Time Series Forecasting

Task

This project aims to build a neural network model for time series forecasting. Given a time series belonging to one of six different categories, our model will predict its future values. The goal is to have a general model able to accurately predict time series of different nature. First, the model will be asked to make predictions about the first 9 future samples. In the second phase, the model should be able to predict 18 future samples.

Data Pre-processing

The first step of our work consisted in the analysis of the provided dataset, consisting of 48,000 time series, with a maximum length of 2,776 samples. Shorter sequences were padded using "0" values in the initial part of the series and the original start and end timestamps were separately provided.
Upon a more detailed examination, we identified the presence of duplicated time series which were consequently removed, ensuring a more accurate and reliable dataset for training. Then, to reduce computational time, all values of the time series were converted from data type "float64" to "float32". Since all values do not have many decimal digits, this approach does not change the characteristics of our dataset.

Training and test datasets

To train our model, the dataset has been split into a training and a test dataset. We constructed both datasets by pairing sequences designated for model input with corresponding values to predict. The length of each dataset is determined by the "window" and "telescope" parameters, respectively.

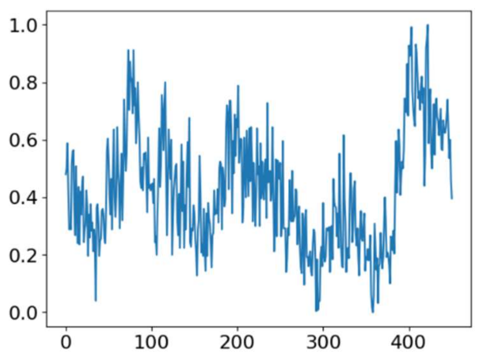
Sequences that were shorter than a window have been inserted in the test set to avoid training the model on less relevant sequences with longer padding. To avoid losing too much data in the training phase, the length of the test set is limited to at most 10% of the original dataset. Although this might affect the reliability of the test set, making it less significant, it allows training the model on higher-quality data. When testing models specialized for each category, the train and test sets have been split by category.

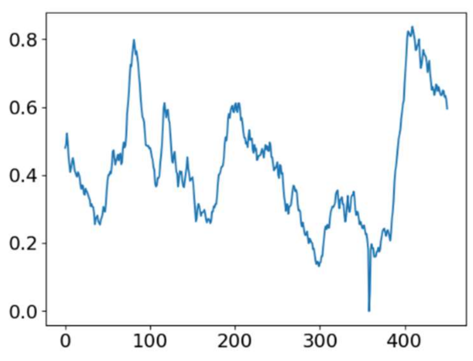
It is also important to note that all models included a masking layer to avoid considering padding during training. This layer creates a mask, based on a given value used as padding, that will make all subsequent layers skip the masked timestamp. Initially, we set all values outside the valid period of the sequence to 1.1, a value outside the normal time series range, and used this value for masking. We then noticed the presence of sequences having a large sequence of zeros inside the valid period. To remove these anomalies, masking was performed on all 0 values (including padding).

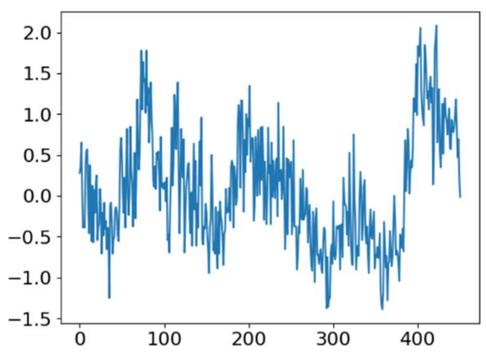
Failed experiments

In our research, we explored various strategies for addressing missing data, outliers, and noise within the time series.

Among those, we tried to apply the moving average [1], a statistical method used to smooth time series data by calculating the average of data points within a moving subset of the series, highlighting underlying trends, smoothing out noise, and attenuating the impact of outliers.
Additionally, we considered various normalization techniques. We decided not to perform MinMax normalization, recognizing its redundancy due to the dataset being already normalized within the 0 to 1 range.
The Robust Scaler is a data normalization technique, consisting in dividing the difference between the value and the median by the interquartile range. The Robust Scaler represents a form of normalization that allows more robust results when dealing with outliers. The predictions of the model need to be scaled back.


Original time series


Time series after Moving Average


Time series after Robust Scaler

Contrary to expectations, both techniques, even though promising, resulted in a decrease rather than an enhancement of our performances.

To deal with sequences shorter than the sum of *window* and *telescope*, different padding techniques have been tested. Among them, we tried to replicate shorter sequences a sufficient number of times to fill the entire length of a window. Since this technique did not show significant improvements, we eventually decided to use a standard 0-padding technique.

To improve the performance of our model, we also experimented with various time series augmentation techniques. Specifically, we attempted to introduce noise, in order to increase the variety of the dataset. However, these attempts did not produce improvements in the generalization capabilities of our model.

To isolate only the useful section of the time series for future timestamp forecasting and resize the window accordingly, we experimented with autocorrelation [2]. We first transformed the time series into stationary form through multiple applications of differencing. Then, we analysed the autocorrelation plot and kept only the segment where the values were not in the neighbourhood of zero, as these indicate the time series portion that is relevant for predicting future timestamps. We decided to abandon this approach given the large computational power required and the negative impact on performance.

Model Definition

To choose the window size, we exploited cross-validation by running the same model and selecting the one that best captured the relationships between the points of the time series. Moreover, we established that its dimension should align with the input size from the Codalab tests to avoid the usage of padding and the final choice fell on a window 200 timesteps long.

We chose a stride that is a divider of the window size, initially experimenting with higher stride values before reducing them, leading to performance enhancement. A stride of 5 proved to be an optimal trade-off between performance and training speed, as lower stride values generate more training samples for the model.

For our loss metric, we opted for Mean Absolute Error (MAE) over Mean Squared Error (MSE). This choice was driven by the observation that error values were mostly between 0 and 1, rendering MSE less effective due to its lower values compared to MAE, and thus less precise in model evaluation. Moreover, to mitigate overfitting and optimize parameter values, we employed strategies such as early stopping and learning rate scheduling.

Model description

Initially, our approach involved developing a single model capable of predicting for all categories indiscriminately. We then attempted to customize individual models for each category to capture their unique characteristics, but this approach surprisingly underperformed compared to the unified model. This observation led us to the conclusion that the differences between the categories are probably minimal.

To further improve our forecasts, we introduced skip connections in our models to mitigate the vanishing gradient problem. This choice was also made to improve the model's ability to capture long dependencies and complex temporal patterns, thus helping to improve overall performance and accuracy.

Finally, we explored two distinct approaches to building the output layer. The first method involved the use of a Cropping layer. In an alternative and more effective approach, we used a combination of a Flatten layer and a Dense layer. This approach was specifically designed to capture spatial dependencies within the sequence, improving the model's ability to capture meaningful patterns in the time series.

ResNet-Like

The architecture employed in this model takes inspiration from the ResNet model commonly used in image analysis. By combining the capabilities of convolutional layers (Conv1D) with skip connections, this architecture captures intricate spatial and temporal patterns within time series data. [3] [4]

BiLSTM

One of the architectures used exploits Bidirectional Long Short-Term Memory to capture both forward and backward temporal dependencies. The model is also designed to take advantage of skip connections to facilitate learning complex patterns within time series data.

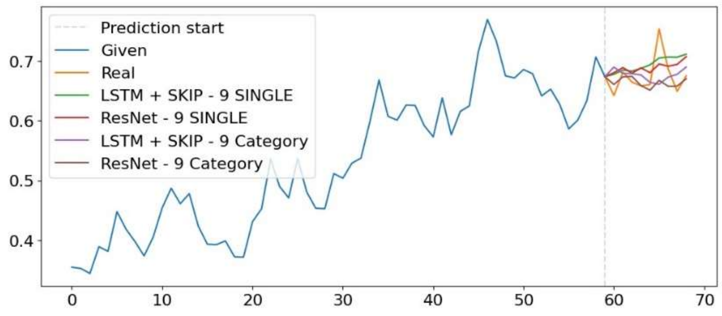
Ensemble

Finally, the choice fell on an ensemble model to consider both Neural Network architectures, LSTM and ResNet-Like with residual connections. We created one model per category and one that took all categories into account. We repeated the procedure for both architectures. At inference, a prediction is made with the two models that consider all the categories and with the two that are instead specialized for the category. The result of the prediction is then averaged to obtain the forecasts.

Autoregression

In the initial phase of our study, we focused on training the model to predict 9 timesteps at once. We explored autoregressive techniques using an autoregressive telescope equal to three.

Moving on to the second phase, where the task involved predicting both the preceding 9 samples and an additional 9 samples into the future, we experimented with two approaches. Initially, we attempted to forecast all 18 samples at once. However, we discovered that better performance was achieved by utilizing an autoregressive model, forecasting 9 samples first and then predicting the subsequent 9 samples.



Model	MSE	MAE
LSTM + SKIP - 9 SINGLE	0.01721	0.09475
ResNet - 9 SINGLE	0.01655	0.09421
LSTM + SKIP - 9 Category	0.01873	0.09932
ResNet - 9 Category	0.01599	0.09227

Comparison between different predictions of different models and the metrics value

Failed Experiments

Single Model that takes categories into account

Since we previously showed a negligible difference among categories, we decided to train a single model incorporating the category as an input feature. This approach allows the model to leverage shared patterns across categories while also discerning deeper category-specific patterns. Contrary to expectations, even though promising, this model did not yield an improvement in our generalization capabilities.

Other tested architectures

In our attempt to develop a comprehensive and versatile model, we systematically explored various network architectures. We started our experimentation with *Fully Connected* networks with poor results [5], despite the literature on the matter. Trying to capture the spatial characteristics, we tried a network composed of convolutional layers (*Conv1D*), obtaining good results. Finally, we tested Recurrent Neural Networks (*RNN*), trying *LSTM* and *GRU* architectures to capture the temporal relationships between the points of the time series. These networks were then replaced by the more efficient *Bidirectional LSTM*. [6]

Transformer

Transformer models have also been tested to solve our problem. An encoder-decoder model, based on the one presented in the “Attention is all you need” paper [7], has been tested with different values for the parameters of the model (e.g., number and size of attention heads, number of attention blocks…). The implementation of this model has been adapted from the Keras tutorial [8]. Positional embeddings have also been included to make the model able to consider time dependencies between samples. However, this model did not perform as well as other models that have been tested.

Time2Vec

Based on the paper “Time2Vec: Learning a Vector Representation of Time” [9], we included in the Birectional LSTM and Transformer models a layer to learn time embeddings [10]. In the specific task we were trying to solve, this method did not show any significant improvement in performance.

Contributions

Our team distributed the responsibilities evenly, ensuring each member actively participated in every stage of the project. We collectively contributed to the ideation of the models, experimenting with diverse techniques, and adjusting parameter values. Since Dario installed Tensorflow and Keras directly on his computer, he leveraged his robust GPU and did not have limits regarding usage time as Gabriele and Alessio did on Google Colab, so he focused more on training the models. To balance the workload, Gabriele and Alessio concentrated on the necessary pre-processing of the data. This report has been written with the contribution of all team members.

In total, each of us dedicated approximately 75 hours to the development of the project.

NOTE: for conciseness reasons, some of the statements in the text are not directly supported with data or graphs. All data to support these claims can be found in the attached Python notebook.

Bibliografy

1. ***Autocorrelation in Time Series Data.*** **[Online] https://www.influxdata.com/blog/autocorrelation-in-time-series-data/.**

2. *Understanding Autocorrelation in Time Series Analysis.* [Online] https://towardsdatascience.com/understanding-autocorrelation-in-time-series-analysis-322ad52f2199.

3. *Deep learning for time series classification: a review.* Ismail Fawaz, H., Forestier, G., Weber, J. et al. Vols. https://doi.org/10.1007/s10618-019-00619-1. s10618-019-00619-1.

4. 1-d Convolutional Neural Networks for Time Series: Basic Intuition. *https://boostedml.com/2020/04/1-d-convolutional-neural-networks-for-time-series-basic-intuition.html.* [Online]

5. *The Effectiveness of Feed-forward Neural Networks in Trend-based Trading.* [Online] https://towardsdatascience.com/the-effectiveness-of-feed-forward-neural-networks-in-trend-based-trading-1-4074912cf5cd.

6. *How to Develop Convolutional Neural Network Models for Time Series Forecasting.* [Online] • https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/.

7. *Attention Is All You Need.* Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. arXiv:1706.03762.

8. Keras. [Online] https://keras.io/examples/timeseries/timeseries_classification_transformer/.

9. *Time2Vec: Learning a Vector Representation of Time.* Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, Marcus Brubaker. arXiv:1907.05321.

10. GitHub: Medium Notebook/Time2Vec. *https://github.com/cerlymarco/MEDIUM_NoteBook/blob/master/Time2Vec/Time2Vec.ipynb.* [Online]