Western Graduate&PostdoctoralStudies

Western University

Scholarship@Western

Electronic Thesis and Dissertation Repository

4-20-2022 10:45 AM

Deep Learning For Load Forecasting With Smart Meter Data: Online And Federated Learning

Mohammad Navid Fekri, *The University of Western Ontario*

Supervisor: Grolinger, Katarina, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering

© Mohammad Navid Fekri 2022

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Artificial Intelligence and Robotics Commons, Data Science Commons, Electrical and Electronics Commons, Other Computer Engineering Commons, Power and Energy Commons, and the Software Engineering Commons

Recommended Citation

Fekri, Mohammad Navid, "Deep Learning For Load Forecasting With Smart Meter Data: Online And Federated Learning" (2022). *Electronic Thesis and Dissertation Repository*. 8466.
https://ir.lib.uwo.ca/etd/8466

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

Electricity load forecasting has been attracting increasing attention because of its importance for energy management, infrastructure planning, and budgeting. In recent years, the proliferation of smart meters has created new opportunities for forecasting on the building and even individual household levels. Machine learning (ML) has achieved great successes in this domain; however, conventional ML techniques require data transfer to a centralized location for model training, therefore, increasing network traffic and exposing data to privacy and security risks. Also, traditional approaches employ offline learning, which means that they are only trained once and miss out on the possibility to learn from new data. Online and Federated Learning are among the potential solutions to alleviate the mentioned concerns. Online models learn from data as they arrive while Federated Learning (FL) approaches train a single ML model in a distributed manner without requiring participants to share their data.

Consequently, this thesis investigates Online and FL for load forecasting with smart meter data. Deep learning typically requires large and diverse data streams; however, this data may not be readily available due to data collection issues/expenses, privacy and security concerns, or other reasons. Therefore, Recurrent Generative Adversarial Network is designed for generating realistic energy data.

To enable continuous learning from newly arriving data and adapting to new patterns without the need to re-train the model, a novel Online Adaptive Recurrent Neural Network (RNN) is proposed. RNN is employed to capture time dependencies while the online aspect is achieved by updating the RNN weights according to new data. The results show that the proposed approach achieves higher accuracy than the standalone offline approaches and other online algorithms.

For FL, an approach based on FedAVG was designed first: this synchronous approach waits for all clients to complete training before the server aggregates weights. Next, FedNorm, an asynchronous approach for FL, is proposed: it aggregates updates without waiting for lagging clients. To achieve this, FedNorm measures the clients' contributions considering similarities of local and global models as well as the loss function magnitudes. The experiments demonstrate that FedNorm achieves higher accuracy than seven state-of-the-art FL approaches.

Keywords: Federated Learning, Distributed Learning, Asynchronous Learning, Load Forecasting, Online Learning, Deep Learning, Recurrent Neural Networks, Generative Adversarial Models

Summary for Lay Audience

Electricity load forecasting has been attracting research and industry attention because of its importance for energy management, infrastructure planning, and budgeting. In recent years, the proliferation of smart meters and other sensors has created new opportunities for sensor-based load forecasting on the building and even individual household level.

Machine learning (ML) has achieved great successes in load forecasting; however, conventional ML techniques require data transfer to a centralized location for model training. This exposes data to privacy and security risks and increases network traffic. In addition, traditional approaches employ offline learning: they are only trained once and miss out on the possibility to learn from new data. Moreover, the energy consumption patterns change over time, for example when new equipment is installed, making traditional ML models built on old data inconsistent with new data. A different strategy in terms of design and learning is required to support training ML models across many clients without sharing data and to embrace changes in data, enable fast model adaptations, and capture new revealing patterns.

Online and Federated Learning are among the potential solutions to alleviate the mentioned concerns. Online models are well suited for load forecasting since they dynamically adjust to new patterns in the data while federated Learning (FL) collaboratively trains a shared ML model across many participants without requiring participants to share their local data.

Consequently, this thesis proposes novel Online and FL approaches for load forecasting with smart meter data. Large and diverse data streams for evaluating the online algorithm may not be available due to data collection issues, expenses, privacy and security concerns, or other reasons. To address the lack of energy data, Recurrent Generative Adversarial Network is designed for generating realistic energy data. Next, Online Adaptive Recurrent Neural Network (RNN) is proposed for continuous learning from new data and adapting to new patterns without the need to re-train the model. Two FL approaches are designed: synchronous and asynchronous. In the synchronous approach, each client must complete its work before the training process can continue while the asynchronous approach can tolerate lagging clients.

Co-Authorship Statement

This thesis contains the following manuscripts that have been published or submitted:

Chapter 4 was published in

> ***Mohammad Navid Fekri****, Ananda Mohon Ghosh, and Katarina Grolinger. Generating energy data for machine learning with recurrent generative adversarial networks. Energies,13(130), 2020.*

Chapter 5 was published in

> ***Mohammad Navid Fekri****, Harsh Patel, Katarina Grolinger, and Vinay Sharma. Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network. Applied Energy, 282, 2021.*

Chapter 6 was published in

> ***Mohammad Navid Fekri****, Katarina Grolinger, and Syed Mir. Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks. International Journal of Electrical Power and Energy Systems, 137, 2021.*

Chapter 7 was submitted

> ***Mohammad Navid Fekri****, Katarina Grolinger and Syed Mir. Asynchronous Adaptive Federated Learning for Load Forecasting with Smart Meters. IEEE Transactions on Industrial Informatics, Submitted.*

In all those publications, *Mohammad Navid Fekri* is the first author and his contribution includes conceptualization, methodology, evaluation, implementation, writing and revising manuscript drafts. *Ananda Mohon Ghosh* and *Harsh Patel* contributed in the implementation, running experiments, and validation. Industrial partners, *Syed Mir* and *Vinay Sharma*, provided knowledge about domain, guidance in respect to practical use, and feedback on the manuscript drafts. Supervisor, *Katarina Grolinger*, provided guidance and advice regarding methodology, experiments, validation, analysis and feedback on drafts.

Acknowlegements

Countless people supported my effort on this thesis. First of all, I would like to express my deepest gratitude to my supervisor, Dr. Katarina Grolinger. This work would not have been possible without her constant support, guidance, and assistance. Her levels of patience, knowledge, and ingenuity are something I will always keep aspiring to.

Thank you to my sister, Niloufar, for always being there for me and for telling me that I am awesome even when I did not feel that way. I am grateful for my mom Lida and dad Nader whose constant love and support keep me motivated and confident. My accomplishments and success are because they believed in me.

Finally, from the bottom of my heart I would like to say big thank you for all the research group members for their energy, understanding and help throughout my research.

Contents

Abstract i

Layaudience iii

Co-Authorship Statement iv

Acknowlegements vi

List of Figures xii

List of Tables xv

List of Abbreviations, Symbols, and Nomenclature xvi

1 Introduction 2

- 1.1 Motivation 2
- 1.2 Contributions 7
- 1.3 Thesis Outline 9

2 Background 11

- 2.1 Recurrent Neural Networks 11
- 2.2 Generative Adversarial Networks 12
- 2.3 Recurrent Batch Normalization 15
- 2.4 Hyperparameter Optimization 16

3 Related Work 19

3.1 Conventional Load Forecasting Models 19
3.2 Generating Synthetic Data 22
3.3 Online Machine Learning 26
3.4 Federated Learning 30
3.5 Asynchronous Federated Learning 32

4 Generating Synthetic Data Streams 34
4.1 Recurrent Generative Adversarial Networks for Generating Energy Data 35
4.1.1 Data Pre-processing 35
Auto Regressive Integrated Moving Average (ARIMA) 37
Fourier Transform 37
Feature Normalization 38
Sliding Window 38
4.1.2 R-GAN 39
4.2 Evaluation Process 41
Train on Synthetic, Test on Real (TSTR) 42
Train on Real, Test on Synthetic (TRTS) 42
Train on Real, Test on Real (TRTR) 43
Train on Synthetic, Test on Synthetic (TSTS) 43
4.3 Evaluation of R-GAN 43
4.3.1 Data sets and Pre-Processing 44
4.3.2 Experiments 45
4.3.3 Results and Discussion - UCI data set 49
4.3.4 Results and Discussion - Building Genome data set 53
4.4 Discussion 55

5 Online Deep Learning 57
5.1 Online Adaptive RNN 58

5.1.1 Prepossessing Module 58

Sliding Window 61

Online Normalization 61

5.1.2 Batch Normalized RNN 63

5.1.3 Buffering Module 64

5.1.4 Tuning Module 65

5.2 Evaluation Methodology 67

5.3 Evaluation of Online Adaptive RNN 70

5.3.1 Dataset and Preliminary Analysis 71

Stationary Analysis 71

Concept Drift Analysis 72

5.3.2 Comparison with offline LSTM 76

Experiments 76

Results 77

5.3.3 Comparison with online models 78

Experiments 78

Results 80

5.3.4 Analysis of modules impact 81

Experiments 81

Results 84

5.3.5 Analysis of the buffer size impact 84

Experiments 84

Results 86

5.3.6 Training time analysis 86

Experiments 88

Results 89

5.4 Discussion 89

6 **Synchronous Federated Learning for Load Forecasting** **92**

6.1 Federated Learning for Load Forecasting 93

6.1.1 Federated Learning Process 94

6.1.2 Local Preprocessing 96

6.1.3 Federated Learning Algorithms 97

6.2 Evaluation 100

6.2.1 Dataset and Evaluation Metrics 100

6.2.2 Comparison of FedAVG and FedSGD with Individual LSTMs and Central Model 102

Forecasting one hour ahead 103

Forecasting 24 hours ahead 106

6.2.3 Evaluation in Dynamic Environment 108

6.2.4 Convergence and Computation Cost 110

6.3 Discussion 115

7 **Asynchronous Adaptive Federated Learning for Load Forecasting** **118**

7.1 FedNorm 119

7.2 Evaluation Methodology 122

7.2.1 Dataset and evaluation Metrics 122

7.2.2 Experimental Setup 124

7.2.3 FL Techniques Included in Comparison 125

7.3 Results and Analysis 126

7.3.1 Predictive Performance Comparison 126

7.3.2 Statistical Significance 127

7.3.3 Analysis of Convergence 130

7.4 Discussion 131

8 **Conclusion and Future Work** **133**

8.1 Conclusion 133
8.2 Future Work 136

Bibliography **139**

Curriculum Vitae **155**

List of Figures

1.1 Standard federated learning 6

2.1 The LSTM cell 12

2.2 Generative Adversarial Network. 13

4.1 Recurrent GAN for Energy Data. 36

4.2 Sliding Window Approach. 39

4.3 R-GAN generator and discriminator. 39

4.4 MH-GAN approach. 42

4.5 ARIMA fitted model (UCI data set). 45

4.6 The results of Fourier transform with 1, 10, 100, and 1000 components (UCI data set). 46

4.7 Generator and discriminator loss for LR=2e-6, 2e-5, and 2e-4 (UCI data set). . 48

4.8 MAPE(%) comparison between TRTS/TSTS and TSTR/TRTR (UCI data set). . 50

4.9 MAE comparison between TRTS/TSTS and TSTR/TRTR (UCI data set). . . . 50

4.10 Two examples of generated data samples compared to real data. 51

4.11 MAPE(%) comparison between TRTS/TSTS and TSTR/TRTR (Building Genome data set). 54

4.12 MAE comparison between TRTS/TSTS and TSTR/TRTR (Building Genome data set). 54

5.1 Online Adaptive RNN: components and the processing flow 59

5.2 Sliding window technique 61

5.3 Online evaluation: a) holdout evaluation b) prequential evaluation. 67

5.4 Prequential-Holdout evaluation. 70

5.5 The electricity load and the concept drift occurrences detected with ADWIN, PH, and DMM: house 1. 74

5.6 The electricity load and the concept drift occurrences detected with ADWIN, PH, and DMM: house 4. 75

5.7 The MAE and MSE errors for offline LSTM and Online Adaptive RNN 79

5.8 Comparison of Online Adaptive RNN with five online regression algorithms: MLP, liner, passive-aggressive, bagging, and KNN regression. 82

5.9 Comparison of error intervals for Online Adaptive RNN and the five online regression algorithms: LP, liner, passive-aggressive, bagging,and KNN regression 83

5.10 The impact of buffering and tuning modules on the forecasting accuracy. 85

5.11 The impact of the buffer size on model's accuracy. 87

5.12 Average training time for different buffer sizes. 87

5.13 Average training time for conventional LSTM with hyperparameter tuning . . . 90

5.14 Average training time for conventional LSTM without hyperparameter tuning . 90

6.1 Federated Learning for Load Forecasting 95

6.2 Electricity load examples for Home 1, Home 10, and Home 15 101

6.3 MAPE errors for FedAVG, FedSGD, LSTMs, and Central Model: one hours ahead prediction 105

6.4 RMSE errors for FedAVG, FedSGD, LSTMs, and Central Model: one hours ahead prediction 105

6.5 Actual versus predicted load for House 13: one hour ahead forecast 106

6.6 MAPE errors for FedAVG, FedSGD, LSTMs, and Central Model: 24 hours ahead forecast 108

6.7 RMSE errors for FedAVG, FedSGD, LSTMs, and Central Model: 24 hours ahead forecast 108

6.8 Actual versus predicted load for House 17: 24 hours ahead forecast 109
6.9 Dynamic environment: MAPE errors for FedAVG, FedSGD, and Central models: one hour ahead forecast 111
6.10 Dynamic environment: RMSE errors for FedAVG, FedSGD, and Central models: one hour ahead forecast 111
6.11 Dynamic environment: MAPE errors for FedAVG, FedSGD, and Central models: 24 hours ahead forecast 112
6.12 Dynamic environment: RMSE errors for FedAVG, FedSGD, and Central models: 24 hours ahead forecast 112
6.13 The training loss for FedAVG and FedSGD for 50 training rounds. 113
6.14 MSE for each FedAVG training round 114
6.15 MSE for each epoch: central model 114
6.16 MSE for each epoch: three examples of individual LSTMs 114

7.1 Asynchronous federated learning. 120
7.2 Electricity load examples for eight houses 123
7.3 Asynchronous settings simulation. 124
7.4 MAPE errors for eight hours ahead prediction. 128
7.5 RMSE errors for eight hours ahead prediction. 128
7.6 MAPE errors for 16 hours ahead prediction. 129
7.7 RMSE errors for 16 hours ahead prediction. 129
7.8 Actual and predicted load: House 3, eight hours ahead forecast 130
7.9 Diebold-Mariano test: DM values for 19 houses. 131
7.10 Errors (median, 25th Percentile, 75th Percentile, maximum, and minimum) over iterations for each of the seven algorithms 132

List of Tables

4.1 TRTS, TRTR, TSTR, and TSTS accuracy for R-GAN (UCI data set) 50
4.2 Kruskal-Wallis H Test and Mann-Whitney U test (UCI data set) 53
4.3 TRTS, TRTR, TSTR, and TSTS accuracy for R-GAN (Building Genome data set) 54
4.4 Kruskal-Wallis H Test and Mann-Whitney U test (Building Genome data set) . 55

5.1 ADF Test 72
5.2 KPSS Test 73

6.1 Average RMSE and MAPE errors for all 19 houses: one hour ahead prediction . 105
6.2 Average RMSE and MAPE errors for all 19 houses: 24 hours ahead forecast . . 107
6.3 Computation Comparison between FedAVG, Central Model, and Individual LSTMs 113
6.4 MAPE and RMSE for FedAVG, Simple Shifting, and Taïk *et al.*: one hour ahead and 24 hours ahead prediction 117

7.1 Average MAPE and RMSE for 8 and 16 hours ahead forecast 126

List of Abbreviations, Symbols, and Nomenclature

AMI	Advanced Metering Infrastructure
ANNs	Artificial Neural Networks
ARIMA	AutoRegressive Integrated Moving Average
BNRNNs	Batch Normalized RNNs
BO	Bayesian optimization
C-RNN-GAN	Continuous recurrent GAN
CNN	Convolutional Neural Network
DL	Deep Learning
EDM	Empirical Mode Decomposition
EEG	electroencephalographic brain signal
EEG-GAN	GAN for generating EEG
ELU	Exponential Linear Unit
ESN	Echo State Neural Network
FedAvg	Federated Averaging
FedSGD	Federated Stochastic Gradient Descent
FL	Federated Learning
FS	Feature Selection
FT	Fourier Transform
LSTM	Long Short-Term Memory
LR	Learning Rate
GA	Genetic Algorithm
GAN	Generative Adversarial Network
GDPR	General Data Protection Regulation
GELU	Gaussian Error Linear Unit
GRU	Gated Recurrent Unit
HC	Hierarchical Clustering

IID	independent and identically distributed
ISCOA	Improved Sine Cosine Optimization Algorithm
MAPE	Mean Absolute Percentage Error
MAE	Mean Absolute Error (MAE)
MLP	Multi-Layer perceptron
ML	Machine Learning
MH-GAN	Metropolis-Hastings GAN
NILM	Nonintrusive Load Monitoring
PA	Passive-Aggressive
R-GAN	Recurrent GAN
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SMOTE	Synthetic Minority Over-sampling TEchnique
S2S	Sequence to Sequence
SBCTL	Similarity-based Chained Transfer Learning
SR	image superresolution (SR)
SRGAN	GAN for image superresolution (SR)
STRSAGA	Stream SAGA
SSVRG	Streaming SVRG
SVRG	Stochastic Variance Reduced Gradient
TPE	Tree-structured Parzen Estimator
TRTR	Train on Real, Test on Real
TRTS	Train on Real, Test Synthetic
TSTR	Train on Synthetic, Test on Real
TSTS	Train on Synthetic, Test on Synthetic
WGAN	Wasserstein GAN
WGAN-GP	Wasserstein GAN with gradient penalty
WGL	Weighted Gradient Learning

Chapter 1

Introduction

This chapter presents thesis motivation and discusses the main research contributions. Finally, the thesis organization is presented.

1.1 Motivation

Energy production is the largest source of greenhouse gas emissions and about two-thirds of global greenhouse gas emissions are the result of burning fossil fuels for energy production [1]. The environmental impact of energy production is expected to become even more exacerbated as it is estimated that world energy consumption will grow by 28% between 2015 and 2040 [2]. At the same time, countries are setting aggressive greenhouse gas emissions targets; European Union, for example aims to reduce emissions for 40% and increase energy efficiency for 27% by year 2030 [1]. Nevertheless, energy demand is continuously increasing making energy management a crucial factor in reducing environmental impact while ensuring that the growing demand is met [3].

Although renewable energy resources are expanding quickly, their intermittent nature and variability pose challenges for balancing supply and demand. In such changing environment, energy management plays a crucial role for ensuring supply-demand balance, reducing environmental impact, and moving towards a smarter grid. Improved energy management also leads to financial benefits for the end consumers in terms of reduced energy costs and associated operating expenses.

Load forecasting is a critical and fundamental part of energy management: accurately estimating future loads assists in maintaining demand-supply balance. Load forecasting helps energy suppliers to reduce the gap between demand and supply by estimating load flows and to make decisions that can prevent overloading. It ensures that consumers have a reliable supply of energy at the most reasonable cost. Moreover, load forecasting provides information for generation scheduling, grid operation, and infrastructure planning. Also, financial benefits from improved load forecasting can be large: for example, a 1% improvement in a six hour ahead wind generation forecast leads to savings of 972 thousand dollars over six months [4].

On the other hand, the expansion of smart meters and other sensors has enabled measuring and recording energy consumption on a large scale. Utility companies have been extensively installing smart meters: in 2016, there were over 70 million smart meters in the USA and over 96 million in China, and the number is continuously growing [5]. This large smart meter data created a backbone for new deeper insights into energy usage patterns and forged new opportunities in hourly load forecasting for individual buildings and even individual homes [6].

In sensor-based forecasting, historical data from smart meters or other sensors are used in combination with meteorological data to infer future energy consumption. Machine learning (ML) techniques have been widely used in a variety of load forecasting and energy prediction tasks [7] as they discover relations within data and identifying patterns. ML-based techniques typically use historical load data collected by smart meters or other sensor to train machine learning models, and then, those models are used to predict future energy consumption. Conventionally, smart meter data are transmitted to a data center or other centralized systems for storage and ML model training. There are two main categories of these centralized solutions: the first category trains a single model for each smart meter and, thus, generates individual forecasts for each meter, while the second category trains a machine learning model with the aggregated data from several smart meters and, hence, provides aggregated load forecasts [8].

The conventional offline models are trained once by repeatedly passing all training data through the model; one pass is referred to as *epoch*. Then, the model is used to infer future loads. This approach is missing out on the information that new data could provide. Of course, the model can be re-trained occasionally using all old data together with new data, but this is very computationally expensive as each time, the model is re-trained from scratch. Ideally, the model would learn from new data as they become available, without the need to re-train or to retain old data.

Although, these centralized solutions have shown great results in load forecasting, they are faced with numerous challenges:

1. **Limited data availability**: ML applications typically require significant quantities of data; however, the data may not be readily available because of challenges such as costs associated with collecting data, privacy and security concerns, or other reasons. Moreover, when it comes to evaluating the online algorithm, the availability of large and diverse data streams is required.

2. **Necessity of online processing**: Predicting energy consumption in near real-time is becoming critical to modern energy management systems. Storing data in the cloud and re-training machine learning models with all historical sensor data is time-consuming and often unfeasible. Consequently, a load forecasting approach capable of continuously learning from new data as they arrive is required.

3. **Presence of Concept Drift**: The data distributions in energy domain change over time, producing what is known as *concept drift* [9]. For example, adding a new appliance or a change in home occupants behaviour patterns will result in different energy consumption profiles. In the presence of concept drift, machine learning models experience weak and degrading predictive performance [10].

4. **Network congestion**: The volume of generated data is increasing exponentially, and

centralized systems require transmitting all data to a centralized location, what leads to increased network traffic [11]. This is true for both types of forecasting, individual and aggregated.

5. **Security and privacy**: A centralized ML not only requires sharing local data with the centralized systems imposing security and privacy concerns, but also makes complying with stringent data regulations such as EU General Data Protection Regulation (GDPR) [12] and The Personal Information Protection and Electronic Documents Act (PIPEDA) [13]. These regulations establish the guidelines for how organizations collect, use, and disclose personal information in the course of their for-profit commercial activities.
6. **Scaling**: A massive amount of smart data have led to a new deeper insights into energy usage patterns as well as large-scale load forecasting at the individual consumer level. However, as the number of smart meters grows, training an individual ML model for meter-level load forecasting become computationally expensive and even infeasible [11].

Although centralized offline machine learning approaches have achieved great results in load forecasting, new techniques are needed to address the discussed challenges. Methods for making machine learning models better suited for learning from sensor data and addressing the aforementioned points are explored under the research themes of ”Online Learning” and ”Federated Learning”.

Online machine learning, sometimes referred to as incremental or continuous learning, referrers to algorithms that learn from data streams by dynamically updating the model as data become available. In online learning, the data can be discarded after they are consumed by the model. In contrast to offline or batch learning techniques, which train the model by learning on the entire training data set at once, in online machine learning, the model is updated as data become available. Moreover, for batch (offline) learning, all data must be available at the start of training while online models learn continuously as new data arrive without the need to store all data.

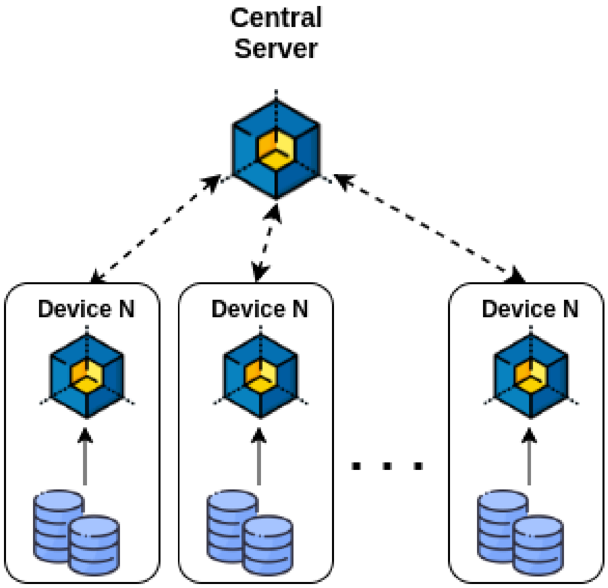


Figure 1.1: Standard federated learning

Federated Learning (FL) presents a possible way to assist with mentioned challenges by decoupling the ability to train the ML model from the need to store the data on the cloud or another centralized system. The fundamental idea of FL is to train local models on local data and periodically exchange learned parameters with the central server to build a global model. Figure 1.1 depicts a FL scheme: each device receives a copy of the global model and improves it by learning from local data. Then, instead of raw data, the updated parameters of the local models are sent to the server to be aggregated and incorporated into the global model.

This thesis investigates and proposes online and federated learning techniques for load forecasting. FL represents a major shift from a costly central analytic system to a distributed ML approach that can exploit numerous distributed computational resources. This will help enterprises to retain data on-site, reduce network traffic, and reduce ML model training time. On the other hand, because online learning does not require re-training with all historical data, it provides a quicker response to changes in data patterns and, thus, is better suited for dynamic data streams. The proposed method also helps with data protection regulations such as GDPR and PIPEDA by keeping personal data on client devices that gives users more direct physical control of their own data while sending model updates to the server.

The approaches proposed in this thesis will allow utility companies to predict energy consumption patterns in near real-time and provide them with an efficient energy management system while respecting end user privacy and building trust. Online and federated learning will enable load forecasting on a large scale without the need to move data to a centralized location. Moreover, the energy domain shares many characteristics with other IoT domains and time-series data; therefore, the online and FL technique developed in this research have the potential to be expanded to benefit other learning tasks in different time-series domains.

1.2 Contributions

The main contributions of this thesis are organized into four main categories.

1. **Recurrent Generative Adversarial Networks (R-GANs)** for generating realistic energy consumption data by learning from real data. R-GANs have the potential to offer a new way of training and evaluating ML methods by generating potentially unbounded data to simulate the data streams for online and offline algorithms. The main contributions of this part of the research are as follows:

    - Propose Recurrent Generative Adversarial Network (R-GAN) to generate synthetic energy data for evaluating ML models.
    - Incorporate Wasserstein GANs (WGANs) and Metropolis-Hastings GAN (MH-GAN) approaches into R-GAN to improve training stability, overcome the mode collapse, and, consequently, generate more realistic data.
    - Evaluate the quality of generated data through comprehensive experiments.
    - Demonstrate the robustness of the proposed R-GAN approach and generate the synthetic data with similar distribution to the original dataset.

2. **Online Adaptive Recurrent Neural Network (RNN)**, an approach for load forecasting capable of continuously learning from newly arriving data and adapting to new patterns.

The main contributions of this part of the research are as follows:

- Propose Online Adaptive RNN, a deep recurrent neural network load forecasting approach capable of continuously learning from new data as they arrive.
- Evaluate the performance of the proposed Online Adaptive RNN through comprehensive studies using the real-world data from residential consumers.
- Propose Prequential-Holdout evaluation method to compare online and offline algorithms.
- Conduct a comprehensive evaluation to compare the proposed method with standard online models.
- Conduct a comprehensive evaluation to compare the proposed method to the offline RNN models.

3. **Federated Learning (FL) for load forecasting** capable of handling statistical heterogeneity in load forecasting data. The proposed FL method encounters statistical heterogeneity challenges as the local nodes correspond to consumers with different energy use patterns and magnitudes. The main contributions of this part of the research are as follows:

    - Adapt Federated Stochastic Gradient Descent (FedSGD) and Federated Averaging (FedAvg) paradigms for energy prediction applications.
    - Propose FL with adaptive learning rate for load forecasting capable of handling statistical heterogeneity.
    - Evaluate the performance of the proposed FL model compared to individual and central models.
    - Conduct case studies to measure the performance of proposed model in dynamic environment where some clients join the federation after the training is complete and only use an already trained model for forecasting.

4. **Asynchronous Adaptive Federated Learning**, an approach for FL learning without need to wait for lagging clients. Here we extend our previous FL work to an asynchronous FL model for load forecasting and proposed FedNorm. In this novel technique, the server aggregates model updates from clients without the need for synchronous processing on clients while still handling statistical heterogeneity. The main contributions of this part of the research are as follows:

   - Propose FedNorm, an approach for load forecasting without the need to wait for lagging clients and without requiring all clients to complete local training in each round.
   - Propose a simulation approach to mimic the asynchronous settings in order to examine and compare the asynchronous algorithms.
   - Conduct comprehensive experiments to compare the proposed algorithm with synchronous and asynchronous methods.
   - Evaluate the performance of the proposed FedNorm through comprehensive studies.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: **Chapter 2** describes the background, which includes Recurrent Neural Networks in Section 2.1, Generative Adversarial Networks in Section 2.2, Recurrent Batch Normalization in Section 2.3, and Hyperparameter Optimization in Section

Chapter 3 discusses the related works. First, Section 3.1 discusses conventional load forecasting models, then Section 3.2 discuss the recent works related to the generating synthetic energy consumption and load forecasting data. Section 3.2 introduces various online machine learning and deep learning algorithms. In Section 3.4 the studies associated with federated learning are investigated and Section 3.5 presents the recent research regarding the asynchronous federated learning field.

Chapter 4 first, in Section 4.1 presents the methodology applied in generating synthetic data streams including pre-processing, feature generation, and the recurrent generative model. Then, in Section 4.2, the evaluation process for measuring the quality of generated data is described, and, in Section 4.3, evaluation results are presented. Finally, results are discussed in Section 4.4.

Chapter 5 first presents the proposed Online Adaptive RNN including pre-processing, batch normalization, buffering, and tuning module in Section 5.1. Next, in Section 5.2, the evaluation methodology is described and two main approaches for evaluating the online algorithms are presented. Consequently, in Section 5.3, the proposed model is evaluated and the results are discussed in section 5.4.

Chapter 2

Background

This chapter introduces Recurrent Neural Networks (RNN), Generative Adversarial Networks (GAN), Recurrent Batch Normalization, and Hyperparameter Optimization. These algorithms were applied in various aspects of our research and are considered the foundations of our work.

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed for sequential data such as those found in language translation or load forecasting [14]. The recurrent connection to the same neurons in the previous time step together with their internal state (memory) make RNNs well suited for modeling temporal behavior. RNNs are mainly trained using backpropagation through time; however, for large sequences, this can lead to the vanishing gradient problem causing the RNN to forget older information. The Long Short Term Memory (LSTM) networks were designed to overcome this problem, and, consequently, they are able to maintain information for longer periods and make better predictions [15].

As illustrated in Fig. 2.1, the LSTM cell contains a cell state c, a hidden state h, an update step g, and three gates: input i, forget f, and output o. LSTM computation at time t is given as follows:

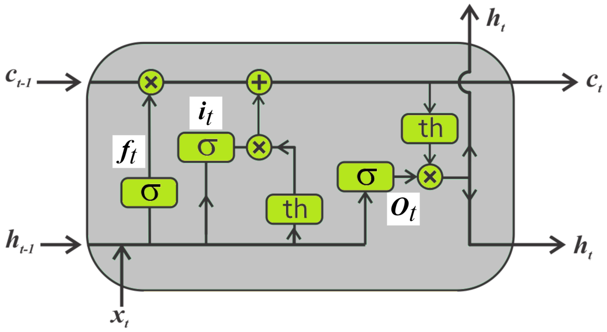


Figure 2.1: The LSTM cell

$$i_t = \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \tag{2.1a}$$

$$f_t = \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \tag{2.1b}$$

$$o_t = (W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \tag{2.1c}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}) \tag{2.1d}$$

$$h_t = o_t \odot \tanh(c_{t-1}) \tag{2.1e}$$

Here, σ is the sigmoid activation function, tanh is the hyperbolic tanh activation function, and $\odot$ represents elementwise multiplication. The W_x's and W_h's are the input-hidden and hidden-hidden weights, respectively, and b_x's and b_h's are the corresponding biases. The LSTM cells are responsible for learning dependencies among the elements in the input sequence. The gates within the cell control how data flow through the cell and regulate which data should be memorized and which can be forgotten.

2.2 Generative Adversarial Networks

In machine learning, generative modelling is an unsupervised learning task that includes automatically detecting and learning patterns in input data so that the model can be used to produce new examples that could have been drawn from the original dataset.

Generative Adversarial Networks (GANs) are a type of generative models that [16] discover and learn patterns present in the training data and are thus capable of generating new data with similar characteristics. As illustrated in Fig. 2.2, a GAN consists of two networks: a generator and a discriminator. The task of the generator is to mimic data samples (e.g. images, video, audio) with the objective of fooling the discriminator into thinking that the generated samples are real. The discriminator learns to recognize whether a sample is real or synthetic. The two try to defeat each other: the generator tries to fool the discriminator into thinking that the produced samples are real while the discriminator tries to identify the fake samples.

The generator produces fake data from the random input as illustrated in Fig. 2.2. The discriminator receives two inputs, the real data and the fake data generated by the generator, and produces an output indicating probabilities that samples are real. The two play min-max game with value function $V(D, G)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[log D(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \tag{2.2}$$

where G and D are the generator and discriminator, x is the real data sample drawn from $p_{data}(x)$, and z is the random (noise) input drawn from $p_z(z)$.

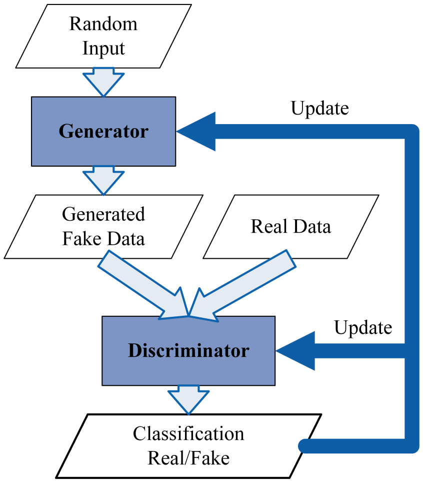


Figure 2.2: Generative Adversarial Network.

GANs have mostly been used in computer vision for tasks such as generating realistic-looking images, text to image synthesis [17], image completion, and resolution up-scaling [18]. If GANs could generate realistic energy data, they could be used to generate data for ML training and for imputing missing values.

Studies have demonstrated that GANs can provide good samples [19, 20, 21]; however, their training may experience diverging or oscillatory behavior resulting in failure to converge [22]. GANs are also prone to *mode collapse* and *vanishing gradients*. Mode collapse refers to a situation in which the generator collapses and always produces the same outputs whereas vanishing gradients indicate a situation in which the generator updates become so small that the overall GAN fails to learn [22].

Wasserstein GANs (WGANs) [23] have been proposed to deal with non-convergence and vanishing gradient problems. The cost function in original GANs has a large variance of gradients what makes the model unstable. WGANs improve stability of the training process by using a new cost function, Wasserstein distance, which has smoother gradients everywhere. The Wasserstein distance calculation can be simplified using Kantorovich-Rubinstein duality to:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)] \tag{2.3}$$

where p_r and p_g are probability distributions, *sup* is the least upper bound, and f is a **K-Lipschitz** function following the following constraint:

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{2.4}$$

Wasserstein distance in WGAN, uses 1-Lipschitz functions, therefore, the $K = 1$.

Metropolis-Hastings GAN (MH-GAN) [24] aims to improve standard GANs by adding aspects of Markov Chain Monte Carlo. In standard GANs, the discriminator is used solely to train the generator and is discarded upon the end of training. In MH-GAN, the discriminator from GAN training creates a wrapper around the generator. The generator produces a set of samples, and the discriminator chooses the sample closest to the real data distribution. This way, the discriminator contributes to the quality of the generated samples.

2.3 Recurrent Batch Normalization

Normalizing the input data for deep neural networks helps the models converge faster. However, this only impacts the input to the first layer while all other layers receive inputs from the previous layers. The distribution of network activations changes during training due to the changes in network parameters. These changes alter the distribution of inputs to the inner layers and slow down the training. This problem is known as *internal covariate shift* [25]. Batch normalization is a mechanism in mini-batch training, which aims to normalize the inputs to inner layers in order to fight the covariate shift problem [25].

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

$$\bar{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.5}$$

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

Next, the input is scaled and shifted as follows:

$$y_i = \gamma \bar{x}_i + \beta \tag{2.6}$$

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

Cooijmans *et al.* [26] proposed a reparameterization of LSTM that brings the benefits of batch normalization to the recurrent neural networks. They demonstrated that the Batch-Normalized RNNs (BNRNNs) lead to faster convergence and improved generalization. The batch-normalizing transform BN($\cdot$,γ,β) is introduced into the LSTM as follows:

$$\begin{pmatrix} f_t \\ i_t \\ o_t \\ g_t \end{pmatrix} = BN(W_h h_{t-1}, \gamma_h, \beta_h) + BN(W_x x_t, \gamma_h, \beta_h) \tag{2.7}$$

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t) \tag{2.8}$$

$$h_t = \sigma(o_t) \odot tanh(BN(c_t, \gamma_c, \beta_c)) \tag{2.9}$$

where BN($\cdot$) is the transformation introduced with equations (2.5) and (2.6). The recurrent $W_h h_{t-1}$ and the input $W_x x_t$ terms are normalized separately. To preserve LSTM dynamics, the normalization is not applied to the cell update c_t. During training, the mean and the variance are calculated independently for each batch, and at test time, the average of the estimates over the training set is used.

2.4 Hyperparameter Optimization

Hyperparameter optimization aims to find a combination of hyperparameters that leads to the optimal ML model performance. It has been shown that tuning hyperparameters plays a major role in the ML model accuracy [27]. In general, the hyperparameter optimization can be represented in an equation form as:

$$x_{best} = argmin[f(x)]|x \in X \tag{2.10}$$

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

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

In contrast to those approaches, Bayesian optimization [28] (BO) keeps track of the past evaluations to form a probabilistic model for mapping hyperparameters to a probability P of an objective function score:

$$P(score|hyperparameters) \tag{2.11}$$

This probability model is referred to as the *"surrogate"* for the objective function. The surrogate is easier to optimize than the actual objective function; thus, BO searches for hyperparameters using the surrogate. BO process is described in Algorithm 1. First, the probabilistic model P_{model} is initialized with a Gaussian process prior on $f(x)$ [28]. Then, in each iteration, the best set x_{best} is found for the current probabilistic model P_{model}, the model $score$ for that set x_{best} is determined, and the P_{model} is updated.

Similar to the described Gaussian process-based optimization, the Tree-structured Parzen Estimator (TPE) also constructs models to approximate the performance of hyperparameters based on historical measurements, and then subsequently chooses the new hyperparameters to test based on this approximate model [28]. However, while Gaussian approach estimates the

Algorithm 1 Bayesian Hyperparameter Optimization

```
$Pmodel \leftarrow$ Surrogate($f(x)$)
$x_{best} \leftarrow \{\}$
while $i < maxIterations$ do
    $x_{best} \leftarrow Pmodel(score, hyperparameters)$
    $score \leftarrow f(x_{best})$
    $Pmodel \leftarrow Update(Pmodel, score)$
```

probabilities directly, TPE estimates them indirectly. Since TPE achieved better accuracy than the Gaussian process-based BO, TPE is used in our study.

Chapter 3

Related Work

This chapter begins with an overview of traditional energy forecasting methods. Next, various strategies for generating synthetic data and methods for evaluating the quality of the generated data are examined. Following that, online machine learning approaches are presented and, finally, various synchronous and asynchronous federated learning techniques are discussed.

3.1 Conventional Load Forecasting Models

Many approaches have been introduced for energy forecasting problems (e.g., physics, statistics, and machine learning-based) [29] but this section focuses on ML-based models as our work belongs to this category.

Alobaidi *et al.* [30] proposed an ensemble-based framework to predict day ahead average household consumption. The framework employs Artificial Neural Networks (ANNs) as the base learners and combines them using multiple linear regression. Their results showed improvement in the generalization ability compared to stand-alone ANN and ANN-based bagging ensemble. Singh *et al.* [31] introduced a hybrid ARIMA-ANN technique for wind power forecasting. The hybrid approach achieved better accuracy than the two models, ARIMA and ANN, working separately. Grolinger *et al.* [32] introduced an approach based on Support Vector Regression (SVR) and local learning for load forecasting with big data. Their method partitions the training set through clustering and then trains a separate SVM model for each cluster. Local learning with SVR improved the load forecasting accuracy while achieving order or magnitude shorter training time than the stand alone SVR.

Deep learning algorithms have been popular in load forecasting because of their ability to capture complex patterns in data; specifically, recurrent neural network architectures have been frequently used as they can capture temporal dependencies [33, 15]. Gao *et al.* [34] proposed a short-term electricity load forecasting based on an Empirical Mode Decomposition Gated Recurrent Unit with Feature Selection (EMD-GRU-FS). The original series is decomposed into sub-series with empirical mode decomposition and the correlation between sub-series and the original series is determined with the Pearson correlation. Finally, the GRU network is trained with the original series and the sub-series with a high correlation. Bouktif *et al.* [35] paired standard Long-Short Term Memory (LSTM) with a Genetic Algorithm (GA) for short to medium term aggregate load forecasting. In their approach, LSTM carries out the forecasting while GA is responsible for finding the optimal time lags and the number of layers for LSTM.

Han *et al.* [36] proposed a prediction model that combines copula function and LSTM network for the estimation of mid-to-long term wind and photovoltaic power generation. First, the copula function is used to extract the key meteorological factors that affect wind and photovoltaic power generation. Then, joint prediction models of wind and photovoltaic power generation based on LSTM performed the forecasting. Long Short-Term Memory (LSTM), a type of RNN, was combined with Support Vector Machine, Random Forest, and a data preprocessing technique to create a multi-step forecasting strategy capable of improving forecasting accuracy [37].

Sehovac *et al.* [15] proposed Sequence to Sequence Recurrent Neural Network (S2S RNN) with attention for electrical load forecasting. Their approach adapts the sequence to sequence architecture from language translation to improve time modeling by combining two RNNs: encoder and decoder. The attention mechanism eases the connection between the encoder and the decoder.

Tian *et al.* [38] also used S2S RNN, but they focused on scaling load forecasting for a large number of smart meter and proposed Similarity-based Chained Transfer Learning (SBCTL). In SBCTL, the model for the first meter is trained in a traditional manner while all other models employ transfer learning to take advantage of the already trained models according to similarities between energy consumption patterns in smart meters data.

Fan *et al.* [39] examined various deep recurrent neural network strategies for short-term load forecasting. Their results confirmed that RNNs are well suited for short-term forecasting and demonstrated that LSTM cells perform better than vanilla RNNs. Somu *et al.* [40] also proposed a solution based on LSTM: they improved the forecasting accuracy by tuning LSTM hyperparameters with Improved Sine Cosine Optimization Algorithm (ISCOA).

Zainab *et al.* [41] proposed a framework for training in parallel individual models on smart meter data to reduce training time. A parallel pre-processing stage is performed on individual smart meter data sets, and then a single model is trained for each dataset concurrently. The framework employed a variety of ML algorithms as the base learners in order to examine trade-offs between model accuracy and execution time.

Although the reviewed models have shown great results in load forecasting, they are all offline approaches: they are trained with a static data set and, to learn from new data, they need to be re-trained. However, the energy consumption data are arriving continuously and new data may have different patterns. To acquire knowledge from new data without re-training, our study proposes an online approach where the model adapts itself to newly arriving data. Moreover, reviewed ML techniques may require data transfer to the cloud or another centralized location for model training. This not only exposes data to privacy and security risks but also, with a large number of smart meters, increases network traffic. We proposed federated learning which has a potential to alleviate mentioned concerns by training a single ML model in a distributed manner without requiring participants to share their data.

3.2 Generating Synthetic Data

ML applications typically require significant quantities of data; however, the data may not be readily available because of challenges and costs associated with collecting data, privacy and security concerns, or other reasons. When it comes to evaluating the online algorithm, the availability of large and diverse data streams is required. Recently, synthetic data has gained popularity since it promises to fulfill the need for large quantities of data without infrastructure or privacy concerns associated with real data. Although different stochastic processes have been introduced to generate data [42], an approach is needed for generating data streams with characteristics and diversity of electricity load data.

Amasyali and El-Gohary [42] surveyed energy forecasting studies: 67% of reviewed studies used real data, 19% simulation data, and 14% public benchmark data. This dominance of real data sets demonstrated the importance of historical data and illustrated the need for new and/or larger energy forecasting data sets. While some of the real data used for forecasting are pubic [38], a number of studies deal with private [43] real-world data sets.

In the same work [42], Amasyali and El-Gohary point out that physical models including simulation software such as EnergyPlus, eQuest, and Ecotect, calculate building energy consumption based on detailed building and environmental parameters, but such detailed information may not always be available. On the other hand, data driven approaches do not require such details about the simulated building, but they require sensors for data collection. From their survey [42], it can be observed that simulation is often used in the design stage while data driven approaches are common in demand and supply management. Physics and data driven approaches do not replace each other; they are applied in different scenarios based on their advantages and disadvantages.

Deb *et al.* [44] reviewed time series techniques for building energy consumption. They observed that simulation and energy modeling software such as EnergyPlus, IES, eQuest, and Ecotect have been greatly successful in energy modeling of new buildings. For energy forecasting, when historical data is not available, computer simulations are very effective [44].

However, many factors govern energy consumption including physical properties of building materials, weather, and occupants' behaviors. Due to complexity of those factors and their interactions, accurate energy forecasting through simulations is challenging [44]. Deb *et al.* [44] noted that for existing buildings, when historical data are available, data driven techniques such as those based on machine learning and statistics have been more accurate than simulations.

Lazos *et al.* [45] investigated approaches for incorporating weather conditions into energy forecasting. The three categories of forecasting approaches were considered: statistics, machine learning, and physics(numerical)-based models. While physics models do not require historical data and are able to provide explanations for the forecasts, they are often highly complex and require extensive details about structural characteristics, thermodynamic, operating schedule, and other properties. Lazos *et al.* [45] also noted that it is challenging to model occupants' behaviors in physics-based systems. In contrast, data driven approaches do not require such details about the buildings and are capable of capturing some of occupants' behavior from historical data, but require significant amounts of data.

According to the reviewed studies [42, 44, 45], simulation can be used for energy forecasting, but machine learning models trained with historical data are much more common. Moreover, synthetic building energy consumption data is especially hard to generate because, in addition to the building properties, use type, and operating schedule, energy consumption also depends on human behavior. Pillai *et al.* [46] proposed an approach for generating synthetic load profiles using available load and weather data. While Pillai *et al.* aim to create benchmark profiles, our work generates data for ML. Ngoko *et al.* [47] generated solar radiation data using Markov models; in contrast, our study is concerned with generating data for energy forecasting.

In recent years, Generative Adversarial Networks (GANs) have seen great progress and received increased attention because of their ability to learn high-dimensional distributions. Originally, GANs were designed for images, and today, the most common use of GAN is in tasks that involve images. For example, works of Mao *et al.* [19], Denton *et al.* [20], and

Karras *et al.* [21] deal with generating high quality images: Mao *et al.* [19] focus on vanishing gradient problem, Denton *et al.* [20] propose the Laplacian pyramid of adversarial networks for generating high-resolution images, and Karras *et al.* [21] improve the human-face generation by using style-based generator. StackGAN [17] synthesizes high-quality images from text descriptions by using a two-stage process: first, a low resolution image is created from text descriptions, and then this image with text goes through a refinement process to create photo-realistic details. SRGAN, a GAN for image superresolution (SR) [48] is capable of inferring photo-realistic images with 4× upscaling factors. Zhang *et al.* [49] used GANs for image de-raining (removing rain from images), consequently improving the visual quality and making images better suited for vision systems.

While the reviewed works [17, 19, 20, 21, 48, 49] deal with images, our study is concerned with time series data in the energy domain. The possibility to apply GANs with sequential and time-series data has been recognised, and GAN-based approaches have been proposed in different domains and for different tasks. Continuous Recurrent Neural Networks with adversarial training (C-RNN-GAN) [50] have been used for music generation and SeqGAN [51] for text generation. While those works [50, 51] deal with discrete sequence generation, the majority of data in the energy domain is real-valued and thus our work is concerned with real-valued data.

Esteban *et al.* proposed a Recurrent GAN [52] for generating realistic real-valued multi-dimensional time series data focusing on the application to medical data. Similar to our study, the work of Esteban *et al.* also uses recurrent neural networks for both the generator and discriminator. In contrast to the work of Esteban *et al.* which uses a GAN similar to the original Goodfellow *et al.* GAN [16], our approach applies Wasserstein GAN. Moreover, we take advantage of Metropolis-Hastings GAN (MH-GAN) [24] approach for generating new data. The same work [52] proposed "Train on Synthetic, Test on Real" (TSTR) approach for evaluating GANs. TSTR is applicable when a supervised task can be defined over the training data: the basic idea is to use synthetic data generated by GAN to train the model and then test the model on a held-out set from the real data. Our study also employs TSTR as a part of the evaluation process.

EEG-GAN generates electroencephalographic (EEG) brain signal using Wasserstein GANs with gradient penalty (WGAN-GP) [53]. While EEG-GAN employs a Convolutional Neural Network (CNN), our work replaces CNN with RNN because of its ability to model temporal dependencies.

To deal with learning from imbalanced datasets, Douzas and Bacao [54] applied GANs to generating artificial data for the minority classes. Their experiments show an improvement in the quality of data when GANs are used in place of oversampling algorithms such as SMOTE. MISGAN, a GAN-based framework for learning from complex, high-dimensional incomplete data [55], was proposed for the task of imputing missing data. VIGAN [56] also deals with imputation, but in this case with scenarios when certain samples miss an entire view of data. While Douzas and Bacao [54], Li *et al.* [55], and Shang *et al.* [56] generate specific part of data (imbalanced classes or missing data), our work deals with generating new energy data samples for training ML models.

Another category of works important to discuss here consists of those related to the GAN evaluation. With image GANs, evaluation often involves visual inspection of the generated sample; however, this is more difficult to do with time-series data. Inception score (IS) has been proposed for evaluating image generative models [57]; IS uses Inception network pre-trained on ImageNet data set to calculate the score. Nevertheless, not only is IS not applicable for non-image data, it is also incapable of detecting overfitting [58].

Sajjadi *et al.* [59] proposed precision and recall for distributions. Precision measures how much of distribution Q can be generated by a part of reference distribution P while recall measures how much of P can be generated by a part of Q. By using a two-dimensional score, they are able to distinguish between the quality of generated images from the coverage of the reference distribution. Still, their study [59] was primarily concerned with images.

Theis *et al.* [60] compared different evaluation approaches and concluded that different metrics have different trade-offs and favour different models. They highlight the importance of matching training and evaluation to the target application. As the main objective of our work is to generate electricity data suitable for training machine learning models, the evaluation is performed in the context of the application by assessing the suitability of the generated data for ML task. Specifically, this is done by training the prediction model on the generated data and testing it on the real data.

3.3 Online Machine Learning

In recent years, Deep Learning (DL) approaches have demonstrated great success in many applications including load forecasting. They can learn feature representations, have strong generalization capabilities, and are able to model complex relationships present in data.

However, several challenges remain open regarding DL techniques for load forecasting. The conventional offline models are trained once by repeatedly passing all training data through the model; one pass is referred to as an *epoch*. Then, the model is used to infer future loads. This approach is missing out on the information that new data could provide. Of course, the model can be re-trained occasionally using all old data together with new data, but this is very computationally expensive as each time, the model is re-trained from scratch. Ideally, the model would learn from new data as they become available, without the need to re-train or to retain old data. Besides, the data distributions in the energy domain change over time, producing what is known as *concept drift* [9]. For example, installing new appliances or changing behavioral patterns in a house will result in different energy consumption profiles. Depending on the factors affecting the data, concept drift can occur within a short time (abrupt drift), gradually over a period of time (gradual drift), steady and slow change in the data distributions (incremental drift), or reoccur after some time (recurrent drift) [61]. In the presence of concept drift, conventional machine learning models experience weak and degrading predictive performance [62]. Online learning has the potential to address these challenges as online models learn from data streams by updating the model as data become available. A few studies have been conducted for energy consumption forecasting using online methods.

Online machine learning approaches can be classified into three main categories: optimization-based, model-based, and hybrid approaches. Optimization-based approaches are various extensions of the gradient descent algorithm to enable updates of the model's parameters as new data become available without changing the model architecture. Defazio *et al.* [63] introduced SAGA, an incremental gradient-based optimization approach with fast linear convergence rates for non-strongly convex problems. Johnson and Zhang [64] proposed stochastic variance reduced gradient (SVRG), a variance reduction method for Stochastic Gradient Descent (SGD). At each step, SVRG keeps a version of estimated parameters and the average gradient and then uses those values in the update rule to reduce the variance of SGD. This approach achieves fast convergence rates for smooth and strongly convex functions. Stream SAGA (STRSAGA) [65] and Streaming SVRG (SSVRG) [66] are extensions of SAGA and SVRG with improved performance.

The model-based approaches modify the model architecture and/or change the number of parameters to achieve a gradually updated model with faster convergence. Sánchez-Medina *et al.* [67] proposed adaptive incremental linear regression for wind forecasting. The model learns gradually as new observations arrive and, when concept drift is detected, the older observations are removed from the model. As this approach is grounded on linear regression, this approach is not suited for non-linear problems such as load forecasting. Vexler *et al.* [68] devised a real-time architecture for energy consumption forecasting by combining LSTM and online density estimation with Hoeffding trees. Liang *et al.* [69] presented an LSTM-based approach for energy forecasting in the smart grid with the model located at the network edge. As new data arrive, the model is continually trained on the small subsets of arriving data reducing computation and training time. Spiral RNN [70] is an RNN architecture that combines a trainable hidden recurrent layer with the Echo State Neural Network (ESN) for online learning. In experiments, Spiral RNN demonstrated stable performance and fast convergence.

The hybrid models combine techniques from optimization and model-based solutions; they benefit from continuous parameter updates, a modified architecture, and various pre- and post-processing techniques, which leads to a simpler model and faster convergence. Guo *et al.* [71] presented Weighted Gradient Learning (WG-Learning) for RNNs to learn from online time series in the presence of anomalies and change points. The local properties of the newly available time series data are exploited to weight the gradients. Madireddy *et al.* [72] proposed a hybrid model combining the two components: an online Bayesian change point detection method to detect the location of the concept drift and a moment-matching transformation technique to convert the data collected before the concept drift to be useful for re-training after the concept drift. Fields *et al.* [73] investigated the sensitivity of various neural networks to concept drift: flavors of RNN, LSTM, and GRU, were less sensitive than the other types of NNs. Ceci *et al.* [74] combined the online adaptive training, entropy-based error measure, and spatial autocorrelation for wind power generation forecasting.

Reviewed optimization-based approaches are well suited for smooth and convex problems. However, neural network training is a non-convex optimization problem, and the optimizer can get stuck in a spurious local optimum, especially when dealing with complex models such as those common in RNNs [75]. Additionally, these models are not designed to handle concept drift [65].

Model-based approaches also do not consider concept drift except for adaptive incremental linear regression [67] which can handle concept drift but is suitable only for linear problems. As non-linearities are present in energy consumption data, adaptive incremental linear regression is not suitable for load forecasting. In contrast, our solution is non-linear and can handle concept drift.

Reviewed approaches from the hybrid category deal with concept drift, but most are in very different domains such as application performance modeling [72] and network traffic [71]. In contrast, our study specifically considers residential load forecasting for individual households which is a difficult forecasting problem due to high load variability and frequent concept drift. Also, in research studies, concept drift is often simulated [73] while we use real-world data.

Online learning approaches proposed by Vexler *et al.* [68] and Liang *et al.* [69] focus on load forecasting. They handle concept drift by transforming the data before feeding the model while our work adjusts the model by tuning the model's parameters as needed. Our online model tuning increases accuracy and improves concept drift handling. The work of Ceci *et al.* [74] is also in the energy domain: to achieve the model updates, they fully re-train the neural network at the end of each day using all historical data. In contrast, our work updates the model as new data arrive without re-training and without the need to retain historical data.

It is also important to mention the five common online algorithms:

1. Multi-Layer perceptron (MLP) Regressor [76] learns by incrementally fitting the MLP on batches of samples with SGD as the optimizer.
2. Online linear regression [77] updates the regression weights with SGD in each learning step.
3. Online Passive-Aggressive (PA) algorithms [78] are a family of online margin-based algorithms: similar to SVR, they aim to maximize the margin. If arriving data are from the same distribution, the algorithm will keep learning, but if the data distribution changes, the weights will slowly forget the previous distribution and learn the new one.
4. The online version of bagging [79] process each data point as it arrives without a need to store it or reprocess while maintaining the current state of the model.
5. The online KNN for regression is a combination of a conventional KNN regression algorithm and the weighted sliding windows.

These algorithms vary in complexity, ability to capture non-linear relations, as well as capability to handle time dependencies. As will be shown in the evaluation, our Online Adaptive RNN outperforms those five algorithms.

3.4 Federated Learning

The reviewed conventional and online load forecasting works achieved good accuracy in load forecasting studies; however, they all train an individual ML model for each smart meter or a group of meters which becomes computationally very expensive when the number of smart meters grows. Also, these techniques train the ML models on a centralized system and, thus, require transferring all data from all smart meters to that server, which results in increased network traffic and latencies. These centralized solutions are also associated with security and privacy vulnerabilities due to sharing and transmitting data from smart meters to the central server.

FL is a burgeoning learning paradigm that has shown promising potential in various fields such as health care [80, 81], autonomous vehicles [82], text analysis [83], and image processing [84], to name but a few [85]. Leroy *et al.* [86] proposed an approach based on federated learning for training speech-based models on mobile devices. They used an adaptive averaging strategy in place of weighted averaging to reduce the number of communication rounds required to reach the target performance. This method achieved competitive accuracy compared to the centralized approaches.

Liu *et al.* [87] introduced FedVision, a visual object detection platform enabled by Convolutional Neural Network (CNN) and FL for training a shared model through a collaboration of multiple clients. Since parameters from diverse local models can have different contributions towards the shared model performance, FedVision applies a compression technique to prune less useful weights while preserving model performance. This results in reduced neural network size and faster transmission.

To enable training personalized ML models with health information without compromising privacy, Yiqiang *et al.* [88] proposed FedHealth. Knowledge from data is aggregated though FL and, then, the local models are personalized by transfer learning. FedHealth adopts the FL paradigm [89] to aggregate the local models; however, rather than just replacing the local model with the aggregated one, transfer learning is used to personalize the global model for each user.

Briggs *et al.* [90] combined FL with hierarchical clustering (FL+HC). The FL+HC training process consists of three steps. First, a typical FL model is trained with the clients local data for a few training rounds. Next, a hierarchical clustering algorithm is employed to iteratively compare and merge the clients with similar weights (grouping similar clients). Once clusters are determined, each cluster trains its specialized model independently.

In the presence of highly skewed non-IID (independent and identically distributed) data, the accuracy of FL can reduce significantly. To address this problem, Zhao *et al.* [91] suggested sharing a small subset of data among all nodes. Although this method has shown the accuracy increase, sharing data introduces security and privacy concerns. Agnostic Federated Learning proposed by Mohri *et al.* [92] updates the shared model using a weighted average of the clients' gradients and a new optimization method. Their study shows that the proposed approach contributes to fairness and reduces bias. Smith *et al.* introduced MOCHA [93], a federated paradigm that combines multi-task learning [94] with FL. Each node in FL may observe data with a distinct distribution, so it is intuitive to fit a separate model for each local node; however, these separate models have relationships and exhibit similarities. MOCHA applies multi-task learning techniques to fit separate weight vectors for each node. Yeongwoo *et al.* [95] introduced dynamic clustering into the FL framework. On the server side, the framework employs ClusterGAN and HypCluster to dynamically group clients into the clusters according to their loss. The aggregation is carried out individually for each cluster, and the updated parameters are sent to clients that are part of the same cluster.

The reviewed FL works [86, 87, 88, 89, 90, 92, 93] propose FL techniques and address challenges in diverse domains; however, they have not been applied for load forecasting, and their capabilities for load forecasting need to be investigated. Diversity of distributions and load patterns observed by individual smart meters imposes challenges and may degrade the accuracy of the single global model created through FL. To address these issues, Taïk *et al.* [96] examined the use of federated averaging for short-term load forecasting. The proposed method employs federated averaging architecture with the weighted averaging as the aggregation technique and LSTM as the global model. Their initial results show that FL is a promising approach for an hour ahead forecasting. Similarly, Li *et al.* [97] also took advantage of the FL with the weighted averaging; however, their study focused on the security aspect rather than the forecasting accuracy, and, as a result, the model's time complexity was high as it included encryption and decryption time. Yuris *et al.* [98] proposed a hybrid of FL and clustering to predict electric vehicle charging station demand on the power grid. The charging stations are first clustered based on their location and then federated averaging approach is applied individually on each cluster. FL with clustering achieved the forecasting accuracy very similar to standalone FL. Zhang *et al.* [99] proposed a probabilistic model for solar irradiation forecasting based on deep learning, variational Bayesian inference, and FL. Although the results showed competitive performance, the main drawback is that the model does not support non-IID data.

Reviewed studies on FL for load forecasting [96, 97, 98] present initial attempts to introduce FL in this domain; however, there is a need for deeper understanding of FL capabilities and limitations. Our study investigates FL with smart meter data, but does not assume clients similarity, and examines FL in presence of clients with different data distributions. Moreover, our work compares two FL techniques, FedSGD and FedAVG, and examines the behaviour of the FL system when some clients join the federation upon the completion of training.

3.5 Asynchronous Federated Learning

One of the main constraints of the standard FL models is that they require high device availability. Synchronous FL requires that all clients selected for training in a specific training round be available and the server waits for updates from all selected clients before the aggregation. Nevertheless, in IoT systems, it is common for sensors to be disconnected for a period of time. This restriction has a considerable effect on the learning process as it can degrade the prediction capacity of the FL model. Also, this highlights the need to develop asynchronous learning approaches capable of handling node disconnects from the FL network.

FL approaches discussed in Section 3.4 have achieved good results in various application and some are even well suited for non-IID data [92, 93, 95, 100]; however, they are synchronously trained and, thus, unfit for heterogeneous devices and unreliable networks. Chen *et al.* [101] proposed an asynchronous online FL framework (ASO-Fed), in which clients perform online learning from local streaming data. ASO-Fed uses a feature Representation learning method on the server to extract cross-device attributes from the clients' updates. Fleet [102] is another online FL framework: it introduced AdaSGD, an asynchronous learning algorithm robust to stale updates (updates that are arriving with delay). For calculating similarities among clients, some labeled data must be shared with the server what creates potentials for privacy leakage. Additionally, AdaSGD has a limited capacity for dealing with non-IID data.

FedAsync [103], an asynchronous FL algorithm, applies regularization on the local clients and uses a weighted average to update the global model. While FedAsync demonstrated some staleness tolerance, for large staleness, FedAsync performs similarly to synchronous FL. Similarly, federated adaptive weighting [104] aims to improve the FL performance in presence of non-IID data through assigning different weights for the participating nodes when updating the global model.

Asynchronous FL works [101, 102, 103, 104] introduced different techniques; however, asynchronous FL is an emerging topic and needs to be further investigated in diverse settings. To the best of our knowledge, our proposed method is the first asynchronous FL algorithm for load forecasting.

Chapter 4

Generating Synthetic Data Streams

The data in the energy sector have a potential to provide insights, support decision making, increase grid flexibility and reliability [105]. Machine Learning (ML) has been used for various smart grid tasks because of its ability to learn from data, detect patterns, provide data-driven insights and predictions. Examples include short and long term demand forecasting for aggregated and individual loads [38], anomaly detection [106, 107], energy disaggregation [108], state estimation, generation forecasting encompassing solar and wind, load classification, and intrusion detection.

All ML applications are dependent on the availability of sufficient historical data. This is especially heightened in the case of complex models such as those found in deep learning (DL) as large data are required for training DL models. Although a few anonymized data sets have been made publicly available [109, 110], many power companies are hesitant to release their smart meter and other energy data due to privacy and security concerns [6]. Moreover, risks and privacy concerns have been identified as key issues throughout data driven energy management [105].

This chapter proposes Recurrent Generative Adversarial Networks (R-GAN), an approach for generating synthetic energy data streams based on GANs. The methodology consists of a pre-processing system, feature generation, and the recurrent generative model. The evaluation process for measuring the quality of generated data is presented and the performance of the proposed R-GAN is examined.

4.1 Recurrent Generative Adversarial Networks for Generating Energy Data

This section proposes a Recurrent Generative Adversarial Network (R-GAN) for generating realistic energy consumption data by learning from real data samples. The focus is on generating data for training ML models in the energy domain, but R-GAN can be applied for other tasks and with different time-series data. Both the generator and discriminator are Stacked Recurrent Neural Networks (RNNs) to enable capturing time dependencies present in energy consumption data. R-GAN takes advantage of Wasserstein GANs (WGANs) [23] and Metropolis-Hastings GAN (MH-GAN) [24] to improve training stability, overcome the mode collapse, and, consequently, generate more realistic data.

Fig. 4.1 depicts the overview of the proposed R-GAN approach. As R-GANs use RNNs for the generator and discriminator, data first needs to be pre-processed to accommodate the RNN architecture. Accordingly, next section first describes data pre-processing consisting of feature engineering (ARIMA and Fourier transform), normalization, and sample generation (sliding window). Then, R-GAN is described and the evaluation process is introduced.

4.1.1 Data Pre-processing

In this research, the term *core features* refers to energy consumption features and any other features present in the original data set (e.g. reading hour, weekend/weekday). In the data pre-processing step, these core features are first enriched through feature engineering using Auto Regressive Integrated Moving Average (ARIMA) and Fourier Transform (FT). Next, all features are normalized and the sliding window technique is applied to generate samples for GAN training.

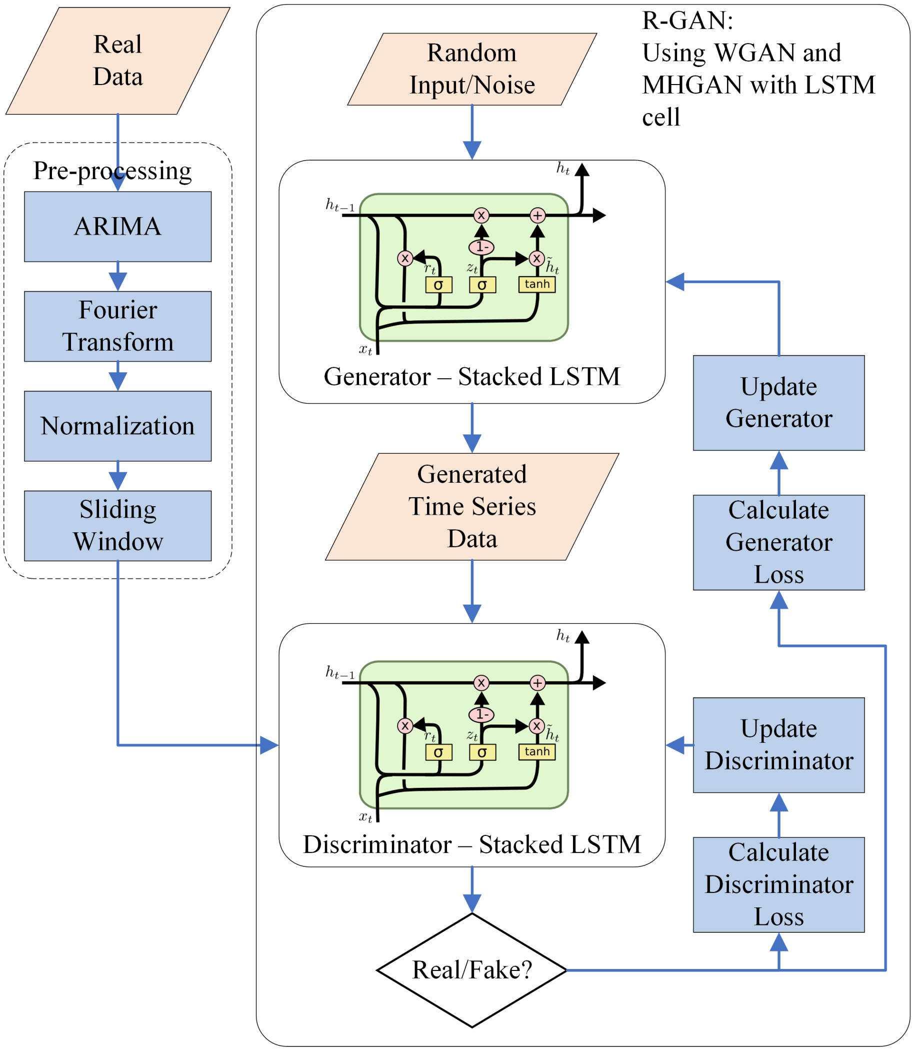


Figure 4.1: Recurrent GAN for Energy Data.

Auto Regressive Integrated Moving Average (ARIMA)

Auto Regressive Integrated Moving Average (ARIMA) [111] models are fitted to time series data to better understand the data or to forecast future values in the time series. The Auto Regressive (AR) part models the variable of interest as regression of its own past values, the Moving Average (MA) part uses a linear combination of past error terms for modeling, and Integrated (I) refers to dealing with non-stationarity.

Here, ARIMA is used because of its ability to capture different temporal structures in time series data. As this work is focused on generating energy data, the ARIMA model is fitted to the energy feature. The values obtained from the fitted ARIMA model are added as a new engineered feature to the existing data set. The RNN itself is capable of capturing temporal dependencies, but adding ARIMA further enhances time modeling and, consequently, improves the quality of the generated data.

Fourier Transform

Fourier Transform (FT) decomposes a time signal into its constituent frequency domain representations [111]. Using FT, every waveform can be represented as the sum of sinusoidal functions. An inverse Fourier transform synthesizes the original signal from its frequency domain representations. Because FT identifies which frequencies are present in the original signal, FT is suitable for discovering cycles in the data.

Like with ARIMA, only the energy feature is used with FT. The energy time series is decomposed into sinusoidal representations, n dominant frequencies are selected, and a new time series is constructed using these n constituent signals. This new time series represents a new feature. When the number of components n is low, the new time series only captures the most dominant cycles, whereas for a large number of components, the created time series approaches the original time series. One value of n creates one new feature, but several values with their corresponding features are used in order to capture different temporal scales.

The objective of using FT is similar to the one of ARIMA: to capture time dependencies and, consequently, improve quality of the generated data. The experiments demonstrate that both ARIMA and Fourier transform contribute to the quality of the generated data.

Feature Normalization

To bring the values of all features to a similar scale, the data was normalized using MinMax normalization. The values of each feature were scaled to values between 0 and 1 as follows:

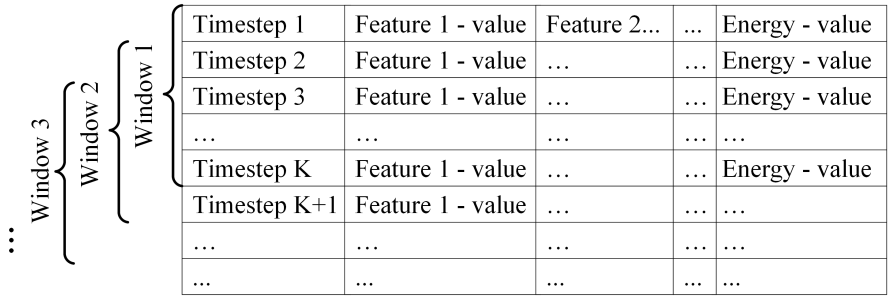
$$x' = \frac{x - Min(X)}{Max(X) - Min(X)} \tag{4.1}$$

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

Sliding Window

At this point, data is still in a matrix format as illustrated in Fig. 4.2 with each row containing readings for a single time step. Note that features in this matrix include all features contained in the original data set (core features) such as appliance status or the day of the week, as well as additional features engineered with ARIMA and FT.

As the generator core is RNN, samples need to be prepared into a form suitable for RNN. To do this, the sliding window technique is applied. As illustrated in Fig. 4.2, the first K rows correspond to the first time window and make the first training sample. Then, the window slides for S steps, and the readings from the time step S to $K + S$ make the second sample. Note that in Fig. 4.2, the step is $S = 1$ although other step sizes could be used. Each sample is a matrix of dimension $K \times F$, where K is the window length and F is the number of features.



Timestep 1	Feature 1 - value	Feature 2...	...	Energy - value
Timestep 2	Feature 1 - value	…	…	Energy - value
Timestep 3	Feature 1 - value	…	…	Energy - value
…	…	…	…	…
Timestep K	Feature 1 - value	…	…	Energy - value
Timestep K+1	Feature 1 - value	…	…	…
…	…	…	…	…
...	...	...	...	...

Figure 4.2: Sliding Window Approach.

4.1.2 R-GAN

Similar to an image GAN, R-GAN consists of the generator and the discriminator as illustrated in Fig. 4.1. However, while an image GAN typically uses CNN for the generator and discriminator, R-GAN substitutes CNN with the stacked LSTM and a single dense layer. The architectures of the R-GAN generator and discriminator are shown in Fig. 4.3. The stacked LSTMs were selected because the LSTM cells are able to store information for longer sequences than Vanilla RNN cells. Moreover, stacked hidden layers allow capturing patterns at different time scales.

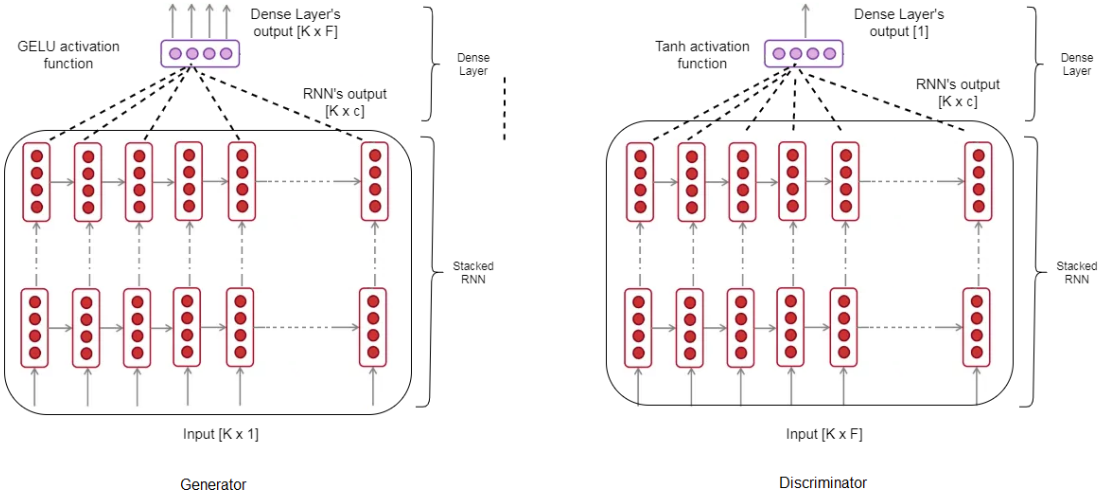


Figure 4.3: R-GAN generator and discriminator.

Both the generator and discriminator have a similar architecture consisting of stacked

LSTM and a dense layer (Fig 4.3), but they differ in dimensions of their inputs and outputs because they serve a different purpose: one generates data and the other one classifies samples into fake and real. The generator takes random inputs from the predefined Gaussian latent space and generates time series data. The input has the same dimension as the siding window length K. The RNN output before the fully connected later has dimension $K \times c$ (window length $\times$ cell state dimension). The generated data (generator output) has the same dimensions as the real data after pre-processing: each sample is of dimension $K \times F$ (window length $\times$ number of features).

GELU (Gaussian Error Linear Unit) activation function has been selected for the generator and discriminator RNNs [112] as it recently outperformed Rectified Linear Unit (ReLU) and other activation functions [112]. The GELU can be approximated by [112]:

$$GELU(x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right) \tag{4.2}$$

Similar to the ReLU (Rectified Linear Unit) and the ELU (Exponential Linear Unit) activation functions, GELU enables faster and better convergence of neural networks than the sigmoid function. Moreover, GELU merges ReLU functionality and dropout by multiplying the neuron input by zero or one, but this dropout multiplication is dependent on the input: there is a higher probability of the input to be dropped as the input value decreases [112]. The stacked RNN is followed by the dense layer in both the generator and discriminator. The dense layer activation function for the generator is GELU because of the same reasons explained with GELU selection in a stacked RNN. In the discriminator, the dense layer activation function is *tanh* to achieve real/synthetic classifications.

As illustrated in Fig. 4.1, the generated data together with pre-processed data are passed to the discriminator which learns to differentiate between real and fake samples. After a mini-batch consisting of several real and generated samples is processed, discriminator loss is calculated and the discriminator weights are updated using gradient descent. As R-GAN uses WGAN, the updates are done slightly differently than in the original GAN. In the original GAN work [16], each discriminator update is followed by the generator update. In contrast, the WGAN algorithm trains the discriminator relatively well before each generator update. Consequently, in Fig. 4.1, there are several discriminator update loops before each generator update.

Once the generator and discriminator are trained, the R-GAN is ready to generate data. At this step, typically only the trained generator is used. If the generator was trained perfectly, the resulting generated distribution, P_G, would be the same as the real one. Unfortunately, GANs may not always converge to the true data distribution, thus taking samples directly from these imperfect generators can produce low quality samples. However, our work takes advantage of the Metropolis-Hastings GAN (MH-GAN) approach [24] in which both the generator and discriminator play roles in generating samples.

Data generation using MH-GAN approach is illustrated in Fig. 4.4. The discriminator, together with the trained generator G, forms a new generator G'. The generator G takes as input random samples $\{z_0, z_1, ..., z_k\}$ and produces time series samples $\{x'_0, x'_1, ..., x'_k\}$. Some of the generated samples are closer to the real data distribution than the others. The discriminator serves as a selector to choose the best sample x from the set $\{x'_0, x'_1, ..., x'_k\}$. The final output is the generated time series sample x.

4.2 Evaluation Process

The main objective of this work is to generate data for training ML models; therefore, the presented R-GAN is evaluated by assessing the quality of ML models trained with synthetic data. As energy forecasting is a common ML task, it is used here for the evaluation too. In addition to Train on Synthetic, Test on Real (TSTR) and Train on Real, Test on Synthetic (TRTS) approaches proposed by Esteban *et al.* [52], two additional metrics are employed:

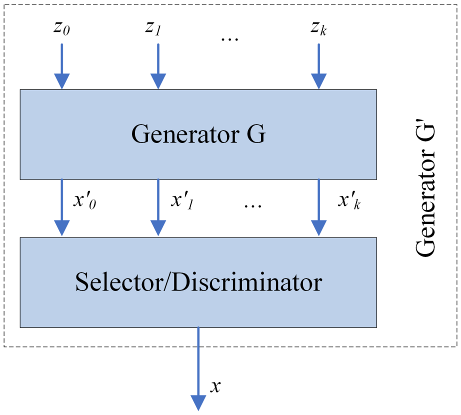


Figure 4.4: MH-GAN approach.

Train on Real, Test on Real (TRTR) and Train on Synthetic, Test on Synthetic (TSTS).

Train on Synthetic, Test on Real (TSTR)

A prediction model is trained with synthetic data and tested on real data. TSTR was proposed by Esteban *et al.* [52]: they evaluated the GAN model on a clustering task using random forest classifier. In contrast, our study evaluates R-GAN on an energy forecasting task using an RNN forecasting model. Note that this forecasting RNN is different than RNNs used for the GAN generator and discriminator, and could be replaced by a different ML algorithm. RNN was selected because of its recent success in energy forecasting studies [43]. This forecasting RNN is trained with synthetic data and tested on real data.

Consequently, TSTR evaluates the ability of the synthetic data to be used for training energy forecasting models. If the R-GAN suffers from the mode collapse, TSTR degrades because the generated data do not capture diversity or real data and, consequently, the prediction model does not capture this diversity.

Train on Real, Test on Synthetic (TRTS)

This is the reverse of TSTR: a model is trained on the real data and tested on the synthetic data. The process is exactly the same as in TSTR with exception of reversed roles of synthetic and real data. TRTS serves as an evaluation of GAN's ability to generate realistic looking data. Unlike TSTR, TRTS is not affected by the mode collapse as a limited diversity of synthetic data does not affect forecasting accuracy. As the aim is to generate data for training ML models, TSTR is a more significant metrics than TRTS.

Train on Real, Test on Real (TRTR)

This is a traditional evaluation with the model trained and tested on the real data (with separate train and test sets). TRTR does not evaluate the synthetic data itself, but it allows for the comparison of accuracy achieved when a model is trained with real and with synthetic data. Low TRTR and TSTR accuracies indicate that the forecasting model is not capable of capturing variations in data and do not imply low quality of synthetic data. The goal of the presented R-GAN data generation is the TSTR value comparable to the TRTR value, regardless of their absolute values: this demonstrates that the model trained using synthetic data has similar abilities as the model trained with real data.

Train on Synthetic, Test on Synthetic (TSTS)

Similar to TRTR, TSTS evaluates the ability of the forecasting model to capture variations in data: TRTR evaluates the accuracy with real data and TSTS with synthetic data. Large discrepancies between TRTR and TSTS indicate that the model is much better with real data than with synthetic, or the other way around. Consequently, this means that the synthetic data does not reassemble the real data.

4.3 Evaluation of R-GAN

This section first introduces the data sets and pre-processing. Next, the quality of synthetic data is assessed using various methods, and finally, the overall results for each dataset are discussed.

4.3.1 Data sets and Pre-Processing

The evaluation was carried out on two data sets: UCI appliances energy prediction data set [113] and Building Data Genome set [109]. UCI data set consists of energy consumption readings for different appliances with additional attributes such as temperature and humidity. The reading interval is 10 minutes and the total number of samples is 19,736. Day of the week and month of the year features were created from reading date/time, resulting in a total of 28 features.

Building Data Genome set contains one year of hourly, whole building electrical meter data for non-residential buildings. In experiments, readings from a single building were used; thus, the number of samples is 24 * 365 = 8,760. With this data set, energy consumption, year, month, day of the year, and hour of the day features were used.

For both data sets, the process is the same. The data set is pre-processed as described in subsection 4.1.1. ARIMA is applied first to create an additional feature: to illustrate this step, Fig. 4.5 shows original data (energy consumption feature) and ARIMA fitted model for UCI data set.

Next, Fourier transform (FT) is applied. FT can be used with a different number of components resulting in different signal representations; in the experiments, four transformations were considered with 1, 10, 100, and 1000 components. The four representations are illustrated in Fig. 4.6 on UCI data set. It can be observed that one component results in almost constant values and 10 components capture only large scale trends. As the number of components increases to 100 and 1000, more smaller-scale changes are captured and the representation is closer to the original data. These four transformations with 1, 10, 100, and 1000 components make the four additional features.

At this point, all needed additional features are generated (total of 33 features), and the pre-processing continues with normalization (Fig. 4.1). To prepare data for RNN, the sliding window technique is applied with the window length $K = 60$ indicating that 60 time steps make one sample, and step $S = 30$ specifying that the window slides for 30 time steps to generate

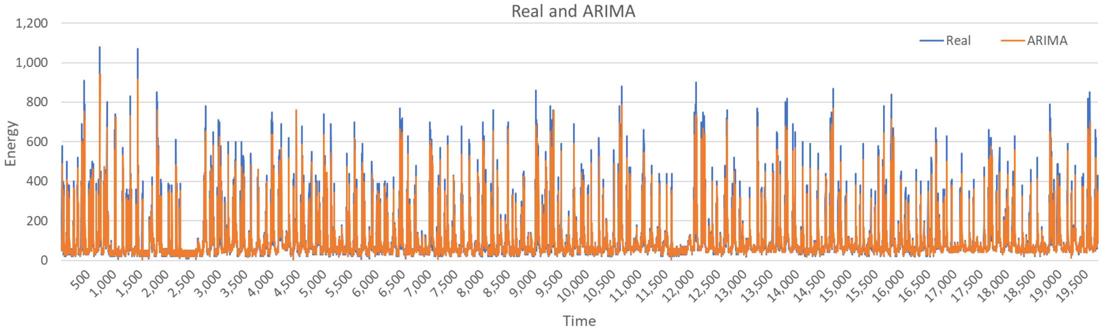


Figure 4.5: ARIMA fitted model (UCI data set).

the next sample. This window size and step were determined from the initial experiments.

The model performance is compared with two metrics: Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE). They were selected as evaluation metrics because of their frequent use in energy forecasting studies [32, 114]. They are calculated as:

$$\text{MAPE} = \frac{100\%}{N} \sum_{t=1}^{N} \left| \frac{y_t - \hat{y}_t}{y_t} \right| \tag{4.3}$$

$$\text{MAE} = \frac{1}{N} \sum_{t=1}^{N} |y_t - \hat{y}_t| \tag{4.4}$$

where y is the actual value, $\hat{y}$ is the predicted value, and N is the number of samples.

4.3.2 Experiments

The R-GAN was implemented in Python with Tensorflow library [115]. The experiments were performed on a computer with Ubuntu OS, AMD Ryzen 4.20 GHz processor, 128 GB DIMM RAM, and four NVIDIA GeForce RTX 2080 Ti 11GB graphics cards. Training the proposed R-GAN is computationally expensive; therefore, GPU acceleration was used. However, once the model is trained, it does not require significant resources and CPU processing is sufficient.

Both discriminator and generator were stacked LSTMs with the hyperparameters as follows:

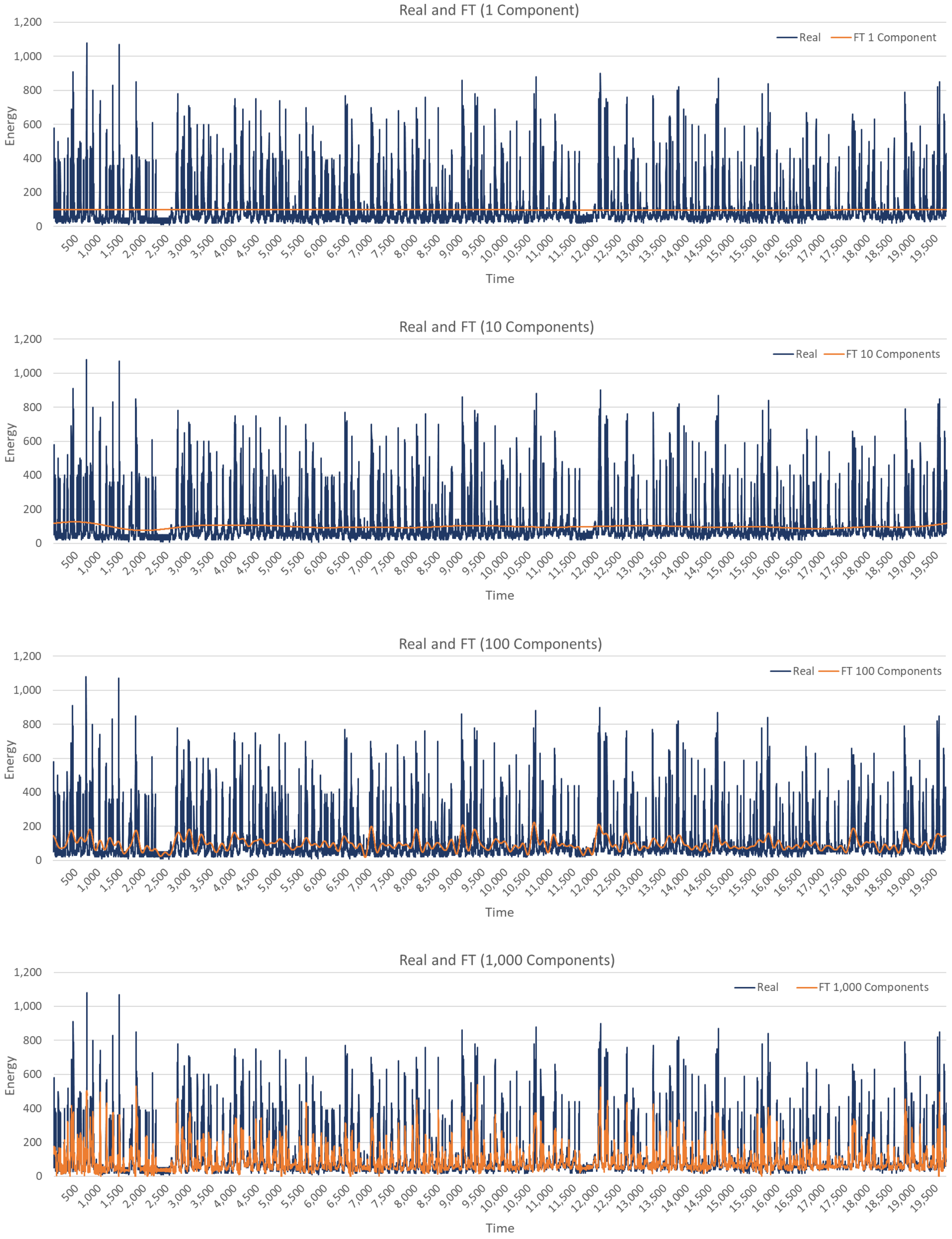


Figure 4.6: The results of Fourier transform with 1, 10, 100, and 1000 components (UCI data set).

- Number of layers L = 2
- Cell state dimension size c = 128
- Learning rate = 2e-6
- Batch size = 100
- Optimizer=Adam

The input to the generator consisted of samples of size 60 (to match the window length) drawn from the Gaussian distribution. The generator output was of size 60×33 (window length $\times$ number of features). The discriminator input was of the same dimension as the generator output and the pre-processed real data.

Hyperparameters were selected according to the hyperparameter studies, commonly used settings, or by performing experiments. Keskar *et al.* [116] observed that performance degrades for batch sizes larger than commonly used 32-521. To keep in that range, and to be close to the original WGAN work [23], batch size 100 was used. For each batch, 100 generated synthetic samples and 100 randomly selected real samples were passed to the discriminator for classification. Increasing the cell state dimension typically leads to the increased LSTM accuracy, but also increases the training time [117]; thus, moderate 128 size was selected.

Greff *et al.* [117] observed that the learning rate is the most important LSTM parameter and that there is often a large range (up to two orders of magnitude) of good learning rates. Fig. 4.7 shows generator and discriminator loss for the learning rates (LR) 2e-6, 2e-5, and 2e-4 for UCI data set and the model with ARIMA and FT features. Similar patterns have been observed with Building Genome data, therefore, here we only discuss loss for UCI experiments. The generator and discriminator are competing against each other; thus, improvement in one results in a decline in the other until the other learns to handle the change and causes the decline of its competitor. Hence, in the graph, oscillations are observed indicating that the networks are trying to learn and beat the competitor. For learning rate 2e-6, the generator stabilizes quite well, while the discriminator shows fluctuations as it tries to defeat the generator. Oscillations of the objective function are quite common in GANs, and WGAN is used in this work to help with convergence. Nevertheless, as the learning rate increases to 2e-5 and 2e-4, the generator and discriminator are experiencing increased instabilities. Consequently, learning rate 2e-6 was used for the experiments presented in this study. Additional hyperparameter tuning has a potential to further improve archived results.

All experiments were carried out with 1500 epochs to allow sufficient time for the system to converge. As can be seen from Fig. 4.7, for learning rate 2e-6, the generator largely stabilizes after around 500 epochs and experiences very little change after 1000 epochs. At the same time, the discriminator experiences similar oscillation patterns from around 400 epochs onward. Thus, training for 1000 epochs might be sufficient; nevertheless, 1500 epochs allow a chance for further improvements.

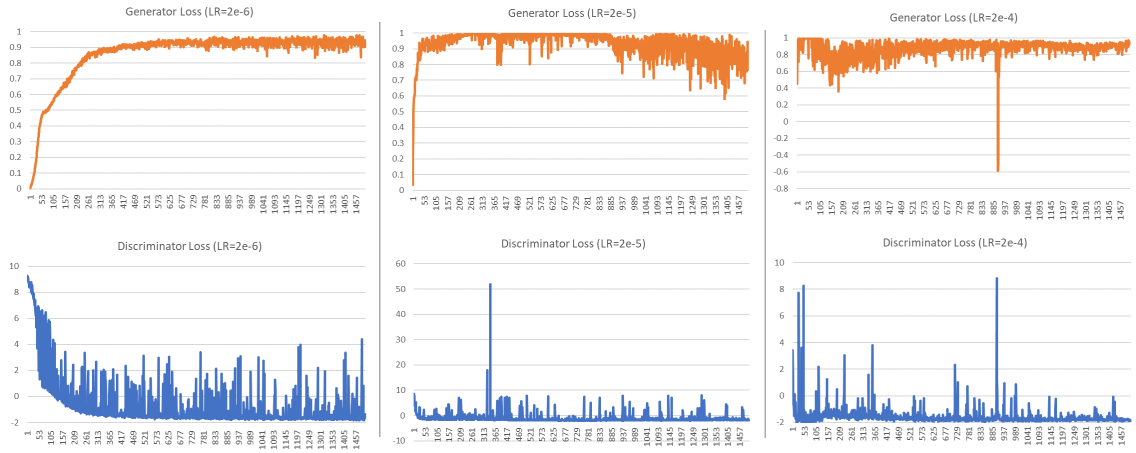


Figure 4.7: Generator and discriminator loss for LR=2e-6, 2e-5, and 2e-4 (UCI data set).

R-GAN was evaluated with the four models corresponding to the four sets of features:

- Core features only.
- Core and ARIMA generated features.
- Core and FT generated features.

- Core, ARIMA, and FT generated features.

As described in section 4.2, ML task, specifically energy forecasting, was used for the evaluation with TRTS, TRTR, TSTS, and TSTR metrics. Forecasting models for those evaluations were also RNNs. Forecasting model hyperparameters for each experiment were tuned using the expected improvement criterion according to Bergstra *et al.* [28], which results in different hyperparameters for each set of input features. This way, we ensure that the forecasting model is tuned for the specific use scenario. The following ranges of hyperparameters were considered for the forecasting model:

- Hidden layer sizes: 32, 64, 128
- Number of layers: 1, 2
- Batch sizes: 1, 5, 10, 15, 30, 50
- Learning rates: continuous from 0.001 to 0.03

For each of R-GAN models, 7200 samples were generated and then TSTR, TRTS, TRTR, an TSTS approaches with an RNN prediction model were applied.

4.3.3 Results and Discussion - UCI data set

This subsection presents results achieved on UCI data set and discusses findings. MAPE and MAE for the four evaluations TRTS, TRTR, TSTR, and TSTS are presented in Table 4.1. For the ease of the comparison, the same data is presented in a graph form: Fig. 4.8 compares models based on MAPE and Fig. 4.9 based on MAE.

As we are interested in using synthetic data for training ML models, TSTR is a crucial metric. In terms of TSTR, addition of ARIMA and FT features to the core features reduces MAPE from 18.67% to 10.12% and MAE from 62.74 to 52.54. Moreover, it can be observed that for all experiments adding ARIMA and FT features improves the accuracy in terms of both MAPE and MAE.

Table 4.1: TRTS, TRTR, TSTR, and TSTS accuracy for R-GAN (UCI data set)

	MAPE(%)				MAE			
Features	TRTS	TRTR	TSTR	TSTS	TRTS	TRTR	TSTR	TSTS
Core features	13.60%	17.98%	18.67%	18.80%	54.26	63.82	62.74	90.90
Core and ARIMA features	8.65%	11.43%	11.37%	8.92%	48.14	62.67	54.00	80.00
Core and FT features	9.07%	15.84%	17.79%	15.10%	48.99	63.12	61.74	90.67
Core, ARIMA and FT features	5.28%	10.81%	10.12%	6.80%	46.41	62.27	52.54	78.35

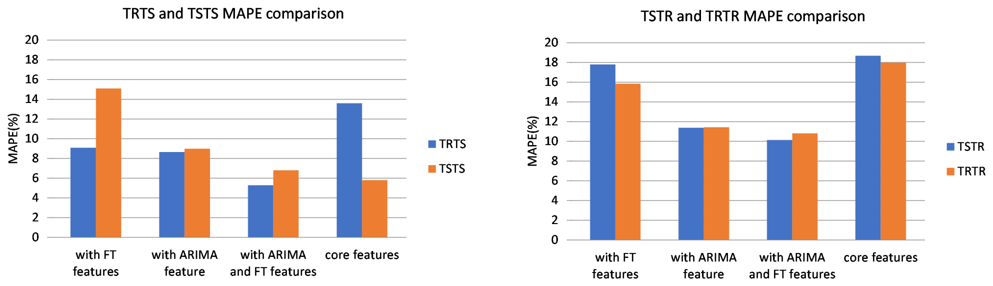


Figure 4.8: MAPE(%) comparison between TRTS/TSTS and TSTR/TRTR (UCI data set).

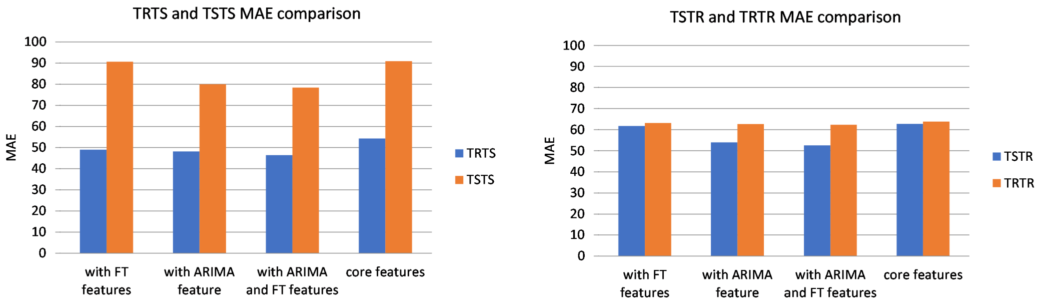


Figure 4.9: MAE comparison between TRTS/TSTS and TSTR/TRTR (UCI data set).

Because forecasting models always result in some forecasting errors even when trained with real data, it is important to compare the accuracy of the model trained with synthetic data with the one trained with the real data. TRTR evaluates the quality of the forecasting model itself. As can be observed from Table 4.1, TSTR accuracy is close to TRTR accuracy for all models irrelevant of the number of features. This indicates that the accuracy of the forecasting model trained with synthetic data is close to the accuracy of the model trained with real data. When the model is trained with real data (TRTR), the MAPE for the model with all features is 10.81% whereas when trained with synthetic data (TRTS) MAPE is 10.12 %. Consequently, comparable TSTR and TRTR values demonstrate the usability of synthetic data for ML training.

The accuracy of TSTR is higher than the accuracy of TRTS in terms of both MAPE and MAE for all experiments. Good TRTS accuracy shows that the predictor is able to generalize from real data and that generated samples are similar to real data. However, higher TSTR errors than TRTS errors indicate that the model trained with generated data does not capture the real data as well as the model trained on the real data. A possible reason for this is that, in spite of using techniques for dealing with mode collapse, the variety of generated samples is still not as high as the variety of real data.

Visual comparison cannot be done in a similar way as in image GANs, but Fig. 4.10 shows examples of two generated samples compared with the most similar real data samples. It can be observed that the generated patterns are similar, but not the same as the real samples; thus, data looks realistic without being a mere repetition of the training patterns. Although Fig. 4.10 provides some insight into generated data, already discussed TSTR and its comparison to TRTR are the main metrics evaluating the usability of generated data for ML training.

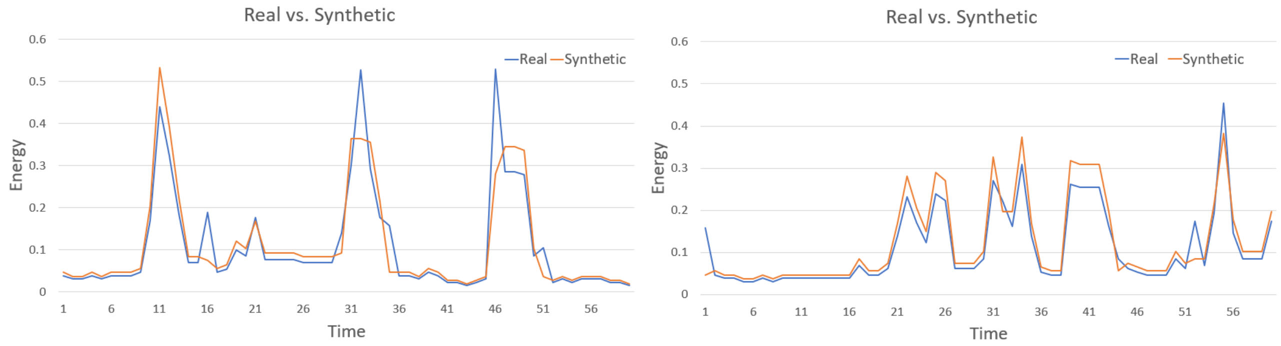


Figure 4.10: Two examples of generated data samples compared to real data.

To further compare real and synthetic data, statistical tests were applied to evaluate if there is a statistically significant difference between the generated and real data distributions. The Kruskal-Wallis H test also referred to ”one-way ANOVA on ranks”, and the Mann-Whitney

U test were used because both of them are non-parametric tests and do not assume normally distributed data. The parametric equivalent of the Kruskal-Wallis H test is one-way ANOVA (analysis of variance). The null hypothesis for the statistical tests was: the distributions of the real and synthetic populations are equal. The H values and U values together with the corresponding p values for each test and for each of the GAN models are shown in Table 4.2. Each test compares the real data with the synthetic data generated with one of the four models. The level of significance $\alpha = 0.05$ was considered.

As the p value for each test is greater than $\alpha = 0.05$, the null hypothesis is not rejected: for each of the four synthetic data sets, irrelevant of the number of features, there is little to no evidence that the distribution of the generated data is statistically different than the distribution of the real data. H and U tests provide evidence about the similarity of distributions; nevertheless, TSTR and TRTR remain the main metrics for comparing among the GAN models.

Note the intuitive similarity between reasoning behind R-GAN and a common approach for dealing with the class imbalance problem, SMOTE (Synthetic Minority Over-sampling Technique) [118]. SMOTE takes each minority class sample and creates synthetic samples on lines connecting any/all of the k minority neighbors. Although R-GAN deals with the regression problem and SMOTE with classification, both create new samples by using knowledge about existing samples. SMOTE does so by putting new samples between existing (real) ones whereas R-GAN learns from the real data, and then it is able to generate samples similar to the real ones. Consequently, R-GAN has a potential to be used for class imbalance problems.

Overall, the results are promising as the forecasting model trained on the synthetic data is achieving similar forecasting accuracy as the one trained on the real data. As illustrated in Table 4.1, both MAPE and MAE for the model trained on the synthetic and tested on the real data (TSTR) are close to MAPE and MAE for the model trained and tested on the real data (TRTR).

Table 4.2: Kruskal-Wallis H Test and Mann-Whitney U test (UCI data set)

	Kruskal-Wallis H Test		Mann-Whitney U test	
Model (Real vs. Synthetic)	H value	p value	U value	p value
Core features	416.50	0.312	0.247	0.619
Core and ARIMA features	428.00	0.375	0.107	0.744
Core and FT features	390.50	0.191	0.775	0.375
Core, ARIMA, and FT features	380.50	0.180	0.885	0.355

4.3.4 Results and Discussion - Building Genome data set

This subsection presents the results achieved with Building Genome data. MAPE and MAE for the four evaluations TRTS, TRTR, TSTR, and TSTS are presented in Table 4.3. The same data is displayed in a graph form for ease of comparison: Fig. 4.11 compares models based on MAPE and Fig. 4.12 based on MAE.

Similar to the UCI data set, TSTR accuracy is close to TRTR accuracy in terms of both, MAPE and MAE, for all models, irrelevant of the number of features. As in the UCI experiments, this indicates that the accuracy of the forecasting model trained with synthetic data is close to the accuracy of the model trained with real data. The best model was with core and FT features: it achieved the MAPE of 4.86% when trained with real data (TRTR) and 5.49% when trained with synthetic data (TSTR).

While with UCI data set, the model with FT and ARIMA features achieved the best results, with Building Genome data, the model with FT (without ARIMA) achieved the best result over all metrics. Note that this is the case even when the model is trained and tested on the real data (TRTR), thus indicating the model behavior and not the data generation characteristics. For Building Genome data, the Pearson-Correlation between energy consumption and the ARIMA feature was 0.987 indicating a high linear correlation between the two. Because of this multicollinearity, the forecasting model achieved better accuracy without the ARIMA features, irrelevant if trained with real or synthetic data.

As with UCI data set, TSTR errors are higher than TRTS errors in terms of both, MAPE and MAE, for all experiments. As noted with UCI exepriments, this could be caused by the variety of generated samples not being as high as the variety of real data.

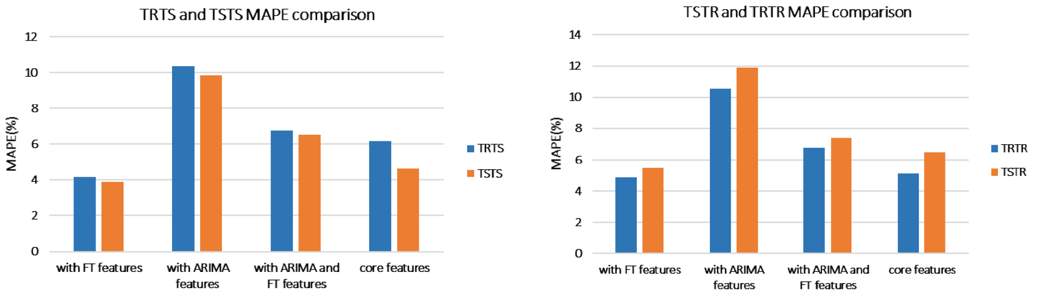


Figure 4.11: MAPE(%) comparison between TRTS/TSTS and TSTR/TRTR (Building Genome data set).

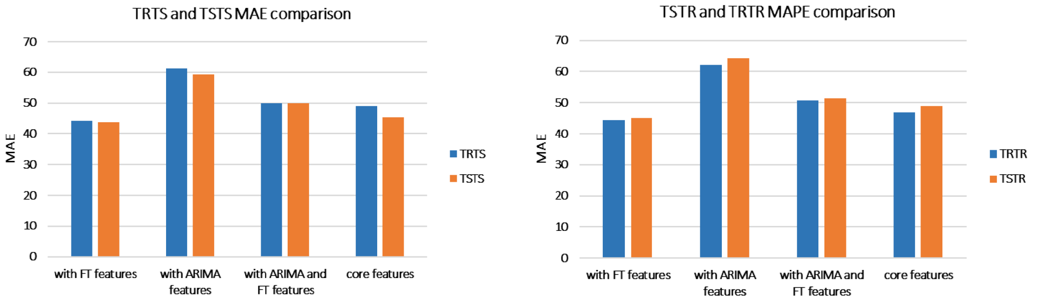


Figure 4.12: MAE comparison between TRTS/TSTS and TSTR/TRTR (Building Genome data set).

Two statistical tests, the Kruskal-Wallis H test and the Mann-Whitney U test, were applied to evaluate if there is a statistically significant difference between the synthetic and real data. Again, the null hypothesis was: the distributions of the real and synthetic populations are equal. Table 4.4 shows H values and U values together with the corresponding p values for each test

Table 4.3: TRTS, TRTR, TSTR, and TSTS accuracy for R-GAN (Building Genome data set)

	MAPE(%)				MAE			
Features	TRTS	TRTR	TSTR	TSTS	TRTS	TRTR	TSTR	TSTS
Core features	6.16%	5.13%	6.48%	4.65%	48.98	46.88	49.00	45.54
Core and ARIMA features	10.37%	10.54%	11.89%	9.84%	61.38	62.15	64.16	59.2
Core and FT features	4.16%	4.86%	5.49%	3.88%	44.13	44.46	45.12	43.84
Core, ARIMA and FT features	6.76%	6.77%	7.37%	6.5%	50.03	50.83	51.33	50.00

Table 4.4: Kruskal-Wallis H Test and Mann-Whitney U test (Building Genome data set)

	Kruskal-Wallis H Test		Mann-Whitney U test	
Model (Real vs. Synthetic)	H value	p value	U value	p value
Core features	430.0	0.387	0.087	0.767
Core and ARIMA features	434.00	0.409	0.056	0.813
Core and FT features	403.50	0.248	0.473	0.492
Core, ARIMA, and FT features	433.00	0.404	0.063	0.802

and for each of the GAN models. The same level of significance $\alpha = 0.05$ was considered.

For Building Genome experiments, same as for UCI experiments, the p value for each test is greater than $\alpha = 0.05$ and the null hypothesis is confirmed: for each of the four synthetic data sets, irrelevant of the number of features, there is little to no evidence that the distribution of the generated data is statistically different than the distribution of the real data.

Overall, the results for both data sets, UCI and Building Genome data set, exhibit similar trends. As illustrated in tables 4.1 and 4.3, accuracy measures, MAPE and MAE, for the models trained on the synthetics data and tested on the real data (TSTR) are close to MAPE and MAE for the model trained and tested on the real data (TRTR) indicating suitability of generated data for training ML models.

4.4 Discussion

This study investigates generating energy data for machine learning taking advantage of Generative Adversarial Networks (GANs) typically used for generating realistic-looking images. Introduced Recurrent GAN (R-GAN) replaces Convolutional Neural Networks (CNNs) used in image GANs with Recurrent Neural Networks (RNNs) because of RNNs ability to capture temporal dependence in time series data. To deal with convergence instability and to improve the quality of generated data, Wasserstein GANs (WGANs) and Metropolis-Hastings GAN (MH-GAN) techniques were used. Moreover, ARIMA and Fourier Transform were applied to generate new features and, consequently, improve the quality of generated data.

To evaluate the suitability of data generated with R-GANs for machine learning, energy forecasting experiments were conducted. Synthetic data produced with R-GAN was used to train the energy forecasting model and then, the trained model was tested on the real data. Results show that the model trained with synthetic data achieves similar accuracy as the one trained with real data. The addition of features generated by ARIMA and Fourier transform improves the quality of generated data.

Chapter 5

Online Deep Learning

DL techniques, especially RNNs, greatly advanced load forecasting and improved its accuracy; however, several forecasting challenges remain: (1) The conventional offline models are trained once by repeatedly passing all training data through the model; one pass is referred to as *epoch*. Then, the model is used to infer future loads. This approach is missing out on the information that new data could provide. (2) The data distributions in energy domain change over time, producing what is known as *concept drift* [9]: for example, installing high-efficiency equipment will reduce energy consumption. In the presence of concept drift, conventional machine learning models experience weak and degrading predictive performance [10].

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

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

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

5.1 Online Adaptive RNN

This section presents Online Adaptive RNN, a load forecasting system that dynamically learns from continuously arriving data and adapts to new patterns in the data. The approach uses batch-normalized RNN (BNRNN) as the base learner and combines Bayesian optimization, performance monitoring, and buffering to tune the BNRNN model on the fly. Online Adaptive RNN is depicted in Fig. 5.1 and Algorithm 2, while details of each component are described in the following subsections.

5.1.1 Prepossessing Module

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

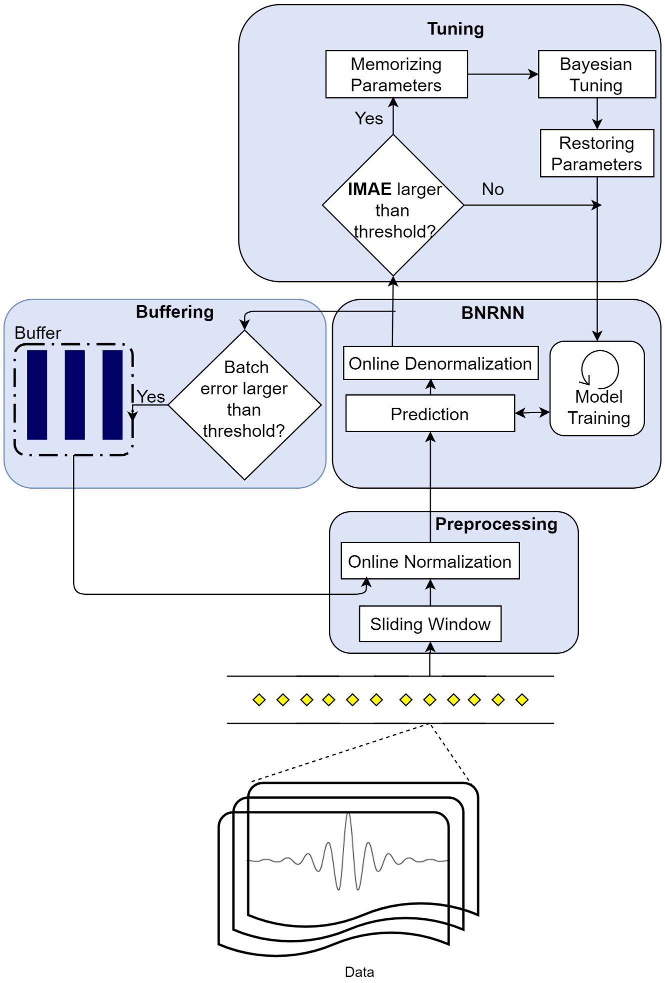


Figure 5.1: Online Adaptive RNN: components and the processing flow

Algorithm 2 Online Adaptive RNN

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

Sliding Window

The sliding window technique (Algorithm 2, line 4) is illustrated in Fig 5.2. The first W readings correspond to the first window and make the first training sample. Then, the window slides for S steps, and the readings from the time step S to $S + W$ make the second sample and so on. Each sample is a matrix of dimension $W \times F$, where W is the window length and F is the number of features. With S<W, there is an overlap between the sliding windows, and a reading from a single time step belongs to multiple windows.

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

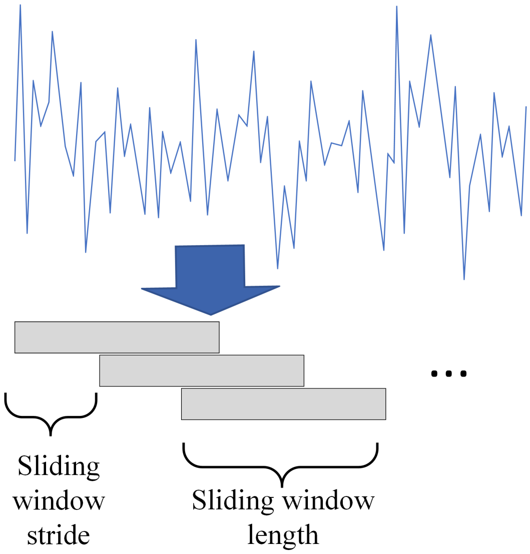


Figure 5.2: Sliding window technique

Online Normalization

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

$$x' = \frac{x - Min(x)}{Max(x) - Min(x)} \tag{5.1}$$

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

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

To address this challenge, the proposed approach carries out *Incremental Min-Max Normalization*. In the main Online Adaptive RNN Algorithm 2, line 7 performs this normalization while the details are presented in Algorithm 3. The maximum and minimum values of the features from the beginning until the current batch are tracked with *globalMax* and *globalMin*. For each new batch, the procedure finds the *max* and *min* values for that batch as shown in Algorithm 3, lines 2 and 3. If the batch *max* is larger than the current global maximum *globalMax* (Algorithm 3, line 4), the *globalMax* is updated (line 5). The same process happens for *globalMin*, lines 6 and 7. Finally, the values from the current batch are normalized using the current *globalMax* and *globalMin*: Algorithm 3, line 8.

Algorithm 3 Incremental Min-Max Normalization

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

5.1.2 Batch Normalized RNN

The RNN was selected as the core learner because of its ability to model temporal dependencies present in the load data. To handle internal covariate shift and reduce training time, batch normalization described in section 2.3 is used. Batch normalization also reduces sensitivity to changes in the learning rate and, consequently, assists the tuning module in dynamically adjusting the learning rate to better capture new data. The BNRNN module consists of three components: prediction, online de-normalization, and model training.

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

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

$$y' = (PredictedValue * (globalMax[load] - globalMin[load])) + globalMin[load] \tag{5.2}$$

where y' is the de-normalized output, $PredictedValue$ is the output of the BNRNN model, $globalMin[load]$ and $globalMin[load]$ are the global maximum and minimum values for the load feature, which were determined during the online normalization step (Section 5.1.1). These de-normalized values are passed to the tuning module for further processing.

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

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

5.1.3 Buffering Module

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

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

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

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

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

5.1.4 Tuning Module

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

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

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

$$IMAE_b = \frac{IMAE_{b-1} + MAE_b}{b} \tag{5.3}$$

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

As shown in Fig. 5.1, if the IMAE is not over the tuning threshold (Algorithm 2, line 13), the BNRNN is trained with the current batch without the learning rate change. If the IMAE is over the tuning threshold (Algorithm 2, line 13), the Bayesian optimizer (BO) is activated to find a new learning rate for the model (Algorithm 2, lines 14 to 16). Here only the learning rate is considered for the tuning, as it has been shown that the learning rate is the most important RNN parameter [117]; however, other non-structural parameters could be tuned using the same approach.

If tuning is required, as illustrated in Fig. 5.1, the weights (parameters) are first preserved (Algorithm 2, line 14) so that they can be restored after the BO process. In offline learning, this preserving is not needed as the optimizer can use all data in all iterations: the BO initializes the model weights with random values every time it evaluates the model hyperparameters causing the model to forget what it has previously learned. In offline learning, this is not a problem as the model will re-learn in the next pass over the same data. In online learning, the model has the access only to a small data segments (batches) at a time, and as parameters are representations of what the model has learned, parameters cannot be forgotten. Because weights represent what the model has learned so far, they are memorized in the parameter preservation step so that they can be restored after the BO changes them during tuning.

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

5.2 Evaluation Methodology

The metrics and the design of experiments for assessing the quality of online learning models are more challenging than those for the offline models because (1) the data are continuously arriving, (2) models evolve over time rather than being static, and (3) the data may be from non-stationary distributions instead of stationary ones (concept drift) [119]. Two possible ways of evaluating online models are holdout and prequential methodologies.

Holdout [119] for online learning is a periodic evaluation method in which a static test set

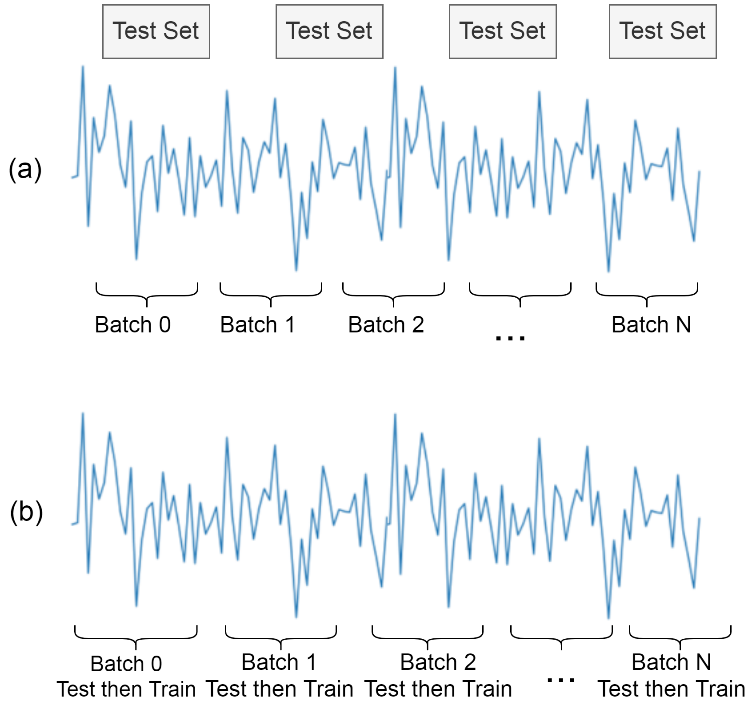


Figure 5.3: Online evaluation: a) holdout evaluation b) prequential evaluation.

is created from unseen samples throughout data stream as illustrated in Fig. 5.3-a. The model is trained on batches or samples as they arrive and evaluated on the test set at regular intervals expressed in terms of time, batches, or samples. For example, after every b batches, the model is evaluated on the test set. The holdout error H_e at the time step i, for the current number of the test set samples M is calculated as:

$$H_e(i) = \frac{1}{M}\sum_{k=1}^{M} E(y_k, \hat{y}_k) = \frac{1}{M}\sum_{k=1}^{M} e_k. \tag{5.4}$$

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

The main drawback of the holdout method comes from the concept drift presence in the data [120]. If the data contains time-evolving concepts, using a static test set to evaluate the model will not provide a good error estimate as the error will change if a different test set is selected.

Predictive Sequential (Prequential) [119] evaluation, as illustrated in Fig. 5.3-b, is an interleaved test-then-train method in which each sample serves two purposes: test and train purposes. As a sample arrives, the model is first tested with this new sample as the input and the error is calculated. Next, the model is trained on this sample. The prequential error P_e, at time i, is calculated as:

$$P_e(i) = \frac{1}{i}\sum_{k=1}^{i} L(y_k, \hat{y}_k) = \frac{1}{i}\sum_{k=1}^{i} e_k. \tag{5.5}$$

where y_k, $\hat{y}_k$, and L are the same as in Equation 5.4. Note that this is the same as Equation 5.4 for holdout error; the only difference is that here the summation is over all samples up to sample i, and in a holdout, it only includes the sample from the test set. By including all samples in the error evaluation, the prequential approach avoids test set selection bias.

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

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

Prequential-Holdout technique combines offline holdout and online prequential techniques for the comparison of online and offline models as shown in Fig. 5.4. The evaluation process consists of the following steps:

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

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

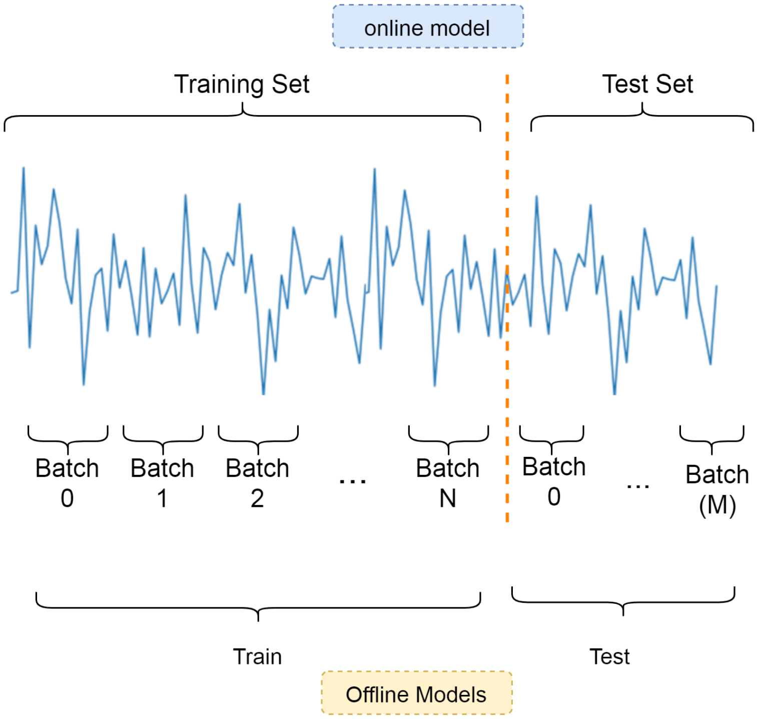


Figure 5.4: Prequential-Holdout evaluation.

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

$$\mathrm{MSE} = \frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2 \tag{5.6}$$

$$\mathrm{MAE} = \frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t| \tag{5.7}$$

where N is the number of sampled in the test set, and y_k, and $\hat{y}_k$ are the same as in Equation 5.4.

5.3 Evaluation of Online Adaptive RNN

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

5.3.1 Dataset and Preliminary Analysis

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

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

Stationary Analysis

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

ADF determines if the series is stationary or not by observing the presence of a unit root.

Table 5.1: ADF Test

Dataset	Test Statistic	CV		
		α= 1%	α=5%	α=10%
House 1 load	-10.3306	-3.4306	-2.8616	-2.5668
House 2 load	-11.8989	-3.4306	-2.8616	-2.5668
House 3 load	-13.2413	-3.4306	-2.8616	-2.5668
House 4 load	-11.9328	-3.4306	-2.8616	-2.5668
House 5 load	-10.3135	-3.4306	-2.8616	-2.5668

The null hypothesis for this test is: The series has a unit root - it is non-stationary. In the ADF test, if the test statistic value is less than the critical value (CV), the null hypothesis is rejected, which means that the series is stationary.

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

Table 5.1 and Table 5.2 show the ADF and KPSS test results respectively for the load from the five homes. The ADF test statistics are lower than the critical values for all home and all significance levels α, indicating that the series are stationary. The KPSS test statistics for all home are greater than the critical values, indicating that the series are non-stationary. The ADT test only checks for one type of non-stationarity, a unit root non-stationarity, which is a possible reason for different results from the two tests. Nevertheless, the possible presence of non-stationarity makes the modeling more difficult and imposes challenges on load forecasting.

Concept Drift Analysis

The major advantage of the online models in load forecasting is their ability to adapt to changes in data patterns. Thus, the data sets from the five houses are first examined for the presence of concept drift. Three concept drift detection methods have been applied Adaptive Windowing

Table 5.2: KPSS Test

Dataset	Test Statistic	CV			
		α= 1%	α=5%	α=10%	
House 1 load	1.1650	0.3470	0.4630	0.5740	0.7390
House 2 load	3.0379	0.3470	0.4630	0.5740	0.7390
House 3 load	14.9290	0.3470	0.4630	0.5740	0.7390
House 4 load	11.6656	0.3470	0.4630	0.5740	0.7390
House 5 load	4.6686	0.3470	0.4630	0.5740	0.7390

(ADWIN) [121], Page-Hinkley (PH) [122], and Drift Detection (DDM) [123]:

- **ADWIN** method uses sliding windows of variable size according to the changes observed from the data. If the difference between the statistics observed in the windows surpasses the threshold, concept drift is detected.

- **PH** method continuously monitors the difference between the time series values and the current mean. The difference greater than the threshold indicates the concept drift.

- **DDM** is based on the idea that as long as the data distribution is stationary, the learner's error rate does not increase. This approach monitors the online error of the algorithm, and when this error increases, concept drift is detected.

Figures 5.5 and 5.6 depict the results of the three aforementioned concept drift detection methods for homes one and four. In ADWIN and PH graphs, the vertical lines indicate the points in which concept drifts have taken place. In the DDM graph, the colorful sections indicate the data with similar distributions and statistics, and, therefore, different sections show different concept drifts occurrences.

It can be observed that the three algorithms detect a different number of concept drift occurrences at different locations in the time series. Nevertheless, all algorithms indicate the extensive presence of concept drift, and changes in load patterns are notable even from the visual observations of the loads. The existence of concept drift is especially pronounced in house

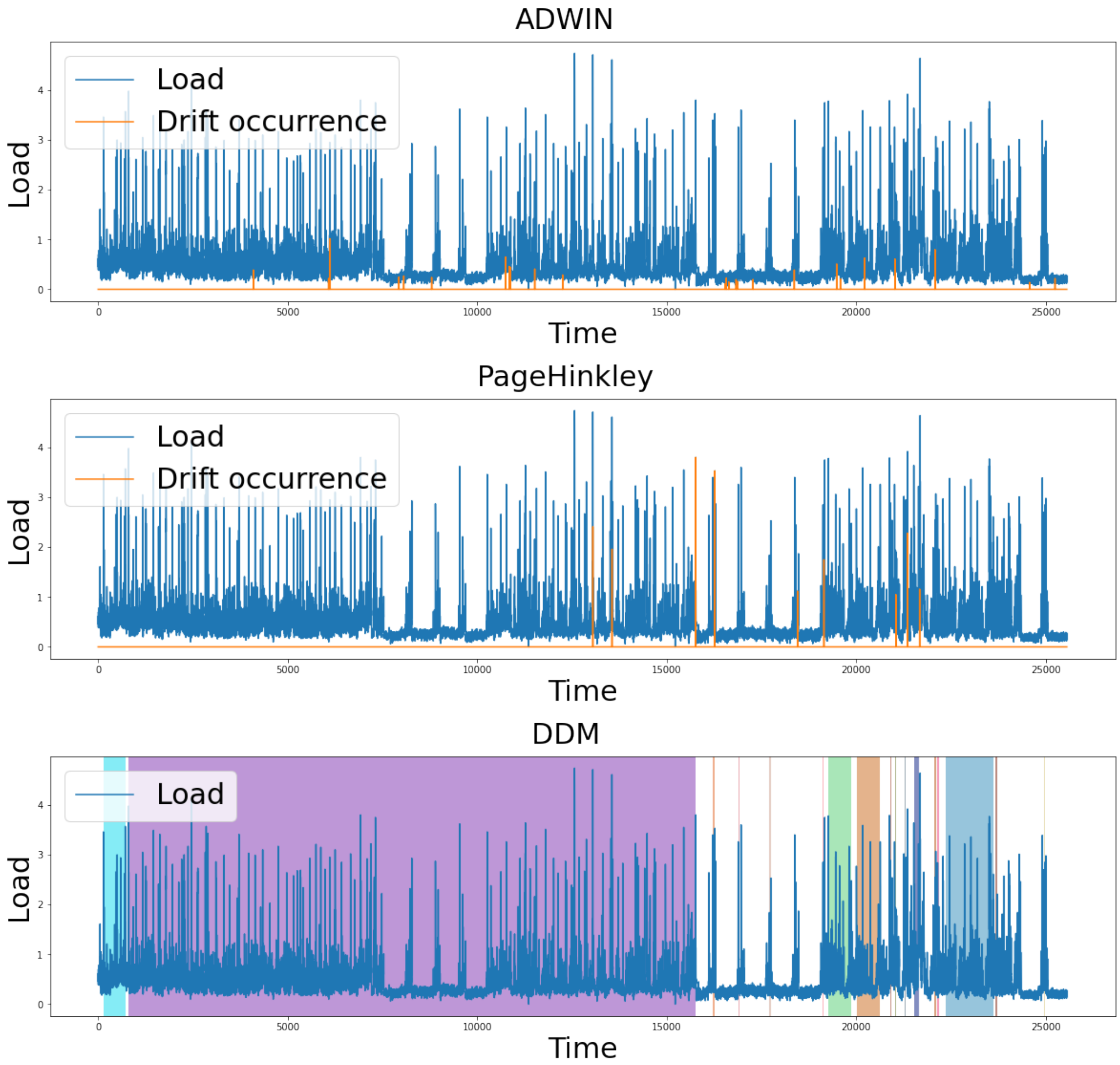


Figure 5.5: The electricity load and the concept drift occurrences detected with ADWIN, PH, and DMM: house 1.

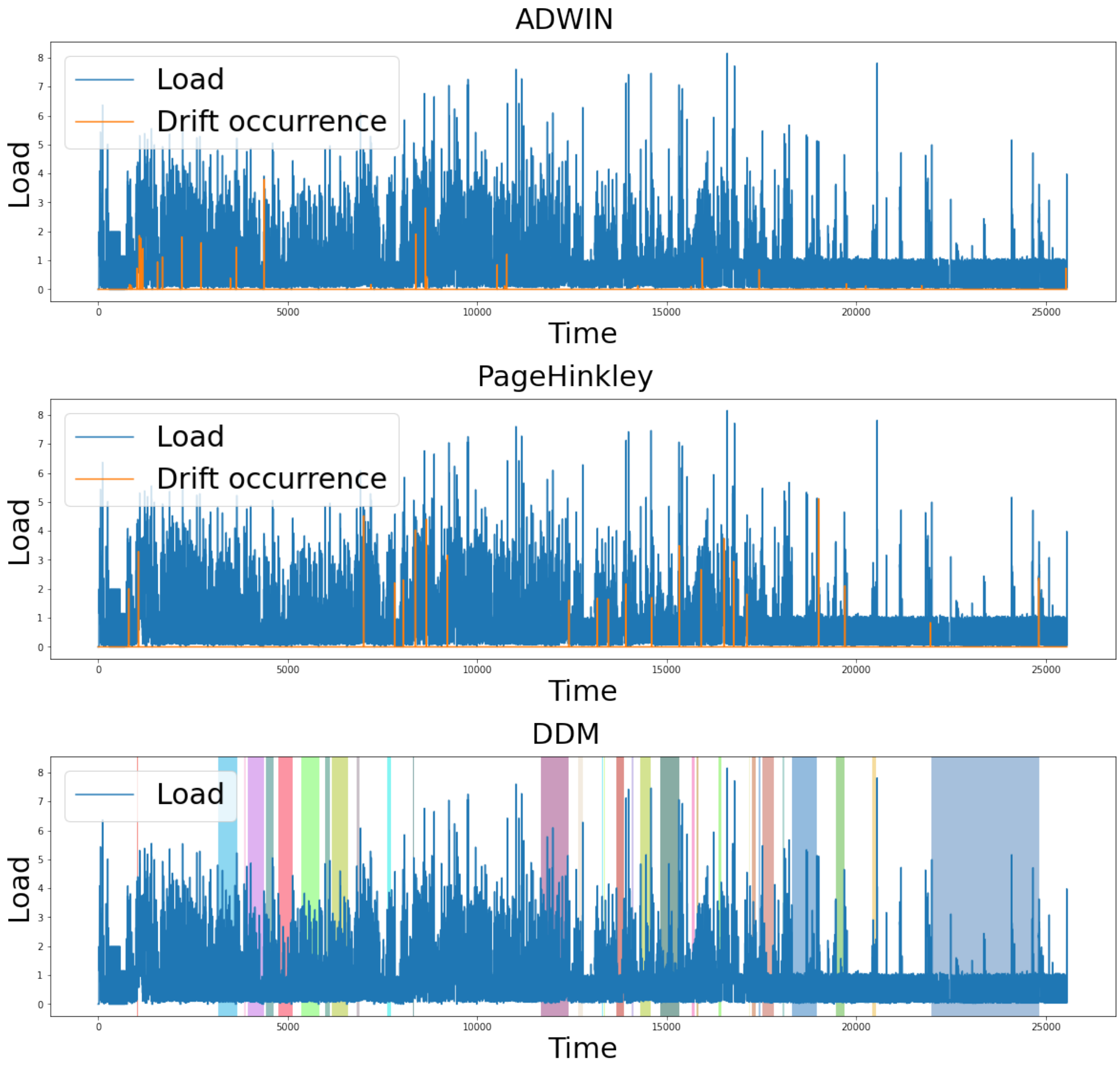


Figure 5.6: The electricity load and the concept drift occurrences detected with ADWIN, PH, and DMM: house 4.

four with the DDM method: a high number of diversely colored segments indicate different distributions thought the series. Such a high presence of concept drift in load data hugely degrades the accuracy of the offline models and imposes challenges for online models as they need to continuously adapt to new patterns. Consequently, for household-level load forecasting, it is critical to use online models capable of handling such changes.

5.3.2 Comparison with offline LSTM

This subsection compares the proposed Online Adaptive RNN with conventional offline LSTM. Online and offline approaches are intrinsically different and cannot be directly compared; however, it is still important to compare them as both are used for load forecasting. The comparison is carried out here with the Prequential-Holdout approach proposed in Section 5.2: the first 70% of data was used for training and the last 30% for testing. This remainder of this subsection first describes experiments and then presents results.

Experiments

In Online Adaptive RNN, the core learner, BNRNN, can be implemented with different recurrent cells: experiment presented here used LSTM cells as these cells can handle longer sequences than vanilla RNN cells. As the online model employed LSTM cells, the comparison offline model is also an LSTM network. Both Online Adaptive RNN and offline LSTM were tuned with BO: while offline LSTM applies the traditional BO tuning process, Online Adaptive RNN employs BO as described in Section 5.1.4. The following ranges of hyperparameters were considered for offline LSTM:

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

- Learning rates: continuous from 0.000001, 0.2

In Online Adaptive RNN, as described in Section 5.1.4, the BO tuning happens online; therefore, only the learning rate was tuned. The batch size was 5, and the buffer size was 10 batches for all Online Adaptive RNN experiments. Further tuning the batch size and buffer size could potentially improve the accuracy, but would incur extensive computation cost. Moreover, a very large buffer would make the proposed approach similar to the offline model as many samples would be reused in each training step. Thresholds for tuning and buffering modules were tuned for individual households.

The load forecasting accuracy is highly dependent on the forecasting horizons: shorter horizons typically lead to higher accuracy. Moreover, the model's forecasting accuracy on one horizon does not necessarily directly translate to its performance on another horizon. Consequently, Online Adaptive RNN and offline LSTM were evaluated for four load forecasting horizons: 1 hour ahead, 50 hours ahead, 100 hours ahead, and 200 hours ahead.

Results

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

Fig. 5.7 compares the accuracy of Online Adaptive RNN and the traditional LSTM in terms of MAE and MSE for the four prediction horizons. As the scale of the errors for the LSTM and Online Adaptive RNN are different, the figure has two vertical axes: the left one is for Online Adaptive RNN and the right one is for the offline LSTM. Moreover, the accuracy values are displayed with each data point. Although the offline LSTM hyperparameters were tuned, the MAE and MSE errors are relatively high. The main reason for this is the presence of the concept drift. Data distribution and load patterns change over time, resulting in the low predictive power of the static offline model. The accuracy of the online approach is much higher as the model adapts to the changes in data patterns as they arrive.

From Fig. 5.7, it can be observed that the accuracy varies greatly among houses, prediction horizons, as well as between measures (MAE and MSE). For offline LSTM, house 5 consistently has the highest forecasting errors for all horizons and both metrics. Similarly, houses 1 and 3 have the lowest errors. In the case of Online Adaptive RNN, MSE for house 3 is lower than for other houses, while in terms of MAE, this house is among two with the lowest error. Possible reasons for this are fewer or weaker concept drifts in the house 3 data.

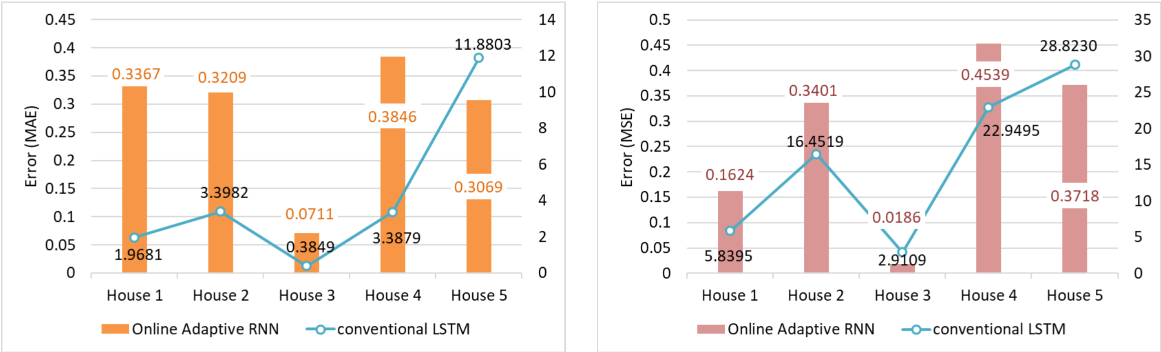
5.3.3 Comparison with online models

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

Experiments

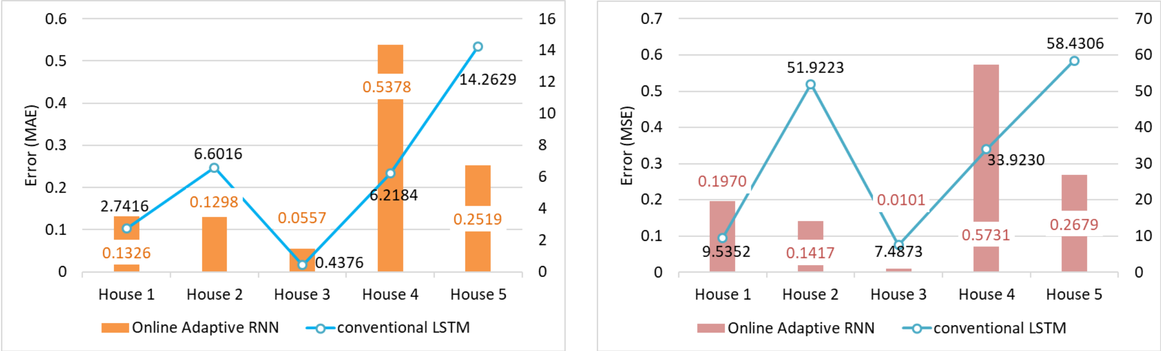
For Online Adaptive RNN, the same setup and tuning have been applied as described in Section 5.3.2 for the comparison with offline models. To keep the comparison fair, parameters for the five competition models were also tuned with BO. The following configurations were considered for tuning:

- Online MLP regression: SGD and Adam optimizers, the hidden layer sizes of 50, 100, 250, 500 neurons.
- Online linear regression: SGD and Adam optimizers, the learning rate in a range of [0.000001, 0.2].
- Online KNN: the window sizes of 25, 50, and 100, and the number of neighbors 3, 5, and 10.



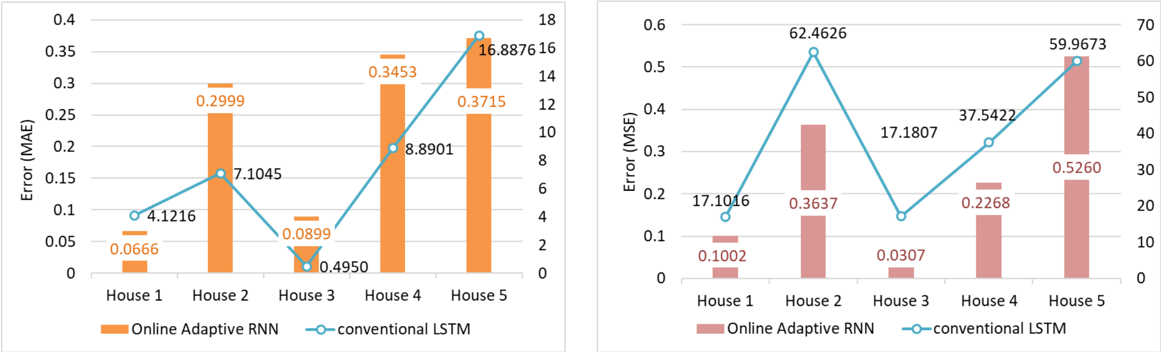
(a) The MAE error for 1 hour ahead.

(b) The MSE error for 1 hour ahead.



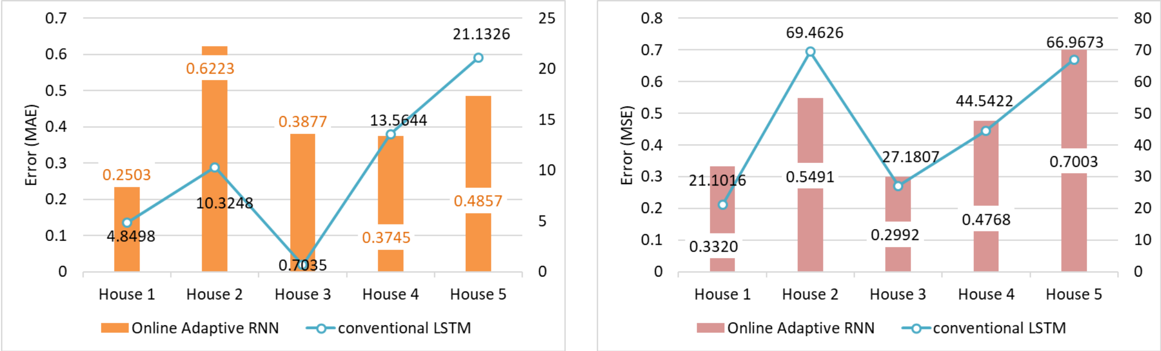
(c) The MAE error for 50 hours ahead.

(d) The MSE error for 50 hours ahead.



(e) The MAE error for 100 hours ahead.

(f) The MSE error for 100 hours ahead.



(g) The MAE error for 200 hours ahead.

(h) The MSE error for 200 hours ahead.

Figure 5.7: The MAE and MSE errors for offline LSTM and Online Adaptive RNN

- Online bagging regression: a linear regression as the base model, and the number of the base models 5 and 10.
- Online passive-aggressive regression: Epsilon insensitive loss function, aggressiveness in a range of [0.01, 10].

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

Results

Fig. 5.8 shows the results of the experiments in terms of MAE and MSE for different prediction horizons. It can be observed that for one hour ahead prediction, Fig. 5.8 a and b, the lowest error was achieved with the online KNN for all households. The KNN predicts based on the similarity of the target to its nearest neighbors and, for one hour ahead prediction, the training data points, and the target are adjacent enough for the KNN to perform well. Nevertheless, the accuracy of Online Adaptive RNN is very close to the KNN.

When the prediction length is increased to 50 hours, the adjacency level of the training data points becomes weaker especially in the presence of concept drifts. For this reason, the error rates for the MLP, Linear, and PA regression models increased significantly as shown in 5.8 c and d. Errors for the KNN and bagging models slightly decreased in comparison to the one hour ahead prediction. For this prediction length, Online Adaptive RNN achieved better accuracy than the remaining five algorithms for all homes.

For 100 hours ahead, Fig. 5.8 e and f, the proposed Online Adaptive RNN remains better than the other models. For 200 hours ahead Online Adaptive RNN outperforms others for all but one household: for house 1, bagging achieves slightly better accuracy than Online Adaptive RNN. However, bagging performs very poorly on household 4 making Online Adaptive RNN an overall better model.

Fig. 5.8 compared the average error for the considered algorithms, but we are also interested in the variability of the error as a model with more consistent prediction is more desirable.

Consequently, Fig. 5.9 examines error intervals for Online Adaptive RNN and the five online algorithms. The error intervals have been calculated based on the error for all households and average, minimum, maximum, first and the third quartile are shown in the figure.

As in Fig. 5.8, in Fig. 5.9, for 1 hour ahead KNN achieved the best average accuracy. From Fig. 5.9 a and b, it can also be observed that error ranges for KNN are among the lowest. Again, performance of Online Adaptive RNN is close to KNN.

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

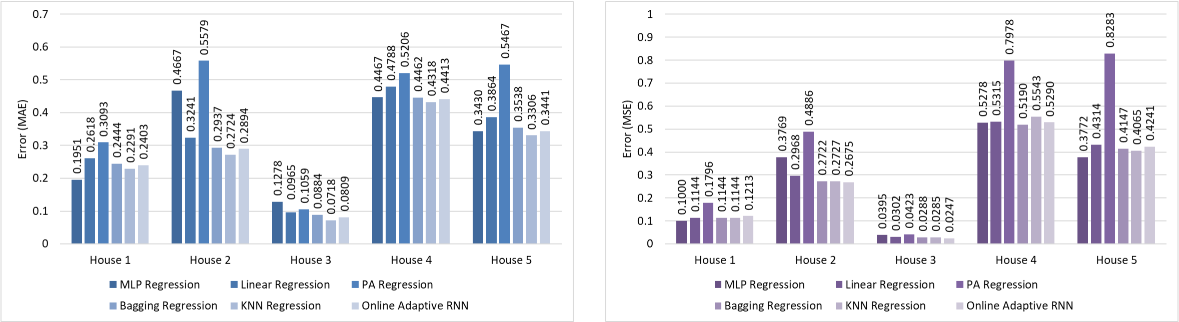
5.3.4 Analysis of modules impact

Experiments presented so far compare the performance of Online Adaptive RN with other models, offline LSTM, and five online models. This section examines the impact of the modules within Online Adaptive RNN on the forecasting accuracy; specifically, the impact of buffering and tuning modules is investigated. As different variants of the proposed online approach are compared, the prequential evaluation described in Section 5.2 is used.

Experiments

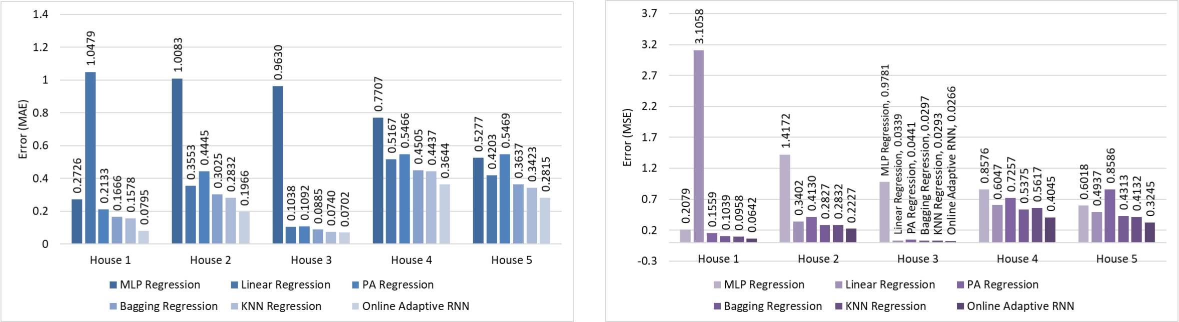
Experiments compare four Online Adaptive RNN variants:

- with buffering and tuning modules
- only with buffering module
- only with tuning module



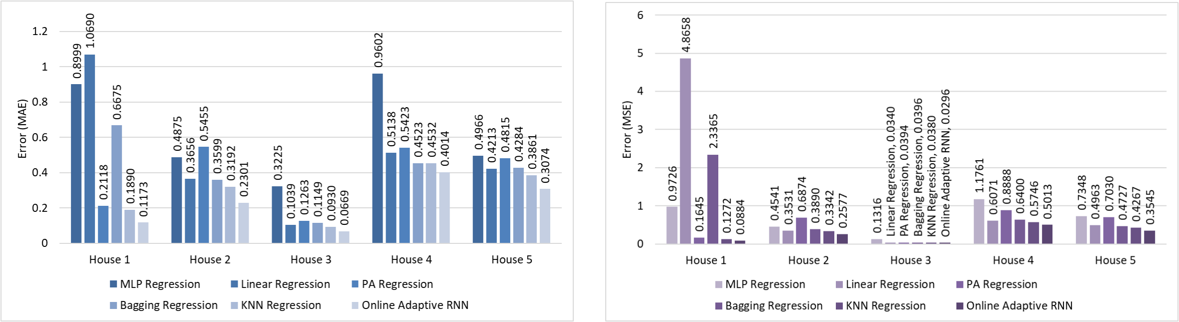
(a) The MAE errors for 1 hour ahead.

(b) The MSE errors for 1 hour ahead.



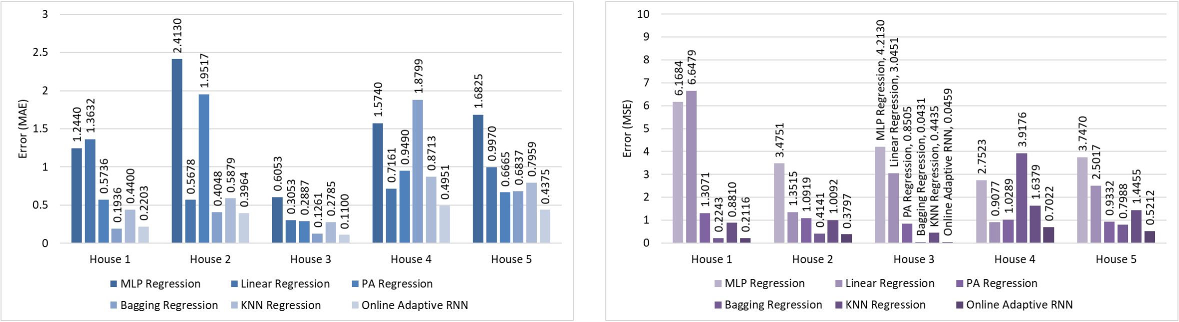
(c) The MAE errors for 50 hours ahead.

(d) The MSE errors for 50 hours ahead.



(e) The MAE errors for 100 hours ahead.

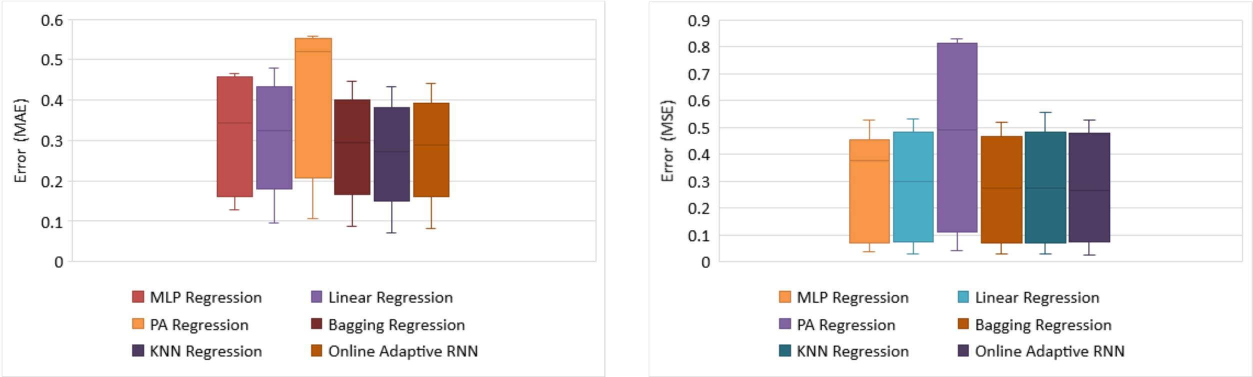
(f) The MSE errors for 100 hours ahead.



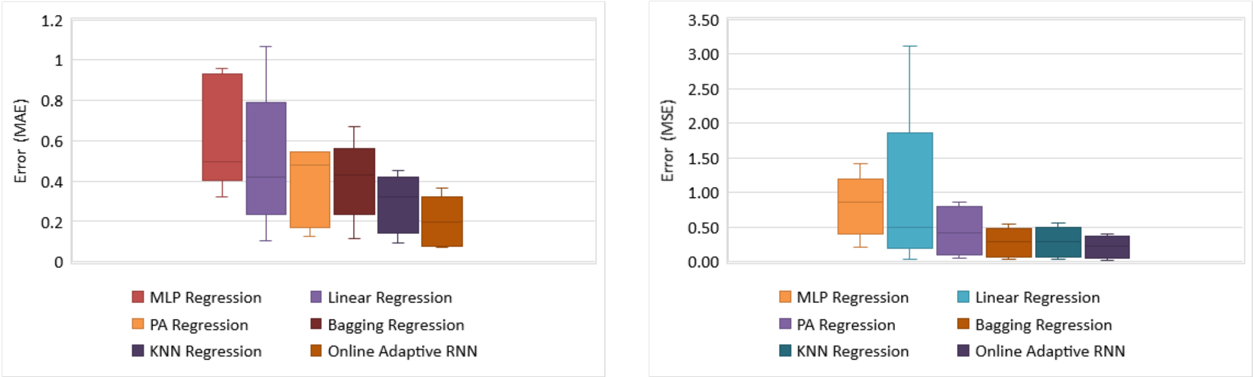
(g) The MAE errors for 200 hours ahead.

(h) The MSE errors for 200 hours ahead.

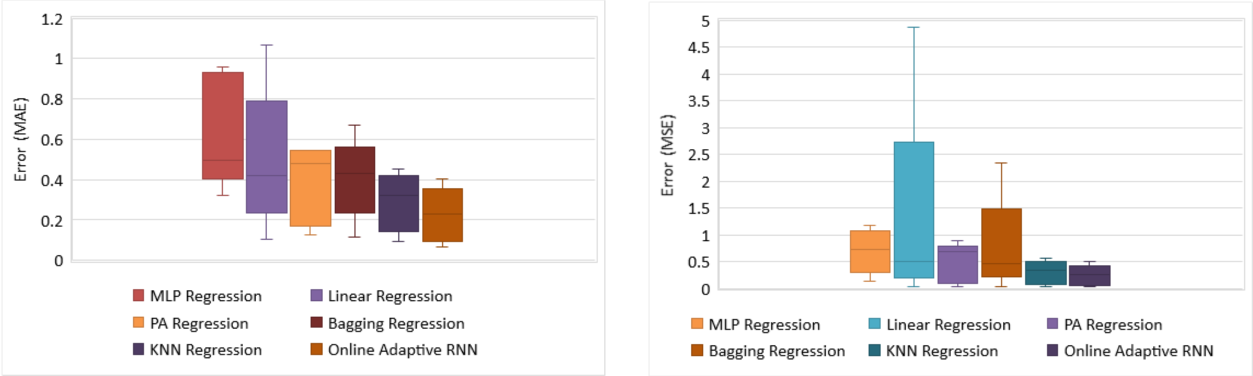
Figure 5.8: Comparison of Online Adaptive RNN with five online regression algorithms: MLP, liner, passive-aggressive, bagging, and KNN regression.



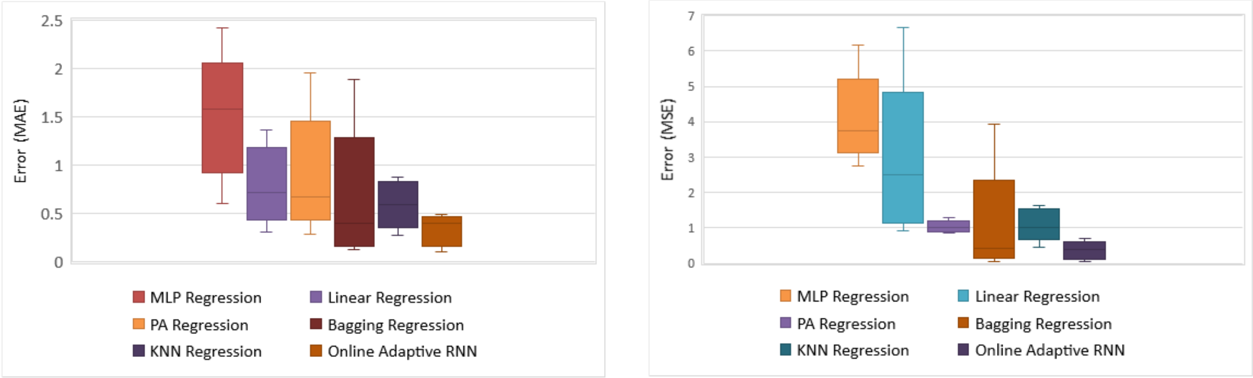
(a) The MAE error intervals for 1 hour ahead. (b) The MSE error intervals for 1 hour ahead.



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



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



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

Figure 5.9: Comparison of error intervals for Online Adaptive RNN and the five online regression algorithms: LP, liner, passive-aggressive, bagging,and KNN regression

- Online Adaptive RNN without buffering and tuning modules

The same setup and tuning have been applied as described in Section 5.3.2; however, note that for the models without the tuning module, there is no tuning. Again, the same forecasting horizons are considered: 1, 50, 100, and 200 hours ahead.

Results

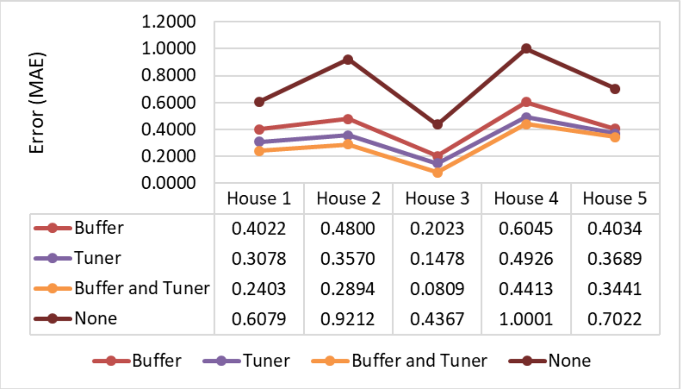
Fig. 5.10 shows the impact of the buffering and tuning modules on the accuracy of Online Adaptive RNN. It can be observed that for all the households, the model with the buffering and tuning modules results in the lowest MAE error, and the model without the two modules leads to the highest errors. This figure also shows that the tuning module has a larger impact on the accuracy than the buffering module as error drops mode when the tuning module is added than when the buffering module is added. This highlights the need to tune network hyperparameters, specifically the learning rate, as new data arrive.

5.3.5 Analysis of the buffer size impact

The previous section demonstrated the impact of the buffering and tuning modules on the performance of Online Adaptive RNN. This subsection further investigates the buffering module and examines the impact of the buffer size on the model accuracy and training time. Since this involves comparing online learning models, Online Adaptive RNNs with different buffer sizes, the prequential evaluation described in Section 5.2 is used.

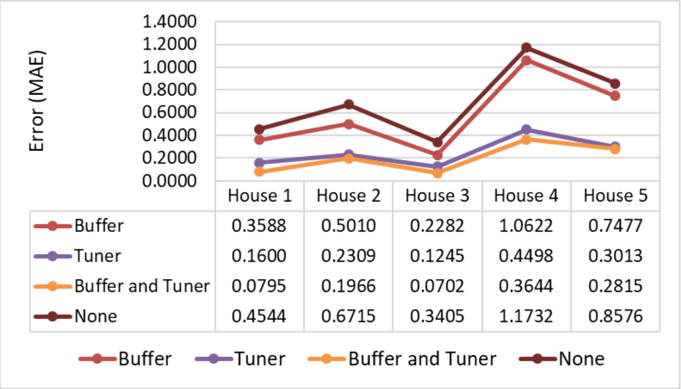
Experiments

For these experiments, the same setup and tuning process have been applied as described in Subsection 5.3.2: LSTM cells are used and the same ranges of parameters are considered in the online tuning. The difference is that here experiments are conducted without buffer and with buffer sizes of 5, 10, 15, 30, and 50 batches.



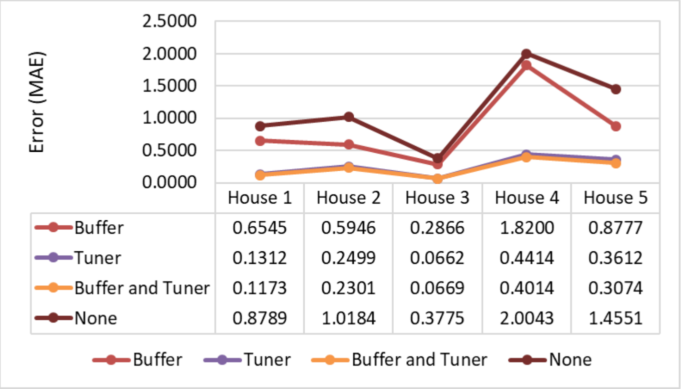
	House 1	House 2	House 3	House 4	House 5
Buffer	0.4022	0.4800	0.2023	0.6045	0.4034
Tuner	0.3078	0.3570	0.1478	0.4926	0.3689
Buffer and Tuner	0.2403	0.2894	0.0809	0.4413	0.3441
None	0.6079	0.9212	0.4367	1.0001	0.7022

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



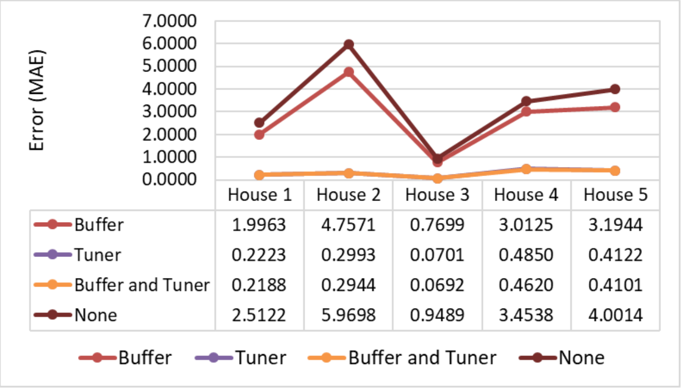
	House 1	House 2	House 3	House 4	House 5
Buffer	0.3588	0.5010	0.2282	1.0622	0.7477
Tuner	0.1600	0.2309	0.1245	0.4498	0.3013
Buffer and Tuner	0.0795	0.1966	0.0702	0.3644	0.2815
None	0.4544	0.6715	0.3405	1.1732	0.8576

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



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

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



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

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

Figure 5.10: The impact of buffering and tuning modules on the forecasting accuracy.

Results

Fig. 5.11 shows the impact of the buffer size on the accuracy of Online Adaptive RNN for 50 hours forecasting horizon. For houses with a high presence of concept dirt, such as houses 4 and 5, a smaller buffer size (5 batches) results in a lower error. The reason for this is that a larger buffer retains more batches, some of which may be older, therefore causing the model to pay attention to those older batches which may not reflect the current situation when there is a high presence of concept drift.

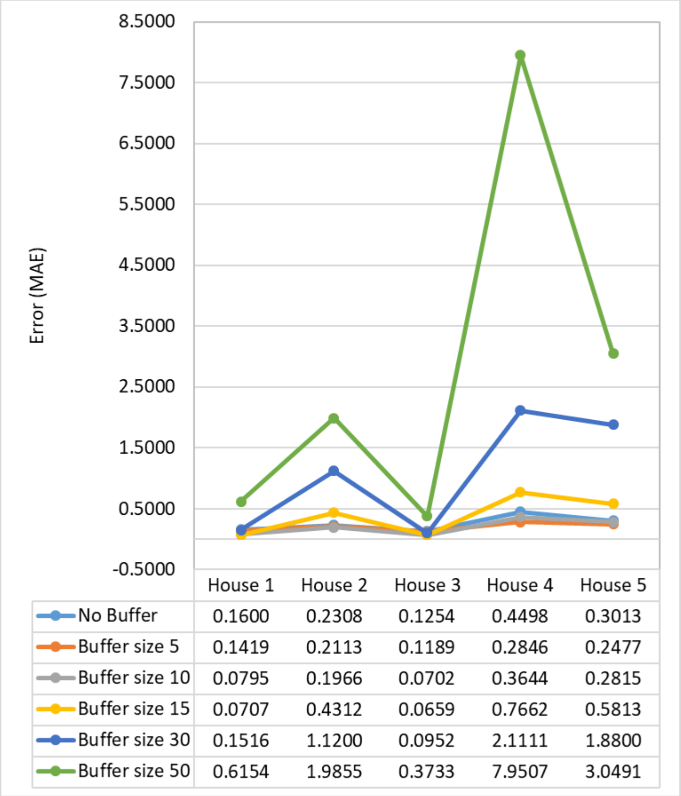
For houses with a lower presence of concept drift, such as houses 1 and 3, the buffer sizes 10 and 15 have the best performance, with size 15 having just slightly better performance than size 10. As the concept drift is not as prominent, having a medium-size buffer improves accuracy as the model sees difficult patterns several times. House 2 is somewhat similar to houses 1 and 3; lower errors are achieved for buffer sizes 5 and 10. It can be observed that for all houses, irrelevant to the degree of concept drift, large buffer sizes, such as 30 and 50, lead to performance degradation.

Fig. 5.12 examines the impact of the buffer size on the computation time: it shows the average training time per day for each of the considered buffer sizes. As expected, increasing the buffer size results in longer training time. Overall, buffer size 10 achieves good results for all houses (although not the best for all houses) irrelevant of concept drift presence while requiring only slightly longer training time than lower batch sizes.

5.3.6 Training time analysis

One of the common ML approaches for dealing with new data and changing patterns is to re-train the model daily or weekly with all data [74]. Assuming that changes between consecutive days or weeks are small, this approach can lead to good accuracy, but the computation burden is extensive.

This section compares the computation time needed for daily re-training a conventional offline model with the training time of the proposed Online Adaptive RNN. Although experi-



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

Figure 5.11: The impact of the buffer size on model's accuracy.

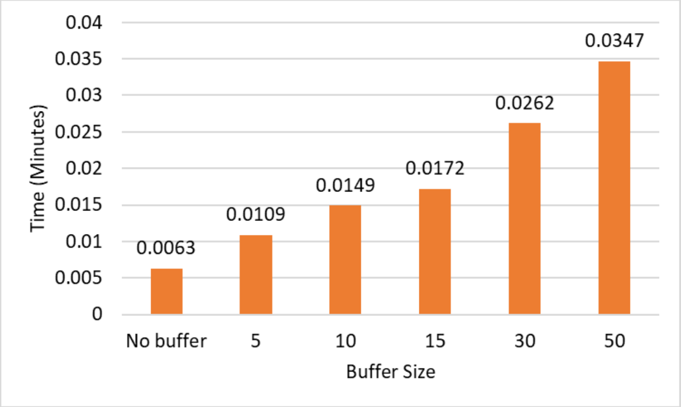


Figure 5.12: Average training time for different buffer sizes.

ments were conducted with a relatively small number of readings, they illustrate the advantages of the proposed approach. Nevertheless, with an increased number of readings, the training time for offline LSTM will increase with each day, while the time for Online Adaptive RNN will remain quite consistent.

The RNN and Tuner are the most computationally expensive components of the online adaptive RNN model. The computation time of the tuner module is dependent on the number of times the concept drift appears, despite the fact that the Bayesian method has a high time complexity. For data with a low number of concept drift occurrences , the tuner is called fewer times, resulting in a faster computation time.

Experiments

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

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

As the diversity of data among homes can result in different training times, the experiments were repeated five times with different homes, and the averages are reported. Two variants of the offline LSTM training were considered: with and without tuning. The hyperparameters considered for tuning and the tuning approach are the same as in subsection 5.3.2.

The variant with tuning leads to higher accuracy, but re-tuning each day is extremely computationally intensive while it may not always be necessary. Consequently, results without tuning are also reported. As described in Section 5.1, Online Adaptive RNN is tuned as needed.

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

Results

Figure 5.13 depicts the average training time for conventional LSTM with tuned hyperparameters and Figure 5.14 shows the average training time for the same model but without tuning. The horizontal axis represents the increment in the number of samples and each data point represents the training time for a specific day. It can be observed that starting from 2000 samples results in at least double the training time compared to starting from 1000, which is to be expected as the offline LSTM is re-trained each day with complete data set. Examining consecutive days (0, 24, 48, and 96 on the horizontal axis), the slow gradual increase can be observed for each starting point and for training with and without tuning. As expected, training times for tuned models are a magnitude larger than for models without tuning.

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

As the variations among days are minimal, there is no need to show this in the plot. Comparing daily training time of Adaptive Online RNN (0.0149 minutes) with time for the offline LSTM presented in figures 5.13 and 5.14, it can be observed that Adaptive Online RNN takes only a fraction of time needed by the offline models. This further demonstrates the benefits of the proposed approach.

5.4 Discussion

Overall, Online Adaptive RNN outperformed the traditional offline LSTM as well as five other online algorithms. Although Online Adaptive RNN employs a traditional LSTM as its base

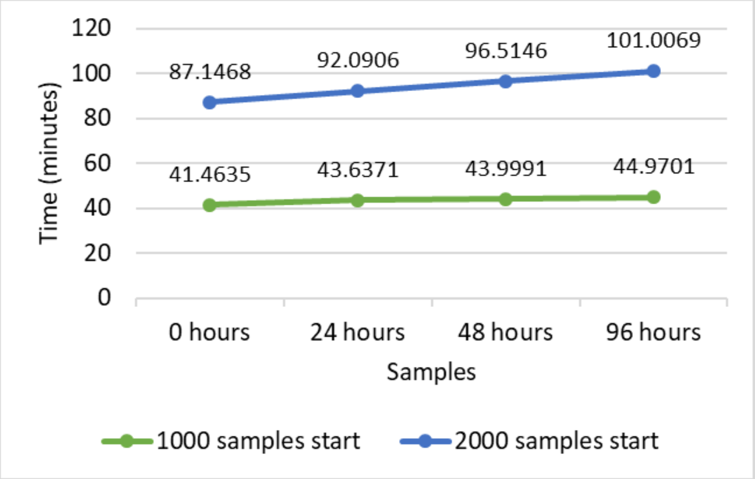


Figure 5.13: Average training time for conventional LSTM with hyperparameter tuning

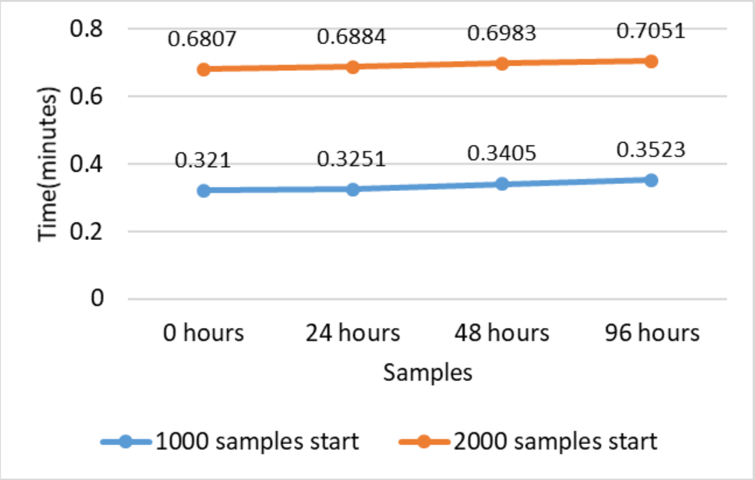


Figure 5.14: Average training time for conventional LSTM without hyperparameter tuning

learner, the unique architecture makes it capable of learning in an online manner and enables it to achieve higher forecasting accuracy than the LSTM itself. Moreover, online learning makes the proposed approach well suited for practice because there is no need for periodical re-training with new data, which is required for offline models.

For each of the five considered homes, Online Adaptive RNN performed better than LSTM (Fig. 5.7): the conventional LSTM was not able to learn well in the presence of concept drift. This highlights the need to evaluate the ability of load forecasting algorithms to handle concept drift. Variability of accuracy among homes demonstrates the necessity of considering different data sets when comparing machine learning approaches for energy forecasting.

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

In offline learning, especially with neural networks, hyperparameter tuning is essential for achieving high accuracy. Our approach adds the hyperparameter tuning capability to the online model through the addition of the tuning module. The experiments show that this online tuning significantly improves forecasting accuracy as illustrated in Fig. 5.10.

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

Chapter 6

Synchronous Federated Learning for Load Forecasting

Although conventional centralized solutions have shown great results, they require transferring all data to a centralized location, which results in significant network traffic [96]. Moreover, a centralized ML not only requires sharing local data with the centralized systems imposing security and privacy concerns, but also makes complying with stringent data regulations [12]. As the number of smart meters grows, training an individual ML model for each smart meter becomes computationally expensive and even infeasible.

Federated Learning (FL) [124] presents a possible solution to these challenges by decoupling the ability to do train the ML model from the need to store the data on the cloud or another centralized system. In FL, a global ML model is shared across many devices: each device receives a copy of the global model and improves it by learning from local data. Then, instead of raw data, the updated parameters of the local models are sent to the server to be aggregated and incorporated into the global model. FL is a major shift from a costly central ML system to a distributed ML approach that can exploit numerous distributed computational resources. This learning technique enhances data privacy because the data stay on the local devices and reduces network traffic by only exchanging model updates as opposed to raw data

[125]. However, FL assumes that a single model can capture patterns across diverse clients. In load forecasting, this approach would entail a single global model capable for generating individual load forecasts for each smart meter. As a diversity of patterns among meters is large, a single model may encounter difficulties in capturing this diversity and may lead to inferior forecasts.

Consequently, this section proposes FL for load forecasting with smart meter data. Two different FL techniques are examined: Federated Stochastic Gradient Descent (FedSGD) and Federated Averaging (FedAVG). Moreover, we examine a dynamic environment in which some devices join the federation after the training and only use the already trained model for forecasting without participating in the training.

6.1 Federated Learning for Load Forecasting

Conventional ML for load forecasting collects the readings from individual smart meters in a data center or another centralized system and then trains ML models on that centralized system. In contrast, Federated Learning (FL) trains an ML model across multiple data holders such as decentralized nodes and edge devices while keeping data local and transfers only the model updates to the central server. Consider K data holders $F_1, ...F_K$ wishing to train a single ML model by consolidating their respective data $D_1, ..., D_K$. A centralized method brings all data together in the centralized location and uses $D = D_1, \cup \cdots \cup, D_K$ to train a single model. In federated learning, data holders $F_1, ...F_K$ collaboratively train a model M_{FED} without data holders F_i sharing their data D_i with others, under the condition that the performance of federation P_{FED} remains very close to the performance of the single central model P_{SUM}. This condition can be stated as:

$$|P_{FED} - P_{SUM}| \leq \omega \tag{6.1}$$

where ω is a small non-negative real number [125].

For load forecasting, we train a deep neural network with multiple smart meter data distributed over the network without explicitly exchanging data samples with the central server. None of the local data are ever transmitted between parties; only model-related parameters are shared. The federated learning process is described in the next subsection, followed by the local preprocessing and the considered FL algorithms.

6.1.1 Federated Learning Process

Fig. 6.1 depicts FL process for load foresting: a single model is trained collaboratively over distributed smart meter data. LSTM model has shown great successes in load forecasting; therefore, it is used here as well. While we consider two FL strategies, FedSGD and FedAVG, the overall FL process is the same with one round of training consisting of the following steps:

Step 1: If this is the first training round, the server initializes the global LSTM weights; otherwise, the server proceeds with the weights obtained from the previous training round. A random subset of the smart meter devices is selected for the current training round, and the server sends a copy of the global model to those selected devices. Only a subset of devices participate in each training round as this improves convergence [125].

Step 2: The devices receive a copy of the global model and train it using only the local data. To enable training with LSTMs, this local data is preprocessed and transformed into a suitable form: as described in Subsection 6.1.2, the preprocessing is the same for both algorithms FedSGD or FedAVG. On the other hand, specifics of the local training depend on the type of FL, FedSGD or FedAVG.

Step 3: The devices that participated in the local training send the updated model parameters to the server. As each device trained the model with different local data, the updated parameters are different among devices.

Step 4: The server receives the local model parameters and aggregates them to construct an improved global model. In this study, we consider two main aggregation approaches, FedAVG and FedSGD. The process repeats from step 1 until convergence.

Step 5: After the model converges, the server sends the trained global model to all participants.

Step 6: The participants replace the out-of-date local models with the updated one received from the server and are now ready to carry our load forecasting. Note that when the trained model is used for forecasting, local data is still prepared with the same preprocessing technique as the one used during the model training in Step 2.

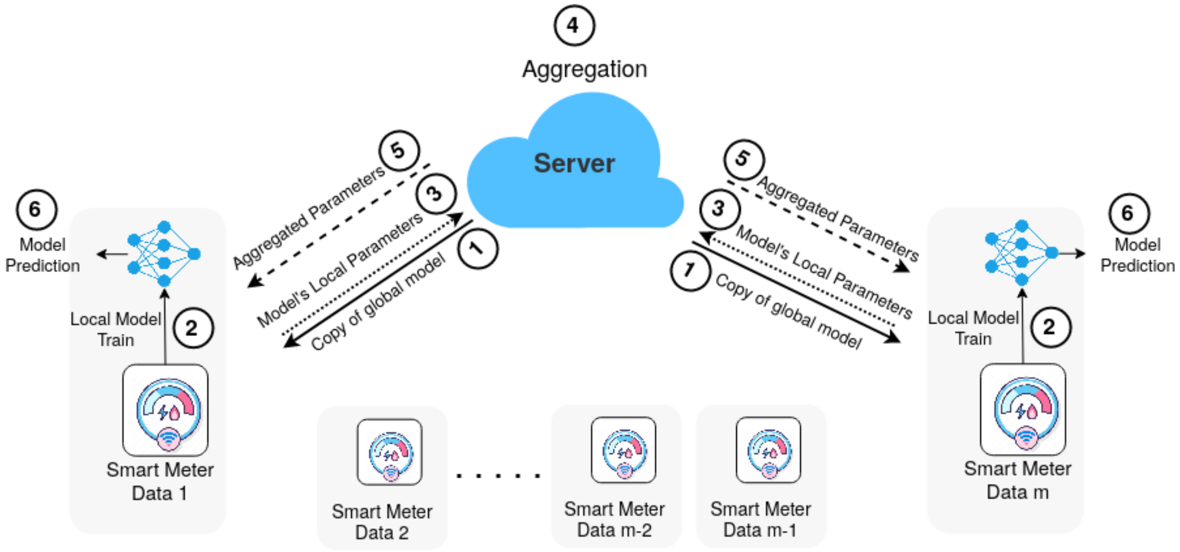


Figure 6.1: Federated Learning for Load Forecasting

Conventional ML typically assumes independent and identically distributed (IID) variables [126, 97, 127]; however, data collection in the FL setup violates this assumption. The non-IID data in load forecasting are caused by the smart meters corresponding to particular users or groups of users with different preferences and behavioral patterns. Alternatively stated, smart meters typically collect data in different contexts, leading to significant differences in the data distributions and patterns among them. Although this imposes challenges for ML training, the proposed FL for load forecasting achieves better accuracy than the conventional ML approaches as will be shown in the evaluation.

6.1.2 Local Preprocessing

Steps 2 and 6 of the federated learning process, as mentioned in Subsection 6.1.1, both involve preprocessing. The preprocessing is significant in conventional ML, but it is even more important in federated setting as data at the local nodes may contain different distribution, patterns, and data scales, which makes the training process more difficult.

In conventional ML, data are scaled to reduce the large features dominance and improve the convergence; however, while conventional ML scales after aggregating all data, here we scale data individually on each node. If smart meters observe similar patterns but with different magnitudes, this individual scaling will make these load profiles more similar to each other and facilitate training. Specifically, Min-Max normalization is applied at each node individually. It transforms x to x' without distorting differences in the ranges of values as follows:

$$x' = \frac{x - Min(x)}{Max(x) - Min(x)} \tag{6.2}$$

where x is the original feature value, $Min(x)$ and $Max(x)$ are the minimum and maximum of that feature on the considered node, and x' is the normalized value in the range of 0 to 1 [128].

Next, data must be transformed into a form suitable for modeling temporal dependencies and for use with LSTMs. As common when RNNs are used with sensor data, the sliding window technique is applied [128]: it converts time-series data into windows of size $w \times f$ where w is the number of time steps contained in the window and f is the number of features. The first window contains the first w smart meters readings. Then, the window slides for s times steps, and the second window contains readings from $s + 1$ to $s + w$, and so on.

Here, features include the load data reading from smart meters and any other features generated from the reading date/time or from meteorological information. In experiments, we include features such as the day of the week, the hour of the day, and the day of the year. Although sequential models, such as LSTM, are able to extract temporal patterns, these additional features assist the model with capturing the date and time-related patterns.

6.1.3 Federated Learning Algorithms

In general, the federated learning objective for K devices can be described in a form of the optimization problem:

$$\min_{w} \ell(w) = \sum_{k=1}^{K} \frac{n_k}{n} L_k(w) \tag{6.3a}$$

$$\text{where } L_k(w) = \frac{1}{n_k} \sum_{i \in P_k} \ell_i(w) \tag{6.3b}$$

where $\ell(w)$ is the global model's loss function, $L_k(x)$ is the loss of the k^{th} device, and $\ell_i(w)$ is the loss for sample i. P_k, $k \in \{1, ..., K\}$ denotes a partition of data points stored on the device k, $n_k = |Pk|$ is the size of P_k, and $n = \sum_{k=1}^{K} n_k$ is the size of all data on all devices. The objective is to find w which minimizes the loss $l(w)$ over the distributed data P with the assumption that P_i may be very different from P_j for devices i≠j.

Here, we consider two ways of solving this optimization problem: Federated Stochastic Gradient Descent and Federated Averaging.

Federated Stochastic Gradient Descent (FedSGD): In FedSGD [129], a distributed stochastic gradient descent algorithm is applied in the federated setting to optimize the model collaboratively: Algorithm 4 depicts the steps of FedSGD. For each communication round (Line 3), a subset of devices S_t is selected randomly (Line 4) to receive a copy of the global model (Line 5). Then, each client device k from S_t (Line 6) takes one step of the gradient descent g locally on the current model w_t using its local data (Line 7). Procedure *GradientStep* (Line 11) is executed on clients: it calculates the gradient $\triangledown$ over local data P_k and returns it to the server. Then, the server aggregates the received gradients by taking the weighted average of the clients gradients proportional to the number of training samples and the global model is updated using this weighted average and learning rate η, as shown in Line 8. The process repeats from Line

Algorithm 4 Federated Stochastic Gradient Descent (FedSGD)

```
 1: Server Execution:
 2: Initialize global model weights $w_0$, and learning rate $\eta$
 3: for each round t=1,2,... do
 4:     $S_t \leftarrow$ random set of $m$ clients
 5:     Send global model to $S_t$ clients
 6:     for each client k $\in S_t$ in parallel do
 7:         $g^k_{t+1} \leftarrow$ GradientStep($k$,$w_t$)
 8:     $w_{t+1} \leftarrow w_t - \eta \sum_{k \in K} \frac{n_k}{n} g^k_t$
 9: Send the model to all participants

10: Client Execution:
11: procedure GRADIENTSTEP($k$,$w$)
12:     $g \leftarrow \nabla \ell(w)$ over $P_k$
13:     return $g$ to server
```

3 until convergence. Finally, the trained model is broadcasted to all participants (Line 9).

Federated Averaging (FedAVG). Like FedSGD, FedAVG also solves the defined FL optimization problem. In contrast to FedSGD, in which the local clients take one step of the gradient descent and exchange the gradients without applying them to the local models, FedAVG allows the devices to update the local model by iterating through weight updates $w \leftarrow w - \eta \nabla \ell(w)$ multiple times before sending the updated model weights to the server. Algorithm 5 presents the FedAVG process. Each round of FedAvg starts the same as FedSGD by randomly selecting a subset of devices S_t and broadcasting the model to the chosen devices (lines 4 and 5). The selected devices train in parallel their local models with the local data for multiple epochs (Line 7). Procedure *ClientUpdate*, Line 11, shows the local model update process: the local data P_k are divided into the batches B of size s_b (Line 12) and the local device trains the received model for multiple epochs with created batches as shown in Lines 13 to 15. Each device from S_t sends the new local weights to the server (Line 16), and the server updates the global model by calculating the weighted average of the received local weight as shown in Line 8. As in FedSGD, the process is repeated from Line 3 until convergence and, finally, the trained model is broadcasted to all participants (Line 9).

FedSGD is an efficient method that guarantees the convergence in FL settings and is barely

Algorithm 5 Federated Averaging (FedAVG)

```
Server Execution:
Initialize global model weights $w_0$
for each round t=1,2,... do
    $S_t$ ← random set of $m$ clients
    Send global model to $S_t$ clients
    for each client k ∈ $S_t$ in parallel do
        $w^k_{t+1} \leftarrow$ ClientUpdate($k$,$w_t$)
    $w_{t+1} \leftarrow \sum_{k \in K} \frac{n_k}{n} w^k_t$
Send the model to all participants

Client Execution:
procedure CLIENTUPDATE($k$,$w$)
    $B$ ← split $P_k$ into batches of size $b_s$B
    for each local epoch $e < E$ do
        for batch $b \in B$ do
            $w \leftarrow w - \eta \nabla \ell(w)$
    return $w$ to server
```

influenced by the non-IID problem under adequate training parameters [130]; however, it requires a large number of training rounds to produce good results [129]. FedAvg is an FedSGD alternative that shows significant improvement in communication and time efficiency [130]. The basic idea behind FedSVG is that if all local devices start from the same initialization parameters, averaging the weights is strictly equivalent to averaging the gradients and, therefore, does not necessarily harm the averaged model performance. Nevertheless, it is shown that heterogeneity of data slows down the convergence of FedAVG [131]. Xiang *et al.* [131] proved that by having an adaptive learning rate, the model can converge on non-IID data; consequently, we use adaptive learning rate as well.

Heterogeneity of smart meter data is high as these data are collected in different contexts and influenced by diverse human behaviour resulting in different load patterns and distributions. FedSGD is well suited for this context as it has shown promising results in learning from heterogeneous data [130]; therefore, it is examined here with respect to load forecasting. To reduce communication rounds and time complexity, we examine FedAVG, but the standard SGD is replaced with an Adam optimizer which embraces adaptive learning rate to accelerate convergence and improve the learning capability on non-IID data [131].

6.2 Evaluation

This section first introduces the dataset and evaluation metrics. Next, the performance of the proposed FL methods is compared with central training and individual LSTM models. Then, we examine a dynamic environment in which some devices do not participate in training, but use the trained model for local load forecasting. Finally, FedSGD and FedAVG are compared in terms of convergence, and the overall results are discussed.

6.2.1 Dataset and Evaluation Metrics

This study is performed in collaboration with London Hydro, a local electrical distribution utility. Federated learning for load forecasting will enable London Hydro to provide large-scale forecasting services to its residential consumers and, consequently, increase return on investment from the smart meter infrastructure. London Hydro provided real-world data for the evaluation of the presented approaches through Green Button Connect My Data (CDM), a platform for secured sharing of energy data with the consumer's consent. The evaluation was conducted with 19 residential consumers, each one containing hourly energy consumption for three years, resulting in 25,560 readings per households, or 485,640 readings in total. As these readings are also used for the billing purposes, they are highly reliable and have the same reading frequency and the number of samples for each household.

Additional features including the day of the year, the day of the month, the week of the year, the day of the week, and the hour of the day were devised from the load reading date/time to assist with modelling daily, monthly, and weekly patterns.

Diversity among consumers in terms of load profiles is large: for illustration, Fig. 6.2 depicts the load data for the three households. It can be observed that load patterns, as well as load magnitudes, vary greatly among consumers. This diversity makes it difficult for a single model to capture all patterns among consumers. Therefore, many studies train a single model per consumer [38]; however, here we examine devising a single model for all consumers through federated learning.

For the evaluation, each individual household dataset is split into 70% for training and 30% for testing. This split remains the same for federated learning experiments as well as for conventional ML experiments, centralized and individual models for each meter, conducted for the purpose of comparison.

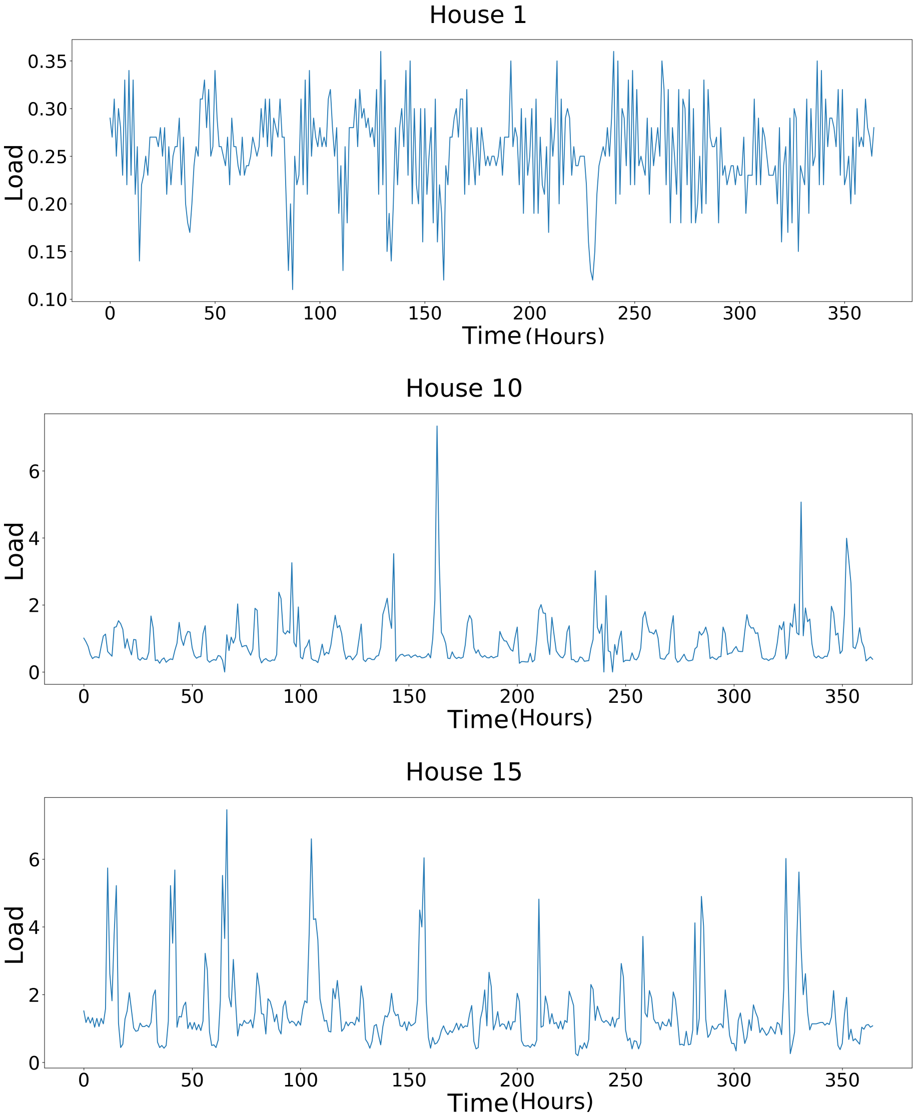


Figure 6.2: Electricity load examples for Home 1, Home 10, and Home 15

The model performance is compared with two metrics: Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE). RMSE measures the deviation of the residuals (prediction errors); in other words, it measures how far the predicted values are form the observed (actual) values. MAPE is expressed as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2} \tag{6.4}$$

where y_t are the actual values, $\hat{y}_t$ are the corresponding predicted value, and N is the number of observations.

The MAPE metric measures the average absolute error and is calculated as follows:

$$\text{MAPE} = \frac{1}{N}\sum_{t=1}^{N}\frac{|y_t - \hat{y}_t|}{y_t} \tag{6.5}$$

Note that RMSE is a scale dependent error metrics: the same RMSE value has a different meaning for different data magnitudes. On the other hand, MAPE expresses errors in terms of percentages and, thus, is better suited for comparison among data sets.

6.2.2 Comparison of FedAVG and FedSGD with Individual LSTMs and Central Model

In this subsection, the proposed FedAVG and FedSGD for load forecasting are compared to the individual LSTMs and the central model. The individual LSTMs approach refers to training an individual LSTM for each household. This approach does not require any exchange of data with the central server, but the drawback is that there is a large number of models that need to be maintained individually. As individual models are trained for a specific client, they are personalized models and, thus, are good at capturing intricacies of the specific clients. A comparison of FedAVG and FedSGD with individual LSTM will examine how good the federated learning strategies are in training a single model to capture diversities among clients.

In the central model, data from all households are combined to form a coherent dataset used to train a single LSTM. In this approach, all data must be transferred to the central location. By comparing FL strategies to the central model, the ability of the proposed FL models to learn from heterogeneous data will be determined.

To keep the comparison fair, all models, federated, central, and individual, use the same architecture consisting of one layer LSTM with 32 hidden units. Tuning hyperparameters individually for each of the considered models has a potential to increase accuracy; however, this would result in massive computational cost. Moreover, in FL, hyperparameter tuning is still an open challenge [126] because of the distributed environment, FL-specific hyperparameters, and the network traffic associated with tuning.

For the federated learning strategies, FedAVG and FedSGD, six clients participate in each round of training. This number was selected as it allows for diversity of clients in each round while still including less then one third of clients per round. In FedAVG, one round of training on the client consists of five epochs, thus allowing clients to make reasonable learning steps before aggregation. Further tuning could improve federated learning results, but would select the parameters only for this specific combination of clients.

Forecasting one hour ahead

Table 6.1 shows the average test error in terms RMSE and MAPE for FedSGD, FedAVG, individual LSTMs, and the central model for one hour ahead forecasts. The two federated learning strategies, FedSGD and FedAVG, achieved lower errors than individual LSTMS and the central model. FedSGD achieved the lowest RMSE test error while FedAVG obtained the lowest MAPE test error. As RMSE for FedAVG is very close to RMSE for FedSGD, the overall better model is FedAVG because of its low MAPE.

Table 6.1 examines the average accuracy for all households, but we also need to investigate how models perform for individual households. Fig. 6.3 depicts MAPE errors obtained by the four approaches, for each house individually. It can be observed that FedAVG outperforms others approaches for all but 5 households. For those 5 households, the central model achieves only slightly better accuracy; however, the central model performs worse in most households making FedAVG an overall better model. It is also worth noting that FedSGD performance is very similar to that of the central model, with almost the same MAPE error. This confirms the findings from Table 6.1 with FedSGD obtaining very close average MAPE to the central model.

While Fig. 6.3 compares the accuracy of the considered algorithms in terms of MAPE, Fig. 6.4 does so in terms of RMSE. In terms of RMSE, all algorithms achieve similar accuracy for most houses, and there is no clear winner. Nevertheless, federated algorithms achieve similar accuracy to conventional ML while not requiring data sharing.

RMSE measures the standard deviation of errors (Equation 6.4) and because of squaring, it imposes high penalty on larger errors and is sensitive to outliers. In contrast, MAPE calculates the average percentage error (Equation 6.5). Moreover, MAPE expresses the error as a percentage while RMSE is scale dependent with the same unit as the measured value. Because magnitude of the energy consumption varies among houses, MAPE is better suited when comparing among houses. In terms of MAPE, FedSGD achieves better accuracy than the remaining algorithms as observed in Table 6.1 and Fig. 6.3.

An example of predicted versus actual values is shown in Fig. 6.5: it depicts the forecasts obtained by each of the four approaches for house 13. It can be observed that for the shown segment, the predicted values better match the actual values for FedAVG than for the other approaches. The remaining approaches, central model, individual LSTMs, and FedSGD, appear to obtain reasonable load forecasts but no distinction can be made regarding which one is better. Nevertheless, for this example, the best predictions are obtained by FedAVG, which corresponds to the observation seen with MAPE metrics shown in Table 6.1 or Fig. 6.3.

Table 6.1: Average RMSE and MAPE errors for all 19 houses: one hour ahead prediction

Error	FedAVG	FedSGD	LSTM	Central Model
MAPE	14.7522	16.7775	19.3123	16.8851
RMSE	0.6138	0.6084	0.6303	0.6200

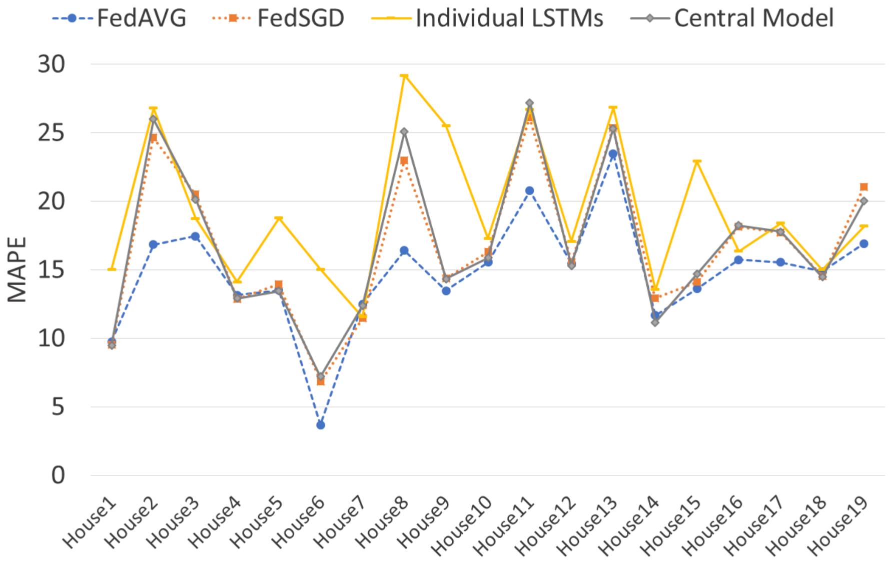


Figure 6.3: MAPE errors for FedAVG, FedSGD, LSTMs, and Central Model: one hours ahead prediction

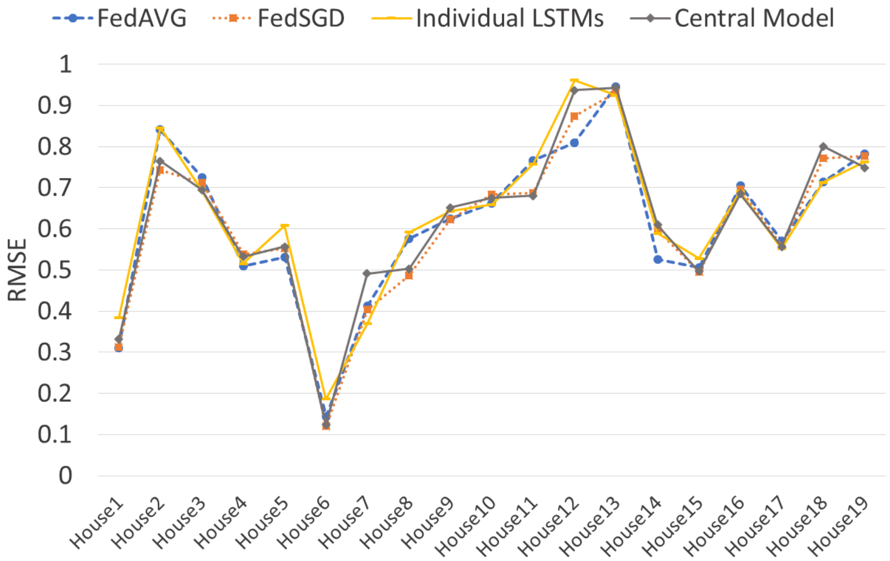


Figure 6.4: RMSE errors for FedAVG, FedSGD, LSTMs, and Central Model: one hours ahead prediction

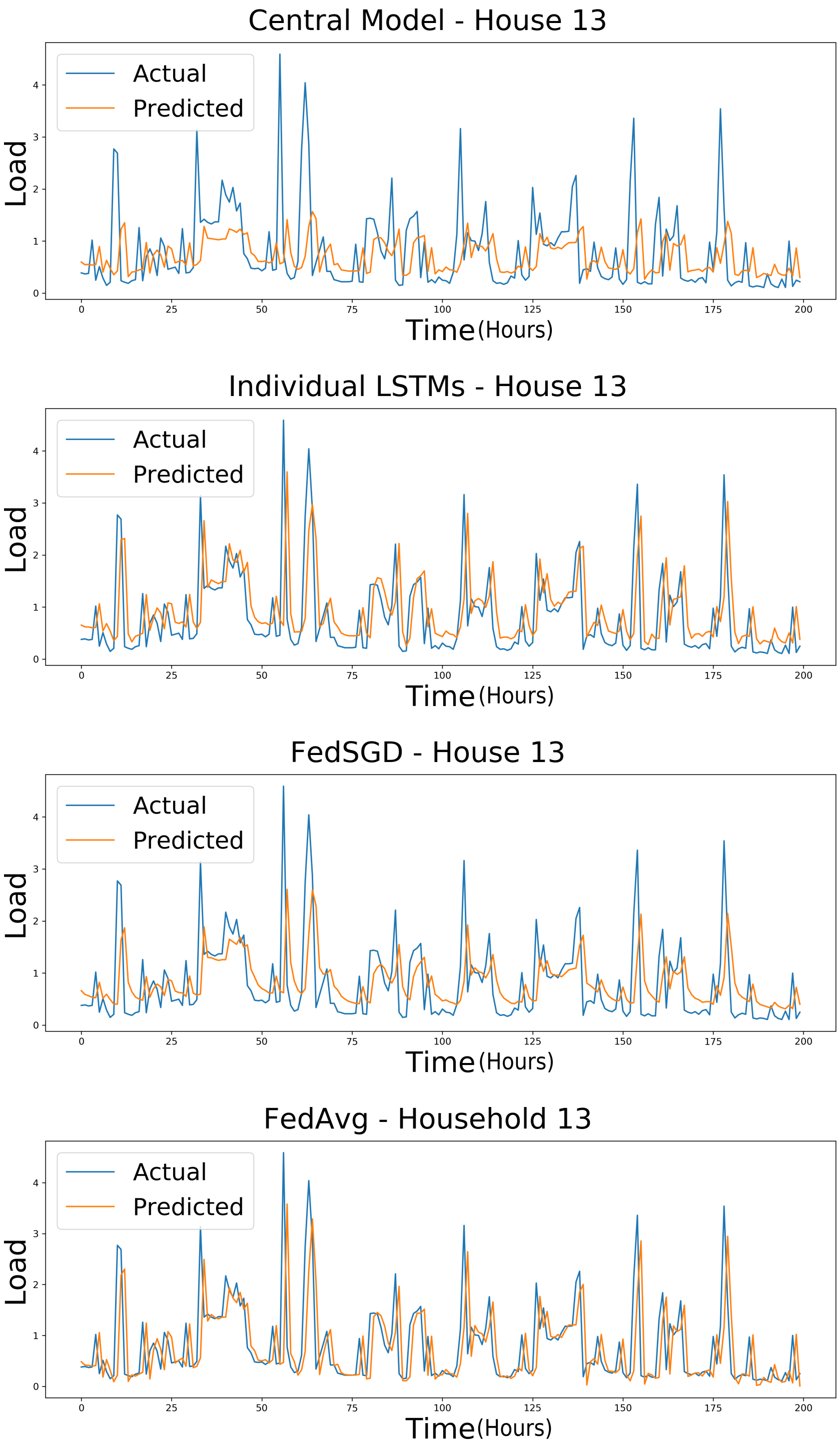


Figure 6.5: Actual versus predicted load for House 13: one hour ahead forecast

Forecasting 24 hours ahead

While Table 6.1, and figures 6.3 and 6.4 show results for one hour ahead prediction, Table 6.2 and figures 6.6 and 6.7 show results for 24 hours ahead forecasting. It can be observed from

Table 6.2: Average RMSE and MAPE errors for all 19 houses: 24 hours ahead forecast

Error	FedAVG	FedSGD	LSTM	Central Model
MAPE	17.3870	31.2705	20.5328	55.2167
RMSE	0.6868	0.6842	0.6439	0.6554

Table 6.2 that FedAVG achieved the best average accuracy in terms of MAPE while in terms of RMSE, the accuracy of FedAVG, FedSGD, LSTMs, and the central model was similar. Thus, FedAVG can be considered the best model for 24h ahead forecast. Comparing accuracy of 24h ahead forecasts, Table 6.2, with one hour ahead, Table 6.1, the average error is lower for shorter forecasting horizon, which is expected as it is, in general, easier to predict fewer hours ahead.

Fig. 6.6 shows MAPE values for each houses individually, for each of the four approaches. It can be observed that FedAVG and individual LSTMs achieve lower accuracy than the other two methods for most of the houses. For a few houses, individual LSTMs achieved lower errors, but overall, FedAVG is better as its average MAPE is lower than for other approaches as can be seen from Table 6.2.

Comparing 24 hours ahead forecasting in terms of RMSE, Fig. 6.7, all four algorithms show similar accuracy for all houses. This matches the observation from Table 6.2, where all four algorithms have similar average RMSE. Nevertheless, as MAPE is better suited for data sets with different magnitudes, and FedAVG outperformed other approaches in terms of MAPE, FedAVG is the preferred algorithm.

Fig. 6.8 shows an example for 24 hours ahead forecast. For this house and for the shown forecasting segment, FedAVG gives predictions closer to actual values. This confirms the findings from MAPE comparison in Table 6.2: FedAVG achieves better accuracy then the other approaches for 24 hours ahead forecasts.

6.2.3 Evaluation in Dynamic Environment

This subsection examines if the model trained with federated learning can be used for the smart meters that did not participate in training. This represents a dynamic environment where some smart meters join the federation after the training is complete and only use an already trained

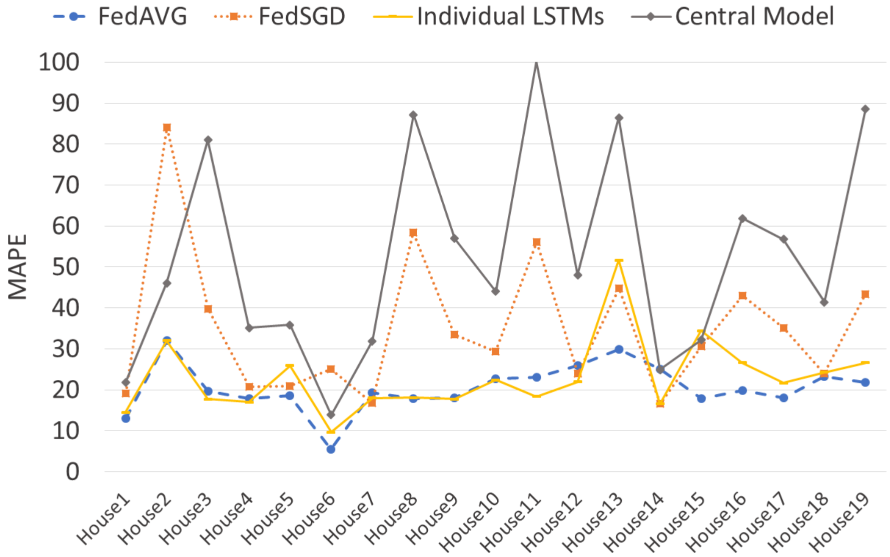


Figure 6.6: MAPE errors for FedAVG, FedSGD, LSTMs, and Central Model: 24 hours ahead forecast

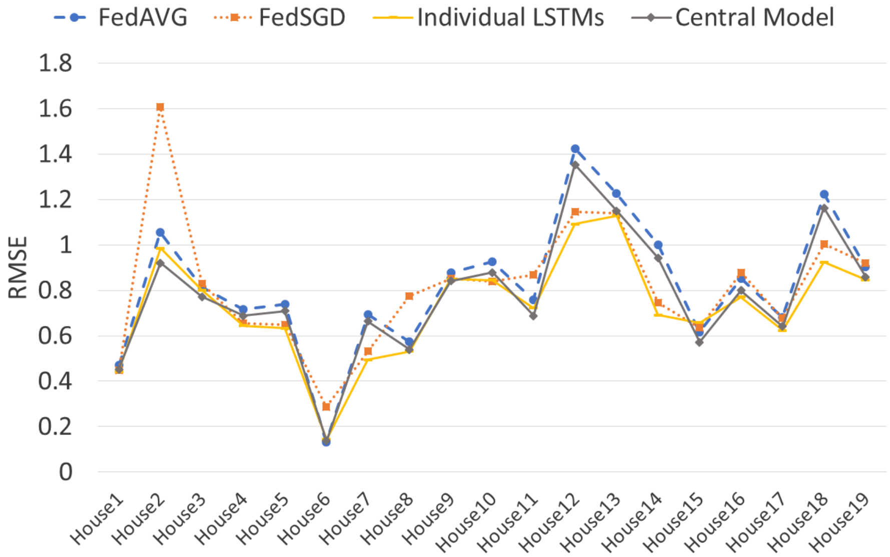
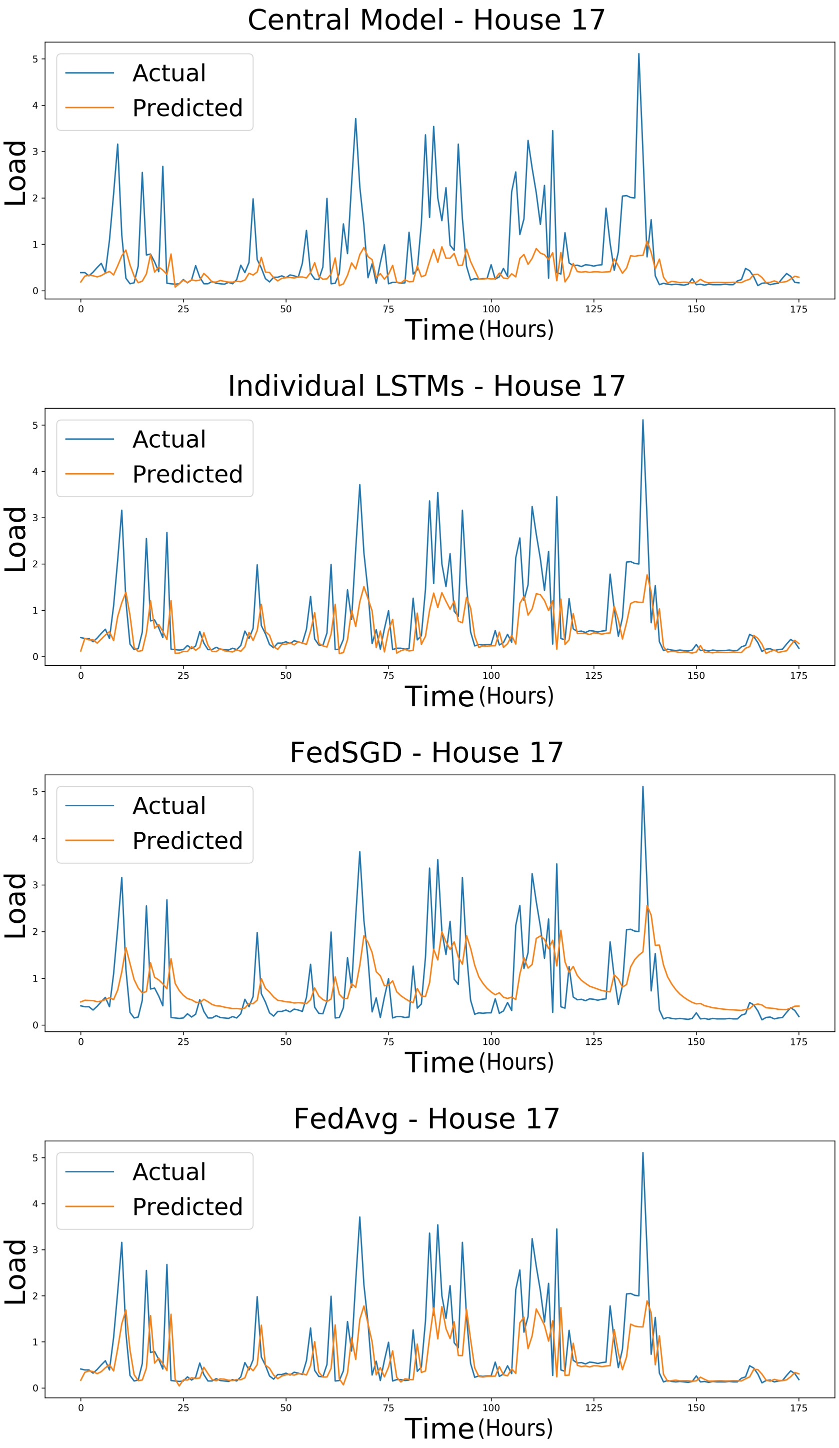


Figure 6.7: RMSE errors for FedAVG, FedSGD, LSTMs, and Central Model: 24 hours ahead forecast

model for forecasting.

For these experiments, the houses are divided into three groups: first 6 houses, second 6,



Central Model - House 17

Individual LSTMs - House 17

FedSGD - House 17

FedAvg - House 17

Figure 6.8: Actual versus predicted load for House 17: 24 hours ahead forecast

and remaining 7 houses. The ML model is first trained without the first group of houses. Then this model is evaluated of the first group of houses and results are compared to the accuracy achieved when all houses participating in training. The same is repeated for the second and the third group of houses. Figures 6.9 and 6.10 show the results in terms of MAPE and RMSE for one hour ahead forecasts. ”Absence” indicates that the specific group of houses did not participate in training, while presence signifies that all houses participated in training. Note that an individual LSTM cannot be considered here as it requires the use of the target house data for training. Also, for each group of houses, the evaluation is always performed only on that group of houses, although other houses participated in training. This is somewhat similar cross-validation, but instead of randomly selecting the validation set samples, a group of houses is assigned to the validation set.

In terms of MAPE, Fig. 6.9, all algorithms achieved better accuracy when all data are used for training, which is to be expected. However, even when a group of houses did not participate in training, the model was able to achieve the accuracy close to that of the model trained with all data. For the last 7 houses, FedSGD and FedAVG achieved almost exactly the same accuracy when those houses participated and did not participate in the training.

In terms of RMSE, Fig. 6.10, there was hardly any difference if the group of houses participated in the training or not. This demonstrates that for one hour ahead forecasting, federated learning strategies are successfully even for smart meters that did not participate in training. Figures 6.11 and 6.12 examine federated learning for 24 hours ahead forecasting in a dynamic environment: Fig. 6.11 shows MAPE while 6.12 shows RMSE metrics. As before, federated learning strategies do not exhibit overall performance degradation when some groups of houses do not participate in training.

6.2.4 Convergence and Computation Cost

In this subsection, the proposed FedAVG and FedSGD algorithms are assessed in terms of convergence. In general, the system converges if further training does not significantly improve

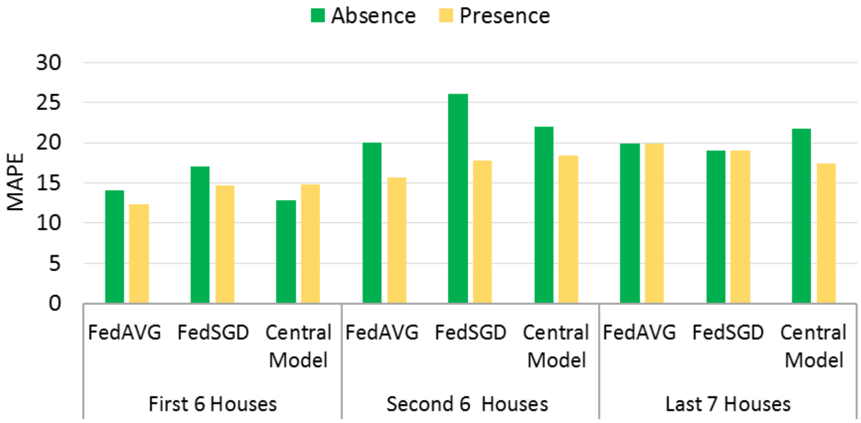


Figure 6.9: Dynamic environment: MAPE errors for FedAVG, FedSGD, and Central models: one hour ahead forecast

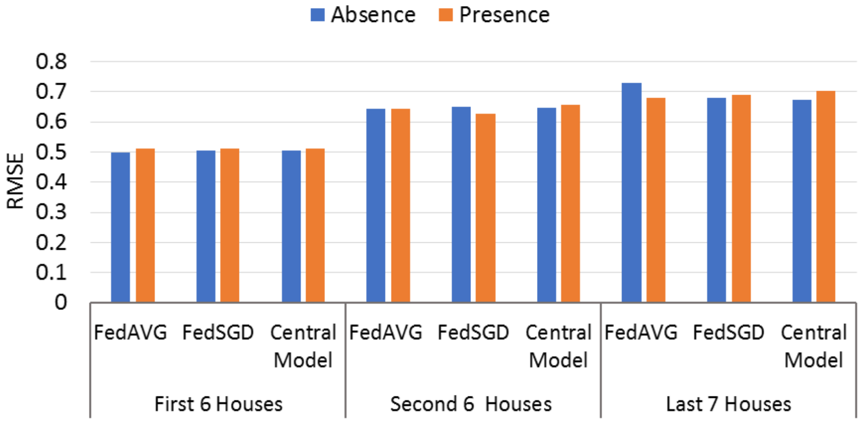


Figure 6.10: Dynamic environment: RMSE errors for FedAVG, FedSGD, and Central models: one hour ahead forecast

the model performance. For neural networks, convergence is examined in respect to epochs; however, in FL we are interested in the training rounds as they drive the communication between clients and the server. In FedSGD, each client performs only a single step of gradient descent in one training round. In contrast, FedAVG carries out several gradient descent steps on the client before communicating the updates back to the server.

To examine convergence, the same setup has been used as described in Subsection 6.2.2. The training errors for up to 150 rounds for the two algorithms, FedSGD and FedAVG, are shown in Fig. 6.13. As expected, both converge to similar errors, but FedSGD takes many more rounds to converge than FedAVG as FedAVG performs multiple steps in a single round.

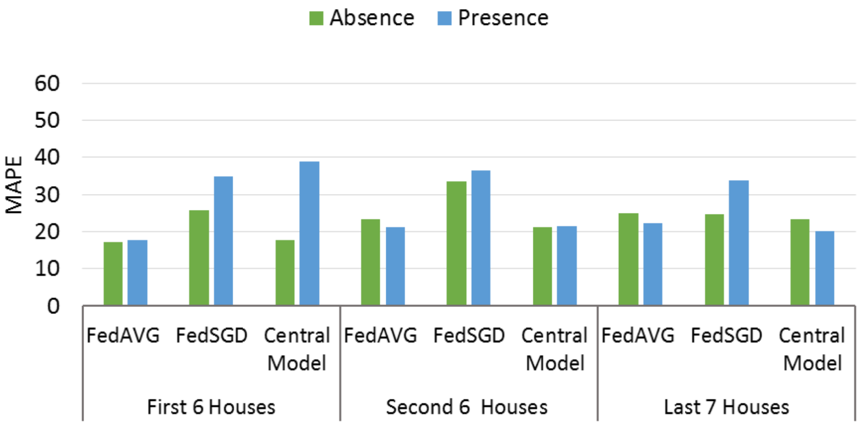


Figure 6.11: Dynamic environment: MAPE errors for FedAVG, FedSGD, and Central models: 24 hours ahead forecast

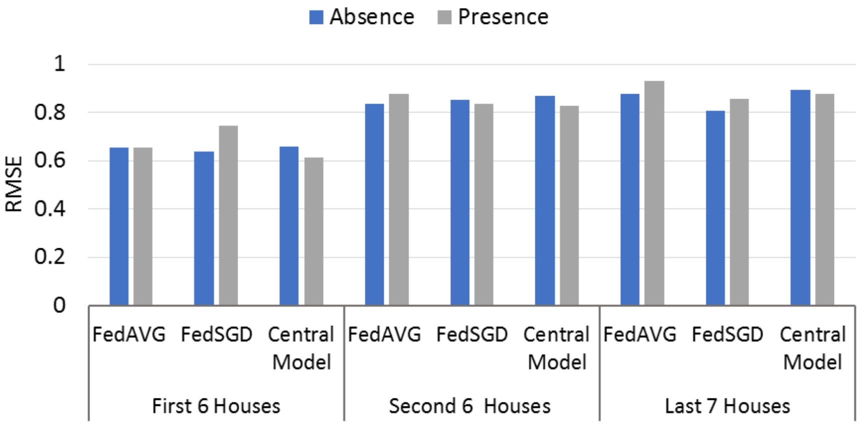


Figure 6.12: Dynamic environment: RMSE errors for FedAVG, FedSGD, and Central models: 24 hours ahead forecast

These training rounds are indicators of the communication and, therefore, it can be concluded that the network traffic in FedAVG is much lower than in FedSGD.

Furthermore, FedAVG computation is compared to that of the central model and individual LSTMs. With five epochs in each training round, FedAVG converged after the fifth epoch as can be seen from Fig. 6.14. The centralized model converged around the eleventh epoch, Fig. 6.15, while the convergence of individual models varied among different houses, Fig. 6.16, with majority converging before 15th epoch; thus, 15 epochs are considered for computation analysis. Note that convergence for the central model and individual LSTMs is shown in respect to epochs while for FedAVG, training rounds are used. This is because in FL, training

Table 6.3: Computation Comparison between FedAVG, Central Model, and Individual LSTMs

Algorithm	Batches/Epoch	Rounds	Clients	Epochs	Total	Time(sec)
FedAVG	80	5	6	5	12000	61.98
Central Model	80	N/A	19	15	22800	117.77
Individual LSTMs	80	N/A	19	15	22800	119.99

rounds represent the steps of the training process.

Table 6.3 compares the training computation of FedAVG, the central model, and individual LSTMs. Each approach has the same quantity of data available and uses the same batch size of 250. Therefore, each one processes 80 batches per epoch. For FedAVG, five rounds are needed, each one with six clients executing five epochs. This results in $80\ batches \times 5\ rounds \times 6\ clients \times 5\ epochs = 12,000$ runs over batches. For the central model and individual LSTM the concept of training rounds does not apply and we assume that 15 epochs are sufficient as the central model and most individual LSTMs converge with 15 epochs (figures 6.15 and 6.16). Each of these two approaches has to process data from 19 houses resulting in a total of $80\ batches \times 19\ houses \times 15\ epochs = 22,800$ runs over batches.

Architectures for all models have the same structure: LSTM with the same number of layers and the same number of parameters. Therefore, the time to process a single batch is

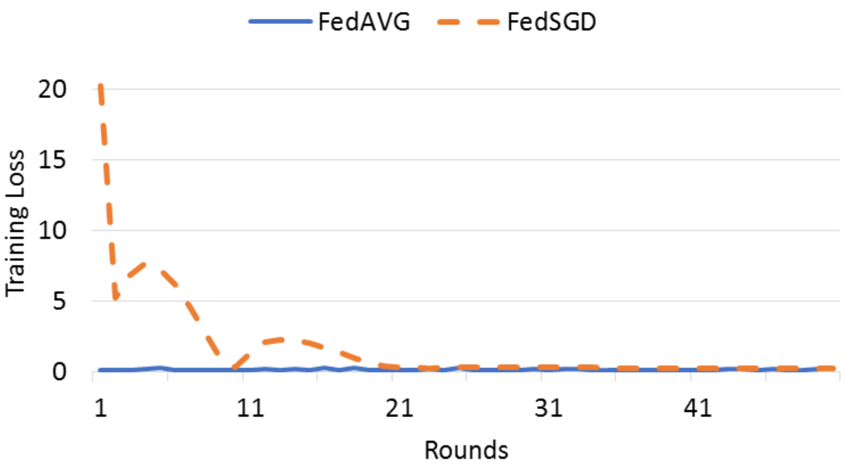


Figure 6.13: The training loss for FedAVG and FedSGD for 50 training rounds.

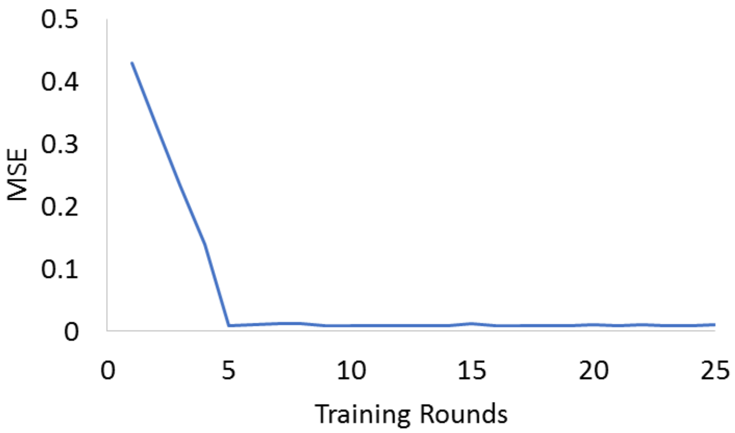


Figure 6.14: MSE for each FedAVG training round

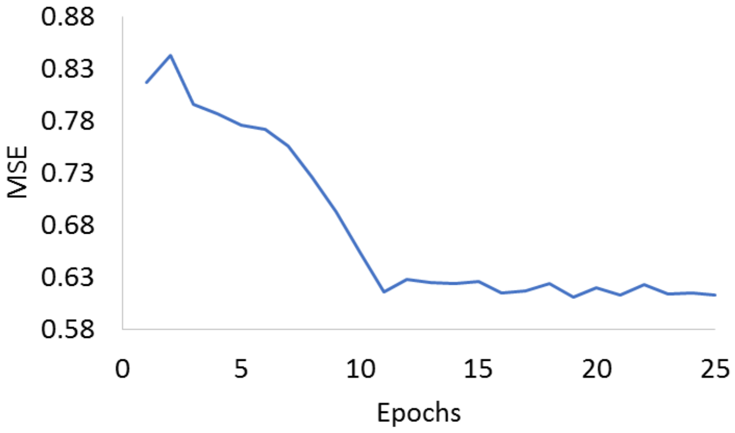


Figure 6.15: MSE for each epoch: central model

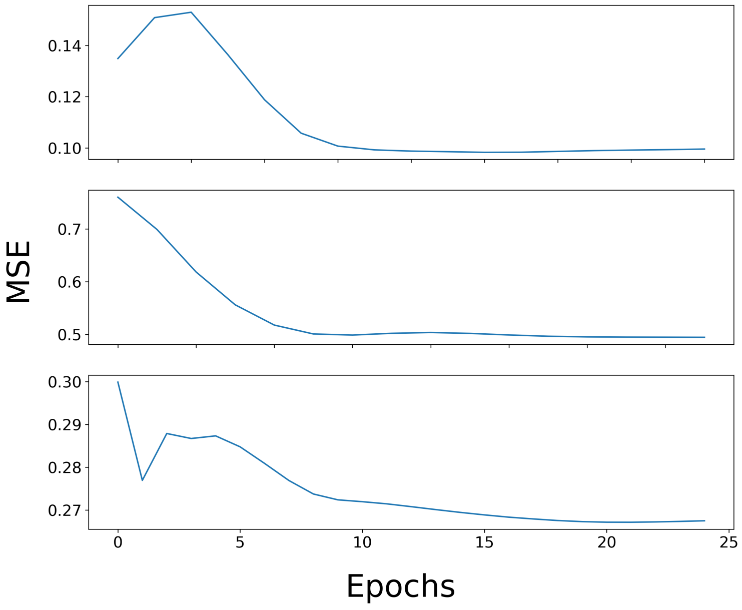


Figure 6.16: MSE for each epoch: three examples of individual LSTMs

similar across all morels. As FedAVG needs 12,000 runs while the other two approaches need approximately 22,800 runs, FedAVG will be significantly faster. Note that this ignores the fact that in FedAVG, in each training rounds, clients train in parallel reducing the computation time. With individual LSTMs, parallelization is possible while the central model requires sequential processing.

However, the main benefit of FL is that it achieves better accuracy than the other approaches while not requiring the clients to share their local data. Moreover, the clients can join the federation after the training is complete and FedAVG still achieves good forecasting accuracy as shown in Section 6.2.3.

6.3 Discussion

With increasing concerns regarding security and privacy, it is becoming more important to develop ML techniques capable of training ML models without requiring participants to share their local model. Federated learning is a step in this direction although it is still in its early stages, and it requires further improvements and examinations in different contexts. This study investigates the abilities of the FedSGD and FedAVG approaches in load forecasting.

Both algorithms, FedSGD and FedAVG, achieved comparable or better accuracy than a single central model or individual local models for one hour ahead forecasts, as shown in Table 6.1. For 24 hours ahead, FedAVG outperformed other algorithms in terms of MAPE while in terms of RMSE there was very little difference among algorithms. Overall, FedAVG was the best algorithm as it is able to achieve high accuracy without requiring the clients to share their local data.

In addition to examining the overall error, it is important to consider performance on individual houses. It terms of MAPE, FedAVG performed better than FedSGD for one hour and 24 hour ahead forecasting, as observed from figures 6.3 and 6.6. Moreover, FedAVG required fewer training rounds than FedSGD (Fig. 6.13).

Our experiments from Subsection 6.2.3 also show that the trained model can be used with a good success for smart meters that did not participate in the training. This is important in scenarios when new smart meters are added to the federation or when there are very little data from some meters.

RNN-based models, including LSTMs, have been outperforming other architectures for load forecasting on individual smart meters [8, 15] and this study demonstrated that FedAVG achieves similar or better results than individual LSTMs. Moreover, with individual models, there must be sufficient data from each individual household to train the model. In contrast, FedAVG is even successful for households that did not participate in training as shown in Subsection 6.2.3.

To further examine the performance of FedAVG, we compare it to simple shifting and the FL approach proposed by Taïk *et al.* [96] in Table 6.4. Simple shifting for one hour ahead forecasting uses the consumption at the current time step t as the forecast for the next time step $t + 1$. For 24 hours ahead forecasting, values from last 24 hours serve as the forecasts for the next 24 hours. The work of Taïk *et al.* [96] is the most related work to ours, as it also employs federated learning for load forecasting. Similar to our work, they use the FedAVG aggregating strategy, but, while we employ an adaptive learning rate, they use a fixed learning rate. Moreover, their study only considers one hour ahead forecasting.

As seen from Table 6.4, our FedAVG technique achieved better results than Taïk *et al.* [96] in terms of MAPE and RMSE for both one hour ahead and 24 hours ahead forecasting. Simple shifting archived better accuracy than FedAVG for one hour ahead forecast; however, for 24 hours ahead, simple shifting achieved very poor results in comparison to our FedAVG or the work of Taïk *et al.* [96]. Consequently, FedAVG is an overall better approach.

Table 6.4: MAPE and RMSE for FedAVG, Simple Shifting, and Taïk *et al.*: one hour ahead and 24 hours ahead prediction

	MAPE(%)		RMSE	
Algorithm	one hour ahead	24 hours ahead	one hour ahead	24 hours ahead
FedAVG	14.7522	17.3870	0.6138	0.6868
Taïk *et al.* [96]	19.7516	21.9798	0.6569	0.7358
Simple Shifting	13.0784	42.5700	0.5555	1.6719

Chapter 7

Asynchronous Adaptive Federated Learning for Load Forecasting

FL represents a shift from centralized to distributed ML, nevertheless, traditional FL employs a synchronous protocol [132]: at each iteration, the server distributes the global model to a subset of clients and then waits for all clients to complete their training before collecting all updates and aggregating them. This is a hindrance due to device heterogeneity and network unreliability. Asynchronous FL is a newly emerged FL method that allows the server to aggregate parameters without waiting for the lagging devices [101]; however, it is still in its early stages.

Another FL challenge is non-IID data (independent and identically distributed). Smart meters collect data corresponding to users with different preferences and behaviours leading to significant differences in the data distributions and patterns. The presence of non-IID data degrades the FL performance, brings instability to the training process, and results in a higher number of rounds required for convergence [104].

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

7.1 FedNorm

This section describes the proposed FedNorm, an asynchronous approach for learning from distributed smart meter data when some of the clients are unable to participate in the training process due to network instability or when the lagging clients do not complete their training timely. As seen in Fig. 7.1, clients' updates are merged into the global model at different time steps; for example, Client 1 changes are merged at t_1 while Client 2 changes are added at t_2. The theoretical analysis shows that the diversity of the node contributions in FL convergence can be measured by the local gradient of each node and the global gradient [104]. This motivates us to measure the contribution of participating nodes by assigning the weights for participating nodes based on the similarity between local weights w_i and global weight w together with the magnitude of local objective functions $\psi_k(w)$. An intuitive weighting design should follow the notion that nodes with larger contributions in reducing the global objective function $L(w)$ deserve higher impact in each global round. Therefore, our process for assigning adaptive weights includes three steps:

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

$$\lambda_k = \overbrace{\|w - w_k\|}^{\alpha_k} * \overbrace{(\psi_k(w) - \sum_{k=1}^{K} \frac{\psi_k(w)}{K})}^{\beta_k} \tag{7.1}$$

Here, α_k represents the degree of similarity between the global and local models, while β_k defines loss contribution of each client including contribution direction. The α_k is calculated

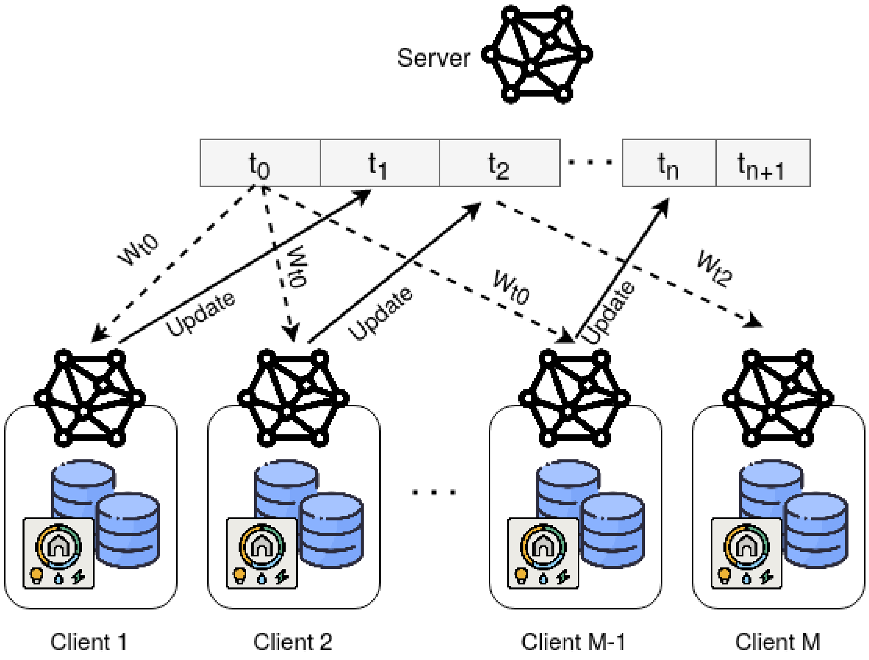


Figure 7.1: Asynchronous federated learning.

as the sum of the absolute vector values, where the vectors are weight differences between the global model and the local models. In FL, the direction of minimizing local objective $\psi_k(w)$ might not align with the direction of minimizing global objective $L(w)$ even though they have similar weights [104]. Therefore, we need β_k as a difference between local objectives and the global objective to define the alignment direction. If β_k is positive, it has an opposite direction to the global aggregation. In contrast, when β_k is negative, the local node positively contributes to the global aggregation.

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

$$f(\lambda_k) = ae^{-\frac{(\lambda_k-\mu)^2}{2\sigma^2}} \tag{7.2}$$

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

3) Calculating Client Contribution Ratios After determining node contributions and scaling them to handle the participants with large values, we use Softmax (equation 7.3) function to calculate the ratio of each participating node in the global model aggregation as follows:

$$\xi_k(w_k) = \frac{e^{f(\lambda_k)}}{\sum_{j=1}^{K} e^{f(\lambda_j)}} \tag{7.3}$$

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

Algorithm 6 presents the FedNorm process. FedNorm employs LSTM, a variant of RNN, as the base learner because LSTM can capture temporal dependencies and has been very successful in load forecasting [128]. First, the global LSTM model is initialized with random weights, Line 2. Each round of FedNorm training starts by randomly selecting a subset of devices S_t and broadcasting the model to the chosen devices, lines 4 and 5. The selected devices then train their local models in parallel with their local data for multiple epochs, Line 7.

Procedure *AsyncClientUpdate*, Line 15, shows the local model training process: the local data P_k are divided into the batches B of size b_s (Line 16) and the local device trains the received model for multiple epochs with created batches as shown in lines 17 to 19. The clients from S_t that have completed their training send their new local weights back to the server for aggregation, Line 20, and the server updates the global model by calculating the weighted average of the received local weights as shown in lines 9-11. These three lines 9-11 are the core of FedNorm and correspond to the three steps described above and represented with equations 7.1-7.3. Note that the server does not wait to receive updates from all clients, but aggregates the global model as client's updates arrive. Next, the updated global model is sent back to all non-running clients, Line 12. The process is repeated from Line 3 until

Algorithm 6 FedNorm

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

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

convergence and, finally, the trained global model is broadcasted to all participants, Line 13.

7.2 Evaluation Methodology

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

7.2.1 Dataset and evaluation Metrics

The FedNorm evaluation is conducted with the real-world residential consumers dataset provided by London Hydro, a local electrical distribution utility. Each consumers' dataset contains hourly energy consumption for three years, resulting in 25,560 readings per household. Additional features, including the day of the year, the day of the month, the week of the year, the

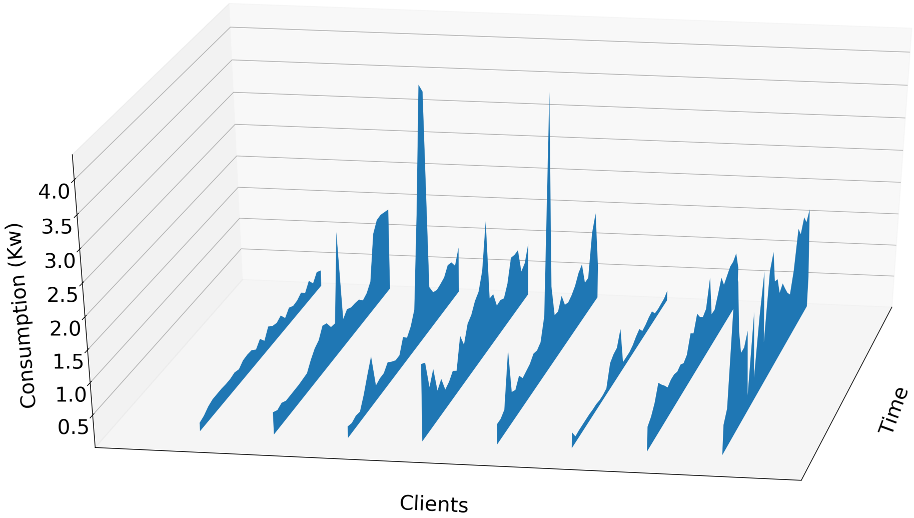


Figure 7.2: Electricity load examples for eight houses

day of the week, and the hour of the day, were devised from the load reading date/time to assist with modeling daily, monthly, and weekly patterns. The study included 19 consumers. Fig. 7.2 depicts a part of the load data for eight example households. It can be observed that load patterns, as well as load magnitudes, differ greatly among households.

Each household with its own data is treated as a local node in FL scenarios. For the purpose of experimental evaluation, each household dataset is divided into 70% training and 30% testing set. This divide remains the same for all FL and non-FL experiments. A single-layer LSTM followed by a fully connected layer is used as the base model. Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) are common error metrics considered for load forecasting [100]; therefore, they are used here as well. It is important to note that RMSE is a scale-dependent error metric, which means that the same RMSE values have different meanings depending on the magnitude of data. MAPE, on the other hand, expresses errors as percentages; thus, it is better suited for comparison among datasets.

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

7.2.2 Experimental Setup

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

Fig. 7.3 illustrates the simulation process. In Step $S1$, a fraction F (in figure five clients) of the clients is selected to train their local models. In step $S2$, the selected clients are randomly divided into two groups C without delay and $R1$ with delay. The clients in C finish their training in the current round and are aggregated into the global model in Step $S3$ but $R1$ clients are still training their local models. In Step $S4$, all clients except $R1$s receive a copy of the updated global model. In the next round of training $S5$, again a fraction of the clients is selected; however, the size of the fraction is equal to $F' = F - count(R1)$. In $S6$, similar to $S2$, the F'

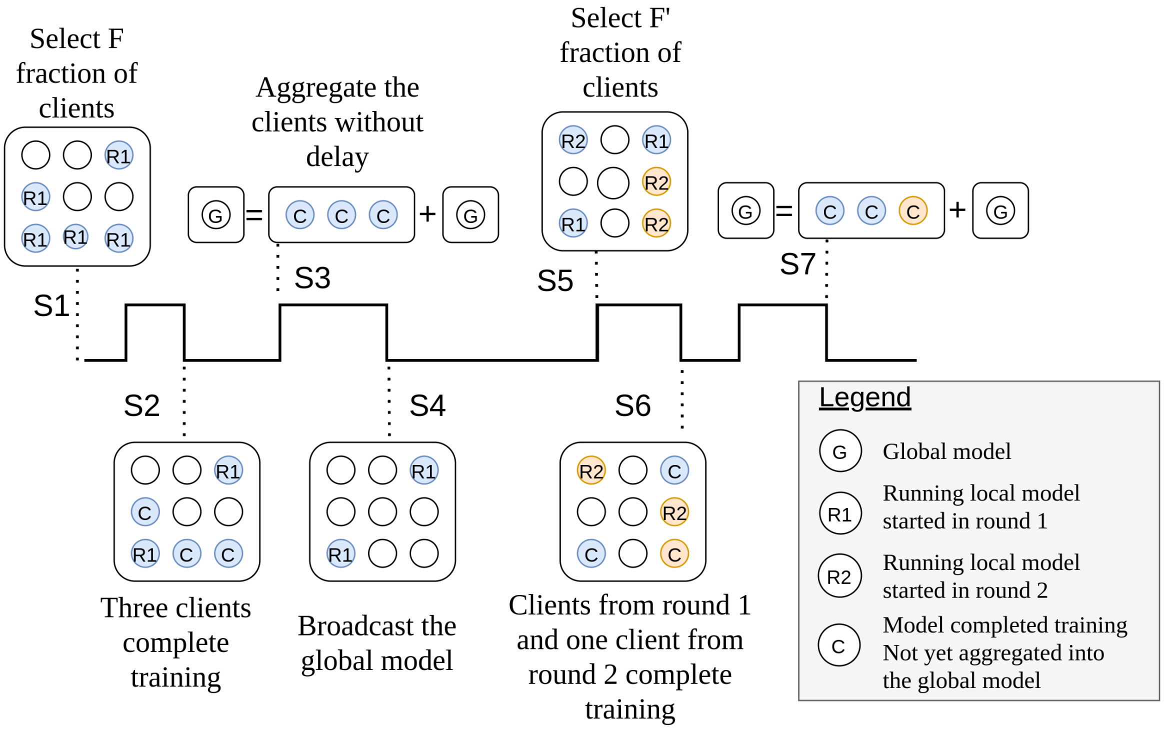

Select F fraction of clients
R1
R1
R1 R1 R1
Aggregate the clients without delay
G = C C C + G
S3
Select F' fraction of clients
R2 R1
R2
R1 R2
S5
G = C C C + G
S7
S1
S2
S4
S6
R1
C
R1 C C
Three clients complete training
R1
R1
Broadcast the global model
R2 C
R2
C C
Clients from round 1 and one client from round 2 complete training
Legend
G Global model
R1 Running local model started in round 1
R2 Running local model started in round 2
C Model completed training Not yet aggregated into the global model

Figure 7.3: Asynchronous settings simulation.

is divided into two groups: without and with delays, C and $R2$ respectively. Thus, in Step 6, the delayed clients $R1$ from the previous round and some clients from round two complete the training, as indicated with C in the figure, while $R2$ clients are still running and will complete training in the next round. Clients C are aggregated into the global model in Step $S7$. This process continues until the model converges.

7.2.3 FL Techniques Included in Comparison

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

- FedSGD: Federated stochastic gradient descent is the first proposed FL method. In this approach, the server averages the local clients' gradients to make a gradient descent step on the global model [133].

- FedAvg: This is a widely used generalization of FedSGD [129], in which the local nodes train for several epochs and then exchange the weights rather than the gradients.

- FedAdam, FedYogi, FedAdaGrad [134]: These approaches are an adaptive variants of the stochastic gradient (FedSGD) method. The adaptive optimizers ADAM, YOGI, and AdaGrad are applied on the server to address the issues of client drift.

- FedProx: This FL framework adds a proximal term on the FedSGD's local objective function to mitigate the data heterogeneity problem and to improve the model stability [135].

- FedAsync: This approach uses a weighted average to update the server model. [103].

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

7.3 Results and Analysis

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

7.3.1 Predictive Performance Comparison

The proposed FedNorm is compared to five other FL methods on the task of predicting load demand eight hours and sixteen hours ahead. The average predictive performance of each algorithm for the two forecasting horizons is shown in Table 7.1. In terms of both evaluation metrics, MAPE and RMSE, FedNorm achieves the lowest error for both forecasting horizons.

Table 7.1: Average MAPE and RMSE for 8 and 16 hours ahead forecast

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

In terms of MAPE error, FedAvg, FedProx, and FedAsync show a similar performance while FedSGD and FedAdagrad exhibit large errors. In terms of RMSE, FedNorm performs the best, while there is little difference among other models.

As households differ greatly in their consumption patterns, it is important to examine the performance for individual houses. Thus, figures 7.4 and 7.5 depict MAPE and RMSE obtained by each algorithm for eight hours ahead prediction, for each house individually. Overall, FedNorm performs better than other approaches. While for a few houses, such as House 6, other approaches achieve slightly lower error than FedNorm, for most houses FedNorm performs much better.

Figures 7.6 and 7.7 show the same metrics, MAPE and RMSE, for sixteen hours ahead prediction. Again, FedNorm performs the best for most houses, and for those houses for which it is not the best, it achieves the error very close to the lowest one.

Fig. 7.8 depicts an example of actual and predicted loads for House 3 and eights hours forecasting horizon. Specifically, forecasts obtained by the top three algorithms (FedNorm, FedAvg, FedProx) according to MAPE are shown. It can be observed that the predicted values better match the actual values for FedNorm than for the other approaches, which supports the results observed in Table 7.1.

7.3.2 Statistical Significance

Standard metrics such as MAPE and RMSE compare models, but they do not consider whether the differences between the models are significant. Diebold-Mariano test [136] can be used to determine if forecasts are significantly different. For two forecasts $f_1, \dots, f_n$ and $g_1, \dots, g_n$, for a time series $y_1, \dots, y_n$, the Diebold-Mariano test is defined as follows:

$$DM = \frac{\bar{d}}{\sqrt{[\gamma_0 + 2\sum_{k=1}^{n^{\frac{1}{3}}+1} \gamma_k] n^{-1}}} \tag{7.4}$$

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

Because DM is a statistical test, when the DM value falls outside the range [-1.96 1.96], the null hypothesis (the difference between two model's performance is not significant) is rejected

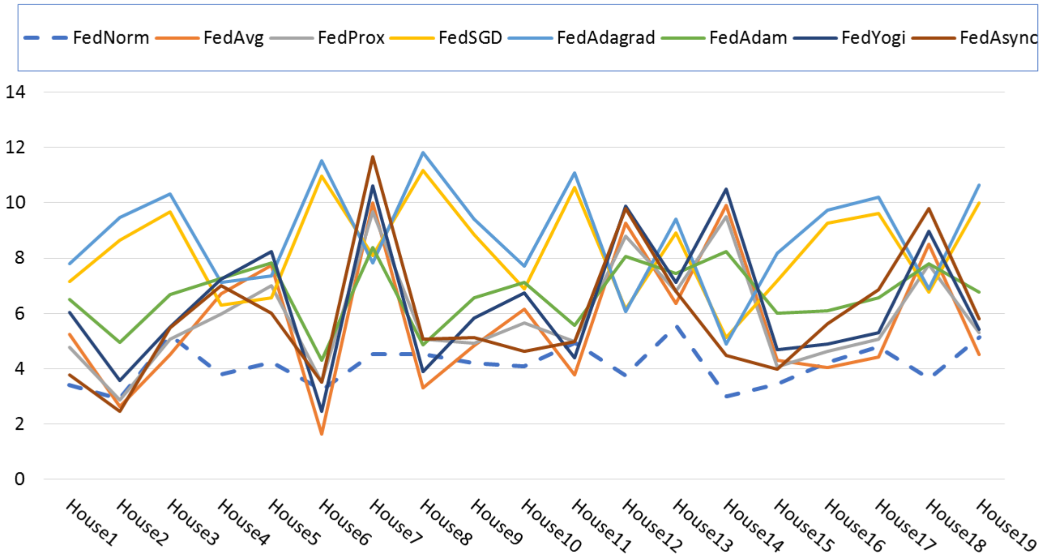


Figure 7.4: MAPE errors for eight hours ahead prediction.

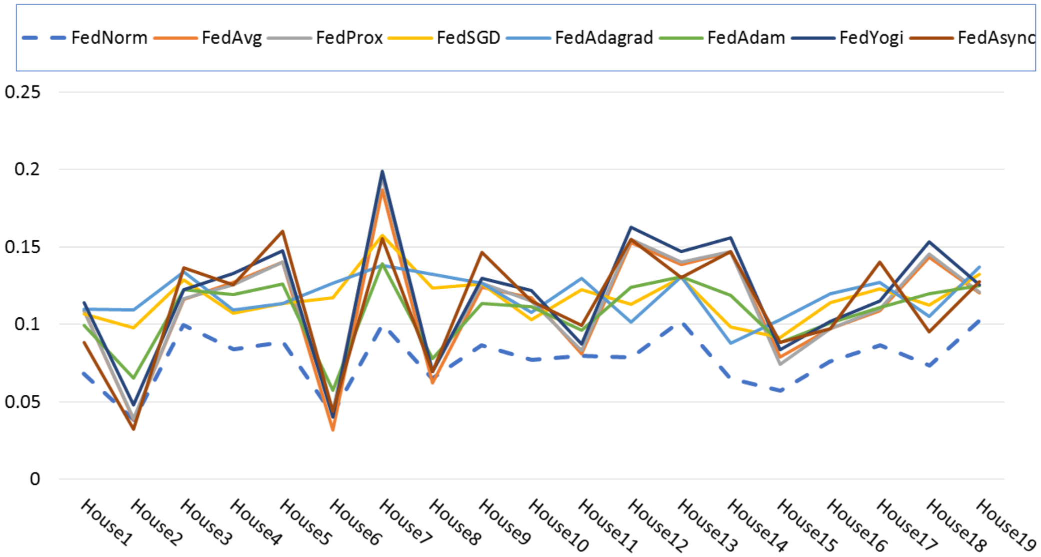


Figure 7.5: RMSE errors for eight hours ahead prediction.

at a 5% confidence level. Fig. 7.9 shows DM values for FedSGD, FedAvg, FedAdam, FedAdagrad, FedYogi, FedProx, FedAsync contrasted to the proposed FedNorm for each house. As the goal is to evaluate the proposed FedNorm against other algorithms, rows in the figure compare FedNorm to each of the other algorithms. It can be observed that the outcome of the Diebold-

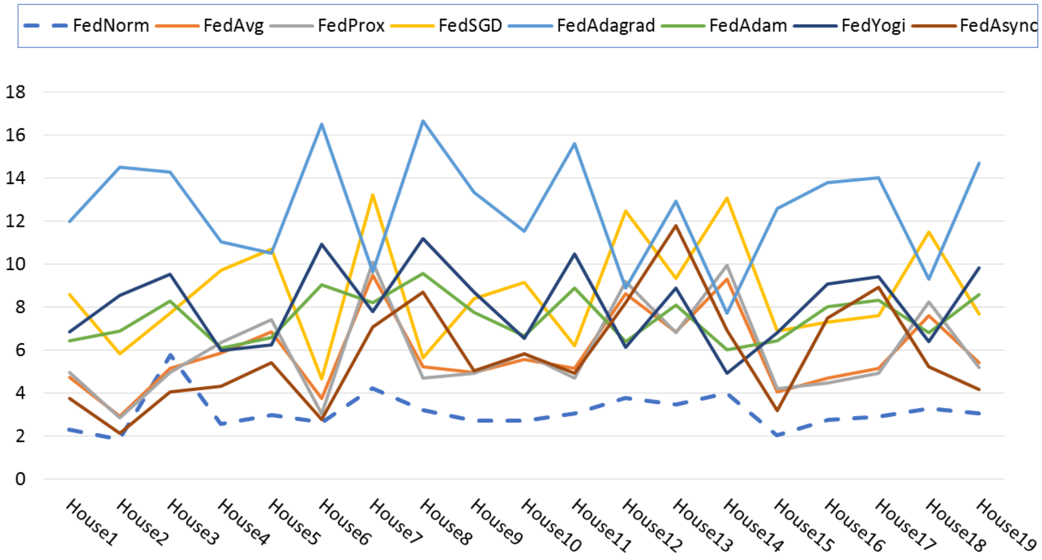


Figure 7.6: MAPE errors for 16 hours ahead prediction.

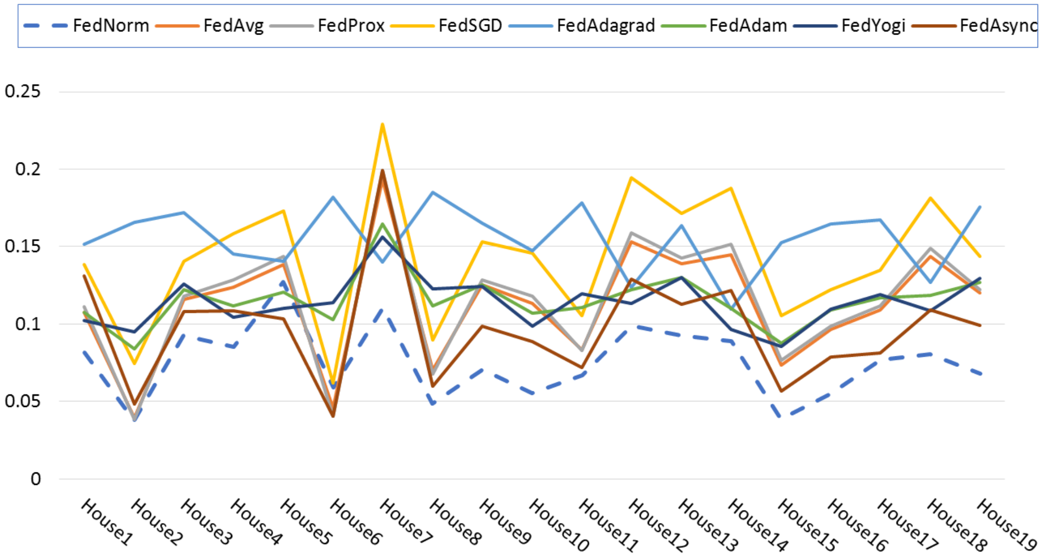
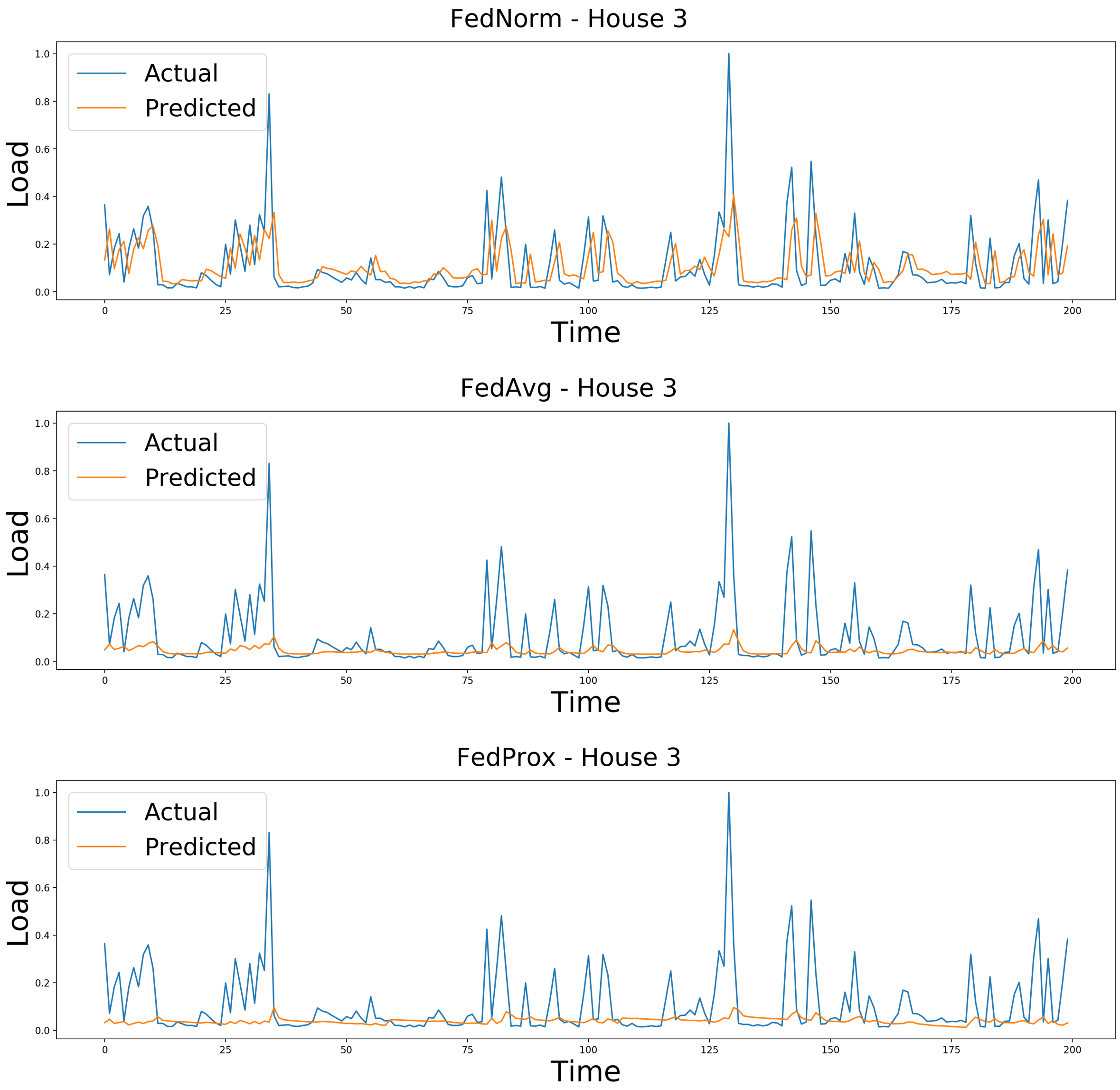


Figure 7.7: RMSE errors for 16 hours ahead prediction.


FedNorm - House 3

FedAvg - House 3

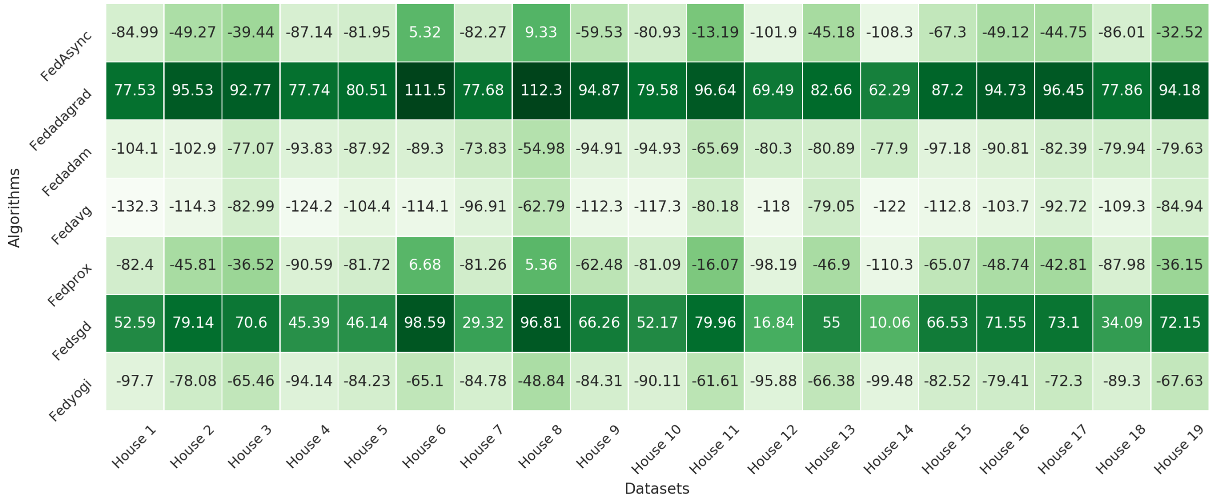
FedProx - House 3

Figure 7.8: Actual and predicted load: House 3, eight hours ahead forecast

Mariano test indicates that the forecasts obtained by the proposed FedNorm are significantly different from all other considered approaches.

7.3.3 Analysis of Convergence

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



	House 1	House 2	House 3	House 4	House 5	House 6	House 7	House 8	House 9	House 10	House 11	House 12	House 13	House 14	House 15	House 16	House 17	House 18	House 19
FedAsync	-84.99	-49.27	-39.44	-87.14	-81.95	5.32	-82.27	9.33	-59.53	-80.93	-13.19	-101.9	-45.18	-108.3	-67.3	-49.12	-44.75	-86.01	-32.52
Fedadagrad	77.53	95.53	92.77	77.74	80.51	111.5	77.68	112.3	94.87	79.58	96.64	69.49	82.66	62.29	87.2	94.73	96.45	77.86	94.18
Fedadam	-104.1	-102.9	-77.07	-93.83	-87.92	-89.3	-73.83	-54.98	-94.91	-94.93	-65.69	-80.3	-80.89	-77.9	-97.18	-90.81	-82.39	-79.94	-79.63
Fedavg	-132.3	-114.3	-82.99	-124.2	-104.4	-114.1	-96.91	-62.79	-112.3	-117.3	-80.18	-118	-79.05	-122	-112.8	-103.7	-92.72	-109.3	-84.94
Fedprox	-82.4	-45.81	-36.52	-90.59	-81.72	6.68	-81.26	5.36	-62.48	-81.09	-16.07	-98.19	-46.9	-110.3	-65.07	-48.74	-42.81	-87.98	-36.15
Fedsgd	52.59	79.14	70.6	45.39	46.14	98.59	29.32	96.81	66.26	52.17	79.96	16.84	55	10.06	66.53	71.55	73.1	34.09	72.15
Fedyogi	-97.7	-78.08	-65.46	-94.14	-84.23	-65.1	-84.78	-48.84	-84.31	-90.11	-61.61	-95.88	-66.38	-99.48	-82.52	-79.41	-72.3	-89.3	-67.63

Figure 7.9: Diebold-Mariano test: DM values for 19 houses.

report the median, 25th and 75th percentile, maximum, and minimum of errors.

Fig. 7.10 shows the error distributions for all FL methods and the first 20 training rounds. It can be observed that FedNorm converges very fast, and after the third round, it converges. Additionally, the error ranges are small, which means that the FedNorm performance for different houses is very similar and with a low error. FedProx takes longer to converge, with large error ranges in early rounds and smaller in later rounds. Although FedAvg converges after the third round as our approach, error ranges for FedAvg are large, which indicates inconsistencies among houses. FedSGD takes longer to converge; at first, the error range is large, but after 12 rounds, the error range is reduced. Errors for FedYogi, FedAdam, FedAsync, and FedAdagrad vary over training rounds, and there is no clear indication of convergence. Overall, FedNorm converges faster than other approaches, and its performance among homes is more consistent.

7.4 Discussion

This chapter proposes FedNorm, an asynchronous FL approach for load forecasting with smart meter data, which does not require participants to share their local data. FedNorm updates the global model asynchronously without waiting for all client devices to complete their training. The contributions of each client node are determined based on the similarity of local

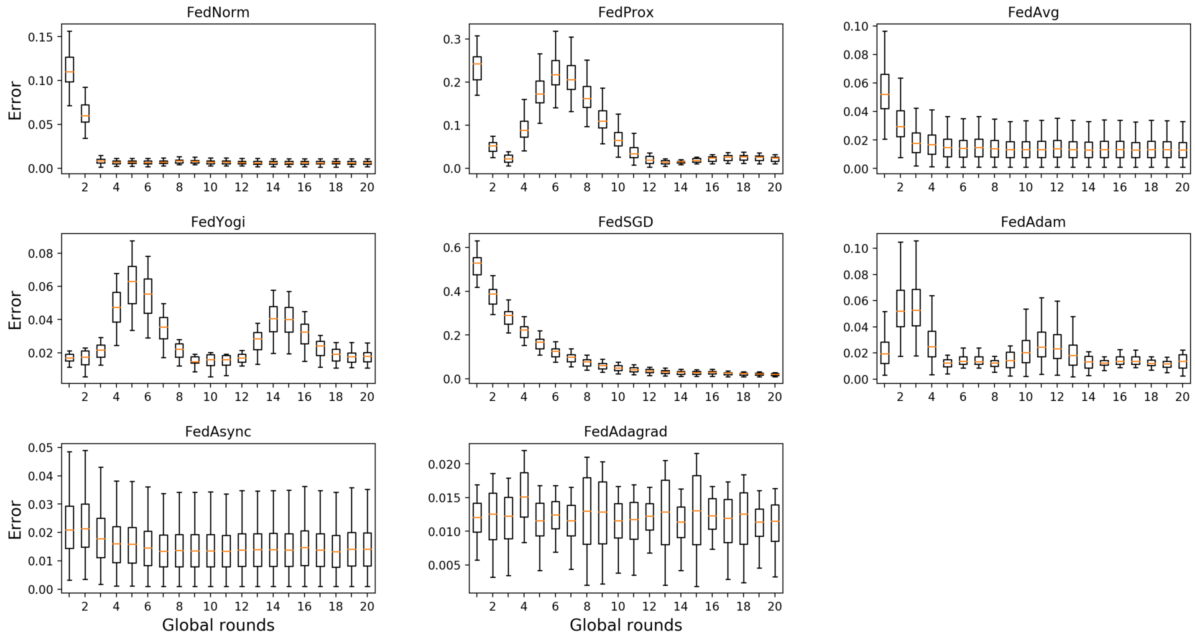


Figure 7.10: Errors (median, 25th Percentile, 75th Percentile, maximum, and minimum) over iterations for each of the seven algorithms

model weights with the global model weights while taking into account the magnitude of the local objective function. The experiments show that FedNorm achieves higher accuracy than seven other FL approaches FedSGD, FedAvg, FedAdam, FedYogi, FedAdagrad, FedProx, and FedAsync for eight and sixteen hours forecasting horizons. The main reason for this is that FedNorm pays more attention to the local models which are more similar to the global model. Statistical tests showed that the difference between FedNorm and other approaches is significant while examination of performance on individual houses demonstrated high consistency and low error rates.

Chapter 8

Conclusion and Future Work

This chapter first summarizes the main findings for the proposed Recurrent Generative Adversarial Networks, Online Adaptive RNN, and Federated learning techniques in Conclusion subsection. Next, the possible extensions of the work are discussed in the Future Work section.

8.1 Conclusion

Energy drives economies and societies, but it has also been the biggest contributor to global warming and accounts for about two-thirds of greenhouse gas emissions [137]. An efficient energy management will be critical in combating environmental challenges and lowering energy production's side effects. Improved energy management also leads to financial benefits for the end consumers by lowering energy costs and operating expenses. Load forecasting is essential in energy management because it helps with power infrastructure planning, generation scheduling, demand and supply balance, and energy budgeting. Thanks to advanced metering infrastructure and widespread deployments of smart meters, utility providers can measure and record energy consumption for particular buildings or even individual residences. Massive amount of smart meter data has paved the way for new, more in-depth insights into energy usage trends, as well as large-scale load forecasts at the individual consumer level. Sensor-based load forecasting techniques train machine learning models by using historical load data gathered by these smart meters or similar technologies, often combined with meteorological data.

Conventionally, smart meter data are transmitted to a data centre or other centralized systems for storage and for training machine learning models. Although these centralized solutions have shown great results, they require transferring all data to a centralized location, which results in significant network traffic [96]. Moreover, a centralized ML not only requires sharing local data with the centralized systems imposing security and privacy concerns, but also makes complying with stringent data regulations challenging. As the number of smart meters grows, training an individual ML model for each smart meter becomes computationally expensive and even infeasible. Furthermore, batch (offline) learning requires all data to be available at the start of training, whereas online models learn as new data is received without the need to store all data.

Consequently, this thesis provides a solution composed of four parts.

1) Recurrent Generative Adversarial Networks (R-GAN) generates realistic energy consumption data by learning from real data samples. Introduced R-GAN replaces Convolutional Neural Networks (CNNs) used in image GANs with Recurrent Neural Networks (RNNs) because of RNNs ability to capture temporal dependence in time series data. To deal with convergence instability and to improve the quality of generated data, Wasserstein GANs (WGANs) and Metropolis-Hastings GAN (MH-GAN) techniques were used. Moreover, ARIMA and Fourier Transform were applied to generate new features and, consequently, improve the quality of generated data.

To evaluate the suitability of data generated with R-GANs for machine learning, energy forecasting experiments were conducted. Synthetic data produced with R-GAN was used to train the energy forecasting model and then, the trained model was tested on the real data. Results show that the model trained with synthetic data achieves similar accuracy as the one trained with real data. The addition of features generated by ARIMA and Fourier transform improves the quality of generated data.

2) Online Adaptive Recurrent Neural Network is an online learning approach for load forecasting where the model is continuously updated as new data arrive. The base learner is a recurrent neural network in order to capture temporal dependencies, while continuous learning is carried out with the addition of preprocessing, buffering, and tuning modules. The preprocessing module prepares data for online learning, the tuning module adapts neural network hyperparameters to newly arriving patterns, and the buffering module facilitates learning from especially difficult patters and assists in handling concept drift.

The evaluation was carried out with five individual households. The proposed approach achieved higher accuracy than the traditional offline long short term memory neural network for all five households for all forecasting lengths. Comparing to five online algorithms, Online Adaptive RNN achieved higher accuracy than the four algorithms for all forecasting horizons. The fifth algorithm, online K Nearest Neighbor, archived slightly better accuracy for one hour ahead forecasting, but Online Adaptive RNN outperformed KNN for 50, 100, and 200 hours ahead. Moreover, training time for the proposed approach is an order of magnitude shorter than that of the traditional offline LSTM.

3) Federated learning (FL) approach was proposed for load forecasting with smart meters capable of training a machine learning model in a distributed manner, without requiring the participant to share their local data. In the proposed FL approach, a global ML model is shared across independent devices corresponding to individual smart meters and each device updates its local copy of the shared model using local data. Then, these local updates are sent to the server to be aggregated and merged into the global model. As recurrent neural networks can capture temporal dependencies and have been very successfully in load forecasting, the proposed approach employs LSTM, a variant of RNN, as the base learner. Two strategies, FedSGD and FedAVG, have been examined: they differ in the way they train the local model and in the frequency of sending the model updates to the server.

The experiments show that both, FedAVG and FedSGD approaches achieve higher accuracy than the individual LSTMs and central models for one hour forecasting horizon. For this horizon, FedAVG achieved slightly better accuracy than FedSGD. For 24 hours ahead, FedAVG outperformed all other approaches while FedSGD experienced conversion difficulties and exhibited higher errors than individual LSTMS. Also, the proposed approach was evaluated in a dynamic environment where some smart meters join the federation after the training is complete and use the already trained model for load forecasting. The results demonstrate that even in this scenario, the FedAVG and FedSGD achieve high accuracy.

4) FedNorm proposes an asynchronous FL approach for load forecasting with smart meter data, which does not require participants to share their local data. FedNorm updates the global model asynchronously without waiting for all client devices to complete their training. The contributions of each client node are determined based on the similarity of local model weights with the global model weights while taking into account the magnitude of the local objective function. The experiments show that FedNorm achieves higher accuracy than seven other FL approaches FedSGD, FedAvg, FedAdam, FedYogi, FedAdagrad, FedProx, and FedAsync for eight and sixteen hours forecasting horizons. The main reason for this is that FedNorm pays more attention to the local models which are more similar to the global model. Statistical tests showed that the difference between FedNorm and other approaches is significant while examination of performance on individual houses demonstrated high consistency and low error rates.

8.2 Future Work

FL techniques successfully learn from distributed smart meter data without requiring to transfer data to a centralized system which confirmed that it can be leveraged to achieve the challenging tasks of load forecasting. Moreover, the propose online learning technique demonstrates the the ML model can learn from streaming data. These works have opened the doors for further investigation. Consequently, future work will include the following:

- Evaluate FL on the larger and more diverse datasets. This theses examined FL with residential consumers, and further investigation is needed to explore its behaviour with other consumer types such as industrial or commercial consumers.

- Examine other techniques for merging weights on the server with objective of improving overall FL accuracy and convergence. In this thesis, we proposed node contribution method; however, other approaches including muti-task learning and aggregation employing an attention method can be explored.
- Examine the proposed Asynchronous FL approach on other time series data. The energy domain shares many characteristics with other IoT domains and time-series data; therefore, the proposed FL techniques developed in this research has the potential to be expanded to benefit other learning tasks in different time-series domains.
- Measuring the effects of non-Identical data distribution on federated global model. Given their distributed nature, the statistics of the data across various devices is likely to differ significantly. Consequently, it is necessary to investigate how different degrees of heterogeneity affect FL.
- Merge online and FL methods where the local devices perform online learning with continuous streaming local data and a central server aggregates model parameters from local clients.
- Personalization has been suggested in asynchronous FL: each client trains their own local models while also contributing to the global model. It is necessary to derive a generalisation bound for a mixture of local and global models, as well as to determine the best mixing parameter.
- Another possible solution to deal with hetoregeinity is multi-center aggregation mechanism. It learns multiple global models from data, and simultaneously derives the optimal matching between users and centers. This approach can be examined and compared with our proposed technique in terms of performance.

- The work described in this thesis on online learning touches on concept drift; however, more research and improvements are needed to better address concept drift.

With transitions to Smart Grids, energy forecasting is becoming even more important especially on the individual consumer level. While this thesis contributed to the load forecasting, as mentioned, there are numerous direction for further investigation.

Bibliography

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

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

[3] Isaac Kofi Nti, Moses Teimeh, Owusu Nyarko-Boateng, and Adebayo Felix Adekoya. Electricity load forecasting: a systematic review. *Journal of Electrical Systems and Information Technology*, 7(1):1–19, 2020.

[4] Gilles Notton and Cyril Voyant. Forecasting of intermittent solar energy resource. *Advances in Renewable Energies and Power Technologies*, 1:77–114, 2018.

[5] Yi Wang, Qixin Chen, Tao Hong, and Chongqing Kang. Review of smart meter data analytics: Applications, methodologies, and challenges. *IEEE Transactions on Smart Grid*, 10(3):3125–148, 2018.

[6] Yi Wang, Qixin Chen, Tao Hong, and Chongqing Kang. Review of smart meter data analytics: Applications, methodologies, and challenges. *IEEE Transactions On Smart Grid*, 10(3):3125–3148, 2018.

[7] Jung-Pin Lai, Yu-Ming Chang, Chieh-Huang Chen, and Ping-Feng Pai. A survey of machine learning models in renewable energy predictions. *Applied Sciences*, 10(17), 2020.

[8] Arooj Arif, Nadeem Javaid, Mubbashra Anwar, Afrah Naeem, Hira Gul, and Sahiba Fareed. Electricity load and price forecasting using machine learning algorithms in smart grid: A survey. In *International Conference on Advanced Information Networking and Applications*, pages 471–483, 2020.

[9] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. In *Big data analysis: new algorithms for a new society*, pages 91–114. 2016.

[10] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(58), 2004.

[11] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys*, 53(2):1–33, 2020.

[12] Nguyen Truong, Kai Sun, Siyao Wang, Florian Guitton, and Yike Guo. Privacy preservation in federated learning: Insights from the gdpr perspective. *arXiv preprint arXiv:2011.05411*, 2020.

[13] Lisa M Austin. Reviewing pipeda: Control, privacy and the limits of fair information practices. *Can. Bus. LJ*, 44:21, 2006.

[14] Tomonobu Senjyu, Hitoshi Takara, Katsumi Uezato, and Toshihisa Funabashi. One-hour-ahead load forecasting using neural network. *IEEE Transactions on Power Systems*, 17(1):113–118, 2002.

[15] Ljubisa Sehovac and Katarina Grolinger. Deep learning for load forecasting: Sequence to sequence recurrent neural networks with attention. *IEEE Access*, 8:36411–36426, 2020.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[17] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

[18] Jiancheng Cai, Han Hu, Shiguang Shan, and Xilin Chen. Fcsr-gan: End-to-end learning for joint face completion and super-resolution. In *IEEE International Conference on Automatic Face and Gesture Recognition*, pages 1–8, 2019.

[19] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[20] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a`[U+FFFC]` laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[21] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[22] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*, 2017.

[23] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[24] Ryan Turner, Jane Hung, Eric Frank, Yunus Saatchi, and Jason Yosinski. Metropolis-hastings generative adversarial networks. In *International Conference on Machine Learning*, pages 6345–6353, 2019.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[26] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. In *International Conference on Learning Representations*, 2016.

[27] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning: Methods, Systems, Challenges*, pages 3–33. 2019.

[28] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[29] Tanveer Ahmad, Hongcai Zhang, and Biao Yan. A review on renewable energy and electricity requirement forecasting models for smart grid and buildings. *Sustainable Cities and Society*, 55(102052), 2020.

[30] Mohammad H Alobaidi, Fateh Chebana, and Mohamed A Meguid. Robust ensemble learning framework for day-ahead forecasting of household based energy consumption. *Applied Energy*, 212:997–1012, 2018.

[31] Pavan Kumar Singh, Nitin Singh, and Richa Negi. Wind power forecasting using hybrid ARIMA-ANN technique. In *Ambient Communications and Computer Systems*, pages 209–220, 2019.

[32] Katarina Grolinger, Miriam AM Capretz, and Luke Seewald. Energy consumption prediction with big data: Balancing prediction accuracy and computational resources. In *IEEE International Congress on Big Data*, pages 157–164, 2016.

[33] Mohammad Navid Fekri, Ananda Mohon Ghosh, and Katarina Grolinger. Generating energy data for machine learning with recurrent generative adversarial networks. *Energies*, 13(130), 2020.

[34] Xin Gao, Xiaobing Li, Bing Zhao, Weijia Ji, Xiao Jing, and Yang He. Short-term electricity load forecasting model based on EMD-GRU with feature selection. *Energies*, 12(6), 2019.

[35] Salah Bouktif, Ali Fiaz, Ali Ouni, and Mohamed Adel Serhani. Optimal deep learning lstm model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches. *Energies*, 11(7), 2018.

[36] Shuang Han, Yan-hui Qiao, Jie Yan, Yong-qian Liu, Li Li, and Zheng Wang. Mid-to-long term wind and photovoltaic power generation prediction based on copula function and long short term memory network. *Applied Energy*, 239:181–191, 2019.

[37] Weilin Guo, Liang Che, Mohammad Shahidehpour, and Xin Wan. Machine-learning based methods in short-term load forecasting. *The Electricity Journal*, 34(1), 2021.

[38] Yifang Tian, Ljubisa Sehovac, and Katarina Grolinger. Similarity-based chained transfer learning for energy forecasting with big data. *IEEE Access*, 2019.

[39] Cheng Fan, Jiayuan Wang, Wenjie Gang, and Shenghan Li. Assessment of deep recurrent neural network-based strategies for short-term building energy predictions. *Applied Energy*, 236:700–710, 2019.

[40] Nivethitha Somu, Gauthama Raman MR, and Krithi Ramamritham. A hybrid model for building energy consumption forecasting using long short term memory networks. *Applied Energy*, 261(114131), 2020.

[41] Ameema Zainab, Dabeeruddin Syed, Ali Ghrayeb, Haitham Abu-Rub, Shady S Refaat, Mahdi Houchati, Othmane Bouhali, and Santiago Bañales Lopez. A multiprocessing-based sensitivity analysis of machine learning algorithms for load forecasting of electric power distribution system. *IEEE Access*, 9:31684–31694, 2021.

[42] Kadir Amasyali and Nora M El-Gohary. A review of data-driven building energy consumption prediction studies. *Renew Sust Energ Rev*, 81:1192–1205, 2018.

[43] Ljubisa Sehovac, Cornelius Nesen, and Katarina Grolinger. Forecasting building energy consumption with deep learning: A sequence to sequence approach. In *IEEE International Congress on Internet of Things*, 2019.

[44] Chirag Deb, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews*, 74:902–924, 2017.

[45] Dimitris Lazos, Alistair B Sproul, and Merlinde Kay. Optimisation of energy management in commercial buildings with weather forecasting inputs: A review. *Renewable and Sustainable Energy Reviews*, 39:587–603, 2014.

[46] Gobind G Pillai, Ghanim A Putrus, and Nicola M Pearsall. Generation of synthetic benchmark electrical load profiles using publicly available load and weather data. *International Journal of Electrical Power and Energy Systems*, 61:1–10, 2014.

[47] BO Ngoko, H Sugihara, and T Funaki. Synthetic generation of high temporal resolution solar radiation data using markov models. *Solar Energy*, 103:160–170, 2014.

[48] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[49] He Zhang, Vishwanath Sindagi, and Vishal M Patel. Image de-raining using a conditional generative adversarial network. *IEEE transactions on circuits and systems for video technology*, 2019.

[50] Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

[51] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI Conference on Artificial Intelligence*, 2017.

[52] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[53] Kay Gregor Hartmann, Robin Tibor Schirrmeister, and Tonio Ball. Eeg-gan: Generative adversarial networks for electroencephalograhic (eeg) brain signals. *arXiv preprint arXiv:1806.01875*, 2018.

[54] Georgios Douzas and Fernando Bacao. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Syst. Appl.*, 91:464–471, 2018.

[55] Steven Cheng-Xian Li, Bo Jiang, and Benjamin Marlin. Misgan: Learning from incomplete data with generative adversarial networks. In *International Conference on Learning Representations*, 2019.

[56] Chao Shang, Aaron Palmer, Jiangwen Sun, Ko-Shin Chen, Jin Lu, and Jinbo Bi. Vigan: Missing view imputation with generative adversarial networks. In *IEEE International Conference on Big Data*, pages 766–775. IEEE, 2017.

[57] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[58] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018.

[59] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. In *Advances in Neural Information Processing Systems*, pages 5228–5237, 2018.

[60] L Theis, A van den Oord, and M Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, pages 1–10, 2016.

[61] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

[62] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, 21(2):6–22, 2019.

[63] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[64] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[65] Ellango Jothimurugesan, Ashraf Tahmasbi, Phillip Gibbons, and Srikanta Tirthapura. Variance-reduced stochastic gradient descent on streaming data. In *Advances in Neural Information Processing Systems*, pages 9906–9915, 2018.

[66] Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass. In *Conference on learning theory*, pages 728–763, 2015.

[67] Javier J Sánchez-Medina, Juan Antonio Guerra-Montenegro, David Sánchez-Rodríguez, Itziar G Alonso-González, and Juan L Navarro-Mesa. Data stream mining applied to maximum wind forecasting in the Canary Islands. *Sensors*, 19(10), 2019.

[68] Julian Vexler and Stefan Kramer. Integrating LSTMs with online density estimation for the probabilistic forecast of energy consumption. In *International Conference on Discovery Science*, pages 533–543, 2019.

[69] Fan Liang, William Grant Hatcher, Guobin Xu, James Nguyen, Weixian Liao, and Wei Yu. Towards online deep learning-based energy forecasting. In *International Conference on Computer Communication and Networks*, pages 1–9, 2019.

[70] Huaien Gao, Rudolf Sollacher, and Hans-Peter Kriegel. Spiral recurrent neural network for online learning. In *European Symposium on Artificial Neural Networks*, pages 483–488, 2007.

[71] Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. Robust online time series prediction with recurrent neural networks. In *IEEE International Conference on Data Science and Advanced Analytics*, pages 816–825, 2016.

[72] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Glenn K Lockwood, Robert Ross, Shane Snyder, and Stefan M Wild. Adaptive learning for concept drift in application performance modeling. In *International Conference on Parallel Processing*, pages 1–11, 2019.

[73] Tonya Fields, George Hsieh, and Jules Chenou. Mitigating drift in time series data with noise augmentation. In *International Conference on Computational Science and Computational Intelligence*, pages 227–230, 2019.

[74] Michelangelo Ceci, Roberto Corizzo, Donato Malerba, and Aleksandra Rashkovska. Spatial autocorrelation and entropy for renewable energy forecasting. *Data Mining and Knowledge Discovery*, 33:698–729, 2019.

[75] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv:1506.08473*, 2015.

[76] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[77] Max Halford, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, and Adil Zouitine. creme, a Python library for online machine learning, 2019.

[78] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[79] Nikunj C Oza. Online bagging and boosting. In *IEEE international conference on systems, man and cybernetics*, volume 3, pages 2340–2345, 2005.

[80] Xing Wu, Zhaowang Liang, and Jianjia Wang. Fedmed: A federated learning framework for language modeling. *Sensors*, 20(14):4048, 2020.

[81] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

[82] Shiva Raj Pokhrel and Jinho Choi. Federated learning with blockchain for autonomous vehicles: Analysis and design challenges. *IEEE Transactions on Communications*, 68(8):4734–4746, 2020.

[83] Yansheng Wang, Yongxin Tong, and Dingyuan Shi. Federated latent dirichlet allocation: A local differential privacy based framework. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 6283–6290, 2020.

[84] Wei Liu, Li Chen, Yunfei Chen, and Wenyi Zhang. Accelerating federated learning via momentum gradient descent. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1754–1766, 2020.

[85] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, page 106854, 2020.

[86] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *IEEE International Conference on Acoustics, Speech and Signal pages=6341–6345, year=2019*.

[87] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 13172–13179, 2020.

[88] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[89] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[90] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. *arXiv preprint arXiv:2004.11791*, 2020.

[91] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[92] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *36th International Conference on Machine Learning*, pages 8114–8124, 2019.

[93] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[94] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[95] Yeongwoo Kim, Ezeddin Al Hakim, Johan Haraldson, Henrik Eriksson, José Mairton B da Silva, and Carlo Fischione. Dynamic clustering in federated learning. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6, 2021.

[96] Afaf Taïk and Soumaya Cherkaoui. Electrical load forecasting using edge computing and federated learning. In *EEE International Conference on Communications*, pages 1–6, 2020.

[97] JianBin Li, YuQi Ren, SuWan Fang, KunChang Li, and MingYu Sun. Federated learning-based ultra-short term load forecasting in power internet of things. In *IEEE International Conference on Energy Internet)*, pages 63–68, 2020.

[98] Yuris Mulya Saputra, Dinh Thai Hoang, Diep N Nguyen, Eryk Dutkiewicz, Markus Dominik Mueck, and Srikathyayani Srikanteswara. Energy demand prediction with federated learning for electric vehicle networks. In *IEEE Global Communications Conference*, pages 1–6, 2019.

[99] Xiaoning Zhang, Fang Fang, and Jiaqi Wang. Probabilistic solar irradiation forecasting based on variational bayesian inference with secure federated learning. *IEEE Transactions on Industrial Informatics*, 2020.

[100] Mohammad Navid Fekri, Katarina Grolinger, and Syed Mir. Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks. *International Journal of Electrical Power & Energy Systems*, 2021.

[101] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *IEEE International Conference on Big Data*, pages 15–24, 2020.

[102] Georgios Damaskinos, Rachid Guerraoui, Anne-Marie Kermarrec, Vlad Nitu, Rhicheek Patra, and Francois Taiani. Fleet: Online federated learning via staleness awareness and performance prediction. In *International Middleware Conference*, 2020.

[103] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv:1903.03934*, 2019.

[104] Hongda Wu and Ping Wang. Fast-convergent federated learning with adaptive weighting. *IEEE Transactions on Cognitive Communications and Networking*, 2021.

[105] Kaile Zhou, Chao Fu, and Shanlin Yang. Big data driven smart energy management: From big data to big insights. *Renew Sust Energ Rev*, 56:215–225, 2016.

[106] Ankur Sial, Amarjeet Singh, Aniket Mahanti, and Mingwei Gong. Heuristics-based detection of abnormal energy consumption. In *International Conference on Smart Grid Inspired Future Technologies*, pages 21–31, 2018.

[107] Ankur Sial, Amarjeet Singh, and Aniket Mahanti. Detecting anomalous energy consumption using contextual analysis of smart meter data. *Wireless Networks*, pages 1–18, 2019.

[108] Chirag Deb, Mario Frei, Johannes Hofer, and Arno Schlueter. Automated load disaggregation for residences with electrical resistance heating. *Energy and Buildings*, 182:61–74, 2019.

[109] Clayton Miller and Forrest Meggers. The building data genome project: An open, public data set from non-residential building electrical meters. *Energy Procedia*, 122:439–444, 2017.

[110] Elizabeth L Ratnam, Steven R Weller, Christopher M Kellett, and Alan T Murray. Residential load and rooftop pv generation: an australian distribution network dataset. *International Journal of Sustainable Energy*, 36(8):787–806, 2017.

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

[112] Kevin Gimpel Dan Hendrycks. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2018.

[113] Luis M Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and buildings*, 140:81–97, 2017.

[114] Mauro Ribeiro, Katarina Grolinger, Hany F ElYamany, Wilson A Higashino, and Miriam AM Capretz. Transfer learning with seasonal and trend adjustment for cross-building energy forecasting. *Energy and buildings*, 165(15):352–363, 2018.

[115] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[116] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2016.

[117] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2016.

[118] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[119] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine learning*, 90:317–346, 2013.

[120] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338, 2009.

[121] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *SIAM international conference on data mining*, pages 443–448, 2007.

[122] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1):100–115, 1954.

[123] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295, 2004.

[124] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[125] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2):1–19, 2019.

[126] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[127] Alexandra L'Heureux, Katarina Grolinger, Hany F Elyamany, and Miriam AM Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 5:7776–7797, 2017.

[128] Mohammad Navid Fekri, Harsh Patel, Katarina Grolinger, and Vinay Sharma. Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network. *Applied Energy*, 282, 2021.

[129] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[130] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. Fedeval: A benchmark system with a comprehensive evaluation model for federated learning. *arXiv preprint arXiv:2011.09655*, 2020.

[131] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[132] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving Google keyboard query suggestions.

[133] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *ACM SIGSAC conference on computer and communications security*, 2015.

[134] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv:2003.00295*, 2020.

[135] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv:1812.06127*, 2018.

[136] Rashpinder Kaur Jagait, Mohammad Navid Fekri, Katarina Grolinger, and Syed Mir. Load forecasting under concept drift: Online ensemble learning with recurrent neural network and arima. *IEEE Access*, 9:98992–99008, 2021.

[137] UN Environment Programme. Energy. https://www.unep.org/explore-topics/energy, 2020.

Curriculum Vitae

Name:	Mohammad Fekri
Post-Secondary Education and Degrees:	The University of Western Ontario (UWO) London, ON, Canada 2019 - 2022 PhD. Electrical and Computer Engineering Iran University of Science and Technology (IUST) Tehran, Iran 2013 - 2016 MSc. Artificial Intelligence The University of Isfahan Isfahan, Iran 2009 - 2013 BSc. Software Engineering
Related Work Experience:	Teaching Assistant and Research Assistant The University of Western Ontario 2019 - 2022

Publications:

Mohammad Navid Fekri, Ananda Mohon Ghosh, and Katarina Grolinger. Generating energy data for machine learning with recurrent generative adversarial networks. Energies,13(130), 2020.

Mohammad Navid Fekri, Harsh Patel, Katarina Grolinger, and Vinay Sharma. Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network. Applied Energy, 282, 2021.

Rashpinder Kaur Jagait , Mohammad Navid Fekri and , Katarina Grolinger and Syed Mir. Load Forecasting Under Concept Drift: Online Ensemble Learning With Recurrent Neural Network and ARIMA. IEEE Access, 2021.

Mohammad Navid Fekri, Katarina Grolinger, and Syed Mir. Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks. International Journal of Electrical Power and Energy Systems, 2021.

Mohammad Navid Fekri, Katarina Grolinger and Syed Mir. Asynchronous Adaptive Federated Learning for Load Forecasting with Smart Meters. IEEE Transactions on Industrial Informatics [Submitted].