NEURAL PROPHET

A PREPRINT

Alexey Voskoboinikov
Skolkovo Institute of Science and Technology
Moscow, Russia
dblokv@gmail.com

Polina Pilyugina
Skolkovo Institute of Science and Technology
Moscow, Russia
polina.pilyugina@skoltech.ru

April 20, 2021

ABSTRACT

This project aims to contribute to the open-source library NeuralProphet by adding state-of-the-art models and refactoring the code to adopt the best machine learning engineering practices.

***K*eywords** Time series forecasting · Neural Prophet · Neural forecasting

1 Problem Statement

NeuralProphet is a new library for time series forecasting built on PyTorch. It is inspired by widely known Facebook library for time series forecasting called Prophet ([7]) and DeepAR model ([6]). However, while Prophet is an additive model focused mostly on seasonal components and holiday effects, NeuralProphet additionally includes AutoRegression components. Moreover, NeuralProphet is built on PyTorch, which allows to configure the model more precisely.

In this project we aim to improve existing NeuralProphet library to allow even more possibilities for its users. Currently, forecasting model is written on pure PyTorch, it has complicated structure, code is hardly reusable, and this makes experiments with implementation difficult for outside users. We aim to use PyTorch Lightning framework in order to structure the code in more concise way for future research. Moreover, current implementation of NeuralProphet does not support distributed training, while PyTorch Lightning allows to introduce distributed running of the models. Another problem of current NeuralProphet implementation is that it has a rather specific API in terms of initial data format and outputs. This makes comparison with other models difficult, as users are required to write additional code, in order to produce comparable results.

2 Description of the project

2.1 Main goals

Taking into consideration existing drawbacks, we outlined the main goals of our project as follows:

- Refactor the main model class from NeuralProphet with PyTorch Lightning
- Refactor the rest of the code to support PyTorch Lightning in accordance with existing API
- Adapt and include existing implementations of state-of-the-art models for time series forecasting under the NeuralProphet API
- Add hyperparameter tuning with Ray Tune as additional module to NeuralProphet
- Recreate LIBRA framework for benchmarking in Python and run it on NeuralProphet and our additionally included models
- Add neccessary tests and documentation for introduced functional

2.2 Existing solutions

First part of the project is to structure existing code in accordance with PyTorch Lightning framework ([2]). PyTorch lightning is a lightweight PyTorch wrapper that allows to organise the code into separate PyTorch Lighting modules. PyTorch Lightning provides multiple advantages compared to usual PyTorch: models become hardware agnostic and structured, it provides integration with popular machine learning tools, while keeping flexibility of original PyTorch.

PyTorch lightning also provides a robust architecture that will help us to implement four state-of-the art models for time series forecasting: N-Beats ([5]), LSTM ([3]), Temporal Fusion Transformers ([4]) and DeepAR ([6]). This will allow users to compare NeuralProphet with these reference models. We rely on existing implementations, available in PyTorch Forecasting library[1].

Another big part of the project is implementation of additional functional to the NeuralProphet model. It will include hyperparameter optimization and modules for evaluation. For hyperparameter optimization, we will use Ray Tune library, as it has hooks to support PyTorch Lightning. Ray Tune has additional functional for distributed hyperparameter tuning, which allows for additional parallelization and scalability. For benchmarking we will rely on the work on LIBRA framework, described in [1]. LIBRA is an evaluation framework that evaluates and ranks forecasting methods based on their performance.

All these changes to the NeuralProphet should be accompanied by extensive testing and documentation. For tests and integration, NeuralProphet already has modules with unit and integration tests, and we will extend them to cover all of the refactored and new code. As for documentation, we will add necessary documentation for all new functional provided. This will additionally require to add corresponding pages into the doc of NeuralProphet, and add minimal examples as notebooks to the GitHub repository.

3 Main Challenges

3.1 API, architecture and style

One of the main challenges in this project is to maintain NeuralProphet style and API. We should structure the code in a reusable way, so it would be easy for ourselves and other people to contribute to the project in the future. To be more precise, it has inherited from Prophet structure of inputs and outputs. It makes NeuralProphet easily comparable with Prophet, but not with other models. Refactoring in accordance to PyTorch Lightning should not affect existing library consumers. And we aim to introduce new models in NeuralProphet in accordance with its existing API to preserve usability. This will require us to understand in detail the explicit structure of inputs and outputs to each model and reformat them accordingly. Another challenge of refactoring is that we need to implement specific solutions for all NeuralProphet model components, such as autoregression component, seasonality, trend, covariates and events. PyTorch Lightning has some classes implemented, while NeuralProphet has specific components, which are not yet implemented.

As for adding state-of-the-art models, existing PyTorch implementations are also built in PyTorch Lightning framework. However, they do not support NeuralProphet API out of the box, so we will need to refactor them in accordance. Additionally, we will adapt existing modules of NeuralProphet for data preprocessing in order to include preprocessing for new models. We will also add wrappers to process precidtions of these models in the same format as NeuralProphet.

3.2 Adapting LIBRA

The other challenge - implement LIBRA framework [1] for our purposes. It's implementation is available[2] only in R, so we will have to adapt it in Python as a part of the NeuralProphet. Further, we will run the benchmarking framework on 400 time series from [1], which will require a lot of training time. In order to perform such benchmarking, we will need to test our pipeline and its scalability.

4 The roles for the participants

We distributed our main tasks and goals evenly, as described on the 1. Both of us will work on refactoring into PyTorch Lightning. Alexey will focus on the main TimeNet model class, while Polina will work on the forecaster code. We also distributed addition of models, such that Polina will work on N-Beats and LSTM, while Alexey will work on Temporal

[1] `https://pytorch-forecasting.readthedocs.io/en/latest/models.html`

[2] `https://github.com/DescartesResearch/ForecastBenchmark`

Fusion Transformers and DeepAR. We will write corresponding tests and documentation of implemented modules. Further, Polina will focus on hyperparameter tuning addition, while Alexey will implement LIBRA framework in python. Afterwards, we will both work on the benchmarking using LIBRA framework and finalization of the project.

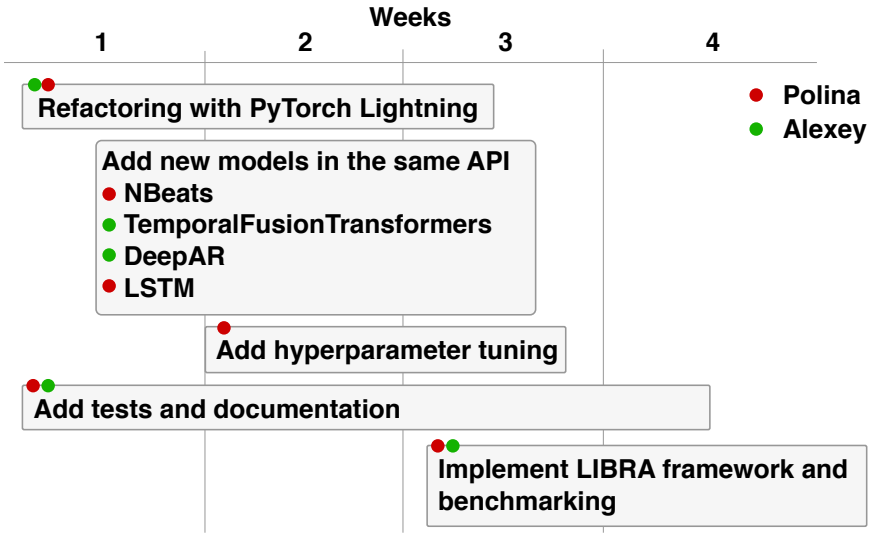

Figure 1: Preliminary Roadmap with Role Distribution

5 A link to the GitHub repository

For this project we forked the original repository and will contribute to it. It will allows us to create a pull request into the master in the future. GitHub repository is available through this link: `https://github.com/OldMindFlayer/neural_prophet`.

References

[1] André Bauer et al. "Libra : A Benchmark for Time Series Forecasting Methods Libra : A Benchmark for Time Series Forecasting Methods". In: April (2021).

[2] WA Falcon and .al. "PyTorch Lightning". In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[3] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

[4] Bryan Lim et al. "Temporal fusion transformers for interpretable multi-horizon time series forecasting". In: *arXiv* Bryan Lim (2019), pp. 1–27. arXiv: 1912.09363.

[5] Boris N. Oreshkin et al. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting". In: *arXiv* (2019), pp. 1–31. ISSN: 23318422. arXiv: 1905.10437.

[6] David Salinas et al. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *International Journal of Forecasting* 36.3 (2020), pp. 1181–1191. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2019.07.001. arXiv: 1704.04110. URL: https://doi.org/10.1016/j.ijforecast.2019.07.001.

[7] Sean J. Taylor and Benjamin Letham. "Forecasting at Scale". In: *American Statistician* 72.1 (2018), pp. 37–45. ISSN: 15372731. DOI: 10.1080/00031305.2017.1380080.