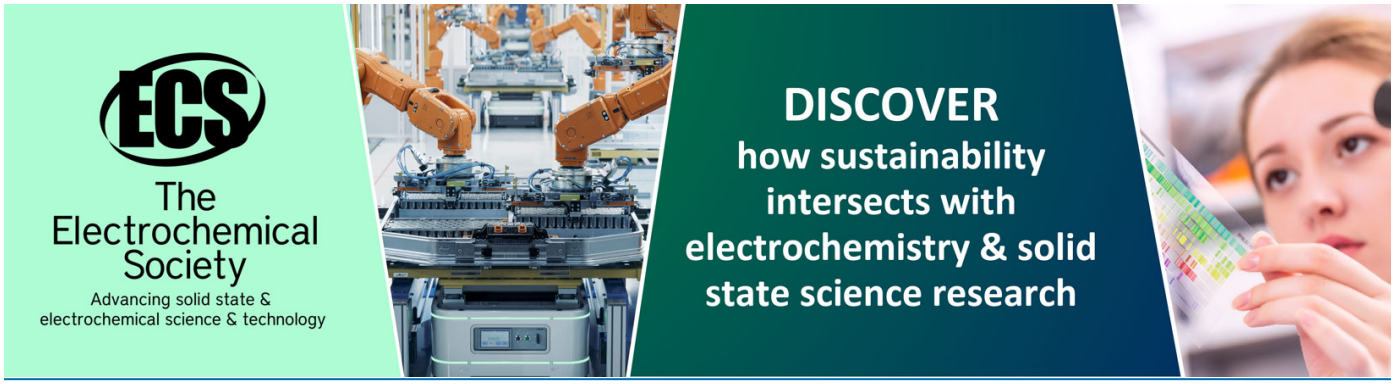
Energy demand prediction with machine learning supported by auto-tuning: a case study



Energy demand prediction with machine learning supported by auto-tuning: a case study

Sorana Ozaki[1], Ryozo Ooka[2] and Shintaro Ikeda[2]

[1] Department of Architecture, The University of Tokyo, Tokyo, Japan
[2] Institute of Industrial Science, University of Tokyo, Tokyo, Japan
[3] Department of Innovation Science, Tokyo Institute of Technology, Tokyo, Japan

E-mail: sorana-ozaki@g.ecc.u-tokyo.ac.jp

Abstract. The operational energy of buildings is making up one of the highest proportions of life-cycle carbon emissions. A more efficient operation of facilities would result in significant energy savings but necessitates computational models to predict a building's future energy demands with high precision. To this end, various machine learning models have been proposed in recent years. These models' prediction accuracies, however, strongly depend on their internal structure and hyperparameters. The time demand and expertise required for their finetuning call for a more efficient solution. In the context of a case study, this paper describes the relationship between a machine learning model's prediction accuracy and its hyperparameters. Based on time-stamped recordings of outdoor temperatures and electricity demands of a hospital in Japan, recorded every 30 minutes for more than four years, using a deep neural network (DNN) ensemble model, electricity demands were predicted for sixty time steps to follow. Specifically, we used automatic hyperparameter tuning methods, such as grid search, random search, and Bayesian optimization. A single time step ahead, all tuning methods reduced the RSME to less than 50%, compared to non-optimized tuning. The results attest to machine learning models' reliance on hyperparameters and the effectiveness of their automatic tuning.

Keywords. energy demand prediction, machine learning, hyperparameter search, auto tuning

1. Introduction

1.1. Background

In recent years, due to an increase in environmental concerns, there has been a growing interest in reducing the operational energy demand of buildings. With the development of Internet of Things (IoT) technologies, more buildings are now managing their entire operational control. To implement efficient and optimal operational control of buildings' energy systems, accurate forecasting methods to predict future energy demands are required. Recently, a variety of machine learning methods, aiming to achieve high prediction accuracies at this task, have been developed.

1.2. Objective

Machine learning models usually come with an abundance of parameters that are required to be set in advance and are known to influence its prediction accuracy. Deciding what parameters to use is a time-intensive process and often requires extensive experience in computational model building. Against this background, automatic hyperparameter tuning of machine learning models using specified

optimization algorithms, instead of manual finetuning, has been on the rise [1]. Such tuning does not only improve prediction accuracy and reduce human effort but, by making the method more accessible, may also contribute to more widespread use of demand prediction as part of a building's operation control. In this study, different auto-tuning methods were introduced into a prediction model for electricity demand based on real data, with the aim of improving accuracy, reducing model building time, and comparing prediction results for different styles of auto-tuning.

2. Methodological framework

In this study, we utilize a feed-forward deep neural network (DNN) as predictive model to probe the effect of auto-tuning. There are four main steps to predict future electricity demands: (1) Pre-processing of the dataset (see 3.1), (2) Selection of the best parameters by tuning, based on the DNN model (see 2.2), (3) Repeated training of the DNN model with the optimal parameters (see 2.1), (4) Evaluation of the model performance when predicting the test data (see 2.4). What follows is an introduction to the models and methods used throughout the paper.

2.1. DNN model

The DNN model used consists of a multi-layered neural network, capable of computing complex non-linear functions. The model, as detailed in Figure 1 and Table 1, is divided into different prediction time steps and is equipped with *i* input nodes and a single output node. When predicting *n* time steps ahead, *n* DNN models are built and the results for each are combined to derive a final prediction. The choice to have separate networks for separate time steps is based on the consideration that a single model making predictions for all time steps may have negative effects on the time steps in which the prediction errors are lower, as each weight is adjusted to reduce the sum of the errors for all time steps. Furthermore, by tuning the hyperparameters for each time step individually, it is possible to build a structure specific to the time step under consideration, allowing analysing its characteristics. It is also possible to compare the results with those of a model built with Auto ML (see 2.4), which has only one target.

The DNN settings are shown in Table 1. The network is composed of three hidden layers and uses the He initialization to effectively initialize the weights as well as the L2 regularization to prevent overfitting, as has been shown previously [2]. The network is using the Adam optimization algorithm and the Scaled Exponential Linear Unit (SELU), expressed by Equation 1, as activation functions, which are known to be capable of yielding certain accuracies in the hidden layers [3]. Except for the hyperparameters under consideration, the DNN structure is the same for auto-tuning and training, but the number of epochs and the index for early stopping patience – a function to stop learning when there is no improvement in the evaluation performance for the number of epochs – is reduced, in order to perform many searches in a short time.

$$y = \lambda \begin{cases} x & (x > 0) \\ a(\exp(x) - 1) & (x \leqq 0) \end{cases} \qquad \lambda \fallingdotseq 1.05, a \fallingdotseq 1.67 \tag{1}$$

Table 1. DNN model structure.

	Hidden layers	Initial value	Regularisation	Optimizer	Activation	Max epochs	Early stopping
Tuning model	3	He	L2	Adam	SELU	300	10 or 1
Training model						30000	500

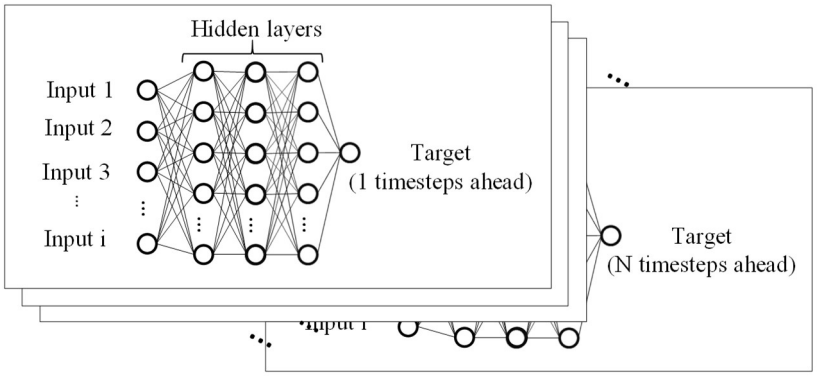


Figure 1. DNN model.

2.2. Auto Tuning

The parameters to be tuned in this paper are hyperparameters with large ranges of continuous values that are difficult to choose manually. The hyperparameters of interest comprise *batch size* (32–512), *learning rate of the optimizer* (0.0001–0.01), *node1*: the number of nodes in the first hidden layer (100 –500), *node2*: the number of nodes in the second/third hidden layer (100–500), *dropout1*: the drop-out rate of the first intermediate layer (0.1–0.5), and *dropout2*: the drop-out rate of the second and third intermediate layer (0.1–0.5). The drop-out rate thereby represents the percentage of nodes that are inactivated to prevent overfitting during network training. The number of searches was set to 100 for random search and Bayesian optimization.

2.2.1. Grid Search: The grid search iterates through the entire set of hyperparameter combinations, within each parameter's range of representative values. This method is intuitive and widely used. As the number of setting parameters increases, the calculation time becomes enormous. Therefore, we selected only the maximum, minimum, and median values within the setting range, restricting the number of calculations to 3^6=729.

2.2.2. Random Search: The random search algorithm uses a random number generator to select of hyperparameter settings each time. Since the evaluation performances of all settings are independent, they can be calculated in parallel. However, no information obtained from the evaluation values during the search can be used. [4] This algorithm uses K-fold cross-validation to train multiple times and compute the average of the evaluation values.

2.2.3. Bayesian optimization: Bayesian optimization utilises the information obtained from the evaluation function to build the model. As the construction of the model and the selection of the next hyperparameter setting to be evaluated are alternating, the evaluation is sequential. Therefore, this method is not suitable for parallelisation, but, in contrast to the random search procedure, can take advantage of the results of the search. The tuning of the hyperparameters is carried out as follows:

1. Select parameters from a set range of parameter values.
2. Train the DNN model with the selected parameters and return the validation loss.
3. Update the hyperparameter to maximize the evaluation value.
4. Repeat 1-3, then select the parameter with the minimum validation loss as the best.

2.3. Auto ML

Automated Machine Learning (Auto ML) can automatically build prediction models by loading a dataset and setting a target. We utilize 'Auto ML Table' that is provided by Google Cloud Platform to automatically build linear regression models to check the performance of Bayesian optimization.

2.4. Evaluation metrics

The coefficient of determination (R^2), Root Mean Square Error (RMSE), and Expected Error Percentage (EEP) [5] are adopted as metrics to evaluate the model performance with different parameter values. RMSE is an absolute value indicator and given by the square root of the mean squared error between the correct (measured) value of the target, *t,* and the predicted output value, *y,* over the period of the prediction time step, *n* (see Equation 3). When evaluating each prediction time step, *n* represents the total number of data points. EEP, to the contrary, is a relative value indicator, given by the RMSE over the maximum value of the target prediction period (in this case, 1310 kWh, the maximum value of the test data; see Equation 4).

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(t_i - y_i)^2}{\sum_{i=1}^{m}(t_i - \bar{y_i})^2} \tag{2}$$

$$\text{RMSE} = \left(\frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2\right)^{1/2} \tag{3}$$

$$\text{EEP [\%]} = \frac{\text{RMSE}}{\text{t}_{\text{max}}} \times 100 \tag{4}$$

3. Case study

3.1. Data collection and pre-processing

A Japanese hospital has considered the installation of automatic control of electricity, cooling, heating, and hot water systems in its buildings. To judge the utility of this approach, a case study was conducted, based on previously recorded Building Automation System (BAS) data, detailing the outdoor temperature and the electricity demand for every 30 minutes from April 1, 2016 at midnight until September 15, 2020 at 5:00 pm. Here, the electricity demands for the 60 time steps following a given recording were to be predicted. The input data used for this prediction included the recorded month, day, and time, the day of the week, whether the target day, or the day to follow, was a holiday (1 if holiday; 0 if not), the outdoor air temperature up to 60 time points following the target time, and the electricity demands of the preceding 144 time points. These input conditions are shown in Table 3.

Following the data collection, the dataset was preprocessed. Due to maintenance, some measurement points were missing. Consequently, intervals with many consecutive, i.e., long-term, missing values were removed from the dataset, whereas temporarily missing values were replaced by measurements of the previous time point. Furthermore, outliers in the dataset were removed by replacing values that had electricity demands below 500 kWh or above 1500 kWh with values obtained in the time step before. This was carried out, in a standardized fashion, for training and prediction.

The total dataset contained 76615 samples, randomly divided into 80% training data (61292 samples) and 20% test data (15323 samples). In addition, 20% of the training data were used for training evaluation.

Table 2. Summary of the building and data collection.

Location	Ibaraki, Japan
Building use	Hospital (800 beds)
Total floor space	79,604 m^2 (including 3,930 m^2 for the Energy Center)
BAS data	Electricity demand [kWh/30min] / Temperature[℃]
Data period	2016.4.1 ~ 2020.9.15

Table 3. Overview of the input and target data.

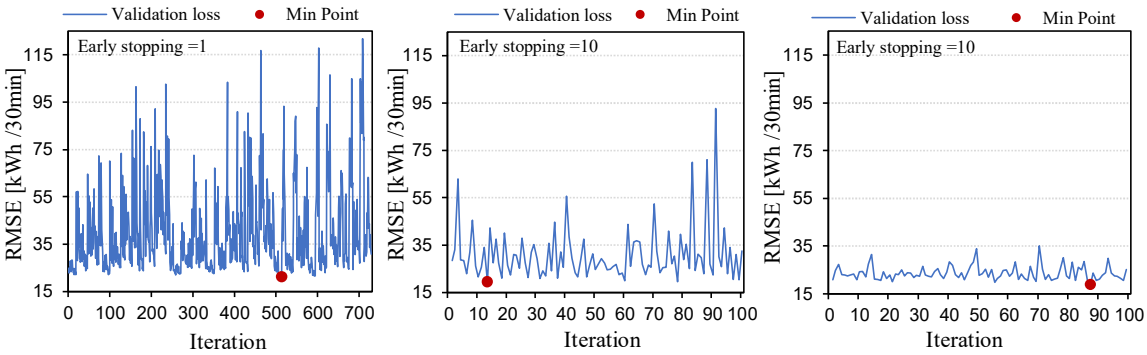
	Input data (Value, Interval)	Target data
1	Month (1-12)	
2	Day (1-31)	
3	Time (0-23, 0.5)	
4	Day of week (0-6)	Electricity demand
5	Holiday value of the day (0/1)	from 1 to 60 steps ahead
6	Holiday value of next day (0/1)	
7~66	Temperature: 60 steps ahead	
67~210	Electricity demand: 144 steps back	

4. Results and discussion

4.1. Tuning result

Figure 2 depicts the evaluation results of the search process for all tuning methods, with smaller RMSEs indicating smaller losses. Due to an early stopping of 1, the settings obtained through grid search tend to result in high RMSEs, with some being below 20 kWh, demonstrating that if the search is carried out orderly, a highly accurate setting can be obtained. Using grid search, the best score was obtained in the 514th out of 729 search runs. While random search yielded its highest score on the 14th out of 100 runs, Bayesian optimization did so on the 87th out of 100 parameter variations, illustrating that, using this method, past search results are used to progressively improve the parameter selection.

The parameter settings that obtained the highest score for each tuning method are shown in Table 4. While the number of batches, as well as the learning rate, differ across tuning methods, the number of nodes (mostly around 300–400) and the dropout rates (0.1–0.2) are comparable. The overall search time is longest for grid search, which requires a large number of search iterations, followed by random search, which uses K-fold cross-validation to train multiple times and average the evaluation results. The parameter selection distribution for the number of nodes for each method is shown in Figure 3. The figure reveals that the search points selected through Bayesian optimization are closer to the minimum point (i.e., the setting yielding the lowest RMSE) than they are when using random search.

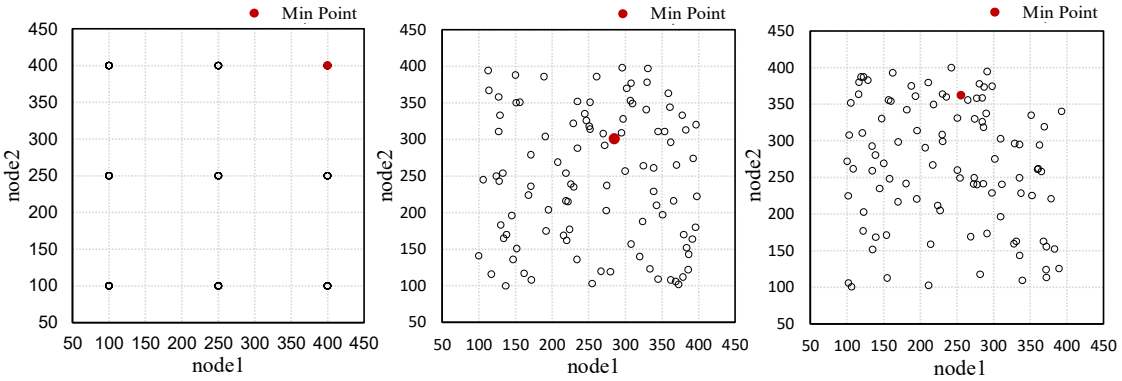


(A) Grid search (B) Random search (C) Bayesian optimization

Figure 2. Evaluation of RMSEs during the search process for different auto-tuning methods.

Table 4. Optimized hyperparameters, and search time, for different auto-tuning methods when predicting one time step ahead.

	Batch Size	*Learning rate*	*Node1*	*Node2*	*Dropout1*	*Dropout2*	Time[s]
(No adjustment)	(128)	(0.01)	(100)	(100)	(0.1)	(0.1)	-
Grid Search	32	0.0001	400	400	0.1	0.1	10785
Random Search	483	0.0015	311	354	0.14	0.2	7832
Bayesian optimization	416	0.00019	255	362	0.168	0.189	2322



(A) Grid Search (B) Random Search (C) Bayesian optimization

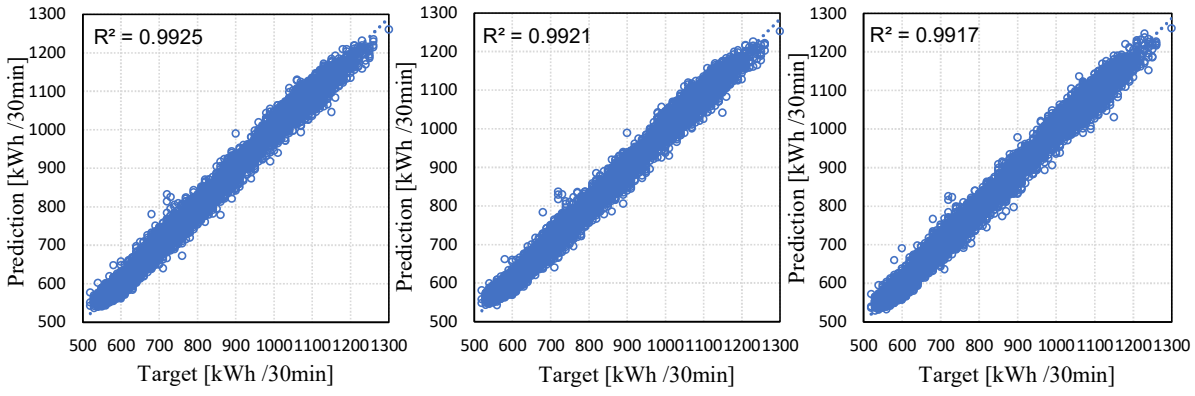
Figure 3. Distribution of values selected for the node hyperparameter throughout the search process.

4.2. Predicting one time step ahead

Table 5 shows the RMSEs and EEPs when predicting the next time step, for the model trained without hyperparameter adjustment, three models trained with the hyperparameters obtained through each of the presented auto-tuning methods, as well as the model trained using Auto ML. Compared to the model trained without any hyperparameter adjustment, the RMSEs of all adjusted models are reduced by at least 50%, to less than 19 kWh. The prediction accuracies of the three auto-tuned models, expressed as R^2, are shown in Figure 4. Despite small differences in individual test predictions, slightly favouring grid search, the overall accuracies are comparable across all methods.

Table 5. Evaluation of prediction model one time step ahead

	RMSE [kWh/30min]	EEP [%]
No adjustment	40.1	3.06
Grid Search	17.2	1.31
Random Search	18.2	1.39
Bayesian optimization	18.6	1.41
Auto ML	17.0	1.30



(A) Grid Search (B) Random Search (C) Bayesian optimization

Figure 4. Accuracy for predictive model one time step ahead.

4.3. Predicting multiple time steps ahead

Given the similarity and high prediction accuracies of all three tuning methods, due to its short search time, we chose Bayesian optimization for predicting the next 60 time steps and compared the results to those obtained when training the model using Auto ML and when training without adjusting the hyperparameters.

The average RMSE and EEP values for each model are given in Table 6. Figure 5 further depicts the prediction loss for each target time. With the exception of a single time step, Bayesian optimization yields stable improvements in RMSEs, relative to the model whose hyperparameters are not optimized. These RMSEs are approaching those of the model trained using Auto ML, but do not fully reach it.

The time series of the prediction results are shown for 19:30 on Wednesday, May 7, 2016 as a sample weekend (see Figure 6a) and for 12:00 on Wednesday, April 28, 2020 as a sample holiday (see Figure 6b). The EEP for weekend prediction is 0.77% for Auto ML and 1.02% for Bayesian optimization, both of which are highly accurate, with Auto ML being slightly superior. The EEP for the holiday prediction is 1.99% for Auto ML and 1.54% for Bayesian optimization, indicating that while Auto ML is more accurate for weekend predictions, Bayesian optimization yields better predictions for holidays, possibly because of better usage of comparatively fewer training data.

Table 6. Average prediction loss by trained model.

	RMSE [kWh/30min]	EEP [%]
No adjustment	32.8	2.49
Bayesian optimization	22.8	1.74
Auto ML	17.7	1.34

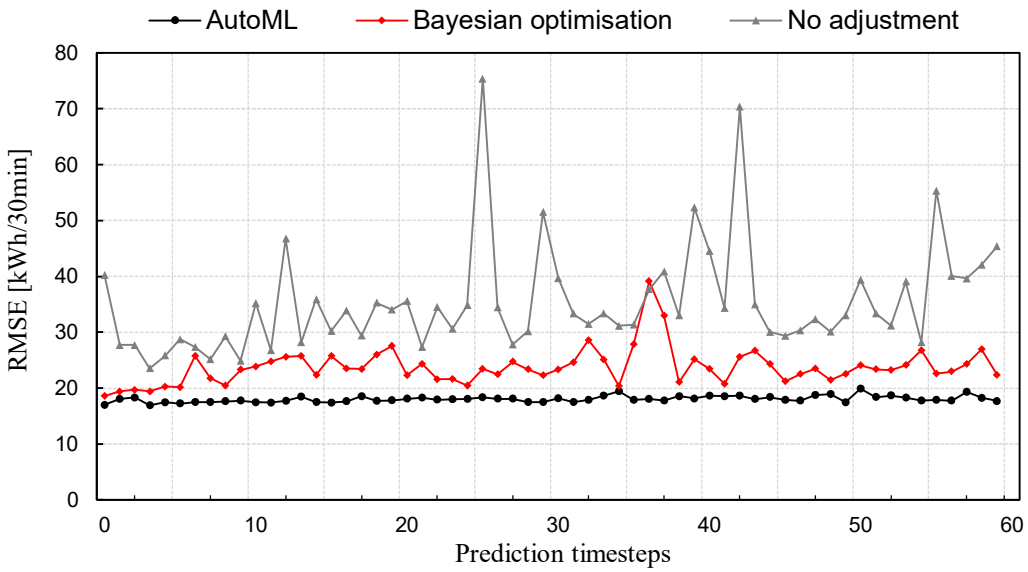


Figure 5. Prediction loss by time step for different trained models.

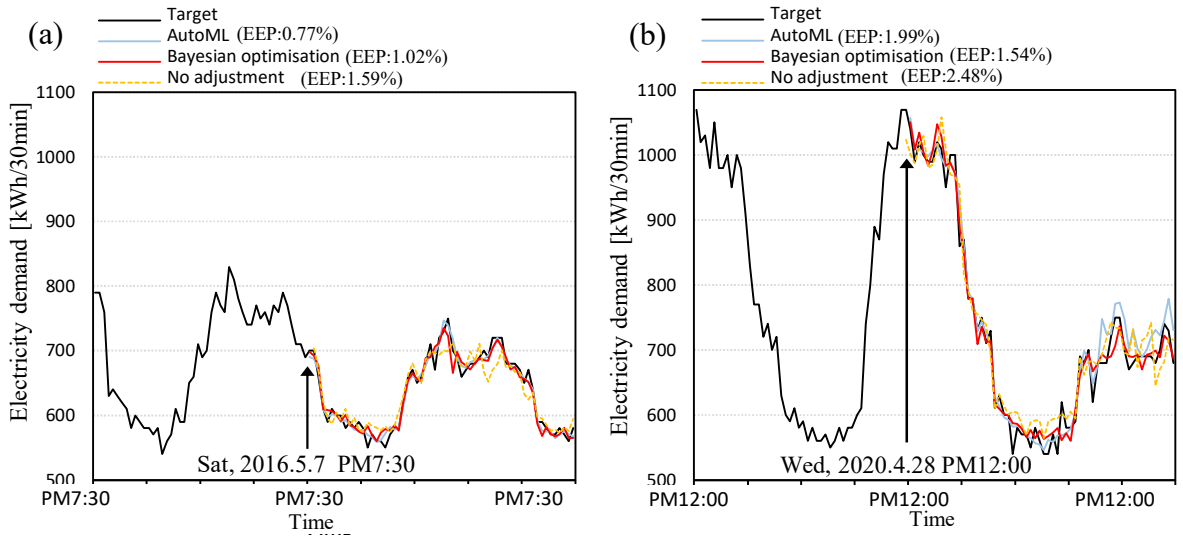


Figure 6. Time series of (a) prediction for weekends and (b) prediction including a holiday.

5. Future work

In this work, the prediction of the first time step was compared across three selected auto-tuning methods. Verifying the accuracy of all methods to predict multiple time steps could help generalize these findings in the future. Further, while, thus far, random search and Bayesian optimization were verified by conducting a hundred search runs, the procedure could be optimized by systematically determining the search times required to achieve stable and accurate predictions. Finally, in the future, we do not only seek to automatically tune the hyperparameters of a specific model but also guide the choice of model structures, for instance, to decide between neural networks and random forests.

6. Conclusion

In this paper, three auto-tuning methods were applied to electricity demand forecasting of a hospital. A single time step ahead, all tuning methods improved the accuracy of the forecast by reducing the RSME to less than 50%, compared to the hyperparameter settings prior to optimization. In addition, Bayesian optimization improved the accuracy for the prediction of multiple time steps ahead, compared to the initial settings, further attesting to the utility of automatic tuning methods. Meanwhile, Auto ML is the most promising method due to its low and stable prediction loss. This reveals the potential for further improvement of the other three auto-tuning.

References

[1] Yoshihiko Ozaki et al, 2020, *Hyperparameter Optimization Methods: Overview and Characteristics (*Transactions of the Institute of Electronics, Information and Communication Engineers, Vol. J103–D) [in Japanese]

[2] Tomoki Nakamaru et al, 2016, *Modification of Initialization Technique for Multilayer Neural Network* ,(SEISAN KENKYU, vol68) [in Japanese]

[3] Dabal Pedamonti et al, 2018, *Comparison of non-linear activation functions for deep neural networks on MNIST classification task ,(arXiv:1804.02763v1)*

[4] James Bergstra et al, Yoshua Bengio, 2012 *Random Search for Hyper-Parameter Optimization* (Journal of Machine Learning Research vol13)

[5] Yuki Nishi et al, 2018, *Research and Development of EMS for the Purpose of Electricity Equalization Part 5 Prediction Accuracy Verification of Unuvariate Prediction System,* (Proceedings of the annual meeting for the Society of the Society of Heating, Air-Conditioning and Sanitary Engineers of Japan,) [in Japanese]