TIME SERIES FORECASTING CHALLENGE 2023

TEAM: CARS

Sophia Barbera 10661106 Caterina Giardi 10917727

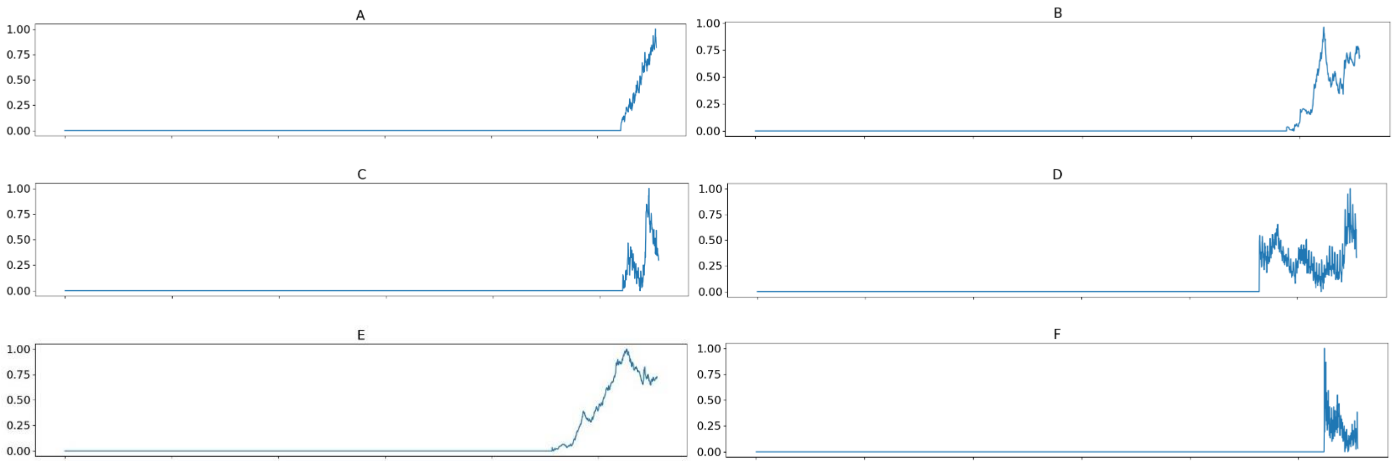
Anna De Simone 10872083 Riccardo Storchi 10915631

1 DATASET SPECIFICATION

The initial dataset consisted of 48000 monovariate time series [1] belonging to six different domains denoted by the characters {A, B, C, D, E, F}. Each time series in the training dataset has a variable length that was padded with zeros to have all the sequences of the same length: 2776. For each of them it was provided the start and end indices without padding.

The time series of each domain don't provide any additional information on the time series but just indicates that they are collected from similar data sources.

The dataset is designed for building models capable of generalizing predictions for the future samples of the 60 time series in the test set. The way the dataset is set up, along with the files it comes with, gives a solid foundation for creating and testing models that predict outcomes for single-feature time series in different areas.



2 DATASET SPLIT, ENCODING AND PREPOSSESSING

In order to subject our model to a valid performance evaluation, it was necessary to split the dataset into a training set, validation set and test set. This split was performed using the train_test_split function, which generated a training set of 95% and a test set of the same size of the validation set consisting of the two remaining 5% of the dataset, respectively. To prepare the data for model training, a sequence creation procedure was implemented. This procedure involved dividing the time series into windows of predetermined length, specified by 'window', with a shift of 'stride' between windows. In addition, a parameter 'telescope' was introduced, representing the length of the sequence to be predicted. To avoid runtime problems and ensure robust training, appropriate controls were applied, such as handling any different padding lengths to ensure that the total number of data points is a multiple of the window length.

We tried two different data preprocessing: one in which we firstly took the last 209 time steps of each time series, and one in which we decided to kept from each category some time series. While going for the first approach, we noticed that some signal were too short to have a training part, so we rebalanced the dataset by moving some signal to the training set, stratifying with respect to the category.

In conclusion, the partitioning of the dataset and the preparation of the data for the training phase were crucial steps in ensuring the validity and effectiveness of the model, enabling accurate evaluation of its performance on the test data.

3 EXPERIMENTS

In the development of our predictive model for the dataset, we conducted a series of experiments organized into distinct phases, each of which provided valuable insights. The following subsections detail the key approaches and findings from our experiments.

3.1 Initial Model Exploration and GRU

We initially faced the challenge by splitting the dataset into test and training sets, considering each category separately. Using a dictionary composed of two sub-dictionaries to contain information about the date and label for the test (such as "test['A']['data']" or "train['C']['label']"), we explored LSTM [2] and GRU [3] models.

However, it emerged that the GRU, despite its leaner architecture and computationally less expensive nature compared to LSTM, struggled to capture the intricate dependencies and long-term relationships present in our data. Evaluation metrics, including the loss function and accuracy, clearly indicated better performance of the LSTM model compared to the GRU during this initial phase.

Consequently, we decided to prioritize the use of the LSTM model for our specific context, as its more complex structure proved more suitable for our data. This strategic choice was driven by the need to obtain more accurate predictions and a more effective representation of the underlying dynamics in our dataset.

3.2 Optimizing Forecasting Performance

Subsequently, after several attempts, we observed that keeping the signals uncategorized led to superior performance. Specifically, we removed padding from each signal, focusing only on valid signals (observing initial and final values), and divided the signals into windows of 200 for testing and 9 for training, with a stride of 5. We then reintegrated padding to ensure the correct window size.

3.3 Introduction of ResNet

In this subsequent phase, we introduced ResNet [4], characterized by the implementation of residual blocks. Each residual block consists of two convolutional layers followed by batch normalization. The use of a skip connection to add the original input to the transformed data allows the model to learn residual errors, addressing the issue of gradient vanishing in deep neural networks.

We experimented with varying numbers of residual blocks, starting with an initial quantity and adjusting it to 10 based on specific needs. Initially, we experimented with a few blocks, but seeing no significant improvements, we later increased their number.

3.4 Decision Between Autoregressive and Direct Models

Continuing the exploration of model architectures, we grappled with the decision between an autoregressive and a direct model. Through numerous experiments, including the addition of convolutional layers, bidirectional and unidirectional LSTM layers, each with varying numbers of neurons, we consistently observed the superiority of the direct model over the autoregressive model. Particularly, models with a direct approach showed significantly lower loss values, with a notable difference between direct models (0.0049) and autoregressive models (with a minimum of 0.0070). This performance disparity guided our preference toward the direct model for the application at hand.

4 Final model

4.1 Hyperparameters

- BATCH: we used mini-batch gradient descent with a batch size of 256 samples.
- EPOCHS: we used a maximum of 50 epochs for the training to find a balance between learning from the data and mitigating the risk of overfitting.
- EARLY_STOPPING_PATIENCE: we used a patience of 12 epochs in the trainings to allow the model sufficient time to navigate potential fluctuations in performance.
- REDUCE_LRO_PLATEAU_PATINCE: we used a patience of 10 epochs facilitating convergence in case of slower progress during training.
- OPTIMIZER: we used Adam Optimization due to its superior speed, lower memory requirements, and ability to deliver strong performance outcomes.
- LEARING_RATE: we used the default value of Adam optimizers which is 1e-3 (0.001) to avoid fast overfitting.
- LOSS: we used Mean Squared Error since it performs better for our problem, as it effectively penalizes larger errors, providing a suitable measure for regression tasks and guiding the training process to minimize the overall squared differences between predicted and actual values.

4.2 LSTM and model Layers

In developing our chosen LSTM direct model, we incorporated a Bidirectional LSTM layer with 128 units, which enables the model to capture complex temporal relationships in the data. This layer is followed by an additional LSTM layer with 64 units.

To improve feature extraction and solve overfitting problems, we introduced a 1D convolutional layer with 64 filters and a kernel size of 3, using a ReLU activation function to introduce nonlinearity. This is followed by a Dropout layer with a 30% rate, which acts as a regularization mechanism to prevent overfitting during training. Global average pooling was then applied to effectively reduce the dimensionality of the output. The implementation of this architecture of this architecture used TensorFlow and Keras [5], taking advantage of their specialized layers for building constructing bidirectional LSTM networks.

It should be noted that we experimented with more complex models by increasing the number of convolutional layers.
However, given the already large number of parameters in our training set, this resulted in reduced performance due to overfitting.

References

[1] Time series forecasting https://royalsocietypublishing.org/doi/full/10.1098/rsta.2020.0209

[2] LSTM https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

[3] GRU https://machinelearningmastery.com/recurrent-neural-network-algorithms-for-deep-learning/

[4] ResNet https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/

[5] TensorFlow and Keras https://www.tensorflow.org/guide/keras?hl=it

Contributions

All team members equally collaborated on our time series forecasting homework, starting from the dataset split, encoding and prepossessing and ending with the testing of the various model. Our teamwork was effective, as everyone actively contributed ideas, collected data, and formulated key insights for successful completion of the forecasting task. Each person played a vital role in achieving our goals.