Applied Energy 282 (2021) 116177

Contents lists available at ScienceDirect

Applied Energy

journal homepage: www.elsevier.com/locate/apenergy

Deep learning for load forecasting with smart meter data: Online Adaptive Recurrent Neural Network

Mohammad Navid Fekri [a], Harsh Patel [a], Katarina Grolinger [a,*], Vinay Sharma [b]

[a] Department of Electrical and Computer Engineering, Western University, London, ON N6A 5B9, Canada
[b] London Hydro, London, ON, Canada N6A 4H6

ARTICLE INFO

Keywords:
Online learning
Recurrent Neural Network
Bayesian optimization
Adaptive learning
Concept drift
Online Adaptive RNN

ABSTRACT

Electricity load forecasting has been attracting research and industry attention because of its importance for energy management, infrastructure planning, and budgeting. In recent years, the proliferation of smart meters and other sensors has created new opportunities for sensor-based load forecasting on the building and even individual household level. Machine learning approaches such as Recurrent Neural Networks (RNNs) have shown great successes in load forecasting, but these approaches employ offline learning: they are trained once and miss on the opportunity to learn from newly arriving data. Moreover, they are not well suited for handling the concept drift; for example, their predictive performance will degrade if the load changes due to the installation of new equipment. Consequently, this paper proposes Online Adaptive RNN, an approach for load forecasting capable of continuously learning from newly arriving data and adapting to new patterns. RNN is employed to capture time dependencies while the online aspect is achieved by updating the RNN weights according to new data. The performance is monitored; if it degrades, online tuning is activated to adapt the RNN hyperparameters to changes in data. The proposed approach was evaluated with data from five individual homes: the results show that the proposed approach achieves higher accuracy than the standalone offline long short term memory network and five other online algorithms. Moreover, the time to learn from new samples is only a fraction of the time needed to re-train the offline model.

1. Introduction

Energy production is the largest source of greenhouse gas emissions and about two-thirds of global greenhouse gas emissions are the result of burning fossil fuels for energy production [1]. The environmental impact of energy production is expected to become even more exacerbated as it is estimated that world energy consumption will grow by 28% between 2015 and 2040 [2]. Therefore, efficient energy management will play a crucial role in combating environmental issues and reducing the side effects of energy production. Improved energy management also leads to financial benefits for the end consumers in terms of reduced energy costs and associated operating expenses.

Load forecasting has been attracting remarkable research and industry interest because of its role in infrastructure development (generation and distribution), operation and management of the supply, and energy budget planning. For example, a 1% improvement in a 6 h ahead wind generation forecast leads to savings of 972 thousand dollars over six months [3].

On the other hand, the expansion of smart meters and other sensors has enabled measuring and recording energy consumption on a large scale. Utility companies have been extensively installing smart meters: in 2016, there were over 70 million smart meters in the USA and over 96 million in China, and the number is continuously growing [4]. This large smart meter data created a backbone for new deeper insights into energy usage patterns and forged new opportunities in hourly load forecasting for individual buildings and even individual homes.

In sensor-based forecasting, historical data from smart meters or other sensors are used in combination with meteorological data to infer future energy consumption. One way of doing this is with Machine Learning (ML) techniques; examples include Auto Regressive Moving Average (ARIMA) [5], Bayesian approaches [6], and Support Vector Regression (SVR) [7]. It has been shown that the performance of these shallow models degrades in the presence of highly-varying and non-smooth target signals such as those present in load forecasting [8]. Furthermore, shallow models work well when the training dataset is relatively small while the proliferation of sensors is driving us into the big data era [8].

In recent years, Deep Learning (DL), a sub-field of ML, has been gaining popularity because it can learn feature representations, it has

* Corresponding author.
E-mail address: kgroling@uwo.ca (K. Grolinger).

https://doi.org/10.1016/j.apenergy.2020.116177
Received 9 July 2020; Received in revised form 9 October 2020; Accepted 30 October 2020
Available online 10 November 2020

strong generalization capabilities, and it can model complex relationships commonly present in big data. DL employs representation learning to automatically discover relevant features needed to perform a specific task. Although various DL algorithms such as feed forward neural networks and convolutional neural networks have been applied for energy forecasting, these approaches are not designed to capture time dependencies as they take only the current input to calculate the output. In contrast, Recurrent Neural Networks (RNNs), a class of DL models, are created to model temporal behavior: the output at a time step t depends on the current input and all past inputs. Consequently, RNNs have demonstrated great surceases in load forecasting [9,10,10].

DL techniques, especially RNNs, greatly advanced load forecasting and improved its accuracy; however, several forecasting challenges remain: (1) The conventional offline models are trained once by repeatedly passing all training data through the model; one pass is referred to as *epoch*. Then, the model is used to infer future loads. This approach is missing out on the information that new data could provide. Of course, the model can be re-trained occasionally using all old data together with new data, but this is very computationally expensive as each time, the model is re-trained from scratch. Ideally, the model would learn from new data as they become available, without the need to re-train or to retain old data. (2) The data distributions in energy domain change over time, producing what is known as *concept drift* [11]: for example, installing high-efficiency equipment will reduce energy consumption. In the presence of concept drift, conventional machine learning models experience weak and degrading predictive performance [12].

A different approach in terms of architecture and learning is needed in order to embrace the changes in data, enable the model to adapt itself quickly and capture the new revealing patterns. Online learning has the potential to address these requirements as online models learn from data streams by updating the model as data become available. The data can be discarded after they are consumed by the model. The online models dynamically adapt to new patterns in the data making them well suited for load forecasting.

Consequently, this paper proposes Online Adaptive RNN, a load forecasting approach capable of continuously learning from new data as they arrive. The model adopts online preprocessing techniques to prepare the data for the RNN model, which is responsible for capturing time dependencies. The performance is tracked, and if it starts to deteriorate, a Bayesian tuning mechanism is activated to adjust the model hyperparameters (learning rates) and improve the accuracy. The buffering mechanism is employed to handle especially difficult patterns and to improve forecasting in the presence of concept drift. Results show that the proposed Online Adaptive RNN outperforms other online models as well as the offline RNN.

The proposed Online Adaptive RNN is better suited for the real-world applications of energy forecasting than the traditional batch learning because it does not require periodical re-training and adapts to new patterns quickly. In practice, energy consumption patterns change, and the proposed approach continuously learns from these newly arriving patterns. Moreover, as the re-training on the complete data set is not required, Online Adaptive RNN reduces computational time in comparison to the batch learning approaches.

The remainder of the paper is organized as follows: Section 2 presents the related works, Section 3 discusses the background, Section 4 describes the proposed approach, Section 6 explains the experiments and corresponding results, and finally Section 7 concludes the paper.

2. Related work

This section first reviews recent conventional (offline) ML work for load forecasting and then discusses online ML.

2.1. Conventional load forecasting models

Many approaches have been introduced for energy forecasting problems (e.g., physics, statistics, and machine learning-based) [13] but this section focuses on ML-based models as our work belongs to this category.

Alobaidi et al. [14] proposed an ensemble-based framework to predict day ahead average household consumption. The framework employs Artificial Neural Networks (ANNs) as the base learners and combines them using multiple linear regression. Their results showed improvement in the generalization ability compared to stand-alone ANN and ANN-based bagging ensemble. Singh et al. [5] introduced a hybrid ARIMA–ANN technique for wind power forecasting. The hybrid approach achieved better accuracy than the two models, ARIMA and ANN, working separately. Grolinger et al. [15] introduced a new approach based on local learning with SVR for energy prediction in the big data domain. These authors increased prediction accuracy while reducing computational complexity and training time.

Deep learning algorithms have been popular in load forecasting because of their ability to capture complex patterns in data; specifically, recurrent neural network architectures have been frequently used as they can capture temporal dependencies [10,16,16]. Gao et al. [17] proposed a short-term electricity load forecasting based on an Empirical Mode Decomposition Gated Recurrent Unit with Feature Selection (EMD-GRU-FS). The original series is decomposed into sub-series with empirical mode decomposition and the correlation between sub-series and the original series is determined with the Pearson correlation. Finally, the GRU network is trained with the original series and the sub-series with a high correlation. Bouktif et al. [18] paired standard Long-Short Term Memory (LSTM) with a Genetic Algorithm (GA) for short to medium term aggregate load forecasting. In their approach, LSTM carries out the forecasting while GA is responsible for finding the optimal time lags and the number of layers for LSTM.

Han et al. [19] proposed a prediction model that combines copula function and LSTM network for the estimation of mid-to-long term wind and photovoltaic power generation. First, the copula function is used to extract the key meteorological factors that affect wind and photovoltaic power generation. Then, joint prediction models of wind and photovoltaic power generation based on LSTM performed the forecasting. Sehovac et al. [10] proposed Sequence to Sequence Recurrent Neural Network (S2S RNN) with attention for electrical load forecasting. Their approach adapts the sequence to sequence architecture from language translation to improve time modeling by combining two RNNs: encoder and decoder. The attention mechanism eases the connection between the encoder and the decoder.

Fan et al. [9] examined various deep recurrent neural network strategies for short-term load forecasting. Their results confirmed that RNNs are well suited for short-term forecasting and demonstrated that LSTM cells perform better than vanilla RNNs. Somu et al. [20] also proposed a solution based on LSTM: they improved the forecasting accuracy by tuning LSTM hyperparameters with Improved Sine Cosine Optimization Algorithm (ISCOA).

Although the reviewed models have shown great results in load forecasting, they are all offline approaches: they are trained with a static data set and, to learn from new data, they need to be re-trained. However, the energy consumption data are arriving continuously and new data may have different patterns. To acquire knowledge from new data without re-training, our study proposes an online approach where the model adapts itself to newly arriving data.

2.2. Online machine learning

Online machine learning approaches can be classified into three main categories: optimization-based, model-based, and hybrid approaches. Optimization-based approaches are various extensions of

the gradient descent algorithm to enable updates of the model's parameters as new data become available without changing the model architecture. Defazio et al. [21] introduced SAGA, an incremental gradient-based optimization approach with fast linear convergence rates for non-strongly convex problems. Johnson and Zhang [22] proposed stochastic variance reduced gradient (SVRG), a variance reduction method for Stochastic Gradient Descent (SGD). At each step, SVRG keeps a version of estimated parameters and the average gradient and then uses those values in the update rule to reduce the variance of SGD. This approach achieves fast convergence rates for smooth and strongly convex functions. Stream SAGA (STRSAGA) [23] and Streaming SVRG (SSVRG) [24] are extensions of SAGA and SVRG with improved performance.

The model-based approaches modify the model architecture and/or change the number of parameters to achieve a gradually updated model with faster convergence. Sánchez-Medina et al. [25] proposed adaptive incremental linear regression for wind forecasting. The model learns gradually as new observations arrive and, when concept drift is detected, the older observations are removed from the model. As this approach is grounded on linear regression, this approach is not suited for non-linear problems such as load forecasting. Vexler et al. [26] devised a real-time architecture for energy consumption forecasting by combining LSTM and online density estimation with Hoeffding trees. Liang et al. [27] presented an LSTM-based approach for energy forecasting in the smart grid with the model located at the network edge. As new data arrive, the model is continually trained on the small subsets of arriving data reducing computation and training time. Spiral RNN [28] is an RNN architecture that combines a trainable hidden recurrent layer with the Echo State Neural Network (ESN) for online learning. In experiments, Spiral RNN demonstrated stable performance and fast convergence.

The hybrid models combine techniques from optimization and model-based solutions; they benefit from continuous parameter updates, a modified architecture, and various pre- and post-processing techniques, which leads to a simpler model and faster convergence. Guo et al. [29] presented Weighted Gradient Learning (WG-Learning) for RNNs to learn from online time series in the presence of anomalies and change points. The local properties of the newly available time series data are exploited to weight the gradients. Madireddy et al. [30] proposed a hybrid model combining the two components: an online Bayesian change point detection method to detect the location of the concept drift and a moment-matching transformation technique to convert the data collected before the concept drift to be useful for re-training after the concept drift. Fields et al. [31] investigated the sensitivity of various neural networks to concept drift: flavors of RNN, LSTM, and GRU, were less sensitive than the other types of NNs. Ceci et al. [32] combined the online adaptive training, entropy-based error measure, and spatial autocorrelation for wind power generation forecasting.

Our work compares the proposed Online Adaptive RNN with five common online models: Multi-Layer perceptron (MLP), linear regression, Passive-Aggressive (PA) algorithm, online bagging, and K-Nearest Neighbors (KNN). (1) MLP Regressor [33] learns by incrementally fitting the MLP on batches of samples with SGD as the optimizer. (2) Online linear regression [34] updates the regression weights with SGD in each learning step. (3) Online PA algorithms [35] are a family of online margin-based algorithms: similar to SVR, they aim to maximize the margin. If arriving data are from the same distribution, the algorithm will keep learning, but if the data distribution changes, the weights will slowly forget the previous distribution and learn the new one. (4) The online version of bagging [36] process each data point as it arrives without a need to store it or reprocess while maintaining the current state of the model. (5) The online KNN for regression is a combination of a conventional KNN regression algorithm and the weighted sliding windows.

Reviewed optimization-based approaches are well suited for smooth and convex problems. However, neural network training is a non-convex optimization problem, and the optimizer can get stuck in a spurious local optimum, especially when dealing with complex models such as those common in RNNs [37]. Additionally, these models are not designed to handle concept drift [23].

Model-based approaches also do not consider concept drift except for adaptive incremental linear regression [25] which can handle concept drift but is suitable only for linear problems. As non-linearities are present in energy consumption data, adaptive incremental linear regression is not suitable for load forecasting. In contrast, our solution is non-linear and can handle concept drift.

Reviewed approaches from the hybrid category deal with concept drift, but most are in very different domains such as application performance modeling [30] and network traffic [29]. In contrast, our study specifically considers residential load forecasting for individual households which is a difficult forecasting problem due to high load variability and frequent concept drift. Also, in research studies, concept drift is often simulated [31] while we use real-world data.

Online learning approaches proposed by Vexler et al. [26] and Liang et al. [27] focus on load forecasting. They handle concept drift by transforming the data before feeding the model while our work adjusts the model by tuning the model's parameters as needed. Our online model tuning increases accuracy and improves concept drift handling. The work of Ceci et al. [32] is also in the energy domain: to achieve the model updates, they fully re-train the neural network at the end of each day using all historical data. In contrast, our work updates the model as new data arrive without re-training and without the need to retain historical data.

3. Background

This section introduces RNNs, recurrent batch normalization, and hyperparameter optimization.

3.1. Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed for sequential data such as those found in language translation or load forecasting [38]. The recurrent connection to the same neurons in the previous time step together with their internal state (memory) make RNNs well suited for modeling temporal behavior. RNNs are mainly trained using backpropagation through time; however, for large sequences, this can lead to the vanishing gradient problem causing the NN to forget older information. The Long Short Term Memory (LSTM) networks were designed to overcome this problem, and, consequently, they are able to maintain information for longer periods and make better predictions.

As illustrated in Fig. 1, the LSTM cell contains a cell state c, a hidden state h, , an update step g, and three gates: input i, forget f, and output o. LSTM computation at time t is given as follows:

$$i_t = \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \tag{1a}$$

$$f_t = \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \tag{1b}$$

$$o_t = (W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \tag{1c}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}) \tag{1d}$$

$$h_t = o_t \odot \tanh(c_{t-1}) \tag{1e}$$

Here, σ is the sigmoid activation function, tanh is the hyperbolic tanh activation function, and $\odot$ represents elementwise multiplication. The W_x's and W_h's are the input-hidden and hidden–hidden weights, respectively, and b_x's and b_h' are the corresponding biases.

The LSTM cells are responsible for learning dependencies among the elements in the input sequence. The gates within the cell control how

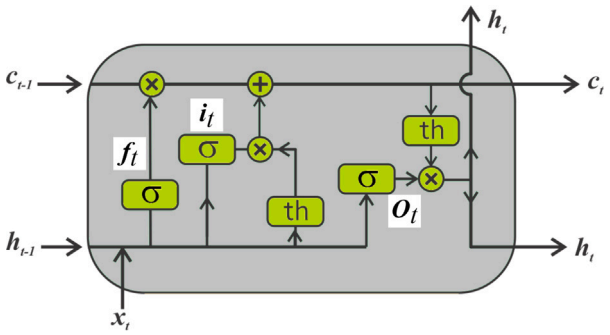


Fig. 1. The LSTM cell.

data flow through the cell and regulate which data should be memorized and which can be forgotten. This memory structure makes LSTMs successful in energy forecasting [10]; however, traditional LSTMs require offline training.

3.2. Recurrent batch normalization

Normalizing the input data for deep neural networks helps the models converge faster. However, this only impacts the input to the first layer while all other layers receive inputs from the previous layers. The distribution of network activations changes during training due to the changes in network parameters. These changes alter the distribution of inputs to the inner layers and slow down the training. This problem is known as *internal covariate shift* [39]. Batch normalization is a mechanism in mini-batch training, which aims to normalize the inputs to inner layers in order to fight the covariate shift problem [39].

Considering a mini-batch B of m samples, the batch normalization is applied to each input dimension x_i independently. The batch normalizing transform starts as follows:

$$\bar{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2}$$

where $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ and $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ are the mean and variance for that dimension, and ϵ is a small constant added for stability.

Next, the input is scaled and shifted as follows:

$$y_i = \gamma \bar{x_i} + \beta \tag{3}$$

where γ and β are parameters learned during training along with other parameters of the network.

Cooijmans et al. [40] proposed a reparameterization of LSTM that brings the benefits of batch normalization to the recurrent neural networks. They demonstrated that the Batch-Normalized RNNs (BN-RNNs) lead to faster convergence and improved generalization. The batch-normalizing transform BN(·; γ, β) is introduced into the LSTM as follows:

$$\begin{pmatrix} f_t \\ i_t \\ o_t \\ g_t \end{pmatrix} = BN(W_h h_{t-1}; \gamma_h, \beta_h) + BN(W_x x_t; \gamma_h, \beta_h) \tag{4}$$

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t) \tag{5}$$

$$h_t = \sigma(o_t) \odot tanh(BN(c_t; \gamma_c, \beta_c)) \tag{6}$$

where BN(.) is the transformation introduced with Eqs. (2) and (3). The recurrent $W_h h_{t-1}$ and the input $W_x x_t$ terms are normalized separately. To preserve LSTM dynamics, the normalization is not applied to the cell update c_t. During training, the mean and the variance are calculated independently for each batch, and at test time, the average of the estimates over the training set is used.

3.3. Hyperparameter optimization

Hyperparameter optimization aims to find a combination of hyperparameters that leads to the optimal ML model performance. It has been shown that tuning hyperparameters plays a major role in the ML model accuracy [41]. In general, the hyperparameter optimization can be represented in an equation form as:

$$x_{best} = argmin f(x) | x \in X \tag{7}$$

where $f(x)$ represents a score that should be minimized, the X is the domain of the hyperparameter values, and x_{best} is a combination of hyperparameters that yields the lowest value of the score $f(x)$.

Finding optimal hyperparameters manually is challenging and computationally expensive, especially in a case of complex models such as neural networks. Grid search and random search are slightly better, but they are unaware of the model's past evaluations, which results in long tuning time and often leads to a sub-optimal set of hyperparameters.

In contrast to those approaches, Bayesian optimization [42] (BO) keeps track of the past evaluations to form a probabilistic model for mapping hyperparameters to a probability P of an objective function score:

$$P(score | hyperparameters) \tag{8}$$

This probability model is referred to as the *"surrogate"* for the objective function. The surrogate is easier to optimize than the actual objective function; thus, BO searches for hyperparameters using the surrogate. BO process is described in Algorithm 1. First, the probabilistic model P_{model} is initialized with a Gaussian process prior on $f(.)$ [42]. Then, in each iteration, the best set x_{best} is found for the current probabilistic model P_{model}, the model *score* for that set x_{best} is determined, and the P_{model} is updated.

Algorithm 1 Bayesian Hyperparameter Optimization

1: $Pmodel \leftarrow$ Surrogate*(f(x))*
2: $x_{best} \leftarrow \{\}$
3: **while** $i < maxIterations$ **do**
4: $\quad x_{best} \leftarrow Pmodel(score, hyperparameters)$
5: $\quad score \leftarrow f(x_{best})$
6: $\quad Pmodel \leftarrow Update(Pmodel, score)$

Similar to the described Gaussian process-based optimization, the Tree-structured Parzen Estimator (TPE) also constructs models to approximate the performance of hyperparameters based on historical measurements, and then subsequently chooses the new hyperparameters to test based on this approximate model [42]. However, while Gaussian approach estimates the probabilities directly, TPE estimates them indirectly. Since TPE achieved better accuracy than the Gaussian process-based BO, TPE is used in our study.

4. Online adaptive RNN

This section presents Online Adaptive RNN, a load forecasting system that dynamically learns from continuously arriving data and adapts to new patterns in the data. The approach uses batch-normalized RNN (BNRNN) as the base learner and combines Bayesian optimization, performance monitoring, and buffering to tune the BNRNN model on the fly. Online Adaptive RNN is depicted in Fig. 2 and Algorithm 2, while details of each component are described in the following subsections.

4.1. Prepossessing module

As over time data from smart meters or other sensors become available, they are passed to the preprocessing module that transforms them into a suitable form for RNNs. These continuously arriving data are represented in line 3, Algorithm 2. The preprocessing consists of two components: the sliding window and online normalization.

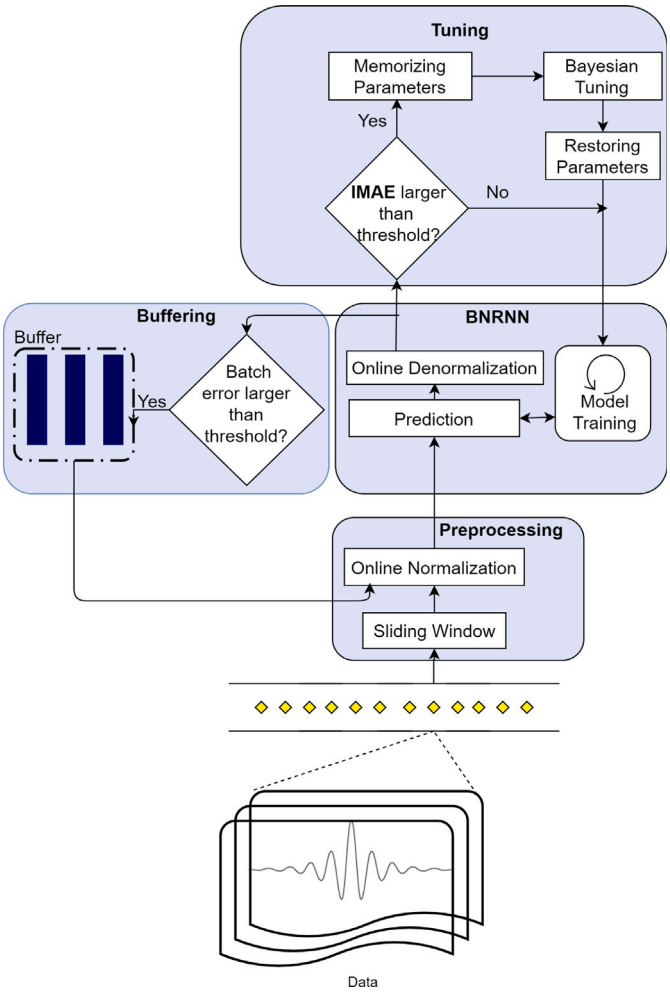


Fig. 2. Online Adaptive RNN: components and the processing flow.

Algorithm 2 Online Adaptive RNN

1: **Input** : *Data* : D, *Hyperparameter Search Space* : S, *Early Stopping Size* : Δ, *MaxEpochs* : N
// Initialize the weights w and learning rate η
2: **Initialization** : $w = w_0, \eta = \eta_0$
3: **while** data are available **do**
4: $B \leftarrow SlidingWindow(D, batch\ size, window\ size)$
5: $B_F \leftarrow$ Get data from Buffer
// Merge current batch and buffer data
6: Q $\leftarrow$ B $\cup$ B_F
7: $Q_N \leftarrow IncrementalMinMaxNormalization(Q)$
// Make prediction with BNRNN
8: $Predicted \leftarrow$ BNRNN(Q_N, w, η)
9: $Predicted \leftarrow De\text{-}normalize(Predicted)$
// Calculate MAE and compare with threshold
10: **if** $MAE(Predicted_{prev}, Q) > tuningThresh$ **then**
11: Store current batch B in the Buffer
// Calculate IMAE, b is the current batch index
12: $IMAE_b = \frac{IMAE_{b-1} + MAE_b}{b}$
13: **if** $IMAE > bufferingThresh$ **then**
14: Memorize Weights
// Tune learning rate η
15: $\eta \leftarrow BO(w_{t-1}, S, Optimizer, Q)$
16: Restore Weights
17: **for** $t = 1, 2, 3, ..., N$ **do**
// Train with Q_N and new learning rate η
18: $w_t \leftarrow Train(Q_N, w_{t-1}, \eta)$
19: $loss_t \leftarrow TrainLoss(w_t, \eta)$
20: **if** $exp(loss_t - loss_{t-\Delta}) > u$ **then** //Check trend
21: *Break*

4.1.1. Sliding window

The sliding window technique (Algorithm 2, line 4) is illustrated in Fig. 3. The first W readings correspond to the first window and make the first training sample. Then, the window slides for S steps, and the readings from the time step S to $S + W$ make the second sample and so on. Each sample is a matrix of dimension $W \times F$, where W is the window length and F is the number of features. With S < W, there is an overlap between the sliding windows, and a reading from a single time step belongs to multiple windows.

Next, bn consecutive samples generated by the sliding windows technique are placed in a group referred to as the *batch*. Once the batch is created, the data move through the remaining modules of Online Adaptive RNN one batch at the time and the learning takes place one batch at the time.

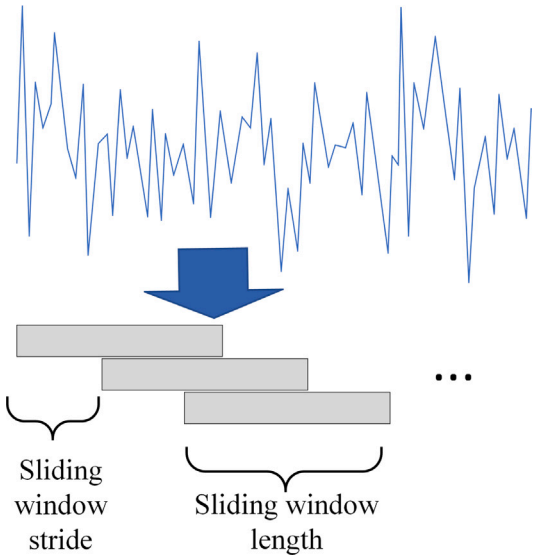


Fig. 3. Sliding window technique.

4.1.2. Normalization

Normalization is a standard preprocessing technique for bringing the values of all features to a common scale with the objective of reducing large feature dominance and improving convergence. Min-Max normalization is a strategy which linearly transforms X to X' as follows:

$$x' = \frac{x - Min(x)}{Max(x) - Min(x)} \tag{9}$$

where x is the original feature value, Min(x) and Max(x) are the minimum and maximum of that feature, and x' is the normalized value.

In offline learning, all training data are available before the training starts, hence, the normalization can be performed using minimum and maximum values of the complete training set. In the online setting, data must be processed as they arrive and the compete training set is not available as the training starts; thus, minimum and maximum values cannot be calculated in the same way.

To address this challenge, the proposed approach carries out *Incremental Min-Max Normalization*. In the main Online Adaptive RNN Algorithm 2, line 7 performs this normalization while the details are presented in Algorithm 3. The maximum and minimum values of the

features from the beginning until the current batch are tracked with $globalMax$ and $globalMin$. For each new batch, the procedure finds the max and min values for that batch as shown in Algorithm 3, lines 2 and 3. If the batch max is larger than the current global maximum $globalMax$ (Algorithm 3, line 4, the $globalMax$ is updated (line 5). The same process happens for $globalMin$, lines 6 and 7. Finally, the values from the current batch are normalized using the current $globalMax$ and $globalMin$: Algorithm 3, line 8.

Algorithm 3 Incremental Min-Max Normalization

```
1: while next batch B is available do
2:     max : Maximum value for B
3:     min : Minimum value for B
4:     if max > globalMax then
5:         globalMax = max
6:     if min < globalMin then
7:         globalMin = min
8:     Normalized_B =
       MinMaxNormal(B, globalMax, globalMin)
```

4.2. Batch normalized RNN

The RNN was selected as the core learner because of its ability to model temporal dependencies present in the load data. To handle internal covariate shift and reduce training time, batch normalization described in Section 3.2 is used. Batch normalization also reduces sensitivity to changes in the learning rate and, consequently, assists the tuning module in dynamically adjusting the learning rate to better capture new data. The BNRNN module consists of three components: prediction, online de-normalization, and model training.

4.2.1. Prediction

When a new batch is preprocessed and passed to the BNRNN, the buffer is checked for the data availability (Algorithm 2, line 5). If there are batches in the buffer, those batches are merged with the current batch (line 6) and normalized (line 7). Next, BNRNN makes the predictions as shown in Algorithm 2, line 8. For each sample (window of length W) within a single batch, the model predicts the load for the next p time steps. At the start of the online learning with the first batch, the buffer is empty and the predictions are poor as the BNRNN is just initialized and will start to learn from this first batch.

4.2.2. Online de-normalization

As the data are normalized before being passed to the BNRNN, the outputs of the BNRNN model, electricity load values, are between 0 and 1. These predicted load values must be transformed back to the original domain to obtain the final predicted load and enable the comparison with the actual values for error evaluation in the following steps. The BNRNN outputs are de-normalized (Algorithm 2 line 9) as follows:

$$y' = (PredictedValue * (globalMax[load] - globalMin[load])) + globalMin[load] \quad (10)$$

where y' is the de-normalized output, $PredictedValue$ is the output of the BNRNN model, $globalMin[load]$ and $globalMin[load]$ are the global maximum and minimum values for the load feature, which were determined during the online normalization step (Section 4.1.2). These de-normalized values are passed to the tuning module for further processing.

4.2.3. Model training

The model is always trained only on the current batch and the batches from the buffer. When the batch is consumed once for training, it is discarded unless the buffering module determines that it should be stored in the buffer. As indicated in Algorithm 2, line 17, the training is repeated for up to N epochs. The model is trained with the learning rate obtained from the tuning module (Algorithm 2, line 18): this training results in updated weights w. The training loss for the current epoch is determined in line 19, Algorithm 2.

Next, the accuracy trend is examined (Algorithm 2, line 20) to determine if the training should continue. This prevents the algorithm from overfitting and reduces the training time. As the training here happens only with the current batch and any batches from the buffer, it is important to stop the training before the model fits the current data too closely and forgets the patterns previously learned from other batches. To do this, the loss at the current epoch t is compared to the loss at epoch $t - \Delta$. If the exponential function of this loss difference between epochs t and $t - \Delta$ is greater then the small constant u, the training stops (Algorithm 2, line 20).

4.3. Buffering module

The purpose of the buffering module is to identify and temporally store batches, where the model could not perform well. This module helps Online Adaptive RNN in terms of generalization and impedes it from being biased towards more repetitive and easy to learn batches. As the system employs online learning, the model has only one pass over each batch and, with the arrival of new batches, the model performance on less repetitive patterns degrades. The buffering module assists with this by temporally storing challenging batches. As the buffer only stores a small number of batches at a time, it is maintained in the memory.

This module is also important in the presence of concept drift as it enables the model to repeatedly see the batches with drift; consequently, the model accuracy in presence of concept drift is improved. Note that the knowledge is retained in the weights of the neural network, and the buffer only assists the model by enabling it to see difficult patterns more than once.

The buffering mechanism starts with determining the Mean Absolute Error (MAE) for the current batch: the actual load values from the current batch are compared with the prediction values obtained in the previous step (Algorithm 2, line 10). If the MAE is higher than the buffering threshold, the batch is considered challenging and thus, the batch data with the corresponding MAE are stored in the buffer (Algorithm 2, line 11).

The buffer size is limited and with the presence of concept drift in the data, the buffer is expected to fill up quickly. If the buffer is full, the batch in the buffer with the lowest error is replaced by the incoming batch. This ensures that the BNRNN sees the challenging batches in several training iterations.

Repeatedly training the model on the batches that have been in the buffer for a long time may cause performance deterioration as an old batch may become irrelevant due to concept drift or other changes in the data. Therefore, batches are removed from the buffer upon expiration of the preset lifespan.

4.4. Tuning module

The tuning module is a crucial element of the proposed Online Adaptive RNN as it adapts models hyperparameters to new data. As already mentioned, well selected hyperparameters are essential for achieving highly accurate deep learning models; however, offline hyperparameter tuning requires several passes over a complete training dataset and, therefore, cannot be applied in the online setting. Nevertheless, due to drastic changes in load data including concept drifts, the hyperparameters still need to be tuned as new data arrive.

For online learning, tuning structural parameters such as the number of layers and the hidden layer size is not suitable because such changes add new network weights and, thus, require complete re-training. However, other parameters, such as the learning rate and batch size, do not require re-training since they do not change the architecture and, therefore, the weights representing the acquired knowledge can be re-used after tuning.

Tuning after each sample or even after each batch is computationally expensive and time consuming. Consequently, Online Adaptive RNN uses *Incremental MAE (IMAE)* to determine if the model needs to be tuned. The IMAE Error is the average error over the batches since the beginning of training and is updated as new batches arrive. It is calculated as follows:

$$IMAE_b = \frac{IMAE_{b-1} + MAE_b}{b} \tag{11}$$

where b is the current batch index, MAE_b is the MAE error for the batch b, and $IMAE_{b-1}$ is the IMEA after the batch $b-1$. This evaluation occurs after the actual load value for batch b are available.

As shown in Fig. 2, if the IMAE is not over the tuning threshold (Algorithm 2, line 13), the BNRNN is trained with the current batch without the learning rate change. If the IMAE is over the tuning threshold (Algorithm 2, line 13), the Bayesian optimizer (BO) is activated to find a new learning rate for the model (Algorithm 2, lines 14 to 16). Here only the learning rate is considered for the tuning, as it has been shown that the learning rate is the most important RNN parameter [43]; however, other non-structural parameters could be tuned using the same approach.

If tuning is required, as illustrated in Fig. 2, the weights (parameters) are first preserved (Algorithm 2, line 14) so that they can be restored after the BO process. In offline learning, this preserving is not needed as the optimizer can use all data in all iterations: the BO initializes the model weights with random values every time it evaluates the model hyperparameters causing the model to forget what it has previously learned. In offline learning, this is not a problem as the model will re-learn in the next pass over the same data. In online learning, the model has the access only to a small data segments (batches) at a time, and as parameters are representations of what the model has learned, parameters cannot be forgotten. Because weights represent what the model has learned so far, they are memorized in the parameter preservation step so that they can be restored after the BO changes them during tuning.

Next, Bayesian learning is carried with the current batch and batches from the buffer to find the new learning rate (Algorithm 2, line 15). When the new learning rate is determined, the weights are restored to their values before BO (Algorithm 2, line 16). Finally, the new learning rate is passed to the BNRNN, and the training is carried out with the new learning rate and the current batch. Note that the old batches, except for those from the buffer, are not reused. This is possible because the knowledge from old buffers is contained in the restored weights. With the next batch, this newly trained model is used for prediction.

5. Evaluation methodology

The metrics and the design of experiments for assessing the quality of online learning models are more challenging than those for the offline models because (1) the data are continuously arriving, (2) models evolve over time rather than being static, and (3) the data may be from non-stationary distributions instead of stationary ones (concept drift) [44]. Two possible ways of evaluating online models are holdout and prequential methodologies.

Holdout [44] for online learning is a periodic evaluation method in which a static test set is created from unseen samples throughout data stream as illustrated in Fig. 4a. The model is trained on batches or samples as they arrive and evaluated on the test set at regular intervals expressed in terms of time, batches, or samples. For example, after every b batches, the model is evaluated on the test set. The holdout error H_e at the time step i, for the current number of the test set samples M is calculated as:

$$H_e(i) = \frac{1}{M}\sum_{k=1}^{M} L(y_k, \hat{y}_k) = \frac{1}{M}\sum_{k=1}^{M} e_k. \tag{12}$$

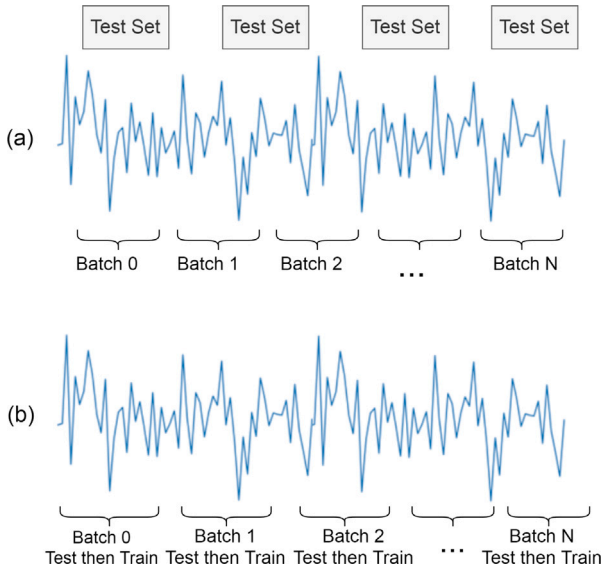


Fig. 4. Online evaluation: (a) holdout evaluation (b) prequential evaluation.

where y_k is the target value, $\hat{y}_k$ is the predicted value, L is a loss function, and e_k is the error.

The main drawback of the holdout method comes from the concept drift presence in the data [45]. If the data contains time-evolving concepts, using a static test set to evaluate the model will not provide a good error estimate as the error will change if a different test set is selected.

Predictive Sequential (Prequential) [44] evaluation, as illustrated in Fig. 4b, is an interleaved test-then-train method in which each sample serves two purposes: test and train purposes. As a sample arrives, the model is first tested with this new sample as the input and the error is calculated. Next, the model is trained on this sample. The prequential error P_e, at time i, is calculated as:

$$P_e(i) = \frac{1}{i}\sum_{k=1}^{i} L(y_k, \hat{y}_k) = \frac{1}{i}\sum_{k=1}^{i} e_k. \tag{13}$$

where y_k, $\hat{y}_k$, and L are the same as in Eq. (12). Note that this is the same as Eq. (12) for holdout error; the only difference is that here the summation is over all samples up to sample i, and in a holdout, it only includes the sample from the test set. By including all samples in the error evaluation, the prequential approach avoids test set selection bias.

The prequential approach has the advantage over holdout as it does not depend on the test set selection, and it provides more reasonable error estimates in the presence of concept drift; therefore, we use it in this paper for a comparison of the proposed Online Adaptive RNN with other online approaches.

However, this approach is not suitable for comparing online and offline learning algorithms. The prequential evaluation involves interleaved test-then-train employing all data points for both train and test purposes, which is not applicable for offline learning when training is repeated several times over the whole training data set. Offline learning evaluation requires a separation between data used for training and testing. The holdout approach could be used for offline approaches, but it is not well suited for online approach-es, especially in the presence of concept drift, as already mentioned. Consequently, to compare online and offline approaches, we propose Prequential-Holdout.

Prequential-Holdout technique combines offline holdout and online prequential techniques for the comparison of online and offline models as shown in Fig. 5. The evaluation process consists of the following steps:

Fig. 5. Prequential-Holdout evaluation.

- The dataset is divided into the training set and test set. Last k samples belong to the test set and the rest make up the training set.
- The online model is trained on the train set by one pass over the data and is evaluated on the test set by applying the prequential method within the test set.
- The offline model is evaluated using a traditional holdout in which the model is trained on the training set and evaluated on the test set.

This way, both online and offline approaches are evaluated on the same samples. As the offline approaches have the advantage of doing several passes over the training set, the online approaches may not be able to achieve comparable accuracy as they only have one pass over the training data. However, the online approaches continue to learn on the test set and, thus, should be better if concept drift is present in the test portion of the data.

To compare online models, we use the prequential technique and to compare between Online Adaptive RNN and offline models, we use prequential-holdout technique. The metrics applied with both techniques are the Mean Square Error (MSE) and the Mean Absolute Error (MAE):

$$\text{MSE} = \frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2 \tag{14}$$

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t| \tag{15}$$

where N is the number of sampled in the test set, and y_k, and $\hat{y}_k$ are the same as in Eq. (12).

6. Evaluation

This section first introduces the data set and presents the preliminary analysis. Next, the proposed Online Adaptive RNN is compared with offline LSTM and with five other online learning algorithms. Then, the impact of different modules in Online Adaptive RNN is examined, the effect or the buffer size is evaluated, and training time is analyzed. Finally, findings are discussed.

6.1. Dataset and preliminary analysis

The proposed approach was evaluated on the real-world data from five residential consumers provided by London Hydro, a local electrical distribution utility involved with this project. Data was obtained through Green Button Connect My Data (CMD) environment, the first cloud-based CDM platform London Hydro developed to provide secured data sharing with the customer's consent. Each household dataset contained three years of smart meter data in one-hour intervals for a total of 25,559 readings. Each reading includes energy consumption and the corresponding date and time. As this data from smart is also used for billing, the high quality is expected. Meteorological information was added including temperature, wind speed and direction, pressure, and humidity. To assist with handling weekly and daily patterns, additional features were extracted from reading date/time including the day of the week and hour of the day. After these additions, the data set consisted of 12 features including five meteorological (temperature, wind speed, wind direction, pressure, humidity), six temporal (month, day of the year, hour of the day, week number, day of the week, season), and the target feature hourly load.

To examine the temporal characteristics of the datasets, two preliminary analyses were conducted: stationarity and concept drift analysis.

6.1.1. Stationarity analysis

A non-stationary time series changes properties over time and, therefore, imposes difficulties for load forecasting. To evaluate if the series are stationary, Augmented Dickey Fuller (ADF) and Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests were conducted.

ADF determines if the series is stationary or not by observing the presence of a unit root. The null hypothesis for this test is: The series has a unit root - it is non-stationary. In the ADF test, if the test statistic value is less than the critical value (CV), the null hypothesis is rejected, which means that the series is stationary.

KPSS determines if a time series is stationary (or not) around a deterministic trend (trend stationery). The null hypothesis is: The series is trend stationary. In the KPSS test, if the test statistic is greater than the critical value, the null hypothesis is rejected indicating that the series is not stationary.

Table 1 shows the ADF and KPSS test results for the load from the five homes. The ADF test statistics are lower than the critical values for all home and all significance levels α, indicating that the series are stationary. The KPSS test statistics for all home are greater than the critical values, indicating that the series are non-stationary. The ADT test only checks for one type of non-stationarity, a unit root non-stationarity, which is a possible reason for different results from the two tests. Nevertheless, the possible presence of non-stationarity makes the modeling more difficult and imposes challenges on load forecasting.

6.1.2. Concept drift analysis

The major advantage of the online models in load forecasting is their ability to adapt to changes in data patterns. Thus, the data sets from the five houses are first examined for the presence of concept drift. Three concept drift detection methods have been applied Adaptive Windowing (ADWIN) [46], Page–Hinkley (PH) [47], and Drift Detection (DDM) [48]:

- **ADWIN** method uses sliding windows of variable size according to the changes observed from the data. If the difference between the statistics observed in the windows surpasses the threshold, concept drift is detected.
- **PH** method continuously monitors the difference between the time series values and the current mean. The difference greater than the threshold indicates the concept drift.
- **DDM** is based on the idea that as long as the data distribution is stationary, the learner's error rate does not increase. This approach monitors the online error of the algorithm, and when this error increases, concept drift is detected.

Table 1
ADF and KPSS tests.

Dataset	ADF				KPSS				
	Test statistic	CV			Test statistic	CV			
		$\alpha = 1\%$	$\alpha = 5\%$	$\alpha = 10\%$		$\alpha = 10\%$	$\alpha = 5\%$	$\alpha = 2.5\%$	$\alpha = 1\%$
House 1 load	−10.3306	−3.4306	−2.8616	−2.5668	1.1650	0.3470	0.4630	0.5740	0.7390
House 2 load	−11.8989	−3.4306	−2.8616	−2.5668	3.0379	0.3470	0.4630	0.5740	0.7390
House 3 load	−13.2413	−3.4306	−2.8616	−2.5668	14.9290	0.3470	0.4630	0.5740	0.7390
House 4 load	−11.9328	−3.4306	−2.8616	−2.5668	11.6656	0.3470	0.4630	0.5740	0.7390
House 5 load	−10.3135	−3.4306	−2.8616	−2.5668	4.6686	0.3470	0.4630	0.5740	0.7390

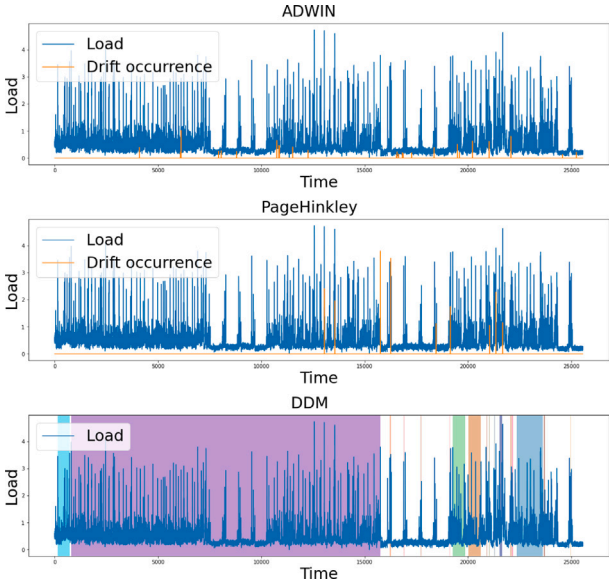


Fig. 6. The electricity load and the concept drift occurrences detected with ADWIN, PH, and DMM: house 1.

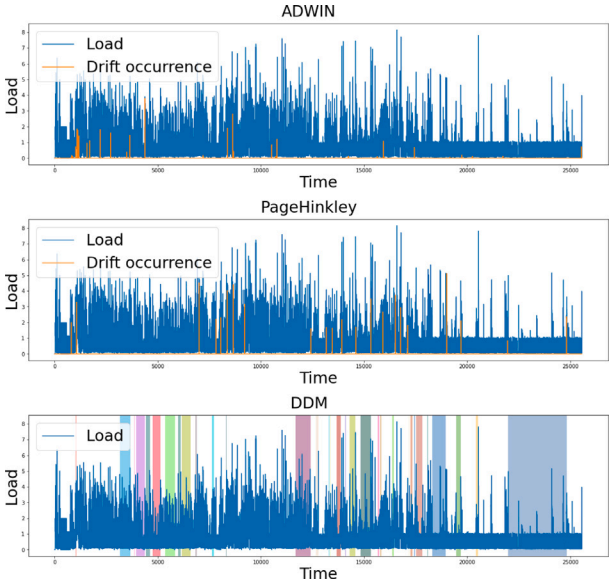


Fig. 7. The electricity load and the concept drift occurrences detected with ADWIN, PH, and DMM: house 4.

Figs. 6 and 7 depict the results of the three aforementioned concept drift detection methods for homes one and four. In ADWIN and PH graphs, the vertical lines indicate the points in which concept drifts have taken place. In the DDM graph, the colorful sections indicate the data with similar distributions and statistics, and, therefore, different sections show different concept drifts occurrences.

It can be observed that the three algorithms detect a different number of concept drift occurrences at different locations in the time series. Nevertheless, all algorithms indicate the extensive presence of concept drift, and changes in load patterns are notable even from the visual observations of the loads. The existence of concept drift is especially pronounced in house four with the DDM method: a high number of diversely colored segments indicate different distributions thought the series. Such a high presence of concept drift in load data hugely degrades the accuracy of the offline models and imposes challenges for online models as they need to continuously adapt to new patterns. Consequently, for household-level load forecasting, it is critical to use online models capable of handling such changes.

6.2. Comparison with offline LSTM

This subsection compares the proposed Online Adaptive RNN with conventional offline LSTM. Online and offline approaches are intrinsically different and cannot be directly compared; however, it is still important to compare them as both are used for load forecasting. The comparison is carried out here with the Prequential-Holdout approach proposed in Section 5: the first 70% of data was used for training and the last 30% for testing. This remainder of this subsection first describes experiments and then presents results.

6.2.1. Experiments

In Online Adaptive RNN, the core learner, BNRNN, can be implemented with different recurrent cells: experiment presented here used LSTM cells as these cells can handle longer sequences than vanilla RNN cells. As the online model employed LSTM cells, the comparison offline model is also an LSTM network. Both Online Adaptive RNN and offline LSTM were tuned with BO: while offline LSTM applies the traditional BO tuning process, Online Adaptive RNN employs BO as described in Section 4.4. The following ranges of hyperparameters were considered for offline LSTM:

- Hidden layer sizes: 64, 128, 256, 512, 1024
- Number of layers: 1, 2
- Batch sizes: 5, 10, 25, 50, 100, 200, 250
- Learning rates: continuous from 0.000001, 0.2

In Online Adaptive RNN, as described in Section 4.4, the BO tuning happens online; therefore, only the learning rate was tuned. The batch size was 5, and the buffer size was 10 batches for all Online Adaptive RNN experiments. Further tuning the batch size and buffer size could potentially improve the accuracy, but would incur extensive computation cost. Moreover, a very large buffer would make the proposed approach similar to the offline model as many samples would be reused in each training step. Thresholds for tuning and buffering modules were tuned for individual households.

The load forecasting accuracy is highly dependent on the forecasting horizons: shorter horizons typically lead to higher accuracy. Moreover, the model's forecasting accuracy on one horizon does not necessarily directly translate to its performance on another horizon. Consequently, Online Adaptive RNN and offline LSTM were evaluated for four load

forecasting horizons: 1 h ahead, 50 h ahead, 100 h ahead, and 200 h ahead.

6.2.2. Results

The offline model has a great advantage of several passes over the complete test set, while with online approaches, data are discarded once processed by the model. These multiple passes over data can lead to higher accuracy of the offline models. However, the online models do not require storing all data, nor do they require re-training to capture new patterns, which makes online models more desirable. Consequently, we consider the online model successful when it archives similar or higher accuracy than the offline LSTM.

Fig. 8 compares the accuracy of Online Adaptive RNN and the traditional LSTM in terms of MAE and MSE for the four prediction horizons. As the scale of the errors for the LSTM and Online Adaptive RNN are different, the figure has two vertical axes: the left one is for Online Adaptive RNN and the right one is for the offline LSTM. Moreover, the accuracy values are displayed with each data point. Although the offline LSTM hyperparameters were tuned, the MAE and MSE errors are relatively high. The main reason for this is the presence of the concept drift. Data distribution and load patterns change over time, resulting in the low predictive power of the static offline model. The accuracy of the online approach is much higher as the model adapts to the changes in data patterns as they arrive.

From Fig. 8, it can be observed that the accuracy varies greatly among houses, prediction horizons, as well as between measures (MAE and MSE). For offline LSTM, house 5 consistently has the highest forecasting errors for all horizons and both metrics. Similarly, houses 1 and 3 have the lowest errors. In the case of Online Adaptive RNN, MSE for house 3 is lower than for other houses, while in terms of MAE, this house is among two with the lowest error. Possible reasons for this are fewer or weaker concept drifts in the house 3 data.

6.3. Comparison with online models

This subsection compares the proposed approach with the five standard online models: MLP, linear regression, passive-aggressive, bagging, and KNN regression. As all compared models are online, prequential evaluation described in Section 5 is used.

6.3.1. Experiments

For Online Adaptive RNN, the same setup and tuning have been applied as described in Section 6.2.1 for the comparison with offline models. To keep the comparison fair, parameters for the five competition models were also tuned with BO. The following configurations were considered for tuning:

- Online MLP regression: SGD and Adam optimizers, the hidden layer sizes of 50, 100, 250, 500 neurons.
- Online linear regression: SGD and Adam optimizers, the learning rate in a range of [0.000001,0.2].
- Online KNN: the window sizes of 25, 50, and 100, and the number of neighbors 3, 5, and 10.
- Online bagging regression: a linear regression as the base model, and the number of the base models 5 and 10.
- Online passive-aggressive regression: Epsilon insensitive loss function, aggressiveness in a range of [0.01, 10].

The same forecasting horizons were examined as in comparison with offline models: 1, 50, 100, and 200 h ahead.

6.3.2. Results

Fig. 9 shows the results of the experiments in terms of MAE and MSE for different prediction horizons. It can be observed that for one hour ahead prediction, Fig. 9a and b, the lowest error was achieved with the online KNN for all households. The KNN predicts based on the similarity of the target to its nearest neighbors and, for one hour ahead prediction, the training data points, and the target are adjacent enough for the KNN to perform well. Nevertheless, the accuracy of Online Adaptive RNN is very close to the KNN.

When the prediction length is increased to 50 h, the adjacency level of the training data points becomes weaker especially in the presence of concept drifts. For this reason, the error rates for the MLP, Linear, and PA regression models increased significantly as shown in 9c and d. Errors for the KNN and bagging models slightly decreased in comparison to the one hour ahead prediction. For this prediction length, Online Adaptive RNN achieved better accuracy than the remaining five algorithms for all homes.

For 100 h ahead, Fig. 9e and f, the proposed Online Adaptive RNN remains better than the other models. For 200 h ahead Online Adaptive RNN outperforms others for all but one household: for house 1, bagging achieves slightly better accuracy than Online Adaptive RNN. However, bagging performs very poorly on household 4 making Online Adaptive RNN an overall better model.

Fig. 9 compared the average error for the considered algorithms, but we are also interested in the variability of the error as a model with more consistent prediction is more desirable. Consequently, Fig. 10 examines error intervals for Online Adaptive RNN and the five online algorithms. The error intervals have been calculated based on the error for all households and average, minimum, maximum, first and the third quartile are shown in the figure.

As in Fig. 9, in Fig. 10, for 1 h ahead KNN achieved the best average accuracy. From Fig. 10 a and b, it can also be observed that error ranges for KNN are among the lowest. Again, performance of Online Adaptive RNN is close to KNN.

For 50, 100, and 200 h ahead, Online Adaptive RNN achieved the best accuracy with small error intervals. In terms of MSE, the error interval for Online Adaptive RNN is visibly smaller than for the other approaches. The box plot reveals the drawback of some approaches: for example, bagging achieved a similar accuracy to Online Adaptive RNN for 200 h ahead forecasting, but it has a large error range, which makes bagging an undesirable solution. Overall, Online Adaptive RNN achieves lower accuracy than the other online models for 50 or more hours ahead prediction and it exhibits low error variability indicating the model consistency.

6.4. Analysis of modules impact

Experiments presented so far compare the performance of Online Adaptive RNN with other models, offline LSTM, and five online models. This section examines the impact of the modules within Online Adaptive RNN on the forecasting accuracy; specifically, the impact of buffering and tuning modules is investigated. As different variants of the proposed online approach are compared, the prequential evaluation described in Section 5 is used.

6.4.1. Experiments

Experiments compare four Online Adaptive RNN variants:

- with buffering and tuning modules
- only with buffering module
- only with tuning module
- without buffering and tuning modules

The same setup and tuning have been applied as described in Section 6.2.1; however, note that for the models without the tuning module, there is no tuning. Again, the same forecasting horizons are considered: 1, 50, 100, and 200 h ahead.

(a) The MAE error for 1 hour ahead.
(b) The MSE error for 1 hour ahead.
(c) The MAE error for 50 hours ahead.
(d) The MSE error for 50 hours ahead.
(e) The MAE error for 100 hours ahead.
(f) The MSE error for 100 hours ahead.
(g) The MAE error for 200 hours ahead.
(h) The MSE error for 200 hours ahead.

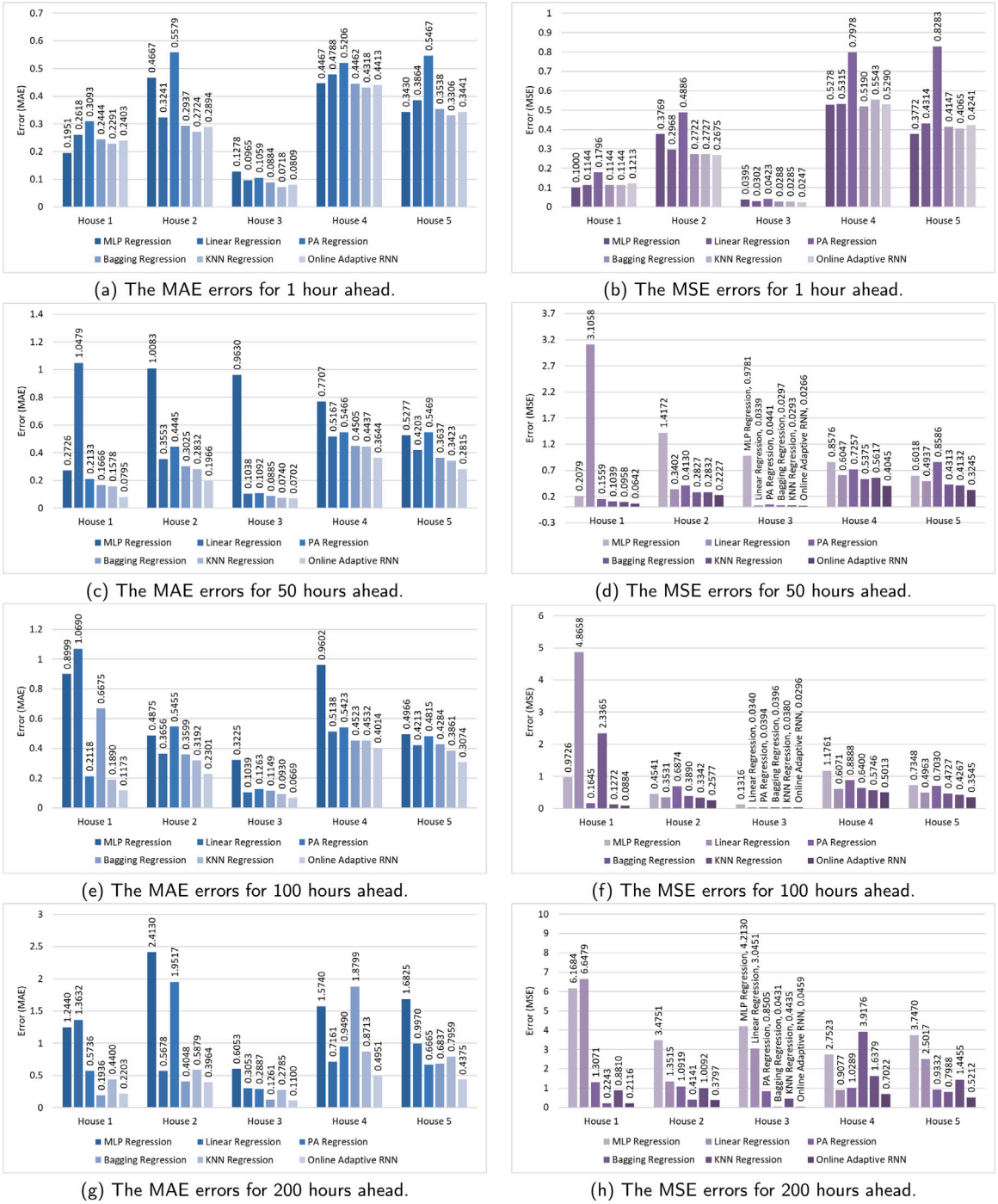
Fig. 8. The MAE and MSE errors for offline LSTM and Online Adaptive RNN.

6.4.2. Results

Fig. 11 shows the impact of the buffering and tuning modules on the accuracy of Online Adaptive RNN. It can be observed that for all the households, the model with the buffering and tuning modules results in the lowest MAE error, and the model without the two modules leads to the highest errors. This figure also shows that the tuning module has a larger impact on the accuracy than the buffering module as error drops more when the tuning module is added than when the buffering module is added. This highlights the need to tune network hyperparameters, specifically the learning rate, as new data arrive.

6.5. Analysis of the buffer size impact

The previous section demonstrated the impact of the buffering and tuning modules on the performance of Online Adaptive RNN. This



(a) The MAE errors for 1 hour ahead.

(b) The MSE errors for 1 hour ahead.

(c) The MAE errors for 50 hours ahead.

(d) The MSE errors for 50 hours ahead.

(e) The MAE errors for 100 hours ahead.

(f) The MSE errors for 100 hours ahead.

(g) The MAE errors for 200 hours ahead.

(h) The MSE errors for 200 hours ahead.

Fig. 9. Comparison of Online Adaptive RNN with five online regression algorithms: MLP, linear, passive-aggressive, bagging, and KNN regression.

subsection further investigates the buffering module and examines the impact of the buffer size on the model accuracy and training time. Since this involves comparing online learning models, Online Adaptive RNNs with different buffer sizes, the prequential evaluation described in Section 5 is used.

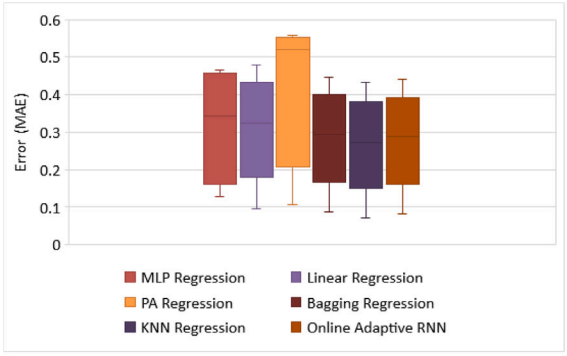
6.5.1. Experiments

For these experiments, the same setup and tuning process have been applied as described in Section 6.2.1: LSTM cells are used and the same ranges of parameters are considered in the online tuning. The difference is that here experiments are conducted without buffer and with buffer sizes of 5, 10, 15, 30, and 50 batches.

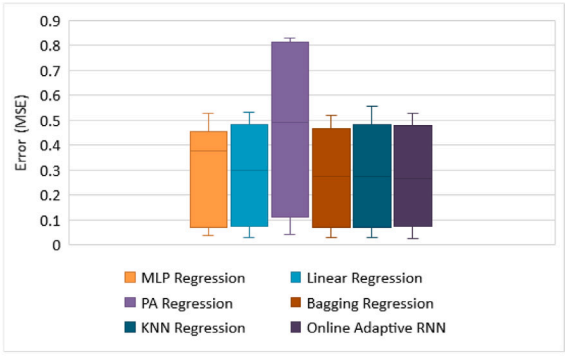
6.5.2. Results

Fig. 12 shows the impact of the buffer size on the accuracy of Online Adaptive RNN for 50 h forecasting horizon. For houses with a high presence of concept drift, such as houses 4 and 5, a smaller buffer size (5 batches) results in a lower error. The reason for this is that a larger buffer retains more batches, some of which may be older, therefore causing the model to pay attention to those older batches which may not reflect the current situation when there is a high presence of concept drift.

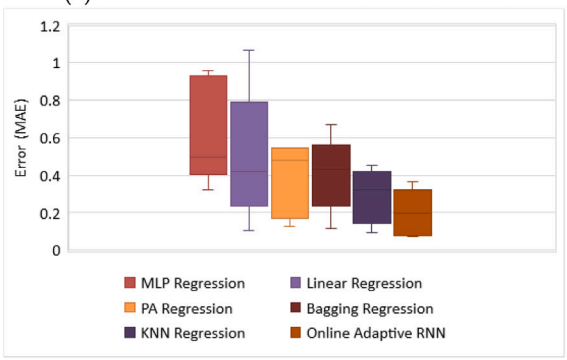
For houses with a lower presence of concept drift, such as houses 1 and 3, the buffer sizes 10 and 15 have the best performance, with size 15 having just slightly better performance than size 10. As the concept drift is not as prominent, having a medium-size buffer improves



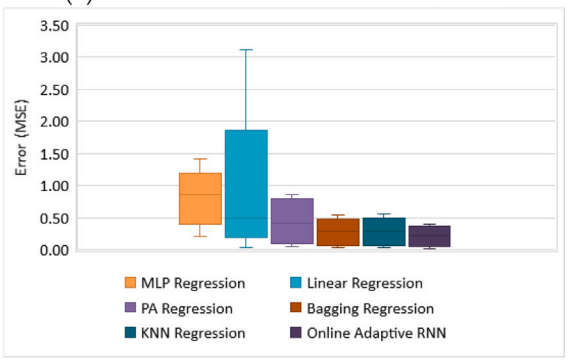
(a) The MAE error intervals for 1 hour ahead.



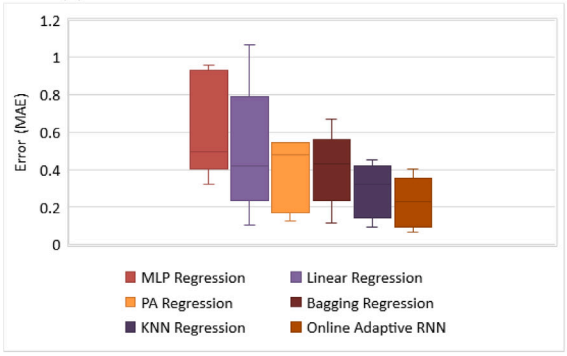
(b) The MSE error intervals for 1 hour ahead.



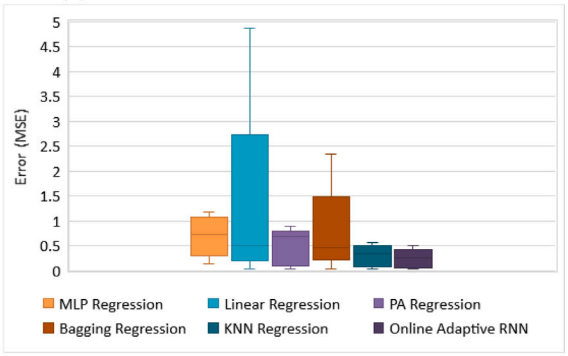
(c) The MAE error intervals for 50 hour ahead.



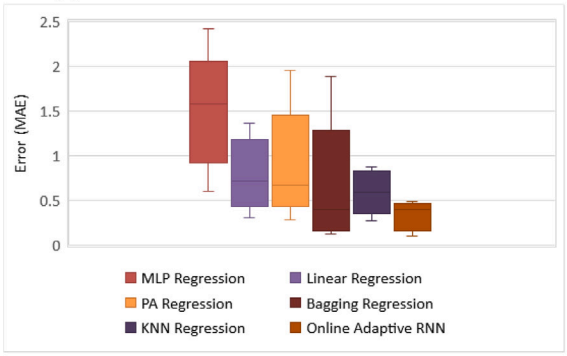
(d) The MSE error intervals for 50 hour ahead.



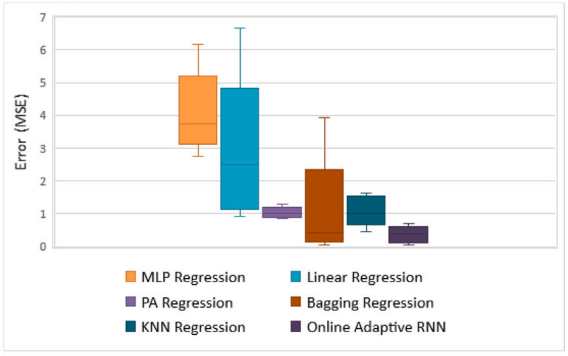
(e) The MAE error intervals for 100 hour ahead.



(f) The MSE error intervals for 100 hour ahead.



(g) The MAE error intervals for 200 hour ahead.

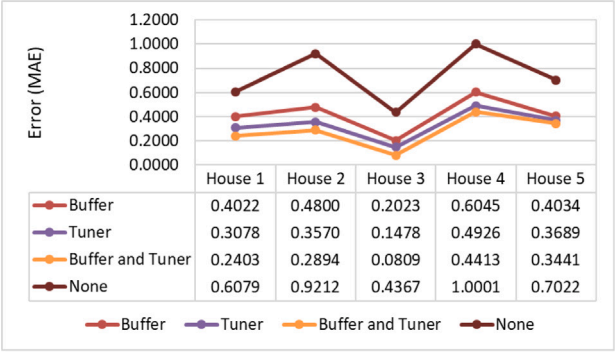


(h) The MSE error intervals for 200 hour ahead.

Fig. 10. Comparison of error intervals for Online Adaptive RNN and the five online regression algorithms: LP, linear, passive-aggressive, bagging, and KNN regression.

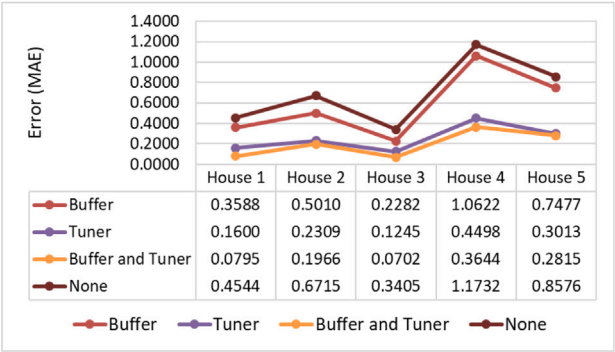
accuracy as the model sees difficult patterns several times. House 2 is somewhat similar to houses 1 and 3; lower errors are achieved for buffer sizes 5 and 10. It can be observed that for all houses, irrelevant to the degree of concept drift, large buffer sizes, such as 30 and 50, lead to performance degradation.

Fig. 13 examines the impact of the buffer size on the computation time: it shows the average training time per day for each of the considered buffer sizes. As expected, increasing the buffer size results in longer training time. Overall, buffer size 10 achieves good results for all houses (although not the best for all houses) irrelevant of concept



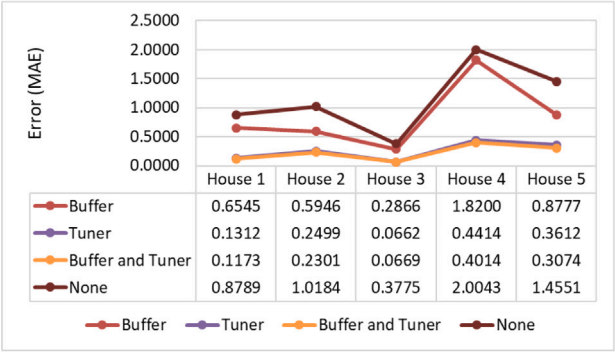
	House 1	House 2	House 3	House 4	House 5
Buffer	0.4022	0.4800	0.2023	0.6045	0.4034
Tuner	0.3078	0.3570	0.1478	0.4926	0.3689
Buffer and Tuner	0.2403	0.2894	0.0809	0.4413	0.3441
None	0.6079	0.9212	0.4367	1.0001	0.7022

(a) The impact of buffering and tuning modules: 1 hour ahead.



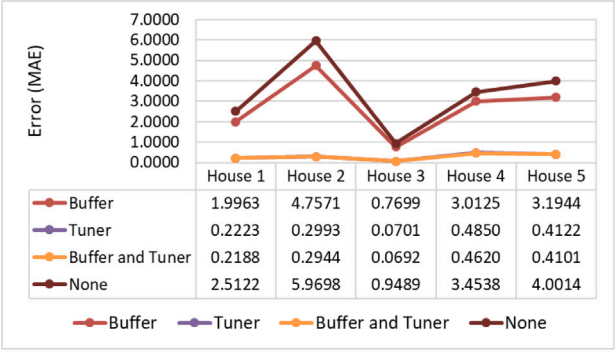
	House 1	House 2	House 3	House 4	House 5
Buffer	0.3588	0.5010	0.2282	1.0622	0.7477
Tuner	0.1600	0.2309	0.1245	0.4498	0.3013
Buffer and Tuner	0.0795	0.1966	0.0702	0.3644	0.2815
None	0.4544	0.6715	0.3405	1.1732	0.8576

(b) The impact of buffering and tuning modules: 50 hours ahead.



	House 1	House 2	House 3	House 4	House 5
Buffer	0.6545	0.5946	0.2866	1.8200	0.8777
Tuner	0.1312	0.2499	0.0662	0.4414	0.3612
Buffer and Tuner	0.1173	0.2301	0.0669	0.4014	0.3074
None	0.8789	1.0184	0.3775	2.0043	1.4551

(c) The impact of buffering and tuning modules: 100 hours ahead.

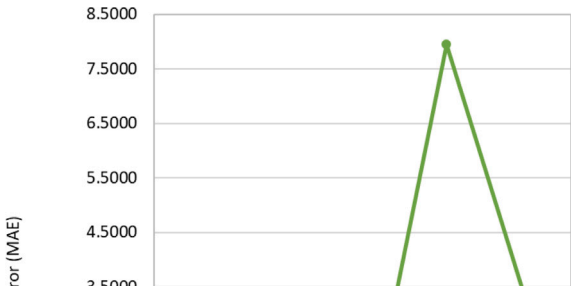


	House 1	House 2	House 3	House 4	House 5
Buffer	1.9963	4.7571	0.7699	3.0125	3.1944
Tuner	0.2223	0.2993	0.0701	0.4850	0.4122
Buffer and Tuner	0.2188	0.2944	0.0692	0.4620	0.4101
None	2.5122	5.9698	0.9489	3.4538	4.0014

(d) The impact of buffering and tuning modules: 200 hours ahead.

Fig. 11. The impact of buffering and tuning modules on the forecasting accuracy.

drift presence while requiring only slightly longer training time than lower batch sizes (see Figs. 14 and 15).



	House 1	House 2	House 3	House 4	House 5
No Buffer	0.1600	0.2308	0.1254	0.4498	0.3013
Buffer size 5	0.1419	0.2113	0.1189	0.2846	0.2477
Buffer size 10	0.0795	0.1966	0.0702	0.3644	0.2815
Buffer size 15	0.0707	0.4312	0.0659	0.7662	0.5813
Buffer size 30	0.1516	1.1200	0.0952	2.1111	1.8800
Buffer size 50	0.6154	1.9855	0.3733	7.9507	3.0491

Fig. 12. The impact of the buffer size on model's accuracy.

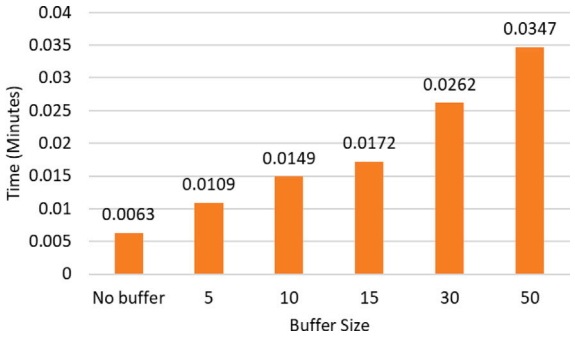


Fig. 13. Average training time for different buffer sizes.

6.6. Training time analysis

One of the common ML approaches for dealing with new data and changing patterns is to re-train the model daily or weekly with all data [32]. Assuming that changes between consecutive days or weeks are small, this approach can lead to good accuracy, but the computation burden is extensive.

This section compares the computation time needed for daily re-training a conventional offline model with the training time of the proposed Online Adaptive RNN. Although experiments were conducted with a relatively small number of readings, they illustrate the advantages of the proposed approach. Nevertheless, with an increased number of readings, the training time for offline LSTM will increase with each day, while the time for Online Adaptive RNN will remain quite consistent.

6.6.1. Experiments

As we are comparing the proposed approach with the LSTM daily re-training, experiments compare the training time per day. Two sets of experiments were conducted:

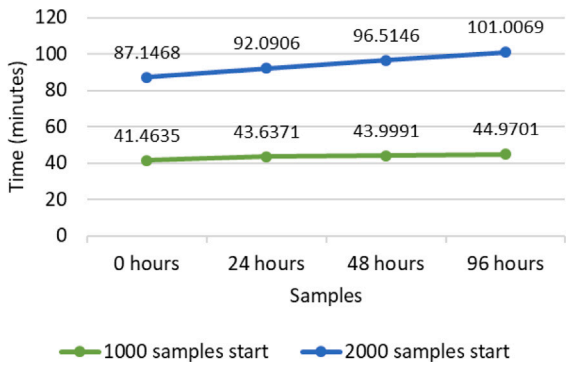


Fig. 14. Average training time for conventional LSTM with hyperparameter tuning.

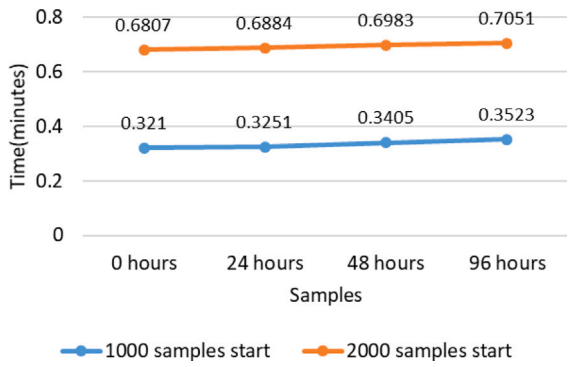


Fig. 15. Average training time for conventional LSTM without hyperparameter tuning.

- Starting with 1000 samples. First, both offline LSTM and Online Adaptive RNN are trained with those 1000 samples. Then, as offline LSTM training is conducted daily will all data, the offline LSTM is trained with 1024, 1048, and 1096 on consecutive days. Online Adaptive RNN just continues training as new data arrive.
- Starting with 2000 samples. This is the same process as the previous set of experiments but it starts with 2000 samples.

As the diversity of data among homes can result in different training times, the experiments were repeated five times with different homes, and the averages are reported. Two variants of the offline LSTM training were considered: with and without tuning. The hyperparameters considered for tuning and the tuning approach are the same as in Section 6.2.1.

The variant with tuning leads to higher accuracy, but re-tuning each day is extremely computationally intensive while it may not always be necessary. Consequently, results without tuning are also reported. As described in Section 4, Online Adaptive RNN is tuned as needed.

The experiments were performed on a computer with Ubuntu OS, AMD Ryzen 4.20 GHz processor, 128 GB DIMM RAM, and four NVIDIA GeForce RTX 2080 Ti 11 GB graphics cards.

6.6.2. Results

Fig. 14 depicts the average training time for conventional LSTM with tuned hyperparameters and Fig. 15 shows the average training time for the same model but without tuning. The horizontal axis represents the increment in the number of samples and each data point represents the training time for a specific day. It can be observed that starting from 2000 samples results in at least double the training time compared to starting from 1000, which is to be expected as the offline LSTM is re-trained each day with complete data set. Examining consecutive days (0, 24, 48, and 96 on the horizontal axis), the slow gradual increase can be observed for each starting point and for training with and without tuning. As expected, training times for tuned models are a magnitude larger than for models without tuning.

For Online Adaptive RNN, the training time for each batch will differ depending if the tuning was triggered for that batch. The average training time per batch is 0.0031 min what with the batch size 5 equates to an average of 0.0149 min per day.

As the variations among days are minimal, there is no need to show this in the plot. Comparing daily training time of Adaptive Online RNN (0.0149 min) with time for the offline LSTM presented in Figs. 14 and 15, it can be observed that Adaptive Online RNN takes only a fraction of time needed by the offline models. This further demonstrates the benefits of the proposed approach.

6.7. Discussion

Overall, Online Adaptive RNN outperformed the traditional offline LSTM as well as five other online algorithms. Although Online Adaptive RNN employs a traditional LSTM as its base learner, the unique architecture makes it capable of learning in an online manner and enables it to achieve higher forecasting accuracy than the LSTM itself. Moreover, online learning makes the proposed approach well suited for practice because there is no need for periodical re-training with new data, which is required for offline models.

For each of the five considered homes, Online Adaptive RNN performed better than LSTM (Fig. 8): the conventional LSTM was not able to learn well in the presence of concept drift. This highlights the need to evaluate the ability of load forecasting algorithms to handle concept drift. Variability of accuracy among homes demonstrates the necessity of considering different data sets when comparing machine learning approaches for energy forecasting.

With respect to other online algorithms, Online Adaptive RNN achieved better forecasting accuracy for the majority of prediction horizons. For one hour ahead, KNN accuracy was slightly better because it was able to take advantage of strong adjacency between the target and training data points. However, when the forecasting horizon increases to 50, 100, and 200 h ahead, this adjacency advantage decreases and Online Adaptive RNN achieves higher accuracy than KNN.

In offline learning, especially with neural networks, hyperparameter tuning is essential for achieving high accuracy. Our approach adds the hyperparameter tuning capability to the online model through the addition of the tuning module. The experiments show that this online tuning significantly improves forecasting accuracy as illustrated in Fig. 11.

While the proposed approach outperformed other online models, there is still space for further improvements. The tuning module could potentially employ a more sophisticated way of tracking accuracy trend, but that would add to computational complexity. There is also a need to explore a relationship between the model behavior and the concept drift occurrence as detected through concept drift detection algorithms.

7. Conclusion

Machine learning approaches, and specifically deep learning architectures, have been extensively used for sensor-based electricity load forecasting. Most techniques involve offline learning, meaning that the model is trained once and then gets used to infer the future load. Hence, these offline techniques do not take advantage of the new data.

Consequently, this paper proposes Online Adaptive Recurrent Neural Network, an online learning approach for load forecasting where the model is continuously updated as new data arrive. The base learner is a recurrent neural network in order to capture temporal dependencies,

while continuous learning is carried out with the addition of preprocessing, buffering, and tuning modules. The preprocessing module prepares data for online learning, the tuning module adapts neural network hyperparameters to newly arriving patterns, and the buffering module facilitates learning from especially difficult patterns and assists in handling concept drift. The evaluation was carried out with five individual households. The proposed approach achieved higher accuracy than the traditional offline long short term memory neural network for all five households for all forecasting lengths. Comparing to five online algorithms, Online Adaptive RNN achieved higher accuracy than the four algorithms for all forecasting horizons. The fifth algorithm, online K Nearest Neighbor, archived slightly better accuracy for one hour ahead forecasting, but Online Adaptive RNN outperformed KNN for 50, 100, and 200 h ahead. Moreover, training time for the proposed approach is an order of magnitude shorter than that of the traditional offline LSTM.

The future work will evaluate the proposed approach with a large number of homes and examine the impact of factors such as the presence of distributed energy resources and the participation in demand response programs. Moreover, employing the concept drift detection algorithms to help the model adapt to changing data will also be examined.

CRediT authorship contribution statement

Mohammad Navid Fekri: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Harsh Patel:** Investigation, Software. **Katarina Grolinger:** Supervision, Funding acquisition, Methodology, Writing - review & editing. **Vinay Sharma:** Funding acquisition, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been supported by London Hydro, Canada and NSERC, Canada under grant CRDPJ, Canada 530743-18. The authors would like to thank London Hydro for supplying industry knowledge and data used in this study.

References

[1] European Environment Agency. Energy and climate change. 2017, https://www.eea.europa.eu/signals/signals-2017/articles/energy-and-climate-change.
[2] US Energy Information Administration. International energy outlook. 2017, https://www.eia.gov/outlooks/ieo/pdf/0484(2017).pdf.
[3] Notton G, Voyant C. Forecasting of intermittent solar energy resource. Adv Renew Energies Power Technol 2018;1:77–114.
[4] Wang Y, Chen Q, Hong T, Kang C. Review of smart meter data analytics: Applications, methodologies, and challenges. IEEE Trans Smart Grid 2018;10(3):3125–48.
[5] Singh PK, Singh N, Negi R. Wind power forecasting using hybrid ARIMA-ANN technique. In: Ambient communications and computer systems. 2019, p. 209–20.
[6] Wang Y, Wang H, Srinivasan D, Hu Q. Robust functional regression for wind speed forecasting based on sparse Bayesian learning. Renew Energy 2019;132:43–60.
[7] Tran D-H, Luong D-L, Chou J-S. Nature-inspired metaheuristic ensemble model for forecasting energy consumption in residential buildings. Energy 2020;191(116552).
[8] Wang H, Lei Z, Zhang X, Zhou B, Peng J. A review of deep learning for renewable energy forecasting. Energy Convers Manage 2019;198(111799).
[9] Fan C, Wang J, Gang W, Li S. Assessment of deep recurrent neural network-based strategies for short-term building energy predictions. Appl Energy 2019;236:700–10.
[10] Sehovac L, Grolinger K. Deep learning for load forecasting: Sequence to sequence recurrent neural networks with attention. IEEE Access 2020;8:36411–26.
[11] Žliobaitė I, Pechenizkiy M, Gama J. An overview of concept drift applications. In: Big data analysis: New algorithms for a new society. 2016, p. 91–114.
[12] Tsymbal A. The problem of concept drift: Definitions and related work, Vol. 106. Computer Science Department, Trinity College Dublin; 2004.
[13] Ahmad T, Zhang H, Yan B. A review on renewable energy and electricity requirement forecasting models for smart grid and buildings. Sustainable Cities Soc 2020;55(102052).
[14] Alobaidi MH, Chebana F, Meguid MA. Robust ensemble learning framework for day-ahead forecasting of household based energy consumption. Appl Energy 2018;212:997–1012.
[15] Grolinger K, Capretz MA, Seewald L. Energy consumption prediction with big data: Balancing prediction accuracy and computational resources. In: IEEE international congress on big data. 2016, p. 157–64.
[16] Fekri MN, Ghosh AM, Grolinger K. Generating energy data for machine learning with recurrent generative adversarial networks. Energies 2020;13(130).
[17] Gao X, Li X, Zhao B, Ji W, Jing X, He Y. Short-term electricity load forecasting model based on EMD-GRU with feature selection. Energies 2019;12(6).
[18] Bouktif S, Fiaz A, Ouni A, Serhani MA. Optimal deep learning lstm model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches. Energies 2018;11(7).
[19] Han S, Qiao Y-h, Yan J, Liu Y-q, Li L, Wang Z. Mid-to-long term wind and photovoltaic power generation prediction based on copula function and long short term memory network. Appl Energy 2019;239:181–91.
[20] Somu N, Gauthama Raman MR, Ramamritham K. A hybrid model for building energy consumption forecasting using long short term memory networks. Appl Energy 2020;261(114131).
[21] Defazio A, Bach F, Lacoste-Julien S. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In: Advances in neural information processing systems. 2014, p. 1646–54.
[22] Johnson R, Zhang T. Accelerating stochastic gradient descent using predictive variance reduction. In: Advances in neural information processing systems. 2013, p. 315–23.
[23] Jothimurugesan E, Tahmasbi A, Gibbons P, Tirthapura S. Variance-reduced stochastic gradient descent on streaming data. In: Advances in neural information processing systems. 2018, p. 9906–15.
[24] Frostig R, Ge R, Kakade SM, Sidford A. Competing with the empirical risk minimizer in a single pass. In: Conference on learning theory. 2015, p. 728–63.
[25] Sánchez-Medina JJ, Guerra-Montenegro JA, Sánchez-Rodríguez D, Alonso-González IG, Navarro-Mesa JL. Data stream mining applied to maximum wind forecasting in the canary islands. Sensors 2019;19(10).
[26] Vexler J, Kramer S. Integrating LSTMs with online density estimation for the probabilistic forecast of energy consumption. In: International conference on discovery science. 2019, p. 533–43.
[27] Liang F, Hatcher WG, Xu G, Nguyen J, Liao W, Yu W. Towards online deep learning-based energy forecasting. In: International conference on computer communication and networks. 2019, p. 1–9.
[28] Gao H, Sollacher R, Kriegel H-P. Spiral recurrent neural network for online learning. In: European symposium on artificial neural networks. 2007, p. 483–8.
[29] Guo T, Xu Z, Yao X, Chen H, Aberer K, Funaya K. Robust online time series prediction with recurrent neural networks. In: IEEE international conference on data science and advanced analytics. 2016, p. 816–25.
[30] Madireddy S, Balaprakash P, Carns P, Latham R, Lockwood GK, Ross R, et al. Adaptive learning for concept drift in application performance modeling. In: International conference on parallel processing. 2019, p. 1–11.
[31] Fields T, Hsieh G, Chenou J. Mitigating drift in time series data with noise augmentation. In: International conference on computational science and computational intelligence. 2019, p. 227–30.
[32] Ceci M, Corizzo R, Malerba D, Rashkovska A. Spatial autocorrelation and entropy for renewable energy forecasting. Data Min Knowl Discov 2019;33:698–729.
[33] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in python. J Mach Learn Res 2011;12:2825–30.
[34] Halford M, Bolmier G, Sourty R, Vaysse R, Zouitine A. Creme, a python library for online machine learning. 2019, URL: https://github.com/creme-ml/creme.
[35] Crammer K, Dekel O, Keshet J, Shalev-Shwartz S, Singer Y. Online passive-aggressive algorithms. J Mach Learn Res 2006;7:551–85.
[36] Oza NC. Online bagging and boosting. In: IEEE international conference on systems, man and cybernetics, Vol. 3. 2005, p. 2340–5.
[37] Janzamin M, Sedghi H, Anandkumar A. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. 2015, arXiv:1506.08473.
[38] Senjyu T, Takara H, Uezato K, Funabashi T. One-hour-ahead load forecasting using neural network. IEEE Trans Power Syst 2002;17(1):113–8.
[39] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015, arXiv:1502.03167.
[40] Cooijmans T, Ballas N, Laurent C, Gülçehre Ç, Courville A. Recurrent batch normalization. In: International conference on learning representations. 2016.
[41] Feurer M, Hutter F. Hyperparameter optimization. In: Automated machine learning: Methods, systems, challenges. 2019, p. 3–33.
[42] Bergstra JS, Bardenet R, Bengio Y, Kégl B. Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems. 2011, p. 2546–54.

[43] Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J. LSTM: A search space odyssey. IEEE Trans Neural Netw Learn Syst 2016;28(10):2222–32.
[44] Gama J, Sebastião R, Rodrigues PP. On evaluating stream learning algorithms. Mach Learn 2013;90:317–46.
[45] Gama J, Sebastião R, Rodrigues PP. Issues in evaluation of stream learning algorithms. In: ACM SIGKDD international conference on knowledge discovery and data mining. 2009, p. 329–38.
[46] Bifet A, Gavalda R. Learning from time-changing data with adaptive windowing. In: SIAM international conference on data mining. 2007, p. 443–8.
[47] Page ES. Continuous inspection schemes. Biometrika 1954;41(1):100–15.
[48] Gama J, Medas P, Castillo G, Rodrigues P. Learning with drift detection. In: Brazilian symposium on artificial intelligence. 2004, p. 286–95.