Recurrent Neural Network: Time Series Forecasting

Artificial Neural Networks and Deep Learning Homework 2 – A.Y. 2023/2024

Marton Barta*, Al Kamber† and Federica Vinciguerra‡
Email: *marton.barta@mail.polimi.it, †al.kamber@mail.polimi.it,
‡federica.vinciguerra@mail.polimi.it
Student ID: *10884623, †10637023, ‡10921587,
Codalab Group: "Gradient Gamblers"

1. Introduction

The provided dataset consists of files within a single folder:

1) `training_data.npy` containing a NumPy array of shape $(48000, 2776)$, representing 48,000 individual time series, each with a length of 2776 data points.
2) `valid_periods.npy` containing another NumPy array of $(48000, 2)$ where each row corresponds to a time series in the `training_data.npy` file. For every time series, this array contains the start and end indices of the non-padded segment within the series.
3) `categories.npy` containing a NumPy array of shape $(48000,)$, providing the categorical classification for each of the 48,000 time series in the dataset. The categories are denoted by codes {'A', 'B', 'C', 'D', 'E', 'F'}. Each element in the array corresponds to the category code of its respective time series, indicating the grouping or classification the series belongs to.

Preliminary examination revealed that the dataset contains a significant amount of missing values that required a different approach than simply zero-padding. Further preprocessing techniques might be necessary to handle these missing values effectively, using `valid_periods.npy` to ensure the utilization of only valid segments within each time series.

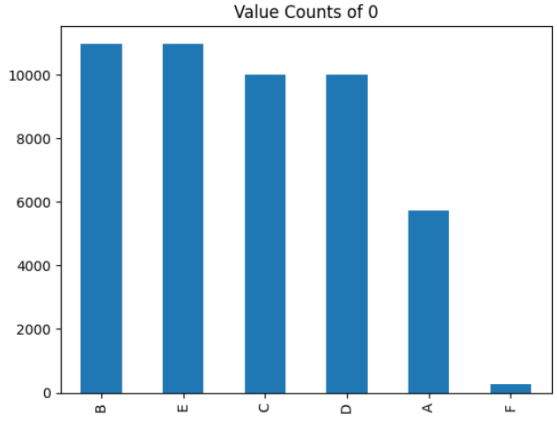

Figure 1. Number of 0s (padded) per category

1.1. Data Exploration

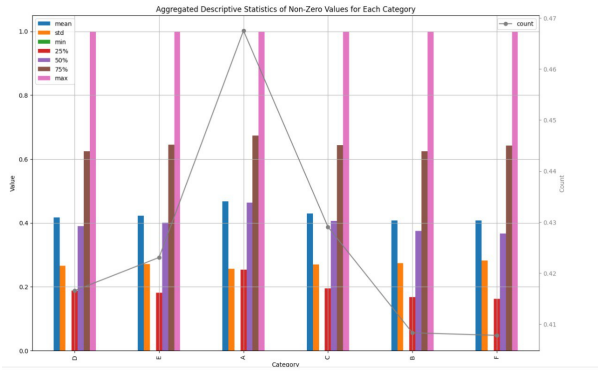

Figure 2. Statistics per each category

An additional exploration of the dataset was done

plotting the descriptive statistics per category such as mean, standard deviation (std), percentile values (25th, 50th, and 75th), minimum, and maximum values. Meanwhile, the gray line graph signifies the count of valid values for each category, showcasing the distribution and count of available data points for analysis.

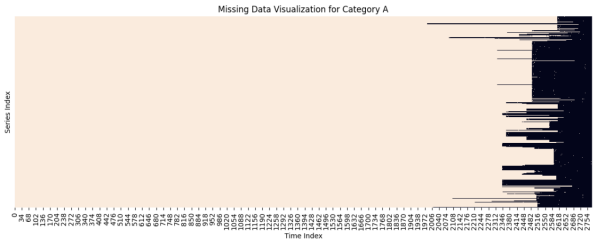

Figure 3. Missing data visualization for cat 'A' through heatmap

With a further analysis of the dataset we aimed at finding the segments afflicted by missing values. In particular, we employed heatmap visualizations to reveal patterns of missing data within each category. This technique allowed us to visualize and analyze the distribution of missing values across multiple time series.

The image depicted in Figure 3 reveals that the missing values occour only at the beginning of the time series. This pattern was consistent across all categories, not limited to Category A.

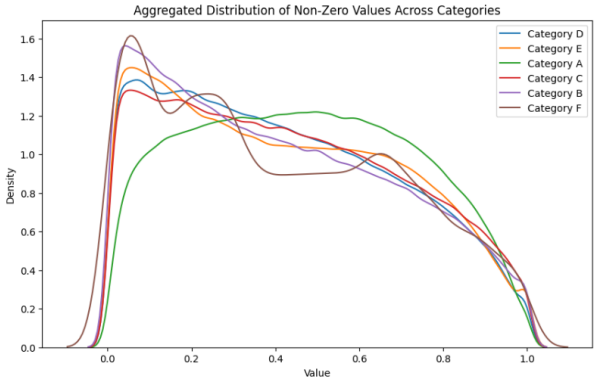

Figure 4. Missing data visualization for cat 'A' through heatmap

Moreover, the image depicted in Figure 4 revealed distinctive patterns among the categories. Category A displayed characteristics of a low kurtosis normal distribution, whereas Categories B, C, D, E, and F exhibited distributions resembling a Weibull distribution. Notably, Category F showed slight deviations compared to Categories B, C, D, and E within the dataset.

2. Data Pipeline

2.1. Pre-Processing

The dataset was partitioned based on category, combining the categories exhibiting `similar distribution patterns`. In particular, categories B, C, D, and E were integrated for collective treatment. To sum up, we have worked on dataset A, BCDE, and F and additionally, the valid periods corresponding to these datasets are stored in vdf_A, vdf_BCDE, and vdf_F.

2.1.1. Expanding Dataset F through ARIMA.

In order to enhance the F dataframe, consisting of only 277 time series with an average length of 194, we employed the `auto_arima` method from the `pmdarima` library. This methodology was applied to each time series, enabling the identification of optimal parameters for fitting an ARIMA model to the respective time series data. Subsequently, these optimal parameters were tested, and the Mean Squared Error (MSE) was computed based on the prediction of 20 time instances. Time series showing satisfactory MSE values were considered. If the MSE met the predefined threshold, the ARIMA model was utilized to append time instances to the existing series. This expansion was applied both forward and backward, allowing us to significantly increase the length of several time series and the create more sequences suitable for training and vdf_F was updated accordingly.

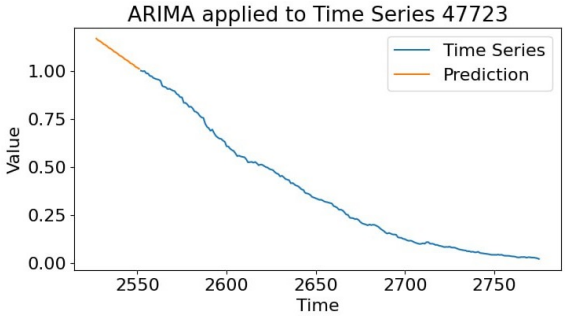

Figure 5. ARIMA backward prediction applied to a random serie

2.2. Sequences building

The way we built sequence utilizes the temporal boundaries, specifically the valid periods from each category within the dataset (vdf_A, vdf_BCDE, and vdf_F). Leveraging the 'start' and 'end' periods specified in the validity dataframes, the function constructs sequences while ensuring the inclusion of only valid time steps for model training (including the expanded ones of category F).

By respecting the designated valid periods for each category, this approach maximizes the utilization of available data, preventing the exclusion of potentially useful information during sequence generation. Consequently, it ensures that the models we've tried are trained on relevant and meaningful temporal sequences without wasting valuable data points.

2.3. LSTM

The architecture of the LSTM model we used consisted of two LSTM layers, each comprising 50 units, with the first LSTM layer set to return sequences and an input shape configured to match the window_size required for the sequences. Additionally, a layer of Normalization was incorporated after each LSTM layer to enhance the model's stability and convergence during training. To finalize the model, a Dense layer with the number of units corresponding to the future time steps to predict was appended. Throughout our experimentation, hyperparameter tuning was conducted to optimize the LSTM model's performance, ensuring the best configuration for the number of units in the LSTM layers, learning rate, batch size, and other relevant parameters.

2.4. GRU

Another architecture we've decided to try is GRU since it is known for its efficiency in certain scenarios compared to LSTM. The reduced complexity of GRU can prevent overfitting on smaller datasets, making it a favorable choice when computational resources or data availability are limited, like in our case. In this case, as before, the architecture we ended up using consisted of two GRU layers and two normalization layers alternated and a final Dense layer with the number of units corresponding to the future time steps to predict.

2.5. Final Submission

The model that performed the best utilizes the approach that worked the best out of all the ones we've tried. More specifically, we opted for the GRU model that, during the first phase, gave us the following results:

MSE	0.00589122 (317)
MAE	0.05330842 (237)

and during the second phase gave us these:

MSE	0.01123828 (257)
MAE	0.07451643 (271)

2.6. Final Conclusions

In summary, using LSTM and GRU models has been really promising for our time series analysis. The LSTM model with its complex layers did a good job predicting sequences, while the simpler GRU model showed quick training and sometimes even better predictions than LSTM due to its faster training speed. Still, there's a lot more to explore and fine-tune with these models. Further experimenting is needed with different setups to get the most out of them. This experience highlights the potential of LSTM and GRU models for time series forecasting but also emphasizes the need for further exploration to make them even better.

3. Contributions

- **Al Kamber:** Data exploration, ARIMA, Models, Tuning
- **Federica Vinciguerra:** Tuning, ARIMA, Report Writing, Models
- **Marton Barta:** Data exploration, Graphs Creation, Models