UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Canberk Özen

A Collaborative Approach for Large-scale Electricity Consumption Using Federated Learning

Master's Thesis (30 ECTS)

Supervisor(s): Assistant Professor Feras Awaysheh
Sadi Alawadi, PhD

Tartu 2022

A Collaborative Approach for Large-scale Electricity Consumption Using Federated Learning

Abstract:
Forecasting energy demand is a crucial topic in the energy industry to keep the balance between supply and demand, hence keeping the grid in effective operation. The adoption of renewable energy sources for the supply makes the forecasting problem ever the more prominent because of the additional uncertainty they bring to the grid, besides the consumers' energy usage patterns. The uncertainty on the demand side forecasting can be theoretically overcome via a centralized predictive model that takes note of the consumers' past electricity usage. However, in practice, forecasting energy demand is challenged by users' concerns for the privacy of their energy data and the scalability of storing it, in addition to completing the model updates in time. Both problems can be solved if the centralized training paradigm is replaced with federated training, where each household trains its model locally, and the centralized server only acts as a coordinator by aggregating the weights of the individual models' and sending the updates back to them, all without seeing the consumers' data. Because of the diversity in energy usage, the convergence of local models may require too much time. This study will investigate federated learning to develop a clustering algorithm that groups similar residences as one node to fasten the model convergence without reducing its accuracy.

Keywords: Federated Learning, Clustered Federated Learning, Federated Energy Forecasting, Deep Learning, Energy Demand Forecasting, Statistical Heterogeneity, Smart Grids

CERCS: T120 Systems engineering, computer technology

Koostööl põhinev lähenemisviis suuremahuliseks elektritarbimiseks liitõppe abil

Lühikokkuvõte:
Energianõudluse prognoosimine on energiatööstuses tasakaalu hoidmiseks ülioluline teema pakkumise ja nõudluse vahel, hoides seega võrgu tõhusas töös. Taastuvate energiaallikate kasutuselevõtt tarnimisel muudab prognoosimisprobleemi üha olulisemaks, kuna need toovad võrku lisaks tarbijate energiakasutusharjumustele täiendavat ebakindlust. Ebakindlus nõudluse prognoosimisel saab teoreetiliselt ületada tsentraliseeritud ennustusmudeli abil, mis võtab arvesse tarbijate varasema elektrikasutuse. Praktikas on aga energiavajaduse prognoosimine kasutajate mure oma energiaandmete privaatsuse ja mastaapsuse pärast selle salvestamist, lisaks mudeli värskenduste õigeaegsele lõpuleviimisele. Mõlemad probleemid võivad olla lahendatud, kui tsentraliseeritud koolituse paradigma asendatakse liitkoolitusega, kus iga leibkond koolitab oma mudelit kohapeal ja tsentraliseeritud server toimib ainult koordinaatorina koondades üksikute

mudelite kaalud ja saates värskendused tagasi aadressile ilma tarbijate andmeid nägemata. Energiakasutuse mitmekesisuse tõttu kohalike mudelite lähenemine võib nõuda liiga palju aega. See uuring uurib liitõpet, et töötada välja rühmitusalgoritm, mis rühmitab sarnased elukohad üheks sõlmeks, et kinnitada mudeli konvergentsi ilma selle täpsust vähendamata.

Võtmesõnad: Liitõpe, rühmitatud liitõpe, ühendatud energiaprognoos, sügav õpe, energiavajaduse prognoosimine, statistiline heterogeensus, nutikas Võred

CERCS: T120 Süsteemitehnika, arvutitehnoloogia

Contents

1 Introduction

Digitalization of everyday life started to unlock high amounts of data available for us, and the energy industry is not exempt from this trend [1], due to the deployment of smart meters which can measure the energy usage of a building. If correctly analyzed, the energy usage data collected by smart meters have the potential to make the energy management processes less error-prone and more specialized with respect to individual households. In addition to operational benefits, correct utilization of smart meter data also has the potential for mitigating climate change due to more efficient use of energy [2].

In simple terms, energy management balances the energy supply with volatile energy demand. With the advance of renewable energy sources, energy generation also started to become more uncertain, mimicking the behavior of demand. Consequently, operating the energy grid gets more difficult due to the difficulties in accurately forecasting supply and demand.

The use of deep learning models [3] has shown significant promise for capturing the sharp fluctuations that commonly occur in energy demand data. Several empirical studies have proved the superiority of deep learning models, particularly that of LSTMs [4], in comparison with the simple machine learning models and traditional time series models [5–7]. However, high performance achieved with these models comes at a cost. In order to achieve accurate forecasts, deep learning models require sizable amounts of building energy consumption data to be stored in a central server; which in turn increases data storage and computation costs significantly [8]. In addition to the scalability challenge, centralized deep learning is also vulnerable in terms of preserving the privacy of households' energy data. For example, a privacy breach at the server can expose the energy data of all the buildings connected to the server; which can be used to deduce the household occupancies [9].

Challanges regarding the scalability and security of centralized deep learning can be overcome by changing the learning paradigm itself. A new paradigm, called Federated Learning [10], offers a solution to both of the aforementioned issues. In FL, all the households are still connected to a server; yet data sharing is not allowed from the households(clients) to the server with most of the computations taking place in the clients. At every FL round, the server sends a global model to the clients and the clients update the global model by training it with their local data. The server then receives the local model parameters and aggregates them to produce the new global model. These FL rounds continue until a time limit is reached or expected performance is achieved. In short, by keeping the data inside the clients and sharing the computational burden among them; the server-client network becomes much more robust to security threats and easier to scale to high number of clients.

So far we have briefly explained that training deep learning models under a federated learning setting has the potential to solve the security and scalability issues commonly encountered under centralized learning. While this is true, federated deep learning has not been tested thoroughly due to it being a new phenomenon. Such an experimental gap constituted the motivation behind this thesis, which will be explained further in the next section.

1.1 Motivation

We first feel the need to explain the security benefits that FL brings. As we noted previously, FL prevents the occurrence of data exchange between the server and clients, thus it is *secure by design*. Security by design does not make an FL system as absolutely secure, yet it makes it more robust against external threats compared to server-client networks operating under centralized learning. Even if we won't be measuring the extent of security that FL provides in this thesis; we thought of exploring demand forecasting under an FL setting would be more suitable than doing it under a centralized setting, due to FL's significant security advantage.

On the other hand, we should also note the existence of several federated energy demand forecasting studies focusing on forecasting accuracy and scalability. For example; [11], [12] and [13] modified the simple aggregation method from [10] by adding a clustering step to increase the forecasting accuracy, with [13] also conducting a large-scale demand forecasting experiment. All of the forecasting studies which used the clustering step noted a significant increase in the forecasting accuracy, regardless of the dataset they used.

Despite these findings, no extensive experiments have been done to compare the different clustering methods that were used so far. Instead, nearly all of the federated energy forecasting research aimed at proposing a new algorithm to achieve better forecasts. In addition, some popular federated clustering algorithms [14] have never been used with energy demand data. The importance of such theoretical advances cannot be denied; yet the proposed algorithms must be benchmarked against each other to really understand their effectiveness. The main focus of this thesis would be to conduct such a benchmark by comparing the performances of several federated learning algorithms that can be used for energy demand forecasting. The selected algorithms will be compared against a centralized learning baseline and the first algorithm proposed in FL research, FedAVG [10].

This being said, this thesis is by no means an exhaustive study due to the number of countless methods in existence. We will be focusing on only a subset of FL algorithms that have a clustering step, since such algorithms proved to be experimentally effective in terms of forecasting accuracy. What we aim is to explore some of these algorithms' relative performances, with the hope of triggering new benchmarking experiments for federated demand forecasting in the future.

Several energy demand forecasting studies have already been done under an FL setting. [15] aimed at measuring the robustness of server-client network with regards to security threats. [11], [12] and [13] modified the simple aggregation method from [10] by adding a clustering step to increase the forecasting accuracy, with [13] also conducting a large-scale demand forecasting experiment. All of the forecasting studies which used the clustering step noted a significant increase in the forecasting accuracy, regardless of the dataset they used. Despite these findings, no extensive experiments have been done to compare the different clustering methods that were used so far. In addition, some popular clustering algorithms [14] have never been used with energy demand data.

1.2 Research Problem

In most cases, the centralized learning process uses independent and identically distributed(iid) data samples. These datasets allow the model to learn better since it only specializes in one task: learning the patterns inside the dataset stored in the central server. In FL, the data sources are most likely non-iid, since storing the client datasets in one location is not allowed. This method creates a conundrum in which one global model tries to learn several different tasks simultaneously since the data distributions in clients are different from each other. Consequently, a significant decrease in model performance compared to the centralized learning setting is possible. In technical terms, this FL drawback is called the *statistical heterogeneity problem.*

Consider $\boldsymbol{C_i}$ as the household i participating in a federated learning setting, which includes one coordinating server and n households in total. Since human behavior can vary greatly, all of these households may have energy usage patterns that may be significantly different from each other, including the households coming from the same or similar socioeconomic groupings. As these energy usage data diverge among the households, it becomes more and more difficult for a global model to capture the patterns in every client since optimizing for one of them may result in insufficient forecasting accuracy for the other. Not to mention the inherent hardship of this task; the first aggregation algorithm proposed in the FL field is simple weighted averaging, with weights corresponding to the ratio of the number of data points that a client holds over all the data points available. Even if using weights to emphasize the important clients is more helpful than just treating every client as the same, it still does not solve the case where the client datasets are similar in size but very different in distribution.

In this thesis, to tackle the limitations mentioned above, we propose to cluster the client datasets differently and use a specific model for each one of those clusters. Instead of using one global model that tries to learn the patterns in every client, the cluster models will focus on learning the intricacies of the clients belonging to their own cluster only. Our primary research goal was to compare the convergence rates and forecasting accuracies of FedAVG with two other CFL algorithms.

1.3 Research Contribution

While new CFL algorithms are frequently developed in federated learning research, the proposed algorithms have yet to be compared against each other; especially in a demand forecasting setting. This thesis aims to fill this gap by evaluating the performance of clustered federated learning algorithms in an energy demand forecasting setting. By doing so, we aim to understand the behaviour of CFL algorithms against each other, and not only against commonly used baselines.

To this end, two popular CFL algorithms will be benchmarked against the simple FedAVG algorithm, in addition to centralized learning baselines. This being said, it would be eventually important to create a more comprehensive framework with numerous CFL algorithms and energy datasets.

Next section will be about other limitations of this thesis that were not specified under 1.1, which are more technical in nature.

1.4 Limitations

Demand forecasting via centralized machine learning methods is itself a complex task whose success depends on many things; such as selecting the correct model, carefully tuning the model's hyperparameters and creating helper features that will help the model to learn faster and better. Optimizing such tasks simultaneously with optimizing the federated learning architecture is a highly time consuming task even if one assumes it to be doable. Thus we will limit ourselves by focusing on several FL scenarios, and by no means our experiments will aim to optimize all of these in a definite manner. Before stating the actual aims of this thesis, we think it would be important to clarify the reasons behind our model choice.

This thesis will use a neural network for forecasting energy demand. There are several reasons behind sticking with this decision and not trying out other approaches that are used in forecasting; such as traditional statistical models like ARIMA or tree-based machine learning models like GBMs. One of these reasons is the neural networks' ability to capture linear or non-linear long-term temporal dependencies in the data in addition to the short-term dependencies; something which the other models cannot skillfully do or cannot do at all.

Another reason is the impossibility of using traditional statistical models in a federated setting, which generally constitute a baseline in centralized learning scenarios. This is due to these models not being able to continuously *learn* as the training progresses. Their predictions depend on some parameters such as *p, q* and *d*, but trying to combine those models will result in ensemble learning, not federated learning; since after the first averaging all of these said parameters controlling the model behaviour will be the same, there will be no point in having another learning round. Thus our choice for the baseline will be different than the baselines used in centralized learning.

Lastly, there are several research articles suggesting the use of neural networks, particularly LSTMs, for forecasting [7, 16]. Although we acknowledge that the findings of these research articles cannot be generalized to every single time-series dataset, we nevertheless think that the usage of LSTMs is somewhat more popular than other machine learning models for forecasting, at least for forecasts under FL scenarios. Thus we won't be focusing on comparably complex models like GBMs or Bayesian Networks, yet admitting the importance of the exploration of such models in FL settings.

Other limitations of this thesis expressed so far are either a research topic of its own, such as hyperparameter tuning under federated learning, or have been carefully explored in other published articles, such as feature engineering for energy demand data [13].

1.5 Outline

This thesis will be an attempt to fill the need of performance evaluation of several Clustered Federated Learning schemes. Motivation behind it was discussed in 1.1, the details of the research problem and thesis objectives was explained in 1.2, research contribution was discussed in 1.3 which is then followed by some technical limitations in 1.4. Section 2 provides an extensive literature review and a preliminary technical background for the thesis. Section 3 describes the FL algorithms used, while section 4 presents the experimental results. Section 5 presents potential research directions for future. We will conclude by presenting an overall summary of the thesis in section 6.

2 Background

This section will present the work done on solving the statistical heterogeneity problem in federated learning and on federated energy demand forecasting. Although two approaches are not strictly separated, with the latter can be grouped under the former, we still think it would be wise to discuss the latter one separately; so not to lose focus of this thesis. After providing the literature review on statistical heterogeneity and federated energy demand forecasting, we will briefly explore the nature of time series forecasting problems, since energy demand data is an example of time-series data. We will finish the chapter by explaining the deep learning model used to obtain demand forecasts.

2.1 Statistical Heterogeneity Problem

Using data sources which have dissimilar distributions is a problem acknowledged even in the first paper published [10] in the federated learning domain. Since that time, methods used for tackling the statistical heterogeneity of clients under federated learning became a lot more varied that several surveys have been published for the topic [17–19]. Due to this fact, we will briefly mention certain methods that are relevant for this thesis;

in addition to the ones which are not directly relevant yet well-known in the domain. Attempting to conduct an exhaustive review is out of scope for this thesis. We will mainly follow the classification put forward by [19].

Probably the easiest way to deal with non-iid client data is by employing data-centric methods. These methods generally violate the data privacy assumption of federated learning to a small degree with the intention to craft a model that is robust to statistical heterogeneity. For example, X proposes to pool a small percentage of client data in the server followed by training the global model on such representative data before sending it to clients. This way the model sent by server to clients would be prepared beforehand to different client data distributions just by providing a small percentage client data to the server. While these data-centric methods may not be acceptable for some training scenarios due to the clients not wanting to share any part of their data at all; they are suitable for cases where clients are open to collaborate with each other to a degree.

In addition to data-sharing, model-based approaches can also be effective in terms of results. One way to deal with the non-iid nature of data in federated learning is to use a global model which can be adjusted to the client data. To this end, Hanzely et al [20] added a regularization term to the global cost function of FedAVG algorithm, so that each device will learn a mixture of global and local model. On the other hand, one can also adjust the behaviour of the global model to every client by adding a personalization layer on top of the base layers of the global model. The algorithm, FedPer, proposed by Arivazhagan et al [21] applies FedAVG only to base layers while the personalization layers are trained locally. While these algorithms have low communication cost and do not demand much computation power on the client-side, their performance is yet to be tested on different types of data other than benchmarks such as FEMNIST. In most cases they are more efficient than the FedAVG baseline, yet they may not produce results which are more accurate than other methods such as Federated Multi-Task Learning.

Algorithms classified under Federated Multi-Task Learning(FMTL) try to produce a unique model for every client in the network to deal with clients' differing data distributions. One well-known example to this is the MOCHA [22] algorithm. By using a primal-dual optimization method, MOCHA tries to learn similarities and differences for each client by creating a model special to each of them. The drawback of it though is its unsuitability for using non-convex optimization tasks. FMTL algorithms generally deal with statistical heterogeneity very well; yet they may have some particular constraints such as the structure of the client network being known beforehand [23]. In short it is difficult to deny the promise of FMTL in federated learning, yet more research is needed to better understand the effectiveness of such algorithms.

Clustered Federated Learning(CFL) is the algorithm family we will use in our benchmarks in this thesis. CFL assumes that clients with significantly different data distributions belong to different clusters in the network, so the goal is correctly identifying such clusters and then running models on a per-cluster basis. In simple terms, CFL tries to create one global model unique to every cluster, after detecting the clusters themselves. At the moment of writing this thesis, available CFL articles are already numerous so it can be said that the CFL methods are in a maturing phase. We will briefly mention these works before presenting the advantages and disadvantages of CFL.

The simplest approaches in clustered federated learning use the existing features in data, which are not privacy-sensitive, to cluster the clients. If such features are unavailable or privacy-sensitive, then a feature engineering approach can be used to get an understanding of the client context. As long as such features exist and can be shared by the server, a clustering algorithm such as K-Means or DBSCAN can be used to cluster the clients. After this step every cluster will have its own model which can be trained by FedAVG.

Another way to cluster the clients is treating the local loss value as a signature of the client's data distribution. In principle, it can be assumed that clients with similar datasets will have similar loss values, thus the clustering procedure can be conducted by using them. Ghosh et al [14] propose IFCA, which calculates the loss values of cluster models sent by the server to the clients on the client-side, and using the cluster model with the lowest loss for local training. After the client trainings are completed the server applies FedAVG to the models on a per-cluster basis. An extension of IFCA is proposed by Li et al [24], which incorporates soft-clustering to IFCA.

Similar to the loss based clustering approach, local model weights can be used to determine the data distributions as well. Briggs et al [25] propose a two-phased approach in which the clients are trained with normal FedAVG for a certain number of warmup rounds, followed by the application of hierarchical clustering algorithm on the model weights received by the server. After the server clusters the clients based on their model weights, every cluster gets its own cluster model and these are separately trained by FedAVG. Though this method showed its effectiveness regarding federated energy demand forecasting in [11], weight-based clustering has a high communication cost by virtue of using the weights for obtaining clusters.

Last but not least, using the model hyperparameters are also possible to cluster the clients. Musilek et al [26] propose to train the clients together with a local hyperparameter tuning scheme. At certain rounds, the client models' hyperparameters are shared with the server and the server clusters the clients, which is followed by FedAVG on a per-cluster basis.

As CFL only tries to create one specialized model for every cluster, rather than every client, it may produce less accurate results than FMTL; but CFL methods don't require any knowledge about the cluster structure of the data and can be used with non-convex optimization tasks. Likewise, it may be a better choice than global models if the main concern is accuracy, but if communication cost or client-side computation cost is more important then an algorithm such as FedPer should be used. Lastly, even if CFL is quite popular in federated learning research, effectiveness of proposed clustering methods are by no means definite.

In addition to these small sample of algorithm families, several other ways were proposed to deal with statistical heterogeneity; adapting known concepts from centralized deep learning such as transfer learning or batch normalization to the federated learning paradigm. To keep this thesis compact, we won't discuss the peculiarities of such methods; yet more information about them are available in [19].

2.2 Federated Energy Demand Forecasting

To our knowledge; the articles written for federated energy demand forecasting either deal with improving the security of the FL system even more [15, 27], or they deal with improving the forecast accuracy. This thesis falls in the latter group, thus we will not discuss the published research written for secure federated learning, even if we consider it as important.

Nearly all the approaches we encountered so far apply a variant of clustered federated learning to obtain better demand forecasts, and reduce the number of federated learning rounds for convergence, if possible. Hong et al [12] compared vanilla FedAVG with a CFL approach using OPTICS algorithm on extracted features from the data; such as house type, number of rental units and heating type. While these features are not always available; their method needed at most 60 rounds before all clusters to reach convergence.

Instead of coming up with a new algorithm, Savi et al [13] designed a complex experiment setting in their work by focusing on feature engineering, hyperparameter tuning and simulation of different training scenarios that tries to mimic real life. Algorithm-wise, they used an approach that clusters the clients based on their socioeconomic aspects; in addition to clustering the clients via K-Means based on the statistics extracted from their energy consumption patterns, such as consumption mean and median. They were able to achieve remarkable forecasts in terms of accuracy in addition to explaining the importance of several hyperparameters on federated training, but one drawback of their work is data privacy since they assume that a small amount of energy data is publicly available for training.

Different from the mentioned approaches, Wang et al [28] adapted the k-means clustering algorithm to the federated learning setting, which can be used directly on the data without the need of any extracted features. They tried their method on two different energy demand datasets and purely on an unsupervised way, by just measuring the quality of extracted clusters according to different clustering metrics.

2.3 Time-Series Forecasting

As its name suggests, the essence of a time-series forecasting problem is predicting the future value of a certain feature; by using the past and present values of the feature itself, generally in combination with the values of some other helper features in order to obtain more accurate forecasts. Examples to this problem include predicting the stock market prices of a company's shares based on the past prices of those shares, predicting the energy supply produced by a solar panel or predicting the energy demand of a household, which is the main focus of this thesis.

The importance of correctly predicting the future energy demand is crucial for grid operators. Overestimating the demand may cause the grid to malfunction, while underestimating it will pave the way for blackouts. Consequently, accurate forecasts are important not only for the firm operating the grid, but also for the entities bound to the grid, whether they represent households or factories.

The nature of the time series data must be understood first before understanding the forecasting methods. Thus, this section will discuss it first before moving on to discuss the varied ways used for forecasting.

2.3.1 Time-Series Data

In its simplest form, a dataset for a forecasting problem come in the form of **(time, value)** pairs. An observation in such datasets include one or several values for every time index. For example, the table below represents the total energy usage of one household with respect to time; between midnight and 9 am, January 1st 2013.

DateTime	**kWh(kilowatts per hour)**
2013-01-01 00:00:00	0.074
2013-01-01 01:00:00	0.000
2013-01-01 02:00:00	0.000
2013-01-01 03:00:00	0.102
2013-01-01 04:00:00	0.090
2013-01-01 05:00:00	0.087
2013-01-01 06:00:00	0.087
2013-01-01 07:00:00	0.085
2013-01-01 08:00:00	0.085
2013-01-01 09:00:00	0.085

Time-series data generally have three crucial elements characterizing it. These components are called as trend, seasonality and noise. Being able to correctly extract such components helps the analyst during model selection and maybe even for engineering new features that can be used in the predictive model. It is also possible to encounter observations without any trend, seasonality and noise through time, yet such constant series are trivial for analysis.

Trend describes the increasing or decreasing behaviour of the observations through time. For example, energy usage in a household may increase from 5 pm to 9 pm, where the house is mostly occupied and the occupants are generally active in the house. The trend of a time-series dataset can follow many patterns, in other words, it can be linear or non-linear.

Seasonality represents the cyclical variations of the data over time. A good example to seasonality is the sales of plane tickets, where the demand generally increases in summer months followed by a decrease after the end of vacation season. Depending on the type of observations; seasonal patterns may occur yearly, monthly, weekly or even hourly. Like trend, seasonality can be linear(cycles with constant width or amplitude) or non-linear(cycles with changing width and amplitude).

Lastly, **noise** refers to the random variation in time series which is not captured by trend or seasonality components.

The components described henceforth can be combined in different ways to constitute a time series dataset. In other words, the observations may have a linear trend and nonlinear seasonality, or vice-versa. There exists several methods to extract the time series components for further analysis, but for simplicity this section will only describe additive and multiplicative decomposition.

Additive decomposition assumes that the series are formed with a linear trend and linear seasonality.

On the other hand, multiplicative decomposition assumes that the series are formed with non-linear trend and non-linear seasonality over time.

It is not easy to detect the trend and seasonality from time-series data due to the different ways the components can come together. Even in cases where one succeeds in such a task, the noise component may prevent the models from making accurate predictions.

2.4 Recurrent Neural Networks

The biggest advantage of Recurrent Neural Networks(RNN) [29] over the feed-forward ones is the former's ability to use the past information present in the data when predicting the future. This is mainly due to the cell memory present in the architecture of an RNN, which is absent in a simple artificial neuron. In technical terms, the memory of an RNN cell is called as its hidden state, **h**.

One layer of an RNN is just a connection of certain number of recurrent neurons, which take the output of the previous neuron in addition to the actual input data being fed to it. After doing some simple summation and multiplication operations with both inputs, the recurrent neuron then outputs a number which would be used by the next neuron. The intuition behind this process is probably best understood by a visual, which is given below.

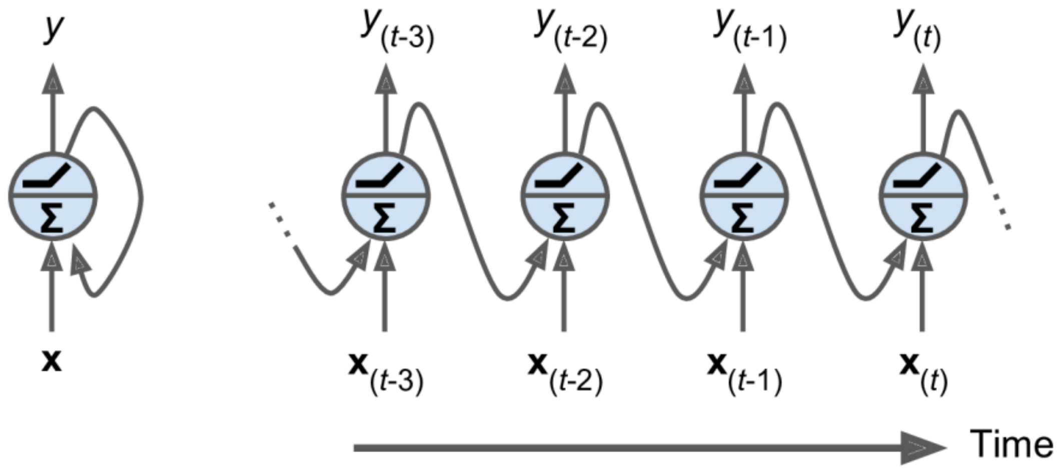


Figure 1. A single recurrent neuron(left) and an RNN Layer(right). Source: Geron [30]

The image above shows an RNN architecture which uses the eventual output **y** to be used in the next neuron. While this is perfectly possible, there are other types of RNN architectures which are a little bit more complex than the one shown above. In actuality, an RNN cell outputs two vectors, **y** and **h**. These vectors can be equal to each other as in the simple RNN architecture above, or they may be different. Similarly, the recurred value can be the value of the hidden state **h**, or the output value **y**. Figure 2 below shows another recurrent neural network but with a hidden state recurrence.

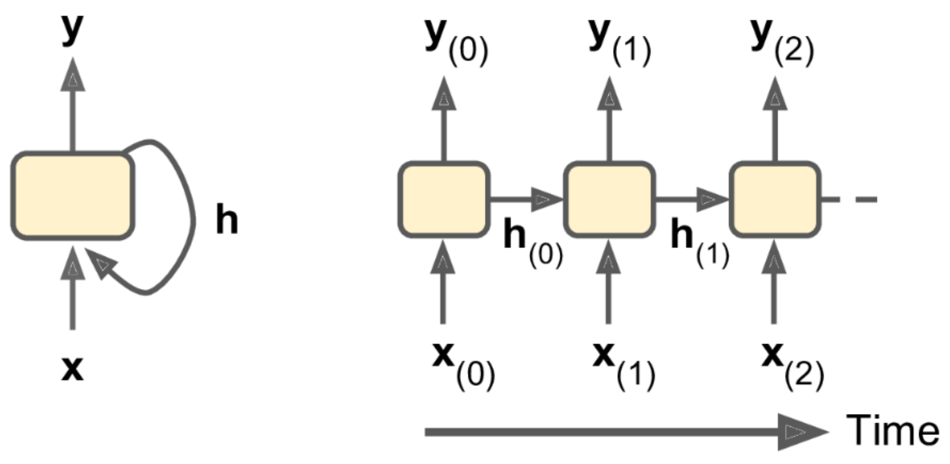


Figure 2. An RNN with hidden state recurrences. Source: Geron [30]

It is also important to clarify the hidden state value that the very first neuron in a recurrent layer gets. For example, in Figure 2 the first neuron takes the input $\mathbf{x_0}$ and produces $\mathbf{y_0}$ and $\mathbf{h_0}$. This neuron seems not to be using any hidden state input at all, but actually this is not the case. Due to the fact that there is no neuron coming before it, it initializes its hidden state value to 0, and uses it together with $\mathbf{x_0}$ in its operations.

With several preliminaries are now settled, one can start to delve deep into the workings of a recurrent neuron. We will use the second neuron in Figure 2 to explain the mathematics behind RNNs. There, $\mathbf{x_1}$ represents the vector input corresponding to timestep t=1, and $\mathbf{h_0}$ is the hidden state output of the recurrent neuron dealing with timestep t=0. Both of these vectors have their own weight matrices, which can be called as $\mathbf{W_x}$ and $\mathbf{W_h}$. The first operation that the recurrent unit does is taking the aforementioned inputs and multiplying them with their weights, followed by summing the results with a bias term, $\mathbf{b_h}$. The resulting is then given into an activation function $\mathbf{f_h}$, which produces the current hidden state value.

$$\mathbf{h_1} = \mathbf{f_h}(\mathbf{W_x}\mathbf{x_{(1)}} + \mathbf{W_h}\mathbf{h_{(0)}} + \mathbf{b_h}) \tag{1}$$

After the hidden state value of the cell is calculated, the neuron then outputs its target value, $\mathbf{y_1}$. Similar to the operations above, y value is calculated by multiplying the hidden state corresponding to present time, $\mathbf{h_1}$, together with a weight matrix $\mathbf{W_y}$. This result is then summed up with a bias term $\mathbf{b_y}$ and fed into a certain activation function $\mathbf{f_y}$.

$$\mathbf{y_1} = \mathbf{f_y}(\mathbf{W_y}\mathbf{h_{(1)}} + \mathbf{b_y}) \tag{2}$$

The operations described so far one neuron can be generalized to any neuron in the recurrent layer with the set of equations below.

$$\mathbf{h_t} = \mathbf{f_h}(\mathbf{W_x}\mathbf{x_{(t)}} + \mathbf{W_h}\mathbf{h_{(t-1)}} + \mathbf{b_h}) \tag{3}$$
$$\mathbf{y_t} = \mathbf{f_y}(\mathbf{W_y}\mathbf{h_{(t)}} + \mathbf{b_y}) \tag{4}$$

The operational pattern described so far repeats until all the vectors corresponding to different timesteps are processed. For example, if the main input tensor $\mathbf{X}$ includes vectors with observations taken from t=0 to t=2, then the recurrences continue until $\mathbf{x_2}$ is reached and its output $\mathbf{y_2}$ is returned.

Even though the simple recurrent neural networks are able to use the contextual information when making predictions, the context they refer to generally falls short while working with more complex data. For example, if an input instance $\mathbf{x}$ contains values spanning 24 timesteps in total, the first timestep value will go through 23 different mathematical operations before affecting the prediction given by 24th neuron. The memory of the network regarding the initial input gets worse as the training progresses, since backpropagation algorithm may not affect any changes for the weights of the first input, because of the gradients getting smaller and smaller from the neurons at the end of the network to the neurons located at the start. One effective way to counter this problem is to use a more complex neural unit for computations, called as Long-Short Term Memory cell.

2.4.1 Long-Short Term Memory Networks

Long-Short Term Memory(LSTM) [4] Networks are a special type of RNNs which are capable of memorizing a much wider contextual information than the simple RNNs. While the working principle of LSTMs are very similar to that of RNNs, there are some differences between the two because of the more powerful neurons(also called as cells) that the LSTM networks use. It would be best to start with a visual description of such cells before presenting the details.

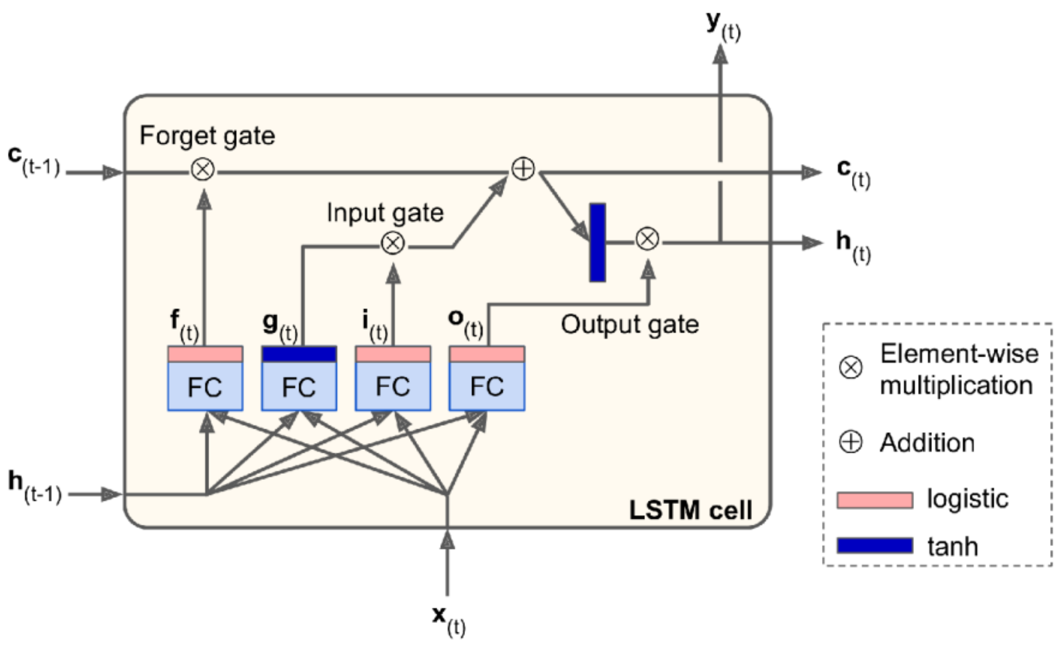


Figure 3. A single LSTM neuron. Source: Geron [30]

Probably the first difference that strikes the eye is the addition of a new vector, $\mathbf{c_t}$, which is called as the cell state. This state tries to store the long-term memory of an LSTM cell, while the hidden state $\mathbf{h_t}$ trying to store the short-term memory. These two variable vectors, in addition to the input $\mathbf{x_t}$, interact through several gates inside the cell to produce the outputs of an LSTM cell.

Let us start with the operations that the cell state goes through. For the timestep t=t, the LSTM neuron gets the value of the previous cell state output, $\mathbf{c_{t-1}}$. This vector is then multiplied element-wise with $\mathbf{f_t}$ in the **forget gate**. The reason for the naming is due to the nature of the logistic activation function that produces $\mathbf{f_t}$, which forces $\mathbf{f_t}$ to take values between 0 and 1. So forget gate either drops or keeps some values inside the cell state vector.

After passing the forget gate, the cell state vector is summed together with the output vector returned from the input gate. The resulting combination forms the new cell state.

This new cell state is also copied and passed into the hyperbolic tangent activation function, which in turn is sent to the **output gate** for another element-wise multiplication. Output gate simply takes two vectors, $\mathbf{tanh(c_t)}$ and $\mathbf{o_t}$ to produce the new hidden state vector $\mathbf{h_t}$ and the LSTM cell's target output $\mathbf{y_t}$. In simple terms, it acts as yet another filter similar to the forget gate, selecting which values from the cell state must be stored in $\mathbf{h_t}$ and $\mathbf{y_t}$.

Now let us look into what happens to $\mathbf{h_{t-1}}$ and $\mathbf{x_t}$. These input vectors are both sent to four different fully-connected layers, with one having a hyperbolic tangent activation function and the others having a logistic one. These four activation functions produce the outputs $\mathbf{f_t}$, $\mathbf{g_t}$, $\mathbf{i_t}$ and $\mathbf{o_t}$. The role of $\mathbf{f_t}$ and $\mathbf{o_t}$ is already described above in connection to the forget gate and the output gate, so we won't go over them again.

The last gate that hasn't been fully explained is the **input gate**, which takes $\mathbf{g_t}$ and multiplies it element-wise with $\mathbf{i_t}$. Here $\mathbf{i_t}$ acts as a gate controller like $\mathbf{f_t}$ because of the logistic activation function producing it. $\mathbf{g_t}$ on the other hand, is formed by combining the latest hidden state value $\mathbf{h_{t-1}}$ and the current input $\mathbf{x_t}$. So the input gate *selects* which parts of $\mathbf{g_t}$ should be added to $\mathbf{c_{t-1}}$.

We can formalize all of the operations with the equations given below:

$$\mathbf{i_t} = \sigma(\mathbf{W_{xi}x_t} + \mathbf{W_{hi}h_{t-1}} + \mathbf{b_i}) \tag{5}$$
$$\mathbf{f_t} = \sigma(\mathbf{W_{xf}x_t} + \mathbf{W_{hf}h_{t-1}} + \mathbf{b_f}) \tag{6}$$
$$\mathbf{o_t} = \sigma(\mathbf{W_{xo}x_t} + \mathbf{W_{ho}h_{t-1}} + \mathbf{b_o}) \tag{7}$$
$$\mathbf{g_t} = \mathbf{tanh}(\mathbf{W_{xg}x_t} + \mathbf{W_{hg}h_{t-1}} + \mathbf{b_g}) \tag{8}$$
$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g_t} \tag{9}$$
$$\mathbf{h_t} = \mathbf{y_t} = \mathbf{o_t} \odot \mathbf{tanh}(\mathbf{c_t}) \tag{10}$$
$$\tag{11}$$

In the equations above, $\mathbf{W_{xi}}$,$\mathbf{W_{xf}}$,$\mathbf{W_{xo}}$,$\mathbf{W_{xg}}$ are the weight matrices corresponding to the input $\mathbf{x_t}$. Similarly; $\mathbf{W_{hi}}$,$\mathbf{W_{hf}}$,$\mathbf{W_{ho}}$,$\mathbf{W_{hg}}$ are the weight matrices acting upon the last hidden state $\mathbf{h_{t-1}}$.

$\mathbf{b_f}$,$\mathbf{b_i}$,$\mathbf{b_o}$ and $\mathbf{b_g}$ represent the bias vectors of the four different fully connected layers inside the network.

σ is the logistic activation function and tanh stands for the for hyperbolic tangent activation function. $\odot$ represents element-wise multiplication operation.

To sum up, the ability of an LSTM cell regarding the storage of long-term dependencies in addition to the short-term ones in a sequence makes it a suitable choice for using more contextual information from the input data while making predictions. Since most of the time-series data, especially the ones regarding the energy consumption, has high complexity embedded inside them; using LSTMs for prediction would very likely to return more accurate forecasts.

3 Methodology

This section will explain the methods used to tackle the statistical heterogeneity problem in federated energy forecasting. As the focus of this thesis is on benchmarking several methods in a federated energy demand forecasting setting, the algorithms we will propose to use are neither our own nor nothing new.

3.1 Federated Averaging(FedAvg) Algorithm

Federated Averaging [10] is the algorithm that kickstarted the federated learning paradigm in the general discipline of machine learning. Being relatively simple as compared to others, it can be said that FedAvg is a good baseline choice for all federated learning experiments. In this section, we will describe the algorithm by introducing it at a higher, conceptual level. This will be followed by providing its formalism.

As described earlier in the Introduction, a typical federated learning setup includes a **server** and several **clients**(can also be called as nodes). The process begins with the server initializing a model f, which is then copied and sent into all the clients; or at least, the available clients at the moment of sending. The clients then train the model on their training data for a specified number of epochs, and after the local trainings are completed, clients send their models' weights back to the server. The server then averages all these weights and updates the model it initialized with them. This marks the end of one **federated training round**. The following rounds follow exactly the same procedure except the initialization part, since at the start of every following round the server model is initialized with the averaged weights of the client models; while at the very beginning it was initialized randomly by the server.

Figure 4 below visualizes the process we have just described. As the image states, it must be emphasized that the only interaction between the server and the clients are through exchanging the model weights, but not **client data**.

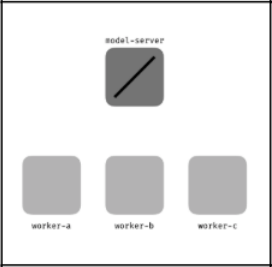
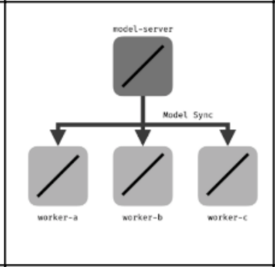
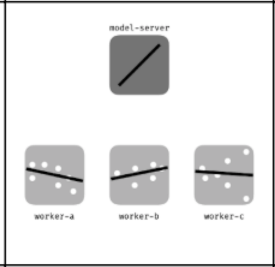
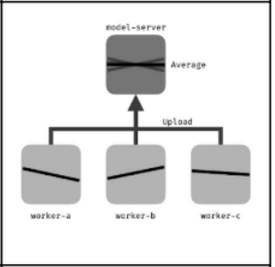
Step 1	Step 2	Step 3	Step 4
	 model-server Model Sync worker-a worker-b worker-c		 model-server Average Upload worker-a worker-b worker-c
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

Figure 4. Federated Averaging. Source: Wikipedia [31]

A more rigorous description of FedAvg algorithm is given below.

Algorithm 1: Federated Averaging

```
1   Server executes:
    initialize $w_0$
    for each round $t = 1,2,\ldots$ do
        $m \leftarrow \max(C \cdot K, 1)$
        $S_t \leftarrow$ (random set of $m$ clients)
        for each client $k \in S_t$ in parallel do
            $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
        end for
        $w_{t+1}^k = \sum_{k=1}^{K} \frac{n^k}{n} w_{t+1}^k$
    end for

    ClientUpdate$(k, w)$: // Run on client k
    B $\leftarrow$ (split into $P_k$ batches of size $B$)
    for each local epoch $i$ from 1 to E do
        for batch $b \in$ B do
            $w \leftarrow w - \eta \nabla \ell(w : b)$
        end for
    end for
    return $w$ to server
```

FedAvg is simple to understand and may turn out to be really effective if the client datasets are coming from the same data distribution. Yet if the datasets are dissimilar to each other; the algorithm may take too many rounds to converge, or may not converge at all. In order to deal with this problem, we will now take a look at clustered federated learning algorithms.

3.2 Federated Learning with Hierarchical Clustering(FL+HC)

Proposed by Briggs et al [25], FL+HC is designed with the intention to deal with client datasets that are not identically distributed. The algorithm itself is very different from vanilla FedAvg, except for the fact that it trying to detect clusters among the clients and conducting the training on a per-cluster basis.

In FL+HC, the server starts the federated training rounds as it was in FedAvg. It initializes the model, sends its copies to clients, averages the client models' weights which are sent to it and updates the server model with the averaged weights. For a set number of warmup rounds, this iterative process continues. Right after the warmup rounds are completed, ie in the subsequent round following the last warmup round, the server clusters the clients based on their model weights. The authors' selected hierarchical clustering algorithm for this procedure, yet one can use a different method. After the clients are clustered based on their weights; the server runs FedAvg inside each cluster, aggregating the client weights on a per-cluster basis and sending the copies of a model unique to every cluster instead of sending a single, global model to every client. The training goes on until the set number of rounds are reached, or if the average validation loss over all the clients starts to stabilize.

FL+HC is formalised as Algorithm 2.

Algorithm 2: FL+HC

```
procedure FL+HC                                          ▷ On server
    Initialise $w_0$
    for each round $t \in [1, n]$ do
        $w_{t+1} \leftarrow$ FEDERATEDLEARNING($w_t, K$)
    end for
    $w \leftarrow w_{t+1}$
    for each client $k \in K$ do                         ▷ In parallel
        $\Delta w^k \leftarrow$ CLIENTUPDATE($k, w$)
    end for
    $C \leftarrow$ HierarchicalClusteringAlgorithm($\Delta w, P$)
    for $c \in C$ do                                     ▷ In parallel
        $w_{c,0} \leftarrow w$
        for each round $t = 1, 2, \ldots$ do
            $w_{c,t+1} \leftarrow$ FEDERATEDLEARNING($w_{c,t}, K_c$)
        end for
    end for
end procedure
1
procedure FEDERATEDLEARNING($w_t, K$)                    ▷ On server
    $m \leftarrow \max(\alpha \cdot K, 1)$
    $S_t \leftarrow$ (random set of $m$ clients)
    for each client $k \in S_t$ do                       ▷ In parallel
        $w_{t+1}^k \leftarrow$ CLIENTUPDATE($k, w_t$)
    end for
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
end procedure

procedure CLIENTUPDATE($k, w$)                           ▷ On client $k$
    $\mathcal{B} \leftarrow$ (Split $\mathcal{P}_k$ into batches of size $B$)
    for each local epoch $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $w \leftarrow w - \eta \nabla \mathcal{L}(w; b)$
        end for
    end for
    return $w$ to server
end procedure
```

In their paper, the authors achieve better results with FL+HC as compared to the plain FedAvg algorithm. This being said, FL+HC introduces several hyperparameters like number of clusters and number of warmup rounds that need to be tuned carefully. Using the model weights as a reference for clustering is also problematic as well, due to the possibility that two similar models having different model parameters because of permutation invariance of the model to the hidden units [14]. In our approach, we will be using the training loss of the model obtained at the end of the warmup rounds, not weights, for clustering the clients.

Static clustering method may not accurately capture the client clusters; since if the first clustering attempt fails to capture the cluster structure, the network will not have any chance to correct this. By dynamically updating the client clusters at every round, Iterative Federated Clustering algorithm aims to tackle this potential shortcoming.

3.3 Iterative Federated Clustering Algorithm(IFCA)

In IFCA, the server stores k different models $\theta_j, j \in [k]$. Each of these models are randomly initialized at beginning of the first round and correspond to a cluster, whose number is controlled by hyperparameter k. In the t-th iteration of algorithm, the server randomly selects a number of clients and sends the cluster models to them. The clients then calculate the loss value of every cluster model on their whole dataset by using their loss function F_i. After calculating the losses, the client then selects the cluster model which returns the lowest loss value for training; in addition to storing its own cluster identity j that corresponds to the cluster model θ_j. By using the said cluster model, the client then runs a set number of local training epochs on its own training set and updates its model. The server then receives the clients' cluster identities together with the cluster models they used. The cluster models are then averaged on the server-side, updating the global cluster model list stored in the server.

While model averaging is one option, one can also use gradient averaging to update the cluster models. In this thesis though, we decided to use the former due to its simplicity. Both of these options can be seen in Algorithm 3 below.

Algorithm 3: IFCA

```
1 Input: number of clusters $k$, step size $\gamma$, $j \in [k]$, initialization $\theta_j^{(0)}$, $j \in [k]$
  number of parallel iterations $T$, number of local gradient steps $\tau$ (for model averaging).
  for $t = 0, 1, \dots, T-1$ do
    center machine: broadcast $\theta_j^{(t)}$, $j \in [k]$
    $M_t \leftarrow$ random subset of worker machines (participating devices)
    for worker machine $i \in M_t$ in parallel do
      cluster identity estimate $\widehat{j} = \mathrm{argmin}_{j \in [k]} F_i(\theta_j^{(t)})$
      define one-hot encoding vector $s_i = \{s_{i,j}\}_{j=1}^k$ with $s_{i,j} = \mathbf{1}\{j = \widehat{j}\}$
      option I (gradient averaging):
        compute (stochastic) gradient: $g_i = \widehat{\nabla} F_i(\theta_{\widehat{j}}^{(t)})$, send $s_i$, $g_i$ to center machine
      option II (model averaging):
        $\widetilde{\theta}_i = \mathsf{LocalUpdate}(\theta_{\widehat{j}}^{(t)}, \gamma, \tau)$, send $s_i$, $\widetilde{\theta}_i$ to center machine
    end for
    center machine:
    option I (gradient averaging): $\theta_j^{(t+1)} = \theta_j^{(t)} - \frac{\gamma}{m} \sum_{i \in M_t} s_{i,j} g_i$, $\forall\ j \in [k]$
    option II (model averaging): $\theta_j^{(t+1)} = \sum_{i \in M_t} s_{i,j} \widetilde{\theta}_i / \sum_{i \in M_t} s_{i,j}$, $\forall\ j \in [k]$
  end for
  return $\theta_j^{(T)}$, $j \in [k]$
  $\mathsf{LocalUpdate}(\widetilde{\theta}^{(0)}, \gamma, \tau)$ at the $i$-th worker machine
  for $q = 0, \dots, \tau - 1$ do
    (stochastic) gradient descent $\widetilde{\theta}^{(q+1)} = \widetilde{\theta}^{(q)} - \gamma \widehat{\nabla} F_i(\widetilde{\theta}^{(q)})$
  end for
  return $\widetilde{\theta}^{(\tau)}$
```

4 Experiments

Here we present the experiments conducted with the intention to answer the research goals stated in Introduction. Our experiments will be about simulating a real life federated learning setting in one machine, so the computational resources we used are limited. The choices made regarding the dataset preparation and hyperparameters specific to federated learning is a consequence of this. In all our experiments, the evaluation metric we used was the mean of mean squared error over all client test sets. If we set $\hat{y}$ as the predicted energy consumption value, y as the real value and n as the number of data points in clients' test set, we can define the MSE for one client as:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{12}$$

If we have the MSE values of k clients, taking the mean of all these client MSEs will give us our final evaluation metric:

$$\frac{1}{k}\sum_{i=1}^{k}(MSE_i) \tag{13}$$

Again, we stress that these loss calculations are made by using the client test sets only.

4.1 Dataset Preparation

For our experiments we used the dataset published by UK Power Networks [32]. This dataset contains the energy consumption values of 5567 households located in London from November 2012 to February 2014, in the form of half-hourly measurements recorded in kilowatts. It also has another feature representing the pricing type used, which is either a standard flat-rate electricity billing tariff or a dynamic tariff based on time.

Since we are simulating the FL setup in one machine, we decided to select the energy data from January 1, 2013 to July 3, 2013. The resulting dataset is filtered further by randomly selecting 50 households among all the available ones, whose energy consumption values are recorded with standard pricing. We then resampled the dataset so that it represented the consumption values in one hour, instead of half an hour. The last operation we conducted was to scale the consumption values with MinMaxScaler, so that the values will be limited between 0 and 1.

The size of the training set was selected as 0.7 times the size of the whole dataset. The respective sizes of validation set and test set was 0.2 and 0.1 times the size of the dataset.

We selected a time window of 12, ie the values of past 12 hours were fed to the model so that it can forecast the 13th hour.

4.2 Model Architecture

For getting the predictions, we used a similar model architecture to the scalable LSTM model put forward by [33]. Our model has two hidden layers followed by a dense layer. The first hidden layer had 32 LSTM cells and the second one had 16. Both layers had a dropout rate of 0.1 and used hyperbolic tangent as activation function. The dense layer, or the output layer, had only one neuron whose task is to return the 13th hour's forecast. The input layer took observations consisted of past 12 hours, ie the timestep dimension was equal to 12 and feature dimension was equal to 1.

In addition to the info above, the selected batch size was 512 and the learning rate for gradient descent updates was 0.002, while the loss function used for training is mean squared error. Number of training epochs was varied, and will be mentioned separately.

4.3 Centralized Forecasting Experiments

Being the first baseline, the setting for this experiment was rather simple. The consumption values from the randomly selected 50 households was concatenated and multi-output predictions was made with the LSTM model defined in 4.2. The only peculiarity of this setting was the number of training epochs, which was set to 200, and the number of early stopping rounds was 20.

The obtained MSE value with centralized forecasting was 0.1169.

4.4 Distributed Forecasting Experiments

This setting is similar to the federated learning experiments, with the only difference being the fact that the clients' model weights not being averaged. As a result, the LSTM models are trained for only one round inside the clients and neither data nor the model weights are transferred between the clients and the server.

For this setting, the number of local training epochs was set to 200 and the number of early stopping rounds was 20, as in centralized forecasting setting. The obtained mean MSE value over all clients was 0.1874.

4.5 Experiments with FedAVG

As it was stated in introduction, FedAvg is the algorithm that we will try to beat in our experiments. Besides functioning as a reference algorithm, we ran experiments with FedAvg in order to see how the number of local epochs and number of federated learning rounds will affect the overall performance. This is done with two intentions; one is to observe the behaviour of FedAvg regarding its convergance, the other is to see if it would make sense to reduce the number of rounds needed for the more computationally demanding clustering algorithms later. To that end, we decided to train for 25 and 50 rounds; and in each setting we set the number of local training epochs to 5 and 10. While selecting the epochs; we aimed to not select too high a number as doing that may be unrealistic in a real life FL scenario, in case such a scenario is conducted by clients with small computational power like sensors. On the other hand, using a too small number for epochs would be unrealistic either; since time-series data are essentially quite complex and a learning model needs some time to learn the temporal dependencies, otherwise underfitting is likely to occur.

Below you can see the plots showing the overall MSE on the y-axis versus the number of FL rounds on the x-axis. On the left we will plot the trainings conducted with 5 epochs, and on the right we will show the trainings done with 10 epochs. We will begin by showing the results of 25 rounds on Figure 5, and the results of 50 rounds of training will be shown on Figure 6.

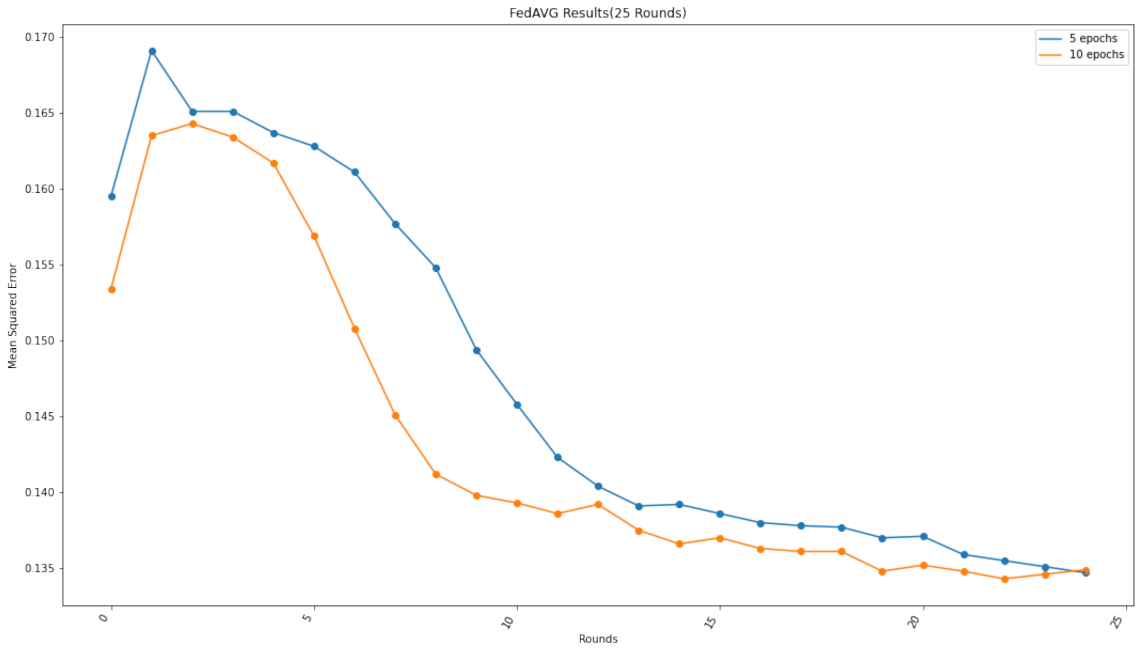


Figure 5. MSE with 25 rounds of FedAvg.

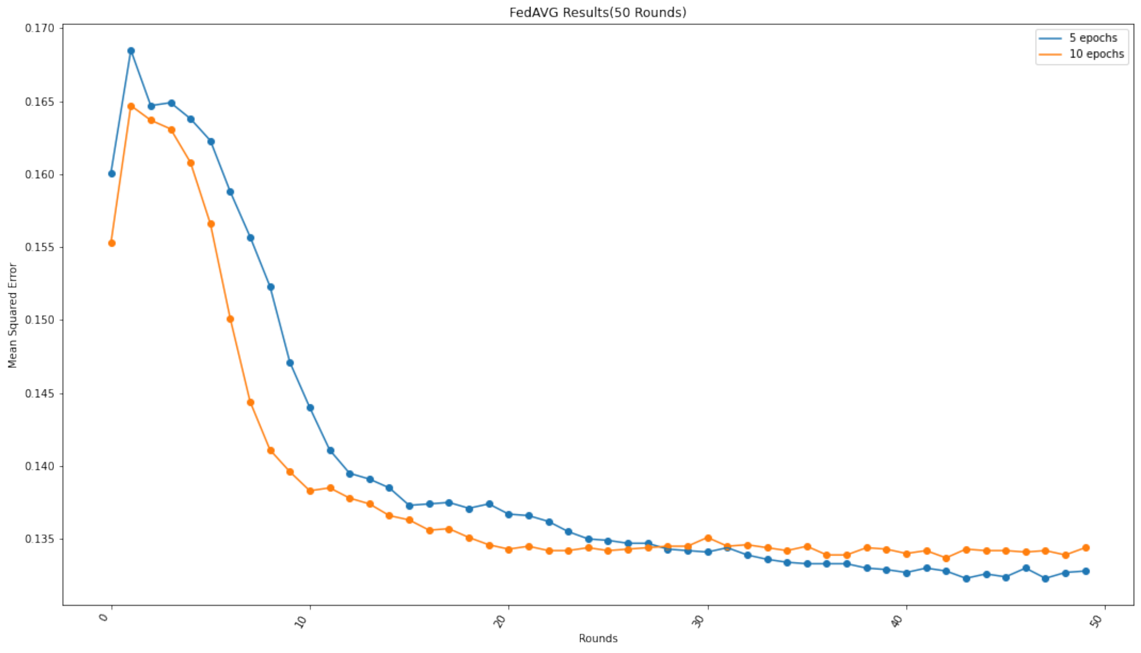


Figure 6. MSE with 50 rounds of FedAvg.

The final results are shown in Table 1 below.

	25 Rounds	50 Rounds
5 Epochs	0.1347	0.1328
10 Epochs	0.1349	0.1344

Table 1. Final MSE values for FedAVG

The results get better as the number of training rounds are increased, though this improvement is much more felt in training setting conducted with 5 epochs. This being said, the difference doubling the amount of training rounds don't bring too much improvement to the forecasting accuracy; with the difference being 0.0019 points(or 1.4 percent improvement) between 50 rounds of training with 5 epochs and 25 rounds of training with 5 epochs. Due to this, we decided to keep the total number of rounds equal to 25 from now on, in order to better make use of the available computational resources. In addition to fixing the rounds to 25, the epochs will also be fixed to 5 since using 10 local epochs seems to overfit the datasets.

One interesting observation is that the shape of the MSE curves, which is not at all surprising. Right after the first round, the MSE values jump and then start to decrease gradually. This MSE spike in the second round is due to averaging the models which are trained on datasets that are dissimilar from each other.

4.6 Experiments with FL+HC

For our second experiment setting, we used the initial 6 rounds for warmup rounds for the FL+HC algorithm, while the remaining 19 rounds were clustered federated learning rounds. We fixed the epochs to 5 and set the number of clusters to 2,3 and 5.

The results can be seen below. As it can be seen from the graph, changing the cluster count didn't change the MSE values very much. 2 clusters yielded the best MSE, which is 0.1352. 5 clusters returned a value of 0.1354 and 3 returned 0.1358.

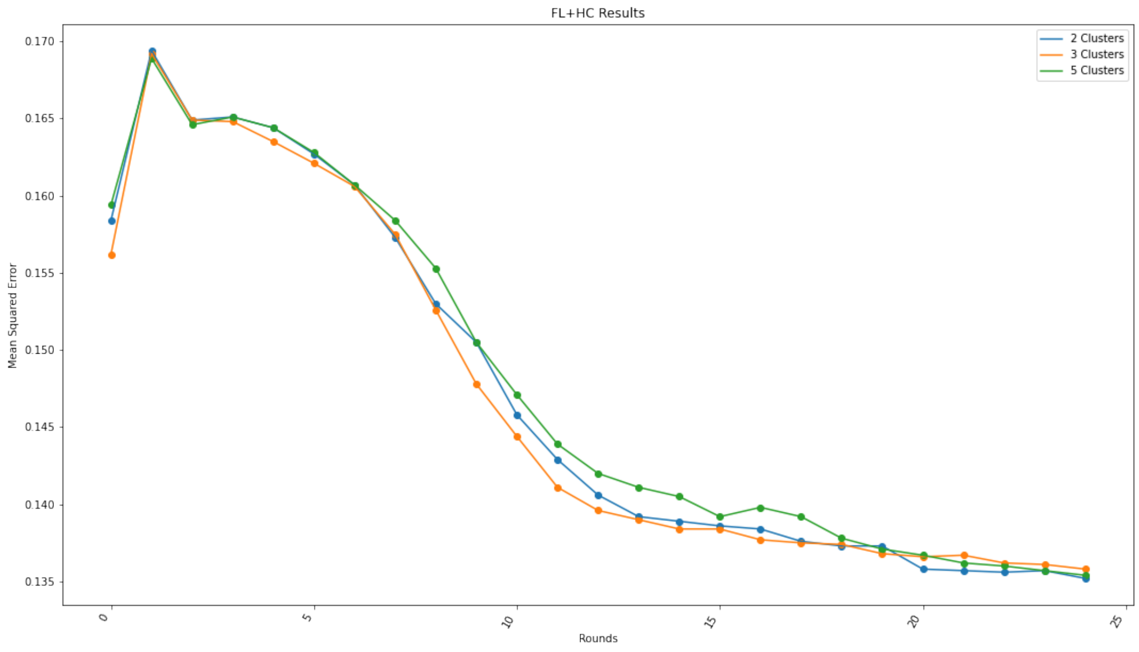


Figure 7. MSE with 25 rounds of FL+HC.

4.7 Experiments with IFCA

IFCA's settings was the same as that of FL+HC's; with epochs equal to 5, FL rounds set to 25 and the number of available clusters to the clients was 2, 3 and 5. We expected this algorithm to produce better results and converge quickly.

Plots of IFCA experiment with 2,3 and 5 clusters can be seen below. The results given by 5 clusters was the best among all, obtaining an MSE value of 0.1320.

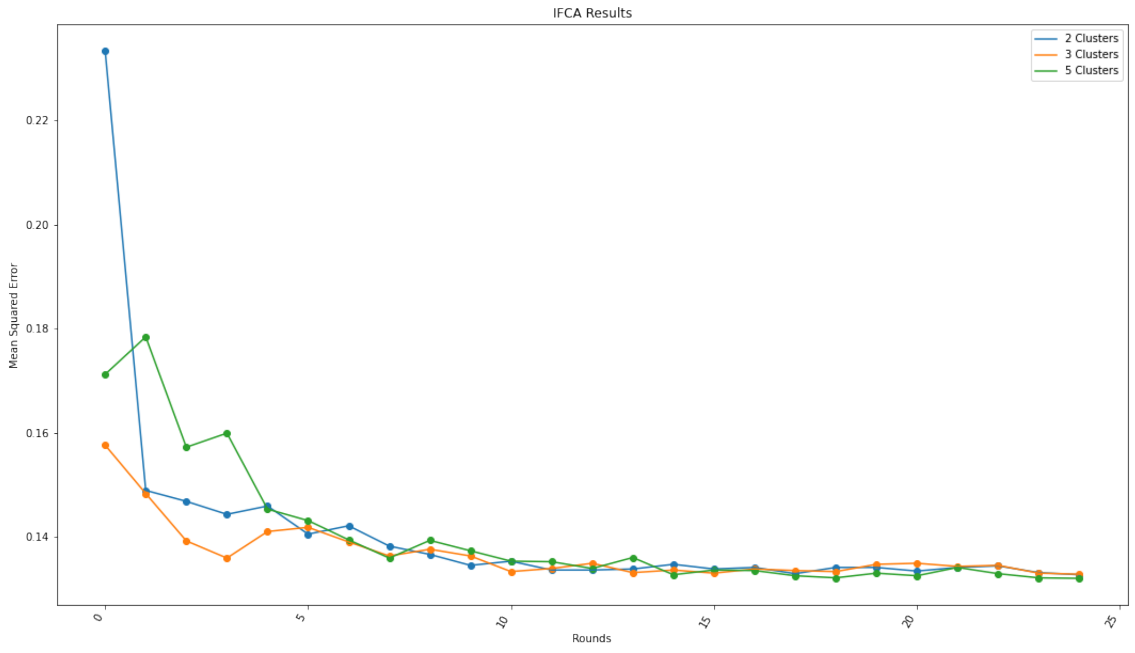


Figure 8. MSE with 25 rounds of IFCA.

Notice the sharp decrease in error for all three settings, in addition to the higher forecasting accuracy returned by IFCA, which surpassed that of FedAVG in just 25 rounds in all training scenarios with cluster count set to 2,3 or 5. Even if the decrease in MSE is not too much compared to the best MSE achieved by FedAvg, IFCA used only 25 FL rounds to reach these metrics. In addition to that, the convergence was much faster.

5 Further Research

As we have stated in 1.1, this thesis is not an exhaustive study of the performances of existing CFL algorithms for energy demand forecasting. Even if the algorithms we selected are quite popular, we have only applied 2 of them, which resulted in many other CFL algorithms to be left out. One research direction can be to expand this work by benchmarking with multiple other CFL algorithms. Other types of popular FL methods that can counter the statistical heterogeneity among the clients can be added to such a benchmark to get a more holistic view on performances. Similarly, using different datasets preferably from different geographical regions would be a significant addition on top of this work.

Another research direction can be about measuring the defensive capabilities of the server-client network in different settings. FL systems are assumed to be secure by design, yet they are not absolutely secure systems. For example, the client datasets can be somewhat recovered by reverse engineering the model weights. If the server is attacked and those weights are recovered, an external threat can discover the client data to some extent. To prevent such security shortages from happening, several encryption techniques such as differential privacy or secure multi party computation are used in combination with FL algorithms. These techniques bring their own tradeoff, the may reduce the forecasting accuracy or may increase the computation cost in the clients. A good way to understand all these tradeoffs and also to measure the defensive capabilities they bring is to experiment with such techniques in different attack scenarios.

Last but not least, measuring the communication costs between the clients and the server is also important. The experiments in this thesis were simulated on a single machine, and as a result the communication costs between server and clients are not measured and the effect of network failures on some clients are not observed. Limited bandwith or low internet connectivity can be significant constraints on the FL network if it is to be deployed in a real-life setting.

6 Conclusion

As more and more energy consumption data is produced by individual households, it becomes important to model them effectively and securely. Federated deep learning is one way to achieve both of these goals, since it is capable to scale better and return more secure forecasts as compared to centralized deep learning. Yet as the number of houses increase, the forecast quality may decrease due to the fact that data exchanges from households to server is not allowed in FL, which uses a single model to discover the patterns in every household simultaneously. This is called as statistical heterogeneity problem in FL.

We tried to solve the aforementioned problem by applying several Clustered Federated Learning algorithms to household energy data. Our experiments have shown that all algorithms have performed worse yet comparable to centralized forecasts, and did not improve the FedAVG baseline significantly. Even though such experimental results are unexpected, this thesis showed that a more secure system can be designed with federated learning if one is willing to sacrifice a little bit from the performance obtained with centralized forecasts. It must also be noted that this work is not exhaustive and experiments with different datasets are needed to understand the performance of used CFL algorithms. In addition, CFL algorithms differ among themselves, the ones not used in this thesis could have returned better results.

In the near future we aim to expand this benchmark with new energy consumption datasets and new CFL algorithms. For the long-term, we aim to add more FL algorithms that can deal with statistical heterogeneity among the clients.

7 Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Canberk Özen**

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   A Collaborative Approach for Large-scale Electricity Consumption Using Federated Learning,

   supervised by Feras Awaysheh and Sadi Alawadi.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Canberk Özen
08/08/2022

References

[1] "Research and markets. global smart meters market—growth, trends, forecasts (2020—2025)," December 23, 2020. [Online]. Available: shorturl.at/hqKNS. [Accessed: February 30, 2022].

[2] "Smart meter benefits: Role of smart meters in responding to climate change," May 2019. [Online]. Available: shorturl.at/gorsS. [Accessed: February 30, 2022].

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[4] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[5] J. Q. Wang, Y. Du, and J. Wang, “Lstm based long-term energy consumption prediction with periodicity,” *Energy*, vol. 197, p. 117197, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544220303042

[6] Y. Xu, W. Gao, F. Qian, and Y. Li, “Potential analysis of the attention-based lstm model in ultra-short-term forecasting of building hvac energy consumption,” *Frontiers in Energy Research*, vol. 9, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/fenrg.2021.730640

[7] S. Muzaffar and A. Afshari, “Short-term load forecasts using lstm networks,” *Energy Procedia*, vol. 158, pp. 2922–2927, 2019, innovative Solutions for Energy Transitions.

[8] H. Chihoub and C. Collet, “A scalability comparison study of data management approaches for smart metering systems,” in *2016 45th International Conference on Parallel Processing (ICPP)*, 2016, pp. 474–483.

[9] F. G. Mármol, C. Sorge, O. Ugus, and G. M. Pérez, “Do not snoop my habits: preserving privacy in the smart grid,” *IEEE Communications Magazine*, vol. 50, no. 5, pp. 166–172, 2012.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282.

[11] C. Briggs, Z. Fan, and P. Andras, “Federated learning for short-term residential energy demand forecasting,” 2021. [Online]. Available: https://arxiv.org/abs/2105.13325

[12] Y. L. Tun, K. Thar, C. M. Thwal, and C. S. Hong, “Federated learning based energy demand prediction with clustered aggregation,” in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2021, pp. 164–167.

[13] M. Savi and F. Olivadese, “Short-term energy consumption forecasting at the edge: A federated learning approach,” *IEEE Access*, vol. 9, pp. 1–21, 07 2021.

[14] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, “An efficient framework for clustered federated learning,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 19 586–19 597.

[15] J. D. Fernandez, S. P. Menci, C. Lee, and G. Fridgen, “Secure federated learning for residential short term load forecasting,” 2021. [Online]. Available: https://arxiv.org/abs/2111.09248

[16] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, “Short-term residential load forecasting based on lstm recurrent neural network,” *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2019.

[17] P. Kairouz, H. McMahan, B. Avent, A. Bellet, M. Bennis, A. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. D’Oliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. Gibbons, and S. Zhao, *Advances and Open Problems in Federated Learning*, 01 2021.

[18] V. Kulkarni, M. Kulkarni, and A. Pant, “Survey of personalization techniques for federated learning,” 07 2020, pp. 794–797.

[19] H. Zhu, J. Xu, S. Liu, and Y. Jin, “Federated learning on non-iid data: A survey,” *Neurocomputing*, vol. 465, 09 2021.

[20] F. Hanzely and P. Richtárik, “Federated learning of a mixture of global and local models,” 2020. [Online]. Available: https://arxiv.org/abs/2002.05516

[21] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, “Federated learning with personalization layers,” 2019. [Online]. Available: https://arxiv.org/abs/1912.00818

[22] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, “Federated multi-task learning,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[23] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, “Networked federated multi-task learning,” 2021. [Online]. Available: https://arxiv.org/abs/2105.12769

[24] C. Li, G. Li, and P. K. Varshney, “Federated learning with soft clustering,” *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7773–7782, 2022.

[25] C. Briggs, Z. Fan, and P. Andras, “Federated learning with hierarchical clustering of local updates to improve training on non-iid data,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–9.

[26] N. Gholizadeh and P. Musilek, “Federated learning with hyperparameter-based clustering for electrical load forecasting,” 2021. [Online]. Available: https://arxiv.org/abs/2111.07462

[27] J. Gao, W. Wang, Z. Liu, M. F. R. M. Billah, and B. Campbell, "Decentralized federated learning framework for the neighborhood: A case study on residential building load forecasting," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 453–459.

[28] Y. Wang, M. Jia, N. Gao, L. Von Krannichfeldt, M. Sun, and G. Hug, "Federated clustering for electricity consumption pattern extraction," *IEEE Transactions on Smart Grid*, vol. 13, no. 3, pp. 2425–2439, 2022.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[30] A. Géron, *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[31] "Federated learning - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Federated_learning#/media/File:Federated_learning_process_central_case.png

[32] "Smartmeter energy consumption data in london households – london datastore." [Online]. Available: https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households

[33] A. M. Alonso, F. J. Nogales, and C. Ruiz, "A single scalable lstm model for short-term forecasting of massive electricity time series," *Energies*, vol. 13, no. 20, 2020.