Time Series Forecasting - 2^{nd} Homework

Francesco Caserta, Nuno Costa, Shodai Fujimoto, Rio Ishibashi
Artificial Neural Networks and Deep Learning, 2023
Politecnico di Milano, Italy

1 INTRODUCTION

This report highlights our approach for time forecasting in the AN2DL course's 2^{nd} homework assignment. The objective of the assignment is to predict future samples of the input time series. The data provided comprises 48,000 distinct time series, each with a length of 2,776 data points and a corresponding category. There are six categories, and the valid time periods corresponding to the non-padded segments are also provided. The goal of the final phase is to predict the next 18 time steps.

2 DATA ANALYSIS

At first, we delved into the dataset's key characteristics. Categories B, E, C, D, and F each include roughly 10,000 time series, while Category A holds 5,000, and Category F contains just 277.

Following that, we analyzed the distribution of the actual lengths of the time series (without any added data) for each category using boxplots. Our observations revealed that, for most series, the Inter-Quartile Range (IQR) spans from 50 to 250, except for Category A, where the time series display a more consistent length (a narrower IQR) and are slightly longer on average. Furthermore, we identified outlier time series with notably longer durations.

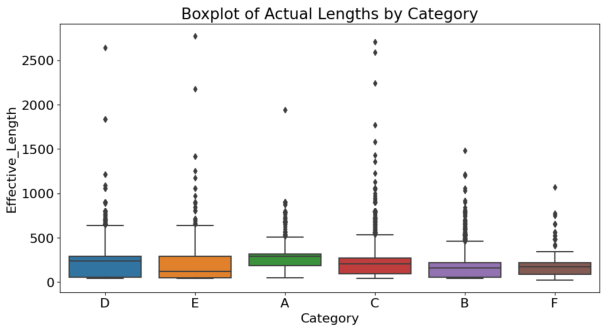

Figure 1: Length distribution by Category

When we grouped the time series by categories and visualized them, we noticed significant differences in trends among some time series within the same class, while others displayed noticeable similarities.

Following this, we conducted a more detailed analysis of specific time series by decomposing them using the 'seasonal_decompose' function from the 'statsmodels' library. This decomposition method separates the time series into trend, seasonality, and residuals.

Upon observing the trend component, it became evident that most time series exhibit non-stationary behavior. Examining the seasonality component, we found that many time series demonstrate periodic patterns. In an attempt to pinpoint the period, we explored autocorrelation using the 'plot_acf' function after detrending the series.

We discovered that the correlation function frequently displays a periodic pattern, often corresponding to a period of 12. This suggests that many categories might be associated with phenomena occurring annually.

3 PREPROCESSING AND DATASET PREPARATION

Our dataset-splitting strategy aimed to maximize information utilization from time series data, even if it involved handling padded sequences. We generated all potential sequences for each time series based on a specific window size and an assumed telescope length (9 for phase 1 and 18 for phase 2). In instances where a time series fell short of meeting the combined window size and telescope length criteria, we formed a single sample using the least padded sequence possible from that series.

Regarding the train, validation, and test splits, we decided to perform these splits on a per-time-series basis. We stratified the splits based on the categories of the available time series. Subsequently, following each split, we constructed sequences as described earlier.

In our final models, we refrained from applying preprocessing techniques. Initially, we tried the Robust

```
Category E:
Original Length: 10975
Length after splitting (train): 10426
Length after splitting (val): 274
Length after splitting (test): 275
```

Figure 2: Splitting example for class E

Scaler, scaling all time series. However, we didn't test it on Codalab because we were uncertain how to appropriately reverse the scaling using the Robust Scaler fitted on our training data.

Moreover, we experimented with data augmentation methods [1], such as Magnitude and Window Warping, Spawner, WDBA, and Guided Warp. Despite this, the performance improvement was minimal, leading us to exclude augmentation from our final results.

Additionally, we explored an alternative approach by training two distinct models—one with a window size of 40 and another with a window size of 100. Both models had an output sequence length of 18. The larger window size model aimed to capture long-term relationships in the time series, while the shorter window size model focused more on short-term fluctuations.

To implement this, we divided the original dataset into two subsets: time series longer than 118 were used for training the model with the larger window size, and those 118 or shorter were employed for the model with the smaller window size. This method implicitly minimized padding, which we suspected might affect our tests—further details are in section 4.3.

4 MODEL ARCHITECTURES

4.1 Bidirectional LSTM + 1D Convolutional layer

Initially, we experimented with a model consisting of a bidirectional LSTM and two 1D Convolutional layers followed by a 1D Cropping layer. We began by adjusting the window size, discovering that smaller window sizes allowed us to forecast finer fluctuations in a time series, while larger window sizes enabled forecasting larger-scale variations.

[1] https://github.com/uchidalab/time_series_augmentation/tree/master

During the development phase, the highest MSE (Mean Squared Error) and MAE (Mean Absolute Error) scores were achieved using a window size 40, with MSE at 0.00535 and MAE at 0.05073.

However, as we aimed to forecast the next 18 time steps in each time series, we considered a window of size 40 to be too small. Consequently, we created models using larger window sizes.

The table below showcases the results from the local test of forecasting 18 time series:

Table 1: Model Performance

Window size	MSE	MAE
40	0.01001	0.06454
80	0.00984	0.06405
100	0.01016	0.06614
140	0.00940	0.06444

4.2 Bidirectional GRU+ 1D Convolutional layer

To address the vanishing gradient issue in RNNs, we compared the performance of a Bidirectional GRU to that of a Bidirectional LSTM. The architecture involved a bidirectional GRU coupled with two 1D convolutional layers, outputting to a 1D cropping layer. The outcomes and patterns closely resembled those obtained using the Bidirectional LSTM layer.

In the initial phase, employing a window size of 40 and a telescope of 9, this model achieved an MSE (Mean Squared Error) score of 0.00545 and an MAE (Mean Absolute Error) of 0.0533.

4.3 Ensemble model

Recognizing that smaller window sizes capture finer fluctuations and larger window sizes forecast larger-scale variations in time series, we aimed to enhance results by combining models using both window sizes. To achieve this, we averaged the outputs from models utilizing each window size—specifically, models with window sizes 40 and 140 were ensembled.

During local testing, this ensemble model yielded an MSE (Mean Squared Error) score of 0.00899 and an MAE (Mean Absolute Error) of 0.06227. However, in the CodaLab test, the MSE score increased to 0.01337, accompanied by an MAE of 0.07918.

4.4 Attention

Recognizing the significance of the Attention Mechanism in learning long-term dependencies within sequential data, we pursued architectures that harnessed this mechanism. Our exploration included both simple models integrating Attention layers and complete Transformer architectures. Notably, the Attention Mechanism enabled us to incorporate samples with significant invalid periods by efficiently masking out those indices from each sample as needed.

For the simpler models utilizing Attention layers, we tried a model with a Bidirectional LSTM encoder followed by an attention layer and two 1D convolutional layers, outputting to a 1D cropping layer. Our experiments yielded lower results than the same model without the attention layer (introduced in 4.1), yielding an MSE of 0.0063 on the first phase of the competition.

Regarding the Transformer approaches, we initially attempted an encoder-only transformer, achieving 0.0085 on the MSE metric locally. Additionally, we experimented with the Informer [4] architecture, which delivered a performance of 0.0074 locally. This architecture also included encoding the categories into the model by leveraging a concatenation layer followed by a dense layer of the target size. However, this local performance was not maintained in Codalab, worsening by almost 150%.

4.5 TSMixer

While searching for diverse architectures, we stumbled upon Google's latest TSMixer architecture [1]. Inspired by the transformer block, this architecture replaces Attention with a Time-Mixing layer. Despite its simplicity, it offers the potential for near state-of-the-art performance.

Indeed, models based on this architecture yielded our best results. However, due to time constraints close to the deadline, we had limited opportunities for fine-tuning. Consequently, we experimented with configurations featuring 8 and 12 TSMixer Blocks and varying Feed-Forward dimensions of 48 and 128. Locally, these models demonstrated performance ranging from 0.0078 to 0.0085 regarding the MSE metric.

5 BEST SUBMISSION

Our top-performing submission was built upon the TSMixer methodology outlined earlier. Our model comprised 8 TSMixer Blocks, featuring a Feed-Forward dimension of 128 in both the Time Mixing and Feature Mixing FFNN components. Additionally, we incorporated category information into each TSMixer block by combining it with the block's input. This combined information was then processed through an FFNN with a window size of 200, which matched the input size.

This model exhibited promising local performance using a basic data split (as previously described), generating an MSE of 0.00801 with a stride of 2 for sample generation. However, on Codalab, the model's performance declined notably to 0.0117. We suspect this discrepancy may be attributed to our data-splitting methodology.

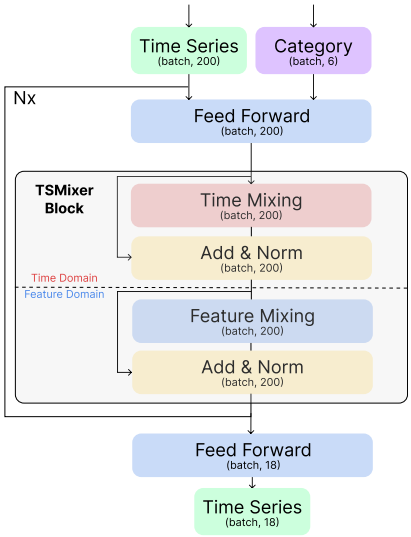

Figure 3: Architecture of Our Best Submission - TSMixer based model with Category Encoding

6 CONTRIBUTIONS

Each team member contributed equally to the project, participating in all aspects. However, their specific contributions were as follows:

- Francesco concentrated on dataset splitting and experimenting with various model structures.
- Nuno specialized in exploring state-of-the-art architectures and delving into data augmentation techniques.

- Shodai dedicated efforts to experimenting with numerous window sizes and the ensemble modeling approach.
- Rio focused on exploring different model structures and dataset-splitting strategies.

REFERENCES

[1] Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O. Arik, and Tomas Pfister. 2023. TSMixer: An All-MLP Architecture for Time Series Forecasting. Transactions on Machine Learning Research (TMLR), 09/2023. (2023). arXiv:arXiv:2303.06053

[2] François Chollet et al. 2015. Keras. https://keras.io.

[3] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[4] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2020. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. arXiv:arXiv:2012.07436