struggle to generalize effectively.

4 FINAL MODEL

Our best model turned out to be a slight adjustment of the initial architecture. First, we dropped the Attention layer since, during the experimental stages, we observed that it did not significantly contribute to improving the performance of the model in our specific context.

In addition, we modified the second Convolutional layer by reducing the number of units to 1. This allowed for capturing specific patterns or local dependencies and recognizing the most relevant information in the input sequence.

The final architecture can be visualized in Figure 2. In particular, the `DataAugmentation` layer was embedded with a probability of 40% while the `Dropout` layer with a frequency parameter set to 0.1. In all the layers we used a `ReLU` activation function, except for the `Bidirectionl LSTM` where we used a `Tanh`.

Evaluation

In the prediction step, we used an autoregressive forcasting technique that involved making the first 9 predictions that are then concatenated to the existing time series from which we removed the first 9 timestamps. This new sequence became the input for predicting the next 9 values (Figure 3). This methodology allowed the model to use its own predictions as input to predict the next steps, thereby creating a sort of feedback loop. This was effective in the context of time series where temporal relationships and dynamics could be captured more effectively through the use of autoregressive predictions.

In order to validate our model we generated a validation dataset sampling a fixed number of time series (in our case set to 180) from the dataset, ensuring their length to be at least 218 as in the hidden test set. To be coherent with the CodaLab test we also sampled the same number of time series from each of the six classes.

This model achieved the best performance in test with an MSE of 0.00874133.

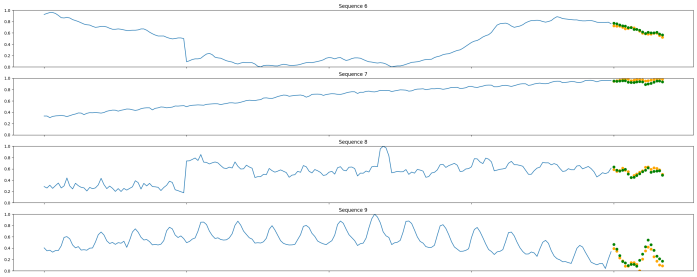

Figure 3. Prediction (in green) of the future 18 samples of 4 time series with respect to the real values (in yellow).

Figure 4. MSE (model evaluation metric) values during training epochs.

5 CONTRIBUTIONS

All the three group members Erica, Davide and Sabrina contributed to the work equally, doing great team work. All the work done for this challenge can be found on the following github repository: timeseries-forecasting

REFERENCES

[1] Artemios-Anargyros Semenoglou, Evangelos Spiliotis, and Vassilios Assimakopoulos. Data augmentation for univariate time series forecasting with neural networks. *Pattern Recognition*, 134:109132, 2023.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.