Time Series Forecasting Report

Artificial Neural Networks and Deep Learning - a.y. 2023/2024

Michael Fiano
Codalab Name: choil
Team Name: MicGPT
10676595 - 232790

Francesco Palumbo
Codalab Name: frapalu
Team Name: MicGPT
10711027 - 221919

Luca Negri
Codalab Name: lucanegri17
Team Name: MicGPT
10674684 - 222048

Federico Lamperti
Codalab Name: fedelampe11
Team Name: MicGPT
10680961 - 233462

***Abstract*—The goal of this challenge was to design and implement forecasting models to learn how to exploit past observations in the input sequences to correctly predict the future. In order to face this problem and to learn as much as possible, our team decided to organize the workflow, doing things step-by-step. In this report we discuss, explore and compare our development and design process.**

I. Dataset Analysis

A. Original Dataset Characteristics

We were given 3 different datasets:

- Training dataset: the principal dataset was composed of 48000 timeseries with a standardized length of 2776, each timeseries had its values and length, they were already normalized between 0 and 1 and the different length of timeseries was managed by adding zeros at the beginning of them in order to have all the series with the same length.
- Valid periods dataset: this dataset gave us information about the training dataset, in particular we were given the information of where timeseries begin and end. We noticed that every series ends at 2776.
- Category dataset: this dataset gave us information about the category in which any series can fit. These categories were in total 6: A, B, C, D, E, F.

The first thing we noticed about the training dataset was the fact that the categories were unbalanced:
B: 10987 E: 10975 C: 10017 D: 10016 A: 5728 F: 277

Then we checked for the possible presence of duplicates and, in fact, found 26 duplicated series. Given the small number of duplicates, we decided to keep them in our training.

The image below presents the visualization of the first timeseries in the dataset with its actual starting point (removing the padding):

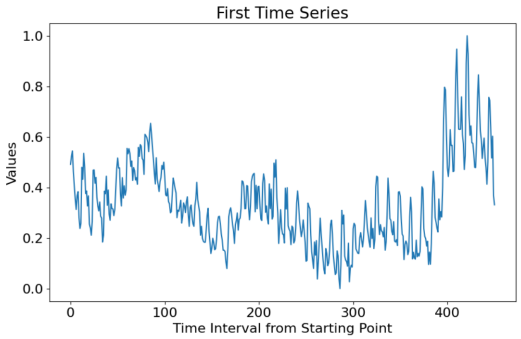

Fig. 1: First Timeseries with no padding

B. Input Pre-processing

In this section, we present every technique that we attempted to use throughout the evolution of the challenge. Not every one of them proved to be consistently useful in improving performance, therefore not all of the techniques are incorporated into our best model. However, for the sake of completeness, we will illustrate all of them.

The training dataset turned out to be extremely heavy in terms of memory, so to lighten the processes and to reduce future training times, we decided to cast every number inside the dataset passing from a float64 type to a float32. Another decision we needed to make was how to handle the padding added to the original dataset. We came out with two different solutions which are almost equivalent in terms of results obtained, but the second one was preferable since it made the process lighter. The first one was to transform the dataset into a dataframe that allowed us to use specific pandas' functions to easily work on it, but we had the constraint that every series had to be of

the same size, so we couldn't eliminate padding unless all the chosen series had padding in the same column. As an alternative we tried to change the padding value using the information in valid dataset to use a masking layer in the model that can detect padding. Another solution was to directly remove padding and concatenate the resulting series in a numpy array. Furthermore, we analyzed the dataset to identify outliers. In practical terms, we applied Z-score normalization to accentuate the outliers, using a threshold of 3 to categorize the timeseries as either valid or not. This analysis revealed that almost 10000, precisely 9744, timeseries could contain outliers. Consequently, we opted to remove them in certain models. Additionally, to ensure a balanced split between training and test sets, we introduced a shuffle to the original dataset. This process was exclusively applied to the models where we utilized the entire dataset or a consistent number of samples.

C. Training & Testing

Regarding the division of the dataset into training and testing, we employed two different approaches throughout our attempts. Initially, we divided the timestamps of each timeseries into training and testing sets using an 80-20% split, in this way we attempted to predict the last samples of each timeseries. Then we applied another approach, which involved splitting the entire dataset into two parts while maintaining the integrity of every timeseries. In this case, we used an 80-20% dataset split for some models and an absolute values split of 40,000-8,000 samples for other models.

Fig. 2: Visualization of a Train-Test splitting

D. Sequences creation

To prepare the data in the correct format for the model, we divided our data into different sequences. These sequences were created from the timeseries using the following parameters:

- window size = 200
- stride = 5
- telescope = 18

The window size represents the length of each sequence, while the stride represents how many timestamps are shifted between one sequence and the next. The telescope represents the forecasting horizon of each sequence, meaning that a sequence of length 200 will forecast 18 different timestamps. We conducted various experiments with different strides and found that a smaller stride was better for our purposes.

II. Model Selection

This section outlines our decision-making processes in model selection, detailing the rationale behind modifications from a basic model to an advanced ResNet architecture and the strategies employed to tackle overfitting.

A. Base Model

The model we used as a starting point was the CONV_LSTM model we saw during the laboratory. It consisted of several layers including an input layer, a Bidirectional LSTM (Long Short-Term Memory), a 1D Convolution, a final convolution to match the desired output shape, and the output layer. Our first concern was fitting the output to the shape required on Codalab. Since an LSTM requires a three-dimensional input but Codalab requires a two-dimensional output, we initially implemented a reshape layer followed by a TensorFlow Keras Lambda layer to squeeze out the third dimension. Subsequently, we explored various modifications to the model. To enhance the model's ability to capture complex and long-term data dependencies, we alternatively implemented two and three stacked bidirectional LSTM layers. We also attempted to increase the dilation rate in the 1D convolution layer, aiming to cover a larger receptive field in the timeseries without significantly increasing computational complexity. Additionally, we added an attention layer to improve model interpretability, which in turn enabled enhanced performance on complex and diverse datasets. Finally, although substituting the LSTM with GRU (Gated Recurrent Units) was considered, these changes unfortunately made the model more complex without yielding any notable benefits. The modification that had the most positive impact involved altering the method used to construct the final shape. By using a Flatten layer followed by a dense layer with 18 units, we added a significant number of parameters to the model. This increase in parameters consequently led to a longer computational time, but we noticed substantial improvements in performance.

B. ResNet

As another option, we decided to build new models by following the style of the Keras pre-trained model 'ResNet', in particular the introduction of 1D Convolutions.

- The first approach consists in creating a residual block by sequentially stacking 1D convolutions, batch normalizations and ReLu activation functions. This residual block was repeated 3 sequential times in the model, then followed by a Flatten layer and 2 Dense layers.
- In order to further considering the importance of the previous inputs, we added an Attention layer and a Bidirectional LSTM to our ResNet model: the first time we used a batch size of 512, then we retrained the same model using a batch size of 64 samples, which slightly improved the overall performance.

Evolution	Test_Loss
ResNet-style	0.083
ResNet-style + Attention + LSTM (batch 512)	0.059
ResNet-style + Attention + LSTM (batch 64)	0.022

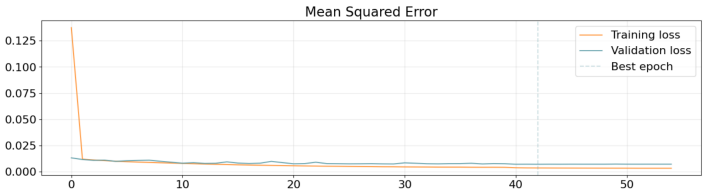

Fig. 3: ResNet Training and Validation Loss

C. Overfitting Management

We employed various methods to mitigate overfitting as much as possible. This included implementing early stopping with a patience of 12 based on validation loss, learning rate scheduling with a patience of 10, also based on validation loss, and, in some cases, applying dropout layers to the model.

III. Final Submission Model

A. Model

The model we selected for submission in the final phase is one of the least complex we investigated during the challenge. According to the results obtained, we decided that simplicity was key to achieve the best predictions. Our decision to focus on 500 series, as opposed to the entire dataset of 48,000 series, was a strategic choice. It balanced the depth of analysis with practical constraints, ensuring that our findings were both meaningful and relevant, especially considering the size of the hidden testing dataset. Additionally, this decision was influenced by considerations of time complexity, as working with a smaller dataset significantly reduced the computational time required for analysis. The decision to focus on the first 500 series was not arbitrary. We analyzed the timeseries of each category and noticed that category D outperformed the others on the Codalab platform. This discovery led us to believe that this category might be more representative of the hidden test set compared to the others.

- Epochs = 200
- Batch size = 64
- Loss = Mean Squared Error
- Optimizer = Adam

Layer (type)	Output Shape	Param #
Input	(None, 200, 1)	0
Bidirectional lstm	(None, 200, 128)	33792
conv1D	(None, 200, 128)	49280
flatten	(None, 25600)	0
Dense	(None, 18)	460818
Reshape	(None, 18, 1)	0
Lambda	(None, 18)	0

Total params: 543,890
Trainable params: 543,890
Non-trainable params: 0

B. Local Performance

We trained the model using a 80-20% dataset split on training and validation. The model achieved its best value at epoch 4.

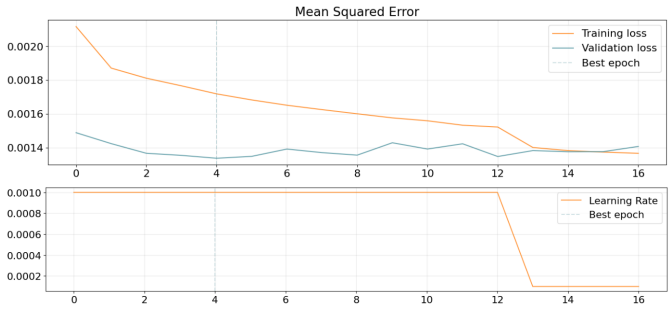

Fig. 4: Mean Squared Error & learning rate evolution

C. Leaderboard Performance

Development Phase MSE: 0.00563349
Development Phase MAE: 0.05651296
Final Phase MSE: 0.01140117
Final Phase MAE: 0.07496400

IV. Contributions

In the first phase of this challenge, we decided to have a first look at the dataset by ourselves to be able to experiment with different techniques of data management and different models. After a while, we reconvened to take stock of our operations and tried to mix the different approaches to have the best possible result. In the end everyone worked on his solutions and informed the others about the results.