Artifical Neural Networks and Deep Learning
Homework 2:

Axel Friberg Hagen
Vegard Sjåvik
Lasse Uttian

December 22, 2023

1 Foundational Overview: Introduction and Motivation

In this project, we explore the task of univariate time-series forecasting. The goal of the project was to forecast 9 (18 in final phase) future timesteps, based on the previous 200 timesteps.

In the following sections, we will discuss our methodology, including data preprocessing, model architecture, training process, and evaluation results.

2 Data preprocessing

The first step after loading the data is to take a look to get a better overview of outliers, potential biases, or mistakes in the loading of the data. We visualized some random sample timeseries from each of the different categories which was provided in the dataset (A-F). Looking at the samples there seemed to be no significant similarity between the timeseries within a category.

We found a lot of strange timeseries in our exploration, but decided not to modify or remove them as this could introduce a bias in the training set and we didn't know how the hidden test set looked like. Domain knowledge would be useful to know which series are "trash" and which are realistic, but in this project we have zero domain knowledge, so cleaning data is more prone to introduce bias.

In order to effectively utilize the data, we needed to preprocess it and create sequences which would fit our models. Before defining these sequences, a random sample of 10% of the series was removed to be used as a test set. Because the timeseries in the competition test-set had a length of 200, with the goal of predicting 9 (18 for last phase of competition) we found it natural to choose a window of 200, and a telescope of 9 (18).

The sequences was found using a rolling window method with a stride of 100, chosen arbitrarily as a good trade-off of getting enough data but without the risk of overfitting. The timeseries with a shorter length than window size + telescope was padded with zeros in the beginning of the timeseries, with the hope that the model would learn that this is irrelevant data.

3 Evolution of Ideas: Experimental Insights

This part of the report is going to address the different experimentations that were done to figure out which models to use for the forecasting task.

3.1 Flat predictor

A baseline model can be used for comparison. We included a model that just predicts the last output of the training sequence for all of the test-sequence. In many applications, there may be extremely complex or very little temporal dependencies. In these cases, a simple, flat predictor might outperform even the most complex and optimized deep learning architectures, as the true trend of the data is random with respect to previous data [2]. Table 1 shows a comparison between the MSE of models developed and the baseline on a test set.

3.2 Further scaling of data

We tried row-wise z-scaling the data for improved performance but it worsened the performance 10-fold. We then went on to try robust-scaling (more robust to outliers), but the result persisted. This could be due to a loss of information in the data when scaling, as some temporal features might be lost or minimized after *squashing*.

3.3 Utilizing the given categories

We tried creating a dataset for each of the given categories, and making a different model for each one. In inference time, we would choose which model to use based on the given category. This didn't yield any significant upgrade however, and was thus not pursued any further as training time took long time after running out of compute on Colab.

3.4 Model architecture

Base model - A simple LSTM network: The first model developed was a basic and shallow LSTM network, with just a LSTM layer with 200 neurons and a dense layer following to adjust the output size. To avoid overfitting, a simple model was desired. However, Figure 1 shows the model quickly converging and flattening. This can indicate underfitting, and a more complex model might be needed to better capture the temporal dependencies [1].

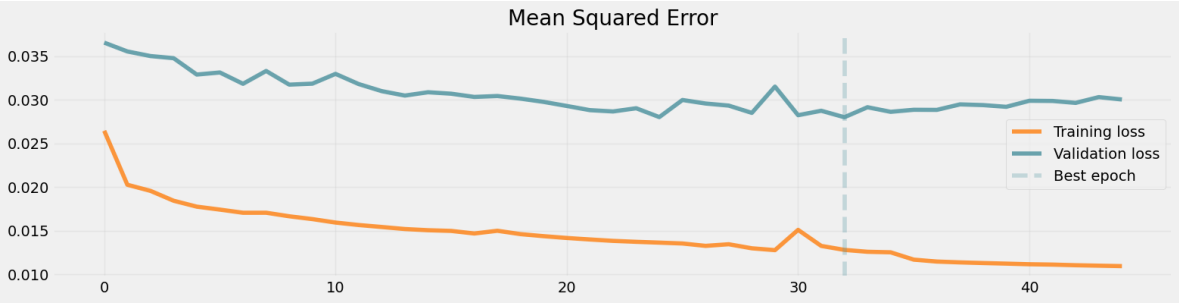

Figure 1: Training of simple LSTM network shows quick convergence.

Model v2: Added two 1d convolutional layers before the LSTM layer to hopefully learn defining features of the timeseries.

Model v3: Added dropout and batch-normalization for the convolutional layers to regularize and

prevent overfitting.

Model v4: Replace the LSTM layer with two bidirectional LSTMs, with the hope to capture more complex temporal dependencies.

Model	Mean Squared Error
Baseline flat predictor	0.0300
Basic LSTM	0.0144
Z-scaled model v4	0.455
Robust scaled model v4	0.386
Model v3	0.0142
Model v4	0.0140

Table 1: Performance Metrics of Different Models on the test set, when forecasting 18 timesteps ahead (Phase 2)

4 The Culminating Model: Final Design and Structure

Increasing the model complexity reduced training error a lot, but did not yield significant improvement on the validation-set. This indicated overfitting, and thus the more simple architecture may be preferred. In the end, it seems the model-complexity was not the most defining, highlighting the importance of data-preprocessing, model-archtecture (introducing mechanisms like recurrence, LSTM or self-attention) and segmenting data beforehand.

For training we always included early stopping with a patience of 12, as well as a linear learningrate-rescheduler with a patience of 10. We then evaluated the models against the test set (more of a validation set in this case, as the *test* is in the competition backend). For the final evaluation we merged the test and training set into one dataset which we trained the most promising model (model v4) on.

5 Future Frontiers: How model performance might be enhanced

This section discusses methods we considered, but didn't fully explore due to time-constraints implementing, testing or running the models.

- Different ways of scaling the data that better keep temporal dependencies might be effective, such as a window-wise scaling.
- The emerging self-attention mechanism used in transformer models have proved very effective in these type of problems where deciding what information is correlated in long sequences of data is important. We tried implementing it, but encountered a lot of runtime errors. We decided it wasn't worth our time to fix it and left it for future work.
- Extensive data-cleaning, such as removing noisy or short series or series with seemingly missing data. As discussed in section (2).
- Different solutions for handling shorter series other than padding with zeros. Such as repeating the signal (which would potentially introduce a fake seasonal bias), masking the missing values, imputing the missing data with mean/median or regressive models, or simply removing the sequences that were too short.

6 Contributions

Contributors

Axel Hagen:

- Experimented with scaling/normalizing data.
- Experimented with different model-architectures and hyperparameters.
- Provided insights to the report.
- Tried implementing self-attention mechanism.
- Data exploration and contemplated how to clean data.

Lasse Uttian:

- Fixed bugs in Axel's scaling/normalizing code.
- Experimented with different models

Vegard Sjåvik:

- Implemented all the code for building the datasets used for training and testing, including cutting the sequences and padding them.
- Preliminary data exploration, building the way for further understanding and insights for rest of team.
- Explored the possibility of building a model for each category and choosing which model to use in inference time.
- Created basic LSTM model which was iterated on by the rest of the team.

References

[1] IBM. Underfitting, 2023.

[2] Aashish Nair. Baseline models: Your guide for model building, 2022.