Artificial Neural Networks and Deep Learning

Time Series Forecasting - Homework 2 - Report 2023

Paolo Pertino, Alberto Sandri, Enrico Simionato

Problem Description In artificial intelligence and deep learning, an important challenge is represented by time series forecasting, which necessitates strategies to unravel the complexities inherent in temporal patterns. Time series analysis is pivotal in various domains, spanning for example finance, weather prediction, stock market analysis, epidemiology, energy consumption modeling, and signal processing. This report outlines the methods, techniques, and results of the analysis conducted to achieve the specified goal, shedding light on the application of different kinds of deep neural networks, including convolutional neural networks, LSTMs, and particular versions of fully connected networks, in addressing the aforementioned task.
The objective is to forecast the future 9 and 18 samples of time series, that in the test set have length 200. The metric used to evaluate models is the Mean Squared Error (MSE).

Data inspection During the data exploration stage, after the visualization of the time series[IMG1], we conducted basic checks to identify and address duplicates. Our examination revealed the presence of 26 duplicated time series, thus we decided to eliminate them from the dataset. Each series is associated with a specific category denoted as 'A', 'B', 'C', 'D', 'E' or 'F', representing their respective subjects of reference. Despite visual inspection failing to reveal any relevant pattern connecting the two types of information[IMG2], we opted to include them in a few models during training for further exploration.

Preprocessing Different preprocessing ideas were tried on the already scaled series in [0, 1]. First of all, as suggested, we tried to apply the Scikit-Learn RobustScaler to our dataset. In general, it is a method for scaling features of a dataset based on their interquartile range and median, making it more robust to outliers. No evident benefits were noticed in the first place when applying this tool, thus it has been dismissed for the latest tests.

For the generation of the training, validation, and test datasets, we began by filtering out instances with a length below the specified window size. Subsequently, the original dataset was partitioned into three distinct subsets. The final datasets were then constructed by generating samples with specified window size, stride, and telescope. From the beginning, our objective was to train the models to forecast 18 samples.

An experiment was conducted to determine the optimal window size based on the order of the random processes representing the time series. We employed the 'auto-arima' function from the pmdarima library, leveraging the Box-Jenkins method [4] to identify the time series order. The objective was to identify the highest order within the time series and employ it as the window size. Regrettably, this method proved to be non-scalable, and thus not applicable to the entire dataset. Through experimentation instead, we observed that optimal results were attained when employing a window size within the range of 100 to 150 samples.

Best model - Ensemble The best model we obtained was the ensemble of many models, in particular, several N-Beats models and Bidirectional LSTM models were grouped together. (Both models are further explained in the following sections.)

By following the N-Beats authors' studies, we acknowledged that ensembling was significantly a better choice for regularizing, which rapidly led to an increase in performances up to 0.005. Since diversity is a fundamental characteristic of ensembles, we diversified all the learners by varying the input window duration (ranging from 2H to 7H, given a horizon H=18) and performing bootstrapping on the dataset. In addition, for N-Beats, we adjusted the number of stacks and tuned the number of neurons in each dense layer. Specifically, following Bayesian optimization facilitated by the Keras tuner, we selected networks with 6 stacks and 512 neurons per dense layer, as well as networks with 8 stacks featuring 256 neurons per dense layer.
In the context of LSTM-based models, we chose the top-performing configuration, which is a singular model with a single layer of Bidirectional LSTM with 64 units and a dropout rate of 0.2 and we trained it on datasets generated with different strides.
Initially, we conducted training and testing phases, both locally and on Codalab, employing homogeneous ensembles consisting exclusively of either N-Beats models or Bidirectional LSTM models. Notably, there was an enhancement in performance. The most significant improvement was instead observed when utilizing an heterogeneous ensemble incorporating both types of neural networks. The performances on Codalab of the best ensemble were: in the first phase MSE: 0.0049; and in the second phase were MSE: 0.0102.

Experiments

LSTM + Conv The model employs a blend of bidirectional LSTM layers, which grasp sequences, and one-dimensional convolutional layers to capture temporal patterns and relationships among the results of the LSTM processing. A cropping operation is then applied to ensure the model generates the intended output by trimming it to the correct length post-convolutions. Despite promising initial results, subsequent hyperparameter tuning revealed that the model exhibited a reduced capacity compared to others in effectively learning patterns within the data, leading to a tendency to overfit.

Bidirectional LSTM This architecture employs a singular layer of bidirectional LSTMs, enabling concurrent forward and backward sequence processing. This bidirectional strategy significantly improves the model's comprehension of temporal patterns, a critical aspect of time series forecasting.
The bidirectional LSTM is utilized to create a representation of the series, outputting only the final state after processing the entire input sequence. Subsequently, a dense output layer is employed for generating predictions. Notably, the incorporation of dropout was observed to enhance generalization. Despite its simplicity and minimal parameter count, the model outperformed several more complex alternatives. This straightforward design not only facilitates easier model interpretation but also contributes to faster training and more efficient utilization of computational resources.
Various architecture variants were explored, involving additional dense layers and LSTM components, yet the simplest configuration demonstrated superior performance both locally and on Codalab, achieving an MSE around 0.0055 in the first phase, reaffirming its efficacy among the considered alternatives.
In summary, the success of this architecture underscores the principle that simplicity in design can yield remarkably effective solutions in the field of time series forecasting.

Category-Integrated Neural Network (CINN 🥂) In our pursuit to leverage category information, we explored two approaches: building separate models for each category or integrating category information within the neural network itself (CINN). Initially, we considered

training individual models for each category but found this approach didn't significantly enhance performance compared to a model trained on all categories. Consequently, we discarded this idea. We then questioned the meaningfulness of the categories and opted to embed this information within the network. This involved employing a one-hot encoding scheme for the categories, which was concatenated with the output of the time series model, before the final output layer. CINN achieved an MSE of 0.0058 during the development phase. Despite efforts to improve its performance, we encountered limitations and couldn't conclusively establish the categories' utility for the forecasting task.

Model on detrended and deseasoned time series Enhancing the outcomes of LSTM models can be achieved by meticulously detrending and deseasoning the input time series. Our attempt involved creating a systematic pipeline that sequentially undergoes a detrending step, followed by a deseasoning step, feeding into the neural network only the remaining time series[IMG3].
In the detrending phase, the objective is to eliminate linear prolonged trends within the time series data, redirecting the model's attention towards capturing more immediate patterns. Whereas, the deseasoning step aims to remove recurring patterns or seasonality, enabling the model to focus on intricate temporal dynamics. Following these preprocessing steps, the refined time series data is directed into a Bidirectional LSTM model. We expected this preprocessing step to enhance the model's generalization and forecasting accuracy.
Despite the careful design of the method, the results obtained were not as promising as thought. These limitations may come from the custom implementation of the decomposition pipeline, lacking the utilization of cutting-edge libraries that could potentially offer a more refined decomposition process. Another potential factor contributing to the method's failure might be its universal application across all series. These series may exhibit diverse trend shapes, extending beyond mere first-order trends, and might not present an additional composition of the factors, but maybe a multiplicative one.

N-BEATS Another attempt, which deviates slightly from the classical architectures that deal with sequences of data, has been made by using the N-Beats architecture. This deep neural network is based on backward and forward residual links and a deep stack of fully-connected layers, whose fundamental building block follows a fork architecture to generate the forecast and backcast output. Further details about the network are given in the notebook [IMG4].
We followed the guidelines provided by the authors for deciding the length of the window with which the model has been fed: given a horizon H=18, then the window W has been set to multiples of H, i.e. from 2H up to 7H.
The overall performance of the single model measured in terms of mean squared error was around 0.0065 on the CodaLab test set (1st phase). Traditional regularization techniques, such as incorporating dropout between the dense layers, were not utilized as they did not demonstrate effective performance (as also remarked by the architecture's authors).

Time2vec Time2Vec (T2V) is a technique designed to encode temporal information into vectors, and it can be particularly useful when building LSTMs for time series forecasting. The goal is to have a representation of time in form of vector embedding to automate, within the learning process, the feature engineering process[3]. A 128-dimensional embedding, generated by the T2V building block, was incorporated right after the input layer, followed by the previously mentioned bidirectional architecture [IMG5]. However, minimal improvements were observed despite these modifications.

Contributions
In our initial project phase, the entire group collaborated on data inspection and preprocessing. When it came to model experimentation, Pertino delved into trials with N-Beats and Time2vec. Sandri and Simionato independently constructed custom models using LSTM and convolutional layers, while Sandri also developed the CINN model. Additionally, Simionato focused on exploring detrending and deseasoning techniques for time series data. All individual efforts converged in the final ensemble model, merging diverse approaches and findings.

References
[1] Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. International Conference on Learning Representations. https://www.openreview.net/pdf?id=r1ecqn4YwB

[2] Remy, P. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In GitHub repository. GitHub. https://github.com/philipperemy/n-beats

[3] Cote, D. (2022) *Hands-On Advanced Deep Learning Time Series Forecasting with Tensors*, *Medium*. Available at: https://medium.com/@dave.cote.msc/hands-on-advanced-deep-learning-time-series-forecasting-with-tensors-7facae522f18

[4] Pertino, P., Sandri, A., Simionato, E. (2023). I'm truly euphoric, this is how I found the definitive method to calculate forecast windows. http://tinyurl.com/window-box-jenkins

Image references
[IMG1] Time series plots - Notebook section 1.1
[IMG2] Time series per category plots - Notebook section 1.4
[IMG3] Linear detrending and deseasoning plots - Notebook section 2.6
[IMG4] N-Beats architecture - Notebook section 2.7
[IMG5] Time2Vec architecture - Notebook section 2.8

After each model shown in the notebook, plots regarding the training history, the time series predictions and the performances are shown but are not listed here.