Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics and Machine Learning
2023 | LIU-IDA/STAT-A--23/031--SE

Federated Learning for Power Consumption Forecasting in Radio Base Stations

Farid Musayev

Supervisor : Filip Ekström Kelvinius
Examiner : Annika Tillander

External supervisor : Agustín Valencia

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Energy consumption remains one of the main challenges in mobile telecommunications industry making it vital to design reliable power management systems for radio base stations. An increased automation enabled by artificial intelligence allows to collect data from access nodes and build power consumption forecasting models for radio base stations that enhance efficiency of energy operations. In a conventional way, the data from many radio base stations will be stored in a cloud-based system, and a corresponding statistical or machine learning model will be built using all available data. This is a centralized learning approach. Another approach is localized learning where a local model for each radio base station is trained using only the data available to it. This approach can be considered a fully private setting where the data is kept at its source. While each of those approaches have advantages and disadvantages, an alternative is federated learning. Federated learning is a machine learning paradigm that allows to build a single model trained on distributed datasets without the need to collect and store these datasets in a single place. Federated learning allows to preserve data sensitivity and privacy that is violated in centralized learning and leverage larger amounts of data in comparison to localized learning. The purpose of this study is to explore the application of federated learning for power consumption forecasting in a large number of radio base stations. The results demonstrate that despite the presence of outlying and non-iid radio base stations, a convolutional neural network model implemented in the federated learning scenario performs better than the same model implemented in the localized learning scenario and worse when compared to the same model in the centralized learning scenario according to root mean squared error evaluation metric. However, the presence of outlying and non-iid radio base stations still makes it challenging for the federated model to reach the performance of the centralized model. As the study shows, it is very important to have a good understanding of the impact of the hyperparameters on the federated training process to ensure a model convergence. The choice of these hyperparameters is not only user driven but also depends on the amount of available computational resources and the desirable accuracy of the final model.

Acknowledgments

I would like to express my gratitude to the admissions staff at Linköping University for giving me a chance to study a master's program in Sweden. I am thankful to my university supervisor Filip Ekström Kelvinius for very insightful discussions in the course of this thesis. Furthermore, I am grateful to my opponent, Jaskirat Marar, and examiner, Annika Tillander, for their feedbacks that helped me a lot in improving the quality of this thesis. Finally, I want to thank Ericsson's AI Research team in Luleå, in particular, Markus Andersson J for giving me a chance to write this thesis in his team and my supervisor, Agustín Valencia, for a great support and assistance in the project.

Contents

List of Figures

List of Tables

Acronyms

AAU active antenna unit

ACF autocorrelation function

AI artificial intelligence

BS base station

CNN convolutional neural network

IEC International Electrotechnical Commission

IID independent and identically distributed

HFL horizontal federated learning

LSTM long short-term memory

LTE long-term evolution

MAE mean absolute error

ML machine learning

MLP multilayer perceptron

MSE mean squared error

NCR number of communication rounds

NLE number of local training epochs

NSC number of sampled clients

NR new radio

PSU power supply unit

RAN radio access network

ReLU rectified linear unit

RBS radio base station

RMSE root mean squared error

RNN recurrent neural network

TCN temporal convolutional network

VFL vertical federated learning

1 Introduction

As the spread of 5G cellular networks continues, energy consumption remains one of the main challenges in the mobile telecommunications industry. The recent study from Ericsson Research [15] demonstrates that cellular networks are responsible for about 0.2% of the global carbon emissions, and about 0.6% of global electricity use. According to the same study, it is expected that the demand on cellular networks will only continue to grow. It is also argued [15] that the rising number of subscriptions and traffic growth has a smaller contribution to increased energy consumption. The deployment of new frequency bands and network equipment reflected in increasing population coverage are listed among the major contributors.

In the cellular networks developed by the long-term evolution (LTE) standards, base stations (BSs) dominated the energy consumption of the radio access network (RAN) comprising around 80 percent of the RAN electricity use [2]. While the development of 5G New Radio (NR) standard was expected to lower energy consumption of BSs, there are challenges that are yet to be addressed.

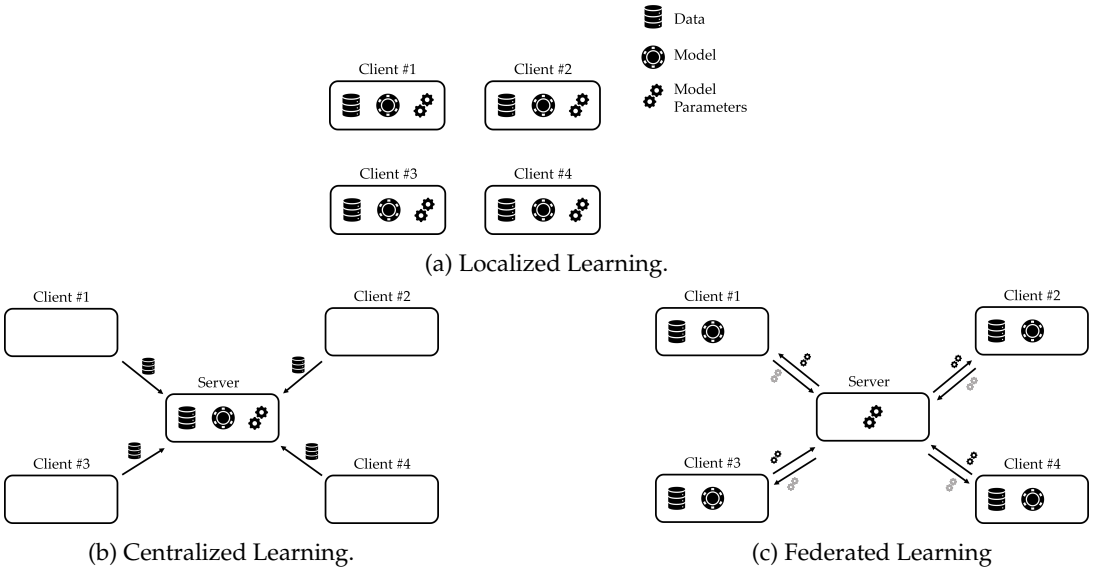
Different approaches can be deployed to navigate the mobile telecommunications industry in an energy efficient way. Among the proposed ones such as effective modernization of existing networks and holistic view to meeting business and sustainability targets [15], there is also an opportunity to leverage the power of artificial intelligence (AI) and machine learning (ML) for energy optimization.

As defined by the International Electrotechnical Commission (IEC) [10] , smart grid is an electrical grid built on standardized software and hardware that ensures seamless integration and interoperability of energy resources using different automated functions. One of these functions includes the ability to self-heal by using real-time information from sensors detecting and responding to system failures. This allows the grid to automatically avoid or mitigate the impact of power outages, power quality problems and service disruptions. In [13], Eleftheriadis et al. discuss the benefits of combining AI and 5G service-based architecture for a better performance in smart grid operation and on-demand balancing services for utilities. Increased automation allows mobile networks to collect data from access nodes and provide real-time performance measurements that enhance efficiency of energy operations. It is also important to consider communication and latency issues when integrating AI/ML functionalities. Eleftheriadis et al. argue that local learning of data and mitigation of mobile network assets in clusters that rely on centralized learning should be prioritized to ensure low latency towards power grids. This can allow to build communication efficient

power forecasting models for radio base stations (RBSs) that help to generate power grid patterns contributing to more reliable power management of smart grids. An example use case of power consumption forecasting models is demonstrated by Valencia et al. in [55]. It is demonstrated how predictions made by a power forecasting model allow to create a better maintenance schedule and preserve operational continuity. In this thesis work, a power forecasting model for RBSs is also under the consideration. However, the focus is on the impact of different learning scenarios on the generalization performance of the forecasting model.

1.1 Motivation

To build a power consumption forecasting model for several RBSs, the logged data from the same RBSs needs to be collected. In the conventional way, the data from many different RBSs will be stored in a cloud-based system, and a corresponding statistical or machine learning model will be built using all available data. This is a *centralized learning* approach illustrated in Figure 1.1b. Among several disadvantages of this approach can be an increasing communication overhead when large amounts of data are transferred between client nodes (RBSs) and a central node (cloud-based system). In addition to that, privacy issues may arise if sensitive data from many different sources is stored in a single space that is not controlled by the data owners. One of the alternatives to the centralized learning is *localized learning*. As illustrated in Figure 1.1a, in localized learning, the local model for each client node is trained using only the data available to it. This approach can be considered a fully private setting where the data is kept at its source. It is expected to reduce communication overheads and handle privacy issues that arise in centralized learning. However, the local models trained in that way may have unique characteristics that do not generalize well to other client nodes. Furthermore, client nodes need sufficient computational resources to train predictive models. These resources may be limited on some client nodes depending on the hardware availability. Another alternative to centralized and localized learning scenarios is *federated learning*.



(a) Localized Learning.

(b) Centralized Learning.

(c) Federated Learning

Figure 1.1: Different learning scenarios can be used for building a power consumption forecasting model.

McMahan et al. [36] describe federated learning as a technique that *"allows users to collectively reap the benefits of shared models from rich data, without the need to centrally store it"*. In federated learning, as illustrated in Figure 1.1c, each local node (client) performs a local training on its private data and shares only the model parameters with the central node (server).

The server aggregates local model parameters using an aggregation function and updates its global model parameters. The updated parameters are then shared with all participating clients. The key advantage is that only local model parameters and not the data itself are shared which allows to preserve privacy and sensitivity. Consequently, communication overhead decreases since only the model parameters are shared with the server. Moreover, the model trained in federated learning can generalize better than the locally trained models [6] if datasets from different clients do not suffer from a statistical heterogeneity [36] [62] [27] as described in section 2.2.2.

When looking into the studies of federated learning in mobile telecommunications industry, there is an ongoing research with the primary focus on radio traffic prediction. Nan et al. [38] utilize a clustering methodology to place BSs with large commonality into similar regions and run a suitable federated aggregation algorithm for several regional models. Phyu et al. [44] propose a federated learning framework that allows to predict traffic at each of BSs that are distributed across different 4G network slices. Here, the network slicing assumes that resources of a given BS are allocated on the application basis. Thus, a performance and resource handling of a given base station are monitored under four (as the number of applications) distinct processes. Finally, a very extensive study is implemented in [42] to compare different deep learning architectures and federated aggregation functions for traffic forecasting of BSs.

As it can be seen the majority of the mentioned studies focus on the traffic forecasting. Valencia et al. [55] in their study demonstrate that traffic metrics are highly correlated with power consumption of RBSs, however, the topic of federated learning for power consumption forecasting in RBSs is yet to be explored. Thus, the focus of this thesis work is on the evaluation of federated learning for power consumption forecasting in RBSs.

1.2 Objective

The objective of this thesis work is to explore the application of federated learning for power consumption forecasting in RBSs. In particular, this study focuses on a large number of RBSs and whether federated learning can help to discover potential correlations between RBSs that can improve generalization performance. To sum up, the following research questions are answered in this study:

- Which modelling scenario (centralized or localized) results in a better generalization performance for power consumption forecasting in RBSs?
- Is it possible to implement a power consumption forecasting model for RBSs in the federated learning scenario that has a better or a comparable generalization performance to the models trained in centralized and localized scenarios?

2 Theory

This chapter provides a theoretical background for the parametric model implemented in this thesis work and federated learning, a machine learning paradigm in which the described parametric model is trained. In the first part of the chapter, a neural network model, its training mechanism and architectures suitable for time series forecasting are discussed. The second part of the chapter focuses on federated learning, in particular, federated training algorithm, and one of the major challenges of federated model training, non-iid data, is also discussed.

2.1 Neural Networks

2.1.1 Neural Network Model

Neural network is a nonlinear parametric model. It means that this model can capture a nonlinear relationship between input and output by learning a parameterizable function of the following form

$$\hat{y} = f_\theta(X), \tag{2.1}$$

where θ are a set of learnable model parameters, X is an input feature vector and $\hat{y}$ is a prediction made by the model.

To understand a neural network, one can start with a simple linear regression model. A prediction $\hat{y}$ made by linear regression is written as (p.39, [33])

$$\hat{y} = w_0 x_0 + w_1 x_1 + ... + w_p x_p + b, \tag{2.2}$$

where $W = [w_0, w_1, ..., w_p]$ is denoted as a vector of model parameters, b as an offset term and $X = [x_0, x_1, ..., x_p]^T$ as a p - dimensional input feature vector. Nonlinear relationship between X and $\hat{y}$ can be obtained by placing the input vector X into the *activation function* g and written in the following form

$$\hat{y} = g(w_0 x_0 + w_1 x_1 + ... + w_p x_p + b), \tag{2.3}$$

or in the vectorized form as

$$\hat{y} = g(WX + b). \tag{2.4}$$

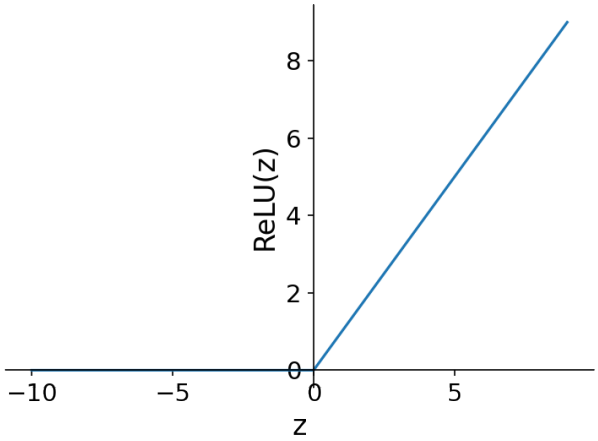


Figure 2.1: ReLU activation function.

The default activation function recommended (p.175, [20]) for use with feedforward neural networks is the rectified linear activation function also known as ReLU. Nonlinear transformation is obtained when applying this function to the linear form described in equation 2.2. As it can be seen from Figure 2.1, it is a piecewise linear function with two linear pieces. Since ReLU is nearly linear, it preserves many of the properties that make linear models easy to optimize with gradient based methods. ReLU activation function of an arbitrary variable z is generally written in the following form

$$g(z) = max(0, z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} \tag{2.5}$$

In equation 2.4, X belongs to the *input* layer and $\hat{y}$ belongs to the *output* layer. Adding a *hidden layer* between input and output layers can help a neural network to capture more complex relationships between input and output. A hidden layer usually consists of *hidden units*. In a neural network with a single hidden layer, the task of each hidden unit is to receive the input from the input layer, apply a nonlinear activation function and produce the output for the output layer. Mathematically, this operation has the form similar to equation 2.3, also known as a generalized linear regression model (p.58–59, [33])

$$q_k = g(w_{k0}x_0 + w_{k1}x_1 + ... + w_{kp}x_p + b_k) \text{ for } k = 1, ..., U, \tag{2.6}$$

where q_k is denoted as k^{th} hidden unit of a hidden layer with U hidden units. Each of the hidden units has a functional form similar to equation 2.6, however, each of those functions is parameterized by a set of different parameters $W_k = [w_{k0}, w_{k1}, ..., w_{kp}]$. Given that there are U hidden units $\{q_k\}_{k=0}^{U}$ in a single hidden layer, they act as a vector of new inputs $q = [q_0, q_1, ..., q_U]$ into the output layer. Mathematically, this is written in the following form

$$\hat{y} = w_0q_0 + w_1q_1 + ... + w_Uq_U + b. \tag{2.7}$$

Equation 2.7 represents a more flexible neural network model with a single hidden layer that consists of U hidden units or U generalized linear regression models. Adding more hidden

layers $l \in \{1, ..., L\}$ increases model flexibility, and can be written in the following form

$$\begin{aligned} q^{(1)} &= g(W^{(1)}X + b^{(1)}), \\ q^{(2)} &= g(W^{(2)}q^{(1)} + b^{(2)}), \\ &... \\ q^{(L-1)} &= g(W^{(L-1)}q^{(L-2)} + b^{(L-1)}), \\ \hat{y} &= W^{(L)}q^{(L-1)} + b^{(L)}, \end{aligned} \tag{2.8}$$

where every l^{th} layer consists of U_l hidden units $q^{(l)} = [q_1^{(l)}, ..., q_{U_l}^{(l)}]$. This also means that the number of hidden units can vary across hidden layers. Finally, each layer l is parameterized with the weight matrix $W^{(l)}$ and the offset vector $b^{(l)}$. The offset terms in the offset vector behave as thresholds for activation of hidden units in a neural network. These are a set of learnable parameters that make a neural network a nonlinear parametric model.

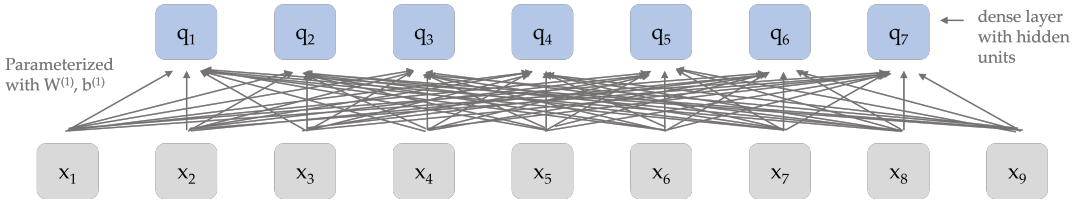


Figure 2.2: An example of a dense hidden layer with 7 hidden units and input sequence length = 9.

The type of a neural network model described in this section is also known as a *multilayer perceptron* (MLP) [48]. MLP architecture typically consists of an input layer, hidden layers and an output layer. Hidden layers in MLP are also called *dense* layers owing to the density of connections between hidden units. Figure 2.2 illustrates an example of a dense hidden layer. For a simplicity of demonstration, connections to the offset vector $b^{(1)}$ are not illustrated.

2.1.2 Neural Network Training

In the section 2.1.1, it was demonstrated that a neural network is a parametric model with the model parameters consisting of the weight matrix W and the offset vector b. In the following sections, all model parameters are denoted as θ and written in the following form

$$\theta = [W^{(1)}\ b^{(1)}\ ...\ W^{(L)}\ b^{(L)}]. \tag{2.9}$$

To learn θ, the optimization problem of the following form must be solved (p.141 [33])

$$\hat{\theta} = \arg\min_{\theta} J(\theta) \quad \text{where} \quad J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i, \theta) \tag{2.10}$$

where $J(\theta)$ is a cost function, and $L(x_i, y_i, \theta)$ is a parameterized loss over a given data point (x_i, y_i).

2.1.2.1 Maximum Likelihood Estimation and Loss Function

A loss function selection is a design choice and different loss functions result into different solutions $\hat{\theta}$. It can also be selected from a statistical perspective. The method of maximum likelihood (p.476, [57]) helps to provide a formal connection between a loss function and probabilistic assumptions on a random noise of data (p.43, [33]). Assume a dataset $D =$

$\{x_i, y_i\}_{i=0}^n$ where for a single data point (x_i, y_i) the relationship between input x_i and output y_i is modelled as

$$y_i = f_\theta(x_i) + \epsilon, \tag{2.11}$$

where ϵ is a random noise drawn from some known distribution. The goal of the maximum likelihood estimation is to identify the parameters θ that maximize the likelihood of observing y_i given x_i. This is formally written as

$$\hat{\theta} = \arg\max_\theta \ p(y|X;\theta), \tag{2.12}$$

where $y = \{y_i\}_{i=0}^n$ are all outputs, $X = \{x_i\}_{i=0}^n$ are corresponding inputs and $p(y|X;\theta)$ is a probability density function showing how likely it is to observe y given X and parameters θ. A common assumption is that the noise terms ϵ are independent and normally distributed

$$\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2). \tag{2.13}$$

This implies that observed data points are independent. Thus, the likelihood $p(y|X;\theta)$ is factorized as

$$p(y|X;\theta) = \prod_{i=1}^n p(y_i|x_i,\theta). \tag{2.14}$$

Working with a product of conditional probabilities can be inconvenient due to computational reasons such as numerical underflow (p.132 [20]). Thus, the product of conditional probabilities is transformed into the sum of logarithms of conditional probabilities, and the following form is obtained

$$\log\ p(y|X;\theta) = \sum_{i=1}^n \log\ p(y_i|x_i,\theta). \tag{2.15}$$

Note that since the logarithm is monotonically increasing function, maximizing the likelihood from equation 2.14 will yield the same solution $\hat{\theta}$ as maximizing the log-likelihood from equation 2.15. Assuming that ϵ is normally distributed, the probability density function of y_i given x_i is written as

$$p(y_i|x_i,\theta) = \frac{1}{\sqrt{2\pi\sigma_\epsilon^2}}\ \exp(-\frac{(y_i - f_\theta(x_i))^2}{2\sigma_\epsilon^2}). \tag{2.16}$$

After inserting equation 2.16 into equation 2.15, the log-likelihood is written in the following form

$$\log\ p(y|X;\theta) = -\frac{n}{2}\, log(\pi\sigma_\epsilon^2) - \frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^n (y_i - f_\theta(x_i))^2. \tag{2.17}$$

Since it is conventional to define the optimization problems as minimization problems, the likelihood maximization problem is transformed into the negative log-likelihood minimization problem

$$\hat{\theta} = \arg\min_\theta \ -\log p(y|X;\theta). \tag{2.18}$$

When removing the terms and the factors independent of θ from equation 2.17 and placing this equation into equation 2.18, the cost function for estimating the optimal parameters $\hat{\theta}$ is obtained

$$\hat{\theta} = \arg\min_\theta \ \sum_{i=1}^n (y_i - f_\theta(x_i))^2 \tag{2.19}$$

Equation 2.19 represents a cost function with a squared error loss $(y_i - f_\theta(x_i))^2$ over a data point (x_i, y_i) and when multiplied by the factor $\frac{1}{n}$, the least squares loss also known as *mean squared error* (MSE) is obtained which is a commonly used loss function in linear regression.

The optimization problem defined in equation 2.10 is now updated by equation 2.19

$$\hat{\theta} = \arg\min_\theta J(\theta) \ \text{ where } \ J(\theta) = \frac{1}{n}\sum_{i=1}^n (y_i - f_\theta(x_i))^2 \tag{2.20}$$

2.1.2.2 Optimization with Gradient Descent

The optimization problem defined in equation 2.20 cannot be solved in the closed form for a general neural network, thus, a numerical optimization approach should be applied. In neural networks, gradient based optimization algorithms are usually applied, and the optimization problem is solved in an iterative manner. The general procedure is the following:

1. Initialize model parameters θ_t at timestep $t = 0$.
2. After each round of iteration t update parameters $\theta_{t+1} = \theta_t - \gamma \nabla_\theta J(\theta_t)$.
3. Terminate optimization when a given convergence criterion is fulfilled and take θ_t, parameters obtained at iteration t, as $\hat{\theta}$.

Gradient descent is an optimization algorithm for finding a local minimum of a given differentiable function. The term *gradient* describes the direction of steepest ascent and is represented by the vector of partial derivatives with respect to the given cost function $J(\theta)$. The vector of derivatives with respect to parameters is denoted as $\nabla_\theta J(\theta)$, and negating this vector $-\nabla_\theta J(\theta)$ allows it to describe the direction of steepest descent in which the value of the cost function $J(\theta)$ decreases. This means that taking a small step in the direction of the negative gradient reduces the value of the cost function and is written as the following inequality (p.117, [33])

$$J(\theta - \gamma \nabla_\theta J(\theta)) \leqslant J(\theta), \tag{2.21}$$

where γ is a learning rate parameter that determines the size of the gradient step at each iteration. The selection of the learning rate γ can also be a part of the optimization problem. Line search is a univariate optimization problem that can be used to find the optimal value of the learning rate. However, conducting line search at each iteration t can be computationally expensive. Alternatively, the choice of the learning rate can be also left to the user(p.55, [29]). In this study, the decision is also to leave the learning rate as a user defined hyperparameter. At each round of iteration t, parameters θ_t are updated so that $J(\theta_{t+1}) \leqslant J(\theta_t)$

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta J(\theta_t). \tag{2.22}$$

In neural networks, the cost functions usually belong to the class of *non-convex* objective cost functions. Non-convex functions are functions that have many local minima (p.284, [20]). The gradient descent algorithm cannot always find the global minimum in non-convex problems; instead, it arrives at one of the local minima or saddle points. In non-convex problems, the choice of an appropriate learning rate and initialization of model parameters become vital. In addition, it can be required to run the algorithm several times to arrive at the local minimum that is closest to the global minimum.

When computing the gradient of the cost function described in equation 2.20, the sum of losses across all data points must be calculated

$$\nabla_\theta J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta (y_i - f_\theta(x_i))^2. \tag{2.23}$$

For large datasets, this can result in significant computational costs (p. 124, [33]). An alternative to that can be taking a random sample of n_b data points, also known as *mini-batches*, and computing the gradient with respect to the sum of losses across a given mini-batch. Then, equation 2.23 can be rewritten in the following form

$$\nabla_\theta J(\theta) = \frac{1}{n_b} \sum_{i=1}^{n_b} \nabla_\theta (y_i - f_\theta(x_i))^2. \tag{2.24}$$

When working with mini-batches rather than whole training datasets, it is important to ensure that a given mini-batch is representative of the whole training dataset. One way of ensuring that can be randomly shuffling the training dataset and dividing it into mini-batches

in an ordered manner (p.125, [33]). This type of gradient descent algorithm working with mini-batches is called *stochastic gradient descent*.

The problem with stochastic gradient descent is that it does not converge with a constant learning rate γ (p.126, [33]). Each gradient calculated from a given mini-batch is an estimate of a true gradient, and this estimate can suffer from a random noise that comes with random sampling. Thus, the gradient descent will continue to wander randomly (p.126, [33]) instead of converging to the local minima. In recent years, several enhancements of gradient descent [12], [28], [22] were proposed to solve the problem of the fixed learning rate with the automatic adaptation.

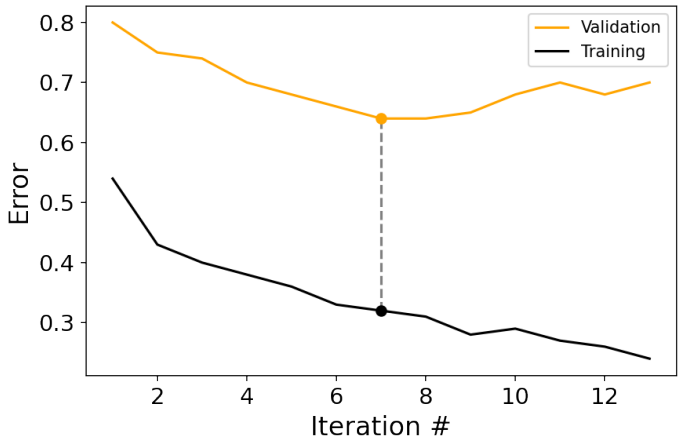


Figure 2.3: Training and validation errors during training of a neural network. A dashed line indicates the smallest validation error and iteration number at which model parameters are saved.

2.1.2.3 Backpropagation

At each iteration of the gradient descent algorithm, it is required to calculate the gradient, the vector of derivatives of the cost function with respect to all the parameters. *Backpropagation* [49] is an algorithm utilized during the training of neural networks that allows to effectively calculate the gradient. It mainly consists of two steps, the *forward* propagation and *backward* propagation. In the forward propagation, the values of all hidden units in each of the hidden layers and the value of cost function are calculated. In the backward propagation, the gradients with respect to hidden units of all hidden layers and model parameters are calculated. The derivatives of the cost function with respect to hidden units and model parameters are calculated in the recursive manner, using the chain rule of calculus (p.116, [1]).

Finally, the training process of a neural network model is summarized in the following way:

1. Initiate parameters θ_0.
2. Implement forward propagation computing q, $\hat{y}$ and $J(\theta)$.
3. Implement backward propagation computing gradient $\nabla_\theta J(\theta)$.
4. Update parameters θ.
5. Iterate steps 2 - 4 until convergence.

In step 5, it is mentioned that steps 2 - 4 are iterated until a convergence criteria is fulfilled. In this study, *early stopping* strategy (p.246, [20]) is used to ensure model convergence. When training a neural network, it can be observed that a training error decreases over time while a validation error reaches its minimum and then starts to increase again. A typical example of such a behavior is demonstrated on Figure 2.3. Early stopping allows to terminate a training process if a validation error does not decrease for a successive number of iterations and save model parameters that correspond to the smallest validation error as optimal model parameters.

2.1.3 Convolutional Neural Network

Convolutional neural network (CNN) is a type of a neural network with a convolutional hidden layer [17] instead of or in addition to dense layers. CNN is a model suitable for working with data that follows a grid-like topology (p.330, [20]). Time series and image data are typical examples of data with a grid-like structure. Time series can thought of as a one-dimensional grid with regular time intervals while an image can be thought of as a two-dimensional grid of pixels.

A convolutional layer of CNN relies on the mathematical operation called *convolution* (p.330, [20]). Convolution operation allows to leverage a spatial structure of the data inherent to time series and image data. In one-dimensional data such as time series, the concept of spatial structure can be translated to the adjacency of points in one dimensional continuous coordinate. To put it differently, it is logical to assume that the observations closer in time are more related to each other that observations further apart. Mathematically, the convolution operation can be written in the following form

$$z_i = \sum_{j=1}^{k} x_{i+j-1} a_j, \tag{2.25}$$

where $a = [a_1, ..., a_k]$ is a vector of weights of the convolutional filter, x_{i+j-1} is an element of the input sequence and z_i is the i^{th} value of some hidden unit vector before applying any nonlinear transformation. A vector of weights a is also a set of parameters and is learned during the training process. Equation 2.25 essentially means that a filter of fixed size k slides over a raw input sequence of data points and generates a different representation of an input. The result is passed further through a nonlinear activation function g (such as ReLU described in chapter 2.1.1) to model nonlinear relations between input and output. There are three

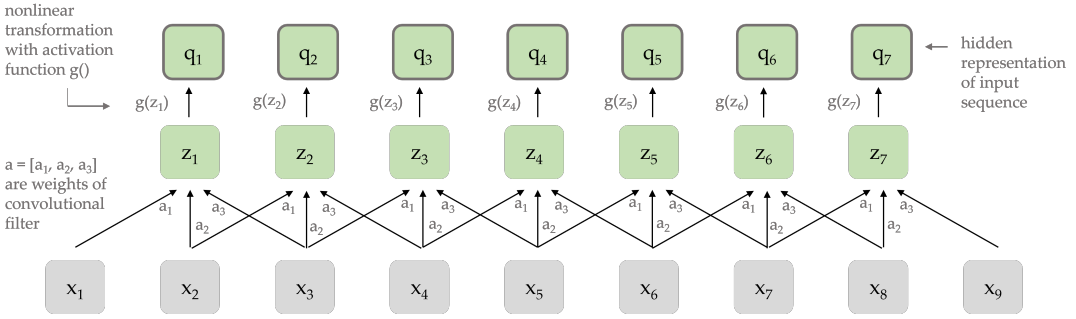


Figure 2.4: An example of a convolutional hidden layer with filter size = 3 and input sequence length = 9.

concepts that differentiate a convolutional layer from a dense layer (p.335, [20]):

1. **Parameter sharing**. In a convolutional layer, each subset of X is multiplied by the same filter weights a and passed through an activation function g to generate a vector of hidden unit values q. This is different from a dense layer where the whole input sequence

X would be multiplied by a different weight matrix W_{kl} (k^{th} hidden unit of l^{th} layer) to obtain a value of each hidden unit in a dense layer as described in chapter 2.1.2. As a result, a smaller amount of parameters is learned during the training of a convolutional layer making it computationally more efficient than a dense layer.

2. **Sparse interactions**. As it can be seen from Figure 2.4, every hidden unit in a convolutional layer depends on the small region of input sequence. In contrast, each hidden unit of a dense layer depends on the whole input sequence. This is also referred to as *sparcity*. Sparcity allows a convolutional filter to learn different local patterns in data diminishing an impact of a noise and leveraging a spatial structure of the data.

3. **Equivariance to translation**. The property of equivariance to translation results from sparse interactions and parameter sharing. In the context of time series, equivariance means that if an event is moved later in time in the input, a similar representation of it will appear in the output, just later in time (p.339, [20]).

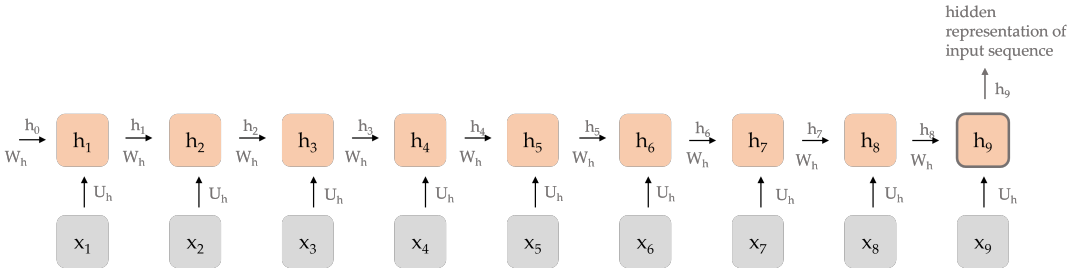


Figure 2.5: An example of a recurrent hidden layer with input sequence length = 9.

2.1.4 Recurrent Neural Network

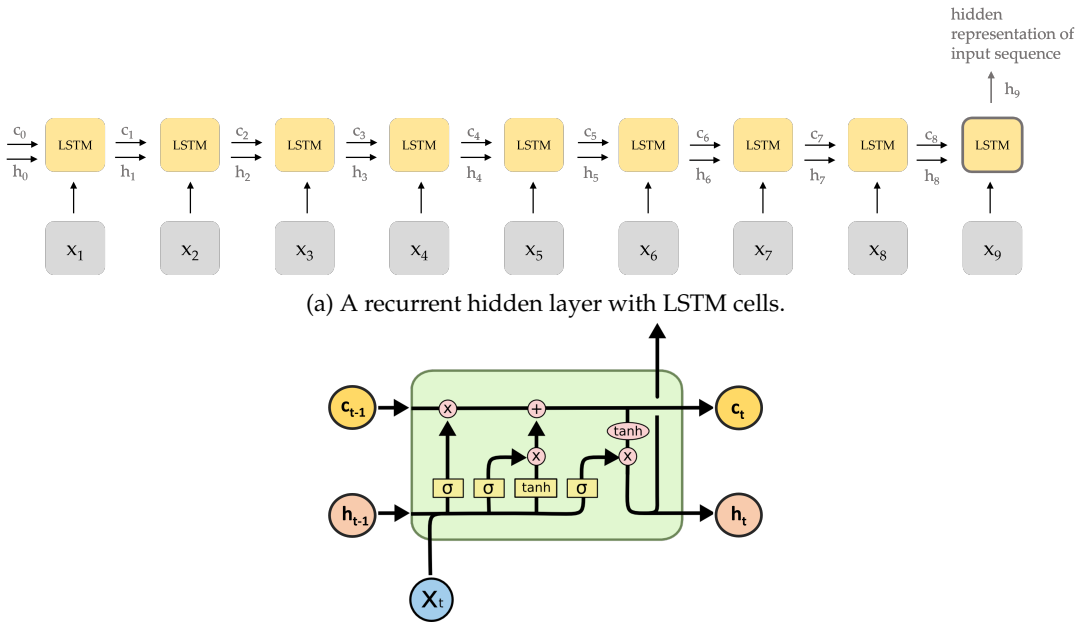
Recurrent neural network (RNN) is a type of a neural network with a recurrent hidden layer. RNN is a model suitable for working with sequential data such as time series or text. Thus, the input X needs to be represented as a sequence of T values and denoted as $X = \{x_t\}_{t=1}^{T}$ where each x_t represents an input value at the t^{th} timestep and depends on the values from previous $t-1$ timesteps. Figure 2.5 illustrates Elman network [14] that is one of the widely used types of RNN. Mathematically, it is represented in the following form

$$\begin{aligned} h_1 &= g(W_h x_1 + U_h h_0 + b_h), \\ h_2 &= g(W_h x_2 + U_h h_1 + b_h), \\ &\dots \\ h_T &= g(W_h x_T + U_h h_{T-1} + b_h), \\ \hat{y} &= g(W_y h_T + b_y). \end{aligned} \tag{2.26}$$

Equation 2.26 demonstrates that for any timestep t, x_t is taken as an input element, and h_t, a hidden activation vector with values of hidden units also known as a *hidden state*, is calculated. When x_{t+1}, the next element of the sequence is processed, h_{t+1} is calculated using the input x_{t+1} and h_t that contains the values from the hidden states of the previous timesteps. As a result, each h_t includes information from the previous hidden states h_{t-1} to h_0. This dependence on the hidden states from the previous timesteps allows RNN to learn the sequential relationship between the elements at different timesteps. From equation 2.26, it can also be observed that the model parameters W_h and U_h are shared across all timesteps. Similar to CNN, this is a parameter sharing property that makes it possible to apply RNN to sequences of different lengths and, more importantly, learn similar patterns at different timesteps (p.373, [20]).

2.1.4.1 Long Short-Term Memory Cell

One of the problems with RNN is that it is unable to capture long-term dependencies due to the problem of vanishing gradients [41]. The gradient of RNN is calculated using backpropagation through time [59], and when the number of layers (or a sequence length) increases, gradient updates become exponentially smaller what makes it difficult for a neural network to learn long-term dependencies in the sequence. A proposed improvement called *long short-term memory* (LSTM) cell [23] has shown useful in solving the problem of vanishing gradients in RNN.



(a) A recurrent hidden layer with LSTM cells.

(b) A closer look at LSTM cell [40].

Figure 2.6: An example of a recurrent hidden layer with LSTM cells.

An LSTM cell [23] can be viewed as an enhancement to the architecture of RNN. Figure 2.6b demonstrates the architecture of an LSTM cell. At a given timstep t, there are two outputs: a hidden state h_t and a cell state c_t both of which contain information from the current and previous timesteps in the sequence. An LSTM cell relies on the *gating mechanism* to decide how to update h_{t-1} and c_{t-1} values from a previous timestep before outputting h_t and c_t to the next timestep in the sequence. Thus, an LSTM cell is designed with three gates:

- *Forget gate* uses $\sigma(\cdot)$ *sigmoid* function of h_{t-1} and x_t to decide which information to forget from c_{t-1}.
- *Update gate* uses $\sigma(\cdot)$ sigmoid function of h_{t-1} and x_t to decide which values to update in c_{t-1} and $tanh(\cdot)$ function of h_{t-1} and x_t to calculate new candidate values $\hat{c}_t$.
- *Output gate* uses $tanh(\cdot)$ to put values of c_t between -1 and 1 and $\sigma(\cdot)$ function of h_{t-1} and x_t to decide which information from c_t to output as h_t

These operations are represented in the form of mathematical expressions as

$$\begin{aligned} \Gamma_f &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\ \Gamma_u &= \sigma(W_u[h_{t-1}, x_t] + b_u) \\ \hat{c}_t &= tanh(W_c[h_{t-1}, x_t] + b_c) \\ c_t &= \Gamma_u \cdot \hat{c}_t + \Gamma_f \cdot c_{t-1} \\ \Gamma_o &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= \Gamma_o \cdot tanh\, c_t \end{aligned} \tag{2.27}$$

where $\hat{c}_t$ denotes candidate values for updating cell state c_{t-1}, and Γ_f, Γ_u, Γ_o are forget, update and output gates respectively. All equations are parameterized with corresponding parameters W and b that are learned during the training process.

The key part of the above equations that allows to solve vanishing gradients problem is derivative $\frac{\partial c_t}{\partial c_{t-1}}$. When solving it, it can be seen that there is no exponential weight accumulation (that is present in RNN) meaning that there is at least one way where the gradient does not vanish.

2.2 Federated Learning

Federated learning is a machine learning paradigm that allows to build a single model trained on several datasets that are distributed across different places. It was introduced by McMahan et al. [36] as a way of leveraging a large amount of data stored in different mobile devices without the need to store this data in a single place. As it is with all machine learning models, in federated learning, there is also an optimization problem that needs to be solved. In section 2.1.2, the optimization problem for training a neural network model was written in the following form

$$\hat{\theta} = \arg\min_{\theta} J(\theta) \quad \text{where} \quad J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2 \tag{2.28}$$

To solve the optimization problem in federated learning, equation 2.28 is rewritten in the following form

$$\hat{\theta} = \arg\min_{\theta} J(\theta) \quad \text{where} \quad J(\theta) = \sum_{k=1}^{K}\frac{n_k}{n}J_k(\theta) \quad \text{and} \quad J_k = \frac{1}{n_k}\sum_{i\in D_k}(y_i - f_\theta(x_i))^2 \tag{2.29}$$

where D_k is the set of indexes of data points that belongs to the k^{th} data source and $n_k = |D_k|$. In equation 2.29, the optimization problem is still the same as in equation 2.28, however, the datasets are distributed across k different data sources.

2.2.1 Federated Training Process

In the federated training process, there are usually two participating entities: *clients* and *server*. Clients can be viewed as separate data sources. Model is trained locally at each of the clients (a local training) using global parameters received from the server. After termination of the local training, each of the clients shares its model parameters with the server. The server receives local model parameters from the clients, performs aggregation of those parameters, updates global model parameters with aggregated ones and sends them back to the clients. This process continues iteratively until some convergence criteria is met.

At the t^{th} communication round, the federated training process can be described in the following way:

1. The server randomly samples S_t, a fixed number of clients.
2. The server sends global model parameters W_t to S_t sampled clients.
3. Each of the sampled Clients $C^k \in S_t$ starts a local training on its dataset D_k using received global model parameters W_t.
4. After a local training, each of the sampled clients S_t sends their own parameters W_t^k to the server.
5. The server aggregates model parameters $\sum_{k \in S_t} \frac{|D_k|}{n} W_t^k$ and sends them as updated global parameters W_{t+1} to a new set of sampled Clients for the next communication round.

The above steps are formally represented in Algorithm 1 that was originally proposed by McMahan et al. [36]. The aggregation principle described in step 5 is also known as *Federated Averaging* (FedAvg) algorithm [36].

Aggregation of the local model parameters is one of the most important steps of the federated training process. FedAvg performs aggregation by computing the weighted average $\sum_{k \in S_t} \frac{|D_k|}{n} W_t^k$ of the local model parameters received from the clients. As it was demonstrated by McMahan et al., a naive parameter averaging achieved significantly lower loss on a given training dataset than the best model achieved by independently training on one of the smaller datasets. In recent years, alternative aggregation functions [32] [47] [58] emerged to handle system and statistical data heterogeneities. While some of them proven to be effective under different settings, a statistical heterogeneity, commonly appearing in the form of non-iid data, still remains one of the major challenges [27] [62] in FL.

Algorithm 1: Federated Averaging (FedAvg) Algorithm

Input : Local datasets D_k, Strategy: C clients, T communication rounds, E local training epochs, aggregation function
Output: W_T global model parameters after the T^{th} communication round

Server:
  Initialize W_0
  for each communication round $t = 1, 2, ..., T$ **do**
    Sample S_t, a set of clients, out of C clients
    $n := \sum_{k \in S_t} |D_k|$
    for each client $k \in S_t$ **in parallel do**
      Send global model parameters W_t to client C^k
      $W_t^k := \textbf{Client}(k, W_t)$
    endfor
    $W_{t+1} = \sum_{k \in S_t} \frac{|D_k|}{n} W_t^k$
  endfor
  return W_T
Client:
  $W_t^k = W_t$
  for each local epoch $e = 1, 2, ..., E$ **do**
    for each batch $b \in D_k$ **do**
      $W_t^k := W_t^k - \gamma \nabla J(W_t^k; b)$
    endfor
  endfor
  return W_t^k

Yang et al. [60] differentiate two types of federated learning: horizontal and vertical. In horizontal federated learning (HFL), clients have different data points but share the same

feature space. HFL is formally represented in the following way

$$X_i = X_j, I_i \neq I_j, \ for \ \forall D_i, D_j \ and \ i \neq j, \tag{2.30}$$

where X denotes a feature space, D_i and D_j denote data held by clients i and j, I denotes sample id space. Vertical federated learning (VFL) refers to a scenario where clients share the same data points but with different feature spaces. VFL is formally represented in the following way

$$X_i \neq X_j, I_i = I_j, \ for \ \forall D_i, D_j, \ and \ i \neq j, \tag{2.31}$$

In this thesis work, HFL is selected as a modelling scenario because the same feature space (see section 4.1) is available across all RBSs, however, the time series describing those features are different meaning that each RBS has different data points.

2.2.2 Non-IID Data in Federated Learning

Kairouz et al. [27] provide a taxonomy of non-iid data regimes that are commonly encountered when datasets are distributed across several clients. The taxonomy is demonstrated for datasets designed as a supervised machine learning task with features x and target variable y. They mention two levels of statistical sampling that are involved in federated learning. The first one is related to sampling a set of clients from all available clients $k \sim C$. For the t^{th} communication round, this set of sampled clients was referred to as S_t in the previous section. The second round of sampling is related to drawing an example (x, y) from a distribution of a given client $P_i(x, y)$. In federated learning, the typical cases of non-iid data are when distributions $P_i(x, y)$ and $P_j(x, y)$ are different between clients i and j present in a set of sampled clients S_t.

$P(x, y)$ can be rewritten using conditional and marginal probabilities (p.52, [57])

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x) \tag{2.32}$$

The following are listed by Kairouz et al. [27] as the common ways in which data deviates from being identically distributed, that is $P_i(x, y) \neq P_j(x, y)$ for different clients i and j:

- *Feature distribution skew (covariate shift).* Marginal distributions $P(x)$ may vary across different Clients i and j, that is $P_i(x) \neq P_j(x)$.
- *Target variable distribution skew (prior probability shift).* Marginal distributions $P(y)$ may vary across different clients i and j, that is $P_i(y) \neq P_j(y)$.
- *Different features, same target variable (concept drift).* Conditional distributions $P(x|y)$ may vary across different clients i and j, that is $P_i(x|y) \neq P_j(x|y)$.
- *Different target variables, same features (concept shift).* Conditional distributions $P(y|x)$ may vary across different clients i and j, that is $P_i(y|x) \neq P_j(y|x)$.
- *Quantity skew.* Different clients can hold different amounts of data, that is $|D_i| \neq |D_j|$

In addition to above-mentioned non-iid data regimes, Zhu et al. mention [62] *temporal skew* as a type of non-iid data regime. Temporal skew is usually encountered when working with spatio-temporal and time series data. For time series data, this implies that $P_i(x, y|t)$ (where t is time index) on the Client i is changing over time. In [42], Vasileios et al. demonstrate how the distribution of daily traffic changes over time following different patterns. Zhu et al. also mention that data collected by clients during different periods of time can also result in temporal skew.

3 Related Work

This chapter provides a literature review of studies that were found to be relevant to this thesis work and serve as a reference point. In particular, the studies covering the following topics are reviewed: power consumption models for BSs, federated traffic forecasting models and forecasting models from other domains and neural networks for federated time series forecasting.

3.1 Power Consumption Models for Base Stations

Throughout the years, analytical power models were implemented to track power consumption of BSs [3, 11, 31]. Arnold et al. [3] focus on BS components, in particular, power amplifier and cooling equipment to develop power models for BSs. It is important to note that their model is concurrent which means that both static and dynamic power consumption parameters of the BS are considered. Debaillie et al. [11] also build a power model by modelling hardware components of BSs and demonstrating how to utilize traffic load to design power savings over different sleep modes. Li et al. [31] utilize historical wireless traffic data in addition to the data from hardware components. While the majority of listed studies mainly focus on analytical power models of BSs, one of the recent studies [45] demonstrates that such analytical models can be also coupled with machine learning driven approaches. Specifically, Piovesan et al. [45] reveal how artificial neural networks can be utilized to build a model for 5G active antenna units (AAUs), the most power consuming components of BSs.

In [31] and [37], it is argued that wireless traffic forecasting data can be utilized to build load forecasting model for BSs. Indeed, Valencia et al. [55] build a model that utilizes traffic metrics as explanatory features to forecast energy consumption of power supply units (PSUs) in RBSs. In their study, Prophet, a generalized additive model, is used to forecast power consumption in RBSs. This is one of the few studies where a power model mainly utilizes PSU load utilization as an autoregressive feature and dynamic behavior of radio traffic together with climate related features instead of relying on the power usage of hardware components.

3.2 Federated Learning for Traffic Forecasting

As it was revealed in the previous section 3.1, the recent studies [55] demonstrate that a power consumption model can be built utilizing available dynamic power consumption data

and traffic features as explanatory variables. When looking at the implementations of the federated learning in telecommunication industry, most of the works focus on federated learning for traffic forecasting. Thus, this literature review is followed by going through those studies.

In recent years, federated learning was widely implemented for traffic prediction in mobile telecommunications industry. Vasileios et al. [42] conduct an extensive study on federated learning for 5G BS traffic forecasting. In their study, they test and compare different types of neural network models and aggregation functions used in federated learning. They also try to address the problem of working with non-iid data at different client nodes. Model comparison in their study reveals that LSTM and GRU types of neural networks outperform the rest of the models such as CNN, MLP and RNN in federated and localized learning scenarios. GRU [9] refers to *gated recurrent unit* that is a less parameterized and a simpler type of RNN than LSTM. When comparing different learning scenarios among themselves, Vasileios et al. reveal that the generalization performance of localized learning is followed by centralized learning and federated learning respectively. However, they also show that local fine-tuning, that is training a learned federated model further on the local data without parameter sharing, results in a better generalization performance than localized and centralized approaches. Some of the limitations of their study can be that the models are designed for a single step forecasting horizon, and that the number of BSs under the study is limited to three which is not a statistically representative sample.

Nan et al. [38] propose a regional-union based federated learning (FedRU) for wireless traffic prediction in 5G network. In their study, the central node trains the shared model according to the regional model, and each region trains its own regional model. Eventually, the shared model captures global user and regional data characteristics. Thus, the predictions become affected by the performance of the region division algorithm. The region division algorithm relies on the estimation of correlation between BSs with similar wireless traffic data changes. This also implies that one should already have access to the data from different BSs to make a proper regional division. As in [42], Nan et al. also refer to data heterogeneity as one of the challenges in improving framework performance.

Phyu et al. [44] propose a federated learning framework that addresses traffic forecasting problem in a sliced network architecture. The main goal is to predict traffic at each BS distributed across different network slices. The network slices are shown in the form of BS resources allocated on the application basis. Consequently, each BS operates under four distinct processes corresponding to the number of applications.

3.3 Federated Learning for Load Forecasting in Other Domains and Non-IID Data

Residential energy forecasting is one of the other domains where federated learning is applied. Petrangeli et al. [43] evaluate federated learning for power consumption forecasting in residential communities. They demonstrate that the performance of LSTM networks in centralized learning is better than in the localized learning scenario attributing this to the fact that data partitioning in federated learning has an adverse impact. Furthermore, they study the impact of communication overhead when implementing a federated learning model. It is demonstrated that the communication overhead in the federated scenario is actually higher that in the centralized scenario, and that is supported by the arguments that in centralized scenario all the data is transmitted to the server once, while in the federated scenario, the model parameters are transmitted from each client to the server at each round of parameter update. Nevertheless, it is argued that the overall data transmitted in the network is low, and it can be supported by any wide area network technology.

In contrast with [43], Fekri et al. in their study [16] propose a federated learning approach that can outperform models in localized and centralized scenarios. The model is an LSTM network implemented with two different aggregation functions (FedAvg and FedSGD) and

both are demonstrated to achieve a higher accuracy for a single step hourly and a multi step 24 hour predictions. The results are even more interesting considering that different energy consumption profiles are present in the data which means that it is non-iid.

To tackle the problem of non-iid data across different clients, several clustering approaches are proposed. The idea behind these approaches is to allow several global models on the server side instead of having a single global model. In that way, each of the global models clusters local clients with similar data distributions into a single space. Briggs et al. [6] propose a clustered variant of federated learning for short-term residential load forecasting using an LSTM network. They apply hierarchical clustering after each communication round taking as an input local parameter updates from all clients. Thus, clients producing similar updates are clustered together, and further federated training proceeds separately for each cluster. Results show that this approach has a better performance than federated learning without clustering and can also outperform the centralized learning scenario. However, their approach still has a worse performance when compared with results from the localized learning scenario. Nevertheless, they demonstrate that if federated learning with or without hierarchical clustering is utilized as a pretraining task and followed by further fine-tuning on the local clients, the performance can be improved and result in highly specialized models that outperform even localized learning scenarios.

3.4 Neural Networks for Time Series Forecasting

In the original paper [36], McMahan et al. focus on non-convex cost functions of neural networks when demonstrating federated averaging (FedAvg) algorithm for the first time. Although it is argued that any algorithm with finite-sum parametric learning objective can be utilized, this part of literature review focuses on neural network models which, as it was demonstrated in sections 3.2 and 3.3, were proven to be effective in federated learning for time series forecasting.

One of the largest studies on application of neural networks for federated time series forecasting was done by Vasileios et al. in [42]. They try five different neural network architectures: MLP, RNN, LSTM, GRU and CNN. The models are compared in centralized, localized and federated learning scenarios. Since two evaluation metrics, *mean absolute error* (MAE) and *root mean squared error* (RMSE), are utilized, results can also change accordingly. LSTM and GRU are the top performing models in individual setting. Mixed results are obtained in centralized learning where LSTM and GRU perform the best regarding MAE, GRU and CNN regarding RMSE and MLP and CNN regarding NRMSE(normalized RMSE). The robustness of the models with regards to the random initialization of parameters is also evaluated. LSTM and GRU outperform other models while CNN suffers the most from the random initialization of parameters. Finally, LSTM and GRU are the top performing models in federated settings. Authors also compare communication costs on server and client sides associated with above listed models in federated learning. Among LSTM and GRU that are top performing ones according to evaluation metrics, GRU is more computationally efficient. This is mainly attributed to the fact that GRU is less parameterized than LSTM. The computational efficiency is presented by the minimum required transmissions of local model parameters to generate a global model.

Although outside of federated learning, there are other extensive studies of neural networks for time series forecasting [30] [8]. In [30], Lara-Benítez et al. conduct a comprehensive analysis of seven types of neural network architectures in terms of accuracy and computational efficiency. Results demonstrate that, although an LSTM network outperforms the rest in terms of accuracy of the forecasts, CNN achieves a comparable performance and, at the same time, is more computationally efficient than an LSTM network due to a smaller number of model parameters. The forecasting horizons are of variable lengths but all of the models are trained and tested for multi step predictions. Another extensive study by Chandra et

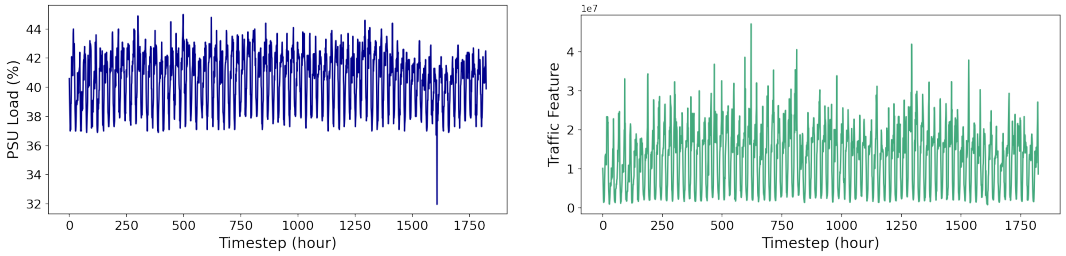
al. [8] makes a similar comparison demonstrating that an LSTM network outperforms other models on the given datasets. However, this study do not provide any information on the computational efficiency of implemented models.

4 Data

This chapter provides an overview of the data that is used in this thesis work and all the required preprocessing steps. In particular, the following are included: a general description of data, missing data imputation, correlation analysis, data standardization and data windowing.

4.1 Data Description

Data is in a time series format and consists of a target variable, *Power Supply Unit* (PSU) Load, and features describing *traffic* characteristics. PSU Load defined in percentage unit % measures an average of power loads for every PSU in RBS. Traffic features are included in this study to test their impact as exogenous variables on the target variable. They describe traffic characteristics in two directions: *uplink* and *downlink* [53]. These directions estimate data transmission from RBSs to a user equipment (downlink) and from a user equipment to RBSs (uplink). In total, 100 RBSs are available for this study. Each of these RBSs has 1824 data points of hourly power consumption measurements, PSU load, which corresponds to two and a half month data. Figure 4.1 demonstrates time series for the target variable, PSU Load, and one of the traffic features for one of the RBSs available for this study.



(a) Target variable

(b) One of the traffic features

Figure 4.1: Examples of time series for one of the RBSs available for this study. On the left is an example of time series for the target variable, PSU Load. On the right is an example of time series for one of the traffic features.

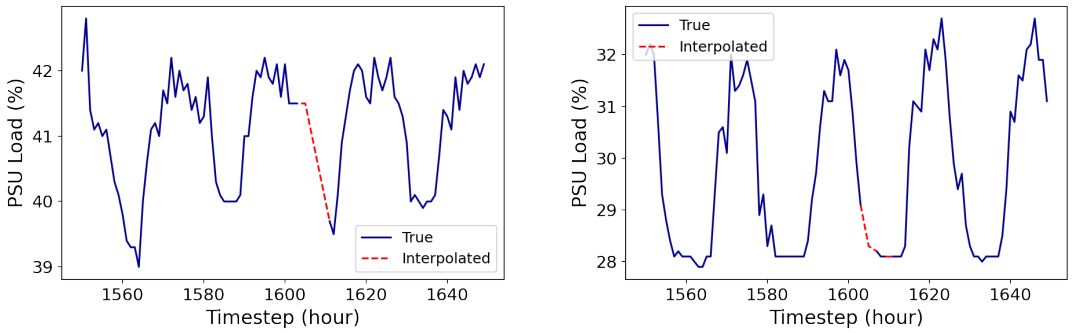


Figure 4.2: Examples of two different RBSs where linear interpolation is applied for missing data imputation.

4.2 Missing Data Imputation

The target variable and traffic features have a certain amount of missing data. When analyzing the target variable, this corresponds to maximum 0.4% (or 7 out 1824 data points) in a given RBS which can be considered a relatively small value in the context of the problem. Similar amount of missing data is observed for the traffic features. Thus, given such a small amount of missing data, the decision is to use a *linear interpolation* method [35] for missing data imputation.

In a linear interpolation, a single missing point y_1 located between equally distanced two points y_0 and y_2 is calculated in the following form

$$y_1 = \frac{y_2 - y_0}{2} + y_0. \tag{4.1}$$

To generalize equation 4.1 for n missing points in a row, the difference between known points is divided by n and a previous point is added

$$y_1 = \frac{y_n - y_0}{n} + y_0. \tag{4.2}$$

The problem with this approach is that initial and final value in a sequence cannot be interpolated. Thus, the approach is to fill the first and the last values of the sequence with the next and the previous values respectively.

Figure 4.2 demonstrates examples of two different RBSs where missing values of the target variable, PSU Load, are imputed using linear interpolation method.

4.3 Correlation Analysis

Correlation analysis is implemented to understand which of the available traffic features are correlated with power consumption and can be utilized in the forecasting model. As it was stated in section 4.1, the target variable and traffic features are time-dependent which means that it is important to account for *spurious correlations* [61]. Spurious correlation describes a relationship in which two variables appear to be associated with each other but not causally related due to the presence of the third variable. In the context of time series, this means that y is dependent on x, however, both y and x are only dependent on a time factor. When a variable y_t at some timestep t is correlated with itself at n previous timesteps $\{y_{t-1}, y_{t-2}, ..., y_{t-n}\}$ also known as *lags*, this behavior is known as *autocorrelation*. To diminish the impact of autocorrelation in the target variable and traffic features, seasonal differencing also known as *deseasonalisation* is applied. The operation is implemented in the following way (p.28, [7])

$$\nabla_d y_t = y_t - y_{t-d}, \tag{4.3}$$

where ∇_d is a lag - d differencing operator and y_t is the value of a time series at the t^{th} timestep. Analysis of *autocorrelation function* (ACF) plots can help to detect seasonality in data and determine if seasonal differencing diminishes the impact of autocorrelations.

Figure 4.3 demonstrates the impact of seasonal differencing with lag - $d = 24$. It can be observed that the impact of autocorrelation in a given time series was significantly reduced. In a similar way, seasonal differencing with lag - $d = 24$ is applied to the traffic features.

Spearman's rank correlation coefficient is selected to estimate a correlation between the target variable and the traffic features. The choice of Spearman's rank correlation coefficient is motivated by the fact that it is less sensitive to outliers and accounts for monotonic relationship between variables [46]. This is important in the context of this study considering that there are outliers present in some of the time series. Spearman's rank correlation coefficient is defined as Pearson's correlation coefficient between $R(x)$ and $R(y)$ which are ranked representations of x and y variables . Mathematically, this correlation is estimated using the following formula

$$r_s = \rho_{R(x),R(y)} = \frac{cov(R(x), R(y))}{\sigma_{R(x)}\sigma_{R(y)}}, \tag{4.4}$$

where $\rho_{R(x),R(y)}$ is Pearson's correlation coefficient, $cov(R(x), R(y))$ is a covariance between ranked variables and $\sigma_{R(x)}, \sigma_{R(y)}$ are standard deviations of ranked variables.

After estimating r_s between the target variable, PSU Load, and 26 traffic features in all RBSs, it can be observed from Figure 4.4 that only a group of traffic features that described traffic in downlink direction (T1_DL, T6_DL, T12_DL, T21_DL, T23_DL) are correlated with the target variable. This means that in total 5 different traffic features are utilized for further evaluation in this study.

4.4 Data Standardization

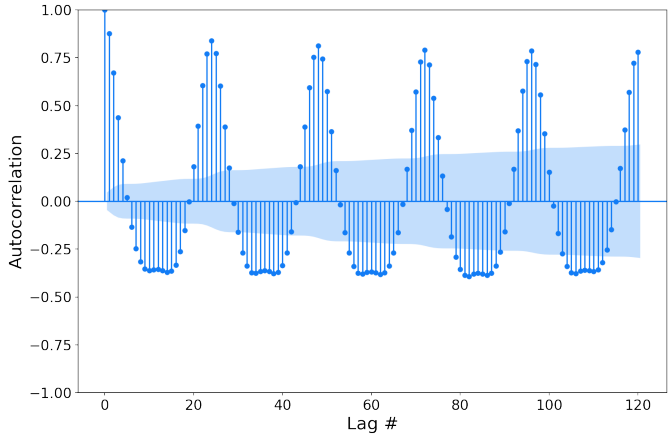
As mentioned in chapter 2, a neural network model is implemented in this study as a power consumption forecasting model. Early [50] [52] and more recent [39][51] studies of neural networks demonstrated that a performance improvement could be achieved when working with normalized or standardized data. In the context of the studied problem, data standardization makes sense due to substantial differences between the magnitude of the target variable and the traffic features. For example, when analyzing the values of the target variable and traffic features, it can be observed that, on average, traffic features are of a magnitude 10^5 larger than the target variable. Since the target variable itself is also considered to be an autoregressive feature, its predictive impact can be significantly diminished due to a relatively smaller magnitude in comparison to the traffic features. In addition, having the features and the target variable on a similar order of magnitude implies that the values of parameter θ should not be significantly different from each other meaning that gradient updates will behave similarly, as a result, facilitating optimization and improving the training process [5] [25]. For $x_i \in \{x\}_{i=1}^n$, data standardization refers to centering each data point x_i by subtracting $\overline{x}$, a mean of all data points, from it and scaling an obtained value by dividing by σ_x, a standard deviation of all data points. Mathematically, this is written in the following form

$$x_i^{st} = \frac{x_i - \overline{x}}{\sigma_x}, \tag{4.5}$$

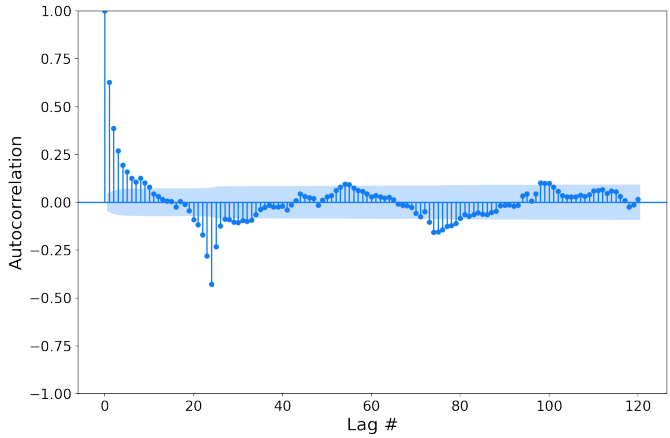
where x_i^{st} is a standardized form of $x_i \in \{x\}_{i=1}^n$.

4.5 Data Windowing

When working with time series data in neural network models such as CNN and LSTM, data should be introduced to the model in the form that is suitable for a supervised machine learning task. One way of doing that is transforming a single time series into many *data*



(a) Before differencing



(b) After differencing

Figure 4.3: Seasonal differencing impact on the target variable PSU Load.

Figure 4.4: Correlation analysis results for a subset of RBSs. Traffic features are denoted by the following principle: "T#" - traffic feature number, "DL" - downlink, "UL" - uplink.

windows of predefined length (p.428, [26]). Data windowing refers to the process of splitting a single time series into many windows of length $h + T$, where h next timesteps are predicted conditional on the previous T timesteps. After data windowing, equation 2.2 obtains the following form

$$\hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h} = f_\theta(y_t, y_{t-1}, ..., y_{t-T}). \tag{4.6}$$

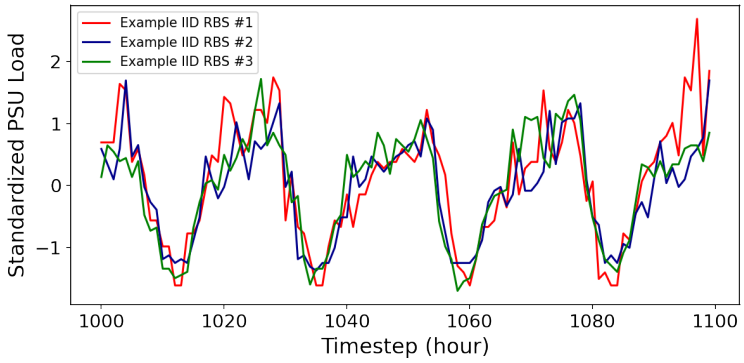
From equation 4.6, the goal is to predict next h timesteps, also known as *forecasting horizon* of length h, given a sequence of T previous timesteps. When adding k-dimensional feature vector, equation 4.6 takes the following form

$$\begin{aligned} \hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h} = f_\theta([y_t, y_{t-1}, ..., y_{t-T}], \\ [x_t^0, x_{t-1}^0, ..., x_{t-T}^0], [x_t^1, x_{t-1}^1, ..., x_{t-T}^1], ..., [x_t^k, x_{t-1}^k, ..., x_{t-T}^k]). \end{aligned} \tag{4.7}$$

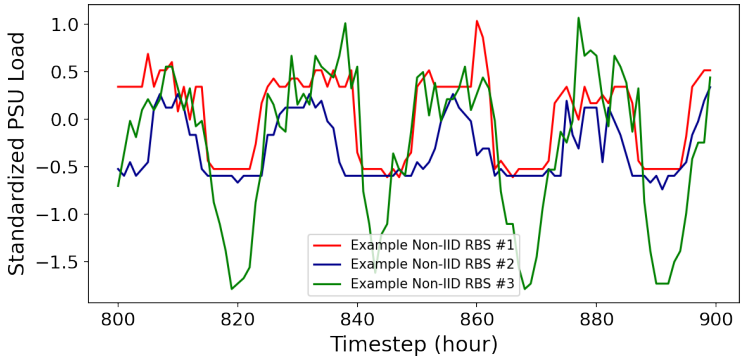
From a supervised machine learning perspective, each data window can be considered a sample data point where *features* are T previous timesteps of an autoregressive PSU Load variable and k-dimensional vector describing traffic characteristics, and a *target* is a sequence of h next timesteps of PSU Load variable.

4.6 Non-IID Data

As introduced in section 2.2.2, one of the open challenges for federated learning is the presence of non-iid data. As it can be observed from Figure 4.5, some of the RBSs are following similar distributions while others are not. In this study, the approach is to include all 100 RBSs into the model training process and evaluate the impact of non-iid RBSs based on the performances of localized, centralized and federated models.



(a) IID RBSs



(b) Non-IID RBSs

Figure 4.5: Examples of different RBSs which are iid and non-iid.

5 Methodology

The following chapter describes the methods and the evaluation scenarios implemented in this thesis work, which are based on the theoretical background that was introduced in chapter 2.

5.1 Data Preparation

As mentioned in section 4.1, 100 RBSs are available for modelling in this study. For each RBS, there is one target variable y and six features $X = \{y, x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}\}$ where subscripts for $x^{(n)}$ denote n^{th} traffic feature. Due to an autoregressive nature, the target variable y is also included in the feature set X. Although the correlation between the target variable and the traffic features was demonstrated in section 4.3, model comparison is done without traffic features as purely autoregressive, $X = \{y\}$, and with traffic features, $X = \{y, x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}\}$, to demonstrate the impact on the generalization performance.

Initially, data is split into training, validation and test sets by proportions of 80%, 10%,10% respectively. Then, data is standardized as per section 4.4, and data windowing is applied to each of the sets to transform time series into data windows. The number of data windows per a set equals to 1412, 135 and 136 for training, validation and test sets respectively.

5.2 Forecasting Task

In this thesis work, the forecasting task is to predict power consumption, PSU Load, of RBSs. Models are designed and evaluated separately for a forecasting horizon $h = 24$ given the input of 24 previous timesteps. Thus, the goal is to predict

$$\{\hat{y}\}_{t+1}^{t+h} = f(X = \{y\}_{t=t-23}^{T=t}) \text{ where } h = 24. \tag{5.1}$$

As mentioned in section 5.1, modelling is also implemented with traffic features included

$$\{\hat{y}\}_{t+1}^{t+h} = f(X = \{y, x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}\}_{t=t-23}^{T=t}) \text{ where } h = 24, \tag{5.2}$$

and a comparative analysis is conducted.

5.3 Modelling

It was demonstrated in section 3.4, that LSTM is a typical choice of architecture when working with a neural network model for time series forecasting. However, an overview of several studies in section 3.4 also demonstrates that CNN can be a competitive choice of architecture since it is also suitable for time series data. In this study, an experimental comparison of both LSTM and CNN is conducted to demonstrate which architecture choice results in a better generalization performance for PSU Load forecasting in RBSs. The performances of these models are compared with the performance of a persistence model also known as a *seasonal naive baseline* that is described in the following section.

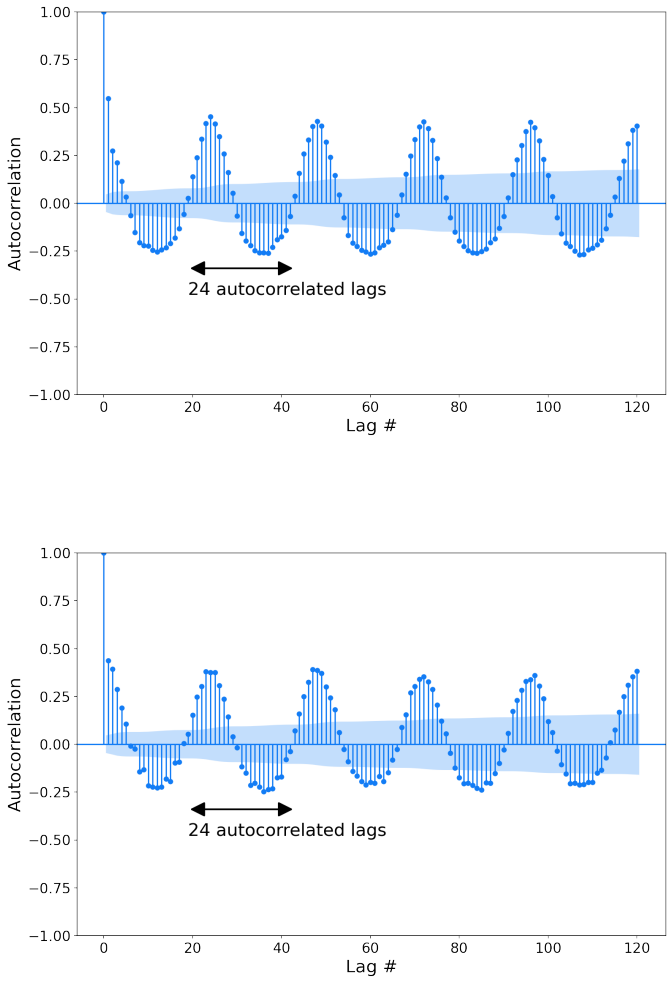


Figure 5.1: 24 autocorrelated lags are observed for two randomly chosen RBSs demonstrating that PSU Load time series has a daily seasonality.

5.3.1 Seasonal Naive Baseline

Seasonal naive model serves as a baseline model for a comparative analysis of model performances. This baseline is suitable for this study because PSU Load time series data has a seasonal pattern, as demonstrated in section 4.3. The working principle of this baseline is to

set each forecasted value of the next hour to be equal to the value at the same hour from the previous day considering that there is a daily seasonality (p.58, [24]). Mathematically, this is represented in the following form

$$\hat{y}_{t+1} = y_{t+1-s}\,, \tag{5.3}$$

where $\hat{y}_{t+1}$ is a prediction for $(t+1)^{th}$ timestep and y_{t+1-s} is a value of $(t+1-s)^{th}$ timestep with a period equal to $s = 24$. As it can be seen from Figure 5.1, the choice of $s = 24$ is validated by observations of 24 autocorrelated lags in time series of different RBSs that describe the target variable, PSU Load. Thus, predictions can be made for any future timestep h within defined period s in the following form

$$\hat{y}_{t+h} = y_{t+h-s}\,, \tag{5.4}$$

where a prediction is made for $(t+h)^{th}$ future timestep.

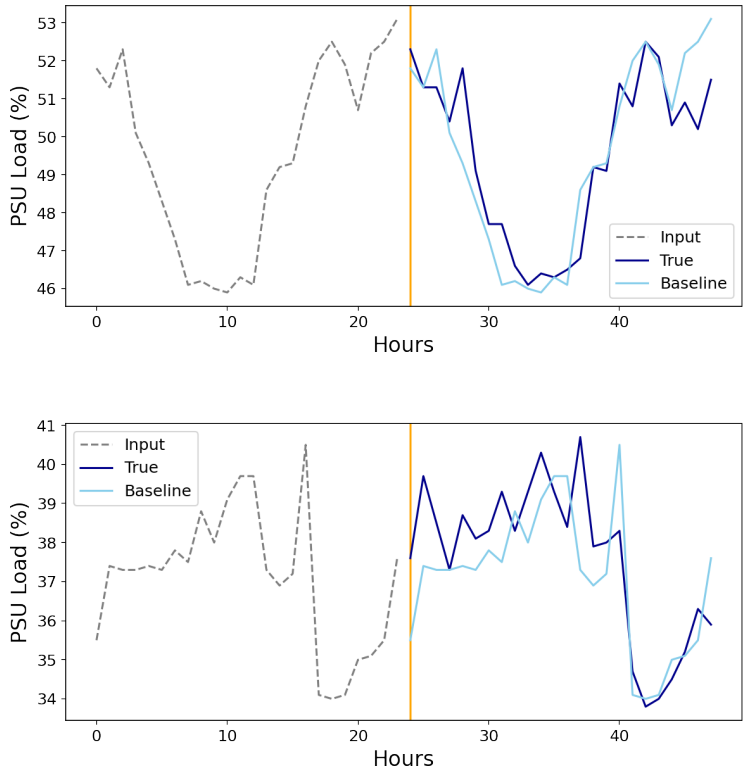


Figure 5.2: Examples of seasonal naive baseline predictions for two different RBSs.

Figure 5.2 demonstrates examples of seasonal naive baseline prediction windows for two different RBSs. It can be seen that the output predictions repeat input predictions as defined in equation 5.4, which is why this model is called a naive baseline.

5.3.2 CNN

CNN is one of the two types of neural network models implemented in this study. Theoretical background for CNN was provided in section 2.1.3, and this section describes the architecture of the implemented model.

As it can be seen from Figure 5.3, CNN architecture implemented in this study includes one convolutional hidden layer. This layer implements convolution operation described in

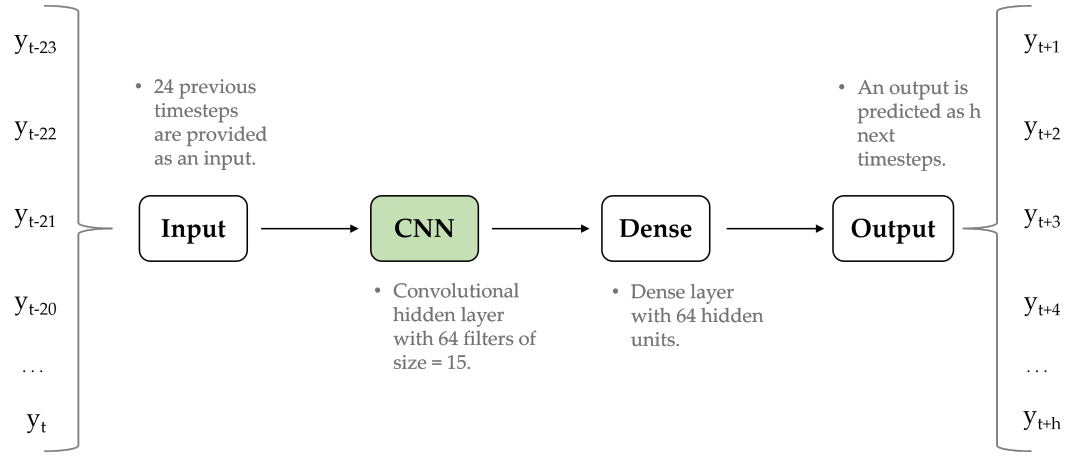


Figure 5.3: A neural network model with CNN architecture.

section 2.1.3 by processing a raw input of $\{y_{t-23}, y_{t-22}, ..., y_t\}$ with filter size equal to 15 and generating 64 convolved representations $\{q^f_{10}, q^f_9, ..., q^f_1\}^{F=64}_{f=1}$ also known as *channels*. These channels are then flattened into one-dimensional vector, which is considered a new representation of input generated by a convolutional layer, and passed through a dense layer. Finally, a dense layer produces an output of h next timesteps $\{\hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h}\}$.

5.3.3 LSTM

LSTM is the second type of neural network models implemented in this study. Similar to CNN, theoretical background for LSTM can be found in sections 2.1.4 and 2.1.4.1, while this section describes the architecture of the implemented model.

As it can be seen from Figure 5.4, a neural network with LSTM architecture consists of a recurrent hidden layer with LSTM cells. As described in section 2.1.4.1, a recurrent hidden layer sequentially processes a raw input $\{y_{t-23}, y_{t-22}, ..., y_t\}$ generating corresponding vectors of hidden $\{H^k_{t-23}, H^k_{t-22}, ..., H^k_t\}$ and cell states $\{C^k_{t-23}, C^k_{t-22}, ..., C^k_t\}$. Each of these vectors consists of $k = 64$ hidden units. After processing the last element of input sequence, a vector of hidden units H^k_t is considered a new representation of input. This new representation is passed through a dense layer. A dense layer processes a new representation of input and produces an output of h next timesteps $\{\hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h}\}$.

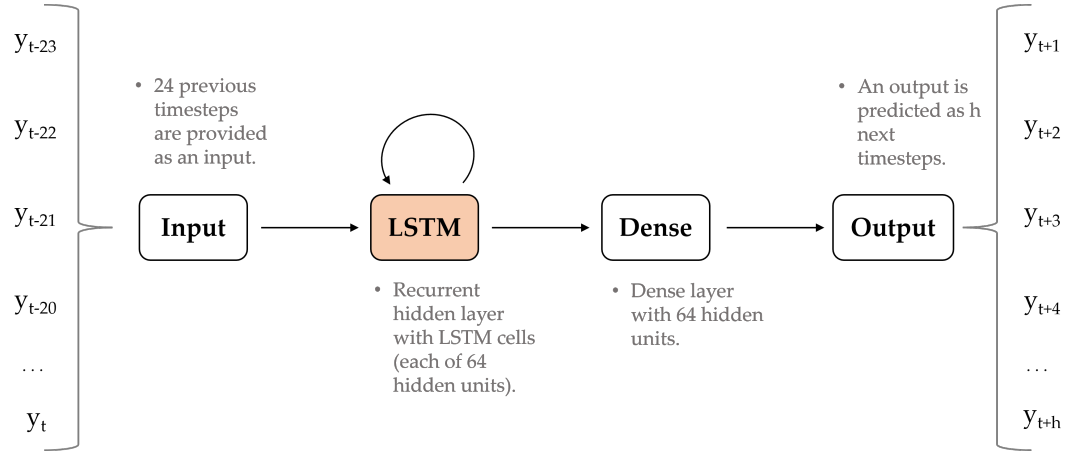


Figure 5.4: A neural network model with LSTM architecture.

5.4 Evaluation

5.4.1 Evaluation Metric

Root mean squared error (RMSE) is selected as a measure of differences between true y and predicted $\hat{y}$ values. This evaluation metric represents how far predictions fall from expected values using the Euclidean distance. This evaluation metric is an appropriate choice in the context of the forecasting problem in this study considering that a neural network was trained using MSE as a loss function. Mathematically, this metric is estimated using the following formula

$$RMSE = \sqrt{\frac{1}{h}\sum_{t=n+1}^{n+h}(y_t - \hat{y}_t)^2} \tag{5.5}$$

where y_t is an element of a true sequence $\{y_{t+1}, y_{t+2}, ..., y_{t+h}\}$, and $\hat{y}_t$ is an element of predicted sequence $\{\hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h}\}$. A use of RMSE instead of MSE as a performance evaluation metric is also motivated by its interpretability. RMSE reports an error in the same units as the target variable which make it easier to understand.

5.4.2 Evaluation Scenarios

Two types of neural networks, CNN and LSTM, and a seasonal naive baseline are initially compared to each other with and without traffic features in *centralized* and *localized* learning scenarios.

In the localized learning scenario, 100 separate models are trained per each RBS and evaluated accordingly. In the centralized learning scenario, the data from all RBSs is available as a single dataset, and a single model is trained using all the data. Then, the same model is used to make PSU Load forecasts for individual RBSs given corresponding input values.

Once the better performing architecture is selected from above-mentioned scenarios, the model with this architecture is then implemented in the *federated* learning scenario as described in section 2.2.1. The federated model is evaluated from different perspectives. First, the impact of hyperparameters specified in strategy 2.2.1 is demonstrated and elaborated. Second, its generalization performance is compared with localized and centralized learning scenarios.

5.4.2.1 Localized Learning

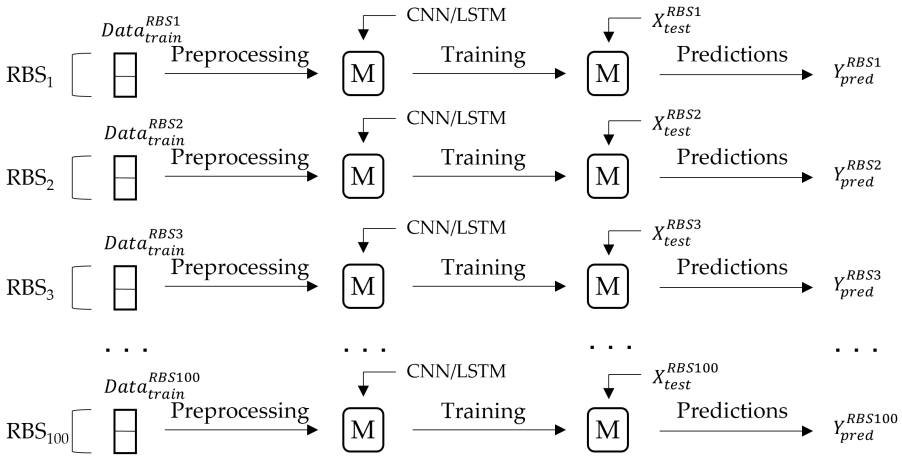


Figure 5.5: Localized Training.

A scheme of the localized learning scenario implemented in this study is demonstrated on Figure 5.5. As it can be seen, the data is distributed across 100 RBSs which are treated as separate data sources. The dataset of each RBS is preprocessed and used for training a separate model that is defined as `M` on Figure 5.5. CNN and LSTM are implemented and evaluated as two different models as described in sections 5.3.2 and 5.3.3 respectively. After an individual training of all 100 models, each RBS has its unique model that is used for making predictions.

5.4.2.2 Centralized Learning

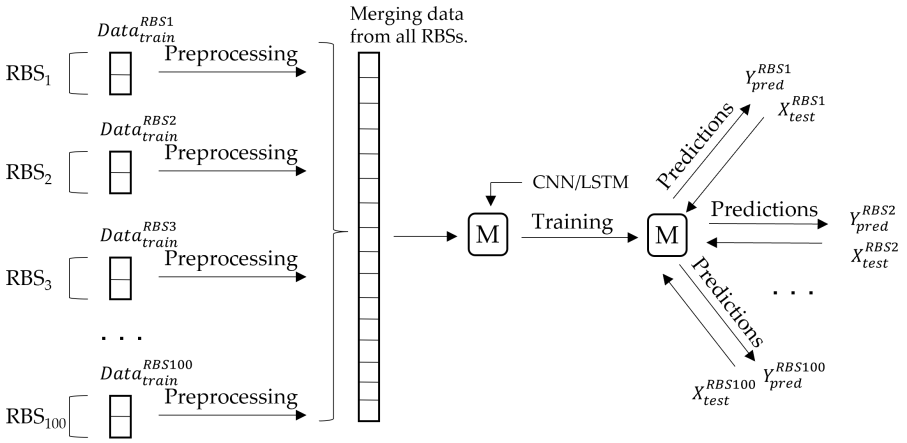


Figure 5.6: Centralized Learning.

A scheme of the centralized learning scenario implemented in this study is demonstrated on Figure 5.6. As it can be seen, the data is distributed across 100 RBSs which are treated as separate data sources. In contrast to the localized learning scenario, all datasets after preprocessing are merged into a single dataset. Then, this merged dataset is provided used for training a single model defined as `M` on Figure 5.6. CNN and LSTM are implemented and evaluated as two different models as described in sections 5.3.2 and 5.3.3 respectively. After training, a single model is obtained that can make predictions for any RBS conditional on the provided input.

5.4.2.3 Federated Learning

A scheme of the federated learning scenario implemented in this study is demonstrated on Figure 5.7. As it can be seen, the data is distributed across 100 RBSs which are treated as separate data sources. Similar to localized learning, each dataset is preprocessed separately. Federated training is governed by the server defined as `S` on Figure 5.7. The training procedure follows Algorithm 1 described in section 2.2.1. As in Figures 5.5 and 5.6, `M` denotes models that are trained at randomly sampled clients during each communication round. As it is demonstrated in the chapter 6, CNN is used as a model of choice during federated training process.

Figure 5.7 demonstrates several hyperparameters that are defined in *strategy* and have an impact on the federated training process. It can be observed that at step 2, the server randomly samples a fixed number of RBSs and sends to them aggregated parameters from the previous communication round. This hyperparameter is referred to as *number of sampled clients* (NSC) in section 6.3, and its impact on the federated training process is studied in the same section. Another hyperparameter of interest is defined in step 3 on Figure 5.7. Once the randomly sampled RBSs receive aggregated parameters from the server, they start local training for a certain number of epochs. This hyperparameter is referred to as *number of*

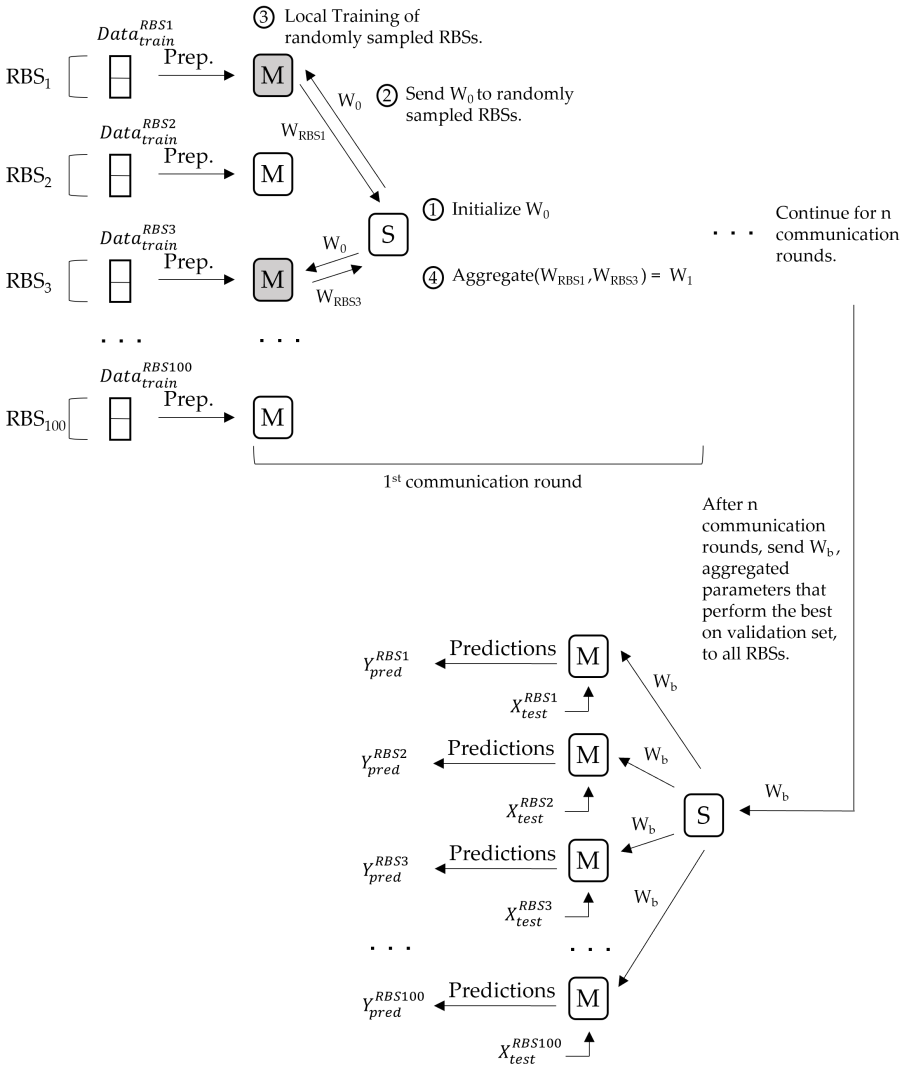


Figure 5.7: Federated Learning.

local epochs (NLE) in section 6.3 and its impact is also studied. Finally, it can be observed that the federated training process continues for n communication rounds which is another hyperparameter that is referred to as *number of communication rounds* (NCR) in section 6.3.

The motivation to evaluate the impact of the above-mentioned hyperparameters comes from the original work [36] by McMahan et al. where different experiments were run to study the federated training process. However, McMahan et al. studied C, a fraction of available clients at each communication round, as an alternative to NSC. In this work, an assumption is that after each communication round a fixed number of clients are available meaning that the process is less heterogeneous. As a part of future work, this assumption can be eased, and the corresponding hyperparameter can be also studied.

5.4.3 Neural Network Uncertainty

As described in section 2.1.2.2, when trying to find optimal parameters $\hat{\theta}$ in a neural network model, a parameter initialization is required to start an optimization process. A training process of neural networks can be strongly affected by the choice of a parameter initialization (p.301, [20]). A parameter initialization can determine whether an algorithm reaches a convergence, how quickly it converges and whether a convergence point has a high or a low

loss value. Eventually, a parameter initialization may have a strong impact on generalization performance of a model.

In this study, the parameters of convolutional, recurrent and dense layers are initialized by sampling from normalized uniform distribution also known as *Glorot initialization*. Mathematically, this sampling is represented in the following form (p.303, [20])

$$W \sim U(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}) \tag{5.6}$$

where m are inputs and n are outputs of a fully connected layer. This is a default choice for a parameter initialization in neural networks that was proven to result in a faster convergence [19]. However, a faster convergence does not always imply a better generalization performance (p.301, [20]. Thus, the performances of the neural models implemented in centralized, localized and federated scenarios are averaged after 10 different parameter initializations, and corresponding variances are also provided to demonstrate uncertainties of predictions and their robustness to random parameter initializations.

6 Results and Discussion

In this section, quantitative and qualitative analyses of methods and learning scenarios introduced in chapter 5 are presented. First, the performances of two neural networks, LSTM and CNN, implemented with and without traffic features in centralized and localized learning scenarios are compared to the performance of a seasonal naive baseline model. Second, the best performing model among CNN and LSTM is selected and analyzed separately in centralized and localized learning scenarios. Third, the corresponding model is implemented in the federated learning scenario, and a discussion is provided on the selection of hyperparameters. Finally, this chapter is concluded by the comparison and analysis of the the selected model implemented in centralized, localized and federated scenarios.

6.1 Performance Analysis of Neural Networks

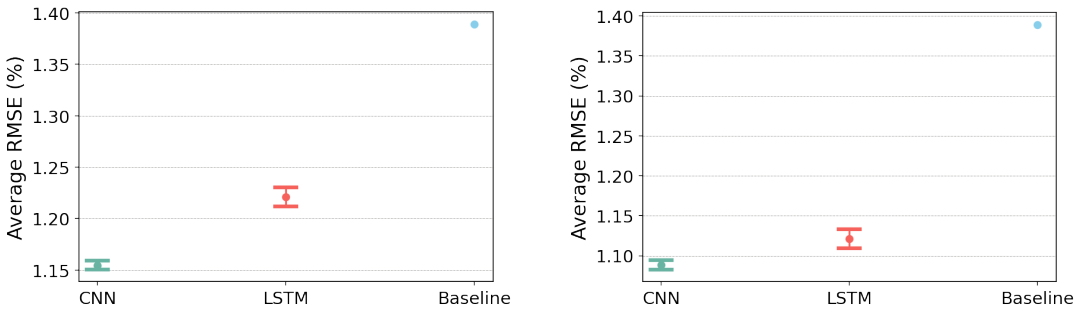
The performances of two types of neural networks, CNN and LSTM, are compared to each other and a seasonal naive baseline in centralized and localized learning scenarios.

From Figure 6.1, it can be seen that both CNN and LSTM models without traffic features outperform a seasonal naive baseline. The performance evaluation metric, RMSE, is estimated as an average RMSE across all 100 RBSs. Furthermore, it can be observed that CNN outperforms LSTM in both scenarios.

When looking at the performance of CNN and LSTM without traffic features on a case-by-case level, it can be observed that there are 25 RBSs for which LSTM outperforms CNN in the localized learning scenario. However, since the interest is in the average model performance across all RBSs, the models are judged on the average performance.

Different prediction windows are demonstrated in Figure 6.2 where LSTM performs better than CNN. It can be seen that there is a very marginal difference between predictions made by both models. In the centralized learning scenario, the results are worse for LSTM, and there are only 13 RBSs for which LSTM outperforms CNN.

From Figure 6.3, when comparing the majority of the cases where CNN outperforms LSTM, it can be observed that CNN is better at capturing more granular information from a true sequence. This behavior can be explained by the ability of CNN to account for a spatial structure of the time series that is captured by a convolutional filter. LSTM learns a more general trend of true sequence smoothing out local ups and downs.



(a) Localized Learning

(b) Centralized Learning

Figure 6.1: Comparison of CNN, LSTM and Baseline model performances without traffic features in localized and centralized learning scenarios.

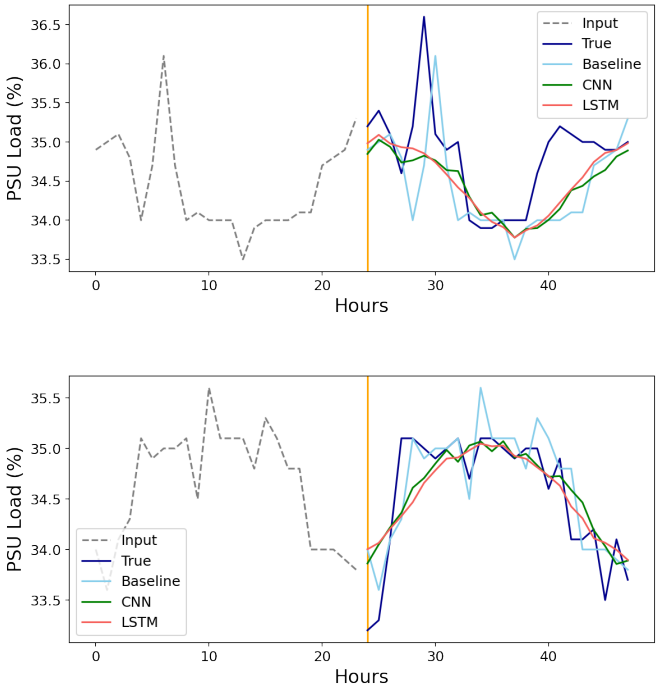


Figure 6.2: Prediction window examples from two different RBSs where LSTM performs better than CNN without traffic features in the localized learning scenario.

When looking at the performance impact of traffic features in Figure 6.4, there is a deterioration in predictive performance of both CNN and LSTM models. However, a case-by-case analysis reveals that for 20 out of 100 RBSs for CNN and 28 out of 100 RBSs for LSTM, the performance actually improves. This validates one of the previous studies by Valencia et al. [55] where it was demonstrated that traffic features in addition to autoregressive and climate related features can improve predictive performance of the model. However, since the interest is in the average performance across all RBSs, traffic features will not be used for the rest of the study.

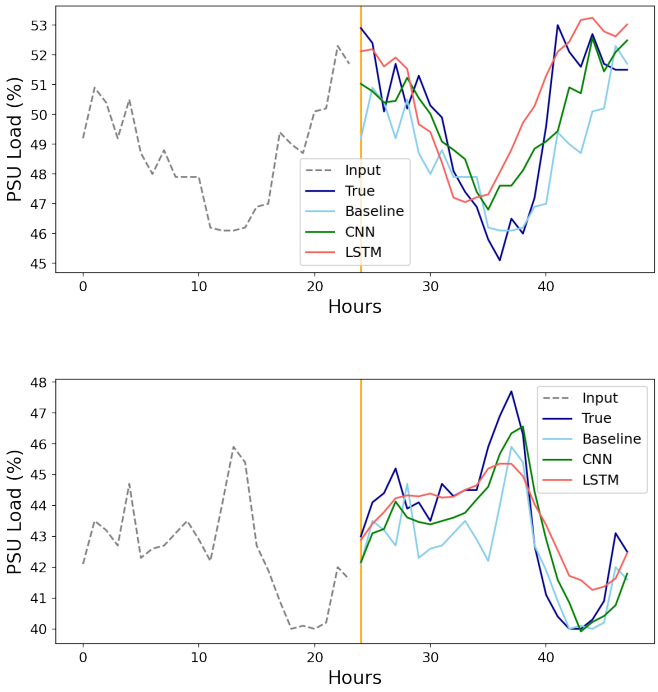
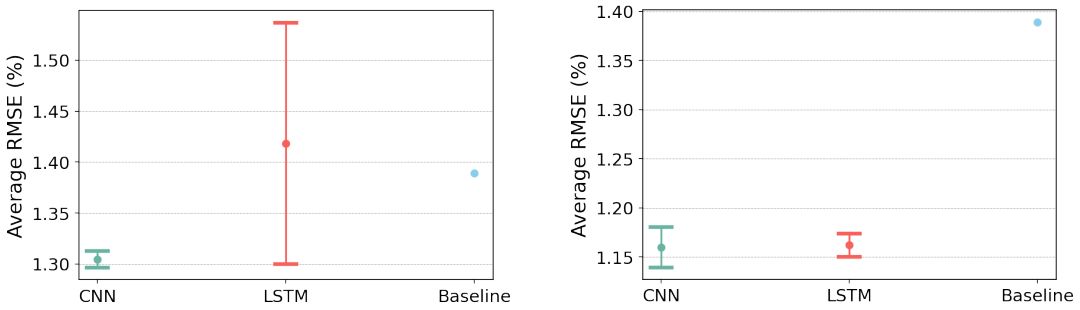


Figure 6.3: Prediction window examples from two different RBSs where CNN performs better than LSTM without traffic features in the localized learning scenario.



(a) Localized Learning

(b) Centralized Learning

Figure 6.4: Comparison of CNN, LSTM and Baseline model performances with traffic features in localized and centralized learning scenarios.

6.2 CNN in Centralized and Localized Learning

The analysis is continued by comparing CNN, as a better performing neural network model in this use case, to a seasonal naive baseline on a case-by-case level. Although CNN outperforms a seasonal naive baseline in 86 out of 100 RBSs in localized learning, a further analysis of the remaining 14 cases shows very interesting patterns. Some of those cases are demonstrated in Figure 6.5. There, the data is intentionally demonstrated as three separate sets: training, validation and test. This color differentiation allows to see that the data in test set follows a different distribution than the data in training set. Such a behaviour can be considered anomalous in comparison with other RBSs where a similar behavior is observed in training, validation and test sets. In Figure 6.6a, when looking at the predictions made by CNN for RBSs in the localized learning scenario, it can also be seen that CNN's predictions

are higher than a true sequence what underlines the inability of a neural network to predict a different pattern that was not encountered in the training set. However, when looking at Figure 6.6b, it can be observed that CNN predictions are much better when trained in the centralized learning scenario. This demonstrates that when a larger amount of RBSs are available in the training set, it allows a neural network to learn a larger amount of patterns improving its predictive ability.

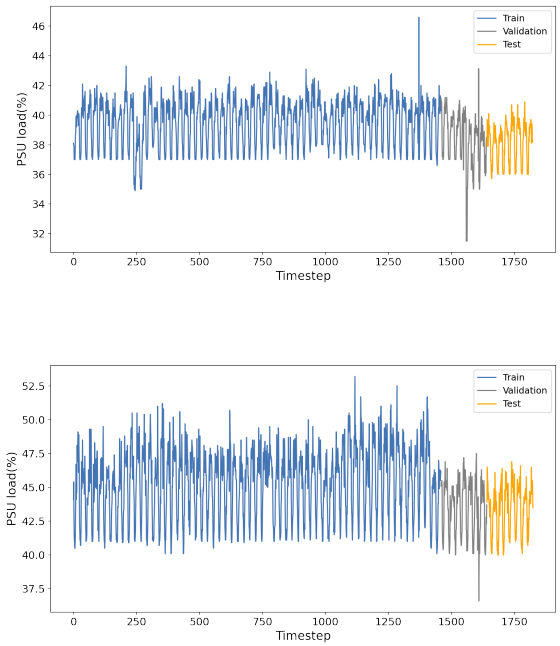
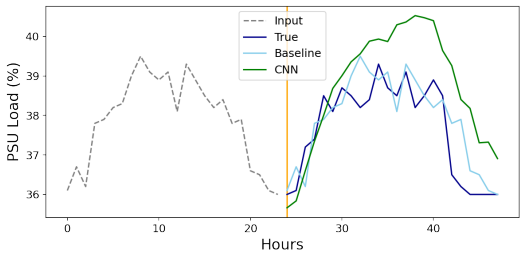


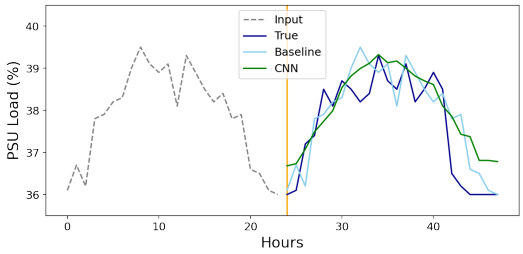
Figure 6.5: PSU Load time series from two different RBSs demonstrating a shift in data distribution where CNN cannot outperform a seasonal naive baseline in the localized learning scenario.

In the centralized learning scenario, there are 98 out of 100 cases where CNN outperforms a seasonal naive baseline. The time series for two underperforming cases are demonstrated in Figure 6.7. Similarly to underperforming RBSs in the localized learning scenario, there is a shift in PSU Load that occurs in the tail of the time series. The comparison of predictions made by CNN in centralized and localized learning scenarios for that RBS demonstrates that there is actually performance deterioration in the centralized learning scenario. Furthermore, it can be observed that there are 20 out of 100 RBSs for which performance deteriorates in centralized learning, and, subsequently, 80 out of 100 RBSs for which performance improves. One of the explanations for that can be the presence of outlying RBSs which cannot learn from each other and have an adverse impact on the training of the remaining RBSs.

As the above-mentioned findings demonstrate, overall, there is an improvement not only in the average but also case-by-case performances when the same CNN model is implemented in localized and centralized learning scenarios. This validates the initial hypothesis that RBSs may have similar characteristics and, if available in a single storage, the overall and individual predictive performances can be improved. However, it should not be disregarded that there can be situations where the performances of some RBSs deteriorate in centralized learning.

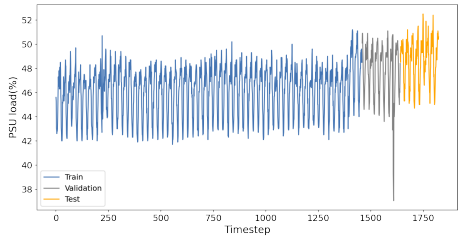


(a) Localized Learning



(b) Centralized Learning

Figure 6.6: Prediction improvement for the same RBS in localized and centralized learning scenarios.



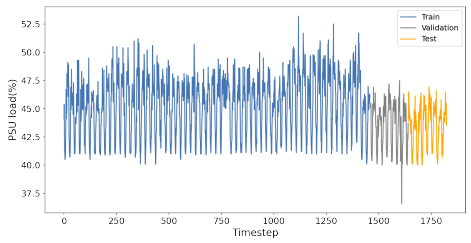
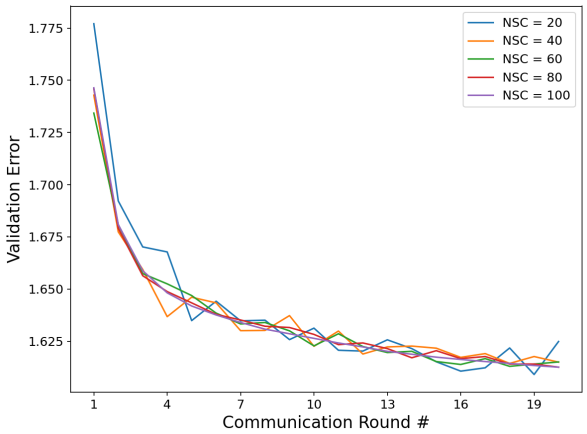


Figure 6.7: PSU Load time series from two different RBSs demonstrating a shift in data distribution where CNN cannot outperform a seasonal naive baseline in centralized learning scenario.

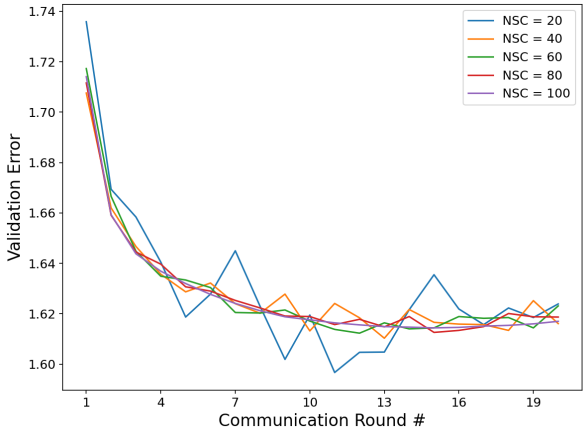
6.3 Hyperparameter Tuning in Federated Learning

As it was discussed in section 5.4.2.3, a federated training process is regulated by a strategy where a set of different hyperparameters are specified. The following hyperparameters: number of communication rounds (NCR), number of sampled clients at each communication round (NSC) and number of local training epochs (NLE), are analyzed in this section to study their impact on the federated training in the context of the forecasting task. Validation errors

demonstrated in the following figures are RMSE values evaluated using the model's predictions on the validation data that is used for hyperparameter tuning in federated learning.



(a) NLE = 5



(b) NLE = 10

Figure 6.8: The impact of different NSC values on the federated training process for NLE = 5 and NLE = 10. In both cases, NCR = 20.

The changes in validation errors of different federated training processes with NLE = 5 and NLE = 10 are shown in Figure 6.8. In both cases, each of the curves corresponds to a certain NSC number. There is a common behavior that is observed in both figures. Although validation errors continue to decrease at each communication round for all NSC values, the curves corresponding to smaller NSC values show a larger variation in validation errors than the curves with higher NSC values. Similar patterns are observed for both NLE = 5 and NLE = 10.

An interesting observation in Figure 6.8b is that it is possible to reach smaller validation errors with smaller NSC values than with higher ones. In addition, when comparing both Figure 6.8a and 6.8b, it can be observed that validation errors reach minima for NLE = 10 between 8^{th} and 15^{th} communication rounds in 6.8b while the errors continue to decrease in 6.8a where NLE = 5. Thus, the validation errors obtained with NLE = 5 should be observed for higher NCR values.

When running experiments with higher NCR and different NSC values, it can be observed in Figure 6.9 that there is no significant change in validation errors after 20^{th} communication round, and validation errors fluctuate irregularly. Interestingly, it can be observed that de-

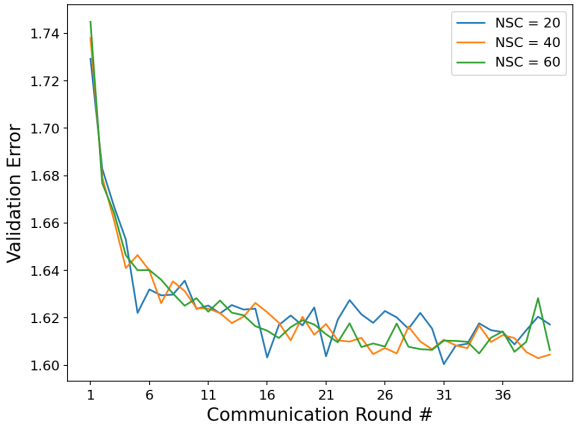


Figure 6.9: The impact of NCR = 40 on the federated training process with NLE = 5.

spite validation errors fluctuating for NSC = 20, on several occasions, the obtained errors are smaller than errors for higher NSC values.

Another observation is that, in general, the validation errors for NLE = 5 and NCR = 40 (on Figure 6.9) are smaller than errors for NLE = 10 and NCR = 20 (on Figure 6.8b) when NSC $\in \{40, 60\}$. Such a behavior can be an indicator of the scenarios where large NLE values may not be desirable for training processes with large NSC values, since the model may converge relatively faster without reaching potentially lower bounds of errors.

An opposite is observed for NSC = 20 and NCR $\in \{20, 40\}$ in Figure 6.8b and 6.9. Despite having a larger variation, the training process with these hyperparameters on several occasions reaches lower errors that the rest of the models. One analogy for such a behavior can be found when comparing gradient descent and stochastic gradient descent optimization algorithms.

In gradient descent, the gradient is estimated with respect to all available training data and, assuming a convex optimization problem, it may gradually reach a global minimum. However, for a stochastic gradient descent, the gradient is estimated with respect to the subset of the training data, commonly referred to as a mini-batch. As it was mentioned in section 2.1.2.2, it is important to ensure that different mini-batches are representative of the whole training data. Due to a random subsampling, mini-batching results in a random noise in the estimated gradients subsequently leading to the noise in estimated validation errors. One advantage of mini-batching is a computational efficiency since the gradient is calculated only with respect to a subset of training data. Furthermore, mini-batching may potentially lead to smaller errors than the gradient calculated with respect to full training data, because it is possible that full training data can contain outliers and noisy information that is omitted by a random subsampling. This implies that a randomly subsampled mini-batch can be a better representative of true data generating distribution than the full training set.

Similar conclusions can be made on the impact of different NSC values when looking at Figure 6.8b and Table 6.1. As it was stated above, for smaller NSC values, there is more variation in validation errors but, at the same time, such a training process may reach smaller errors than the training processes with higher NSC values. This process reminds of stochastic gradient descent due to a random sampling of 20 different RBSs out of 100 available RBSs at each communication round, and it is possible that at some communication rounds RBSs with outlying behavior are not presented in a subsample while at the next round they are.

Since it is of interest to find the model parameters that result in the smallest validation error with the smaller number of communication rounds, it is reasonable to proceed with NSC = 20 and NCR = 20 as hyperparameters for a federated model. In general, this is a trade-

off to deal with when considering the amount of available computational power, the number of outlying RBSs and a targeted margin of error.

Communication Round #	NSC = 20	NSC = 40	NSC = 60	NSC = 80	NSC = 100
8	1.6019	1.6278	1.6215	1.6191	1.6188
9	1.6195	1.6132	1.6171	1.6189	1.6176
10	1.5967	1.6241	1.6138	1.6157	1.6164
11	1.6047	1.6184	1.6123	1.6178	1.6156
12	1.6048	1.6103	1.6164	1.6148	1.6149

Table 6.1: Validation errors corresponding to the training processes illustrated in Figure 6.8b between 8^{th} and 12^{th} communication rounds. The values shaded in a gray color indicate the smallest errors across all NSC values.

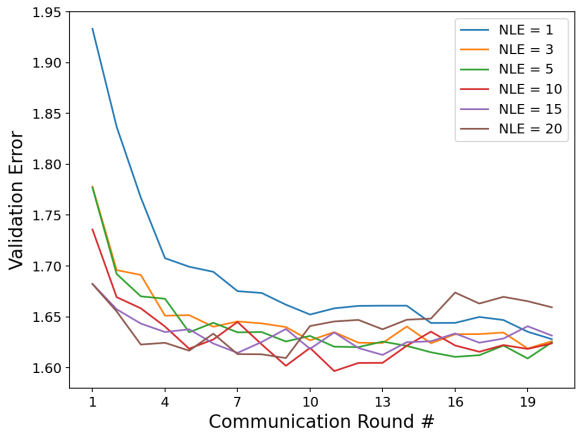


Figure 6.10: The impact of different NLE values on the federated training process with NSC = 20.

Finally, for NSC = 20 and NCR = 20, different NLE values are also tested on Figure 6.10. It can be observed that for NLE = 1 and NLE = 20 , lower and upper bounds of validation errors are obtained. For example, when NLE = 1, in general, validation errors are higher than for other NLE values. When NLE = 20, the errors are smaller at the initial communication rounds but start to increase and eventually become larger than for the case with NLE = 1. This validates previous choices of an optimal NLE $\in \{5, 10\}$ being between 1 and 20 as shown in the previous paragraph.

6.4 Comparison of Centralized, Localized and Federated Learning

In section 6.2, CNN model was implemented in centralized and localized learning scenarios. In this section, as a final step, CNN model is implemented in the federated learning scenario using best found hyperparameters (NSC = 20, NCR = 20, NLE = 10) defined for federated training in section 6.3.

Figure 6.11 demonstrates the generalization performance of CNN model implemented in localized (LM), centralized (CM) and federated learning (FM) scenarios. It can be seen that CM has a better generalization performance that both LM and FM. FM has a better generalization performance than LM model but performs worse than CM.

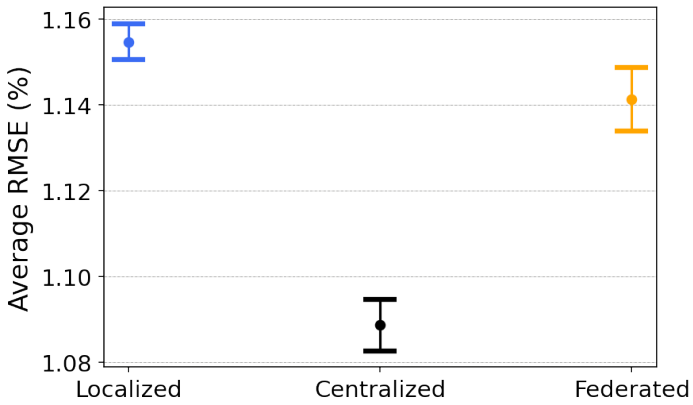


Figure 6.11: A model performance comparison in centralized, localized and federated learning scenarios.

When looking on a case-by-case level at the performance of FM model, it is observed that, on average, FM outperforms a seasonal naive baseline for 88 out of 100 RBSs. This performance is also better than the one observed for LM (86 out of 100 RBSs). Furthermore, the performances of 60 RBSs are improved with FM when compared to LM. This indicates that although the overall performance of FM is not as good as CM, FM is still able to improve predictions for individual RBSs by leveraging a larger amount of data that is not available to LM.

One of the reasons for the performance gap between FM and CM can be the presence of RBSs that are non-iid as it was demonstrated in Figure 4.5. However, it appears that the impact of such RBSs is relatively small given the fact that both overall and individual performance improvements are observed in FM when compared to LM.

It is also important to mention that evaluation metrics are averaged across 10 different parameter initializations of CNN model in all learning scenarios. From Figure 6.11, it can be seen that the smallest variance is obtained by LM, followed by CM and FM. Having FM with a higher variance of RMSE than LM and CM is expected considering the heterogeneity of training process driven by a random sampling of 20 out 100 clients at each communication round. A higher variance of generalization performance in FM also indicates that a very good understanding of not only the training process but the data generating distribution is required. In federated learning, there is usually no access to all available training data, and it becomes difficult to judge the presence of outliers (similar to above-mentioned anomalies) or RBSs that are non-iid (as it was demonstrated in section 4.6).

7 Conclusion

The following chapter finalizes this thesis work by answering the research questions from section 1.2 and addressing limitations of this study that can be considered in future work.

7.1 Answers to Research Questions

The purpose of this thesis work is to explore the application of federated learning for power consumption forecasting in a large number of RBSs. First, two types of neural network models, CNN and LSTM, are implemented in centralized and localized learning scenarios. The performance comparison of these models reveals that CNN outperforms LSTM according to RMSE evaluation metric and is also more robust to random parameter initializations of a neural network when considering the variance of errors across 10 parameter intializations. The superior performance of CNN can be attributed to the situations where it learns to account for a spatial structure in data better than LSTM network which can mainly predict the general trend. This is not always the case and depends on the amount of noise in time series. It is also demonstrated that traffic features do not necessarily improve model performances, and if they are to be utilized in the model, it is important to evaluate their impact on an individual level.

The first research question is answered by comparing the performances of CNN model in centralized and localized learning scenarios. It is shown that the model trained in the centralized learning scenario outperforms the same model trained in the localized learning scenario. An average performance improvement is also followed by individual performance improvements resulting in a higher number of RBSs that outperform a seasonal naive baseline. These results demonstrate that it is possible to obtain a better model when collecting data from many different RBSs in a single place. However, an individual performance deterioration of some RBSs should not be overlooked meaning that although PSU Load profiles of most RBSs can be similar to each other, there can be RBSs with anomalous or simply different profiles that cannot learn from others. Thus, the answer to the first research question makes it feasible to study the second research question.

As it is discussed in 1.1, federated learning can allow to leverage large amounts of data in situations when centralized learning is not possible. In this case, the lower and upper benchmarks for evaluating federated learning performance are localized and centralized learning respectively. In this study, CNN model implemented in federated learning outperforms

the model implemented in localized learning and performs worse when compared to the same model in centralized learning. Moreover, individual performance improvements are observed in RBSs that are trained in federated learning when compared to localized learning. The presence of outlying and non-iid RBSs still makes it challenging for the federated model to reach the performance of the centralized model. It is also very important to have a good understanding of hyperparameters' impact on the federated training process to ensure a good model convergence. The choice of these hyperparameters is not simply user driven and can depend on the amount of computational resources that are available and the desirable accuracy of the final model.

7.2 Limitations and Future Work

There are several limitations in this study that can be considered in future work. First of all, individual architectures of both neural network models can be improved. For example, there is an extension for CNN model that is, in particular, suitable for working with time series data known as *temporal convolutional network* (TCN) [4]. As an alternative to LSTM model, there is a more lightweight model, GRU, that trains faster [9] by learning smaller number of parameters and may achieve similar if not a better performance. Since the forecasting task is to predict a 24-hour window, both CNN and LSTM models can be augmented with a sequence-to-sequence architecture [18] [54]. One interesting alternative to neural network models can be tree-based methods that rely on gradient boosting technique. In recent years, these methods were proven to be very successful and achieved promising results [34]. Second, the feature set can be augmented further to account for a seasonal component of the time series. As the model predictions show, this component is still learned by neural networks, however, adding it as a separate feature can help a neural network to learn different patterns leading to a potentially better generalization [21].

Hyperparameters play an important role in the federated training process. The trade-off between the number of communication rounds, the number of sampled clients at each communication round and the number of local training epochs can be studied further with different aggregation functions [32] [47] [58] taking into consideration the amount of available computational resources. This can be of interest in situations when not all RBSs have a similar amount of computational resources. In addition to that, it can be interesting to study different fractions of available clients at each communication round and corresponding impact on the training, since a local training of different RBSs can terminate at different periods of time.

Split learning [56] is another type of distributed learning paradigm that can be utilized as an alternative to or in combination with federated learning. In comparison with federated learning, there is a topology step in split learning that allows to divide a model architecture into several parts some of which are located at the client side, and some are at the server side. Thus, the model training process is initiated at the client side and continued at the server side. This can allow to offload some parts of the local model training to the server and add more flexibility to the management of the computational resources at each RBS.

In this study, the model trained in centralized learning outperforms the model trained in localized learning, however, there are individual RBSs for which performance actually deteriorates. This indicates that not all RBSs can learn from each other and some may behave anomalously or follow different distributions. Consequently, this impacts the federated training process. Considering that in the federated training process, a set of different RBSs is sampled at each communication round, it may happen that at a given communication round a very heterogeneous set of RBSs is sampled that can result in a model divergence. To account for non-iid data, the possibility of designing several federated models instead of a single one can be studied as proposed by Briggs et al. [6]. This can be done by clustering RBSs with similar parameter updates under separate federated models that could lead to the customization of these models and improve both overall and individual performances.

7.3 Ethical Considerations

In this thesis work, federated learning is studied as a machine learning paradigm that allows to build a global model with a competitive generalization performance that does not have a direct access to data. At the same time, it is worth mentioning that privacy preservation and data sensitivity are among the major reasons of why federated learning was introduced in the first place. Privacy and data sensitivity issues may arise when trying to collect a health record data from multiple hospitals or energy consumption data of individual households and store this data in a single place. Thus, when putting a federated model into production, it is necessary to prioritize its privacy preservation properties alongside a competitive generalization performance.

Bibliography

[1] R. Adams and C. Essex. *Calculus: a Complete Course + Mylab Math with EText*. Pearson Education Australia, 2017. ISBN: 9780134588674. URL: https://books.google.se/books?id=REK9tQEACAAJ.

[2] Cecilia Andersson, Jonas Bengtsson, Greger Byström, Pål Frenger, Ylva Jading, and My Nordenström. "Improving energy performance in 5G networks and beyond". In: *Ericsson Technology Review* 2022.8 (2022), pp. 2–11. DOI: 10.23919/ETR.2022.9911220.

[3] Oliver Arnold, Fred Richter, Gerhard Fettweis, and Oliver Blume. "Power consumption modeling of different base station types in heterogeneous cellular networks". In: *2010 Future Network Mobile Summit*. 2010, pp. 1–8.

[4] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

[5] C.M. Bishop. *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics. Clarendon Press, 1995. ISBN: 9780198538646. URL: https://books.google.se/books?id=T0S0BgAAQBAJ.

[6] Christopher Briggs, Zhong Fan, and Peter Andras. "Federated Learning for Short-Term Residential Load Forecasting". In: *IEEE Open Access Journal of Power and Energy* 9 (2022), pp. 573–583. DOI: 10.1109/OAJPE.2022.3206220.

[7] P.J. Brockwell and R.A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer International Publishing, 2016. ISBN: 9783319298528. URL: https://books.google.se/books?id=EGkljwEACAAJ.

[8] Rohitash Chandra, Shaurya Goyal, and Rishabh Gupta. "Evaluation of Deep Learning Models for Multi-Step Ahead Time Series Prediction". In: *IEEE Access* 9 (2021), pp. 83105–83123. DOI: 10.1109/ACCESS.2021.3085085.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[10] International Electrotechnical Commission. *Smart energy and smart grids*. https://https://www.iec.ch/energies/smart-energy. 2023.

[11] Bjorn Debaillie, Claude Desset, and Filip Louagie. "A Flexible and Future-Proof Power Model for Cellular Base Stations". In: *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. 2015, pp. 1–7. DOI: 10.1109/VTCSpring.2015.7145603.

[12] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[13] Lackis Eleftheriadis, Johan Pettersson, and Helene Hallberg. *Ancillary services to utilities using mobile network power infrastructure*. Apr. 2021. URL: https://www.ericsson.com/en/reports-and-papers/white-papers/balance-smart-grids-with-5g-backup-for-utilities.

[14] Jeffrey L. Elman. "Finding structure in time". In: *Cognitive Science* 14.2 (1990), pp. 179–211. ISSN: 0364-0213. DOI: https://doi.org/10.1016/0364-0213(90)90002-E. URL: https://www.sciencedirect.com/science/article/pii/036402139090002E.

[15] Ericsson. *Breaking The Energy Curve*. https://www.ericsson.com/en/news/2022/10/ericsson-publishes-breaking-the-energy-curve-report-2022. 2022.

[16] Mohammad Navid Fekri, Katarina Grolinger, and Syed Mir. "Distributed load forecasting using smart meter data: Federated learning with Recurrent Neural Networks". In: *International Journal of Electrical Power Energy Systems* 137 (2022), p. 107669. ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2021.107669. URL: https://www.sciencedirect.com/science/article/pii/S0142061521008991.

[17] Kunihiko Fukushima. "Neocognitron: A hierarchical neural network capable of visual pattern recognition". In: *Neural Networks* 1.2 (1988), pp. 119–130. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90014-7. URL: https://www.sciencedirect.com/science/article/pii/0893608088900147.

[18] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. "Convolutional sequence to sequence learning". In: *International conference on machine learning*. PMLR. 2017, pp. 1243–1252.

[19] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[21] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. "Recurrent neural networks for time series forecasting: Current status and future directions". In: *International Journal of Forecasting* 37.1 (2021), pp. 388–427.

[22] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012), p. 2.

[23] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[24] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[25] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[26] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer US, 2021. ISBN: 9781071614181. URL: https://books.google.se/books?id=5dQ6EAAAQBAJ.

[27] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. "Advances and Open Problems in Federated Learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210. ISSN: 1935-8237. DOI: 10.1561/2200000083. URL: http://dx.doi.org/10.1561/2200000083.

[28] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[29] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. The MIT Press, 2019. ISBN: 0262039427.

[30] Pedro Lara-Benıtez, Manuel Carranza-Garcıa, and José C. Riquelme. "An Experimental Review on Deep Learning Architectures for Time Series Forecasting". In: *International Journal of Neural Systems* 31.03 (2021). PMID: 33588711, p. 2130001. DOI: 10.1142/S0129065721300011. eprint: https://doi.org/10.1142/S0129065721300011. URL: https://doi.org/10.1142/S0129065721300011.

[31] Junshuang Li, Yuli Feng, and Yan Hu. "Load Forecasting of 5G Base Station in Urban Distribution Network". In: *2021 IEEE 5th Conference on Energy Internet and Energy System Integration (EI2)*. 2021, pp. 1308–1313. DOI: 10.1109/EI252483.2021.9713622.

[32] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. "Federated Optimization in Heterogeneous Networks". In: *Proceedings of Machine Learning and Systems*. Ed. by I. Dhillon, D. Papailiopoulos, and V. Sze. Vol. 2. 2020, pp. 429–450. URL: https://proceedings.mlsys.org/paper_files/paper/2020/file/38af86134b65d0f10fe33d30dd76442e-Paper.pdf.

[33] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. URL: https://smlbook.org.

[34] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "M5 accuracy competition: Results, findings, and conclusions". In: *International Journal of Forecasting* 38.4 (2022). Special Issue: M5 competition, pp. 1346–1364. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2021.11.013. URL: https://www.sciencedirect.com/science/article/pii/S0169207021001874.

[35] Encyclopedia of Mathematics. *Linear Interpolation*. https://encyclopediaofmath.org/index.php?title=Linear_interpolation. 2012.

[36] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *International Conference on Artificial Intelligence and Statistics*. 2016.

[37] Alexandra Mourato, David Duarte, Iola Pinto, and Pedro Vieira. "A novel and realistic power consumption model for multi-technology radio networks". In: *URSI Radio Science Bulletin* 2018.364 (2018), pp. 20–29. DOI: 10.23919/URSIRSB.2018.8486764.

[38] Jiahao Nan, Ming Ai, Aijuan Liu, and Xiaoyan Duan. "Regional-union based federated learning for wireless traffic prediction in 5G-Advanced/6G network". In: *2022 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. 2022, pp. 423–427. DOI: 10.1109/ICCCWorkshops55477.2022.9896702.

[39] Nazri Mohd Nawi, Walid Hasen Atomi, and Mohammad Zubair Rehman. "The effect of data pre-processing on optimized training of artificial neural networks". In: *Procedia Technology* 11 (2013), pp. 32–39.

[40] Chris Olah. *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. 2015.

[41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the Difficulty of Training Recurrent Neural Networks". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318.

[42] Vasileios Perifanis, Nikolaos Pavlidis, Remous-Aris Koutsiamanis, and Pavlos S. Efraimidis. *Federated Learning for 5G Base Station Traffic Forecasting*. 2022. DOI: 10.48550/ARXIV.2211.15220. URL: https://arxiv.org/abs/2211.15220.

[43] Eugenia Petrangeli, Nicola Tonellotto, and Carlo Vallati. "Performance Evaluation of Federated Learning for Residential Energy Forecasting". In: *IoT* 3.3 (2022), pp. 381–397. ISSN: 2624-831X. DOI: 10.3390/iot3030021. URL: https://www.mdpi.com/2624-831X/3/3/21.

[44] Hnin Pann Phyu, Diala Naboulsi, and Razvan Stanica. "Mobile Traffic Forecasting for Network Slices: A Federated-Learning Approach". In: *2022 IEEE 33rd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. 2022, pp. 745–751. DOI: 10.1109/PIMRC54779.2022.9977882.

[45] Nicola Piovesan, David López-Pérez, Antonio De Domenico, Xinli Geng, Harvey Bao, and Mérouane Debbah. "Machine Learning and Analytical Power Consumption Models for 5G Base Stations". In: *Comm. Mag.* 60.10 (Oct. 2022), pp. 56–62. ISSN: 0163-6804. DOI: 10.1109/MCOM.001.2200023. URL: https://doi.org/10.1109/MCOM.001.2200023.

[46] Andrijana Rebekić, Zdenko Lončarić, Sonja Petrović, and Sonja Marić. "Pearson's or Spearman's correlation coefficient-which one to use?" In: *Poljoprivreda* 21.2 (2015), pp. 47–54.

[47] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečn, Sanjiv Kumar, and H Brendan McMahan. "Adaptive federated optimization". In: *arXiv preprint arXiv:2003.00295* (2020).

[48] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[49] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[50] M. Shanker, M.Y. Hu, and M.S. Hung. "Effect of data standardization on neural network training". In: *Omega* 24.4 (1996), pp. 385–397. ISSN: 0305-0483. DOI: https://doi.org/10.1016/0305-0483(96)00010-2. URL: https://www.sciencedirect.com/science/article/pii/0305048396000102.

[51] Jie Shao, Kai Hu, Changhu Wang, Xiangyang Xue, and Bhiksha Raj. "Is normalization indispensable for training deep neural network?" In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13434–13444.

[52] J. Sola and J. Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems". In: *IEEE Transactions on Nuclear Science* 44.3 (1997), pp. 1464–1468. DOI: 10.1109/23.589532.

[53] ScienceDirect Topic Survey. *Downlink and Uplink Transmission*. https://www.sciencedirect.com/topics/engineering/downlink-and-uplink-transmission. 2023.

[54] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

[55] Agustín Valencia, Oleg Gorbatov, and Lackis Eleftheriadis. "Hardware alarms reduction in Radio Base Stations by forecasting Power Supply Units headroom". In: *Electric Power Systems Research* 213 (2022), p. 108519. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2022.108519. URL: https://www.sciencedirect.com/science/article/pii/S0378779622006186.

[56] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. "Split learning for health: Distributed deep learning without sharing raw patient data". In: *arXiv preprint arXiv:1812.00564* (2018).

[57] D.D. Wackerly, W. Mendenhall, and R.L. Scheaffer. *Mathematical Statistics with Applications*. Mathematical Statistics with Applications. Thomson Brooks/Cole, 2008. ISBN: 9780495385080. URL: https://books.google.se/books?id=d6IMnwEACAAJ.

[58] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. "Tackling the objective inconsistency problem in heterogeneous federated optimization". In: *Advances in neural information processing systems* 33 (2020), pp. 7611–7623.

[59] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[60] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[61] G Udny Yule. "Why do we sometimes get nonsense-correlations between Time-Series?–a study in sampling and the nature of time-series". In: *Journal of the royal statistical society* 89.1 (1926), pp. 1–63.

[62] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. "Federated learning on non-IID data: A survey". In: *Neurocomputing* 465 (2021), pp. 371–390. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.07.098. URL: https://www.sciencedirect.com/science/article/pii/S0925231221013254.