Windpower Forecasting with Machine Learning

-

Felix Bötte - 4107268

1 Forecasting Wind Power with Sequence-2-Sequence Neural Networks

In this report, I present my findings from a wind power forecasting task that focuses on leveraging recurrent neural networks, specifically LSTM (Long Short-Term Memory) encoder-decoder networks, to capture temporal dynamics within time series data. The objective is to accurately predict the power output of at least one turbine within each wind farm for the next time step, hour, and day, utilizing data from British and Brazilian wind farms. To achieve this, I employed feature selection techniques to identify the most critical variables.
My main emphasis was on harnessing the capabilities of recurrent neural networks to capture temporal dependencies and patterns in time series data. To optimize the performance of the forecasting model, I initiated the process with various data preparation techniques, including normalization, handling missing values, and outlier management to make the data suitable for training. Subsequently, I assessed variable importance through automated approaches such as Pearson and Spearman correlation, time-shifted correlation analysis, XGBoost, and L1 regularization, with the goal of identifying the most influential factors. Finally, I conducted training and evaluation on a manually curated dataset comprising 10 selected columns in addition to the target column, resulting in an 11-dimensional dataset spanning one year for a single wind turbine.

1.1 Code

The discussion below references my code hosted on GitHub at GitHub. Within the repository, the **data_inspection.py** directory is employed to gain insights into the dataset's individual variables and their dimensions. The **requirements.txt** file lists all the necessary packages that must be installed to execute the code.

Furthermore, the Jupyter Notebook **windfarm_britain_data_preprocessing.ipynb** plays a pivotal role by reading the raw data of a single turbine from the British wind farm dataset into a pandas dataframe. It also imports the **data_preprocessing.py** file, which houses a collection of functions tailored for data processing tasks. Tthe data is divided into two subsets: the target data, encompassing the primary variable of interest **Power (kW)** for Britain and **active_power_total** for Brazil), and the input data, which includes selected additional variables. Following this division, both datasets undergo normalization, and any missing values (NaN) are systematically addressed and removed.
In the following step, the user is presented with the option to select a feature selection mechanism ("pearson,spearman,xgboost,time_lag_corr") along with the desired number of most important features to be retained. Notably, "time_lag_corrïnvolves analyzing the correlation between variables when the target dataset is shifted by a specified number of

timesteps. Subsequently, the notebook generates and stores a new CSV file within the "preprocessed_data"directory, containing only the data corresponding to the identified most important columns as determined by the chosen feature selection algorithm. The parallel notebook, labeled **windfarm_brazil_data_preprocessing.ipynb**, replicates the aforementioned processes but reads data from an NC file instead of a CSV file.

Within the **lstm_training.py** file, the newly created CSV file containing variables with the highest correlation for predicting the target variable is reloaded as a pandas dataframe and subjected to further normalization to achieve a mean of 0 and a standard deviation of 1. Following this, the data is partitioned into training and test sets, and the input window and output window for time series prediction are specified. Each timestep in the data corresponds to a 10-minute interval, meaning that if the input and output windows are both set to 6, the model predicts the target variable one hour into the future while considering the previous one hour of data. A function is employed to structure the data into the appropriate time series splits used for training the neural network model.
Next, the hyperparameters for the model are defined, and the model is initialized accordingly. The training loop commences based on these hyperparameters. Notably, an option for teacher-forcing is introduced in the training loop to assist the model in learning temporal dependencies, occasionally providing it with target data during extended prediction horizons. Upon completing the training, the hyperparameters, model state dictionary, optimizer state dictionary, and training/test loss information are saved to files for subsequent evaluation and plotting.

The **LSTM_enc_dec.py** file encompasses the architecture of the neural network, including the associated training loop. Additionally, it offers a function for preparing input data for compatibility with time series prediction.

The **utility_functions.py** file includes two minor functions for loading text as JSON and appending two pandas dataframes.

Moving on to the **plot_saved_model.py** file, it serves the purpose of loading a saved model and preprocessed data, subsequently plotting the time series of the first variable (representing the target variable) for both training and test data. Furthermore, it generates two plots illustrating the convergence of training and test loss over a specified range of epochs. Importantly, this file relies on **plots.py**, which houses individual functions for the respective plots.

The **testing_britain.py** file has a distinct role; it loads a saved model and a preprocessed dataset that differs from the dataset on which the model was originally trained. It evaluates the pre-trained model across a range of 144 prediction horizons, assessing the model's performance on varying output lengths. The resulting mean loss is plotted to illustrate its performance across these diverse horizons.
The file **testing_combined.py** is designed for the simultaneous loading and evaluation of multiple models over the same range of 144 prediction horizons. The models employed in this work include vanilla LSTM, encoder-decoder LSTM with and without input and teacher forcing, and a Gated Recurrent Unit (GRU).

The **transfer_learning.ipynb** file engages in the process of loading a saved model and preprocessed dataset and continues training the model on a previously unseen dataset.

This approach leverages transfer learning to enhance the model's generalization capabilities.

Lastly, the **global_model_approach.ipynb** file appends three preprocessed datasets into a single, larger dataset. This consolidation aims to augment data variability, potentially enhancing the model's performance on new, unseen datasets.

2 Results

In the subsequent section, I will provide an overview of the results obtained from my selected models for forecasting wind power within intervals of the next 10 minutes, 1 hour, and 1 day. While I conducted experiments to fine-tune hyperparameters, the lack of substantial differences in outcomes leads me to omit presenting the ablation studies. It appears that the prediction task does not necessitate a highly complex model. The results are predicated on the following hyperparameters:

```
config = {
    "wandb": True,
    "name": name,
    "num_features": 11,
    "hidden_size": 256,
    "dropout": 0,
    "weight_decay": 0,
    "input_window": windows[0][0],
    "output_window": windows[0][1],
    "learning_rate": 0.0005,
    "num_layers": 1,
    "num_epochs": 40,
    "batch_size": 256,
    "train_data_len": 50000,
    "training_prediction": "recursive",
    "loss_type": "RMSE",
    "model_label": model_label,
    "teacher_forcing_ratio": 0.5,
    "dynamic_tf": True,
    "shuffle": True,
```

Abbildung 1: Hyperparameters of neural network model.

The following results are assessed across an incrementally expanding prediction horizon, where each model undergoes evaluation on a segment of the training dataset not previously encountered during training or evaluation. For each horizon duration, the mean RMSE-test-score is calculated and depicted individually, facilitating the comparison of each model's performance across the entire prediction range spanning from 10 minutes to 24 hours. It's important to note that each timestep corresponds to a 10-minute interval.

2.1 Britain Prediction Results

Figure 2 shows four different models, each with the same input length as output length respective to the forecasting times specified in the task: 10 minutes, 1 hour, 24 hours. It becomes apparent that the model with an input/output length of 6 performs the best until a prediction horizon of around 24. Afterwards the model with an input/output length of 144 outperforms every other combination significantly until the the 24 ($144 \cdot 10 minutes$) hour mark. Figure 3 shows there is no much benefit to further increase the input/ouput length.

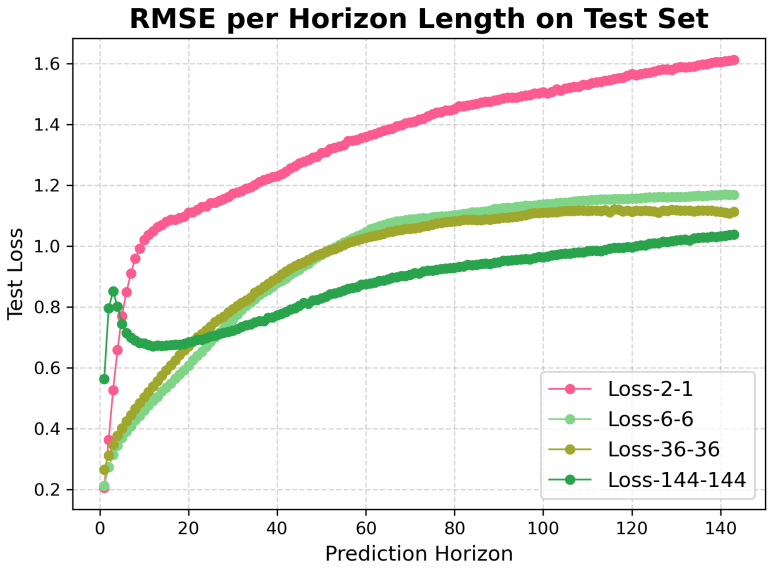

Abbildung 2: Encoder-Decoder LSTM with different input/output-window length.

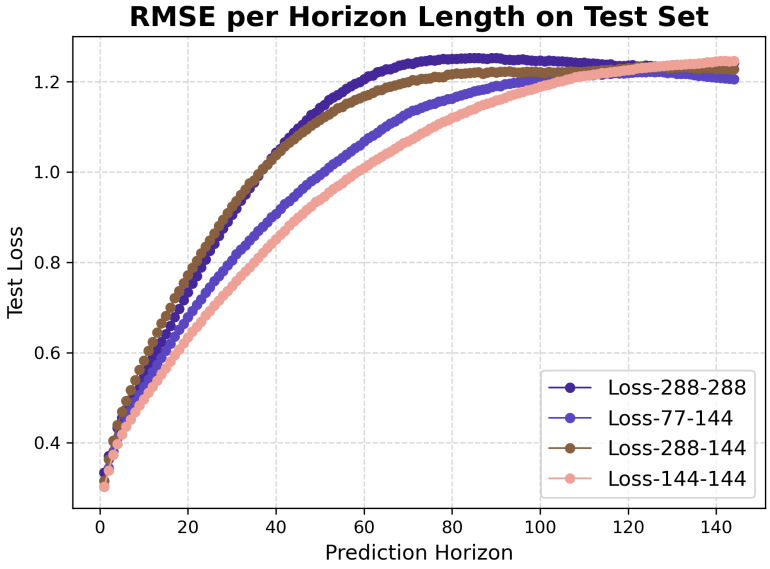

Abbildung 3: Encoder-Decoder LSTM with different input/output-window length.

In Figure 4, I conducted a comparative analysis of various neural network architectures to determine whether any of them excel in capturing the underlying temporal dynamics of the data. The models under examination included a vanilla LSTM, an encoder-decoder LSTM with only hidden states as input, an encoder-decoder LSTM with both the last actual state and the hidden states of the encoder as input, an encoder-decoder LSTM with input and teacher forcing, where the real target sequence is occasionally incorporated into the prediction sequence to enhance longer prediction horizons, and an encoder-decoder GRU. All models where trained with an input/output length of 144.
The results notably reveal that the encoder-decoder LSTM with input surpasses all other models, particularly in later lead times.

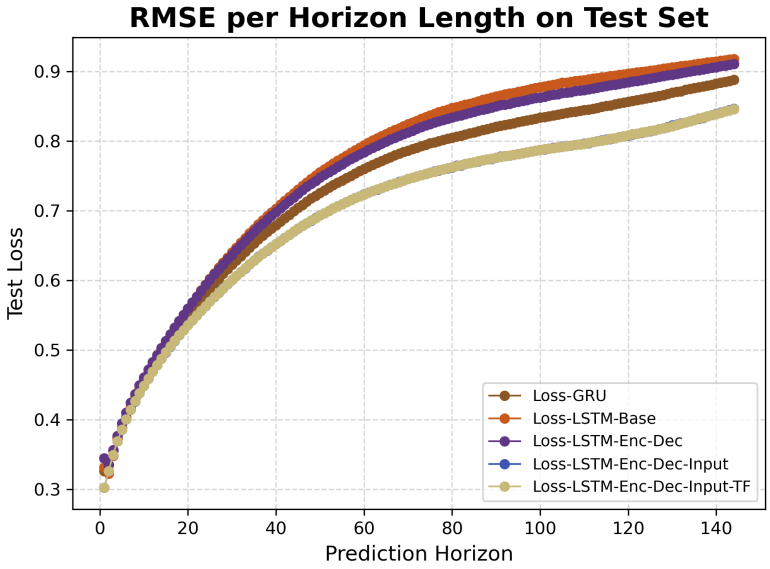

Abbildung 4: Comparison of different neural network architectures. All trained with an input/output-length of 144.

In Figure 5 I was additionally employing a global model approach where I concatenated two years of preprocessed data in order to increase the amount of data for training and consequently increase the variability. I chose the same wind turbine over a timespan of 2019-2021. The results show that there is an overall improvement in performance when using more data for training which makes sense since the model can adjust to more complex situations during prediction.

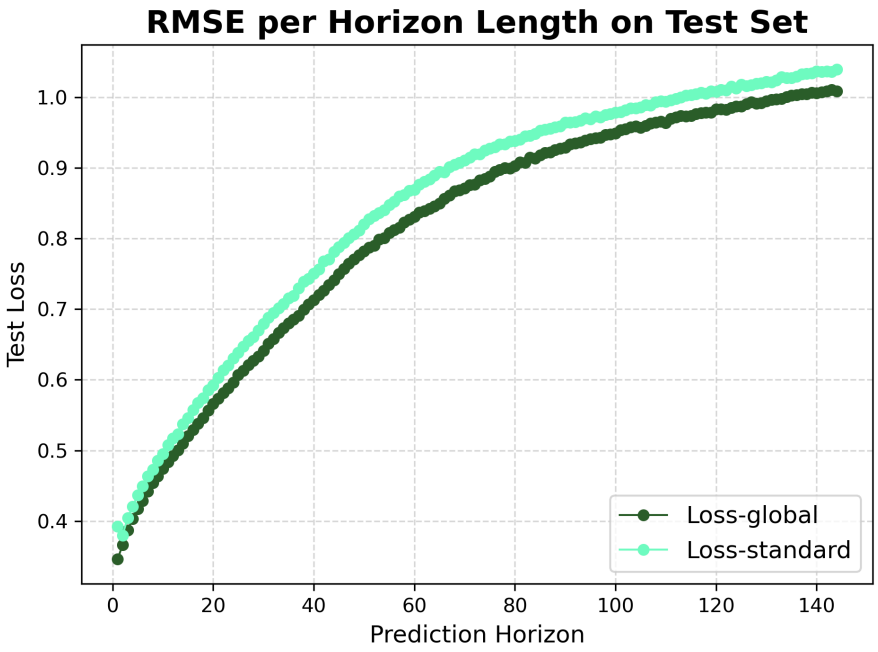

Abbildung 5: Comparison of amount of data used during training. All trained with an input/output-length of 144.

Lastly, Figure 6 shows one encoder-decoder LSTM model trained with input/output length of 144 without normalizing the test loss for benchmark comparison. Additionally the test loss was calculated only for the target variable **Power (kW)** and not over the 11-dimensional tensor. It shows better performance than the benchmark provided.

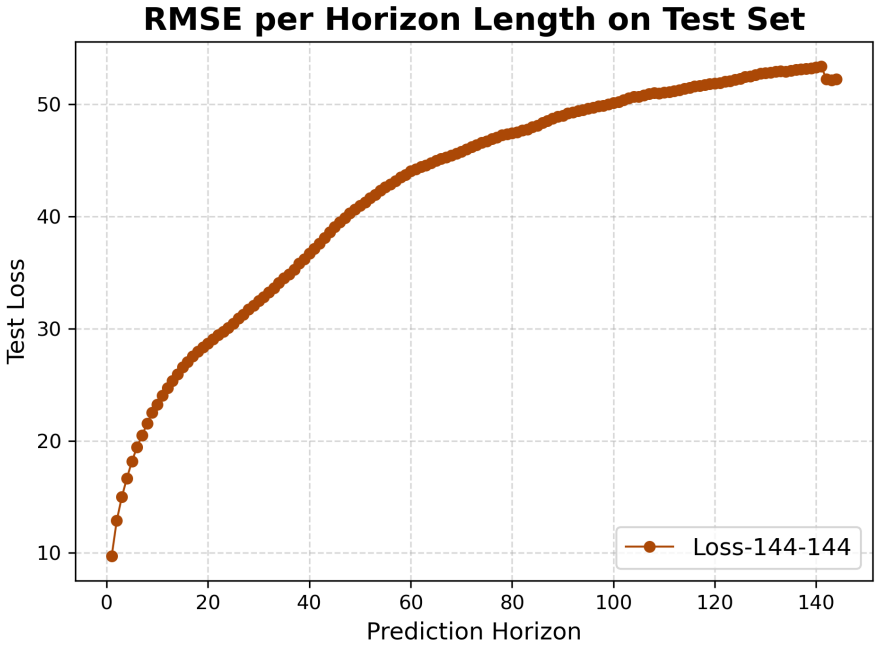

Abbildung 6: Encoder-decoder LSTM with an input/output-length of 144 without loss normalization.

2.2 Brazil Prediction Results

Figure 7 shows again four different models, each with the same input length as output length respective to the forecasting times specified in the task: 10 minutes, 1 hour, 24 hours. It becomes apparent that the model with an input/output length of 6 performs the best until a prediction horizon of around 24. Afterwards the model with an input/output length of 144 outperforms every other combination significantly until the the 24 ($144 \cdot 10 minutes$) hour mark.

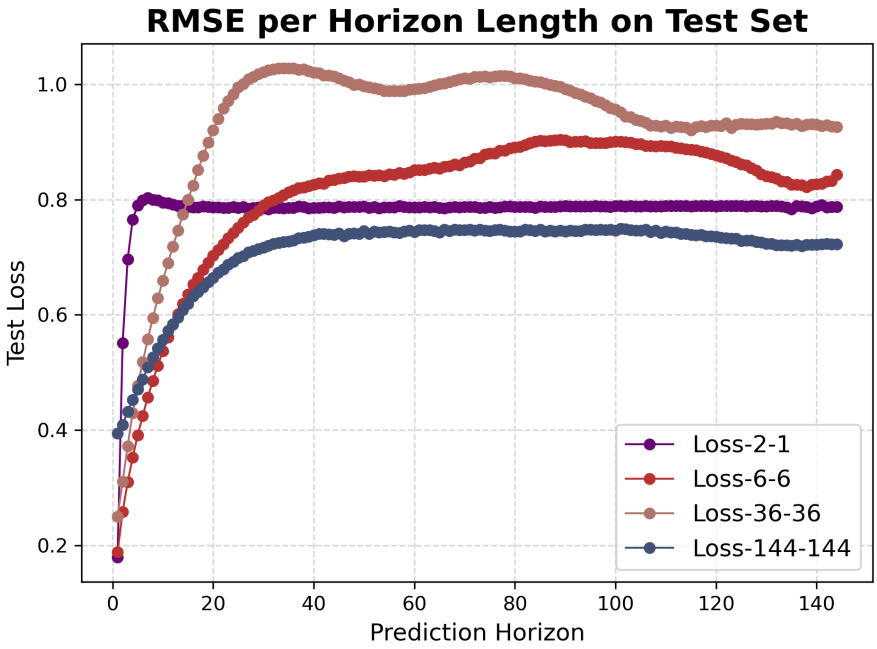

Abbildung 7: Encoder-Decoder LSTM with different input/output-window length.

In Figure 8, I again compared the different architecture types.
For the brazil dataset it becomes apparent that the GRU encoder decoder variant outperforms all other models, particularly in later lead times. It seems to capture the underlying temporal dependencies of the data better than LSTM models which aligns with the idea that GRUs capture longer sequences better than LSTM models.

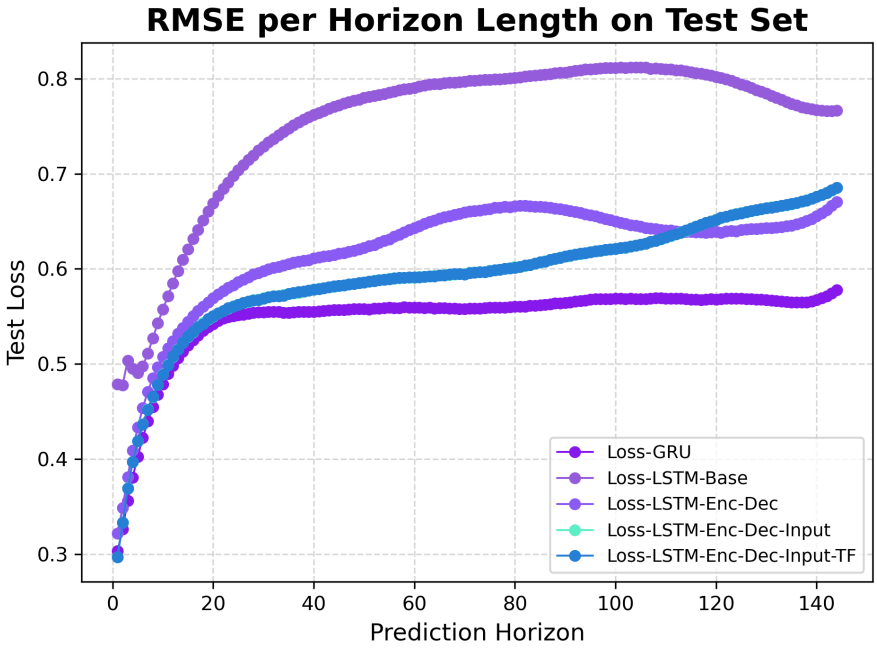

Abbildung 8: Comparison of different neural network architectures. All trained with an input/output-length of 144.

3 Appendix

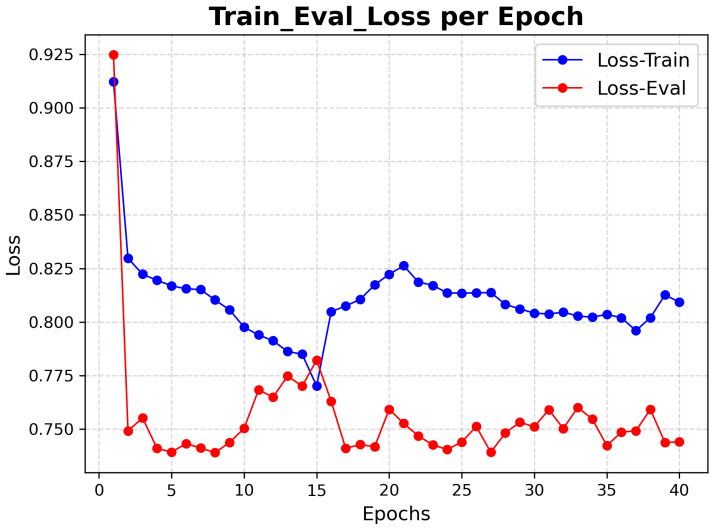

Abbildung 9: Plotting train and eval loss of 144-144 input/output length model (britain data; the respective on for brazil looked similar)

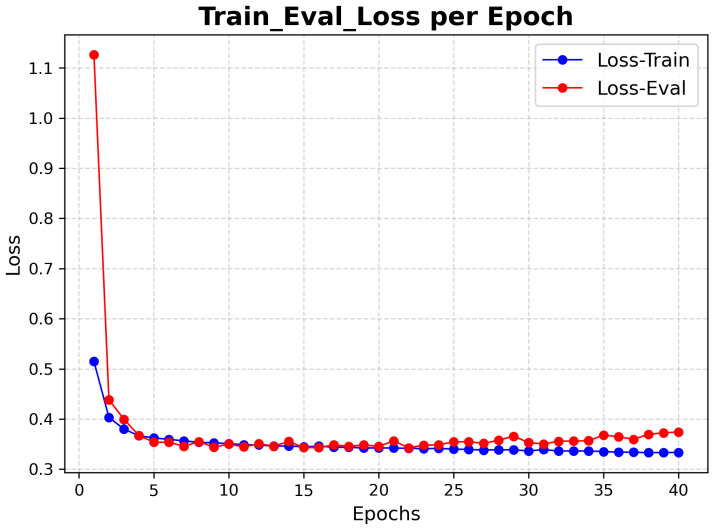

Abbildung 10: Plotting train and eval loss of 36-36 input/output length model (britain data; the respective one for brazil looked similar)

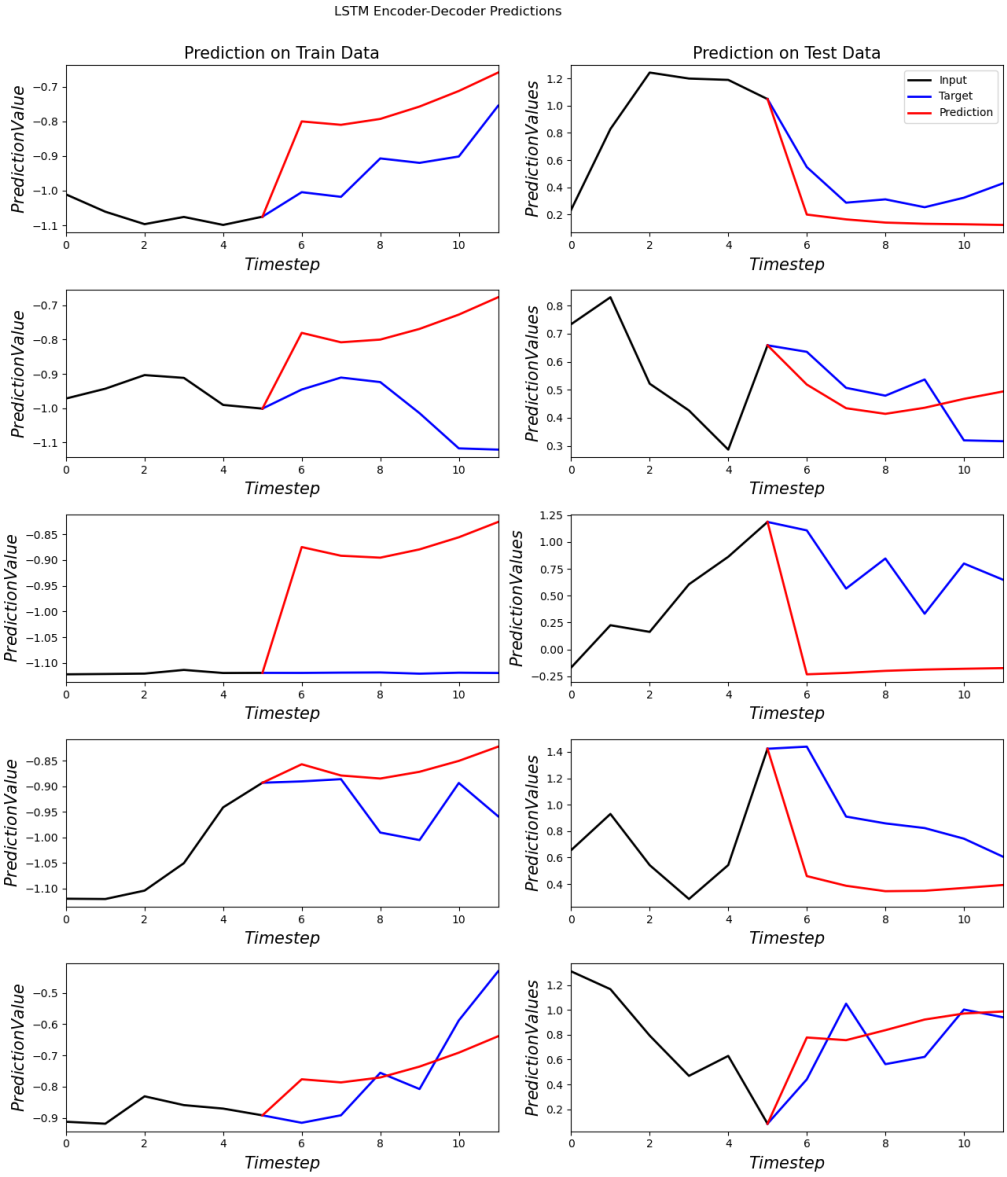

Abbildung 11: Plotting of 5 different time series predictions of the target variable over an output length of 6 (britain data)

0.48

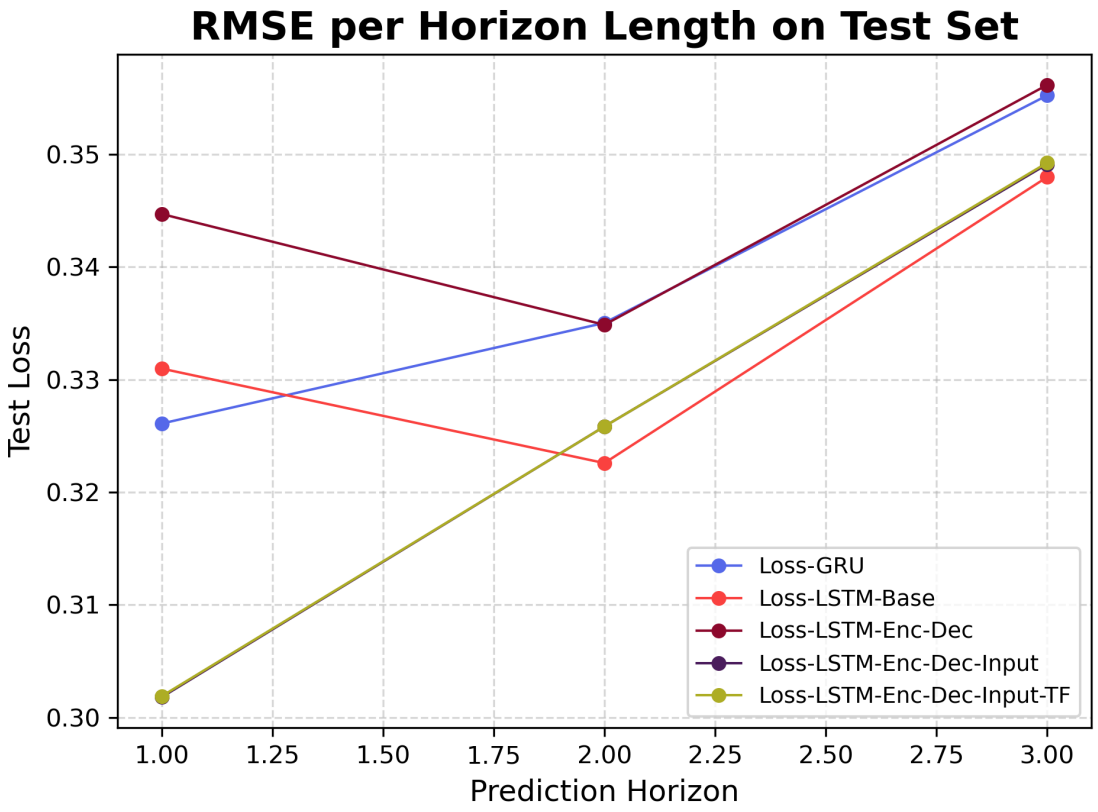

Abbildung 12: Comparison of different neural network architectures. All trained with an input/output-length of 144. Specified horizon of [1, 2, 3] (britain data)

0.48

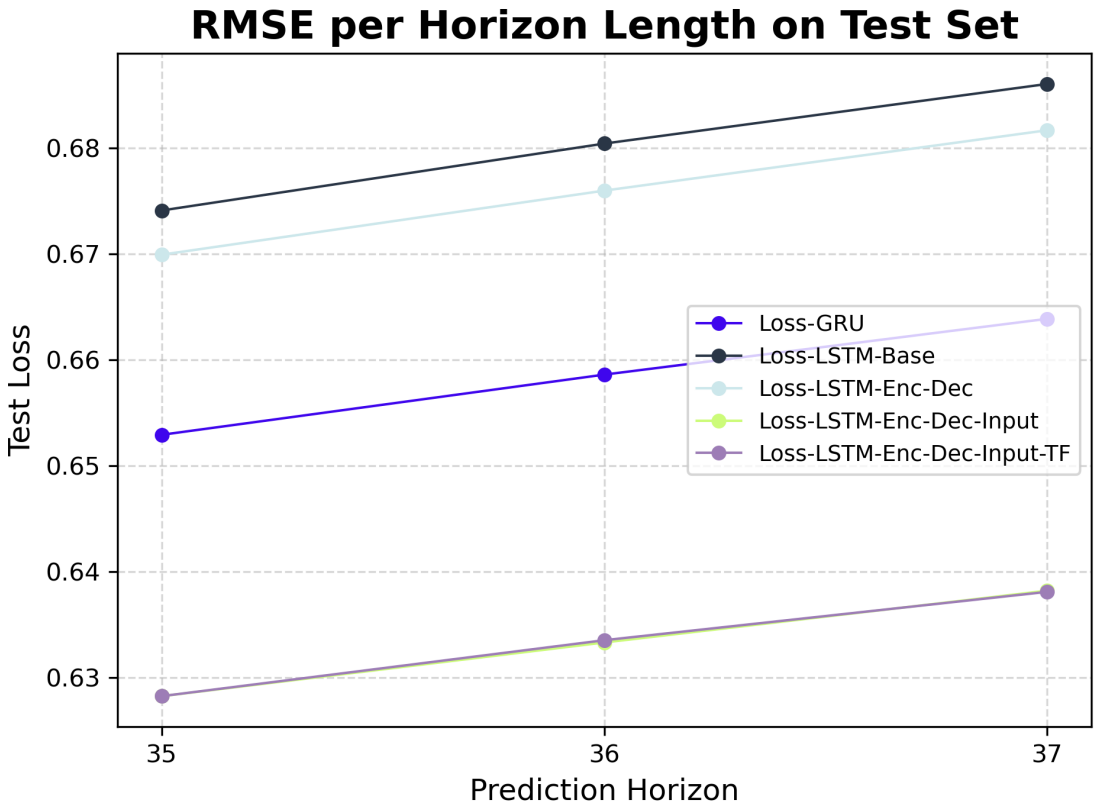

Abbildung 13: Comparison of different neural network architectures. All trained with an input/output-length of 144. Specified horizon of [35, 36, 37] (britain data)