Artificial Intelligence in Computer Systems

Predicting energy consumption in households using time series

Authors:
- Szymon Budziak,
- Kacper Kłusek,
- Jakub Kosmydel,
- Michał Trzebuniak,
- Bartłomiej Wajdzik.

Kraków, 2024

Contents

1 Introduction **4**
- 1.1 Main goal 4
- 1.2 Motivation 4
- 1.3 Dataset Description 4

2 Related Work **6**
- 2.1 Time-series data analysis using LSTM 6

3 Plan **8**
- 3.1 Design Phase 8
- 3.2 Implementation Phase 8
- 3.3 Testing Phase 8

4 Energy Consumption Prediction **9**
- 4.1 Programming Language 9
- 4.2 Models used 9
- 4.3 Method of comparing models 10
- 4.4 Preprocessing 10
- 4.5 Model training with caching 11
- 4.6 Plotting Helper Function 12
- 4.7 Baseline model (Naive Seasonal) 12
- 4.8 Results comparison 13
- 4.9 LSTM Model 13
  - 4.9.1 Data cleaning and exploration 13
  - 4.9.2 Normalizing the data 14
  - 4.9.3 Train test split 14
  - 4.9.4 Model architecture 15
  - 4.9.5 Prediction 15
  - 4.9.6 Hyperparameter tunning 16
  - 4.9.7 Tuned LSTM architecture 16
  - 4.9.8 Tuned LSTM prediction 17
- 4.10 More complex LSTM model architecture 18
  - 4.10.1 Complex and tuned LSTM architecture 19
  - 4.10.2 Complex and tuned LSTM prediction 20
- 4.11 Models comparison and summary 21

List of Figures **24**

Chapter 1

Introduction

1.1. Main goal

The main task is implementation and testing of various AI models which will be used in predicting changes in household energy consumption. The algorithms being tested are part of the *darts* python library.

The analyzed value is the hourly-averaged power consumption denoted in kilowatts, which will be predicted for various time-horizon values.

1.2. Motivation

The motivation to study energy consumption lies in newly implemented regulations mandating hourly electricity settlement or net-billing [2]. With this system, accurately predicting energy usage becomes crucial to managing power resources efficiently.

Accurate demand prediction informs decisions regarding operations of energy storage units. Knowing when to charge or discharge these units in response to the predicted demand is essential for optimizing energy use and reducing costs.

Moreover, predicting energy demand serves a larger purpose of fostering efficient energy utilization. It aids in strategic planning, balancing the energy markets, improving resource allocation and enhancing overall energy efficiency.

In essence, the emphasis on energy consumption prediction is a response to the evolving legislative requirements and the drive towards more sustainable and efficient energy systems. The future of energy consumption thus highlights the need for effective predictive models for energy usage.

1.3. Dataset Description

For the analysis, we used the "Household Electric Power Consumption" dataset available on Kaggle. It consists of approximately 2 million records representing energy consumption for a selected household over the period December 2006 - November 2010 (47 months).

Newer datasets were considered, but few were close in size and consistency compared to the aforementioned. The ones that matched this criteria were coming from developing countries, for example Uruguay, where the power usage profile would significantly differ from the one in Europe.

The description of columns from the chosen dataset - "Household Electric Power Consumption" - can be seen below.

Column Name	Unit	Description in English
Date	dd/mm/yyyy	Date in the format dd/mm/yyyy
Time	hh:mm:ss	Time in the format hh:mm:ss
global_active_power	kilowatt	Household global minute-averaged active power
global_reactive_power	kilowatt	Household global minute-averaged reactive power
voltage	volt	Minute-averaged voltage
global_intensity	ampere	Household global minute-averaged current intensity
sub_metering_1	watt-hour	Energy sub-metering No. 1. It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
sub_metering_2	watt-hour	Energy sub-metering No. 2. It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
sub_metering_3	watt-hour	Energy sub-metering No. 3. It corresponds to an electric water-heater and an air-conditioner.

Table 1.1: Data Description

Chapter 2

Related Work

2.1. Time-series data analysis using LSTM

The first paper that was analysed was Time-series data analysis using LSTM [1]. The aim of the paper is to show how to build the simplest Long Short-Term Memory (LSTM) recurrent neural network for the Household Electric Power Consumption.

After some initial data exploration, the author observer some missing values 'nan' in the dataset and dealt with them by substituting with mean of the columns.

Then some data visualization with key notes was performed. The author conducted very important note from the plots that resampling over larger time inteval, will diminish the periodicity of system as we expect. This is important for machine learning feature engineering. Another important note is that the mean of *Voltage* over month is pretty much constant compared to other features. Also from the jointplot between *Global intensity* and *Global active power* present a linear correlation, which is an important observation, but *Voltage* and *Global active power* are less correlated. This can be well observed under the Data visualization section.

Before the train, validation and test set split was performed, the data was prepared for LSTM model by framing and feature engineering. The LSTM architecture that author proposes is as follows:

- 100 neurons in the first layer
- Dropout layer with 20 % of drop
- 1 neuron in the output layer for predicting *Global active power*
- The input shape will be 1 time step with 7 features
- As a loss function Mean Absolute Error (MAE) was used
- Adam version of stochastic gradient descent
- The model was fit for 20 training epochs with a batch size of 70

The whole model architecture was written in **Tensorflow**. The author highlights that to improve the model, epochs and batch size should be increased, which is an obvious suggestion as 20 training epochs and batch size of 70 are quite small numbers for training nowadays.

In the final remarks of the paper, author stands that LSTM was the state-of-the-art model for sequential problems at that time. The problem is that the paper was released 6 years ago. Newer, better and more efficient models for time-series prediction were created which forces us for further investigation in this topic. The neural-network architecture that was designed is a toy model. It should be improved by adding CNN and dropout layers to receive better results.

Chapter 3

Plan

The scope of this work involves designing and testing a predictive model for forecasting household energy consumption.

3.1. Design Phase

1. Understanding and analyzing data: The work will begin with understanding the structure of energy consumption data. This phase involves examining trends, patterns, and seasonality in the data.
2. Feature engineering: Based on data analysis, this stage involves creating new features and modifying existing ones to better fit the models to the data.
3. Model selection: The latest models available in libraries will be considered.

3.2. Implementation Phase

1. Model implementation: In the next step, the selected model will be implemented. Programming and testing of energy consumption prediction algorithms will be necessary.
2. Model training: The model will be trained on available historical data regarding company conditions. This task involves tuning the hyperparameters and optimizing the model.

3.3. Testing Phase

1. Model evaluation: After implementation, the model will undergo testing to assess its effectiveness. Appropriate metrics will be applied to measure prediction quality.
2. Model optimization: If tests reveal that the model requires improvements, appropriate modifications will be made.

Chapter 4

Energy Consumption Prediction

4.1. Programming Language

Python was chosen as the primary language for this analysis due to its robust data science ecosystem and its strong support for time series analysis libraries like Darts. Unlike traditional machine learning libraries, Darts is specifically designed for time series forecasting, offering a streamlined workflow for data ingestion, model building, and evaluation specifically tailored to this domain. This targeted approach allows for the incorporation of time-specific features and dependencies within the machine learning models, leading to more accurate beta coefficient predictions that consider the inherent temporal nature of financial data.

4.2. Models used

The following models were selected:

- **NaiveSeasonal** - a deterministic, non-parametric forecasting model that exploits the seasonality inherent in a time series by directly using past observed values from previous seasonal cycles to make future predictions. It is particularly suitable for time series exhibiting strong and regular seasonal patterns. It is used as a simple benchmark to compare to the next to models.

- **XGBModel**- a machine learning-based time series forecasting model that uses the the XGBoost (Extreme Gradient Boosting) algorithm. It constructs a feature matrix from the past values of the series and other optional covariates, which it uses to train the XGBoost model to predict future values.

- **ARIMA** - (AutoRegressive Integrated Moving Average) a deterministic time series forecasting method that combines autoregressive, differencing, and moving average components.

- **LSTM** - (Long Short-Term Memory) model is a type of recurrent neural network (RNN) designed to effectively capture long-term dependencies and sequences in data by using special memory cells and gates to regulate the flow of information. It addresses the vanishing gradient problem, making it suitable for tasks such as time series prediction, natural language processing, and speech recognition.

4.3. Method of comparing models

Each model except *NaiveSeasonal* is being trained on the same dataset and after that all models perform predictions on the testing set. The models are later compared based on metrics:

- MAPE
- RMSE
- MSE
- MAE

The plot predicted and real values is also being made in order to graphically represent the performance of the models.

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(Y_t - \hat{Y}_t)^2 \tag{4.1}$$

4.4. Preprocessing

This section discusses data preprocessing, which prepares the dataset for subsequent analysis. The Python code snippet provided defines a function `get_series_from_dataframe`.

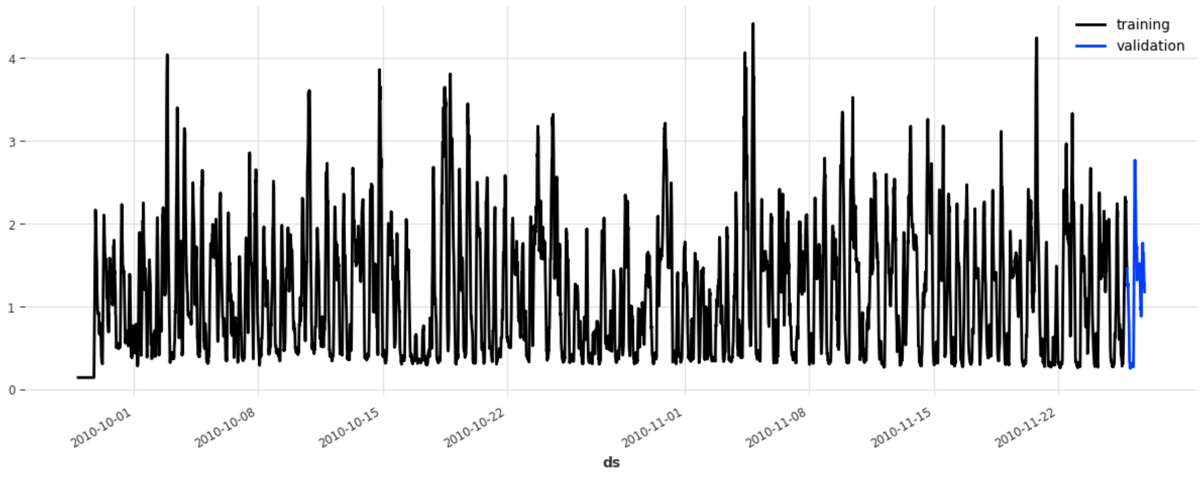
This function accomplishes the following:

- Generates a time series from a dataframe,
- Slices the data to include only the last 60 days (as defined by `TRAIN_DAYS`),
- Resamples the series at 10-minute intervals,
- Applies a Moving Average Filter for smoothing.

```python
TRAIN_DAYS = 60

def get_series_from_dataframe(dataframe, column_name):
    s = TimeSeries.from_dataframe(dataframe, time_col='ds', value_cols=column_name,
    s = s.slice_n_points_before(s.end_time(), TRAIN_DAYS*24*60)
    s = s.resample('10min', method='pad')

    ma_filter = MovingAverageFilter(12)
    s = ma_filter.filter(s)

    return s

series = get_series_from_dataframe(df, 'Global_active_power')
```

```python
train, val = series.split_after((TRAIN_DAYS - 1) / TRAIN_DAYS)
plt.figure(figsize=(15, 5))
train.plot(label="training")
val.plot(label="validation")

to_predict = len(val)
series_after_train = series.slice_intersect(val)
```

```python
xgb_forecast = xgb_model.predict(to_predict)
```



4.5. Model training with caching

```python
from darts.models.forecasting.xgboost import XGBModelnaive

try:
    with open('xgb_model.pkl', 'rb') as f:
        xgb_model = pickle.load(f)
except FileNotFoundError:
    xgb_model = None

if xgb_model is not None:
    print('xgb_model loaded')
else:
    xgb_model = XGBModel(
        lags=24 * 7 * 6,
        output_chunk_length=100,
        retrain=False
    )
    xgb_model.fit(train, verbose=True)

    pickle.dump(xgb_model, open('xgb_model.pkl', 'wb'))
```

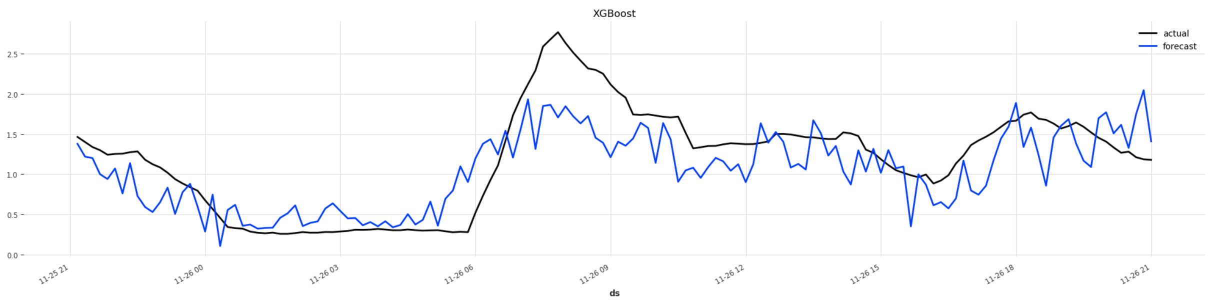
4.6. Plotting Helper Function

The `plot_forecast` function is a helper function aimed to visualize time series forecasting results, to better evaluate the model performance visually. It's built upon `matplotlib.pyplot`, a popular Python library for generating plots and other 2D data visualizations. This function facilitates the juxtaposition of the actual series and the forecasted one on a common graph, which simplifies the task of identifying discrepancies between the real and predicted results. The function includes labels and a title for easy identification of data.

The function takes three parameters: the actual series, the forecasted series, and the title of your plot.

```python
def plot_forecast(series, forecast, title):
    plt.figure(figsize=(25,5))
    series.plot(label="actual")
    forecast.plot(label="forecast")
    plt.title(title)
    plt.legend()
    plt.show()
```

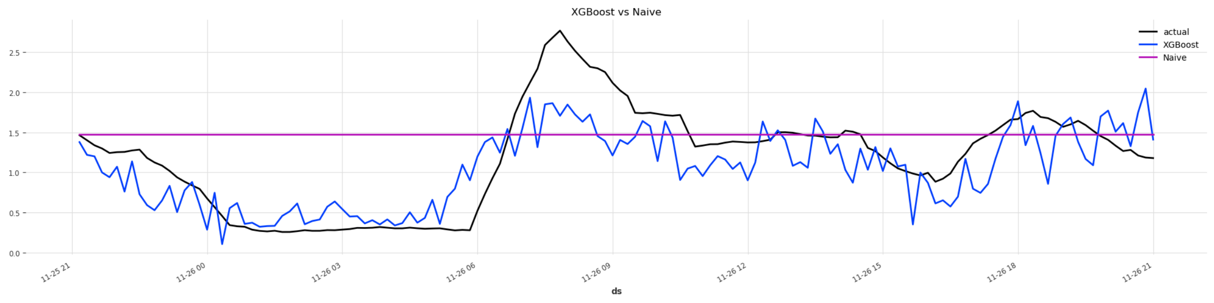
```python
plot_forecast(series_after_train, xgb_forecast, "XGBoost")
```



4.7. Baseline model (Naive Seasonal)

```python
from darts.models import NaiveSeasonal

naive_model = NaiveSeasonal(K=1)
naive_model.fit(train)
naive_forecast = naive_model.predict(to_predict)
```

4.8. Results comparison



XGBoost metrics

- MAPE = 36.21%
- RMSE = 0.41
- MSE = 0.17
- MAE = 0.32

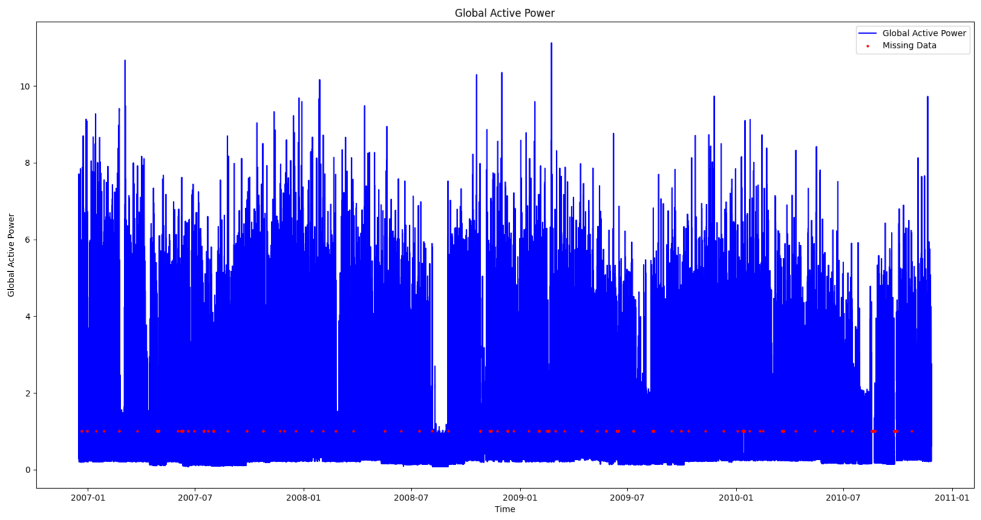
Baseline metrics

- MAPE = 113.92%
- RMSE = 0.69
- MSE = 0.48
- MAE = 0.52

4.9. LSTM Model

4.9.1. Data cleaning and exploration

Before training the LSTM model, a comprehensive data cleaning and exploration process was undertaken. Initially, all instances of '?' in the dataset were substituted with NaN values. The dataset contains 1.25% missing values which are plotted below. We can see that the missing values



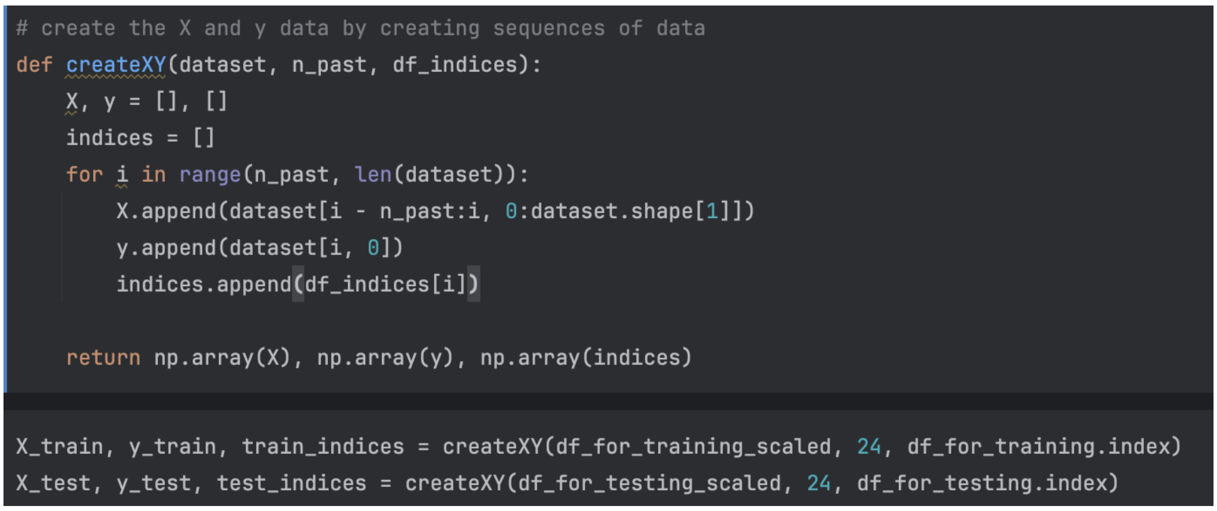
Subsequently, forward fill imputation was applied to fill these NaN values, ensuring continuity in the data. Additionally, all columns originally of the object data type were converted to float to facilitate numerical computations. Finally, the dataset was resampled to an hourly frequency to standardize the time intervals, preparing the data for effective training of the LSTM model.

4.9.2. Normalizing the data

The dataframe loaded using Pandas has a shape of (2 075 259 rows and 7 columns). Our goal is to predict the **Global Active Power** using the other columns. Given the varying ranges of values across the columns, data scaling is necessary. Therefore, MinMaxScaler was used to normalize the data within each column to the range (0, 1).

4.9.3. Train test split

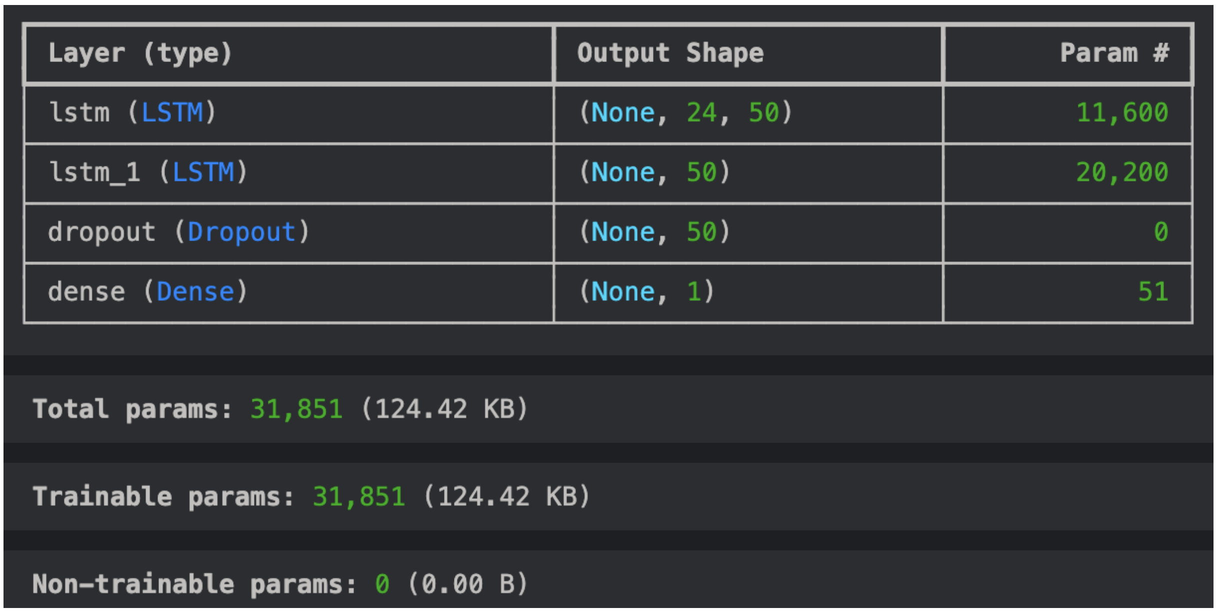
To create X and y for test and training a specific function was implemented to generate sequences of data for a time series problem.



```
# create the X and y data by creating sequences of data
def createXY(dataset, n_past, df_indices):
    X, y = [], []
    indices = []
    for i in range(n_past, len(dataset)):
        X.append(dataset[i - n_past:i, 0:dataset.shape[1]])
        y.append(dataset[i, 0])
        indices.append(df_indices[i])

    return np.array(X), np.array(y), np.array(indices)
```

```
X_train, y_train, train_indices = createXY(df_for_training_scaled, 24, df_for_training.index)
X_test, y_test, test_indices = createXY(df_for_testing_scaled, 24, df_for_testing.index)
```

4.9.4. Model architecture



Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 24, 50)	11,600
lstm_1 (LSTM)	(None, 50)	20,200
dropout (Dropout)	(None, 50)	0
dense (Dense)	(None, 1)	51

```
Total params: 31,851 (124.42 KB)

Trainable params: 31,851 (124.42 KB)

Non-trainable params: 0 (0.00 B)
```

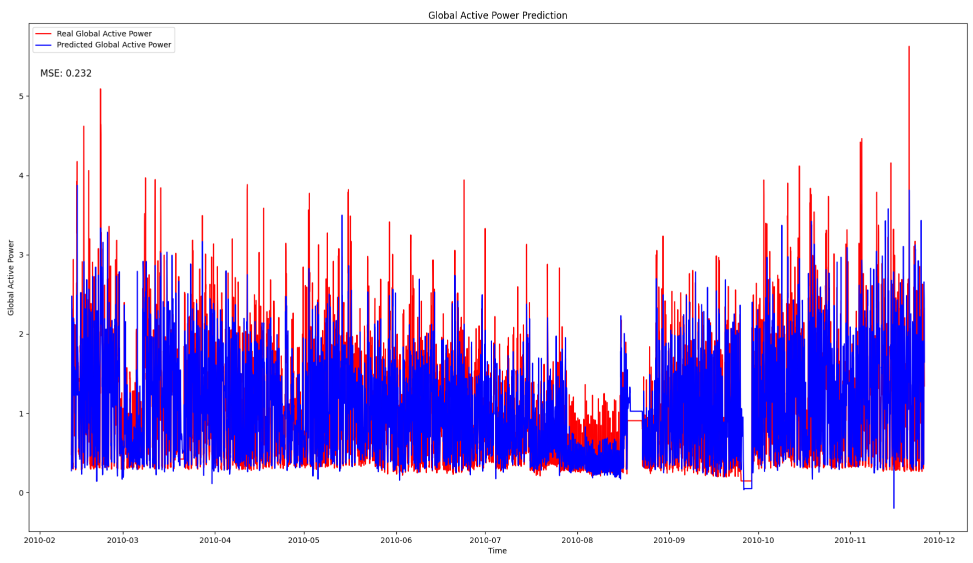
This model after 30 epochs and batch size of 64 has achieved loss: **0.0072.**

LSTM metrics

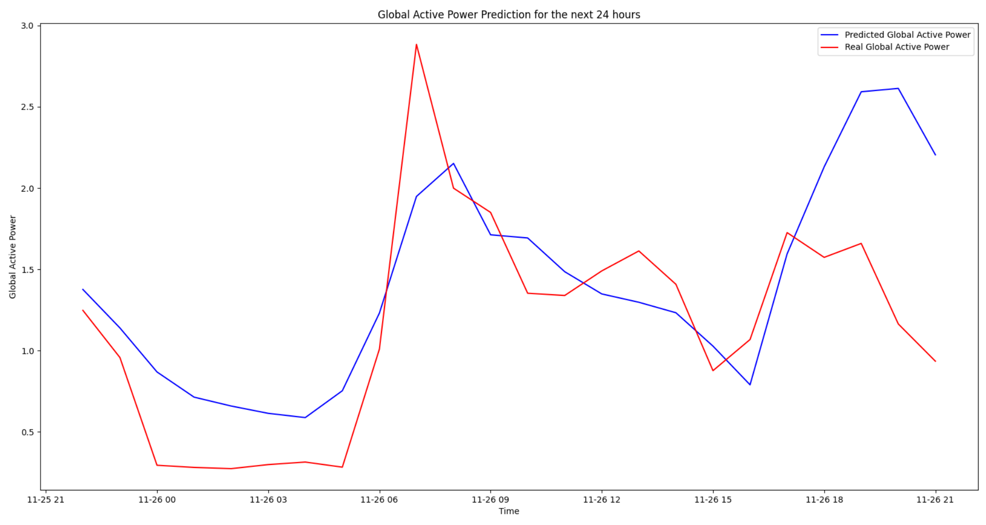
- MAPE = 44.10%
- RMSE = 0.4788
- MSE = 0.2293
- MAE = 0.3370

4.9.5. Prediction

Test prediction - period 02.2010 - 12.2010



24 hours prediction



4.9.6. Hyperparameter tunning

To optimize the hyperparameters of our LSTM model architecture, we leveraged the **Optuna** library. This process involved tuning the following:

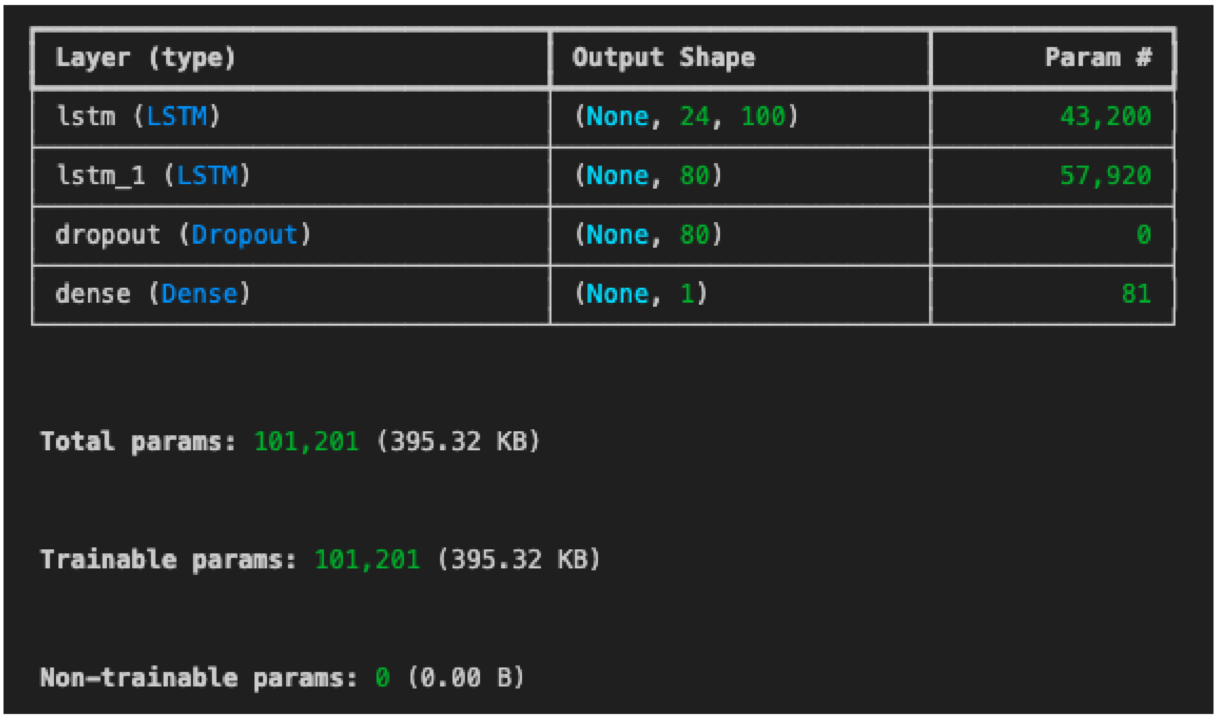
- units1 - units in 1st LSTM layer. Values: **[20, 50, 80, 100]**
- units2 - units in 2nd LSTM layer. Values: **[20, 50, 80, 100]**
- dropout - factor between 0 and 1 that controls the probability of randomly dropping out neurons during training. Values: **[0.2, 0.4, 0.6]**
- batch_size - number of data samples processed by the network in one training iteration. Values: **[32, 64, 128]**

Hyperparameter search was performed for 40 trials and with number of jobs - 5.
The best hyperparameters are:

- units1 in 1st LSTM layer - 100
- units2 in 2nd LSTM layer - 80
- dropout ratio - 0.2
- batch size - 32

4.9.7. Tuned LSTM architecture

Architecture of the model with the best hyperparameters:



```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ lstm (LSTM)                     │ (None, 24, 100)        │        43,200 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_1 (LSTM)                   │ (None, 80)             │        57,920 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 80)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 1)              │            81 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 101,201 (395.32 KB)

 Trainable params: 101,201 (395.32 KB)

 Non-trainable params: 0 (0.00 B)
```

This model after 30 epochs and batch size of 32 has achieved loss: **0.0063**.

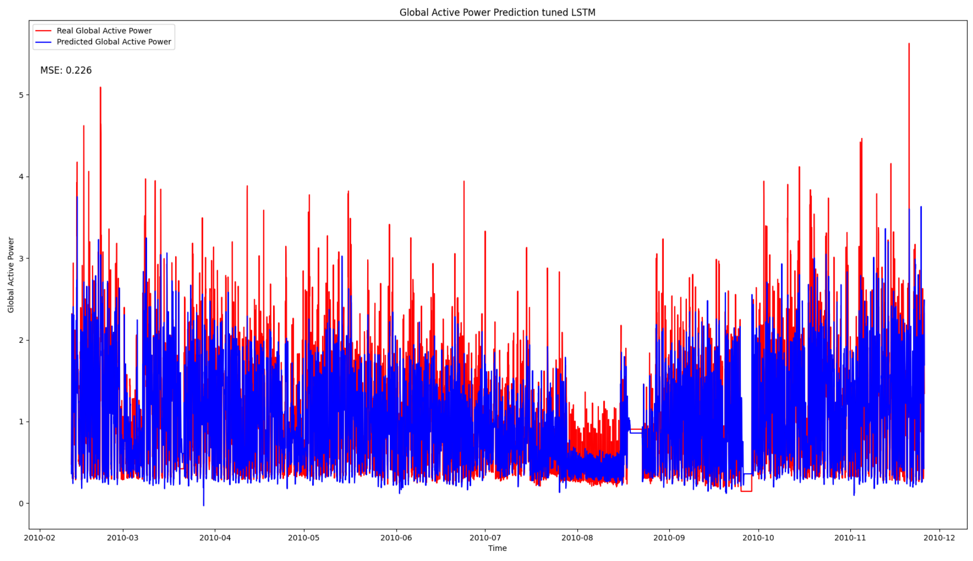
Tuned LSTM metrics

- MAPE = 42.07%
- RMSE = 0.4755
- MSE = 0.2261
- MAE = 0.3272

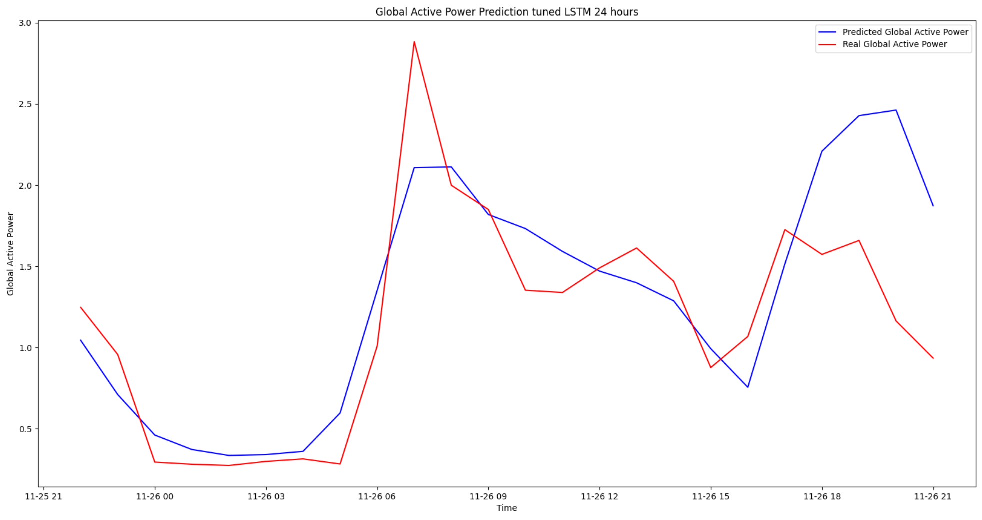
We can see that this model currently has the best error metrics, likely due to effective hyperparameter tuning and optimization.

4.9.8. Tuned LSTM prediction

Test prediction - period 02.2010 - 12.2010



24 hours prediction



4.10. More complex LSTM model architecture

After a brief team discussion, a new model was created with a more complex architecture and an additional regularization layer. The updated LSTM model consists of three LSTM layers. Each LSTM layer is followed by Batch Normalization, which stabilizes and accelerates the training process, and Dropout, which helps prevent overfitting.
To find the best parameters for this more complex model, the Optuna library was used for hyperparameter tuning once again. Over the course of 60 trials, which took approximately 3 hours with 10 jobs, the following parameters were evaluated:

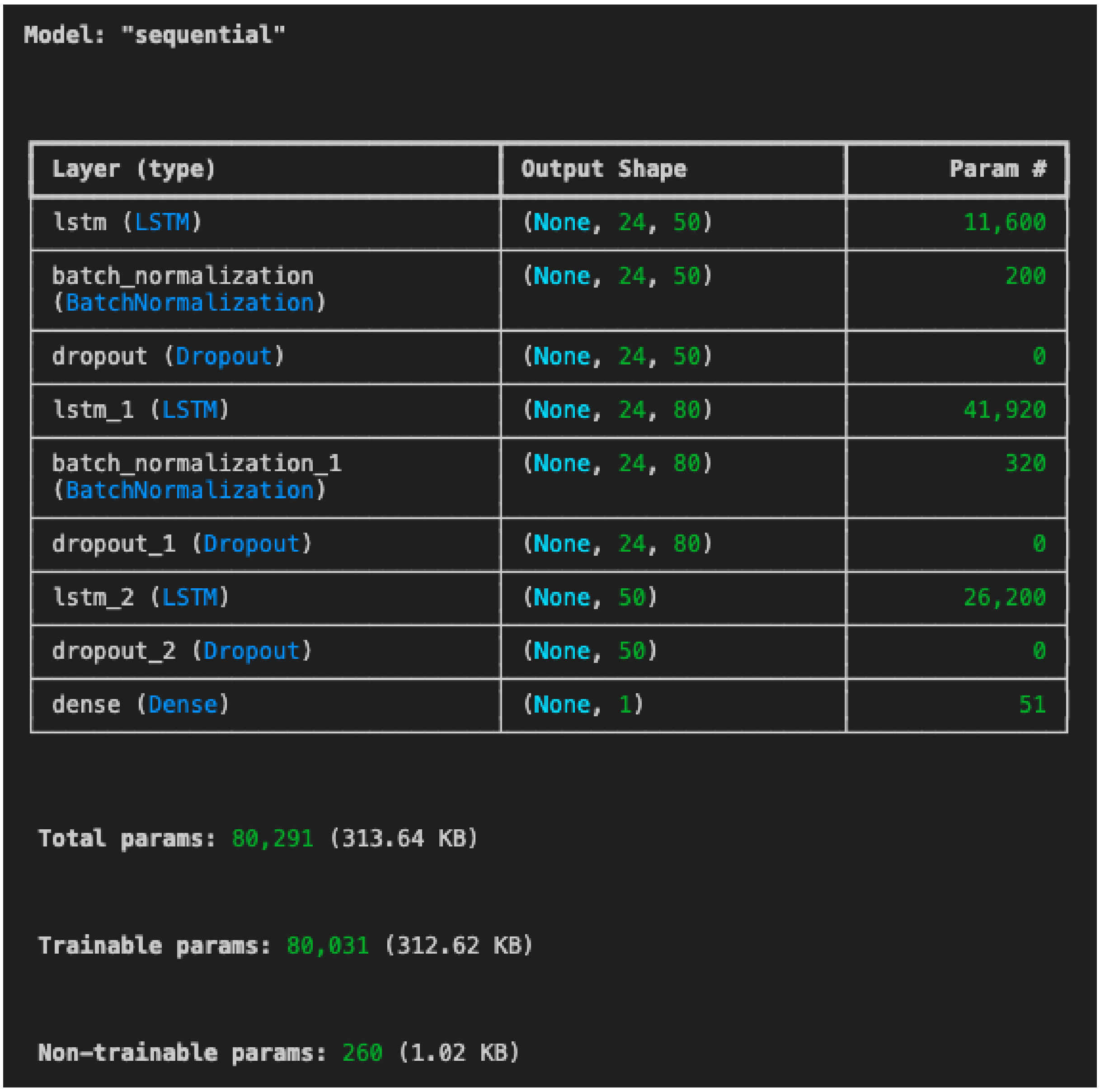
- units1 values in 1st LSTM layer: **[20, 50, 80, 100]**
- units2 values in 2nd LSTM layer: **[20, 50, 80, 100]**

- units3 values in 3rd LSTM layer: **[20, 50, 80, 100]**
- dropout ratio: **[0.2, 0.4, 0.6]**
- batch size: **[32, 64, 128]**

After the training, the best parameters selected by Optuna were:

- units1 in 1st LSTM layer - 50
- units2 in 2nd LSTM layer - 80
- units3 in 3rd LSTM layer - 50
- dropout ratio - 0.2
- batch size - 32

4.10.1. Complex and tuned LSTM architecture



```
Model: "sequential"

┌──────────────────────────────────┬─────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape            │      Param # │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ lstm (LSTM)                      │ (None, 24, 50)          │       11,600 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ batch_normalization              │ (None, 24, 50)          │          200 │
│ (BatchNormalization)             │                         │              │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dropout (Dropout)                │ (None, 24, 50)          │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ lstm_1 (LSTM)                    │ (None, 24, 80)          │       41,920 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ batch_normalization_1            │ (None, 24, 80)          │          320 │
│ (BatchNormalization)             │                         │              │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dropout_1 (Dropout)              │ (None, 24, 80)          │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ lstm_2 (LSTM)                    │ (None, 50)              │       26,200 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dropout_2 (Dropout)              │ (None, 50)              │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dense (Dense)                    │ (None, 1)               │           51 │
└──────────────────────────────────┴─────────────────────────┴──────────────┘

 Total params: 80,291 (313.64 KB)

 Trainable params: 80,031 (312.62 KB)

 Non-trainable params: 260 (1.02 KB)
```

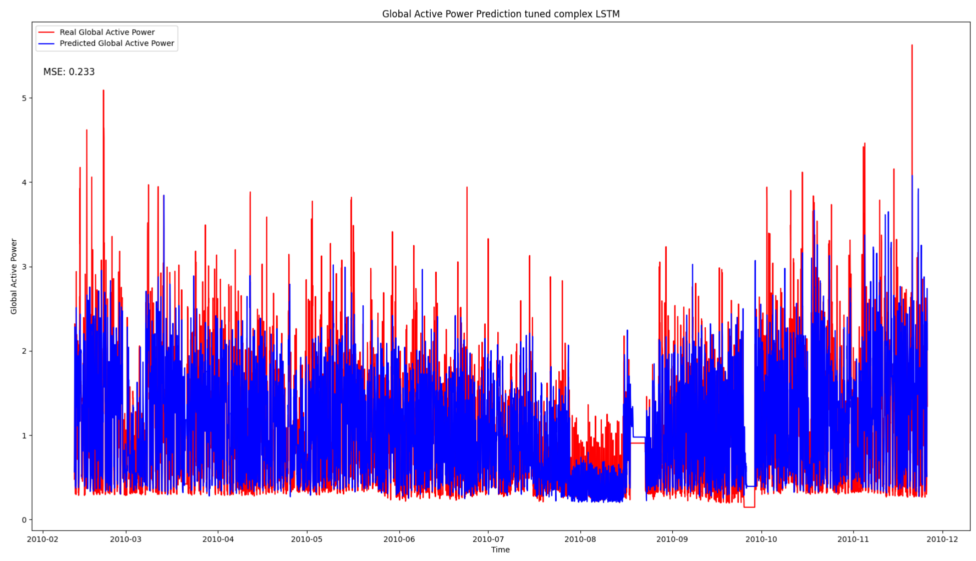
Complex and tuned LSTM metrics

- MAPE = 43.54%
- RMSE = 0.4826
- MSE = 0.2329
- MAE = 0.3311

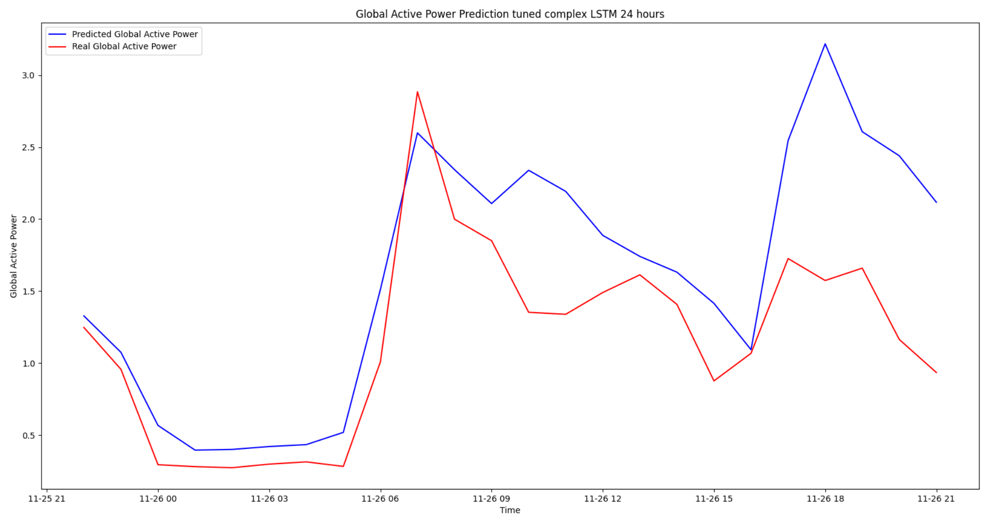
We can see that this model has slightly worse error metrics compared to the previous one. This could be due to too much of regularization or overfitting to the training data.

4.10.2. Complex and tuned LSTM prediction

Test prediction - period 02.2010 - 12.2010



24 hours prediction



4.11. Models comparison and summary

In this project, we evaluated several models to predict household electric power consumption. The dataset underwent extensive data cleaning and preprocessing, including checking for and filling NaN values using forward fill techniques. We also applied data scaling to normalize it, resampling data hourly and train/test split.

Model name	MAPE	RMSE	MSE	MAE
LSTM paper work	-	0.62	-	-
Baseline	113.92%	0.69	0.48	0.52
XGBoost	36.21%	0.41	0.17	0.32
LSTM	44.10%	0.4788	0.2293	0.3370
tuned LSTM	42.07%	0.4755	0.2261	0.3272
tunned comples LSTM	43.54%	0.4825	0.2329	0.3311

Analysis

- **Baseline Model**: The baseline model performed poorly with a MAPE of 113.92%, indicating a high prediction error.
- **XGBoost**: This model significantly improved the predictions, achieving the best results across all metrics with a MAPE of 36.21%, RMSE of 0.41, MSE of 0.17, and MAE of 0.32.
- **LSTM Models**: The standard LSTM model showed a notable improvement over the baseline but lagged behind XGBoost.
- **Tuned LSTM**: Hyperparameter tuning on the LSTM model further improved the performance, reducing MAPE to 42.07% and achieving better RMSE, MSE, and MAE compared to the standard LSTM.
- **Tuned Complex LSTM**: Although this model included more complex architecture, it did not outperform the simpler tuned LSTM, suggesting potential overfitting or inefficiencies in the model structure.

The better error values of the XGBoost model might be because it was trained only on Global Active Power data. In contrast, the LSTM models were trained on additional features that might not always accurately describe Global Active Power, potentially leading to less precise predictions.

Conclusion The **XGBoost** model emerged as the best performer, likely due to its robust nature and ability to handle the complexities of the dataset effectively. The **tuned LSTM** models showed improved performance over the standard LSTM but was still a little bit worse than XGBoost. This suggests that while LSTM models are powerful, they may require more careful tuning and optimization to outperform tree-based models like XGBoost.

Potential improvements

- **Exploring GRU Models**: A better version of LSTM, called GRU (Gated Recurrent Unit), has not been tested yet. GRUs often provide similar performance with less computational cost and may improve prediction accuracy.

- **Feature Engineering**: Enhancing feature selection and engineering could help improve model performance. Identifying and incorporating more relevant features may lead to better predictions.
- **Ensemble Methods**: Combining multiple models, such as using an ensemble of LSTM and XGBoost, might leverage the strengths of each model and result in improved accuracy.
- **Advanced and longer Hyperparameter Tuning**: More sophisticated hyperparameter tuning techniques, such as Bayesian optimization, could be employed to find the optimal settings for the LSTM models.

Bibliography

[1] A. Rezaeian. *Time-series data analysis using LSTM*. URL: https://www.kaggle.com/code/amirrezaeian/time-series-data-analysis-using-lstm-tutorial.

[2] M. K. i Środowiska. *Net-billing – system rozliczeń w praktyce*. 2021. URL: https://www.gov.pl/web/klimat/nowy-system-rozliczania-tzw-net-billing (visited on 04/16/2024).

List of Figures