AN2DL TIME SERIES FORECASTING REPORT
TEAM: back_in_black

1. Scope and first contact with data:

The primary objective of this project was to construct a resilient and high-performing neural network model proficient in forecasting the trends of time series data. The initial phase of our endeavor involved a meticulous analysis of the provided dataset, aimed at gaining a better understanding of the inherent characteristics of the data under consideration.

The dataset in question comprises a collection of time series, categorized into six distinct groups, each operating independently. This categorization necessitates the treatment of these time series as univariate entities for forecasting. In our initial approach, we treated the categories as independent entities and aimed to develop distinct models for each category. However, upon further understanding of the data, we realized that dividing the entire dataset into six smaller datasets, each corresponding to a specific category, was not giving us the best results.

The models described in the following sections are implemented using the Keras-TensorFlow syntax. To evaluate the model performance, we utilized Mean Squared Error (MSE) and Mean Absolute Error (MAE) as the chosen metrics. These metrics were selected due to their prominence as monitored criteria in the competition.

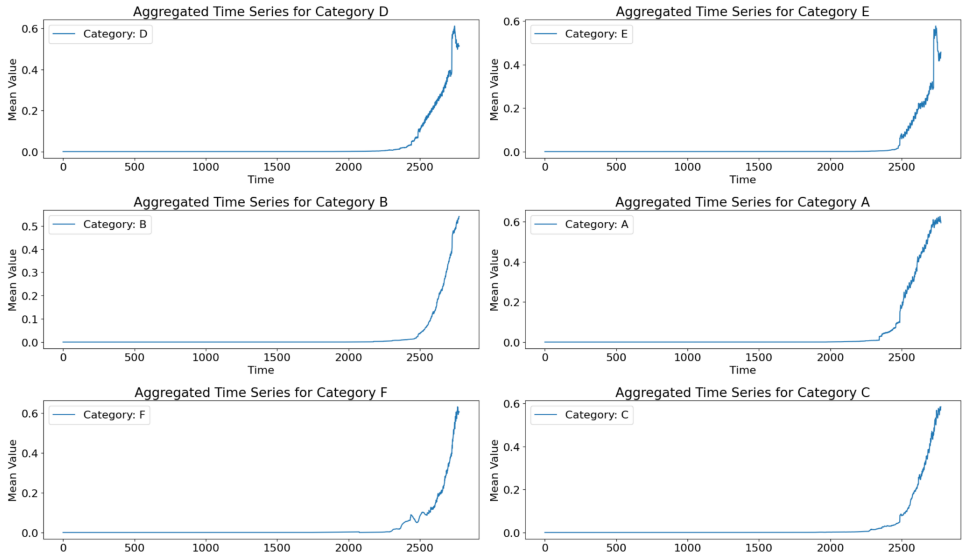
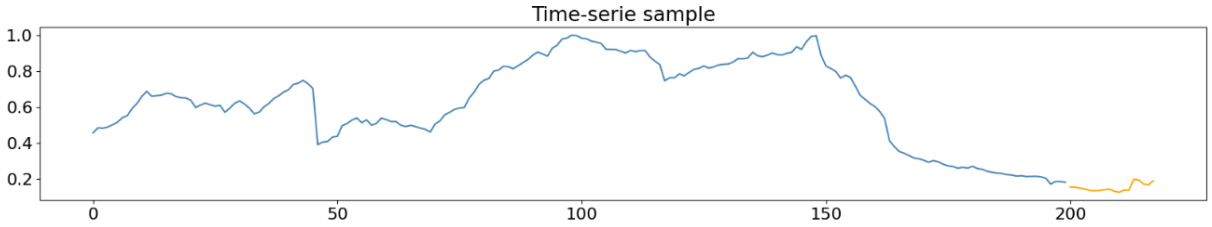


Figure 1: Mean of the series for each category

2. Data pre-processing

Employing a multistep approach, the data underwent several key transformations to enhance its analytical integrity. Firstly, the detrending process was applied to eliminate long-term trends, allowing for a more focused examination of underlying patterns. Subsequently, series exceeding the average length were systematically removed to mitigate the impact of outliers, ensuring a more balanced dataset. To further homogenize the data, we excluded the series commencing exceptionally early compared to the median, thereby minimizing the influence of initial trends that may not be representative of the overall temporal dynamics.



A sample of a time serie prepared for the training

3. Experimentation

In our initial methodology, we did not employ dataset division. We visually inspected the time series data to discern any discernible trends. It was observed that the pertinent information predominantly manifested towards the end of the series, with earlier time instants often padded with zeros. Subsequently, we pursued two distinct approaches: utilizing the complete series and truncating the data to focus solely on the relevant periods, as indicated in the 'periods.csv' file.

An early challenge we encountered was the sheer volume of data, surpassing the memory capacities of the tools at our disposal (Google Colab and Kaggle). To address this constraint, we opted to experiment with six distinct models. The rationale was to distribute the data across multiple models, allowing each to be trained on a larger subset and thereby facilitating a more nuanced comprehension of time series within specific categories.

Due to the temporal dependency inherent in the provided dataset and the specific nature of the problem we aimed to address, our initial approach involved the comparison of basic models utilizing various recurrent units. We began by defining straightforward architectures to establish a baseline for model complexity, facilitating subsequent comparisons.
The chosen recurrent architectures for comparison included LSTM, stacked-LSTM, and sequence-to-sequence. To ensure a fair evaluation, these models shared common characteristics: 256 recurrent units per layer, 50 epochs, a batch size of 256 samples, and an input/output window size of 200/18.

To address oscillating behaviors observed during the training process, we introduced a learning rate reduction schedule. Upon comparing these architectures, it became evident that LSTMs exhibited significantly better performance during training. Specifically, LSTMs demonstrated a time per epoch roughly 50% less than that of Simple RNNs. Furthermore, the behavior of LSTMs was noticeably smoother throughout the training process.

The models subjected to testing were as follows:

1. Two layers of Long Short-Term Memory (LSTM): the model consists of two LSTM layers. The first LSTM processes input sequences and a RepeatVector layer duplicates its output to match the target sequence length; the second LSTM generates sequences, followed by a Flatten layer to simplify the output. The model is completed with a Dense layer producing predictions for each element in the target sequence. Early stopping monitors validation loss for regularization

2. Bidirectional LSTM: This sequential neural network features three Bidirectional LSTM layers with dropout for regularization. It also incorporates two 1D convolutional layers with ReLU activation and padding.

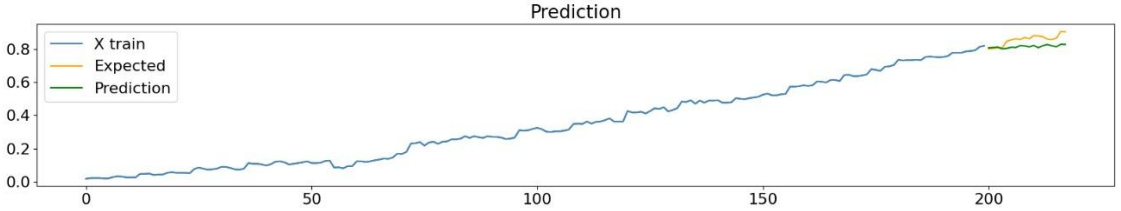
3. Sequence-to-Sequence (seq2seq) model: the encoder processes input sequences through an LSTM layer and applies batch normalization; the decoder, with a similar LSTM structure, incorporates batch normalization and receives attention-weighted information. Concatenating the decoder's output and attention results, the model adds a TimeDistributed dense layer to predict output sequences. The Adam optimizer minimizes mean squared error during training. This architecture aims to enhance sequence learning by incorporating attention mechanisms, normalizing intermediate representations, and mitigating overfitting through dropout regularization.

4. Sequence-to-Sequence (seq2seq) model with attention mechanism: this architecture is very similar to the latter described, enhanced by adding *Luong Attention Mechanism* (Luong et al., 2015), allowing the model to compute context vectors from the encoder's output.

4. Final model

Our final model is indeed composed of a sequence-to-sequence architecture, divided in encoder and decoder, both including a Bidirectional LSTM layer. Attention is also introduced to the decoder to focus on relevant parts of the input sequence during decoding. In particular:

- The encoder is implemented using a Bidirectional LSTM layer with n_units units (512 in this case) to capture information from both forward and backward sequences simultaneously. The encoder returns sequences, as well as the final hidden state (encoder_h) and cell state (encoder_c) for initializing the decoder.
- The decoder starts with a RepeatVector layer, which replicates the final hidden state (encoder_h) of the encoder to match the length of the input sequence. The decoder also includes a Bidirectional LSTM layer with half the number of units compared to the encoder. This layer captures contextual information in both directions.
- The attention mechanism is implemented using the Dot layer to compute dot products between the decoder output and encoder output, followed by a softmax activation to obtain attention weights.
- The concatenated and flattened tensor is passed through a Dense layer with 18 units. The output layer uses a linear activation function (default for Dense layer).
- The model is compiled using the Adam optimizer with a mean squared error loss function and mean absolute error as the metric.



Example of a time serie with the expected and predicted values

5. Contributions

Simone Zacchetti: rewriting of model.py, the first implementation of LSTM and Bidirectional LSTM, followed by several attempts of improvements with different architectures and/or different hyperparameters.

Matteo Zamuner: data analysis and pre-processing, first try with LSTM and Bidirectional LSTM models, building sequence-to-sequence models with attention. Trying different parameters tuning to try to improve performance.

Lorenzo Torsani: Data analysis, first implementation of an LSTM model, first try at building a sequence to sequence model, both with and without attention. Fine tuning of the hyperparameters and of the various possible activation functions

Lorenzo Auletta: LSTM models tuning and training, sequence-to-sequence models building, both with and without attention. Training of seq2seq models with hyperparameter tuning to lower validation loss.

6. References

1. Building Seq2Seq LSTM with Luong Attention in Keras for Time Series Forecasting, https://levelup.gitconnected.com/building-seq2seq-lstm-with-luong-attention-in-keras-for-time-series-forecasting-1ee00958decb
2. Understanding Autocorrelation in Time Series Analysis, https://towardsdatascience.com/understanding-autocorrelation-in-time-series-analysis-322ad52f2199
3. Multiple Time Series Forecasting With Holt-Winters In Python, https://forecastegy.com/posts/multiple-time-series-forecasting-with-holt-winters-in-python/
4. Hierarchical Time Series Forecasting with Python, https://forecastegy.com/posts/hierarchical-time-series-forecasting-python/#:~:text=Hierarchical%20forecasting%20is%20a%20method,different%20aggregation%20of%20the%20data