Asynchronous Adaptive Federated Learning for Distributed Load Forecasting with Smart Meter Data

Mohammad Navid Fekri[a], Katarina Grolinger [a,*] and Syed Mir[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:
Asynchronous Learning
Distributed Learning
Federated Learning
FedNorm
Load Forecasting

ABSTRACT

Load forecasting is essential for the operation and planning of a utility company. Recent large-scale smart meter deployments enabled the collection of fine-grained load data and created opportunities for sensor-based load forecasting. Machine learning has achieved great successes in load forecasting; however, conventional machine learning techniques require data transfer to the cloud or another centralized location for model training. This not only exposes data to privacy and security risks but also, with a large number of smart meters, increases network traffic. Federated Learning (FL) has a potential to alleviate mentioned concerns by training a single ML model in a distributed manner without requiring participants to share their data. Consequently, this paper proposes FedNorm, a novel asynchronous FL approach for load forecasting with smart meter data. While most FL strategies are synchronous and require all clients to complete local training in each round of aggregation, FedNorm is asynchronous and aggregates updates without waiting for lagging clients. To achieve this, FedNorm measures the clients contributions considering similarities of local and global models as well as the loss function magnitudes. The experiments demonstrate that FedNorm achieves higher accuracy than seven state-of-the-art FL techniques. Moreover, experiments show that FedNorm converges in fewer communication rounds compared to other FL models.

1. Introduction

According to the United States Energy Information Administration (EIA), global electricity consumption continues to outpace global population growth [1]. At the same time, countries are setting aggressive greenhouse gas emissions targets; European Union, for example, aims to reduce emissions for 40% and increase energy efficiency for 27% by year 2030 [2]. Renewable energy resources are expanding quickly; however, their intermittent nature and variability pose challenges for balancing supply and demand.

In a such challenging environment, load forecasting has a critical role in balancing supply and demand, providing information for generation scheduling, grid operation, and infrastructure planning, as well as for transitioning towards a smarter grid. Today, electricity consumption data gathered by widely adopted smart meters provide tremendous information about historical consumption patterns on building and even household level, consequently creating new opportunities for fine-grained load forecasting [3].

Machine learning (ML) approaches have been widely used in load forecasting tasks [4] as they discover relations within data and identify patterns. ML-based techniques are typically sensor-based: they use historical load data collected by smart meters, merged with meteorological and other data to train ML models. Then, those trained models are used to infer future energy consumption. Conventionally, smart meter data are transmitted to a data center or other centralized systems for ML model training. In such centralized systems, a single model is trained individually for each smart meter or data from several smart meters are aggregated to train a model [5].

Although these centralized solutions have shown great results in load forecasting, they are faced with numerous challenges. First of all, transferring smart meter data may expose personal information about a consumer's household such as routines, habits, and individual appliance usage [3]. Consumers' concerns about data privacy and security are among the most often mentioned reasons for smart meter installation hesitancy and a major obstacle to smart meter adaptation despite the benefits of improved energy management [6]. Moreover, centralized systems require transmitting all data to a centralized location, which leads to increased network traffic [7]. As the number of smart meters grows, training an individual ML model for meter-level load forecasting becomes computationally expensive and even infeasible [7].

Federated learning (FL) presents a possible solution to these challenges as it is capable of training an ML model across many decentralized edge devices that hold local data without the need to share raw data among devices or even with the centralized system [8]. Fig. 1 depicts an FL scheme: each device receives a copy of the global model and improves it by learning from local data. Then, instead of raw data, the updated parameters of the local models are sent to the server to be aggregated into the global model.

FL represents a shift from centralized to distributed ML, nevertheless, traditional FL employs a synchronous protocol [9]: at each iteration, the server distributes the global model to a subset of clients and then waits for the selected subset of clients to complete their training before collecting all updates and aggregating them. This is a hindrance as device

*Corresponding author
Email address: kgroling@uwo.ca (K. Grolinger)
ORCID(s): 0000-0001-8079-7117 (M.N. Fekri); 0000-0003-0062-8212 (K.G.)

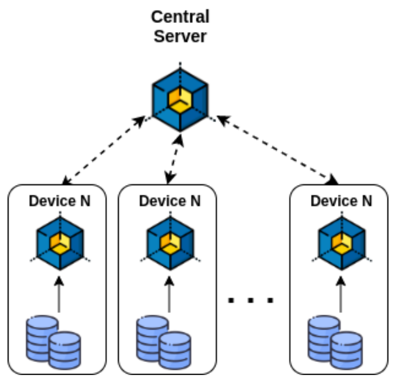

Figure 1: Standard federated learning

heterogeneity can cause some clients to complete training much faster than others while network unreliability can lead to delayed or even dropped transmissions. Asynchronous FL is a newly emerged FL method that allows the server to aggregate parameters without waiting for the lagging devices [10]; however, it is still in its early stages.

Another FL challenge is non-IID (independent and identically distributed) data. Smart meters collect data corresponding to users with different preferences and behaviours leading to significant differences in the data distributions and patterns. The presence of non-IID data degrades the FL performance, brings instability to the training process, and results in a higher number of rounds required for convergence [11].

Consequently, this paper proposes FedNorm, a novel asynchronous FL strategy for load forecasting with smart meter data under a non-IID setting. To deal with staleness and non-IID data, FedNorm measures the contribution of participating nodes taking into consideration the similarity between local and global model weights as well as the magnitude of local objective functions. Then, the global model is updated based on the client's contributions. The proposed approach accelerates the training loss reduction in each communication round and, therefore accelerates convergence. Experiments show that the proposed algorithm converges quickly under asynchronous and non-IID conditions and outperforms most common synchronous and asynchronous FL strategies.

The remainder of the paper is organized as follows: Section 2 presents the related works, Section 3 provides preliminaries, the proposed FedNorm is presented in Section 4, and Section 5 explains the evaluation methodology and 6 discusses results. Finally, Section 7 concludes the paper.

2. Related Work

This section first reviews recent deep learning works for load forecasting including FL methods. Next, techniques for FL with non-IID data and asynchronous FL are discussed.

2.1. Load Forecasting

Load forecasting techniques based on Deep Learning (DL) have gained popularity because of their ability to model complex relationships [5]. Although DL algorithms such as feed-forward neural networks and convolutional neural networks have been used for load forecasting [5], these methods are not designed to capture temporal dependencies present in data. In recent years, Recurrent Neural Networks (RNNs) have improved load forecasting due to their ability to capture time dependencies [12].

Sehovac *et al.* [13] proposed Sequence to Sequence Recurrent Neural Network (S2S RNN) with attention for load forecasting. S2S model improves time modeling by employing two RNNs, encoder and decoder RNN, and authors add the attention to strengthen the link between the encoder and decoder. Tian *et al.* [14] also employed S2S model, but their work focused on scaling load forecasting for a large number of smart meters.

Jagait *et al.* [15] introduced Online ARIMA-RNN ensemble, a load forecasting approach capable of learning from new data as they arrive. The approach combines Online Adaptive Recurrent Neural Network [12] and Rolling ARIMA using the adaptive weighted aggregation technique. Long Short-Term Memory (LSTM), a type of RNN, was combined with Support Vector Machine, Random Forest, and a data preprocessing technique to create a multi-step forecasting strategy capable of improving forecasting accuracy [16].

For load forecasting tasks, researchers combined LSTM with other techniques to improve the accuracy of predictions. Li *et al.* used empirical mode decomposition and sample entropy with LSTM for ultra-short-term load forecasting [17]. LSTM was merged with Convolutional Neural Network (CNN) to forecast the monthly peak load for a time horizon of three years [18]. Sekhar and Dahiya [19] also combine LSTM with CNN, but they added Grey Wolf Optimization (GWO) to obtain the optimal set of parameters for CNN and LSTM. Moreover, the transformer architecture from natural language processing has recently been adapted for load forecasting by modifying the transformer workflow, incorporating N-space transformation, and adding a technique for handling contextual features [20]. The transformer-based architecture achieved a slightly better forecast than Seq2Seq neural networks.

Besides deep learning approaches, techniques based on traditional ML and statistical models have also been explored. Chodakowska *et al.* [21] used Auto Regressive Integrated Moving Average (ARIMA) for regional load forecasting and examined ARIMA's robustness to noise. Tan *et al.* [22] proposed an approach based on multi-task learning and least square Support Vector Machine (SVM). Similarly, Wang *et al.* [23] also used SVM, but they combined SVM with LSTM for multi-energy load forecasting. Tree-based models have also been used, although they are commonly combined with another technique: for example, Gradient Boosting Decision Trees (GBDT) were combined with LSTM for multi-energy load forecasting [24]. Nevertheless, in recent years, deep learning models have been dominant in load forecasting [25].

The reviewed works showed an increased load forecasting performance; however, they are all centralized techniques burdened with the aforementioned challenges of large network traffic, and privacy concerns. To address these issues, Taïk *et al.* [26] investigated edge computing and FL for household load forecasting with smart meters. They employed the FL paradigm to learn from a group of houses with similar profiles; however, the performance of the suggested FL model is degraded by non-IID data.

Briggs *et al.* [27] explored the use of FL and FL with hierarchical clustering (FL+HC) for energy load forecasting on the household level. Their results showed that the proposed approaches outperform centralized learning but not local learning. Zhang *et al.* [28] proposed a probabilistic model for solar irradiation forecasting based on deep learning, variational Bayesian inference, and FL. Although the results showed competitive performance, the main drawback is that the model does not support non-IID data.

All reviewed FL-based load forecasting studies [26, 27, 28] only consider a synchronous scenario, while we deal with asynchronous systems. This means that in our system, clients may send updates to the server at different times due to differences in the computation abilities of each node or due to communication delays or interruptions.

2.2. FL with non-IID data and Asynchronous FL

The FL performance degrades with non-IID data [29], but we cannot expect an IID distribution in many real-world settings. To deal with the non-IID problem, Smith *et al.* [30] introduced MOCHA, a paradigm that combines multi-task learning with FL. MOCHA fits a separate model for each local node and uses multi-task learning in the aggregation process. Mohri *et al.* [31] introduced agnostic FL, which uses a weighted average of the clients' gradients and a novel optimization approach to update the shared model. Their research demonstrated that the proposed method promotes fairness and reduces bias.

The MOCHA [30] and Agnostic FL [31] consider non-IID challenges in diverse domains; however, they have not been applied to load forecasting and their capabilities for load forecasting need to be investigated. Fekri *et al.* [32] proposed an FL model with a dynamic learning rate for load forecasting with non-IID smart meter data. Their FL model outperformed both centralized and local models for each smart meter. To overcome the non-IID data challenges in time-series data including load forecasting, Yeongwoo *et al.* [33] introduced dynamic clustering into the FL framework. On the server side, the framework employs ClusterGAN and HypCluster to dynamically group clients into the clusters according to their loss. The aggregation is carried out individually for each cluster, and the updated parameters are sent to clients that are part of the same cluster.

Discussed FL approaches [30, 31, 32, 33] are well suited for non-IID data; however, they are synchronously trained and, thus, unfit for heterogeneous devices and unreliable networks. Chen *et al.* [10] proposed an asynchronous online FL framework (ASO-Fed), in which clients perform online learning from local streaming data. ASO-Fed uses a feature Representation learning method on the server to extract cross-device attributes from the clients' updates. Fleet [34] is another online FL framework: it introduced AdaSGD, an asynchronous learning algorithm robust to stale updates (updates that are arriving with delay). For calculating similarities among clients, some labeled data must be shared with the server what creates potentials for privacy leakage. Additionally, AdaSGD has a limited capacity for dealing with non-IID data. Fleet keeps all the gradients including stale ones assuming that they may contain valuable information. In the presence of clients with large delays and non-IID data, gradients may be more dissimilar what can cause challenges in AdaSGD convergence.

FedAsync [35], an asynchronous FL algorithm, applies regularization on the local clients and uses a weighted average to update the global model. While FedAsync demonstrated some staleness tolerance, for large staleness, FedAsync performs similarly to synchronous FL. Similarly, federated adaptive weighting [11] aims to improve the FL performance in presence of non-IID data through assigning different weights for the participating nodes when updating the global model.

Asynchronous FL works [10, 34, 35, 11] introduced different techniques; however, asynchronous FL is an emerging topic and needs to be further investigated in diverse settings. To our knowledge, our proposed method is the first asynchronous FL algorithm for load forecasting.

3. Preliminaries

Assume that we have K distributed devices. Let P_k denote a partition of data points stored on the device k, with n_k indicating the size of P_k, and $N = \sum_{k=1}^{K} n_k$ representing the total number of data points on all devices. For any $k \neq k'$ and $P_k \bigcap P'_k = \emptyset$, the FL objective can be described in a form of the optimization problem:

$$\min_{w} L(w) = \sum_{k=1}^{K} \frac{n_k}{N} \psi_k(w) \tag{1a}$$

$$\text{where } \psi_k(w) = \frac{1}{n_k} \sum_{i \in P_k} \ell_i(x_i, y_i; w_k) \tag{1b}$$

where $L(w)$ is the global model loss function, $\psi_k(w)$ is the loss of the k^{th} device, $\ell_i(x_i, y_i; w_k)$ is the loss function for data point $\{x_i, y_i\}$, and w_k represents the local model parameters.

FedAVG is the widely used synchronous FL approach [8]. Each round of FedAVG starts by randomly selecting a subset of devices and broadcasting the model to those devices. The selected devices train in parallel their local models with the local data for multiple epochs. Then, these local devices send model updates to the server, and the server aggregates the changes when it receives updates from all selected clients. The updating mechanism of most synchronized FL algorithms is similar to FedAvg [36]. One drawback of the synchronized FL is that during each global iteration, if one or more clients experience significant network latency or have more data and require longer training time, complete FL system must wait. This results in idling and prolonged training time [10].

The IID condition in stochastic gradient-based learning is crucial as it allows models to have an unbiased estimate of the full gradient [29]. FedAvg has proved to be successful when data distribution for local nodes is similar to that of local nodes' data gathered centrally [11]. However, in load forecasting, the local data distribution is generally non-IID since local nodes contain different usage patterns. In FL, the local objective $\psi_k(w)$ is closely related to data distribution P_k. A large number of local updates on non-IID data lead the local model $\psi_k(w)$ towards optima of its local objective as opposed to the global objective $L(w)$. This discrepancy between local models w_k and the global model w accumulates along training, resulting in increased communication rounds before training converges [11].

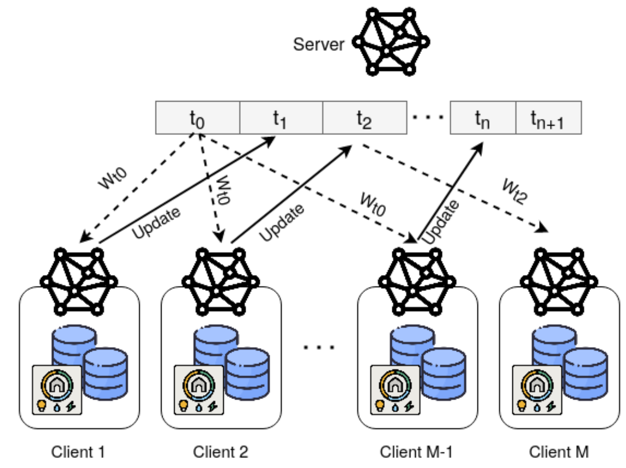

Figure 2: Asynchronous federated learning in the proposed FedNorm

4. FedNorm

This section describes the proposed FedNorm, an asynchronous approach for learning from distributed smart meter data when some of the clients are unable to participate in the training process due to network instability or when the lagging clients do not complete their training timely. Each client carries out its own data preprocessing using its own local data. First, Min-Max normalization is performed individually for each smart meter to scale data to [0:1] range. This enables the system to handle diverse magnitudes of smart meter readings among participants, reduces the dominance of the large features, and consequently results in improved convergence. Next, the sliding window technique [12] is applied locally, for each smart meter independently, to prepare data for the ML model. The first window contains the first w readings and makes the first sample. Next, the window slides for s steps to make the second sample containing readings from step s to $s+w$, and so on. As in standard FL, each local model trains with its local data and sends parameters to the server.

As seen in Fig. 2, clients' updates are merged into the global model at different time steps; for example, Client 1 changes are merged at t_1 while Client 2 changes are added at t_2. The theoretical analysis shows that the diversity of the node contributions in FL convergence can be measured by the local gradient of each node and the global gradient [11]. This motivates us to measure the contribution of participating nodes by assigning the weights for participating nodes based on the similarity between local weights w_i and global weight w together with the magnitude of local objective functions $\psi_k(w)$. An intuitive weighting design should follow the notion that nodes with larger contributions in reducing the global objective function $L(w)$ deserve higher impact in each global round. Therefore, our process for assigning adaptive weights includes three steps: determining node contribution, scaling node contributions, and calculating client contribution ratios.

Determining Node Contributions: We specifically measure each node's contribution at each global round using contribution λ, which is defined as:

$$\lambda_k = \overbrace{\|w - w_k\|}^{\alpha_k} * \overbrace{(\psi_k(w) - \sum_{k=1}^{K} \frac{\psi_k(w)}{K})}^{\beta_k} \tag{2}$$

Here, α_k represents the degree of similarity between the global and local models, while β_k defines the direction of contribution. The α_k is calculated as the sum of the absolute vector values, where the vectors are weight differences between the global model and the local models. In FL, the direction of minimizing local objective $\psi_k(w)$ might not align with the direction of minimizing global objective $L(w)$ even though they have similar weights [11]. Therefore, we need β_k as a difference between local objectives and the global objective to define the alignment direction. If β_k is positive, it has an opposite direction to the global aggregation. In contrast, when β_k is negative, the local node positively contributes to the global aggregation.

Scaling Node Contributions: Since α_k has no upper bound, the $\lambda_k = \alpha_k * \beta_k$ swings over a broad range of negative and positive values amplifying the difference between large positive and negative values. This wide difference between the large positive and negative values causes the weighting function to be skewed toward large numbers. To solve this issue and reduce the impact of large λ_k, we scale node contributions using a non-linear mapping based on the Gaussian function:

$$f(\lambda_k) = ae^{-\frac{(\lambda_k-\mu)^2}{2\sigma^2}} \tag{3}$$

where a is the maximum, μ is the mean, and σ is the standard deviation of λ_k. The designed mapping function gives local nodes with positive impact in the aggregation more weights and reduce the effect of less effective participant while controlling the impact of large λ_k values.

Calculating Client Contribution Ratios: After determining node contributions and scaling them to handle the participants with large values, we use Softmax (Equation 4) function to calculate the ratio of each participating node in the global model aggregation as follows:

$$\xi_k(w_k) = \frac{e^{f(\lambda_k)}}{\sum_{j=1}^{K} e^{f(\lambda_j)}} \tag{4}$$

Softmax is a mathematical function that transforms a vector of numbers into a vector of probabilities, with the probability of each value proportional to the vector's relative scale.

Algorithm 1 presents the FedNorm process. FedNorm employs LSTM, a variant of RNN, as the base learner because LSTM can capture temporal dependencies and has been very successful in load forecasting [12]. First, the global LSTM model is initialized with random weights, Line 2. Lines 3 to 12 describe the process that happens in each global training round. A subset of devices S_t is randomly selected, Line 4, and the global model is broadcast to those devices, Line 5. The selected devices then train their local models in parallel with their local data for multiple epochs, Line 7. Procedure $AsyncClientUpdate$ in Line 6 represents the parallel execution for the selected clients while Line 7 indicates initiation of the procedure $AsyncClientUpdate$ executed on each client individually.

Procedure $AsyncClientUpdate$, starting at Line 15, shows the local model training process. First, the local data P_k are divided into the batches B of size b_s (Line 16). Next, for E epochs (Line 17) and B batches (Line 18), the local weights are updated (Line 19).

The clients from S_t that have completed their training send their new local weights back to the server for aggregation, Line 20, and the server updates the global model by calculating the weighted average of the received local weights as shown in lines 9-11. These three lines 9-11 are the core of FedNorm and correspond to the three steps described above and represented with equations 2-4. Note that the server does not wait to receive updates from all clients, but aggregates the global model as client's updates arrive. Next, the updated global model is sent back to all non-running clients, Line 12. The process is repeated from Line 3 until convergence and, finally, the trained global model is broadcasted to all participants, Line 13.

Algorithm 1 FedNorm

```
1:  Server Execution:
2:  Initialize global model weights $w_0$
3:  for global iterations t=1,2,..., T do
4:      $S_t \leftarrow$ random set of $m$ clients
5:      Send global model to $S_t$ clients
6:      for each client k $\in S_t$ in parallel do
7:          $w^k_{t+1} \leftarrow$ AsyncClientUpdate($k$,$w_t$)
8:      Receiving parameters from clients
9:      $\lambda_k = \|w - w_k\| * (\psi_k(w) - \sum_{k=1}^{K} \frac{\psi_k(w)}{K})$
10:     $\xi_k(w_k) = \frac{e^{f(\lambda_k)}}{\sum_{j=1}^{K} \frac{n_j}{N} e^{f(\lambda_j)}}$
11:     $w_{t+1} \leftarrow \sum_{k \in K} \xi_k(w_k) w^k_t$
12:     Send the model to all non-running participants
13: Send the model to all participants

14: Client Execution:
15: procedure AsyncClientUpdate($k$,$w$)
16:     $B \leftarrow$ split $P_k$ into batches of size $b_s$
17:     for each local epoch $e < E$ do
18:         for batch $b \in B$ do
19:             $w \leftarrow w - \eta \nabla \ell(w)$
20:     return $w$ to server
```

5. Evaluation Methodology

This section presents the dataset and evaluation metrics, followed by Non-IID analysis, experimental setup, and benchmark algorithms.

5.1. Dataset and evaluation Metrics

The FedNorm evaluation is conducted with the real-world residential consumers dataset provided by London Hydro, a local electrical distribution utility. Each consumers' dataset contains hourly energy consumption for three years, resulting in 25,560 readings per household. Additional features, including the day of the year, the day of the month, the week of the year, the day of the week, and the hour of the day, were devised from the load reading date/time to assist with modeling daily, monthly, and weekly patterns. The study included 19 consumers. Fig. 3 depicts a fragment of the load data for each of the households. It can be observed that load patterns, as well as load magnitudes, differ greatly among households.

Each household with its own data is treated as a local node in FL scenarios. For the purpose of experimental evaluation, each household dataset is divided into 70% training and 30% testing set. This divide remains the same for all FL and non-FL experiments. Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) are common error metrics considered for load forecasting [32]; therefore, they are used here as well. It is important to note that RMSE is a scale-dependent error metric, which means that the same RMSE values have different meanings depending on the magnitude of data. MAPE, on the other hand, expresses errors as percentages; thus, it is better suited for comparison among datasets.

The algorithms were implemented in PyTorch and all experiments were conducted with AMD Ryzen 4.20 GHz processor and NVIDIA GeForce RTX 2080 Ti graphics card.

5.2. Non-IID Analysis

Non-IID data in FL typically means the differences between distributions P_i and P_j for different clients i and j. This subsection examines data distributions for the considered houses to investigate how similar or different is energy consumption among those houses.

Fig. 4 visualizes data distributions for all houses using a Ridgeline plot; specifically, this figure shows a kernel den-

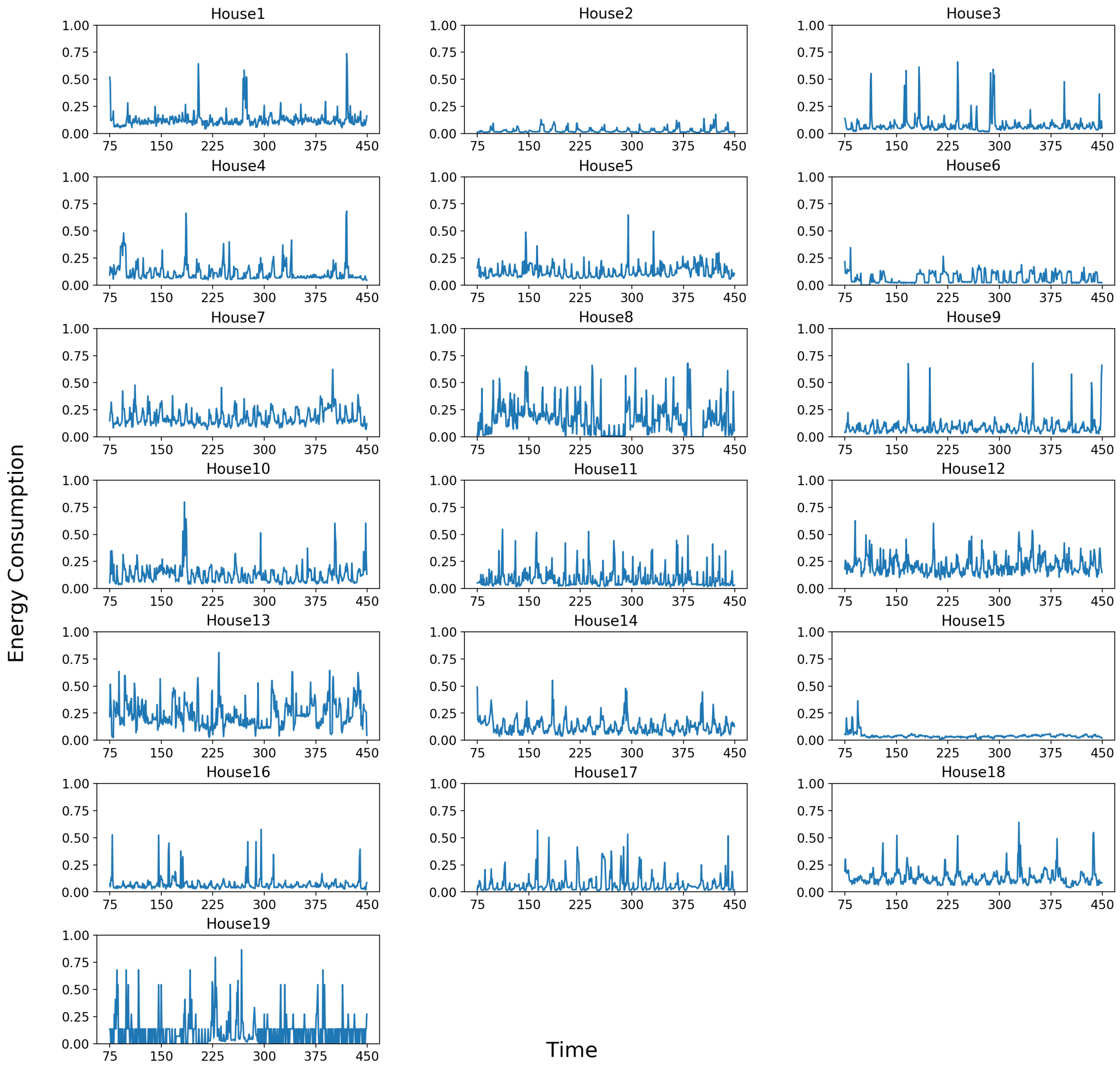

Figure 3: Sample energy consumption data from each of the houses

sity distribution for each house aligned to the same horizontal scale and slightly overlapped. It can be observed that houses have different shapes of the distributions which suggests that the consumption data distributions among houses are different.

To further examine the differences among houses, Fig. 5 shows the box plot of energy consumption for each of the houses. It can be observed that the houses greatly differ in terms of the mean energy consumption as well as in terms of ranges.

Another way of comparing data distributions among the houses is with the Q-Q (quantile-quantile) plot which plots the quantiles of the two distributions against each other. If the two distributions are the same, the points in the Q–Q plot will approximately lie on the 45° line. Fig. 6 shows the Q-Q plot comparing House 2 with randomly selected houses 6, 13, 14, and 19, while Fig. 7 does the same for House 11. In these plots, there is a significant distance between data points and the 45° line, which indicates that the distributions in the pairs are different.

Consequently, comparisons presented in figures 4, 5, 6, and 7 show that energy consumption data distributions among houses are different and that we are dealing with non-IID data.

5.3. Experimental Setup

po simulate the asynchronous setting, in each round of FL, a fraction of clients is randomly selected, and, then, the selected clients are again randomly divided into two groups, (1) those that complete training in the current round (without delay) and (2) those that need additional time to complete training (with delay). The first group is aggregated with the global model immediately after finishing their training, and the second group waits for the next rounds to be aggregated. This separation into without and with delay simulates delayed training. As in FL, the process depends on the round

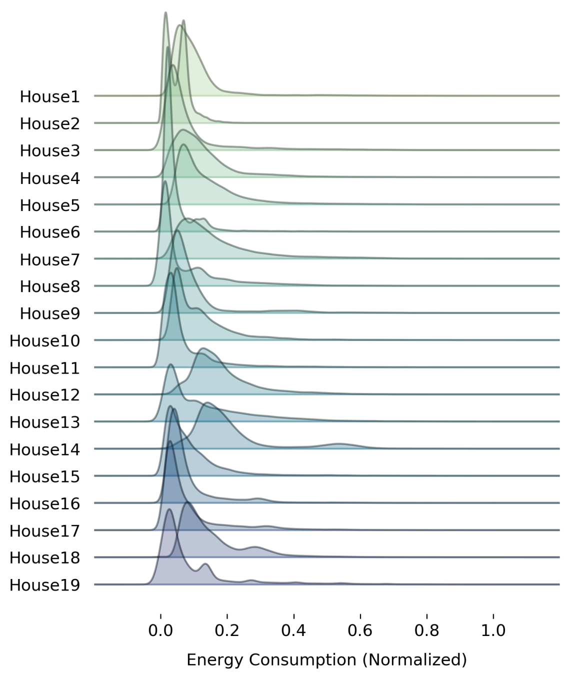

Figure 4: Ridgeline plot showing a density distribution of energy consumption for each house aligned to the same horizontal scale

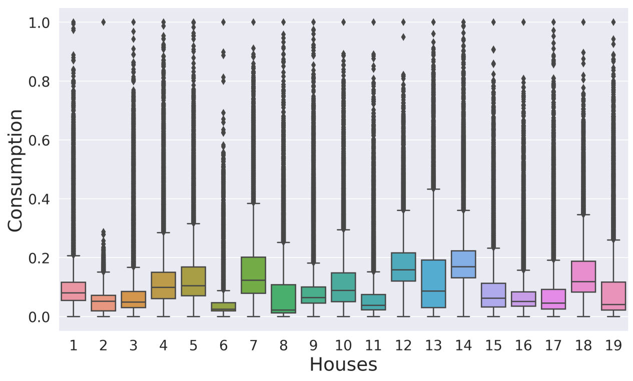

Figure 5: Energy consumption box plot for the houses

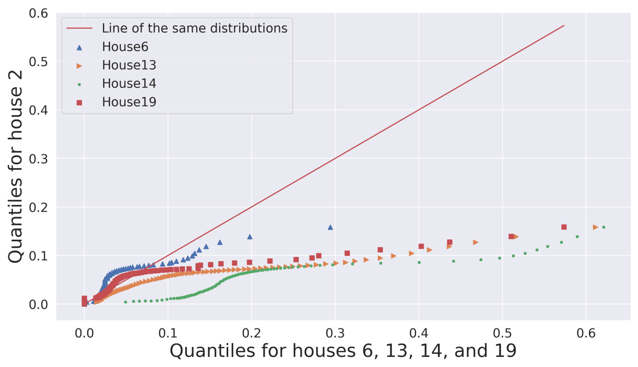

Figure 6: Q-Q plot comparing energy consumption distributions between House 2 and Houses 6, 13, 14, and 19

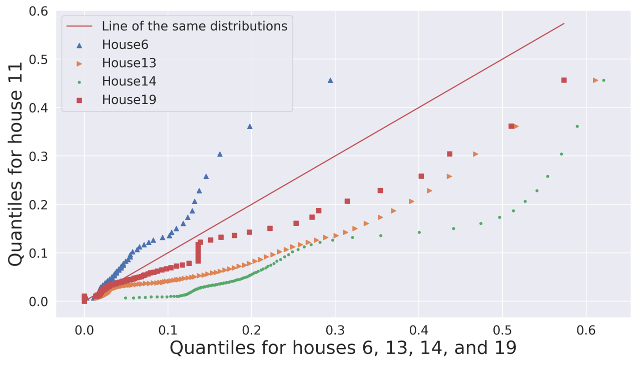

Figure 7: Q-Q plot comparing energy consumption distributions between House 11 and Houses 6, 13, 14, and 19

in which the gradients are aggregated, for the group (2) 'with delay' we push the aggregation to the next round. Note that the client that was already delayed in the previous round can again be delayed as the delayed clients are selected randomly.

Fig. 8 illustrates the simulation process. In Step $S1$, a fraction F (in figure five clients) of the clients is selected to train their local models. In step $S2$, the selected clients are randomly divided into two groups C without delay and $R1$ with delay. The clients in C finish their training in the current round and are aggregated into the global model in Step $S3$ but $R1$ clients are still training their local models. In Step $S4$, all clients except $R1$s receive a copy of the updated global model. In the next round of training $S5$, again a fraction of the clients is selected; however, the size of the fraction is equal to $F' = F - count(R1)$. In $S6$, similar to $S2$, the F' is divided into two groups: without and with delays, C and $R2$ respectively. Thus, in Step 6, the delayed clients $R1$ from the previous round and some clients from round two complete the training, as indicated with C in the figure, while $R2$ clients are still running and will complete training in the next round. Clients C are aggregated into the global model in Step $S7$. This process continues until the model converges.

5.4. FL Techniques Included in Comparison

The proposed FedNorm is compared to the following synchronous and asynchronous FL techniques:

- FedSGD: Federated stochastic gradient descent is the first proposed FL method. In this approach, the server averages the local clients' gradients to make a gradient descent step on the global model [37].

- FedAvg: This is a widely used generalization of FedSGD [8], in which the local nodes train for several epochs and then exchange the weights rather than the gradients.

- FedAdam, FedYogi, FedAdaGrad [38]: These approaches are an adaptive variants of the stochastic gradient (FedSGD) method. The adaptive optimizers ADAM, YOGI, and AdaGrad are applied on the server to address the issues of client drift.

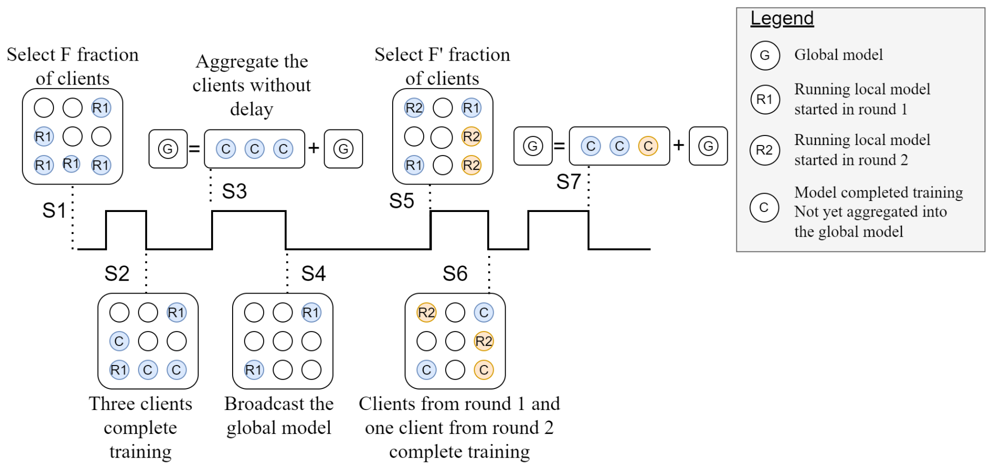
Select F fraction of clients
Aggregate the clients without delay
Select F' fraction of clients
Legend
Global model
Running local model started in round 1
Running local model started in round 2
Model completed training Not yet aggregated into the global model
S1
S2
S3
S4
S5
S6
S7
Three clients complete training
Broadcast the global model
Clients from round 1 and one client from round 2 complete training

Figure 8: Asynchronous setting simulation.

- FedProx: This FL framework adds a proximal term on the local objective function to mitigate the data heterogeneity problem and to improve the model stability [39].
- FedAsync: This approach uses a weighted average to update the server model. [35].

All listed algorithms are synchronous, except for FedAsync which is asynchronous. Comparing FedNorm against these algorithms allows us to examine the performance from several perspectives. The comparison with the commonly used FedSGD and FedAVG algorithms shows the difficulties of load forecasting in FL settings and in the presence of heterogeneous data. AdaGrad, ADAM, and YOGI show the efficiency of various adaptive federated optimizers in the face of heterogeneous data while FedProx improves handling heterogeneous data by generalization and re-parametrization of FedAvg. Thus, comparing FedNorm to those algorithms examines behaviour in presence of heterogeneities. Finally, FedAsync, asynchronous FL, is needed to show tolerance to staleness.

6. Results and Analysis

This section presents comparison results, investigates the statistical significance, and examines convergence.

6.1. Predictive Performance Comparison

The proposed FedNorm is compared to five other FL methods on the task of predicting load demand eight hours and sixteen hours ahead. Table 1 shows the average predictive accuracy of each algorithm for both forecasting horizons, calculated across all homes in the dataset. In terms of both evaluation metrics, MAPE and RMSE, FedNorm achieves the lowest error for both forecasting horizons. In terms of MAPE error, FedAvg, FedProx, and FedAsync show a similar performance while FedSGD and FedAdagrad exhibit large errors. In terms of RMSE, FedNorm performs the best, while there is little difference among other models.

Table 1
Average MAPE and RMSE for 8 and 16 hours ahead forecast

Methods	MAPE		RMSE	
	8 Hours ahead	16 Hours ahead	8 Hours ahead	16 Hours ahead
FedNorm	4.14	3.11	0.0773	0.0754
FedAvg	5.67	5.85	0.1115	0.1124
FedProx	5.86	5.92	0.1132	0.1151
FedSGD	8.30	8.71	0.1167	0.1427
FedAdagrad	8.81	12.60	0.1183	0.1556
FedAdam	6.68	7.51	0.1076	0.1153
FedYogi	6.38	8.11	0.1188	0.1139
FedAsync	5.94	5.78	0.1132	0.0971

As households differ greatly in their consumption patterns, it is important to examine the performance for individual houses. Thus, figures 9 and 10 depict MAPE and RMSE obtained by each algorithm for eight hours ahead prediction, for each house individually. Overall, FedNorm performs better than other approaches. While for a few houses, such as House 6, other approaches achieve slightly lower error than FedNorm, for most houses FedNorm performs much better.

Figures 11 and 12 show the same metrics, MAPE and RMSE, for sixteen hours ahead prediction. Again, FedNorm performs the best for most houses, and for those houses for which it is not the best, it achieves the error very close to the lowest one.

Fig. 13 depicts an example of actual and predicted loads for House 3 and eights hours forecasting horizon. Specifically, forecasts obtained by the top three algorithms (FedNorm, FedAvg, FedProx) according to MAPE are shown. It can be observed that the predicted values better match the actual values for FedNorm than for the other approaches, which supports the results observed in Table 1.

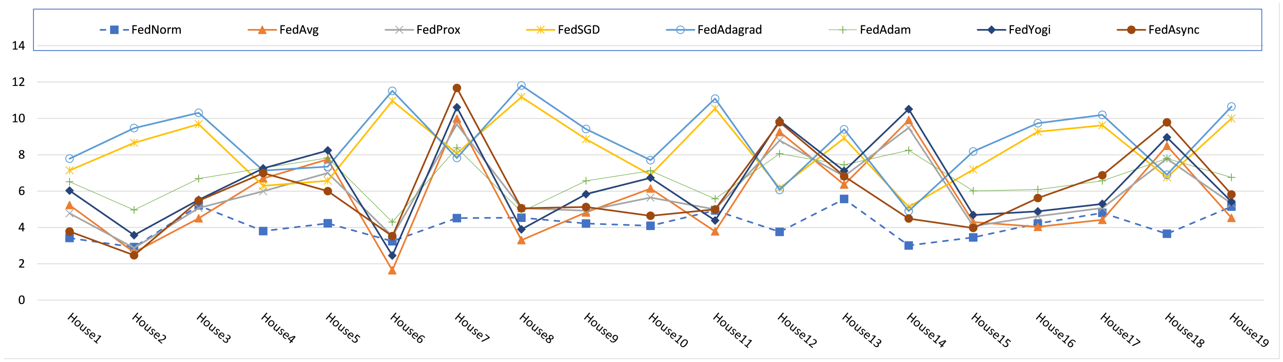

Figure 9: MAPE errors for eight hours ahead prediction

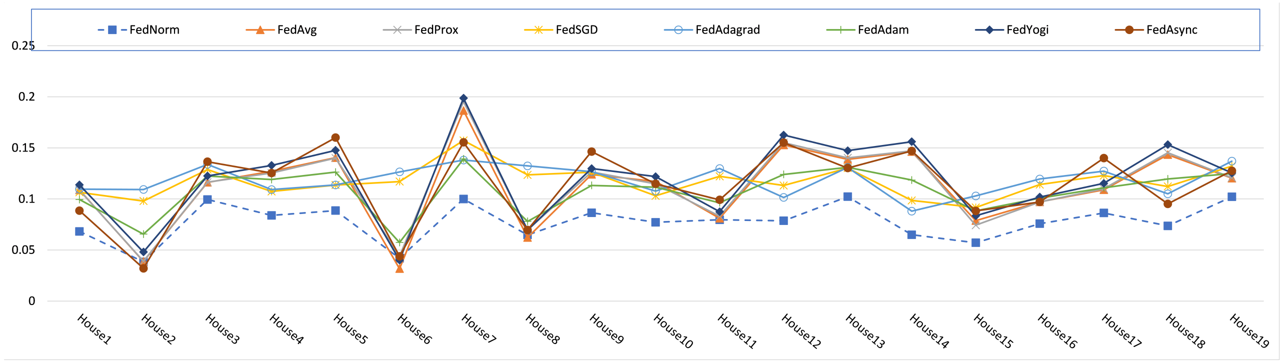

Figure 10: RMSE errors for eight hours ahead prediction

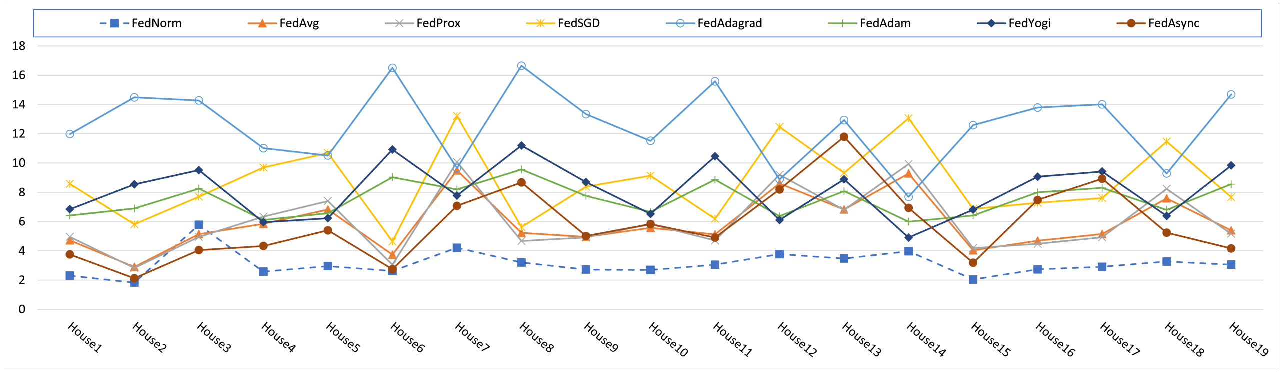

Figure 11: MAPE errors for 16 hours ahead prediction.

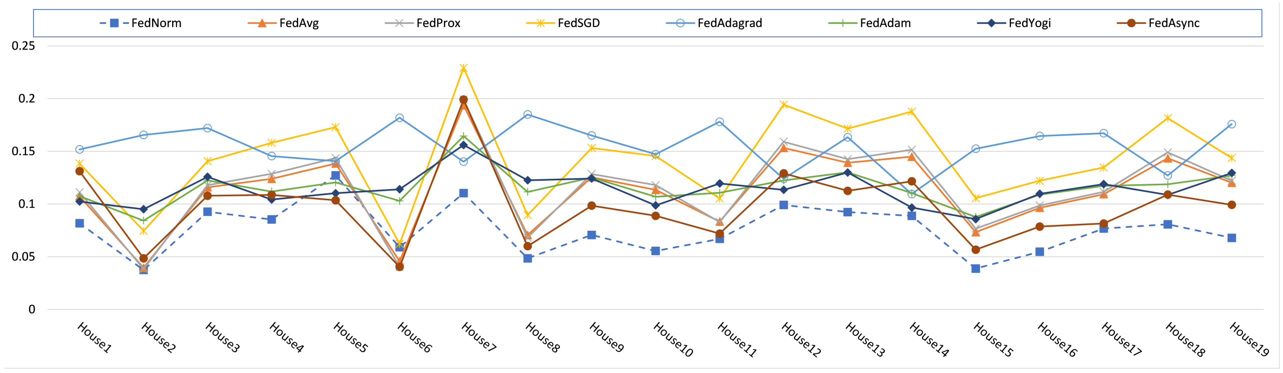

Figure 12: RMSE errors for 16 hours ahead prediction.

6.2. Statistical Significance

Standard metrics such as MAPE and RMSE compare models, but they do not consider whether the differences between the models are significant. Diebold-Mariano test [15] can be used to determine if forecasts are significantly different. For two forecasts $f_1, \dots, f_n$ and $g_1, \dots, g_n$, for a time series $y_1, \dots, y_n$, the Diebold-Mariano test is defined as fol-

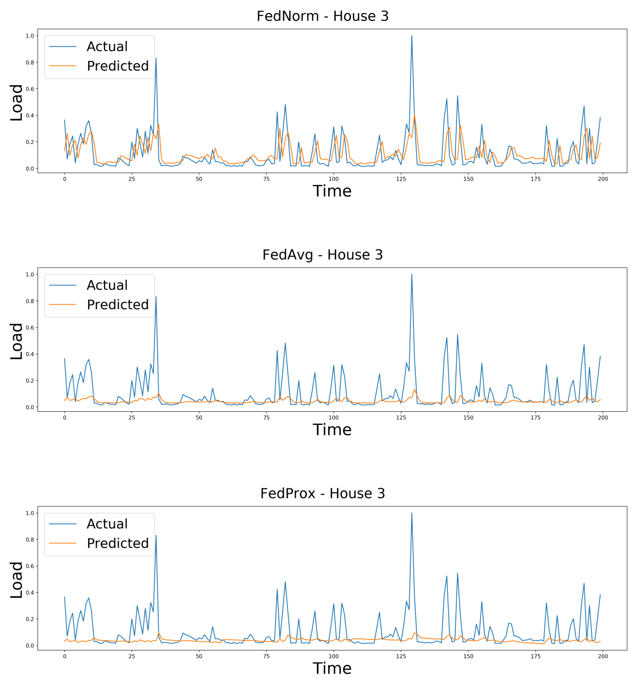

Figure 13: Actual and predicted load: House 3, eight hours ahead forecast

lows:

$$DM = \frac{\bar{d}}{\sqrt{[\gamma_0 + 2\sum_{k=1}^{n^{\frac{1}{3}}+1} \gamma_k] n^{-1}}} \tag{5}$$

where $\gamma_k = \frac{1}{n}\sum_{i=1}^{n}(d_i - \bar{d})(d_{i-k} - \bar{d})$ is autocovariance at lag k, $d_i = e_i^2 - r_i^2$ is the loss differentials where $e_i = y_i - f_i$ and $r_i = y_i - g_i$ are the residuals (errors) for the two forecasts, and $\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_t$ is the sample mean loss differential.

Because DM is a statistical test, when the DM value falls outside the range [-1.96 1.96], the null hypothesis (the difference between two model's performance is not significant) is rejected at a 5% confidence level. Fig. 14 shows DM values for FedSGD, FedAvg, FedAdam, FedAdagrad, FedYogi, FedProx, FedAsync contrasted to the proposed FedNorm for each house. As the goal is to evaluate the proposed FedNorm against other algorithms, rows in the figure compare FedNorm to each of the other algorithms. It can be observed that the outcome of the Diebold-Mariano test indicates that the forecasts obtained by the proposed FedNorm are significantly different from all other considered approaches.

6.3. Analysis of Convergence

The performance of FL depends on its convergence characteristics. For the convergence analysis, in each round of training, the clients are first trained and then tested, recording the test error for each individual house. To analyze overall model convergence across all houses, we report the median, 25th and 75th percentile, maximum, and minimum of errors.

Fig. 15 shows the error distributions for all FL methods and the first 20 training rounds. It can be observed that FedNorm converges very fast, and after the third round, it converges. Additionally, the error ranges are small, which means that the FedNorm performance for different houses is very similar and with a low error. This can be explained by FedNorm giving the nodes with larger contribution to the reduction of global objective function higher impact in each round. The contribution of each node is based on the similarity of local and global weights together with the magnitude of local objective functions.

FedProx takes longer to converge, with large error ranges in early rounds and smaller in later rounds. Although FedAvg converges after the third round as our approach, error ranges for FedAvg are large, which indicates inconsistencies among houses. FedSGD ta-kes longer to converge; at first, the error range is large, but after 12 rounds, the error range is reduced. Errors for FedYogi, FedAdam, FedAsync, and FedAdagrad vary over training rounds, and there is no clear indication of convergence. Overall, FedNorm converges faster than other approaches, and its performance among homes is more consistent.

6.4. Traffic Analysis

This subsection examines the quantity of data transferred between the clients and the server in traditional ML learning and in the proposed FedNorm. As described in Subsection 5.3, to simulate the asynchronous setting, in each round of FL, a fraction of clients is randomly selected to represent training with delay. In all experiments, the maximum number of clients selected for training in that round has been set to nine; therefore, in the worst-case scenario for network traffic, all nine complete training.

From Fig. 15, it can be seen that FedNorm converges after the third round. In each round, the server sends the global model to clients selected for training, and the clients send the updated model back to the server. With the model size of 0.021MB and for the worst case scenario involving nine clients in both transfer directions, this results in 9 clients * 3 rounds *2 directions * 0.021MB model size = 1.135MB traffic. At the end of the training process, the trained model is sent to all clients resulting in 19 * 0.021MB = 0.399MB traffic. The total traffic for FedNorm is 1.135MB + 0.399MB = 1.533MB.

On the other hand, each household dataset size is 0.636MB. The traditional centralized training requires transferring the complete dataset to the server resulting in network traffic of 19 * 0.636MB = 12.084MB. Compared to FedNorm, this centralized training results in more than seven times larger transfer. It is worth bearing in mind that as the size of datasets grows, data traffic increases for the centralized training while it remains the same for FedNorm as only parameters are transferred.

7. Conclusion

Conventional machine learning-based load forecasting transfers smart meter data to a centralized location where the ML model is trained. The drawback of this approach is that all data must be transferred to a remote location, which

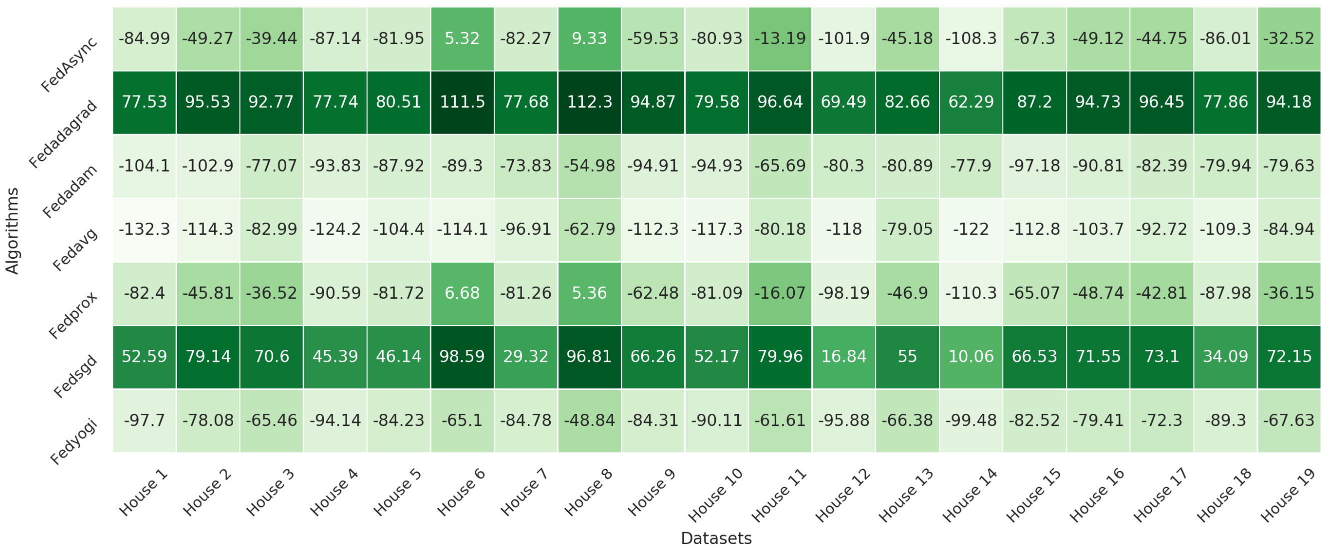

Figure 14: Diebold-Mariano test: DM values for 19 houses.

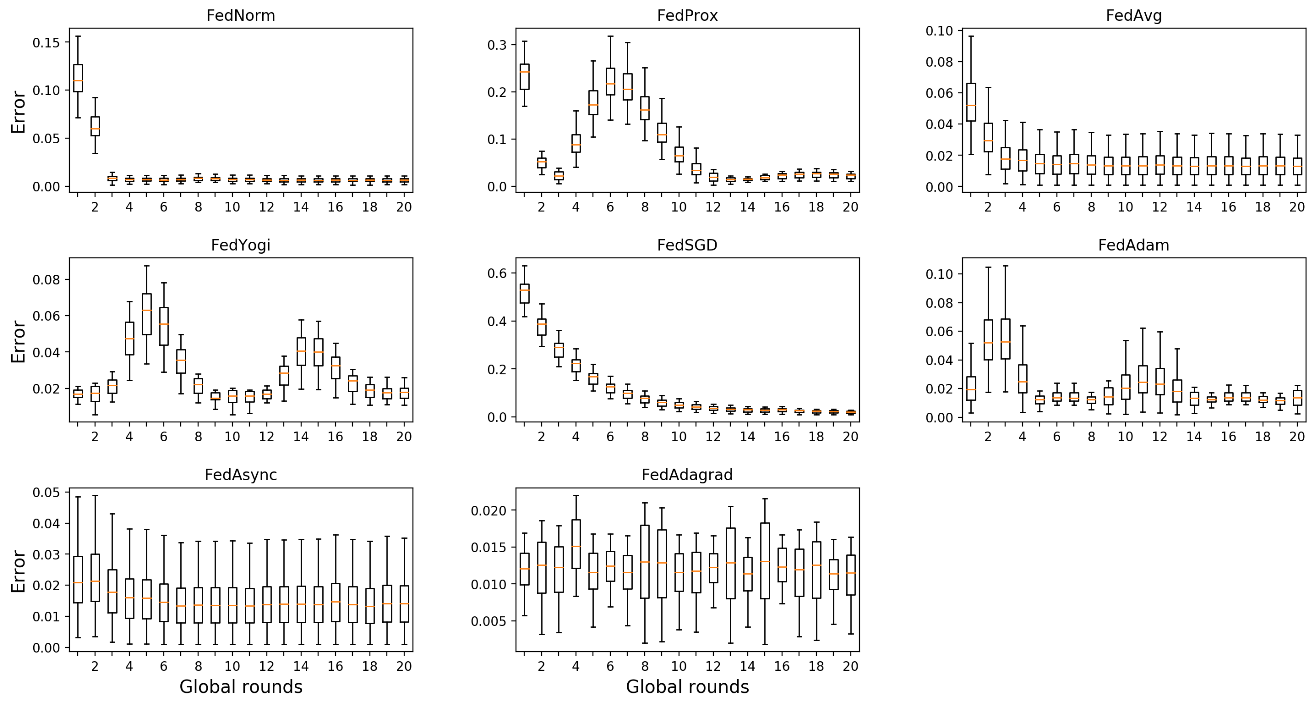

Figure 15: Errors (median, 25th percentile, 75th percentile, maximum, and minimum) over iterations for each of the seven algorithms

increases network traffic, and exposes data to security and privacy risks.

Therefore, this study proposes FedNorm, an asynchronous FL approach for load forecasting with smart meter data, which does not require participants to share their local data. FedNorm updates the global model asynchronously without waiting for all client devices to complete their training. The contributions of each client node are determined based on the similarity of local model weights with the global model weights while taking into account the magnitude of the local objective function. The experiments show that FedNorm achieves higher accuracy in terms of MAPE and RMSE than seven other FL approaches FedSGD, FedAvg, FedAdam, FedYogi, FedAdagrad, FedProx, and FedAsync for eight and sixteen hours forecasting horizons. While forecasting accuracy varies among houses, FedNorm consistently achieves lower or similar error rates compared to the other algorithms. Moreover, FedNorm converges faster then the mentioned algorithms. The main reason for this is that FedNorm pays more attention to the local models which are more similar to the global model. Statistical tests showed that the difference between FedNorm and other approaches is significant while examination of performance on individual houses demonstrated high consistency and low error rates. This study considered two forecasting horizons, 8 and 12 hours ahead, and in both scenarios FedNorm outperformed the other seven techniques. While FedNorm achieved comparable accuracy for the two horizons, further investigation is needed to examine FedNorm behavior with longer forecasting horizons.

Future work will examine FedNorm with a larger dataset that contains dissimilar as well as similar clients to gain a deeper understanding of FedNorm behavior in the presence of occasional similar clients. Moreover, future work will examine the behavior of FedNorm in the presence of concept drift, when load patterns change due to building or behavioral factors.

Acknowledgements

This research has been supported under grant ALLRP 571890 - 21 by London Hydro and NSERC. Computation was enabled in part by the Digital Research Alliance of Canada. The authors would like to thank London Hydro for supplying industry knowledge and data used in this study.

References

[1] U.S. Energy Information Administration, International energy outlook, https://www.eia.gov/todayinenergy/detail.php?id=44095, 2017.

[2] European Environment Agency, Energy and climate change, https://www.eea.europa.eu/signals/signals-2017/articles/energy-and-climate-change, 2017.

[3] Y. Wang, Q. Chen, T. Hong, C. Kang, Review of smart meter data analytics: Applications, methodologies, and challenges, IEEE Transactions on Smart Grid 10 (2018) 3125–3148.

[4] J.-P. Lai, Y.-M. Chang, C.-H. Chen, P.-F. Pai, A survey of machine learning models in renewable energy predictions, Applied Sciences 10 (2020).

[5] A. Arif, N. Javaid, M. Anwar, A. Naeem, H. Gul, S. Fareed, Electricity load and price forecasting using machine learning algorithms in smart grid: A survey, in: International Conference on Advanced Information Networking and Applications, 2020, pp. 471–483.

[6] N. Balta-Ozkan, O. Amerighi, B. Boteler, A comparison of consumer perceptions towards smart homes in the UK, Germany and Italy: reflections for policy and future research, Technology Analysis & Strategic Management 26 (2014) 1176–1195.

[7] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J. S. Rellermeyer, A survey on distributed machine learning, ACM Computing Surveys 53 (2020) 1–33.

[8] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, 2017, pp. 1273–1282.

[9] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, et al., Applied federated learning: Improving Google keyboard query suggestions, Google Research (2018).

[10] Y. Chen, Y. Ning, M. Slawski, H. Rangwala, Asynchronous online federated learning for edge devices with non-iid data, in: IEEE International Conference on Big Data, 2020, pp. 15–24.

[11] H. Wu, P. Wang, Fast-convergent federated learning with adaptive weighting, IEEE Transactions on Cognitive Communications and Networking (2021).

[12] M. N. Fekri, H. Patel, K. Grolinger, V. Sharma, Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network, Applied Energy 282 (2021).

[13] L. Sehovac, K. Grolinger, Deep learning for load forecasting: Sequence to sequence recurrent neural networks with attention, IEEE Access 8 (2020) 36411–36426.

[14] Y. Tian, L. Sehovac, K. Grolinger, Similarity-based chained transfer learning for energy forecasting with big data, IEEE Access 7 (2019) 139895–139908.

[15] R. K. Jagait, M. N. Fekri, K. Grolinger, S. Mir, Load forecasting under concept drift: Online ensemble learning with recurrent neural network and arima, IEEE Access 9 (2021) 98992–99008.

[16] W. Guo, L. Che, M. Shahidehpour, X. Wan, Machine-learning based methods in short-term load forecasting, The Electricity Journal 34 (2021).

[17] K. Li, W. Huang, G. Hu, J. Li, Ultra-short term power load forecasting based on CEEMDAN-SE and LSTM neural network, Energy and Buildings 279 (2023).

[18] X. Zhang, T. K. Chau, Y. Chow, T. Fernando, H. H.-C. Iu, A novel sequence to sequence data modelling based CNN-LSTM algorithm for three years ahead monthly peak load forecasting, IEEE Transactions on Power Systems (2023).

[19] C. Sekhar, R. Dahiya, Robust framework based on hybrid deep learning approach for short term load forecasting of building electricity demand, Energy 268 (2023).

[20] A. L'Heureux, K. Grolinger, M. A. Capretz, Transformer-based model for electrical load forecasting, Energies 15 (2022).

[21] E. Chodakowska, J. Nazarko, Ł. Nazarko, Arima models in electrical load forecasting and their robustness to noise, Energies 14 (2021).

[22] Z. Tan, G. De, M. Li, H. Lin, S. Yang, L. Huang, Q. Tan, Combined electricity-heat-cooling-gas load forecasting model for integrated energy system based on multi-task learning and least square support vector machine, Journal of Cleaner Production 248 (2020).

[23] S. Wang, K. Wu, Q. Zhao, S. Wang, L. Feng, Z. Zheng, G. Wang, Multienergy load forecasting for regional integrated energy systems considering multienergy coupling of variation characteristic curves, Frontiers in Energy Research 9 (2021).

[24] S. Wang, S. Wang, H. Chen, Q. Gu, Multi-energy load forecasting for regional integrated energy systems considering temporal dynamic and coupling characteristics, Energy 195 (2020).

[25] J. Zhu, H. Dong, W. Zheng, S. Li, Y. Huang, L. Xi, Review and prospect of data-driven techniques for load forecasting in integrated energy systems, Applied Energy 321 (2022).

[26] A. Taïk, S. Cherkaoui, Electrical load forecasting using edge computing and federated learning, in: EEE International Conference on Communications, 2020, pp. 1–6.

[27] C. Briggs, Z. Fan, P. Andras, Federated learning for short-term residential energy demand forecasting, arXiv:2105.13325 (2021).

[28] X. Zhang, F. Fang, J. Wang, Probabilistic solar irradiation forecasting based on variational bayesian inference with secure federated learning, IEEE Transactions on Industrial Informatics (2020).

[29] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, et al., Adaptive federated learning in resource constrained edge computing systems, IEEE Journal on Selected Areas in Communications 37 (2019) 1205–1221.

[30] V. Smith, C.-K. Chiang, M. Sanjabi, A. S. Talwalkar, Federated multi-task learning, in: Advances in Neural Information Processing Systems, 2017, pp. 4424–4434.

[31] M. Mohri, G. Sivek, A. T. Suresh, Agnostic federated learning, in: International Conference on Machine Learning, 2019, pp. 8114–8124.

[32] M. N. Fekri, K. Grolinger, S. Mir, Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks, International Journal of Electrical Power & Energy Systems (2021).

[33] Y. Kim, E. Al Hakim, J. Haraldson, H. Eriksson, J. M. B. da Silva, C. Fischione, Dynamic clustering in federated learning, in: ICC 2021-IEEE International Conference on Communications, 2021, pp. 1–6.

[34] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, F. Taiani, Fleet: Online federated learning via staleness awareness and performance prediction, in: International Middleware Conference, 2020.

[35] C. Xie, S. Koyejo, I. Gupta, Asynchronous federated optimization,

arXiv:1903.03934 (2019).

[36] L. Li, Y. Fan, M. Tse, K.-Y. Lin, A review of applications in federated learning, Computers & Industrial Engineering (2020).

[37] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: ACM SIGSAC conference on computer and communications security, 2015.

[38] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, H. B. McMahan, Adaptive federated optimization, arXiv:2003.00295 (2020).

[39] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, arXiv:1812.06127 (2018).