Time Series forecasting
Artificial Neural Networks and Deep Learning – A.Y. 2023/2024

Reza Shams*, Mahdieh Jalali† and Romina Saljooghian‡
M.Sc. Automation and Control Engineering, Politecnico di Milano - Milan, Italy
*Email: *reza.shams@mail.polimi.it, †mahdieh.jalali@mail.polimi.it, ‡romina.saljooghian@mail.polimi.it*
*Student ID: *10811768, †10754004, ‡10888850*
Codalab Group: "Soft98"

1. Introduction

This challenge aimed to train models capable of forecasting time series using past input sequences. The given training set contains 48000 time series samples padded with zeros and a maximum length of $T = 2776$ steps. There are 6 different classes that the samples belong to.

We divided the dataset using an 85:15 train-validation ratio. The report's results were obtained from the validation set. Our approach involved creating various models and analyzing their performances. Ultimately, we determined the best-performing model based on the MSE value.

2. Data Pipeline

2.1. Pre-Processing

For better modeling of the data, we decided to standardize the data. Although the data was already normalized between 0 and 1, we experimented with other methods such as robust scalar after separation of zero padding and valid data. By applying the robust scaler, we ensured that extreme values or anomalies in the time series did not disproportionately influence the model's performance, contributing to more robust and reliable forecasts. The same transformation was applied to the validation and remote test set for consistency.

2.2. Data-Stationarity

Differentiating time-series signals is used to make them stationary, so that the statistical properties of the signal, such as mean, and variance remain constant over time. While this technique aims to stabilize the data and simplify the modeling process, our initial expectations of achieving more accurate predictions were not met. Consequently, we opted to utilize the output of the robust scaler for subsequent modeling.

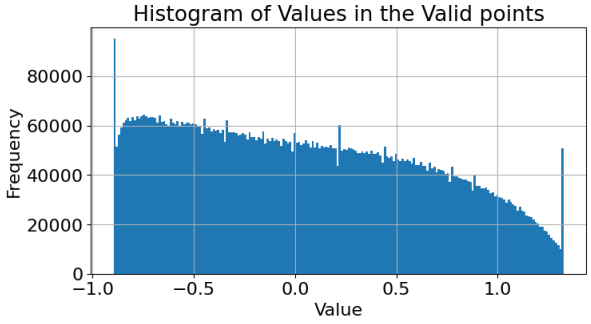

Figure 1. Frequency of valid data points after Robust-scaler implementation

2.3. Data Autocorrelation

In determining the optimal window size to extract the most valuable information from past samples for all time series, we employed the Autocorrelation function. By averaging the computed optimal windows for each time series, a window size of 60 was selected. When defining individual windows, values exceeding 0.2 (as indicated in Figure 2) were deemed to exhibit significant temporal dependencies with the original time series.

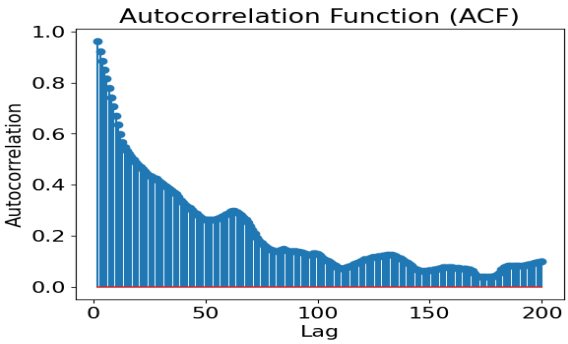

Figure 2. Autocorrelation Function on a random time series sample

3. Models

We assessed various model structures, including Vanilla LSTM, Convolutional LSTM, Bidirectional LSTM, Attention-based Model, Attention-based LSTM, and Encoder-Decoder. Throughout all modeling phases, we employed an optimal window size of 60 and a stride of 20 (except Vanilla LSTM(1)). In phase 1, a telescope size of 9 was chosen, while in phase 2, two approaches were explored for forecasting an output size of 18: either utilizing a telescope size of 18 directly or employing an Autoregressive approach to generate 18 timestamps step-by-step.

3.1. Vanilla LSTM

We began with the simplest model as we don't believe in unnecessary complexity. So 2 layers of LSTM and 1 Dense with a dropout of (0.2) was used (called Vanilla LSTM). Then we enhanced model complexity by incorporating residual connections to facilitate the learning of identity mappings. Residual connections prevent over-reliance on specific pathways and improve generalization. They also serve as a form of regularization and help overcome the vanishing gradient problem. Also add `GlobalAveragePooling` and `BatchNormalization` to avoid overfitting even more (called Skip LSTM). To boost the model's performance, we can add a convolutional layer after one of the LSTM branches (called Conv. LSTM). This improves the ability to capture spatial features.

3.2. Bidirectional LSTM

The bidirectional LSTM architecture considers both past and future context, which further enhances LSTM's ability to model temporal relationships.

The chosen model comprises an input layer and a sequence of Bidirectional LSTM and Dense layers. The first Bidirectional LSTM layer (bilstm1) processes the input data, capturing temporal dependencies in both directions with 256 units. The dropout layer (dropout1) helps prevent overfitting, by preventing the model from memorizing the training data too closely. The second Bidirectional LSTM layer (bilstm2) further refines the representation and captures more complex patterns with 512 units and another dropout layer (dropout2) for regularization.

After passing through the LSTM layers, the output is flattened and processed through a dense layer (dense1) with 512 units. The final step involves predicting the next time step with a dense layer (dense2) and a leaky ReLU activation function for learning complex relationships between the input features and the predicted values..

3.3. Transformer

The transformer architecture has been successfully applied in various natural language processing tasks, and its ability to capture long-range dependencies makes it well-suited for time series forecasting. The implementation consists of two main components: the encoder and the decoder. The encoder processes the input time series data, while the decoder generates predictions for future time steps. The decoder follows a similar architecture to the encoder, but it also incorporates masked self-attention to prevent the model from peeking at future time steps. Here are the components of the decoder part:

- **Self-Attention**: The decoder applies self-attention to its own output, allowing it to focus on the most relevant parts of its own predictions.
- **Encoder-Decoder Attention**: The decoder also utilizes encoder-decoder attention, allowing it to incorporate information from the encoder's processed input. This helps the model learn long-range dependencies between past and future time steps.
- **Feed Forward Network**: Similar to the encoder, the decoder's output is passed through a feed forward network to learn more complex relationships.

Although the proved capabilities of attention mechanism, in this case we couldn't achieve a better result

w.r.t simpler architectures that we used previously. It can be due to have inadequate hyperparameters, or non-stationarity features of the timeseries.

4. Training techniques

Our models were trained using the following specifications:

- Batch size: 64 and 256
- Optimizer: Adam
- Learning rate: 0.00001
- Callbacks: **EarlyStopping** with patience 10, **ReduceLROnPlateau** with patience 5 and factor 0.5

4.1. Class imbalance

Class imbalance played an important role, as can be seen in Figure 3.

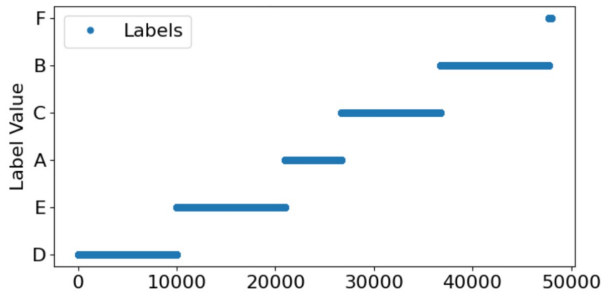

Figure 3. Number of samples in each class based on index

During the model training process, we adopted a technique that involved adjusting class weights using the function `compute_class_weight`. This approach allowed us to penalize the misclassification of the minority class by assigning a higher weight to it.

4.2. Autoregressive

The autoregressive model is a statistical technique that offers numerous advantages, such as the ability to model time-series data by detecting associations between a variable and its previous values. Additionally, it can flexibly adapt to evolving patterns in the data, leading to enhanced predictive accuracy. In the first phase, we attempted to train our model with three samples to utilize autoregressive techniques for prediction. Unfortunately, we did not observe any improvement in the results when compared to predicting all nine samples. However, in the second phase, we were able to achieve the best possible outcome by making use of autoregressive methods.

5. Model choice

We compared the different models, both with and without Autoregressive and also Normalization.

Models	MSE
Skip LSTM	0.0160
Vanilla LSTM	0.0164
Conv. LSTM	**0.0094**
Bidirectional LSTM	0.0153
Attention-based	0.0213
Attention-based LSTM	0.0153
Encoder-Decoder	0.0149

6. Conclusion

Although we employed advanced models to attain optimal results for the given time series forecasting task, our experiments showed that the Vanilla LSTM model outperformed the others in the first phase. This doesn't mean that more advanced models are incapable of achieving superior results; instead, it indicates that they require further refinement to deliver enhanced outcomes. In the second phase of our experiment, we evaluated three branches of LSTM models and discovered that the most promising outcome was attained by incorporating a convolutional layer at the end. We also computed the MSE on the test data, which turned out to be 0.0109. After applying autoregressive techniques, we were able to reduce the MSE to 0.01.

7. Further Developments

We acknowledge that there is room for enhancement in certain aspects of our work. A more thorough analysis of the dataset and the incorporation of statistical feature extraction methods could enhance our approach.

Addressing class imbalance, particularly in class "F", by applying the weights we get an MSE of 0.0064 for 9 samples on the test data, we suggest two strategies: training for 5 major classes and fine-tuning for the under-represented class, or training 6 specialized models and creating an ensemble.

While we briefly explored "Attention" layers, we found classical techniques more effective for now. We

believe architectural improvements can further refine our approach.

8. Contribution

This work is the result of the collaboration of all team members. Particularly Mahdieh worked on Data preprocessing. Mahdieh explored the statistical feature of data and trained the Vanilla LSTM model. Designing more advanced models is the result of the collaboration of Reza and Romina together. Reza pushed the development of the transformers and designed the attention based models in addition to applying autocorrelation for the selection of window size. Romina utilized various LSTM models to create a general model. She accomplished this by increasing model complexity and applying autoregression.