Artificial Neural Network and Deep Learning Challenge 2 Report

Rishabh Tiwari [Icon1c], 10987397, rishabh.tiwari@mail.polimi.it
Erik Gustav Malmsten [gmalmsten], 10992793, erikgustav.malmsten@mail.polimi.it
Nicolas Filip Johansson [nicolasj], 11005901, nicolasfilip.johansson@polimi.it
Sophie-Claire Antoun [sophieantoun], 10989152, sophieclaire.antoun@mail.polimi.it

M.Sc. Computer Science and Engineering
Team Name: The Avengers

1 Introduction

In this project, we, as a team, focus on using what we've learned to predict future events in different time series. Our main goal is to build a model that can look at past data and make accurate predictions about what comes next. We're working with a unique set of data that includes different kinds of time series, and our challenge is to make a model that generalises to all of them. This report will share our journey, the steps we took, and how we solved problems along the way.

At the outset of our project in Time Series Forecasting(1) for the "Artificial Neural Networks and Deep Learning 2023" course, our primary goal was to develop a basic model using Gated Reccurent Unit (GRU) networks, a type of Recurrent Neural Network (RNN) well-suited for time series data (4). These GRU models are adept at capturing temporal dependencies and patterns in time series while also being faster and more efficient to compute than LSTM. Our initial GRU model consisted of single GRU layer, designed to process and remember information over shorter periods. Despite its promising structure, which utilized features like dropout layers to prevent overfitting, we encountered challenges in achieving the desired level of accuracy and generalization with our dataset.

Confronted with these challenges, we pivoted our focus towards exploring more complex models, particularly involving the Long Short-Term Memory (LSTM) unit which allows processing longer sequences than GRU. We delved into various LSTM configurations, experimenting with different numbers of layers, units and even convolutions to optimize our model's performance. This exploration into LSTM models enabled us to better capture the complex long-time dependent patterns in our time series data, striving for a balance between model complexity and the ability to generalize across different time series scenarios.

2 Data Pipeline

We start by loading our data from a specific folder using np.load. Then, we make sure to keep only the good data by using 'valid_periods' as a guide. We change the data into sequences that our model can learn from. This means we take a bunch of data points to predict the next few points in the future. We also shuffle our data to keep the model from learning the order of the data instead of the patterns.

After shuffling, we split our data into two groups: one to train the model and the other to check how well it's learning. We're careful to make sure that the same data isn't in both groups. We decide how much data to use for checking with the val_size setting.

2.1 Data reading

In analyzing our dataset, we found that it's grouped into six categories, with a notable imbalance in their sizes. Categories B, C, D, and E are well-represented, each having over 10,000 time series, but Category A has about half that number, and Category F has significantly fewer, only 277. This uneven distribution is important for us because it means some types of data are much more common than others. When we're training our model, we need to keep this in mind

so that it doesn't get too focused on the more common categories and ignores the less common ones.
Our deep dive into the 'valid_periods.npy' and 'training_data.npy' files gave us more insights. The 'valid_periods.npy' file showed that our data, which includes 48,000 time series, varies in length from 24 to 2,776 data points. This tells us that our model needs to handle different lengths of data effectively. The 'training_data.npy' file, with its uniform data type and instances of zero-padding (empty data points), highlighted the need for our model to distinguish between important and unimportant parts of the data. So, as we work on our model, we're focusing on making sure it can understand and learn from the patterns in the data, regardless of how long each series is or how much padding it has.

2.2 Scalars and Normalization

To prepare our data for the model, we use a tool called RobustScaler from scikit-learn. This helps our model by making sure it doesn't get confused by a few weird data points that are not like the rest. We only fit this tool with our training data, so it knows what typical data should look like. After it learns this, we use it to fix our training and validation data, making sure everything is consistent and ready for the model to learn from.

We also talk about the process of normalization, which is when we adjust the data to make sure it's all on a similar scale. This helps the model learn better because it means the model doesn't get thrown off by some data being much bigger or smaller than the rest. Scalars are the tools we use for normalization, like the RobustScaler. They change the data to a scale that works best for the model. We keep these scalars by saving them with joblib.dump, which means we can use them again for new data and keep everything nice and tidy. This step makes sure that when we tell the model to look at new data, it treats it the same way it did with the data it learned from.

3 Model Development

In our project, we worked on a special model to understand and predict patterns(2) from a mix of different types of data, as we saw in our 'categories.npy' file. We noticed that our data came in various lengths and had different kinds of patterns, which we saw in the 'valid_periods.npy' and 'training_data.npy' files. Our main job was to adjust our model to work well with all these differences. We made sure it could handle data that was both long and short, and we fixed it so that all the data was measured in the same way. Our goal was to make a model that could accurately predict things based on our specific data, which was quite varied and unique.

3.1 Feature Extraction Network

In Phase One, our models extracted features predominantly using GRU and LSTM layers(3). The simplistic architecture of the GRU-v3 model with a single GRU layer effectively captured time-dependent information from the series. We introduced bidirectional LSTM layers and CNNs for deeper feature extraction as we went to more complicated models like LSTM-v1 and LSTM-v3. The bidirectional nature of LSTM-v1 and the convolutional layers in LSTM-v3 enabled a more nuanced comprehension of time series patterns, which was critical for dealing with the various lengths and characteristics discovered in our dataset study.

3.2 Classifier

Our models' classifiers were trained to make predictions based on the features extracted by the GRU and LSTM layers. The classifier in early models, such as GRU-v3, was a basic dense layer. We experimented with different classifier setups as our models grew. For example, in LSTM-v4, the classifier was intended to comprehend normalized time series data, ensuring accurate predictions even when patterns and lengths vary. Our categorization strategy took into account the necessity to properly handle various categories, particularly those underrepresented in the dataset.

3.3 Model Building

When we build our model, we use TensorFlow's Keras API to make an LSTM network. This kind of network is good at understanding data that changes over time, like the data we're using. We add layers to our model that look at the data forwards and backwards, which helps it learn better. We also use some tricks to stop our model from learning the data by heart, which could make it bad at predicting new data. These tricks include things like L2 regularization, batch normalization and dropout.

The final layer in our model is a dense layer that gives us the predictions for future data points. We use an optimizer called Adam and a way to measure errors called mean squared error. This setup is good for predicting numbers, which is what we're doing.

We also use something called early stopping. This stops the training if the model isn't getting better at predicting the checking data. It saves time and stops the model from just memorizing the training data. After training, we check how good our model is using the checking data, and then we save the model so we can use it again easily.

3.4 Training process

We used tactics to optimize our models' performance against the unique challenges of our dataset throughout the training phase. In GRU-v2 and v3, for example, we concentrated on generalizing the models to fresh, uncorrelated time series by shuffling data and drawing independent samples. The strategy of LSTM-v2 of training separate models for each category was developed in response to the class imbalance and variation in time series lengths. Additional normalization steps were used in the training of LSTM-v4 using robust scalers, exhibiting our iterative approach to improving model correctness. The LSTM-v4 yielded the best results on the hidden test of phase one, and its training loss is plotted in figures 1 and 2. The values shown are the scaled MSE, and the best validation epoch corresponds to a real (inverse scaled) MSE of 0.005323.

We extended our models' capabilities in Phase Two to forecast further into the future, a job that required careful management of longer sequence data for effective forecasting. We first tried to retrain the LSTM-v4 with a forecast length of 18, however with only decent results and a hidden test error far greater than the validation. The better approach was to simply reuse our best model from phase one; predicting nine samples, concatenating to the input time series, and then predict the final nine samples.

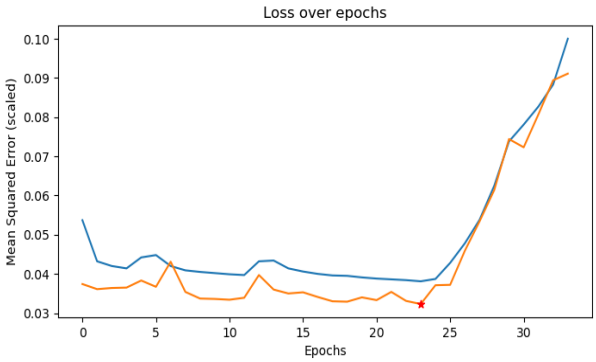

Figure 1: Loss over time

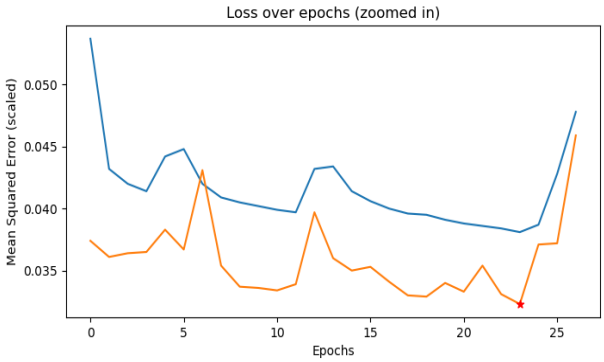

Figure 2: Loss over time, zoomed in

4 Hyperparameter optimisation

In the realm of deep learning, hyperparameter optimization plays a important role in enhancing the predictive performance of models, and time series forecasting involves several optimizeable parameters. It involves fine-tuning parameters such as sequnece length, forecasting length, batch sizes, and architectural configurations such as the amount of LSTM units to achieve optimal results. However, one significant challenge we encountered in our project was the prohibitively long training times associated with our deep learning models. The large datasets and extensive computational requirements made conventional hyperparameter optimization methods impractical within the constraints of our project timeline.

To address this limitation, we adopted an iterative approach, experimenting with various hyperparameter settings during the development phase. By systematically adjusting parameters and closely monitoring their impact on model performance, we sought to strike a balance between computational efficiency and forecasting accuracy. This allowed us to navigate the challenges posed by extended training times and iteratively refine our models for optimal performance.

Contributions

- Rishabh Tiwari - Creating and running the tests on LSTM and GRU models. Experimented with different data scaling techniques.
- Nicolas Johansson - Developed the GRU-vx and LSTM-vx models; experimented with architectures of varying complexity, including convolutional LSTM.
- Sophie Antoun - Creating and running the tests on several LSTM and GRU model. Experimented with 1D fully convolutional networks.
- Gustav Malmsten - Creating and running tests on LSTM, GRU, autoregressive and ensemble of models.

References

[1] Jason Brownlee. Deep learning for time series forecasting: Predict the future with mlps, cnns and lstms in python. *Machine Learning Mastery*, 2018. Practical approaches to handling time series data and developing forecasting models using deep learning techniques.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Comprehensive guide to deep learning, covering aspects of model development and neural network architectures.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural Computation*, volume 9, pages 1735–1780. MIT Press, 1997. Foundational paper on LSTM networks, discussing their structure, properties, and relevance to handling time series data.

[4] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. 2015.