Numerical Analysis and Machine Learning Project

Energy Price Prediction

Emanuele Caruso[1] **and Angelo Prete**[2]

[1]*emanuele1.caruso@mail.polimi.it*
[2]*angelo2.prete@mail.polimi.it*

Abstract—In this project, we reimplemented the study “Deep learning-based electricity price forecasting: Findings on price predictability and European electricity markets” by Aliyon and Ritvanen (2024), focusing on reproducing the forecasting methodology and validating its results. We tested it to predict day-ahead electricity prices for the Belgian and French markets. Our implementation followed the original. Hyperparameter optimization and feature selection were performed using the Hyperband algorithm. Our results align with the original findings, confirming the model’s ability to capture electricity price dynamics in these markets.

1. Introduction

Day-ahead price prediction in the energy market refers to forecasting the electricity prices for each hour of the following day. These predictions are important because electricity cannot be stored easily, so producers, suppliers, and large consumers need to plan their operations in advance. In most electricity markets, participants submit their bids for buying or selling power one day before delivery. Accurate day-ahead forecasts help reduce costs and manage risks related to price changes.

For this project we follow the approach used in the paper "Deep learning- based electricity price forecasting: Findings on price predictability and Euro- pean electricity markets" [2], a deep neural network combined with tuning of hyperparameters through Hyperband [1].

2. Data

Before being able to train the model and generate predictions, the dataset must first undergo a series of transformations. These include normalization, processing of time-related features and categorization of input variables based on their availability at prediction time. Once these steps are completed, the dataset is split into three subsets and is prepared in a way such that it can be fed to the NN.

2.1. Normalization

Normalization was performed using Z-score standardization

$$Z = \frac{X - \mu}{\sigma}$$

where the mean and standard deviation were computed over a reduced dataset, containing all samples only from the start of the training set up to the end of the validation set. In a real scenario, data from the test set is not available, so it cannot be used to normalize the data.

At first, we also experimented with min-max scaling (normalization), but since the results were very similar, we decided to go back to Z-score.

2.2. Historical and future features

The dataset includes both features available in advance w.r.t. the time of the prediction (e.g., "forecasted load") and historical features (e.g., "solar energy generated"). These two categories must be differentiated so that only data that would have been actually available is used during the training phase. As a result, only the forecasted load is used as a future feature. The Pearson correlation matrix of all features is shown in Fig. 1. As expected, a strong correlation is observed between the forecasted load and the actual load.

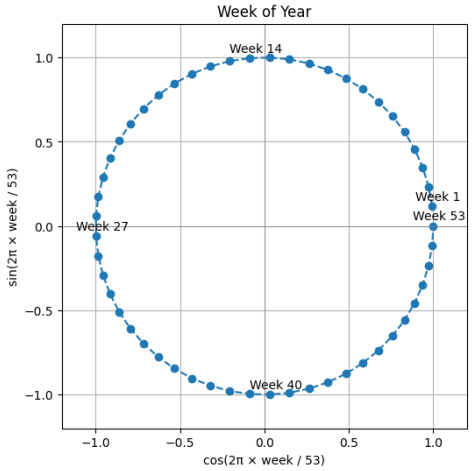


Figure 2. Transformation of the week of year time feature.

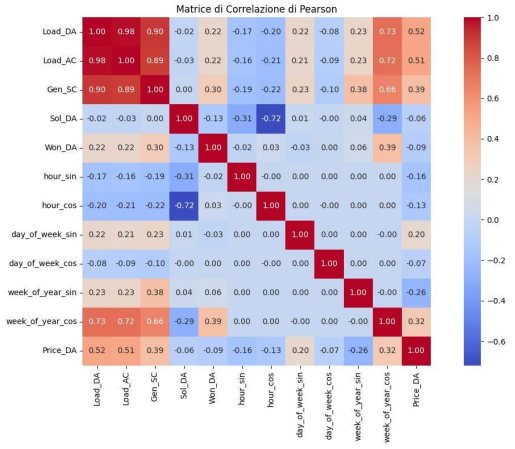


Figure 1. Pearson correlation matrix of all features

2.3. Time features

Due to their cyclical nature, time-related features require special pre-processing. Each such feature (hour of the day, day of the week, and week of the year) was transformed into two separate cyclical components using sine and cosine functions to preserve their periodic behavior.

Before applying the transformations, the features were normalized by their maximum value to scale them to the $[0, 1]$ range. The normalized values were then converted to radians and used as inputs to the sine and cosine functions.

For instance, the week of the year was transformed as follows

$$\text{week of year}_{\text{sin}} = \sin\left(2\pi \cdot \frac{\text{week}}{53}\right)$$

$$\text{week of year}_{\text{cos}} = \cos\left(2\pi \cdot \frac{\text{week}}{53}\right)$$

having as result the plot in figure 2.

2.4. Dataset split

The dataset is divided into three subsets, each serving a specific role for the model development (that will be explained in detail in the

next section):

- **Training set:** Usally contains data from the first two years of the three-year period preceding the target year. It is used to train the model and perform feature selection and hyperparameter tuning.
- **Validation set:** Consists of data from the year immediately preceding the target year. It is used to evaluate model performance during to select the best configuration of hyperparameters and features.
- **Test set:** Used only once at the end of the process to evaluate the final performance of the model build with the best hyperparameters found so far on unseen data.

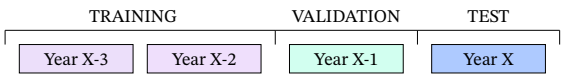


Figure 3. Split of the dataset in train, validation and test set.

2.5. Construction of the input data for the neural network

To be able to feed the data to the neural network for the training phase and to compute predictions later, the data is restructured as follows.

```
    # Inputs:
    # - previous_days: past days to include (e.g., [1, 7 ↪ ])
    # - X_history: historical features (features of ↪ previous days)
    # - X_future: future features (for the prediction ↪ day)
    # - y: energy prices

    start = max(previous_days) * 24
    initialize empty lists: X_nn, X_nn_future, y_nn

    for each time step from start to end:
        gather data from previous_days (24 hours each) ↪ from X_history

        combine all previous days data into one feature ↪ vector
        add this vector to X_nn

        add next 24 hours from X_future to X_nn_future
        add next 24 hours from y to y_nn

    flatten and combine X_nn and X_nn_future into ↪ X_combined
    convert y_nn to array

    return X_combined, y_nn
```

Code 1. Pseudocode for constructing X and y (inputs and targets)

As can be seen in the pseudocode, while historical features are only used for the days preceding the prediction day, the future features are appended at the end to the input vector, and they contain information about the prediction day that are available in advance. For example, while the true load is available only after it is measured, for the prediction day the forecasted load can be used instead.

3. Neural network

In this section we provide a brief overview of the neural network architecture, how it is trained and it's parametrization with respect to the hyperparameters, that will be useful to apply the Hyperband algorithm, explained in detail later in section 4.

3.1. Libraries used

For the construction, training, and validation of the neural network, we used the `Keras` library. Initially, the model was manually implemented using `JAX`, but we later switched to `Keras` to have more structured code and to be able to use the hyperparameter optimization mechanism `Hyperband` algorithm provided by the `Keras Tuner` library. Hyperband, as we expected, turned out to be much better that the random search approach we implemented for the first JAX implementation of this project.

3.2. Structure of the neural network

The network follows a `Sequential` model, consisting of an input layer whose size depends on the number of selected historical days and available features, followed by a variable number of dense layers with ReLU activation, L1/L2 regularization, and dropout to prevent overfitting. The output layer is a dense layer with 24 neurons, corresponding to hourly forecasts for the next day.

3.3. Regularization

As stated in the previous section, we decided to use both L1 and L2 regularization (Elastic net). In the original paper, it is mentioned that only L1 (Lasso) is used.

3.4. Initialization of the NN

We initialized the weights with the Glorot normal initialization, which draws weights from a normal distribution with zero mean and variance defined as

$$W \sim \mathcal{N}\left(0, \frac{2}{\text{fan\_in} + \text{fan\_out}}\right)$$

where fan_in and fan_out are the numbers of input and output units, respectively.

Biases are initialized to zero instead.

3.5. HyperModel

We defined a custom `HyperModel` class that allows the dynamic construction of the neural network based on the hyperparameters selected during the tuning process. Among the hyperparameters considered are: the number of hidden layers (from 2 to 5), the number of neurons per layer (from 64 to 1024, in steps of 64), the L1 and L2 regularization coefficients (log-scaled between 10^{-5} and 10^{-2}), the dropout rate (between 0.01 and 0.3), the learning rate (also selected on a logarithmic scale), and the batch size (between 7 and 56). Additionally, the choice of historical days to include as input (e.g., previous day, two days before, three days before) is also treated as a hyperparameter, directly impacting the dimensionality of the input vector.

3.6. Input data

The training data includes both historical features (such as past solar production) and known-in-advance features (such as load forecasts or time-of-day information), properly combined into a structured input that is built as previously explained in subsection 2.5.

3.7. Training

Training is carried out using the `Adam` optimizer and the loss function is the Mean Absolute Error (MAE), while the evaluation metric used is the Symmetric Mean Absolute Percentage Error (sMAPE). Furthermore, the model is trained with an early stopping strategy, which halts training when validation performance stops improving, to reduce the computational costs (and not waste time on hyperparameter configurations that are not promising).

4. Hyperband

We used `Hyperband` algorithm from Keras Tuner, an efficient method for automatic hyperparameter optimization. Initially, it evaluates a large number of randomly selected hyperparameter configurations using limited computational resources (e.g., a small number of training epochs). It then progressively eliminates the poorly performing configurations, allocating more resources to the top-performing ones. This approach allows Hyperband to find good hyperparameter settings while avoiding unnecessary computation on suboptimal configurations — making it particularly suitable for deep neural networks.

In our implementation, the computational resource used was the number of training epochs, with a minimum of 20 epochs allocated to each configuration to ensure that all configurations are explored a little bit before being pruned.

Example Output After 1 Hour of Hyperband Optimization on BE dataset

Table 1. Neural network architecture after 1 hour of Hyperband tuning

Layer (type)	Output Shape	Param #
Dense (dense_4)	(None, 192)	133,824
Dropout (dropout_3)	(None, 192)	0
Dense (dense_5)	(None, 768)	148,224
Dropout (dropout_4)	(None, 768)	0
Dense (dense_6)	(None, 24)	18,456

Note: This table reports the architecture of the best-performing model configuration identified after 1 hour of Hyperband-based hyperparameter tuning.

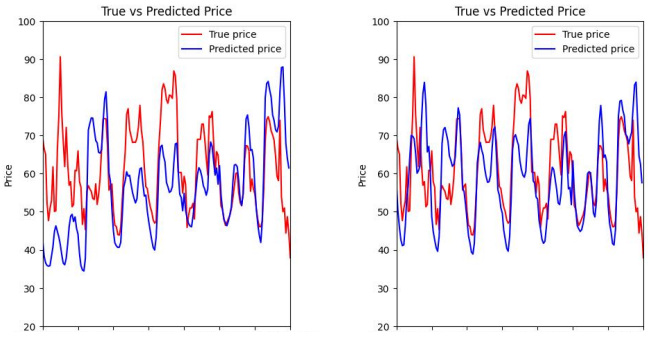
5. Computing Predictions

After selecting the best hyperparameters found by Hyperband, a model is build with them and trained on the testing and validation data. In particular, the model is retrained for each day in the test set using a sliding window approach described in detail below. Before training the model for a new day, the previous weights are restored to make the learning faster.

5.1. Sliding Window Procedure

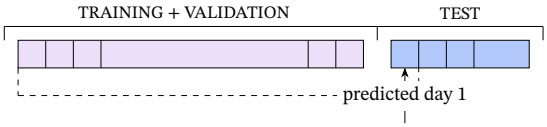
To simulate a real-time forecasting, we used the following sliding window strategy:

- **Initialization:**
  - *Data:* Years 2016–2018 (used for training and validation).
  - *Window length W:* The total duration of training and validation (e.g., 3 years).
- **Daily Iteration:** For each target day in the year 2019 (test set):
  1. **Dataset Construction:**
     - *Input:* Data from the most recent W days.
     - *Target:* The actual electricity price for the current day.
  2. **Model Retraining:**
     - The model is retrained starting from the weights obtained for the previous day, using the best hyperparameters selected by the Hyperband algorithm.
  3. **Prediction:**
     - The forecast for the current day is generated.
     - The window is then updated: the most recent day (including actual historical data) is added, and the oldest day is removed, as shown in figure 4

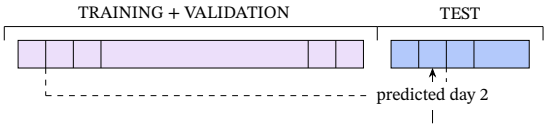


(a) patience=40 **(b)** patience=200

Figure 5. Comparison of different patience values over a week of predictions.



(a) First prediction window, that includes all days of the training and validation datasets.



(b) Second prediction window, notice how the first day of the training dataset was removed

Figure 4. Sliding window

5.2. Patience

A manually tunable parameter is the patience used when computing predictions, that is the number of epochs after which we stop if no improvements during the optimization step are found.

In figure 5 we compare a patience of 40 epochs with a patience of 200. We can see that the in the latter case the prediction is much closer to the true price. While in figure 5a the model isn't even able to follow the price spike, in figure 5b it reproduces it closely.

As a tradeoff, we get a much increased computational time, from tens of seconds to a couple minutes for a daily prediction, tested on a Macbook Air M3.

Since when model is retrained for the next prediction the previous weights are restored, a possibility we explored is using an higher patience for the first weeks and then switching to a lower one.

6. Metrics

During the model performance evaluation phase, two metrics were mainly used: sMAPE (Symmetric Mean Absolute Percentage Error) and rMAE (Relative Mean Absolute Error).

sMAPE measures the percentage error symmetrically, reducing excessive penalization on very small actual values. Formally, it is defined as:

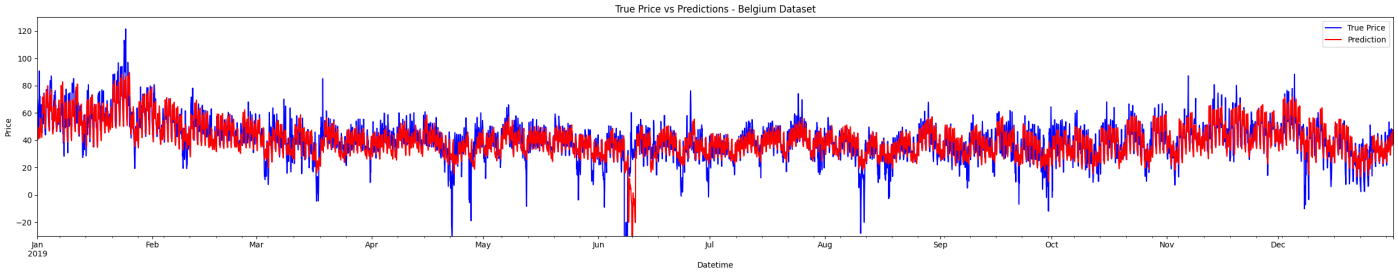


Figure 6. Predictions of energy prices in 2019, Belgium dataset.

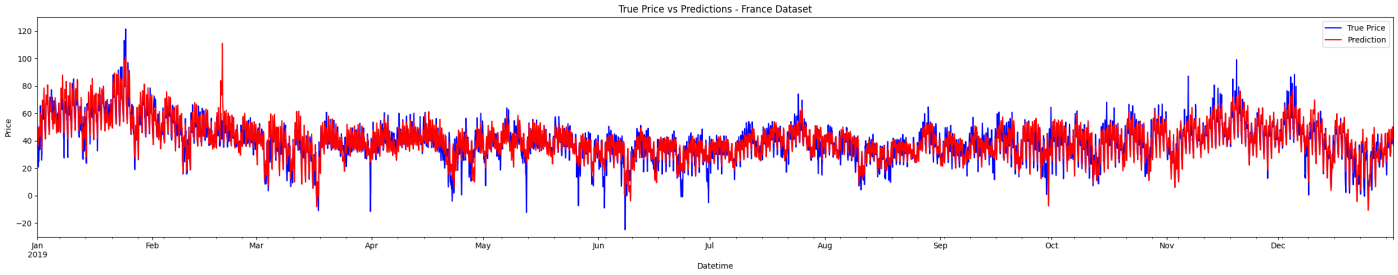


Figure 7. Predictions of energy prices in 2019, France dataset.

$$\text{sMAPE } \% = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(y_i + \hat{y}_i)/2}$$

where y_i is the true value and $\hat{y}_i$ is the predicted value. sMAPE returns an average percentage error, useful for comparing models on heterogeneous time series.

rMAE evaluates the model's accuracy relative to a "naive" baseline model, which represents a simple forecast (e.g., assuming the future value equals the previous day's value). It is calculated as:

$$\text{rMAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\sum_{i=1}^{n} |y_i - y_i^{\text{naive}}|}$$

where y_i^{naive} is the naive forecast. An rMAE less than 1 indicates the model is more accurate than the naive approach, while a value greater than 1 suggests the opposite.

We used both metrics because together they consider both absolute and relative performance.

7. Results

We tested the model on two datasets containing data from the French [3] and Belgian [4] energy markets. The specific datasets we used are published in the GitHub repository of the original paper we follow. In general, the data is available through ENTSO-E (European Network of Transmission System Operators for Electricity), which provides open-access information on electricity markets and grid operations across Europe.

7.1. Predictions

As mentioned earlier, we tested our model using two different datasets: one with more stable energy prices (France) and another with considerably higher volatility (Belgium).

Both datasets cover the period from January 1, 2016, to December 31, 2019.

For the Belgian dataset, the model closely follows the price trends, as shown in Figure 6, capturing weekly and daily patterns quite well. However, the model struggles during sudden, big price drops, which are not well predicted.

In contrast, the French dataset shows lower rMAE values. Figure 7 shows a strong overlap between true and predicted prices, due to the dataset's more stable behavior.

7.2. Comparison with the original paper

A report of our model performance compared with the one of the original paper can be found in table 2. We obtained the same rMAE while having a smaller sMAPE with respect to the original paper on the French dataset, while both metrics where a little bit higher on the Belgian one.

Table 2. Forecasting performance metrics for Belgium and France

Country	Metric	Value
Belgium	sMAPE (ours)	17%
	rMAE (ours)	0.70
	sMAPE (original paper)	23%
	rMAE (original paper)	0.67
France	sMAPE (ours)	14%
	rMAE (ours)	0.63
	sMAPE (original paper)	18%
	rMAE (original paper)	0.63

Note: Values represent forecasting metrics from our model and those reported in the original paper for Belgium and France energy markets.

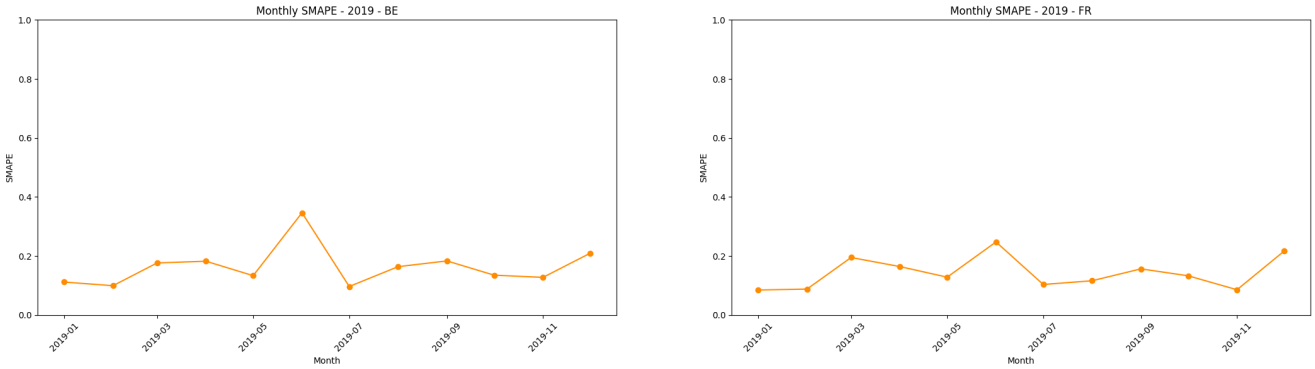
7.3. sMAPE comparison across datasets

Figure 8 shows the average monthly sMAPE of our model for the France and Belgium datasets. As anticipated, the French dataset performs slightly better, showing a lower sMAPE due to its higher predictability and the absence of significant price drops.

8. Attempts at further improvements

At the beginning of the paper we followed, it is mentioned that other studies have experimented with models such as LEAR, which are faster but less accurate in prediction. We attempted to combine both approaches; however, this did not provide the desired improvements.

We also tried to preprocess the data to make it stationarity by removing simple, easily identifiable trends. For example the price at a given hour tends to be similar to the price at the same hour on the previous day, as well as across the same weekday or week of the year. The



(a) Monthly SMAPE of BE dataset.

(b) Monthly SMAPE of FR dataset.

Figure 8. Comparison of averaged monthly SMAPE of our model on 2019 data.

aim was to help the neural network focus on more complex patterns. Unfortunately, this approach led to unstable behavior when big price fluctuations occurred between consecutive days, resulting in a high rMAE. The unstable behaviour is show in figure 9. Ultimately, we decided to continue with the previously described approach, which provided more consistent results.

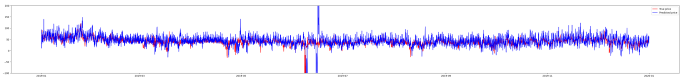


Figure 9. Predictions with alternative approach using differenced data

References

[1] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization", *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[2] K. Aliyon and J. Ritvanen, "Deep learning-based electricity price forecasting: Findings on price predictability and european electricity markets", *Energy*, vol. 308, p. 132 877, Nov. 2024. DOI: 10.1016/j.energy.2024.132877.

[3] ENTSO-E, *Belgium dataset*, Accessed: 2025-06-23. [Online]. Available: https://github.com/Kasra-Aliyon/Deepforkit/blob/main/data/FR.csv.

[4] ENTSO-E, *France dataset*, Accessed: 2025-06-23. [Online]. Available: https://github.com/Kasra-Aliyon/Deepforkit/blob/main/data/BE.csv.