

Report - Challenge 2

Introduction

In this homework we were asked to predict future samples of the input time series implementing forecasting models to learn how to exploit past observations in the input sequences to correctly predict the future. We were provided of only the train set, to be used to learn the models. The submissions were then evaluated on a hidden test set.

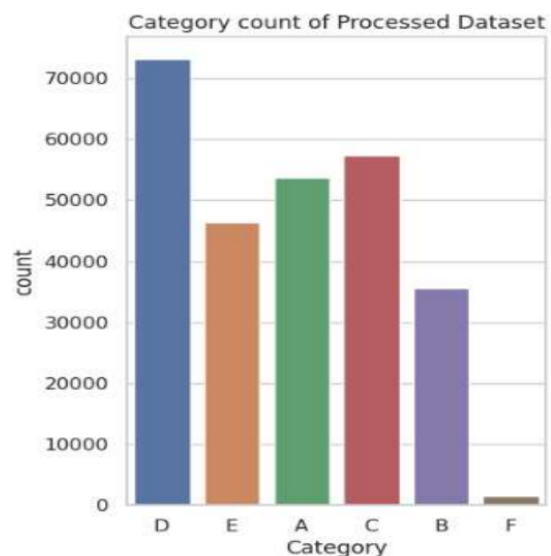
This report outlines the development process of a time series forecasting model.

The task was to accurately forecast future values based on historical time sequence data. The report explores the journey from the initial data exploration and preprocessing, through the iterative model development process, to the final model architecture. Also the different challenges and learning experiences encountered in dealing with time series data are discussed!

Data Exploration

Data Preparation

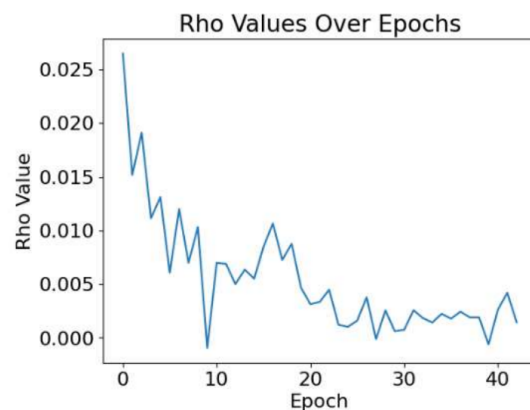
The initial phase was focused on comprehending the dataset. It started by **un-padding the data** within the *valid periods* and removing all sequences that were shorter than 200 plus the output window size. Then, **sequences of specific lengths (e.g., 209, 218, 206, etc.) were systematically extracted** along with their corresponding categories, experimenting with different strides. Of course, the smaller the stride, the higher the **correlation** between sequences. Eventually, the selected stride was equal to the output window size.



This process yielded a substantially **larger dataset** exclusively composed of sequences of predefined lengths. To address the issue of sequences with excessive zeroes, the ones with more than 5% zeroes were discarded. Finally, to enhance the model's generalization capabilities, the sequences were shuffled to prevent any learning bias.

Autocorrelation

To address autocorrelation in time series forecasting, where errors are not iid but dependent on their predecessors, a strategy from Sun, Lang & Boning (2021) [3] was adopted. The approach involved a custom neural network layer dynamically adjusting the input, to model the first order autocorrelation: $input = X_t - \rho X_{t-1}$.



There were two main issues: ρ wasn't stable enough, fluctuating significantly; moreover, for implementation issues, we could only use the last sequence of the previous batch as X_{t-1} . Ultimately, as ρ converged to zero across all models, it was decided not to include this layer in the final model.

Robust Scaling

To make the models more robust against outliers a RobustScaler was fitted on the Train data and then used to transform the validation and test sets accordingly. Comparing models trained with normalized and non-normalized data it was found out that unfortunately, this technique didn't lead to any improvement in the performances. In fact, it resulted in higher MSE.

Data Augmentation

We tried adding noise to the Train data to perform data augmentation through a data generator.

We added Gaussian noise $\sim N(0, \sigma)$ to them, choosing σ^2 as a fraction of the empirical variance of the sequence. Unfortunately, this technique did not lead to better performing models. Probably, having non normalized data prevented us from properly choose the hyperparameters.

Models

Persistence model

Guided by the suggestion of Ughi, Lomurno & Matteucci (2023) [2], a **performance baseline** using a Persistence model (a simple benchmark model used to predict future values based solely on the most recent observation in the dataset. This type of model assumes that the best prediction for the next time step is the current value) was established: when applied

to the test set, it achieved a MSE of 0.02 on forecasting the next 18 values from sequences of length 200. We compared this baseline to the performance of other models we experimented with.

Conv1D ResNet

A ResNet architecture composed by blocks of Conv1D layers, hypothesizing that their ability to **identify patterns over sliding windows** could yield meaningful results. Initially, it was implemented the full 3 blocks architecture, which proved **overly complex**: it featured over 500,000 parameters, resulting in lengthy training times and overfitting.

To mitigate these issues, the last Conv1D block was

frozen and the training was continued, which resulted in better performance (0.0098 Validation MSE).

Attention Layer

While exploring ResNet architectures, we recognized skip layers were functioning as **basic forms of attention mechanisms**, with each layer adding the entire input to the latent representation. Moreover, we noticed that the Global Average Pooling was acting as a **learning bottleneck**.

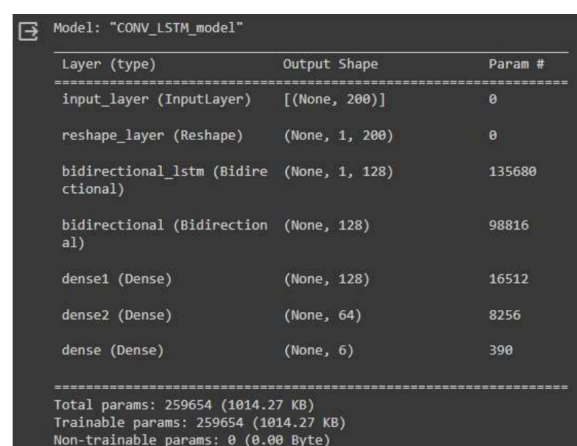
Combining these insights, the architecture was modified by replacing the final GAP layer with a

Bahdanau Attention layer [4] - a form of multi-layer perceptron attention. This modification achieved a slight but notable improvement, bringing the Validation MSE down to 0.0069.

LSTM, Conv1D & Dense Models – Best Performing Model

Conv1D layers were combined to achieve **dimensionality reduction**, followed by LSTM serving as **feature extractors**, which were then combined by Dense layers.

Through the extensive exploration of various architectures and hyperparameters, we discovered the **critical role** of Dense layers on the performance. Instead, Conv1D layers were not significant.



The image shows a screenshot of a terminal window displaying the Keras model summary for a model named "CONV_LSTM_model". The summary lists the layers, their types, output shapes, and the number of parameters. The layers are: input_layer (InputLayer), reshape_layer (Reshape), bidirectional_lstm (Bidirectional), bidirectional (Bidirectional), dense1 (Dense), dense2 (Dense), and dense (Dense). The total number of parameters is 259654 (1014.27 KB), with 259654 trainable parameters and 0 non-trainable parameters.

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	[(None, 200)]	0
reshape_layer (Reshape)	(None, 1, 200)	0
bidirectional_lstm (Bidirectional)	(None, 1, 128)	135680
bidirectional (Bidirectional)	(None, 128)	98816
dense1 (Dense)	(None, 128)	16512
dense2 (Dense)	(None, 64)	8256
dense (Dense)	(None, 6)	390

=====
Total params: 259654 (1014.27 KB)
Trainable params: 259654 (1014.27 KB)
Non-trainable params: 0 (0.00 Byte)

We also experimented with the output time window prediction obtaining that the better result by

sequentially predicting 6 values at a time, incorporating these predictions in the input to

predict the next 6 and so on until reaching 18 output values. Our experiments revealed that, given the data's linear scaling, employing sigmoid output layers was **less effective** compared to **linear layers**.

Cross-Category Learning

Models trained exclusively on data from a specific category acquired knowledge also about others, **learning patterns common between them**. Consequently, we adopted a strategy of initially training models on mixed category data, followed by **fine-tuning** them on individual categories. This approach led to the creation of six distinct models (one for each category), which improved our prediction accuracy.

Bibliography

1. Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., ... & Brubaker, M. (2019). Time2vec: Learning a vector representation of time. arXiv preprint arXiv:1907.05321.
2. Ughi, R., Lomurno, E., & Matteucci, M. (2023). Two Steps Forward and One Behind: Rethinking Time Series Forecasting with Deep Learning. arXiv preprint arXiv:2304.04553.
3. Sun, F. K., Lang, C., & Boning, D. (2021). Adjusting for autocorrelated errors in neural networks for time series. Advances in Neural Information Processing Systems, 34, 29806-29819.
4. Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.