Western Graduate&PostdoctoralStudies

Western University

Scholarship@Western

Electronic Thesis and Dissertation Repository

8-12-2021 1:45 PM

Generative Learning in Smart Grid

Samer M. El Kababji, *The University of Western Ontario*

Supervisor: Srikantha, Pirathayini, *The University of Western Ontario*
Joint Supervisor: McIssac, Ken, *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

© Samer M. El Kababji 2021

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

Part of the Computational Engineering Commons, and the Power and Energy Commons

Recommended Citation

El Kababji, Samer M., "Generative Learning in Smart Grid" (2021). *Electronic Thesis and Dissertation Repository*. 7989.
https://ir.lib.uwo.ca/etd/7989

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

If a smart grid is to be described in one word, that word would be ’connectivity’. While electricity production and consumption still depend on a limited number of physical connections, exchanging data is growing enormously. Customers, utilities, sensors, and markets are all different sources of data that are exchanged in a ubiquitous digital setup. To deal with data complexity, many researchers recently focused on machine learning (ML) applications in smart grids. Much of the success in ML is attributed to discriminative learning where models define boundaries to categorize data. Generative learning, however, reveals how data is generated by learning the underlying distribution functions. In the past few years, generative models brought new dimensions to various domains. Computers became painters and composers.

This thesis identifies three applications in the smart grid where generative learning has great potential. On the demand side, residential loads such as dishwashers and clothes driers are simulated using generative models. In specific, the latest developments in generative adversarial networks and kernel density estimators are levered to learn the underlying distributions of individual loads for both power consumption patterns and usage habits. Being data-driven, the learning process eliminates any biases introduced by rule-based models where predetermined fixed formulas describing each load are considered. The study demonstrates the flexibility, viability, and remarkable accuracy of the proposed framework. The resulting synthetic power consumption patterns and usage habits for individual loads are valuable sources for researchers to build or improve their data-driven models for demand-side studies.

Non-intrusive load monitoring (NILM) is the second topic researched on the demand side. The goal in NILM is to identify the status of individual loads in a household by merely relying on a smart meter’s measurements without any hardware installations. The research focuses on identifying the operational condition of individual loads by developing a novel hybrid algorithm that combines the widely used generative technique, namely, hidden Markov model, with k-means clustering. The hybrid model is demonstrated to accurately identify the operation conditions of individual loads based on the ingested aggregate signal.

Finally, for power transmission, a combination of generative models is proposed to estimate power states from a set of redundant measurements. Power state estimation is a fundamental technique in shedding light on the operational condition of the grid. A traditional state estimator is typically executed online and, in its non-linear formulation, involves a high level of computational complexity. Generative models shift that burden to the offline learning process. On the other hand, bad data detection and identification is a central feature in traditional estimators. As such, the developed framework integrated that feature in the data-driven state estimator by incorporating forward and backward generative adversarial networks. Simple domain knowledge is further incorporated in the model to improve its accuracy against the benchmark data-driven model. The proposed framework remarkably detected tampered measurements including false data injection.

Keywords: Non-intrusive load monitoring, load simulation, power state estimation, machine learning, generative adversarial network, false data injection

Summary for Lay Audience

As per Hegel, ontological categorizations are concepts that a simple person uses to recognize the world [1]. Indeed, classification is a fundamental rational activity that scientists brought to machines in the era of ”machine learning”. Artificial neural networks made this possible through discriminative learning, e.g. machines learn to discriminate dogs from cats. But this is not the end of the story. People’s mental power extends to more than a mere classification of objects. People can extract common features among a set of objects and generate new ideas. While architects can discriminate between good and bad designs, they can learn from these historical records to come up with novel designs. This is why it is widely accepted among researchers that brains learn generative models [2]. To mimic human intelligence, generative learning is considered an essential part of artificial intelligence.

This thesis applies recent developments in generative modelling by tackling three problems in the modern electrical grid. First, a model is developed to simulate the operation of electrical loads in the residential sector. The generative model eventually learns how to behave like a dishwasher, a cloth dryer, and so on. If the model is instructed to simulate a dishwasher, it will generate a signal that looks *similar* to that generated by a real dishwasher in terms of both signal’s shape and its occurrence in time. The generated synthetic data is a valuable tool that can be used by researchers in further studies such as non-intrusive load monitoring (NILM). This is the second area where generative modelling is applied. In simple words, NILM is the process of identifying the operational status of individual loads inside a house by just reading the measurements recorded by a modern electric meter. The recorded aggregate power consumption for a household is passed to the developed generative model that will, in turn, identify which individual appliances were operating during that time. One of the advantages of NILM is to help consumers to have a better planning of their power consumption and, hence, reduce electricity bills.

Finally, generative techniques are leveraged to assist in quickly monitoring the status of the power grid. Typically, in an electrical grid, measurements (e.g. power values) are taken at various locations along the grid and sent back to a control centre. These measurements are fed to the pre-trained generative model which will instantly provide the operator with a snapshot of the voltage complex values at different buses, i.e. common connections. These values can be used to determine the status of the whole grid. In addition, the generative model can identify any bad measurements that do not seem to be within the acceptable range of error that is usually associated with the measuring process. The developed framework reflects the great benefit that generative models bring to the industry.

Dedicated

To the memory of my mother, Feryal, who was long awaiting this moment, but could not make it in time.

To my father, Maher, a great unique man, who always shows up to push me forward.

&

To my wife, Serin, the only person that can draw a smile on my face with her magical brush of serenity and love.

Acknowledgments

I would like to express my deep gratitude to my advisor, Dr. Pirathayini Srikantha for her support, encouragement, guidance, and patience throughout my Ph.D. journey. I met many people in my life, but not too many with Dr. Srikantha's dedication. Her vision has always inspired me. Her invaluable directions paved the road for me to explore the world of academia. I was privileged to work under her supervision.

Also, I would like to thank my supervisory committee Dr. Firouz Ajaei and Dr. Katarina Gronlinger for their insightful comments and valuable suggestions.

Finally, I extend my thanks to the ECE department at Western University for their financial and administrative support.

Contents

Abstract ii

Summary for Lay Audience iv

Dedication vi

Acknowledgments vii

List of Figures xi

List of Tables xii

1 Introduction 1

1.1 Machine learning in smart grid 1
1.2 Generative vs. discriminative models 3
1.3 State-of-the-art 3
1.3.1 Residential load simulation 4
1.3.2 Non-intrusive load monitoring 6
1.3.3 Power system state estimation 9
1.4 Research motivations 12
1.4.1 Residential load simulation 12
1.4.2 Non-intrusive load monitoring 13
1.4.3 Power system state estimation 14
1.5 Research objectives 15
1.6 Thesis outline 16

2 Residential load simulation 18

2.1 Background 18
2.1.1 Neural networks 18
2.1.2 Generative adversarial networks 21
2.1.3 Maximum-mean discrepancy 25

2.1.4 Kernel density estimator 26
2.1.5 Matched-filter 28
2.1.6 Cross validation 30
2.2 Problem definition 31
2.3 Proposed framework 35
2.4 Experimental studies 40
2.4.1 Preprocessing 40
2.4.2 Generating patterns 43
2.4.3 Generating habits 48
2.5 Summary 54

3 Non-intrusive load monitoring 56
3.1 Background 56
3.1.1 Hidden Markov model 56
3.1.2 k-means clustering 58
3.1.3 Quality metrics 59
3.2 Problem definition 60
3.3 Proposed framework 62
3.4 Experimental studies 67
3.5 Summary 71

4 Power state estimation 74
4.1 Background 74
4.1.1 Power state estimation 74
4.1.2 Bad data detection 77
4.1.3 Cycle GAN 79
4.1.4 Embedding 80
4.2 Problem definition 81
4.3 Proposed framework 82
4.3.1 Vanilla cycle GAN 82
4.3.2 Grid-aware cycle GAN 85
4.3.3 Evaluating cycle GAN for state estimation 86
4.3.4 Tampered data identification (TDI) 87
4.4 Experimental studies 88
4.4.1 Generating load scenarios 88
4.4.2 Training cycle GAN 88
4.4.3 Benchmarking 89

4.4.4 Identifying bad measurements 92
4.5 Summary 94

5 Conclusions 95
5.1 Summary 95
5.2 Contributions 96
5.3 Future works 98

Bibliography 100

A Public datasets 109

Curriculum Vitae 111

List of Figures

1.1 NIST smart grid conceptual model [3] 2
1.2 Thesis outline including the main tasks in the research 17

2.1 Examples of activation functions 20
2.2 Neural Network 21
2.3 Generative Adversarial Network 22
2.4 Conditional Generative Adversarial Network 25
2.5 Typical kernels used Kernel Density Estimation 27
2.6 Varying the bandwidth while using Gaussian kernel in KDE 28
2.7 Repetitive patterns detected using matched filer 29
2.8 Power demand patterns for different residential individual loads 32
2.9 Power demand *patterns* for the same individual load at different times 33
2.10 Power demand *habits* for the same individual load at different times 34
2.11 Proposed framework for learning patterns and habits of residential individual loads. 35
2.12 Typical example used for training *patterns* 37
2.13 Typical example used for training *habits* 38
2.14 Trained frameowrk for simulating residential loads 41
2.15 Raw patterns for different loads 42
2.16 Impact of reducing granularity (T_g) on the training samples for cloth dryer . . . 42
2.17 Raw usage habits of different loads for around 11 days. 43
2.18 Evolution of the inverted loss during training. 44
2.19 Synthetic patterns from proposed GAN. 45
2.20 Comparison of real versus synthetic patterns. 46
2.21 Mode collapse in CGAN-Patterns for Cloth Dryer [4]. 47
2.22 Training and synthesis of the GAN system. 47
2.23 Normalized confusion matrix for synthetic data. 48
2.24 Synthetic (triangles) vs. real distribution (circles) while training CGAN-Habits. 50
2.25 Loss while training CGAN-Habits 50

2.26 Correlation among engineered features for all loads. 51
2.27 Histograms for usage habits of Cloth Dryer (CDE). 52
2.28 Histograms for usage habits of Dishwasher (DWE). 52
2.29 Histograms for usage habits of Fridge (FGE). 53
2.30 Histograms for usage habits of Heat pump (HPE). 53
2.31 MMD for training CGAN-Habits 54

3.1 Hidden Markov Model 57
3.2 Factorial hidden Markov model where the aggregate power o_t^A at any point in time t is dependent on the internal states of all the connected loads at that time instance. 61
3.3 Hidden Markov chain for load l with univariate emission distribution such for observed power. 61
3.4 Hybrid Algorithm for NILM 64
3.5 Illustration of steps 4 and 5 in HTA algorithm. Clean observations $\mathbf{o^l}$ (left) for load l are converted to sequence of states $\mathbf{q^l}$ (right) by applying a threshold τ (red dotted line). The four transition frequencies between states are calculated and the load's transition matrix A^l is updated accordingly. 65
3.6 Illustration of step 10 in HTA algorithm. Clean observations $\mathbf{o^l}$ (left) for load l are converted to sequence of lables $\mathbf{y^l}$ (middle) using k^l-Means algorithm.The distribution of data is found (right) and the load's emission matrix B^l is updated accordingly. 67
3.7 Raw whole house active power signal 68
3.8 Removing outliers from training signals 68
3.9 Conversion of observed power values for various appliances to two-state sequence. 69
3.10 k-Means clustering of various load signals. 70
3.11 Converting aggregate power signal to labels 72
3.12 Inferring states of aggregate signal 72
3.13 Decoding states of aggregate signal into individual loads 73

4.1 Main blocks of a cycle GAN. 80
4.2 Cycle GAN setup for state estimation. Load scenario indices i and j are omitted for clarity. 83
4.3 A benchmark MLP is used to evaluate the cycle GAN accuracy for estimating states. Unlike cycle GAN, labelled input data is used and predicted states are measured against the paired ground truth states. 87

4.4 Proposed cycle GAN in inference mode used fr tampered data identification . . 87
4.5 Constructing Training and Testing Datasets for cycle GAN 89
4.6 Adversarial losses for vanilla cycle GAN. The forward generator (G) and its adversary D_y are trained to estimate states, while the backward generator (F) and its adversary (D_x) are trained to cycle measurements. 91
4.7 Adversarial losses for grid-aware cycle GAN. 91
4.8 Loss of benchmark MLP for both train and test datasets 92
4.9 F1 and Accuracy scores against threshold in grid-aware cycle GAN. 94

List of Tables

1.1 Accuracy comparison of various data-driven state estimators 10

2.1 Architecture of CGAN-Patterns [4] 44
2.2 Evaluator net architecture [4]. 48
2.3 KDE optimized parameters using 10-fold cross-validation [4]. 49
2.4 Architecture of CGAN-Habits [4]. 49
2.5 MMD comparison of usage habit synthesis. 54

3.1 Results of proposed non-intrusive load monitoring showing remarkable disaggregation power. 71

4.1 Cycle GAN Architecture 90
4.2 Benchmark MLP Architecture 90
4.3 Mean squared-error for various models where the MLP is used only for benchmarking purposes. 91
4.4 Accuracy and F1 scores for various types of losses against different attacks. . . 93

Chapter 1

Introduction

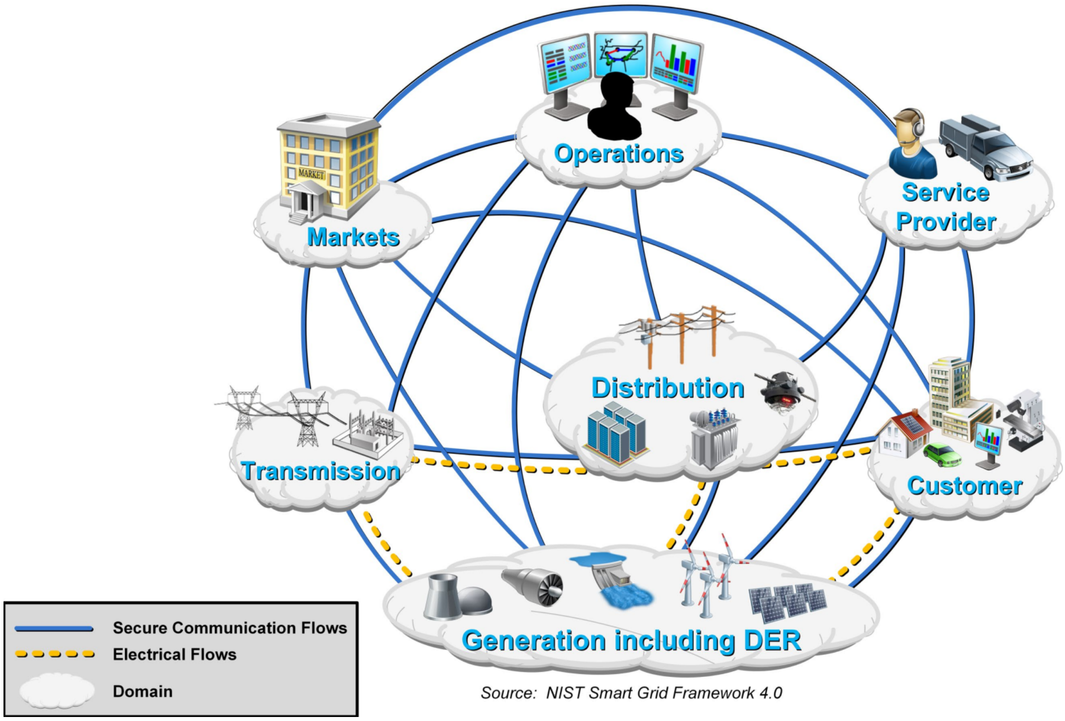
In the past few years, several industries witnessed breakthroughs in artificial intelligence by employing *generative* learning. Computers started to generate paints [5] and turn design mock-ups into running codes [6]. Generative models will play a fundamental role in the future development of machine learning. This chapter presents the importance of machine learning and highlights the differences between discriminative and generative models. The next sections identify three smart grid applications where generative learning is of special interest. Each corresponding state-of-the-art is thoroughly discussed. Finally, research motivations, objectives and thesis outline are presented.

1.1 Machine learning in smart grid

The electrical power grid has evolved from a one-way electromechanical grid to a two-way smart grid. In their latest release, the National Institute of Standards and Technology (NIST) lays down a conceptual model of *smart grid* (Figure 1.1) [3]. While energy production and consumption remain straightforward, information exchange has grown into high complexity. As clearly depicted by the conceptual model, connectivity is a major aspect of a smart grid. It is even anticipated that a smart grid will eventually evolve into a *neural grid* wherein ubiquitous connectivity and cloud-based artificial intelligence will play major roles [7].

The large volume of data brought by connectivity motivates advanced techniques for planning and control. On the other hand, the emerging complexity is accompanied by cybersecurity concerns. The need for machine learning techniques to deal with data volume, variety, and cybersecurity becomes inevitable.

Machine learning (ML) is the science of programming computers so they can learn from data [8]. So, typically, machine learning does not entail the knowledge of physics. In its basic form, training data is ingested into the model's function or hypothesis, the output is evaluated against *ground truth* and the function is updated to minimize the error. ML becomes very useful when the system under consideration is complex, i.e. a long list of physical rules is required to describe the system. In general, rule-based systems are inconvenient to address complex problems. Further, they lack the flexibility to adapt to new requirements [9].



Source: NIST Smart Grid Framework 4.0

Figure 1.1: NIST smart grid conceptual model [3]

ML is usually classified based on the amount of supervision required during training. In *supervised* learning, the training data includes the target i.e. labels. Consider a *classification* problem where an image is needed to be identified whether it is for a dog or cat. The training dataset will include different images of dogs and cats (i.e. *observations*) along with their proper labels. If the targets are numeric values (e.g. load forecasting) rather than labels, the problem is categorized as *regression* problem. Alternatively, observations may be referred to as *examples* or, simply, *samples*. Some common ML algorithms for supervised learning include logistic regression, k-nearest neighbours, support vector machines, decision trees, and neural networks. In unsupervised learning, the training dataset is unlabelled. The ML algorithm detects patterns among the training examples based on their input *features*. Clustering is a typical example of unsupervised learning. For example, a person may need the ML algorithm to group various songs in few playlists based on their audio features such as speechiness, loudness, instrumentalness,..etc [10]. Several algorithms for clustering are developed such as k-means and Gaussian mixture. When true cluster labels are available, the model's performance can be validated against these labels.

ML has several applications in the smart grid such as load forecasting, price prediction, fault detection, detection of malicious attacks, and others. References [11, 12, 13] review various contemporary applications of ML in smart grid.

1.2 Generative vs. discriminative models

The previous section mentioned that ML models are broadly classified as either supervised or unsupervised. ML models may be also classified as either discriminative or generative. Discriminative models are synonymous with supervised models where labelled datasets need to be used for training. Given a specific training example, a discriminative model estimates the probability of a specific label. For instance, consider a discriminative model that is trained to discriminate the paintings of Oscar-Claude Monet from others. If a new painting is fed to the model, the output will be the probability that the input sample is painted by Monet. However, unlike generative models, the discriminative model will not able to generate a painting that seems to be painted by Monet. Hence, generative models, estimate the probability of observing a training example. Generative models will be discussed further in 2.1.2.

On the demand side of the smart grid, generative models are quite useful. In this thesis, generative models are used to simulate residential electrical loads such as dishwashers, cloth dryers..etc. Further, a generative model is used to disaggregate the smart meter readings to identify the status of individual loads (i.e. whether on or off) without any hardware installations. This is also known as non-intrusive load monitoring. On the transmission side, power state estimation and bad data detection is another area where generative models can be exploited. State-of-the-art related to these three topics are discussed in subsequent sections.

1.3 State-of-the-art

The following subsections present a review of the literature and state-of-the-art proposals related to the three core topics, namely, residential load simulation, non-intrusive load monitoring and power state estimation. While the focus of this thesis is on data-driven models, traditional techniques are occasionally presented for comparison purposes. When discussing the literature related to state estimation, state-of-the-art data-driven proposals for bad data detection are also included.

1.3.1 Residential load simulation

Broadly speaking, residential load simulation (RLS) can be classified into two categories: rule-based and data-driven. The rule-based approach involves the representation of a certain load using a set of electrical components with known physical behaviours (e.g. resistors, inductors..etc.). Measurement-based modelling is another common form of data-driven technique. In this case, measurements are taken for a specific load and a statistical model (e.g. polynomial) is fitted with the measured quantities in order to find the model's parameters (e.g. polynomial's coefficients). Although it is built on statistics, measurement-driven techniques involve a certain level of expert knowledge since the model structure needs to be pre-determined (e.g. an expert shall decide the type and order of the function to be used). With loads becoming more sophisticated and with the necessity to model the temporal behaviour of the load in addition to its electrical characteristics, more complex expert-free data-driven techniques are devised [14].

When the temporal dimension is not considered, this is usually called 'load modelling'. Here, the overall goal is to develop a mathematical representation of the electrical characteristics. A common example is ZIP model where the model includes a constant impedance (Z), current (I) and power (P). In some literature, the same model structure can be applied to both individual loads and aggregate loads. For instance, the Electric Power Research Institute (EPRI) proposes a model that combines ZIP, exponential and frequency-dependent models. If the model is used to represent an individual load rather than an aggregate load, then the initial operating conditions stated in the devised model equation shall be replaced by the ratings of the individual load under consideration [14].

RLS involves modelling *both* the electrical characteristics and the consumer's behaviour. Occasionally, literature refers to RLS as the process of generating *load profiles*. Once again, the word 'load' may be used for both the individual and the aggregate loads. However, it is common to use the term "load profile" to refer to modelling the aggregate signal at the bus level in the distribution network rather than modelling an individual load.

A comprehensive RLS model was early proposed by [15]. The proposal includes models to capture both electrical characteristics and consumer behaviour. The model is meant to be used by utilities to predict the temporal magnitude variations of the aggregate residential loads for better power generation planning. The author follows a bottom-to-top approach by modelling the individual loads in a residence, then sums them up to reach the model for the 'residential load'. By the same token, an 'area load' is the summation of all 'residential loads'. The author had to classify individual loads into types, then model each single expected function corresponding to each type. Each specific function is thoroughly studied. For instance, the switching function is impacted by the availability (i.e. probability that someone is available), the proclivity (i.e. the probability that someone will operate the load's switch) and the normal-cycle which is impacted by the internal automatic switching mechanism of the load. The availability is further impacted by a consumer's sleeping hours and so forth. Similarly, proclivity is impacted by consumer's convenience and so forth. For every load, a probabilistic model with many random variables (e.g. breakfast length, travel time to work, ..etc.) is constructed. This results in a highly sensitive complex model for each individual load. Moreover, if a new load needs to be added, the whole design process shall start over. This complex process makes it impractical to cope with the pace of introducing new appliances to the consumers' market.

Reference [16] proposes and develops Smart Residential Load Simulator using MATLAB's Simulink toolbox. The modelled loads included: smart thermostat, air conditioner, furnace, water heater, dishwasher, clothes washer, dryer, light, pool pump and fridge. In addition, wind and solar power generation, as well as battery sources, are modelled. The user selects his loads of interest and enters the current ambient temperature, the number of family members and their ages. Appliances may be simulated individually or as a group. The simulation result outputs the consumed and generated power by each appliance and source. Consumption/Generation profiles may be plotted individually or as a group. Other useful information is provided such as cost and gas consumption. The underlying models are rule-based where various loads are represented in terms of electrical components. The framework lacks the flexibility of easily adding new loads. Further, the consumers' habits are manually entered by consumers rather than being learnt from consumers' historical behaviours.

Reference [17] categorizes individual loads into load types. For each type, three variables are modelled: pre-set power demand per day per operation, probability of operation and the number of appliances of each type per country. The model uses historical data without availing the advancement in machine learning. The complexity of the model is not far from what was proposed by [15] and, hence, it makes it difficult to add any new loads.

Some researchers focused on simulating special types of loads. For instance, [18] uses physical measurements for an AC and reproduces these measurements without employing any probabilistic model. The procedure is load-specific and can not be extended to other loads. Reference [19] provides an overview of HVAC simulation techniques and highlights the complexity-uncertainty trade-off.

Instead of tackling individual loads, reference [20] categorizes consumers based on dwelling type and constructs a load profile for each type. A probabilistic mathematical model is proposed. Each dwelling type is assumed to comprise a set of appliances. Each appliance is represented by its average power consumption rather than a consumption pattern. This level of abstraction makes the model ineffective for detailed studies on the demand side such as NILM.

Reference [21] uses MATLAB's Simulink library to model and simulate several appliances. The authors combine both finite-state and state-space modelling techniques to simulate the appliances. The models incorporated many parameters that increased error margins.

In data-driven approaches, the operational characteristics of loads are not imposed by the designer; rather, they are learnt from historical operational data. This reduces complexity while achieving a high level of accuracy. However, both rule-based and data-driven models perform better if more features (such as the weather condition) are incorporated in modelling the system. A comprehensive review of data-driven approaches is given by [14]. Gaussian Mixture and Markov Models are used in references [22] and [23]. Recurrent neural networks are used by [24] to learn the power-voltage functions of various loads. [25] uses an unsupervised learning technique to model loads. Another ML-based solution is proposed by [26].

Following the success of Generative Adversarial Networks (GANs) in generating synthetic images, researchers tried to carry that success to other disciplines. References [27] and [28] applied GANs to generate residential load profiles. In both cases, the load profiles simulate the aggregate load at the household level. Such models may be used by utilities for planning and load forecast purposes; however, they remain very limited for other demand-side studies such NILM.

In its basic form, a GAN consists of two neural networks (namely the generator and the discriminator) playing a min-max game. The vanilla GAN was first proposed by [29] and it will be thoroughly discussed in section 2.1.2. Later, several types of GANs (e.g. DCGAN, Wasserstein, ..etc.) emerged as presented by [30]. Evaluation of GAN's performance is a topic of research by itself. Since a major application of GANs is to generate synthetic images, inception score (IS) is occasionally used for evaluating a GAN's performance. IS measures the KL-divergence between the response produced by the generated image and the average response of all generated images [31]. Alternatively, some distance measures between the real and synthetic data are calculated. Among the divergence scores used are Jensen-Shannon Divergence, f-divergence, Wasserstein distance and Maximum-Mean Discrepancy [32].

1.3.2 Non-intrusive load monitoring

Non-intrusive load monitoring (NILM) is a technique used to identify both *power consumption* and *operational schedule* of individual loads from the measurements of aggregated power consumption data [33]. NILM does not involve any hardware installations; rather it involves an algorithm to identify individual loads based on measurements recorded by the smart meter. Historical data about the consumption patterns of individual loads along with the aggregate load are typically necessary to train NILM algorithms. Reference [34] lists some of the most common datasets used in NILM studies. NILM is mainly used to propose demand response programs to consumers based on their power consumption patterns. For instance, if a consumer operates an appliance during on-peak hours, a utility may send a notification message about the savings that can be made if that appliance is operated during off-peak hours. Detailed bill information may be also sent to improve the usage habits of consumers. Other benefits include occupancy detection and illegal load detection [35].

Reference [36] proposes a classifier that classifies appliances into three distinct units: motor, resistance and electronic. Appliances are then identified using both steady-state and transient responses. A high sampling rate of 10 Hz is needed for that purpose. Besides, the algorithm heavily depends on power factor measurement.

Hidden Markov Model (HMM) is widely used to disaggregate the main feed power signal into individual loads. In general, HMM for NILM involves both steps of structure modelling and parameter estimation. This will be described in details in section 3.1.1. In [37], power consumption of selected appliances is sampled at a rate of 0.1Hz. The profile for each load is pre-processed for edge detection. After that, HMM is used to solve the load identification problem. While the proposed sampling rate is considered low, it is still not favourable to many utilities. In general, NILM algorithms that are intended to be applied to smart meters from all over the world need to be built based on a low sampling rate of active power only [38]. In practice, a sampling interval of few minutes of active power is well entertained by utilities.

In [39], the number of loads is identified manually. Dataset is preprocessed and HMM is built for each load. Gaussian distribution is considered to relate the hidden states with the power emitted by each state. Once transition and emission matrices are calculated for each load, the model for the aggregated signal may be obtained. Viterbi algorithm is finally used to determine the likely states for specific emissions. Gaussian distribution is associated with restrictive assumptions that do not capture all nuances and relationships between hidden states and observations for all types of loads. When there is a large number of loads, the process of combining such loads becomes cumbersome. The sparsity of HMM is capitalized in [39] to increase computational efficiency. Factorial HMM is utilized in [40] to develop separate Markov chains for each appliance. However, these do not capture inter-dependencies between loads.

On the other hand, neural networks are frequently used in NILM. For instance, [41] disaggregates Air Conditioning load. Training Data of one-minute granularity is taken from houses in Austin Texas for one month. The next month is used for validation. Both Feed Forward Multi-Layer Perceptron (MLP) network and Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) are explored. MLP outperformed LSTM. The study is limited to one load which makes it less attractive as an overall solution for NILM.

Reference [42] suggests modelling the individual loads using HMM while modelling the aggregated signal using Deep Neural Network (DNN). The model is experimented on REDD dataset, which is a public dataset for energy disaggregation research [43]. The framework calculates the most likely state sequence of each load. Viterbi algorithm is used to estimate the underlying state sequence. Finally, the most likely observation sequence of the aggregate signal is estimated by maximizing the conditional distribution given the estimated state sequence. In addition to being computationally intensive, deep learning requires large volumes of data to be processed for learning purposes.

Integer Programming Optimization is also used to tackle the NILM problem [44]. Appliances are assumed to attain one or more states when they are ON, i.e. Non-Off states. The aggregated power is a summation of these non-off states. A binary vector (0 or 1) is used to indicate the current state of the appliance at time instance k. Basically, these are the unknown decision variables for each appliance. The aggregated power at the k^{th} time instance, which is known for us, is equal to the combination of all loads with their corresponding power values multiplied by their unknown binary vectors. The objective is to find the current binary values for each load that minimizes the squared difference between the known aggregated value and the said equation. The constraints added to the optimization problem include fine details of the load patterns that may not be simply obtained from appliances' data sheets.

Reference [35] lists other techniques such as Denoising Autoencoders, Convolution 1-Dimensional Neural Networks and LSTM. Reference [45] extensively researched neural networks for NILM. Three architectures of neural networks are investigated: LSTM, Denoising Autoencoders and Regression Network. For each architecture, a separate network is trained for each load. This makes it a costly solution. Five appliances are considered in this study. Hence, five neural networks is constructed for each of the architectures above. The input to every NN is a *window* of aggregated power. The output is the power consumed by the appliance pertaining to that NN. In case of the regression network, the output consists of three nodes standing for average power consumed by the pertaining appliance, start time and end time of the appliance's cycle relevant to the aggregated input window. The sampling time considered is 6 seconds. As discussed earlier, granularity in the range of minutes is usually more attractive to utilities.

LSTM is also proposed by [46]. The architecture is meant to estimate the ON/OFF state of each appliance without an estimation of the average power consumed. The number of output nodes is equal to the number of appliances with each assuming either 1 (ON) or 0(OFF). Typically, the input is only the aggregated power. An ensemble model that combines both LSTM and Feed Forward Neural Network is proposed by [47].

Reference [48] extensively explores various methods used in NILM. The parametric approach assumes probabilistic distribution while the non-parametric approach induces the model from the data. Researchers propose Multi-Label Classification as an effective solution for the problem. However, when evaluated against Factorial Hidden Markov Chain (FHMM), Multi-Label *k*NN approach (ML*k*NN) was superior only for selected appliances [48]. It is imperative that the importance of multi-labelling stems from the fact that the aggregated signal is dealt with as a summation of independent individual signals representing each load. This concept of superposition is fundamental in building the model.

1.3.3 Power system state estimation

Power system state estimation (PSSE) is a fundamental tool used to monitor the operation of a power grid [49, 50]. A set of redundant measurements acquired by various measuring instruments is used to estimate the network's states i.e. voltage phasor of each bus [51]. Power state estimation will be discussed in details in section 4.1.1. Traditionally, model-based techniques to estimate states such as Weighted least-squares (WLS) are well established [52, 53, 54]. WLS estimates states by minimizing the measurement error which involves iterative calculations of non-linear equations. Occasionally, decoupled formulation and DC approximation may be used to reduce complexity [55, 56, 57]. WLS is further developed to detect bad data and identify tampered measurements. Some assumptions are typically made about the probability distributions and correlations among measurement errors. Since WLS is executed online and due to its high computational complexity, researchers investigated data-driven models to estimate states. In their inference mode and once trained, data-driven models substantially reduce computational complexity.

Data-driven techniques utilize machine learning constructs to extract underlying patterns from grid measurement and state data logs for PSSE. Reference [58] proposes Multilayer Perceptron Neural Network (MLP) to estimate states. Power flows and voltages are used as inputs without adding any noise. Estimated states are restricted to voltage magnitudes. The proposal was tested only on IEEE-14 bus system and resulted in 2.18% of mean absolute error.

An MLP is also used by [59] to estimate states. The input includes both voltage angles and magnitudes as recorded by phasor measurement units (PMUs). When a combination of power flow measurements and four PMU measurements were considered in 47-bus system, an average estimation error of 1-4% was achieved. The error of estimating angles was substantially higher than that of estimating magnitudes.

Reference [60] proposes MLP with initial states included in the input vector in addition to flow measurements. The authors focused on optimizing the hyper-parameters and exploring the best back-propagation algorithm. On IEEE-118 bus system, the best-proposed architecture scored 0.178 RMSE for magnitude estimation and 0.164 RMSE for angle estimation.

A physics-aware data-driven framework is proposed by [61] by unrolling the prox-linear iterative solution developed by [62] for least-absolute-value estimation. The proposal comprised a prox-linear Deep Neural Network (DNN) for state estimation. The proposed DNN is a combination of vanilla Feed Forward Neural Network with skip-connections that connect the input (i.e. measurements) to intermediate layers of the network. Tested on IEEE-118 bus system, the prox-linear DNN achieved 2.97×10^{-4} of RMSE normalized by the number of buses. The proposal is further extended by the same authors in [63] to count for pseudo-measurement. RNN is directly used to generate pseudo-measurements based on historical data. Then, the output of the RNN is concatenated with the real measurements to estimate states using the previously developed prox-linear DNN estimator. While the paper shows a competitive performance of the proposed DNN for estimating states, it does not discuss the topic of bad data detection. Further, it is assumed that historical measurement-state pairs are readily available for training purposes.

Reference [64] approached the state estimation problem by partitioning it into smaller problems. The partitioning is based on the installation of μPMU at various locations. The location is optimized by minimizing the size of the resulting block partitions. A neural network is used to estimate the states within a block. The number of layers of the NN is determined by the optimization problem of locating the μPMUs. The proposed estimator scored a normalized MSE of 1.273×10^{-3} when simulated for IEEE-37 network.

To compare various proposals, we consider normalized RMSE. By normalization, we mean dividing the resulting RMSE by the number of buses in the test system. Table 1.1 summarizes the results of the reviewed data-driven based proposals.

Reference	Normalized RMSE% (Magnitudes)	Normalized RMSE% (Angles)	Identifying individual bad measurements	Paired Measurement/State vectors for training
[60]	0.1508	0.139	No	Yes
[61]	0.0297	0.0297	No	Yes
[64]	0.1273	0.1273	No	Yes

Table 1.1: Accuracy comparison of various data-driven state estimators

In general, all reviewed data-driven models for estimating states assume that measurements are authentic. The models perform a direct mapping between measurements and states and no inherent mechanism is provided to detect bad measurements. This is no practical since the resulting estimated states are assumed to be correct, which is not always the case since the input measurement vector may include some bad measurements that result in incorrect states. Further, the proposed models assume the availability of paired measurement/state vectors at various load conditions of the grid. In other words, at a given time instance (aka load scenario or snapshot), the set of observed measurements and the corresponding (*paired*) set of states shall be available for training the model.

The detection of bad measurements is separately tackled using machine learning (ML) algorithms such as multidimensional scaling [65], support vector machine [66], k-nearest neighbour [67] and others. Some approaches combine traditional methods for detecting bad data via Chi-squared test with ML algorithms [68]. The performance of an algorithm depends on the type of the launched attack. In [69], a comparative study is conducted to evaluate common ML algorithms against different types of attacks. In all these studies, bad data detection is approached apart from power state estimation. This involves additional training overhead after state estimation.

Reference [70] proposes a combination of GAN and adversarial auto-encoder (AAE) to detect the unobservable FDI. The AAE is trained to encode the labelled and unlabelled measurements into two hidden representations. The two hidden representations are used in the GAN as conditions added to the GAN generator's noise input. The paper has several vague points with some typos. For instance, the authors assumed that hidden representations follow Gaussian distribution without any justification. In summary, unlike our proposal, the proposed model is based on the availability of labelled measurements. In practice, attack-labelled measurements are not readily available. Another limitation is the fact that, once an FDI is detected, the model is not capable to identify and eliminate the individual tampered measurements. The highest reported classification accuracy for 123-bus distribution system is 96.7%.

Reference [71] focuses on Synchrophasor data attacks when PMUs are deployed. Several types of attacks are defined. The proposed model detects the type of attack. The input is the PMU data collected for a window length of half a minute with an adjustable step from 1 to 10 seconds. The model comprises feature extraction block, modified CNN and a multi-class classifier. Clearly, the model is trained on labelled data. The actual synchrophasor data is provided by FNET/GridEye which monitors the power grid in North America [72]. The attacks are not real; they are generated by the authors.

Reference [73] proposes a framework that detects bad data and re-constructs the contaminated states. The classification part is achieved using an ensemble of ELMs. ELM is a single hidden layer feed-forward neural network that eliminates the need for back prorogation. However, it is quite sensitive to initialization and hence, the authors proposed an initialization scheme. In any case, the proposed classifier needs to be trained on labelled data that include both authentic and compromised samples. On the other hand, the exact compromised (i.e. tampered) measurements are further identified. This is achieved by replacing the contaminated states with forecasted values. The measurement matrix is used to work out the corresponding measurements. Then the normalized residuals are calculated in the same traditional way of state estimation. Finally, the corrupted measurements are eliminated and the new states are re-calculated. Using the forecast states involves forecasting errors. The paper considers 726 measurements for the 118 bus system. They reported detection accuracy of 81.94%.

In summary, and to the best of our knowledge, no model is yet proposed to integrate both state estimation and bad data detection.

1.4 Research motivations

In the following three sections, we present some challenges encountered in the industry and the motivations behind our research.

1.4.1 Residential load simulation

Today, utilities, like London Hydro [74], actively engage customers in maintaining a healthy power grid. Incentives are provided to encourage more economical and sustainable energy consumption patterns [75]. To encourage the active participation of consumers in demand response programs, further research needs to be advanced at the demand side. For instance, load disaggregation studies require data for training and testing. Several datasets are available [34] in different countries with some variations in the quantities being measured and their granularity. Locked datasets need special permission for access. Some public datasets, e.g. PLAID, comprise measurements for a short duration with high sampling rates. Others, e.g. ECO, do not record time stamps. Data acquired on an hourly basis is not that useful since operating cycles of appliances are usually less than one hour. Occasionally, data is acquired on the branch circuit level and not for individual loads. Reference [76] provides a comparison of household energy datasets.

For demand-side studies such as load disaggregation, we define the following criteria to develop useful disaggregation models that can be adopted by utilities :

- Dataset shall include Energy measurements for both individual appliances and the corresponding aggregated smart meter readings.
- Dataset shall indicate the timestamp of each measurement.
- Dataset shall not include erratic readings resulting from power outage or any other reason.
- Dataset shall include various loads and households throughout the year.
- Granularity shall be in minutes. For instance, a common one-hour granularity can not be used in disaggregation studies.

Obtaining physical measurements, i.e. intrusive monitoring, is a challenging process. It involves getting the consent of various consumers to install measuring instruments on their premises. As such, only a few public datasets meet the above criteria. In general, the available public data is quite limited in terms of the number of intrusively monitored houses, the logging duration, and the sampling rates.

We are motivated by the lack of data volume and versatility to design a data-driven generative simulator that learns the physical data distribution from datasets that meet the criteria above (e.g. appendix A) and generates synthetic data from the learnt distribution. As discussed earlier in 1.3.1, rule-based models lack the flexibility of easily adding new loads and learning consumers' habits. On the other hand, among various data-driven solutions, GAN performs well on reduced datasets as demonstrated by reference [77]. The synthetic patterns and habits generated by GANs resemble real-life data. The generated synthetic data is a valuable source for researchers in the fields of demand response and load disaggregation. Reference [78] summarizes the benefits of synthetic data and their applications in various industries.

1.4.2 Non-intrusive load monitoring

As part of Demand Response (DR) programs, utilities encourage consumers to level out their power consumption and avoid peak hours. This has the direct benefit of producing energy at higher efficiency. For instance, a customer may use his cell phone to track his hourly aggregate consumption and manage his appliances in a way to avoid peak hours. However, the consumption of individual appliances is not usually readily available. Providing disaggregate consumption has several advantages:

- Provides energy consumers with almost instant feedback (e.g. using text messaging) when operating an appliance during peak hours. Instant low-level feedback even becomes more important for prosumers (producers/consumers) who use Distributed Energy Resources (DERs) such as solar Photovoltaic to produce energy.
- Collects information about the usage of certain appliances. Such information may be used to identify inefficient appliances, predict failures and propose the replacement of appliances ahead of time.
- Collects information about customers' usage habits of various electrical loads. With such information, optimal schedules to operate electrical loads are proposed to customers to minimize cost while maximizing convenience.
- With minimal hardware installations (e.g. smart hub and smart plugs for selected appliances), automated schedules based on disaggregate data can be used to automatically turn on or off certain appliances.

In brief, smart meters allow customers to be more informed about their aggregated energy consumption. NILM studies go a step further and identify the usage pattern of individual loads without the need to install any additional hardware. In this research, a well-established model used in NILM, i.e. HMM, is combined with another machine learning algorithm, i.e. k-means, to propose a hybrid algorithm that reduces computational cost during training.

1.4.3 Power system state estimation

Electric utilities made substantial investments in the installation of supervisory control and data acquisition (SCADA) systems. The master station or control center processes the gathered data. The main elements of SCADA system include Human Machine Interface (HMI), application servers, communication front-end and external communication servers for data exchange with other control centres. The application servers support Energy Management System (EMS) and historical databases besides other functions [79]. Remote terminal units (RTUs) send back various measurements to the master control center through communication links. Reliable estimation of states needs to be done by the application servers for system monitoring and other applications.

In practice, the transmitted measurements may not comprise the active and reactive power injections. Recall that, for PQ buses, power injections are necessary inputs to conduct power flow analysis. If any of the inputs is missing, traditional power flow analysis cannot be used to estimate states. Moreover, the transmitted measurements are prone to various types of errors such as large measurement and telecommunication errors [80]. Bad data maybe even injected as stealthy attacks in both transmitted measurements and control signals. As such, researchers closely studied the problem of power state estimation which is an important topic in power engineering.

State estimation was first proposed by [55]. After that, traditional state estimators (e.g. WLS estimator) were extensively researched. Typically, state estimators comprise several functions such as topology processing (i.e gather the states of circuit breakers and switches), observability analysis (i.e. determining if a state estimation solution can be obtained from the available observed measurements), bad data processing and network parameter data processing [80]. Yet, in its non-linear formulation, WLS has high computational complexity [81] and occasionally encounters ill-conditioning and non-convergence [82]. As per [83], ill-conditioning can occur in the presence of a large number of injection measurements or due to other factors. Another challenge is the assumption made by WLS about the error distribution. Reference [84] indicates that a least-squares estimator is an unbiased estimator if and only if the model is accurate and the measurement error is statistically distributed. In practice, both conditions may not be true. Finally, traditional estimators may fail to detect all types of attacks. For instance, reference [57] shows that an attack designed as a linear combination of the column vector of the measurement matrix will not be detected by WLS estimator. The measurement matrix will be discussed in details in 4.1.1. Data-driven models are recently researched to tackle these problems. We are motivated to investigate and propose a generative data-driven framework that will estimate power states. The framework will learn the mapping functions and the underlying distribution from the grid's historical data during normal operation. Our proposal shall integrate both states estimation and bad data identification. The topics of topology processing, network parameter estimation and network observability are beyond the scope of this thesis.

1.5 Research objectives

Machine learning (ML) techniques are penetrating various aspects of life and bringing great benefits to users. The application of ML in the smart grid is rapidly evolving. The vast majority of ML applications focus on discrimination-based models. In our thesis, we investigate the applications of generative models on both sides of power demand and power transmission. In the light of the motivations discussed in 1.4, objectives are defined as follows:

- Propose data-driven generative framework to simulate electrical loads that are usually present in households such as dishwashers, cloth dryers and others. The framework shall learn loads' patterns and usage habits. Usage habits include user's availability, proclivity (i.e. tendency to operate the load) and the load's internal automatic cycle. Being generative, the data-driven model will learn the underlying distribution of the habits associated with each load.

- Investigate the application of hidden Markov models in disaggregating the smart meter signal into individual loads. Given the discrete-time sequence of power samples, the proposed model shall identify the on/off status of individual loads (e.g. dishwasher, dryer..etc.) at each time instance.

- Propose a generative data-driven framework to map a set of redundant measurements into a grid's power states. Besides, state estimation, the proposed framework shall be capable of identifying tampered measurements.

The next section presents the thesis outline with the main tasks necessary to accomplish the objectives.

1.6 Thesis outline

Figure 1.2 shows the outline of this thesis. The research focuses on the application of data-driven generative models in the smart grid. Generative frameworks are developed for specific applications in two areas, namely, power demand and power transmission. On the demand side, we focus on two problems: simulation of residential loads and secondly non-intrusive load monitoring. In the area of power transmission, we focus on power state estimation and the associated bad data identification.

Chapter 2 begins with a necessary technical background that is also important for chapter 4. The problem is defined and the proposed framework is presented. Experimental studies then follow with model's training and ending with model's evaluation. A brief description of the public datasets that are used in training and testing the model is given in the appendix.

Chapter 3 begins with a necessary technical background related to the disaggregation problem. The problem is defined and the proposed algorithm is presented. Experimental studies then follow with models' training and evaluation. The same dataset used in chapter 2 is used in this chapter.

Chapter 4 begins with a necessary technical background including an overview of the widely adopted traditional state estimation method. The problem is defined and the proposed framework is presented. Experimental studies start with constructing the necessary dataset that is used in training and testing the proposed model. This is followed by the model's training and evaluation. The model's performance is tested for both state estimation and bad measurement identification.

Chapter 5 concludes with a brief summary and thesis contribution. Possible future work is also discussed.

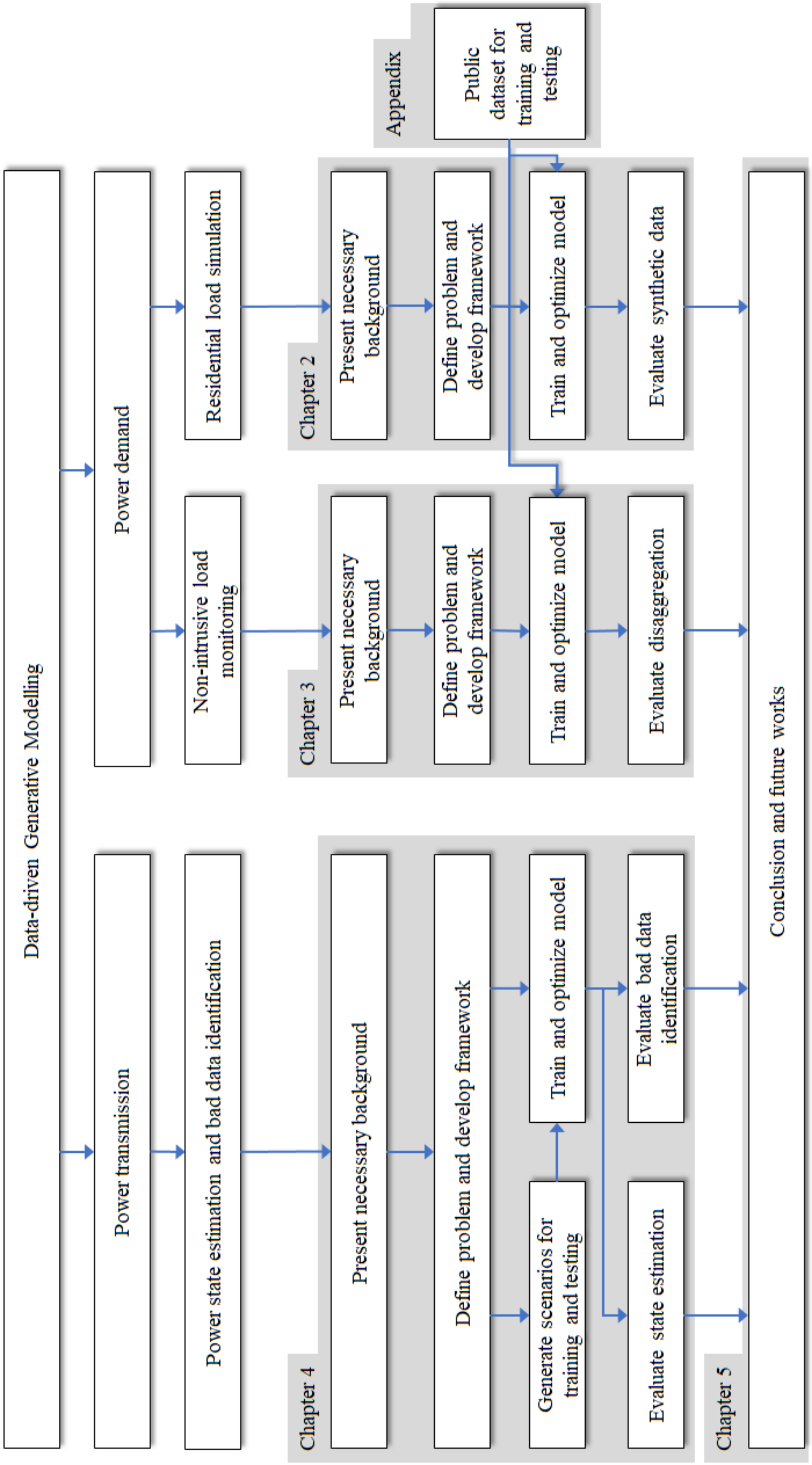

Data-driven Generative Modelling
Power demand
Power transmission
Residential load simulation
Non-intrusive load monitoring
Power state estimation and bad data identification
Appendix
Public dataset for training and testing
Chapter 2
Present necessary background
Define problem and develop framework
Train and optimize model
Evaluate synthetic data
Chapter 3
Present necessary background
Define problem and develop framework
Train and optimize model
Evaluate disaggregation
Chapter 4
Present necessary background
Define problem and develop framework
Generate scenarios for training and testing
Train and optimize model
Evaluate bad data identification
Evaluate state estimation
Chapter 5
Conclusion and future works

Figure 1.2: Thesis outline including the main tasks in the research

Chapter 2

Residential load simulation

This chapter begins with background about Machine Learning (ML) tools and statistical methods that are used to develop our framework. First neural networks (NNs) are introduced as the basic building blocks in the subsequent discussion about generative adversarial networks (GANs). Vanilla GAN and its variations are *generative* models that we use in both this chapter and chapter 4. As a quality measure of GANs, Maximum Mean Discrepancy (MMD) is then discussed. Kernel Density Estimator (KDE), as a statistical non-parametric density estimation tool, follows. The background section is concluded by a discussion about matched-filers that are normally encountered in the literature of digital signal processing.

Following the background, the problem entailing load simulation is defined and followed by the proposed framework. Experimental studies applying the proposed framework are then presented and discussed. Finally, the chapter concludes with a summary.

2.1 Background

2.1.1 Neural networks

The basic block of a neural network is a neuron or a node. The node computes the weighted sum of its inputs and applies an activation function f to the output. A neural network consists of many connected nodes arranged in layers. Since each node is a mathematical function mapping inputs into an output, the neural network is essentially a nested mathematical function [85]. Consider the neural network in figure 2.2. The network consists of four inputs, two hidden layers and two outputs. The layers are fully connected as depicted by the arrows. This is typically called Multilayer Perceptron (MLP) neural network. The input vector is $\mathbf{x} = [x_1\, x_2\, x_3\, x_4]^T$. The output vector is $\mathbf{y} = [y_1\, y_2]^T$. Each node is represented by the gray rectangle. For the i^{th} node in l^{th} layer, its input x_i^l is the weighted sum of all the outputs of the previous layer, i.e. $\mathbf{a}^{l-1}$. Assume the number of nodes in the l^{th} layer is M and the number of nodes in the $l-1^{th}$ layer is N, then we can write:

$$\mathbf{z}^l = W^l \mathbf{a}^{l-1} \tag{2.1}$$

$$\mathbf{a}^l = f^l(\mathbf{z}^l) \tag{2.2}$$

where

$$\mathbf{z}^l = [z_1^l\, z_2^l \ldots z_M^l]^T$$

$$\mathbf{a}^{l-1} = [1\, a_1^{l-1}\, a_2^{l-1} \ldots a_N^{l-1}]^T$$

$$W^l = \begin{bmatrix} w_{10}^l & w_{11}^l & w_{12}^l & \cdots & w_{1N}^l \\ w_{20}^l & w_{21}^l & w_{22}^l & \cdots & w_{2N}^l \\ \vdots & \ddots & & & \\ w_{M0}^l & w_{M1}^l & w_{M2}^l & \cdots & w_{MN}^l \end{bmatrix}$$

Note that W^l is $M \times (N+1)$ matrix since its first column corresponds to *biases* added to nodes. For the first hidden layer, the input to the nodes is $\mathbf{x} = [1\, x_1\, x_2 \ldots x_{N_x}]^T$. Clearly, this can fit in equation 2.1 by setting $\mathbf{a}^0 = \mathbf{x}$. Similarly, the output $\mathbf{y}$ may be thought of as $\mathbf{z}^{L+1}$ where L is the number of hidden layers.

Given an input $\mathbf{x}$, the neural network is optimized to predict an output $\mathbf{y}$. For a neural network with L hidden layers, the optimization process finds all weight matrices i.e. $W^l\ \forall l = 1 \ldots L+1$. Due to the non-linearity introduced by the activation functions $f^l\ \forall l = 1 \ldots L$ in equation 2.2, no global optimization is guaranteed. Optimization is carried out using a training dataset. Three main processes are involved during optimization. When the training dataset is ingested to the input of the neural network, a *forward propagation* according to equation 2.1 and 2.2 takes place. The output resulting from the forward pass is compared with *ground truth* using a *cost* or *loss* function C. The cost function is chosen to match the problem under consideration. For instance, mean-squared error may be used in problems where an input vector needs to be mapped to an output vector. In this case, the squared errors between the resulting output components and their ground truth values are averaged. If a problem is related to binary classification, cross-entropy may be used as a loss function.

Each component w_{ij} in all weight matrices is adjusted in a way to minimize the loss C. The is typically achieved by gradient descent algorithm which is summarized by equation 2.3.

$$w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial C}{\partial w_{ij}} \tag{2.3}$$

To calculate the gradients in equation 2.3, the *back propagation* algorithm uses the chain rule. For instance, in figure 2.2 and given the loss function $C = f(\mathbf{y})$, we can write:

$$\frac{\partial C}{\partial w_{13}^2} = \frac{\partial C}{\partial z_1^2} \cdot \frac{\partial z_1^2}{\partial w_{13}^2} = \frac{\partial C}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial z_1^2} \cdot a_3^1 = \frac{\partial C}{\partial a_1^2} \cdot f^{2\prime}\left(z_1^2\right) \cdot a_3^1 \tag{2.4}$$

Clearly, $a_3^1 = \partial z_1^2 / \partial w_{13}^2$ follows from equation 2.1. Further, $f^{2\prime}$ is the derivative of the activation function f^2. Usually, activation functions are differentiable. Some examples include rectified linear (ReLU), leaky ReLU and Sigmoid as shown in figure 2.1 [1]. The sigmoid function is quite useful as it maps its input between 0 and 1 so it can be used in the output node for a binary classification problem.

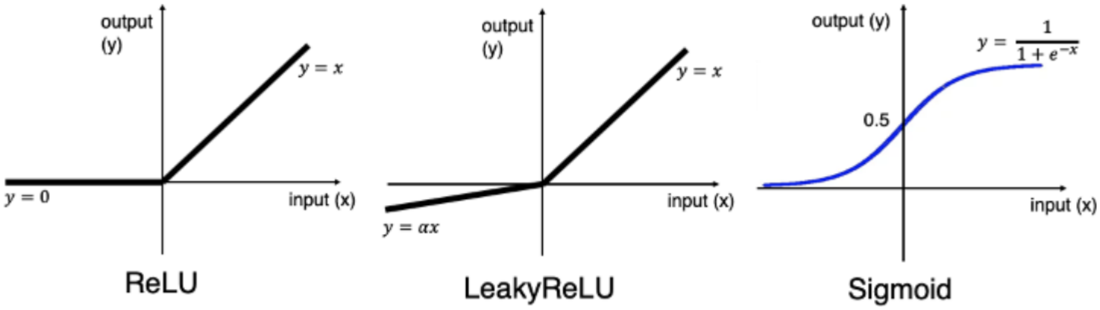


Figure 2.1: Examples of activation functions

In summary, weights are first initialized with random values, forward propagation is executed, the loss function is calculated, and backpropagation is executed to calculate the gradients and update them according to gradient descend. This is iterated until the desired accuracy is achieved.

For neural networks that are used for multi-class classification (e.g. classify an image if it is for dog, cat or others), the soft-max function given in equation 2.5 is typically used at the output of the classifier [8].

$$\hat{p}_k = \frac{\exp\left(s_k(\mathrm{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathrm{x})\right)} \tag{2.5}$$

where:

- $\hat{p}_k$ is the estimated probability of class k given the sample x.

---

[1] Image courtesy of reference [86]

- K is the number of classes.
- $s(\mathrm{x})$ is a vector containing the scores of each class for the sample x.

A widely used loss function in multi-classification problem is the categorical cross entropy given in equation 2.6 [8].

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right) \tag{2.6}$$

where:

- m is the number of training samples.
- $y_k^{(i)}$ equal to 1 if the ground-truth class for the i^{th} sample is k; otherwise, it is equal to 0.

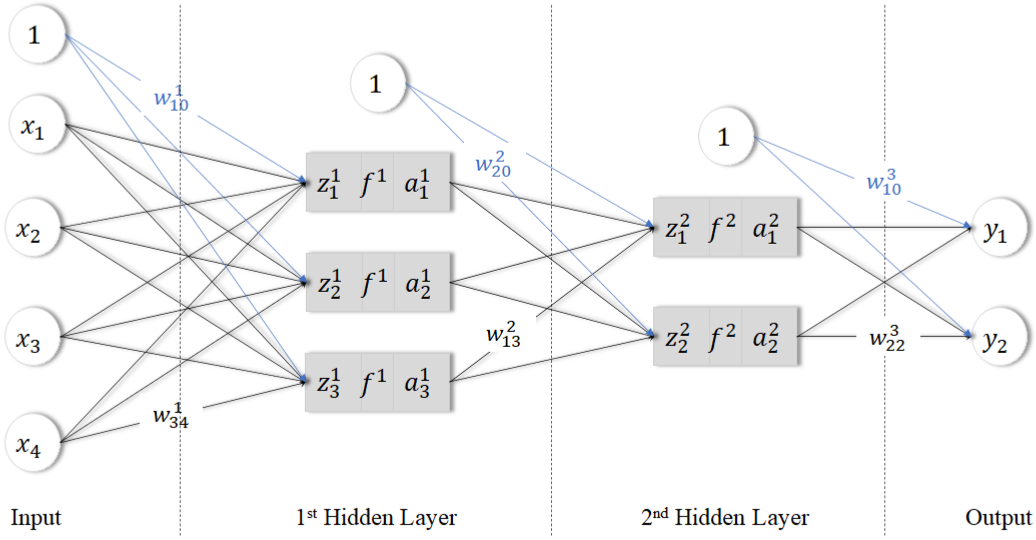


Figure 2.2: Neural Network

2.1.2 Generative adversarial networks

Statistical models can be classified as *discriminative* or *generative*. Discriminative models are typically encountered in classification problems. For instance, a model that classifies an image as being for 'dog' or 'cat' is a discriminative model. The model's training set comprises several observations and their corresponding labels. Observation is typically multi-dimensional since it includes a number of features. Discriminative models estimate conditional probabilities while generative models learn distributions. Let an observation be denoted **x** and its label

y. A discriminative model is trained to estimates $p(y|\mathbf{x})$ while a generative model is trained to estimate $p(\mathbf{x})$ [86]. Occasionally, generative models estimate distributions conditioned on labels i.e. $p(\mathbf{x}|y)$.

Discriminative models have more applications than generative models[86]. For instance, we are more interested to classify tweets as positive or negative than generating tweets. Further, evaluating generative models is more challenging. In discriminative models, estimated labels are tested against ground truth labels. On the other hand, consider a generative model that produces fake paints for Vincent van Gogh. These synthetic images have no real pairs painted by Gogh in order to make a one-to-one comparison. Accordingly, when developing generative models, special care needs to be given to the model's evaluation.

A Generative Adversarial Network (GAN) has many variations such as Deep Convolutional GAN (DCGAN), Stack GAN, Info GAN, Wasserstein GAN and others. These variations are typically used in the context of generating images [87]. In its basic form, a vanilla GAN consists of two multi-layer perceptron networks (MLPs) namely the *generator* (G) and *discriminator* (D) (figure 2.3. The generator's input is noise ($\mathbf{z} \sim p(\mathbf{z})$) and its output $G(\mathbf{z})$ is synthetic or fake data. In other words, the generator's target is to learn the underlining distribution of the training data p_{data}. On the other hand, the discriminator accepts two inputs: the first is the *'real'* training data ($\mathbf{x} \sim p_{datat}(\mathbf{x})$) and the second is *'fake'* synthetic data generated by the generator. The discriminator outputs a single scalar indicating whether the input samples are real or fake. The discriminator's output plays a vital role in optimizing the parameters of both the discriminator and the generator (dotted lines in figure 2.3).

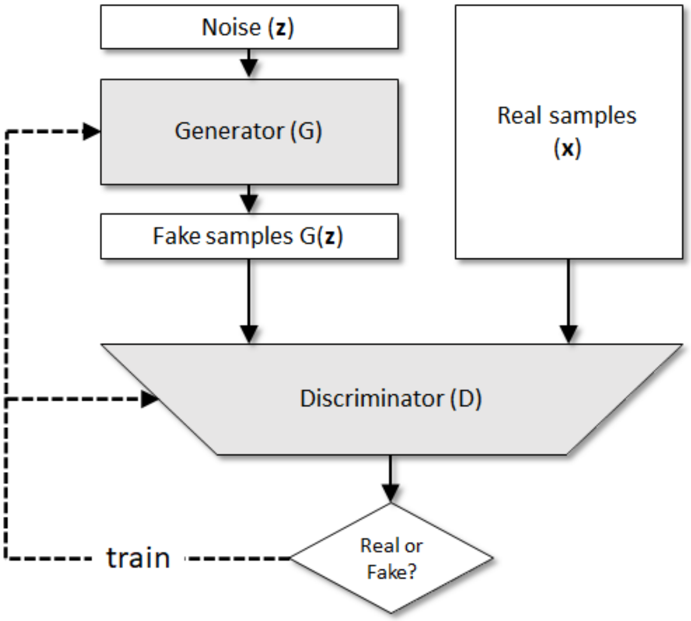


Figure 2.3: Generative Adversarial Network

Refer to equation 2.7 which is first proposed by [29]. The generator (G) is trained to minimize $\log(1-D(G(\mathbf{z})))$ or essentially maximize $D(G(\mathbf{z}))$. On the contrary, the discriminator is trained to maximize $(\log(1-D(G(\mathbf{z})))$ or essentially minimize $D(G(\mathbf{z}))$. This means that the generator and the discriminator are playing a min-max game. GAN is a zero-sum non-cooperative game. If one player wins, the other loses. Zero-sum game converges when Nash equilibrium is reached. This is the state reached when the first player will not change action regardless of the second player and vice versa.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))] \tag{2.7}$$

The generator tries to generate samples that will not be labelled by the discriminator as fake, while the discriminator tries to label all samples generated by the generator as fake. Further, the discriminator is trained to maximize $D(\mathbf{x})$, i.e. labelling true samples as real. In other words, the generator learns the distribution p_g over data $G(z)$ where $z \sim p_z$.

Consider training G till it learns the distribution p_g and fix it. Then train D to attain optimality, i.e. reach the best D that can discriminate p_{data} from p_g. Reference [29] shows that such optimal discriminator is given by equation 2.8 below.

$$D_G^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2.8}$$

Note that in equation 2.8, the less that G learns the data distribution, the closer D is to 1, i.e. D can easily discriminate real and fake samples. Substituting equation 2.8 in equation 2.7, we reach equation 2.9 below.

$$C(G) = \max_D V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \tag{2.9}$$

Divergence scores measure the difference between two probability distributions and can be used to evaluate GAN's performance. The min-max game illustrated by equation 2.7 attempts to learn the underlying distribution from which the real samples were drawn. In essence, this reduces the min-max game to the problem of minimizing a divergence score between the synthetic data and the real data distribution. As per [29], it is shown that his proposed objective function in equation 2.7 approaches a problem of minimizing the Jensen-Shannon divergence (JSD).

JSD is based on Kullback-Leibler (KL) divergence which is also called relative entropy. Let P and Q be the probability mass functions with the same support χ, then the entropy, cross-entropy and KL divergence are given receptively by [88]:

$$H(P) = -\sum_{x \in \mathcal{X}} P(x) \log P(x) \tag{2.10}$$

$$H(P, Q) = -\sum_{x \in \mathcal{X}} P(x) \log Q(x) \tag{2.11}$$

$$KL(P \| Q) = H(P, Q) - H(P) \tag{2.12}$$

JSD is a symmetric version of KL divergence above and given by:

$$\mathrm{JSD}(P \| Q) = \frac{1}{2} KL(P \| M) + \frac{1}{2} KL(Q \| M) \tag{2.13}$$

where $M = \frac{1}{2}(P + Q)$

Setting $P = p_{data}$ and $Q = p_g$ in equation 2.13, and as showed by [29], we can write 2.9 as follows :

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{2.14}$$

Hence, the min-max problem defined in 2.7 approaches a problem of minimizing the Jensen-Shannon divergence (JSD).

In *conditional* generative adversarial networks, proposed by [89], both generator and discriminator are conditioned on extra information such as class label. For instance, instead of having a GAN trained to generate *any* fake paint of Vincent van Gogh, a conditional GAN may be constructed to generate a fake paint exclusively for natural scenes. The conditional GAN is illustrated in figure 2.4. The difference is mainly due to the addition of a label (i.e. condition) to the inputs of both the generator and the discriminator. It is imperative that the training set shall be labelled. Consider labels $y^i \ \forall i : 1, \ldots, N$ where N is the number of labels associated with M number of data points (i.e. examples). If the first training example has label y^i , then y^i shall be concatenated to the noise vector $\mathbf{z}$ and the generated fake example $G(\mathbf{z})$. Embedding may be used alternatively with concatenation.

The min-max game between the generator G and the discriminator D in a conditional GAN is described by equation 2.15.

$$\min_G \max_D V(D, G) = \mathbb{E}_{(\mathbf{x},y)\sim p_{data}(\mathbf{x},y)}[\log D(\mathbf{x}|y)] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}|y)|y))] \tag{2.15}$$

where $(\mathbf{x}, y)$ is a pair of real example $\mathbf{x}$ and its corresponding class label y drawn from the distribution $p_{data}(\mathbf{x}, y)$, and $\mathbf{z}$ is the input noise vector drawn from the random noise $p_{\mathbf{z}}(\mathbf{z})$. The same min-max game applies for both equations 2.7 and 2.15.

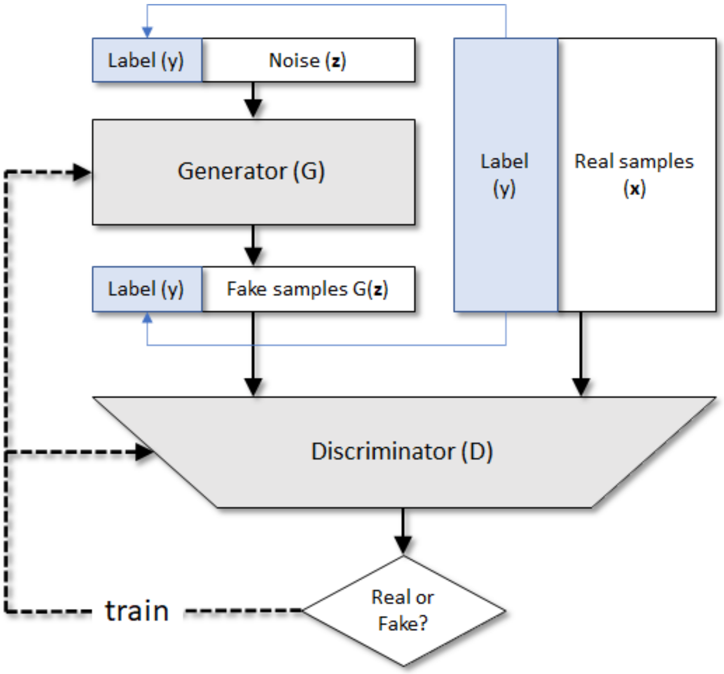


Figure 2.4: Conditional Generative Adversarial Network

Equation 2.7 is the loss function used in vanilla GAN. In general, GAN is notoriously difficult to train [90]. As such, other variations of vanilla GAN are introduced such as Wasserstein GAN [91], least-square GAN [92] and others. In this chapter, we focus on the conditional GAN. While the addition of a label (i.e. condition) affects the architecture of the generator and the discriminator, it has no impact on the objective function as defined by the vanilla GAN in equation 2.7. Hence, we alternately refer to the conditional GAN as vanilla GAN.

In addition, several problems are encountered. Non-convergence occurs when the training parameters (i.e. weights and biases) fail to converge. Model collapse occurs when the generator keeps generating the same set of samples. Over-fitting occurs when the generator produces outputs indistinguishable from the training set [93]. Last, but not least, GAN is quite sensitive to the selection of the hyper-parameters such as the number of nodes in a certain layer.

2.1.3 Maximum-mean discrepancy

Divergence scores measure the difference between two probability distributions and can be used to evaluate GAN's performance. As per [29], when the discriminator is optimized, GAN reduces to a generative model that minimizes the Jensen-Shannon divergence (JSD). JSD is based on Kullback-Leibler (KL) divergence which is also called relative entropy.

Recall that in GAN, the generator is trying to learn a distribution Q that is as close as possible to the underlying distribution P from which the training examples were drawn. Both P and Q are defined on the same probability space $\mathcal{X}$. Since we only have samples from both distributions (i.e. distribution of real data and distribution of fake data), we need to use a two-sample test as a quality metric. Maximum Mean Discrepancy (MMD) has a lower computational cost when compared with other two-sample tests [32]. MMD is used to test whether two distributions are different by finding a well-behaved function that is large on the points drawn from the first distribution and small on the points drawn from the second distribution. The difference between the mean function values on the two samples is called MMD. When MMD is large, the samples are likely from different distributions.

Let the features vector for i^{th} real example be $\{\mathbf{x}_i\}_{i=1}^{m} \sim p(\mathbf{x})$ and the features vector for i^{th} fake example be $\{\mathbf{y}_i\}_{i=1}^{n} \sim q(\mathbf{y})$. Also, let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$, then MMD is defined as:

$$\mathrm{MMD}[\mathcal{F}, p, q] := \sup_{f \in \mathcal{F}} \Big(\mathbf{E}_{\mathbf{x}}[f(\mathbf{x})] - \mathbf{E}_{\mathbf{y}}[f(\mathbf{y})] \Big) \tag{2.16}$$

The is found to reduce to [32]:

$$MMD = \Big(\frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k\big(\mathbf{x}_i, \mathbf{x}_j\big) + \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k\big(\mathbf{y}_i, \mathbf{y}_j\big) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k\big(\mathbf{x}_i, \mathbf{y}_j\big) \Big)^{\frac{1}{2}} \tag{2.17}$$

where $k(.)$ is a kernel and typically chosen to be Gaussian radial kernel defined as:

$$k(\mathbf{x}_i, \mathbf{y}_j) = \exp\left(- \frac{\left\| \mathbf{x}_i - \mathbf{y}_j \right\|^2}{2\sigma^2} \right) \tag{2.18}$$

where σ is a free parameter.

2.1.4 Kernel density estimator

In parametric estimation, the sample is assumed to be independent and identically distributed drawn from a distribution family (e.g. Gaussian) with unknown parameters (e.g. mean and standard deviation). We find the unknown parameter by maximizing the product of the likelihoods of the sample points. This is called maximum likelihood estimation. In non-parametric estimation, no assumption is made about the distribution family that the sample was drawn from. Rather, estimation is made exclusively based on the available training data points.

A histogram is typical non-parametric estimator. Consider a histogram with a bin width of h and a sample $X = \{x_t\}_{t=1}^{N}$ where N is the number of sample points., the histogram estimator is [94]:

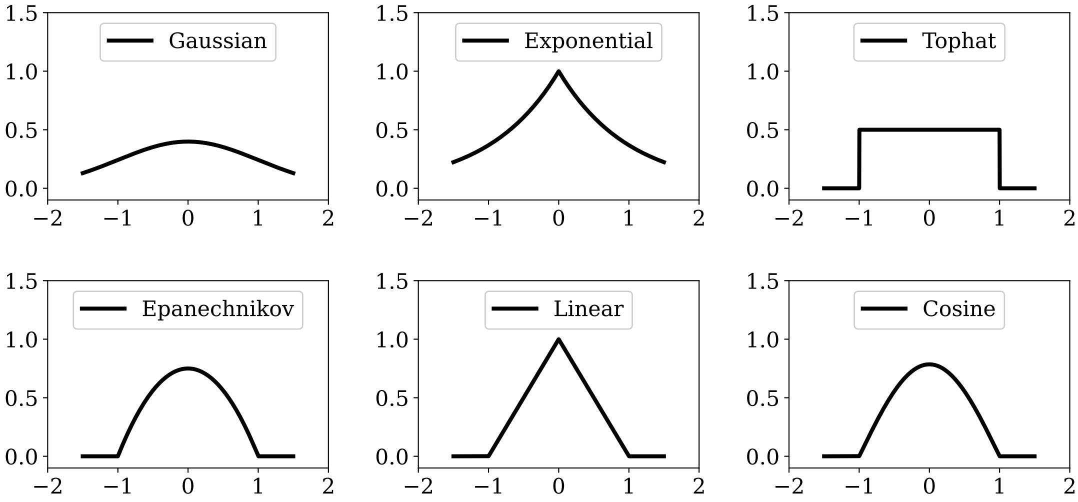


Figure 2.5: Typical kernels used Kernel Density Estimation

$$\hat{p}(x) = \frac{\#\{x_t \text{ in the same bin as } x\}}{Nh} \tag{2.19}$$

If we take x to be always the center of the bin, we can write 2.19 as:

$$\hat{p}(x) = \frac{\#\{x - h/2 < x_t \leq x + h/2\}}{Nh} \tag{2.20}$$

As presented in [94], equation 2.20 can be rewritten as :

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} w\left(\frac{x - x_t}{h}\right) \tag{2.21}$$

where the weight function is:

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

Apparently, the weight w makes 2.21 discontinuous at $x = y_i \pm h/2$; hence it is replaced by a smooth weight function, i.e. the kernel function. This takes us to the kernel density estimator. We further extend the univariate case in 2.21 to d-dimensional case. As such, we can write the Kernel Density Estimator (KDE) as:

$$\hat{p}(\boldsymbol{x}) = \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_t}{h}\right) \tag{2.22}$$

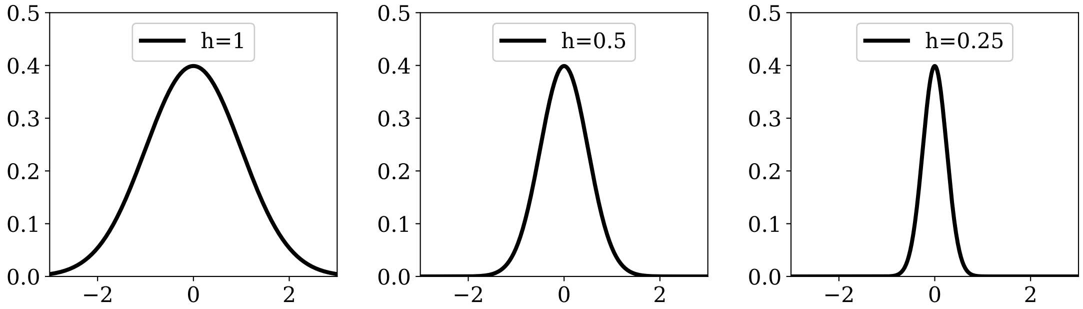


Figure 2.6: Varying the bandwidth while using Gaussian kernel in KDE

where $\mathbf{x}$, $\mathbf{x}_t \in \mathcal{R}^d$, h is a hyperparameter and K is the Kernal function. The most popular kernel is Gaussian. It is given as [95]:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2} \tag{2.23}$$

Other functions such as exponential, rectangular (tophat), Epanechnikov, triangular (linear), and cosine are also available. Figure 2.5 shows these kernels.

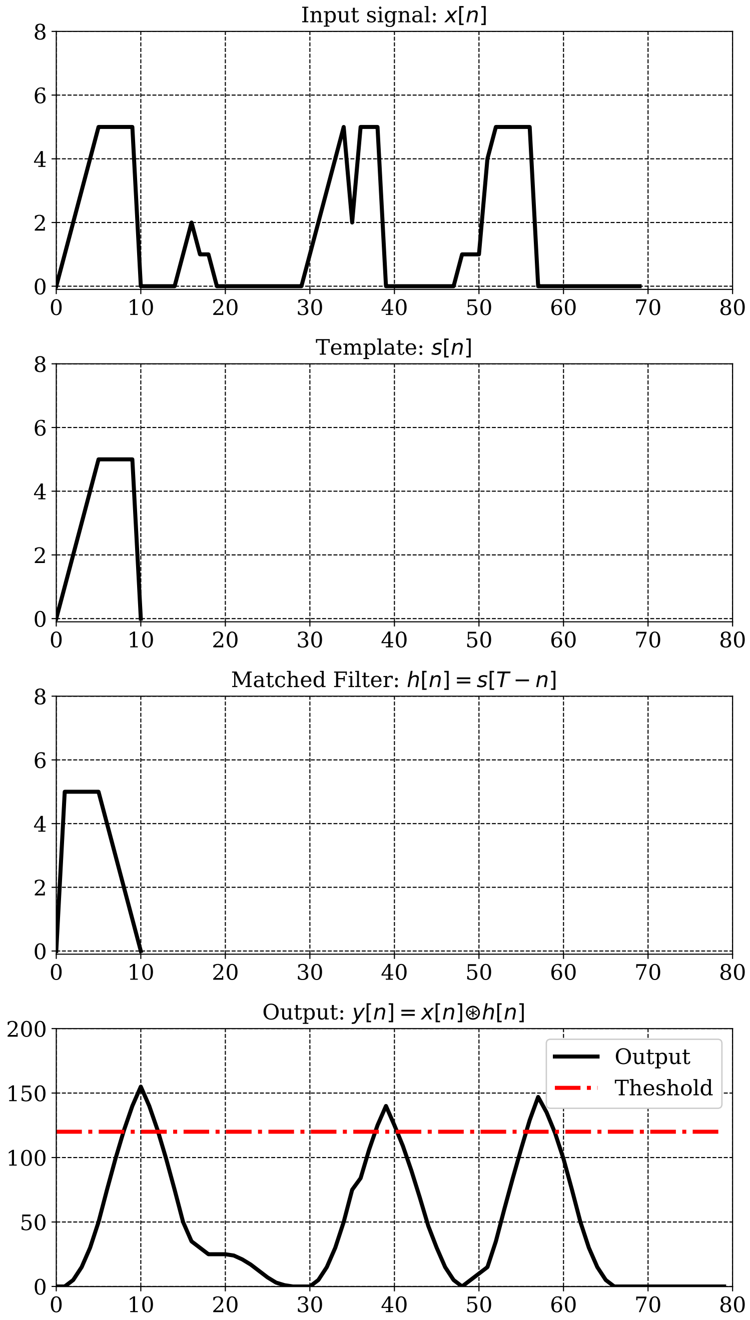
By combining both 2.35 and 2.23, we notice that two important parameters impact KDE. These are the selected kernel function k and the bandwidth h. Figure 2.6 shows the impact of the bandwidth on the Gaussian kernel.

The bandwidth h is occasionally called the smoothing parameter. As h increases, the estimated density function becomes smoother; however, this may risk capturing the variation in the underlying distribution function which we are trying to estimate. This is called oversmoothing. Conversely, smaller h leads to undersmoothing.

2.1.5 Matched-filter

A matched filter is used to extract a known pattern (a.k.a. template) from signals corrupted with noise. It has several applications in communication such as radar. For example, a radar station may transmit a known signal (template) toward an object and decide how far the object is by detecting the reflected signal. Typically, the reflected signal is corrupted with noise, so a matched filter is used to extract the original template and correctly estimate the distance from the object. A matched filter may be also used to classify the body's activities that are recorded by body-worn sensors as discussed in the paper [96]. Matched filters are also used in communication over power lines [97].

Define:



Input signal: $x[n]$

Template: $s[n]$

Matched Filter: $h[n] = s[T - n]$

Output: $y[n] = x[n] \circledast h[n]$

Figure 2.7: Repetitive patterns detected using matched filer

$x[n]$: Discrete signal embedded in noise while n is an independent variable representing the time step.

$s[n]$: Template signal which needs to be extracted from $p[n]$.

T: A scaler representing the time steps occupied by the template and indicated as template's window size.

$h[n]$: The impulse response of the matched filter.

$y[n]$: Filtered output signal.

The matched filter is the optimal linear filter that maximizes the output signal-to-noise ratio. In other words, in order to find the impulse response $h[n]$ of the matched filter, the signal-to-noise ratio needs to be formulated and maximized. Upon solving the optimization problem, the impulse response of the matched filer is found to be [98]:

$$h[n] = s[T - n] \tag{2.24}$$

To find the filtered signal $y[n]$, the noisy signal $x[n]$ is convolved with the filter's impulse response $h[n]$ as per equation 2.25.

$$y[n] = x[n] \circledast h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{2.25}$$

Combining 2.24 and 2.25, we can write:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]s[T-(n-k)] \tag{2.26}$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]s[T-n+k] \tag{2.27}$$

Figure 2.7 clarifies the filtering process. In the figure, the input signal $x[n]$ has three patterns embedded in noise. The patterns occur at around 30 and 47 time steps. The first pattern extending in time from 0 to 10 is used as a template $s[n]$. The matched filter $h[n]$ is constructed from the template. Finally, the input signal is convolved with the matched filter to produce the output $y[n]$. The output has three peaks that occur at the end of each detected pattern. These peaks can be easily detected by applying a threshold (the red dotted line) and the three embedded patterns can be retrieved. In general At time step $n = T$, the output reaches the maximum value of $\sum x[k]s[k]$. In general, the output $y[n]$ of the matched filter is compared with a threshold λ to decide whether a pattern exists (i.e. whenever $y[n] \geq \lambda$) or not.

2.1.6 Cross validation

Neural networks and machine learning algorithms usually involve *parameters* (e.g. weights of neural network) that are optimized during training. In addition, *hyperparameters* are usually encountered and need to be optimized or tuned. For instance, for a neural network, hyperparameters include the number of hidden layers, number of nodes in each layer and others. Tuning hyperparameters may be accomplished using grid search. For instance, a researcher may assign a different number of hidden layers and check the results using the pre-defined quality metric. For the purpose of comparing models and deciding the best hyperparameters, other than the training set shall be used. This is usually called the validation set. The testing set is reserved for the final testing of the model after all its hyperparameters are selected. Instead of isolating a specific set of data samples for validation purposes, k-fold cross-validation is usually adopted.

In k-fold cross-validation, the set of available training examples are divided into k partitions (i.e. folds) of equal sizes. The first fold is held out for testing while training is carried out using the remaining k-1 folds. This is repeated k times with each one of the k partitions being held out for testing. The resulting k scores are averaged. k-fold cross-validation is performed with different values of the hyperparameter to be tuned. Finally, the hyperparameter resulting in the best score is chosen. Clearly, an increased number of folds will result in fewer testing samples. Widely used values of k are 5 and 10.

2.2 Problem definition

It is quite challenging to gather physical measurements for electrical loads in occupied houses. Assume we need to measure the current consumptions of loads in a house for a whole year. This means that we need to keep the measuring instruments (e.g. current transformers) connected to the loads throughout that year. Installing such instruments causes a great deal of inconvenience. Clearly, it is not a matter of accessing the main electrical panel and installing these measuring instruments at the branch circuits; rather each specific load (e.g. dishwasher, furnace, toaster, etc.) needs to be plugged into a standalone local meter. Even if smart plugs are used to sense the currents and wirelessly transmit the data to a central hub, it is still expensive and inconvenient. For instance, a smart plug connected to an electric range may be bulky due to power requirements. To install it, either an electrician needs to be hired to install a recessed receptacle that can accommodate the smart plug, or the house owner needs to live with his range misaligned with the rest of the kitchen cabinets. On the other hand, to ensure variety in data, measuring instruments need to be deployed in several houses of different sizes and in different areas. In practice, it is unlikely that consumers keep eye on the measuring instruments to ensure that they are functional throughout the year. Data may end up being inconsistent and inadequate.

Despite the challenges, there were several decent attempts to gather physical measurements for residential electrical loads [99]. However, these datasets differ in granularity, the number of houses considered, the number of individual loads and duration. Many of the datasets are for a single home only. Others provide data for multiple homes but at very low granularity which will not allow testing non-intrusive load monitoring algorithms. In this chapter, a framework based on state-of-the-art generative adversarial networks is developed to generate synthetic data for residential individual loads.

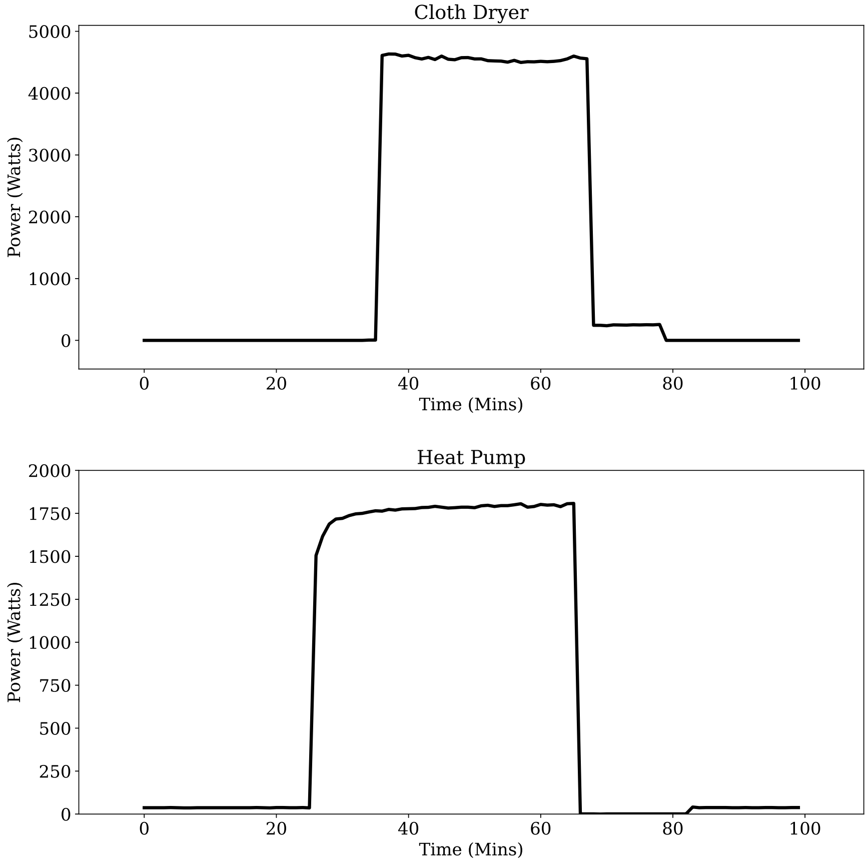


Figure 2.8: Power demand patterns for different residential individual loads

Figure 2.8 shows the power demand of two individual loads in a house when they are switched on. We can see that *patterns* differ in several aspects including cycle duration and power amplitude. However, for a specific load l, its pattern is *not* consistent whenever it is switched on. For example, consider the patterns for the same cloth dryer in figure 2.9. The x-axis shows the time steps in minutes starting from the date stamped at the origin. In the upper figure, the dryer was turned on twice on the 4^{th} of April 2012. Both patterns seem similar. However, the patterns on 5^{th} of April (lower figure) seem quite different. As such, modelling the dryer with a set of circuit elements (e.g. resistors, inductors, etc.) with deterministic values will result in erratic conclusions. Incorporating randomness in the physics-aware model is ineffective as each individual load shall be studied separately to model the stochastic process.

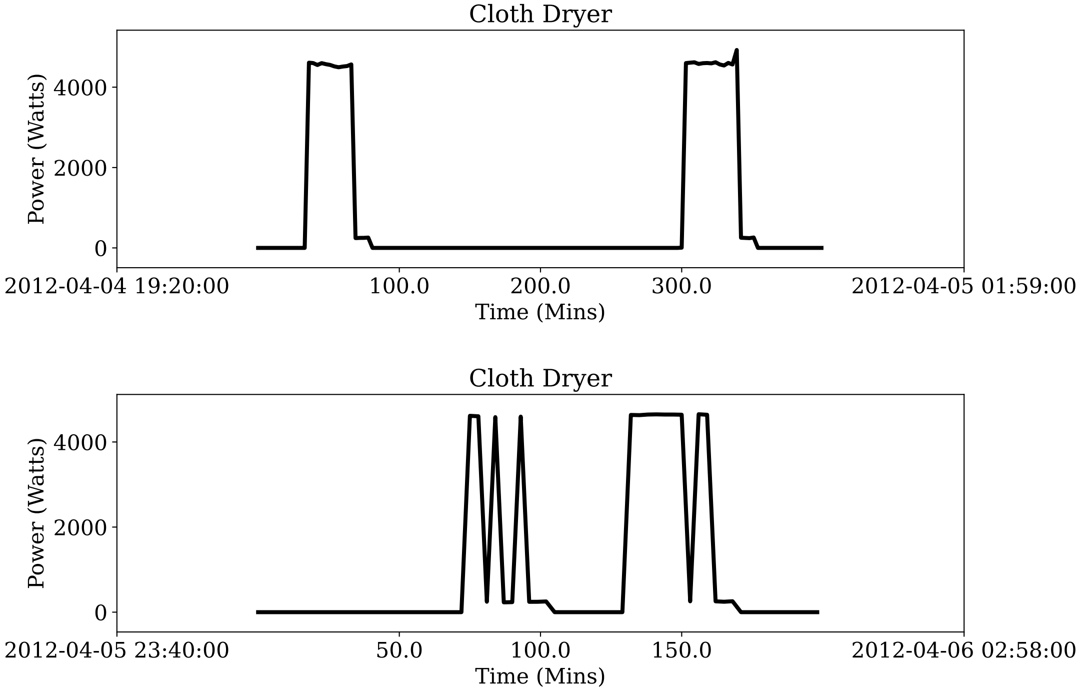


Figure 2.9: Power demand *patterns* for the same individual load at different times

In addition to patterns, each individual load exhibits differences in usage *habits* throughout time. Consider the dryer in figure 2.10. The long bars represent the time when the dryer was switched on. The upper figure shows the usage habits in April while the lower figure shows the habits in May.

In summary, both patterns and usage habits for individual loads incorporate randomness. *Patterns* are decided by the internal characteristics of the equipment and the applied load. For instance, a cloth dryer may not be loaded with the same weight of clothes every time. Usage *habits*, on the other hand, are decided by three functions: user's availability, user's tendency to switch the load and any built-in automatic control [15]. A furnace is a good example where it is usually controlled by a thermostat, however, occupants can interfere any time to switch it on or off. Accordingly, we can think about patterns and habits as two separate random variables. In this chapter, a framework to learn the underlying distributions of these random variables is proposed. Once learnt, simulation is conducted by sampling from the learnt distributions.

Consider residential individual load $l \in \{1, \ldots, L\}$ as any electrical load in a house of L number of load whereas l can be switched on or off by an occupant, automatic control or both. In practice, l may be a combination of loads, e.g. a single switch that controls several

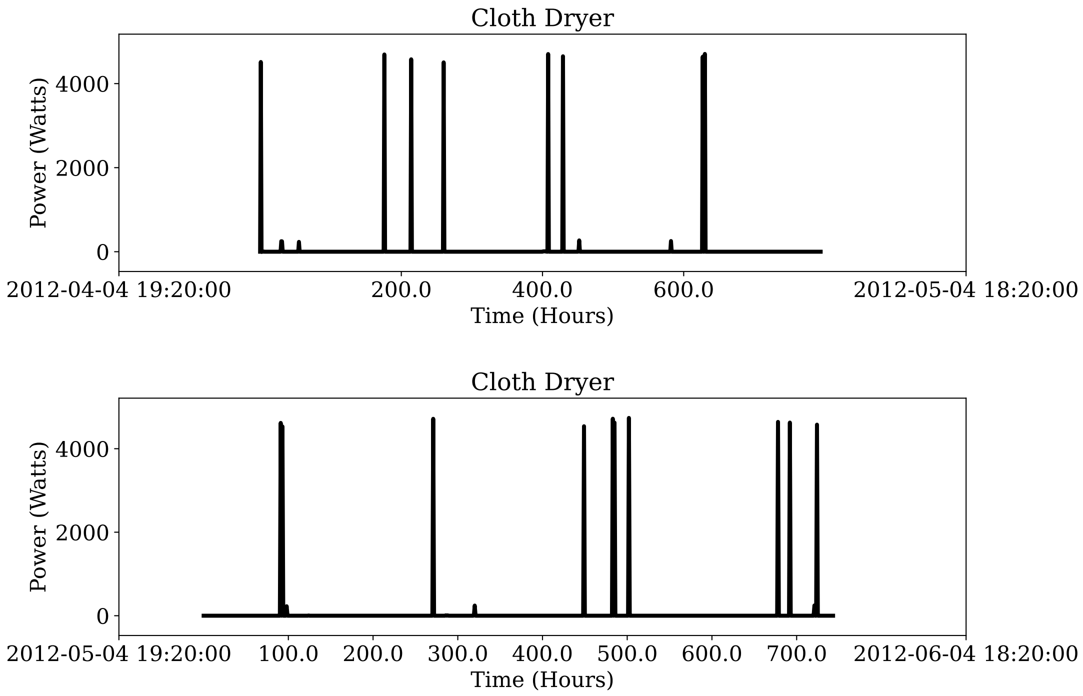


Figure 2.10: Power demand *habits* for the same individual load at different times

light fixtures. Define discrete random power *sequence* for any load l over a period T as $\mathbf{o}^l = \{o^l_1 \dots o^l_t \dots o^l_T\}$, we need to:

- Extract from $\mathbf{o}^l$ the set of *pattern* examples $\{\mathbf{x}^l_i\}^M_{i=1}$ where M is the number of extracted examples and $\mathbf{x}^l_i \in \mathcal{R}^d$ such that d is the dimension of the extracted pattern.

- Find the underlying *pattern* distribution for load l, i.e. $p(\mathbf{x}^l|l)$.

- Extract from $\mathbf{o}^l$ the set of *habit* examples $\{\mathbf{y}^l_i\}^M_{i=1}$ where $\mathbf{y}^l_i \in \mathcal{R}^3$ such that the extracted dimension of a habit example represent month of year, day of week and hour of day.

- Find the underlying *habit* distribution for load l, i.e. $p(\mathbf{y}^l|l)$.

- Generate synthetic pattern and habits from the learnt distributions above, i.e. simulate load l.

As we will see in the following sections, we use generative models to construct both pattern and habit distributions.

2.3 Proposed framework

The proposed framework in this chapter is based on the author's work published in reference [4]. As explained in 2.2, our main goal is to construct a framework that can estimate both $p(\mathbf{x}^l|l)$ and $p(\mathbf{y}^l|l)$. The proposal is based on the author's work published in reference [75].

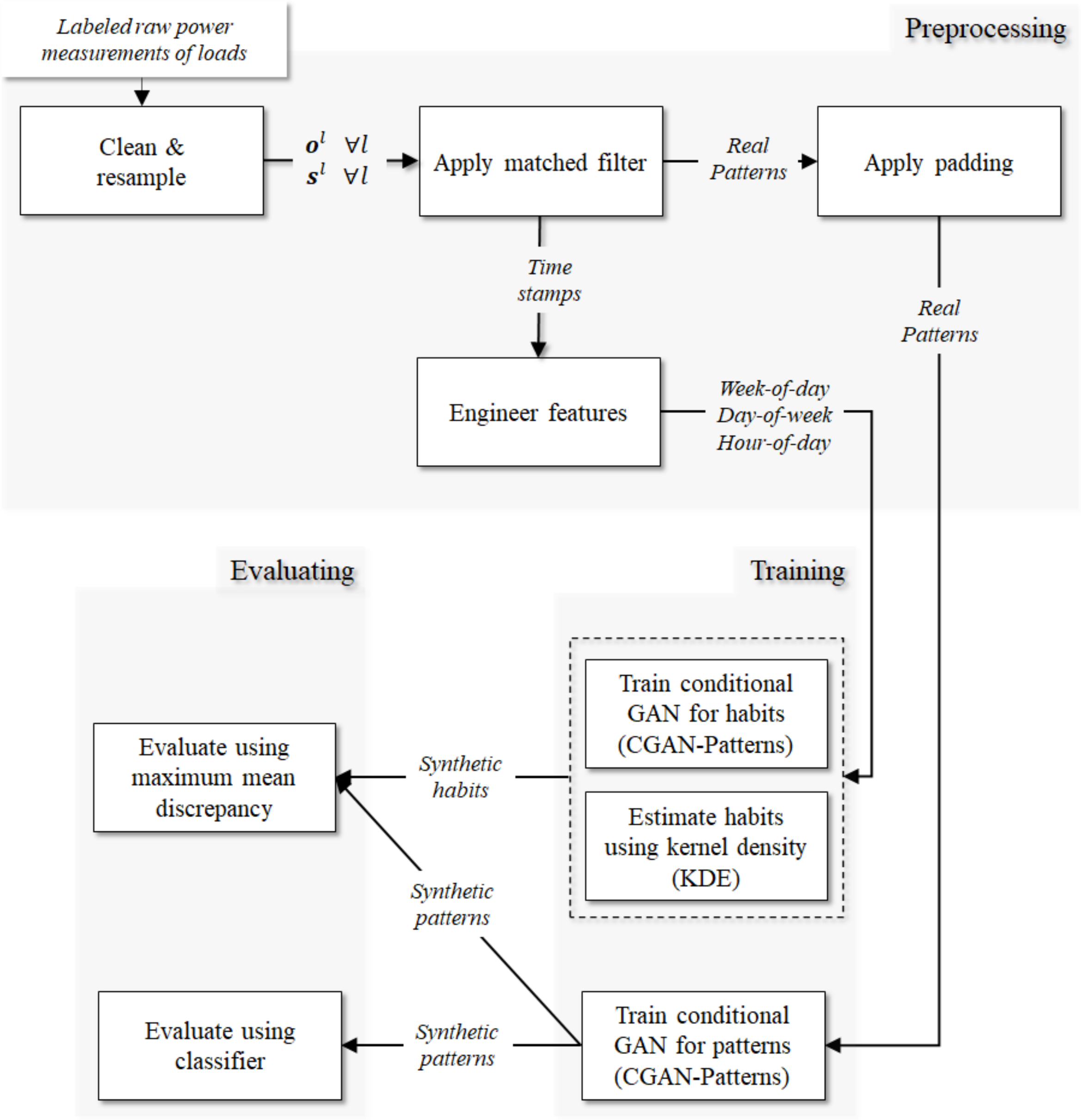

Labeled raw power measurements of loads
Preprocessing
Clean & resample
$\mathbf{o}^l \ \forall l$
$\mathbf{s}^l \ \forall l$
Apply matched filter
Real Patterns
Apply padding
Time stamps
Real Patterns
Engineer features
Week-of-day
Day-of-week
Hour-of-day
Evaluating
Training
Train conditional GAN for habits (CGAN-Patterns)
Estimate habits using kernel density (KDE)
Evaluate using maximum mean discrepancy
Synthetic habits
Synthetic patterns
Evaluate using classifier
Synthetic patterns
Train conditional GAN for patterns (CGAN-Patterns)

Figure 2.11: Proposed framework for learning patterns and habits of residential individual loads.

Figure 2.11 shows the main blocks in our proposal. Estimating distributions for patterns and habits involves three stages: preprocessing, training and evaluation. The raw data used for training is typically provided as a discrete sequence of real power measurements for each load $l \in \{1, 2, \ldots, L\}$ where L is the number of loads processed by the proposed model. First, the sequences shall be preprocessed to clean time inconsistency and remove outliers. Time inconsistency, if any, is corrected using interpolation. Outliers are replaced by the mean of adjacent sample points. Once cleaned, the matched filter is used to extract all available patterns and the associated timestamps when patterns start. Templates that the matched filter uses to extract the patterns are either manually extracted from the input power sequence or provided by the manufacturer of that electrical load or appliance.

The proposed model in figure (2.11) allows power measurements to be received from different datasets and for different loads. Datasets may have different sampling rates. For the model to process data from different datasets, a unified granularity interval T_g (in seconds) shall be defined for all datasets used in training. If the sampling interval of a dataset T_s (in seconds) is less than T_g, the corresponding power measurements $\mathbf{o}^l$ shall be down-sampled. The load template shall be down-sampled as well and the adjusted load cycle window (i.e. number of sample points in the template) W^l is calculated.

Loads have different cycle windows W^l. For instance, a dishwasher may have a maximum operational cycle window of 2 hours while a toaster may not exceed 5 minutes. A master window W for all loads shall be calculated. This is typically taken as the maximum cycle window among all loads. Algorithm 1 summarizes the calculation of W^l and W. Note that these are unitless scalar quantities as they represent counts of samples. W is important for designing the conditional generative adversarial network of patterns while W^l is used in the matched filter.

Algorithm 1 Computation of W and W^l

```
Input:
    Unified granularity interval in seconds ($T_g$)
    Dataset sampling interval in seconds ($T_s$)
    Cleaned load templates $\mathbf{s}^l$ for all loads $l \in 1 \ldots L$
Output:
    Adjusted load cycle window ($W^l$) $\forall l = 1 \ldots L$
    Master window ($W$)
for $l$=1 to $L$ do
    $W^l \leftarrow \text{ceiling}(T_s/T_g \times \text{length}(\mathbf{s}^l))$
end for
$W \leftarrow \max\{W^l\}\ \forall l = \{1, 2, \ldots, L\}$
```

The matched filter in figure 2.11 receives two inputs: the cleaned and re-sampled power sequence $\mathbf{o}^l$ for each load and its respective template $\mathbf{s}^l$. The matched filter is constructed as per equation 2.27 and convolved with the input signal $\mathbf{o}^l$. Substituting the obtained W^l in algorithm 1, we can re-write equation 2.27 for each load l as:

$$\mathbf{y}^l[n] = \sum_{k=-\infty}^{\infty} \mathbf{o}^l[k] s[W^l - n + k] \tag{2.28}$$

The indices of the peaks are detected by applying a threshold to $\mathbf{y}^l$. The indices represent timestamps when a pattern in $\mathbf{o}^l$ is detected. In specific, the SNR is maximized when the *end* of each pattern is detected. Hence, W^l number of samples that occur before the detected index is extracted. If $W^l < W$, the extracted pattern is padded with zeros to have a dimension of W. The extracted power values are saved as the i^{th} example of pattern which is concatenated with the load's label, l. This is repeated for all detected patterns to end up with the set of training examples $\{\mathbf{x}^l\}_{i=1}^{M}$ $\forall l$ such that $\mathbf{x}_i^l \in \mathcal{R}^{W+1}$. Figure 2.12 shows a typical example of the dataset to be used for training *patterns* using conditional GAN.

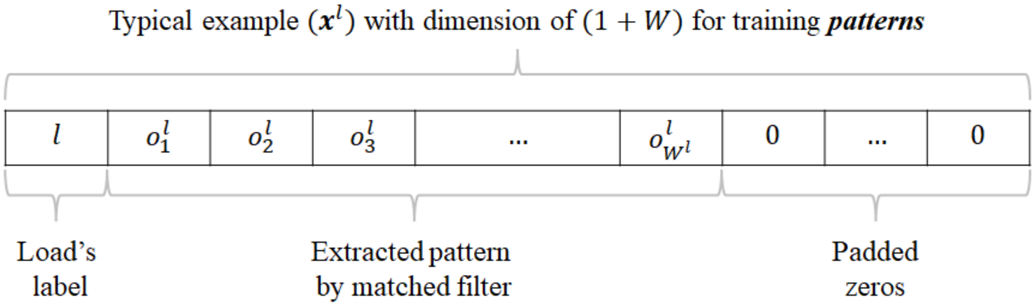

Typical example ($\boldsymbol{x}^l$) with dimension of $(1 + W)$ for training ***patterns***
l
o_1^l
o_2^l
o_3^l
...
$o_{W^l}^l$
0
...
0
Load's label
Extracted pattern by matched filter
Padded zeros

Figure 2.12: Typical example used for training *patterns*

On the other hand, the time stamp when the patterns are detected is passed on to the feature engineering module shown in figure 2.11. The module extract from the timestamps the three features mentioned in 2.2, i.e. week-of-year, day-of-week and hour-of-day. The engineered features are concatenated with the load's label l. This is repeated for all detected time stamps to end up with the set of training examples $\{\mathbf{y}^l\}_{i=1}^{M}$ $\forall l$ such that $\mathbf{y}_i^l \in \mathcal{R}^4$. Figure 2.13 shows a typical example of the dataset to be used for training the *habits*.

The above assumes that loads are encoded using integer numbers. This is usually known as label encoding and it is used throughout our discussion for simplicity. However, using integer numbers implies that loads are ordinal. i.e. load encoded 1 is less than load encoded 2 and so forth. However, this is not true, so a better way to encode loads is to use *one-hot-encoding*. In such a case, each load is represented by a unit vector. For instance, if we have 3 loads, the first load will be encoded as [0 0 1], the second as [0 1 0] and so forth. Hence, one-hot-encoding will increase the dimension of the datasets shown in figure 2.12 and 2.13 by the number of loads less 1. The one is subtracted since the one-hot-encoding will replace integer encoding. Accordingly, the dataset used for patterns can be expressed as $\{\mathbf{x}^l\}_{i=1}^{M}$ $\forall l$ such that $\mathbf{x}_i^l \in \mathcal{R}^{W+L}$

where L is the number of loads. Similarly, the dataset used for habits can be expressed as $\{\mathbf{y}^l\}_{i=1}^M\ \forall l$ such that $\mathbf{y}_i^l \in \mathcal{R}^{3+L}$.

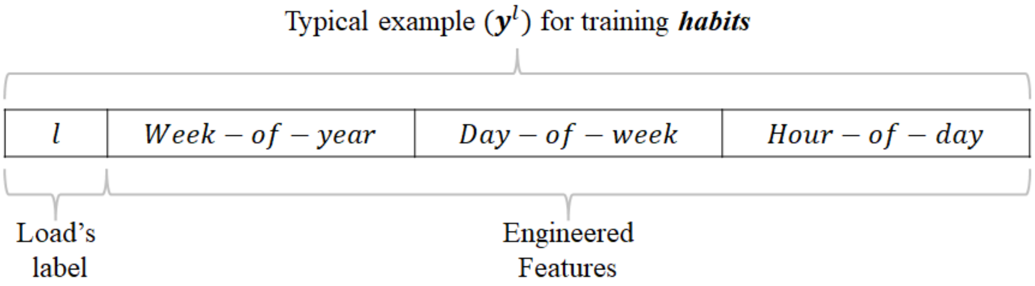


Figure 2.13: Typical example used for training *habits*

In practice, the datasets $\{\mathbf{x}^l\}_{i=1}^M\ \forall l$ and $\{\mathbf{y}^l\}_{i=1}^M\ \forall l$ are usually split into training and testing. Further, the batch size is defined so training examples can be ingested into generative adversarial networks in batches. For clarity, in subsequent discussions, we will refer to a batch of patterns as $\mathbf{x}$ and to a batch of habits as $\mathbf{y}$. The bold symbol indicates that each example in the batch is multi-dimensional. Recall that loads' labels are already concatenated in $\mathbf{x}$ and $\mathbf{y}$ as explained in figures 2.12 and 2.13. Further, $\mathbf{z}$ refers hereafter to the batch compromising noise vectors concatenated with labels matching these concatenated in the real examples. In other words and without loss of generality, although our model involves conditional generative adversarial networks as demonstrated by figure 2.4, we will treat these as generative adversarial networks as per figure 2.3. The conditional GAN for patterns is abbreviated CGAN-Patterns, while the conditional GAN for habits is abbreviated CGAN-Habits.

In figure 2.11, CGAN-Patterns receives $\mathbf{x}$. Initially, the loss defined by 2.7 is used. However, during experimental trials, we encountered instability using the vanilla loss, so we introduced *inverted vanilla* loss that resulted in substantial improvement in the performance. The introduced loss is given in 2.29 below.

$$\min_G \max_D L(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log(1-D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log D(G(\boldsymbol{z}))] \tag{2.29}$$

Following the steps of [29] and as discussed in 2.1.2, we write our *inverted vanilla* optimal discriminator (for fixed G) as:

$$D_G^*(\boldsymbol{x}) = \frac{p_g(\boldsymbol{x})}{p_g(\boldsymbol{x}) + p_{data}(\boldsymbol{x})} \tag{2.30}$$

The proof is straight forward and follows the same steps of [29] with 2.29 used instead of 2.7. We may simplify 2.29 by showing that we are sampling from p_g instead of sampling from

p_z. Accordingly, we write our objective as:

$$\min_G \max_D L(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log(1 - D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{g}}}[\log D(x)] \tag{2.31}$$

Substituting the optimal discriminator in 2.31, we obtain the *virtual training criterion*:

$$C(G) = \max_D L(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \tag{2.32}$$

which is identical to the equation derived by [29]. Similarly, our training criterion will similarly reduce to:

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{2.33}$$

where JSD is the Jensen-Shannon divergence between the real and model distributions. Hence, our *inverted vanilla* GAN is related to JSD in exactly the same way the *vanilla* GAN is related to JSD. ▮

While the parameters (i.e. weights and biases) of CGAN-Patterns are learnt during the training process, hyperparameters need to be optimized (i.e. tuned) as well. Below is the list of the hyperparameters for CGAN-Patterns.

1. Batch size.
2. Number of epochs.
3. Number of iterations within an epoch for the discriminator.
4. Number of iterations within an epoch for the generator.
5. Weight initializers for both generator and discriminator.
6. Number of hidden layers for the discriminator.
7. Number of nodes in each layer for the discriminator.
8. Type of activation functions used in each layer for the discriminator.
9. Applied node drop-out percentage for the discriminator.
10. Input noise type for generator.
11. Input noise dimension for generator.
12. Number of hidden layers for generator.
13. Number of nodes in each layer for generator.
14. Type of activation functions used in each layer for generator.
15. Applied node drop-out percentage for generator.
16. Type of loss for both generator and discriminator.

For learning habits distribution, two methods are used for further comparison. First, CGAN-

Habits is constructed using the inverted vanilla loss as shown in equation 2.34 below.

$$\min_{G} \max_{D} L(D, G) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log(1 - D(y))] + \mathbb{E}_{z \sim p_z(z)}[\log D(G(z))] \tag{2.34}$$

where $\mathbf{y} \in \mathcal{R}^4$ with an individual example as shown in figure 2.13. Recall that $\mathbf{z}$ represents noise vectors concatenated with labels paired with labels that are concatenated with the engineered features. Clearly, likewise CGAN-Patterns, in CGAN-Habits only one conditional GAN is used for all loads $l \in \{1, 2, \ldots, L\}$ where L is the number of loads. The hyperparameters of CGAN-Habits are similar to those listed previously for CGAN-Patterns.

Secondly, the engineered features are passed to kernel density estimators as given in 2.35. Unlike CGAN-Habits, a separate kernel density estimator (KDE) has to be established for each load. Let $\{\mathbf{v}^l\}_{i=1}^{M}$ such that $\mathbf{v}^l \in \mathcal{R}^3$ be the set of engineered features extracted from $\mathbf{o}^l$ for for load l. Then for each load $l = \{1, 2, \ldots, L\}$, a KDE shall be constructed as given in the equation below.

$$\hat{p}(\boldsymbol{x}) = \frac{1}{Mh^3} \sum_{i=1}^{M} K\left(\frac{\boldsymbol{x} - \boldsymbol{v}_i}{h}\right) \tag{2.35}$$

Once constructed for each load, any number of samples may be drawn from the distribution to reflect learnt usage habits for load l. Both kernel K and bandwidth h are treated as hyperparameters and shall be optimized.

Ultimately, and once models are trained, the framework is used to generate patterns and habits based on the given load. For instance, a user may be interested in simulating a cloth dryer. The name of the load and the duration of the simulation are ingested into the framework. The user, in return, receives a specified number of patterns and habits. Figure 2.14, shows the trained framework in the simulation mode where the blue ovals represent the output of the proposed framework. The user may choose to randomly combine habits and patterns if needed. Note that, for habits, the best of the two trained models (i.e. CGAN-Habits and KDEs) is used.

2.4 Experimental studies

2.4.1 Preprocessing

The experiment is conducted using four loads that are selected from the dataset described in appendix A. The loads include cloth dryer (CDE), dishwasher (DWE), fridge (FGE) and heat pump (HPE). Figure 2.17 shows the patterns of various loads before any preprocessing. The signals differ in both shapes and amplitudes.

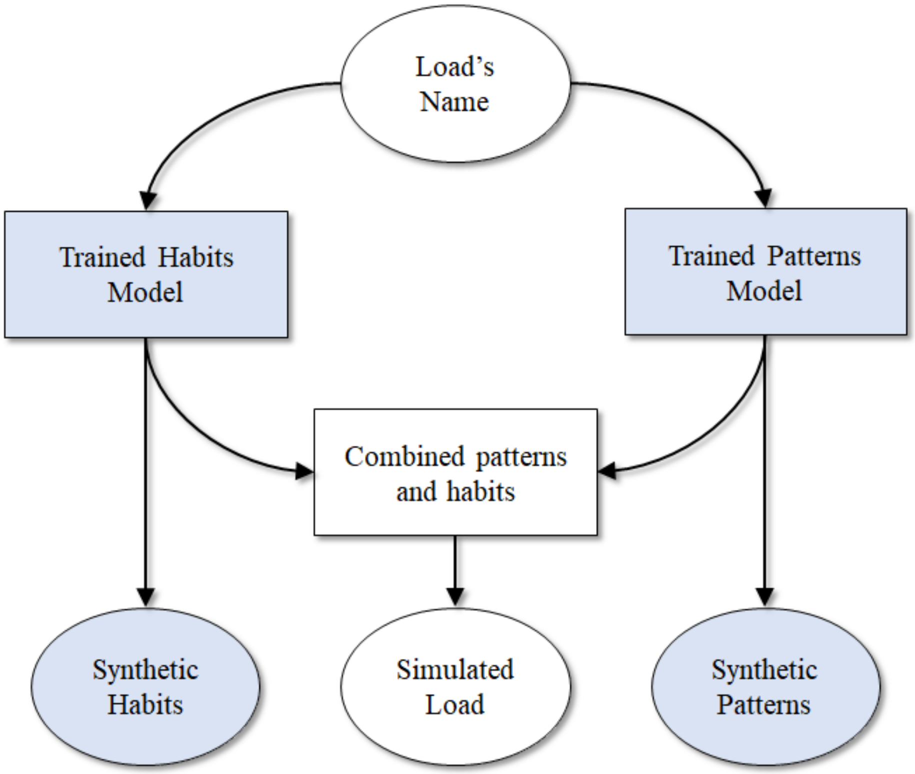


Figure 2.14: Trained frameowrk for simulating residential loads

The data is provided in granularity of both one minute (AMPd dataset) and one second (RAE dataste). Reducing granularity by down-sampling the training patterns directly impacts the dimension of the input to the conditional GAN (CGAN-Patterns) which, in turn, leads to less computational complexity. However, further reduction of granularity may lead to losing important features of the patterns. Figure 2.16 shows how some features are lost when granularity is reduced to 5 minutes for a cloth dryer. Reduced granularity causes the learnt distribution for synthetic patterns to deviate from the distribution of real patterns. We found a three-minute granularity to be a good compromise between dimensionality and maximum mean discrepancy, which is the quality metric used to measure the distance between the learnt distribution and the underlying real distribution.

The usage habits of various loads differ widely. Figure 2.17shows that the usage frequency of a cloth dryer is far less than that of a heat pump. Clearly, the fridge has the highest frequency, which is dictated by the automatic start-ups of the compressor.

As per figure 2.11, the dataset is cleaned by the basic removal of outliers. Data for all four loads are imported from AMPd dataset. Further data for CDE and HPE is imported from RAE dataset. The unified granularity T_g indicated in algorithm 1) is defined as 180 seconds

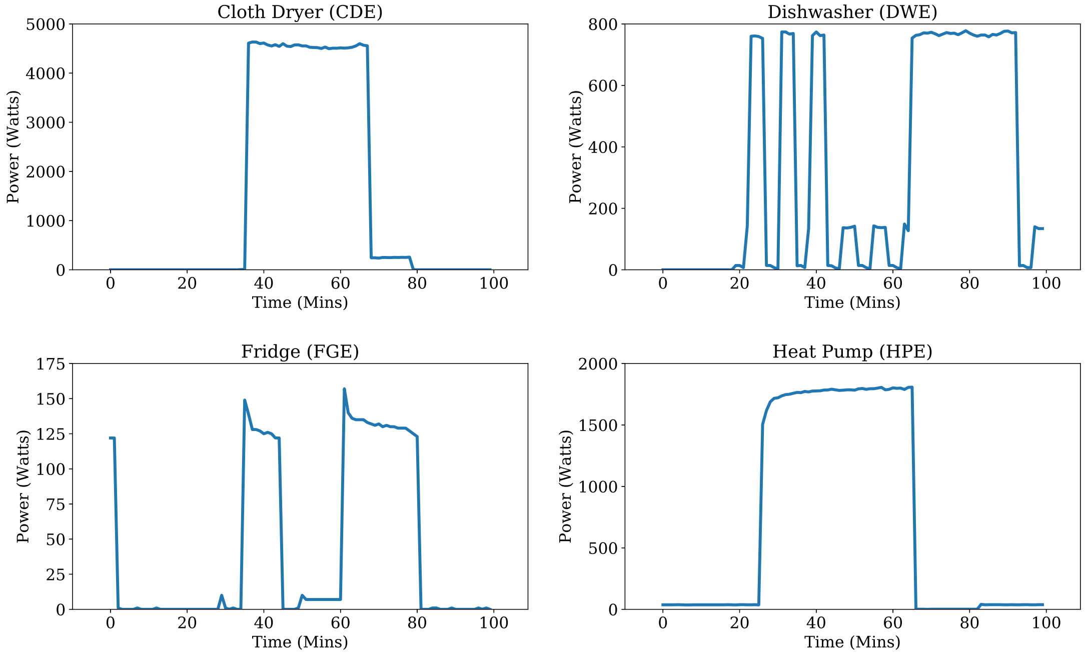


Figure 2.15: Raw patterns for different loads

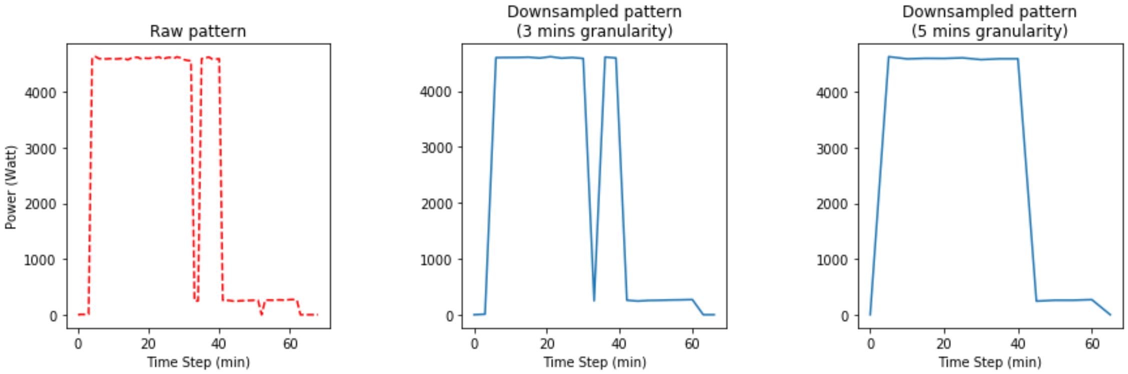


Figure 2.16: Impact of reducing granularity (T_g) on the training samples for cloth dryer

(3 minutes). The resulting lengths of templates (i.e. *length*($\mathbf{s}^l$) for CDE, DWE, FGE, HPE are 23,42,29 and 16 respectively. This corresponds to lengths of 69, 126, 87 and 48 minutes receptively. Clearly, the window size W is the largest, i.e. 42. The matched filter is then applied to each load to extract all the patterns and their corresponding time stamps. The timestamps are further processes and three features are extracted, i.e. week-of-year, day-of-week and hour-of-day. The datasets for both CGAN-Patterns and CGAN-Habits are finally arranged in the forms of figures 2.12 and 2.13.

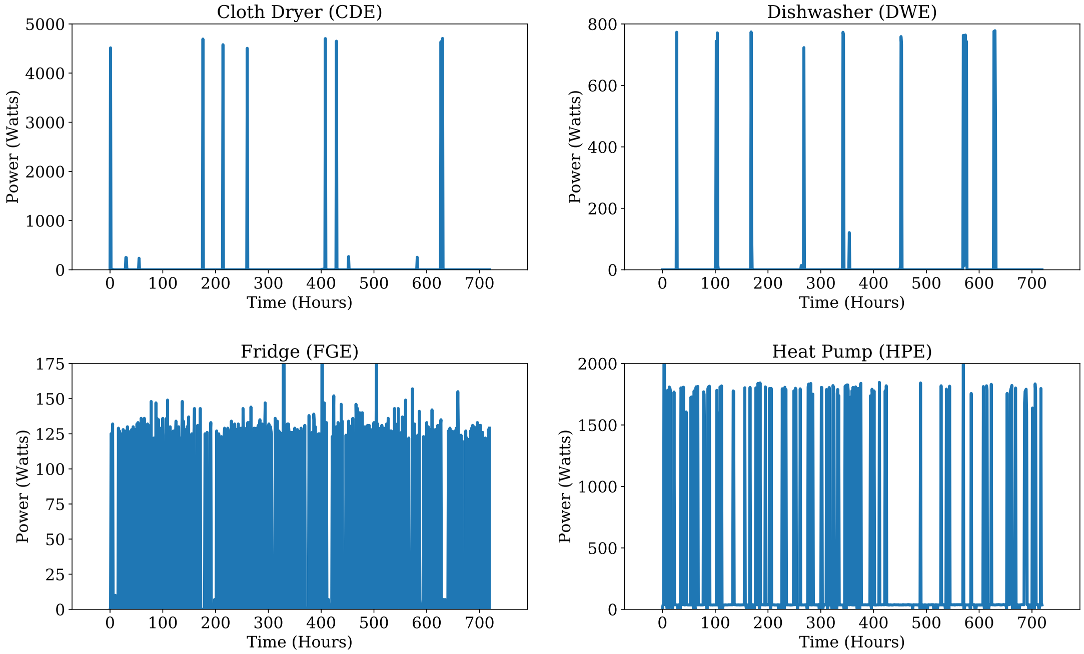


Figure 2.17: Raw usage habits of different loads for around 11 days.

2.4.2 Generating patterns

The processed dataset for patterns is split 80/20 for training and testing datasets. The training dataset is used to train CGAN-Patterns. The architecture of CGAN-Patterns is given in table 2.1. Since we have four loads for our experiment, their labels are encoded as unity vectors of dimension 4. For the generator, the encoded label is concatenated as a condition to a normally distributed noise vector of dimension 100, ending with a generator's input nodes of 104. Leaky ReLU activation functions are used in the hidden layers while the output layer is Tanh. The output of the generator has a dimension of 42 standing for a synthetic pattern corresponding to the encoded input condition. On the other hand, the encoded label of dimension 4 is concatenated to either the real or synthetic patterns, ending up with 46 input nodes. The discriminator has a binary output representing the probability of an input being real or fake. A sigmoid activation function is used at the discriminator's output which will limit the output between 0 and 1.

Three types of losses are tested: the vanilla loss (equation 2.7), the inverted vanilla (equation 2.29) and the Wasserstein loss proposed by reference [91]. When using the vanilla loss, the model did not converge. Figure 2.22 shows the evolution of the loss functions for both generator and discriminator during training. Clearly, the losses reach equilibrium in the min-max game.

Master Window W	42
Conditions (loads)	4 (One-hot Encoded)
Generator	5 Layers
Nodes/layer:	L1: 104, L2: 100, L3: 150, L4: 100, L5: 42
Activation/layer	Leaky ReLU, Leaky ReLU, Leaky ReLU, Tanh
Discriminator	5 Layers
Nodes/layer	L1: 46, L2: 100, L3: 150, L4: 100, L5: 1
Activation/layer	Leaky ReLU, Leaky ReLU, Leaky ReLU, Sigm

Table 2.1: Architecture of CGAN-Patterns [4]

.

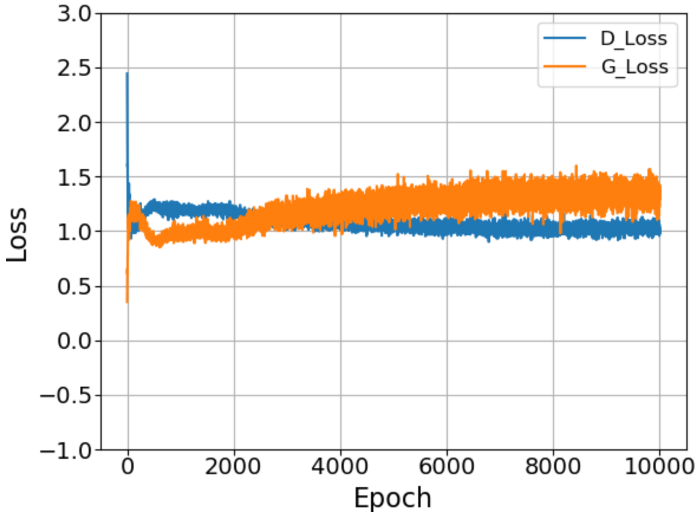


Figure 2.18: Evolution of the inverted loss during training.

Figure 2.22 shows random samples of the synthetic patterns generated by the CGAN-Patterns with the inverted loss function. The patterns look similar to the real patterns provided in the testing dataset. No over-fitting is experienced as the synthetic patterns are similar but not identical to the training dataset. This indicates that the CGAN-Patterns has successfully learned the distribution of the real data and managed to generate patterns that are drawn from the learnt distribution.

When Wasserstein loss was used, CGAN-Patterns did not learn the underlying distribution properly. Figure 2.20 clearly shows the difference between the inverted vanilla GAN and the Wasserstein GAN vs. samples drawn from the real distribution. Both vanilla and Wasserstein were excluded from further testing and the inverted vanilla was used for optimizing CGAN-Patterns.

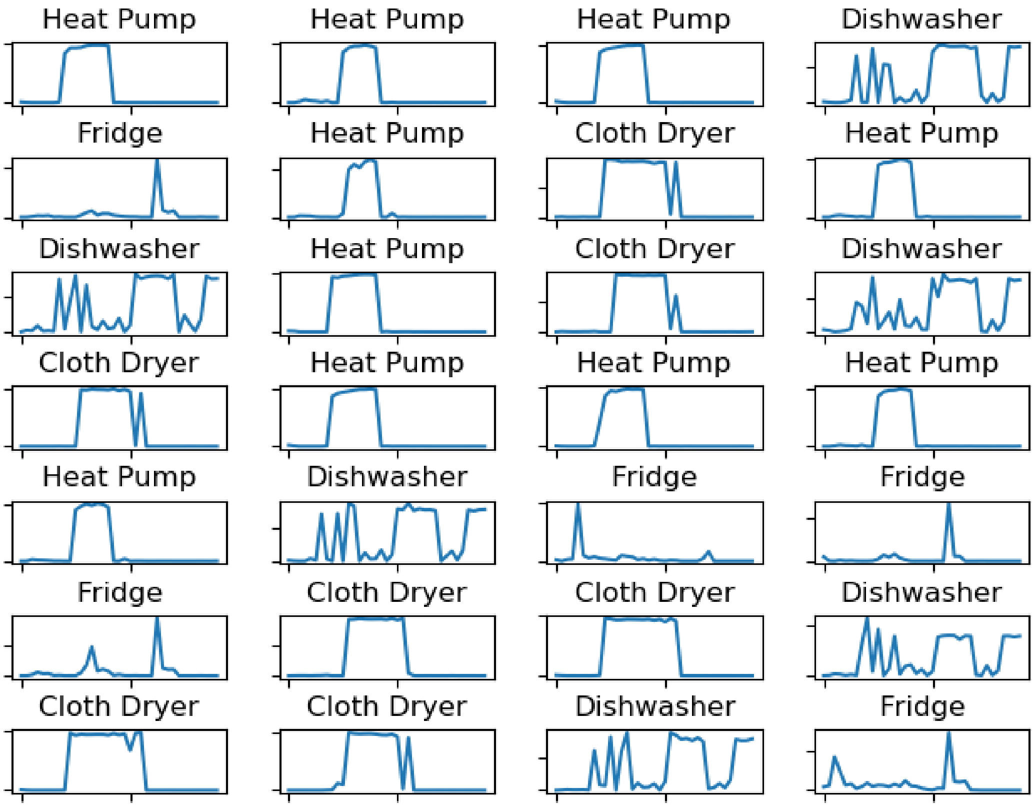
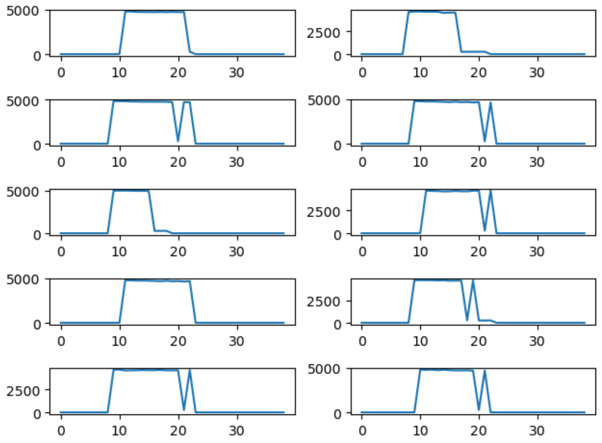
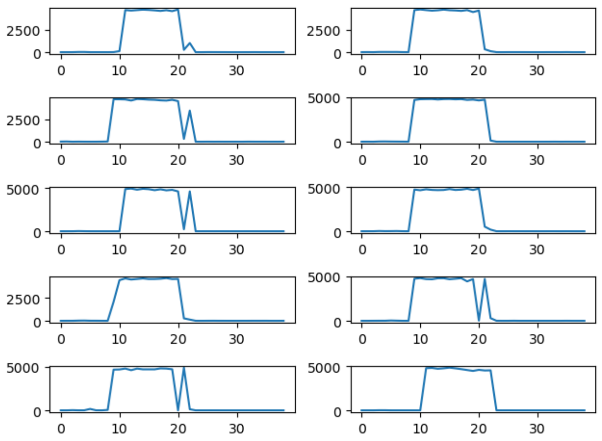


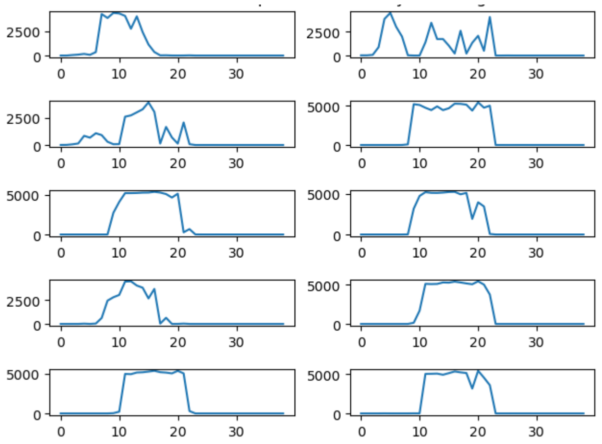
Figure 2.19: Synthetic patterns from proposed GAN.



(a) Real Patterns



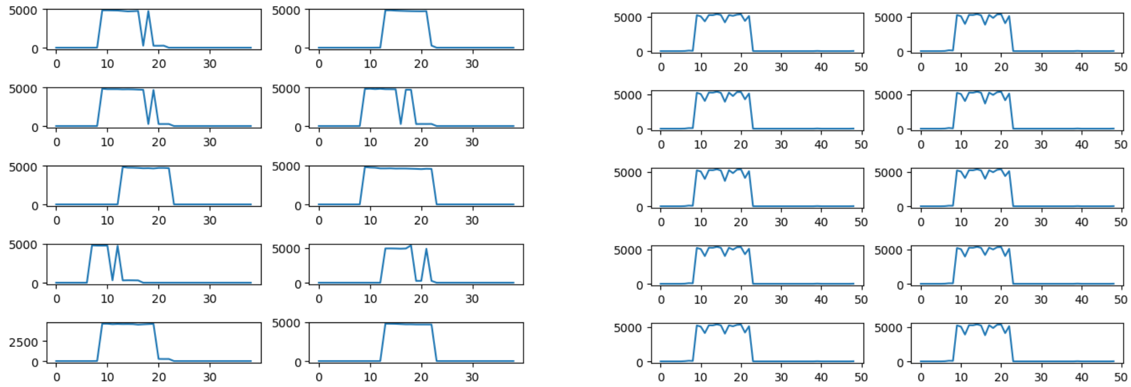
(b) Synthetic Patterns using inverted vanilla loss



(c) Synthetic Patterns using Wasserstein loss

Figure 2.20: Comparison of real versus synthetic patterns.

While training CGAN-Patterns, we faced a common phenomenon called *mode collapse*. This is when the generator replicates the same output every time as depicted in figure 2.21. The problem was eliminated by optimizing the number of iterations that the generator needs to be trained before the discriminator gets updated and vice versa.



(a) Random real patterns. (b) Collapsed synthetic patterns.

Figure 2.21: Mode collapse in CGAN-Patterns for Cloth Dryer [4].

For qualitative evaluation of the results, the maximum mean discrepancy (MMD) is plotted throughout the training process. Clearly, MMD for each load decreases reflecting successful learning of the underlying distribution from which the real samples were drawn.

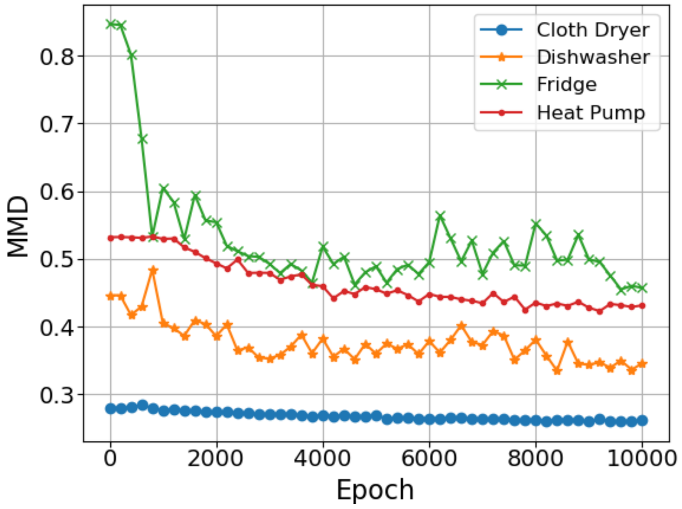


Figure 2.22: Training and synthesis of the GAN system.

For further objective evaluation, a classifier neural network is constructed to resemble a person's eye in judging whether a generated pattern is good or bad. We call this classifier *Evaluator* and its architecture is given in table 2.2. In this case, the encoded labels are used as output and not input. The evaluator has an input of 42 nodes corresponding to the input pattern for any load. The evaluator is trained to identify the load associated with the input pattern. The evaluator is trained using categorical cross-entropy function given in 2.6. The evaluator is trained using 80% of the real patterns dataset. Once trained the synthetic patterns are passed to the evaluator to check if the evaluator fails to recognize the label of any of the ingested synthetic patterns.

Cost Function	Categorical Cross Entropy
Features	42
Classes	4 (One-Hot Encoded)
Layers	6
Nodes/layer	L1: 42, L2: 8, L3: 10, L4: 10, L5: 10, L6: 4
Activation/layer	ReLU, ReLU, ReLU, ReLU, Softmax

Table 2.2: Evaluator net architecture [4].

Figure 2.23 shows that the evaluator identified the correct labels for almost all the ingested synthetic examples generated by CGAN-Patterns.

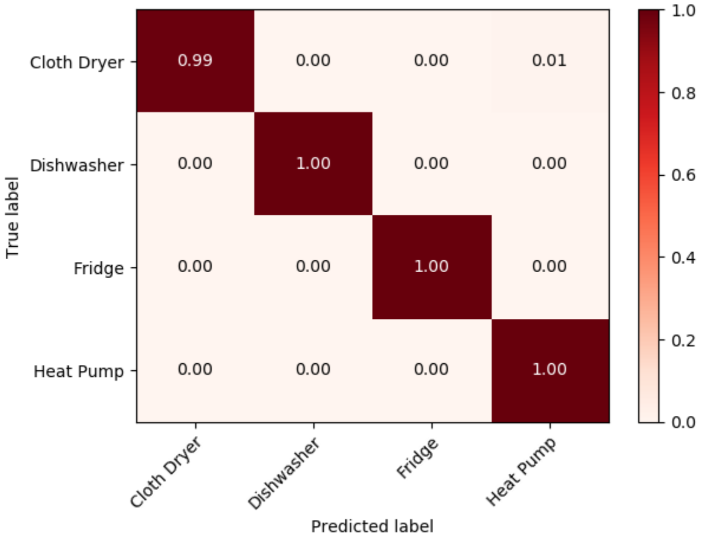


Figure 2.23: Normalized confusion matrix for synthetic data.

2.4.3 Generating habits

The prepared dataset of engineered features is used to develop models using two generative techniques, namely Kernel Density Estimator (KDE) and CGAN-Habits.

KDE is established for each load using equation 2.35. In training phase, the kernel K and the bandwidth h are selected using 10-fold cross validation. Table 2.3 shows the optimized parameters for all four loads.

Load	Cloth Dryer	Dishwasher	Fridge	Heat Pump
K	Gaussian	Gaussian	Exponential	Gaussian
h	0.177827941	0.177827941	0.177827941	0.177827941

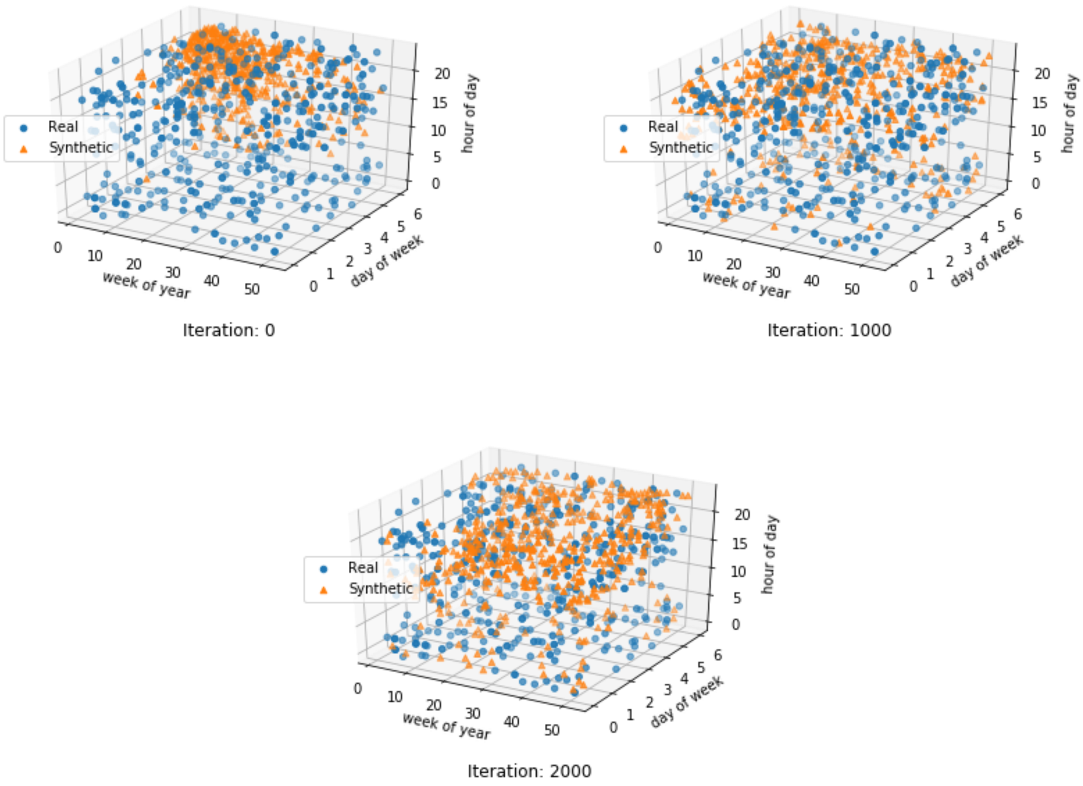
Table 2.3: KDE optimized parameters using 10-fold cross-validation [4].

The same engineered dataset is split 80/20 to train the CGAN-Habits. Table 2.4 shows the architecture of the network. The input of the discriminator is the three engineered features concatenated with the one-hot-encoded label as a condition. This results in 7 input nodes with one output node representing the probability of the input data being real or fake. The input of the generator is a noise vector of dimension 50 concatenated with the condition resulting in 54 input nodes. The output of the generator is the synthetic habits, i.e. synthetic week-of-year, day-of-week and hour-of-day.

Features	3
Conditions (loads)	4
Discriminator	5 Layers
Nodes/Layer	7, 100, 200, 2, 1
Activation/Layer	Leaky ReLU, Leaky ReLU, Leaky ReLU, Sigm
Generator	5 Layers
Nodes/Layer	54, 120, 240, 120, 3
Activation/Layer	Leaky ReLU, Leaky ReLU, Leaky ReLU, Tanh

Table 2.4: Architecture of CGAN-Habits [4].

While training, snapshots of the distribution of the three features are taken. In figure 2.24, the real training samples for all loads are plotted in blue (circles) while the synthetic samples are in orange (triangles). The real samples maintain the same distribution throughout the training process. Initially, the generator in CGAN-Habits constructs synthetic samples from the input random noise. Since the network's parameters are not yet optimized, the distribution generated samples (iteration 0) is completely different from that of the real samples. After 1000 iterations, the difference between the two distributions becomes smaller. The learnt distribution keeps oscillating about the target distribution until it reaches equilibrium in the min-max game between the generator and discriminator. This is further illustrated in figure 2.25 which shows the fluctuating losses of both adversaries.



Iteration: 0

Iteration: 1000

Iteration: 2000

Figure 2.24: Synthetic (triangles) vs. real distribution (circles) while training CGAN-Habits.

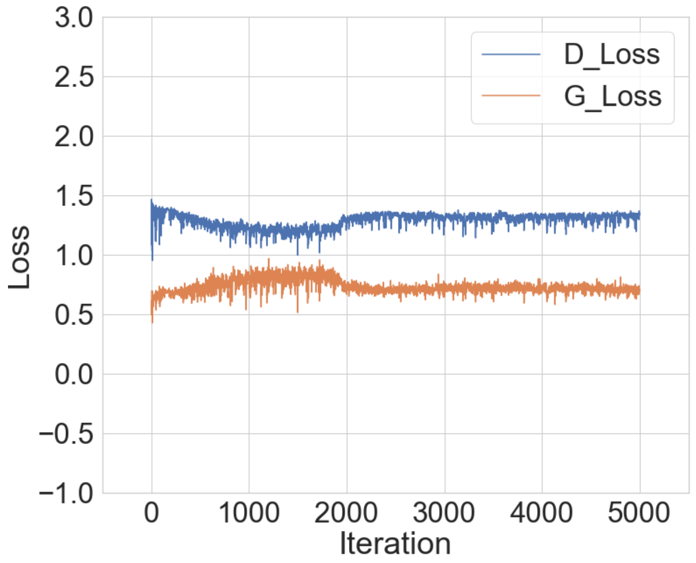
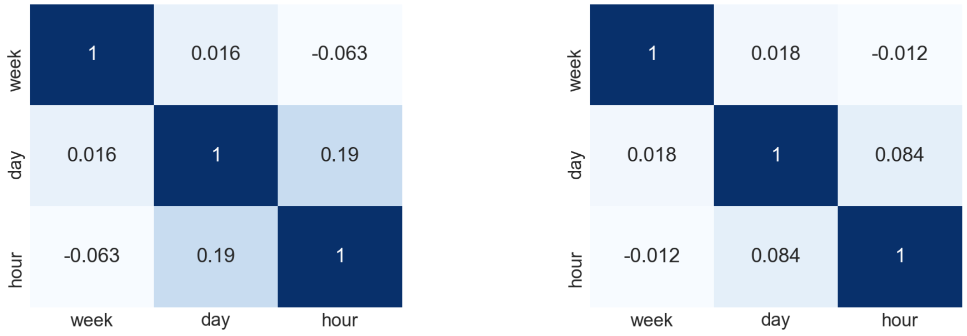
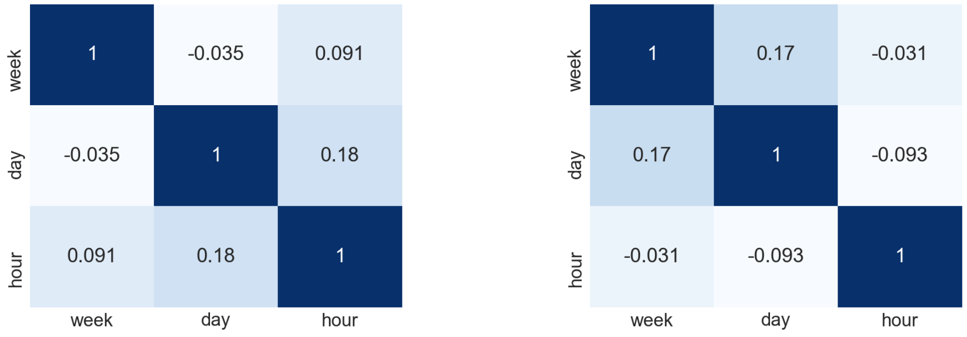


Figure 2.25: Loss while training CGAN-Habits

Figure 2.24 does not allow us to clearly see the final distribution of synthetic data compared to that of the real data. To simplify representation, we need to plot the final distribution of each feature for each load. However, this representation of features assumes that features are correlated. For that purpose, we investigate the correlation among the three features for each load. The correlations are shown in figure 2.26. Since correlations among the engineered features are relatively small, we can plot the distributions per load per feature. Distributions of the synthetic features generated by both CGAN-Patterns and KDE are plotted against the distribution of real samples. This is shown in figures 2.27, 2.28, 2.29 and 2.30. In general, the plots indicate that both KDE and CGAN=Habits successfully learnt the underlying distributions of the real engineered features. In some loads, the day-of-week exhibit the largest discrepancy. This discrepancy shall not be fully attributed to the error in learning the target distribution since features are correlated as mentioned earlier.



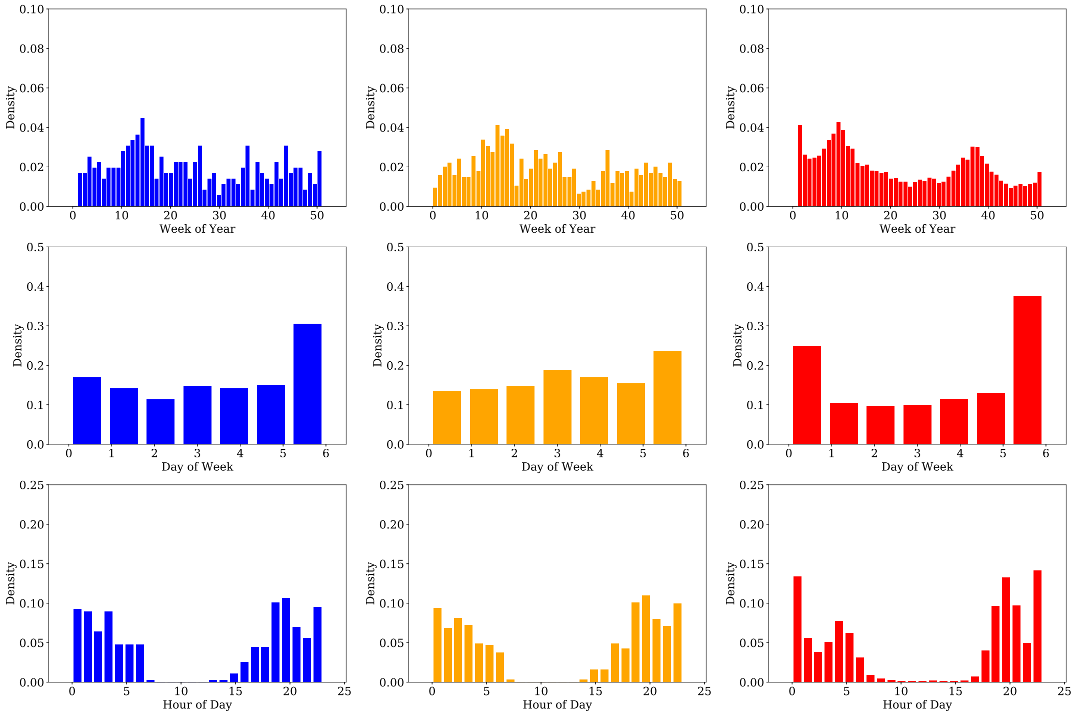
(a) Cloth Dryer (CDE) (b) Dishwasher (DWE)



(c) Fridge (FGE) (d) Heat pump (HPE)

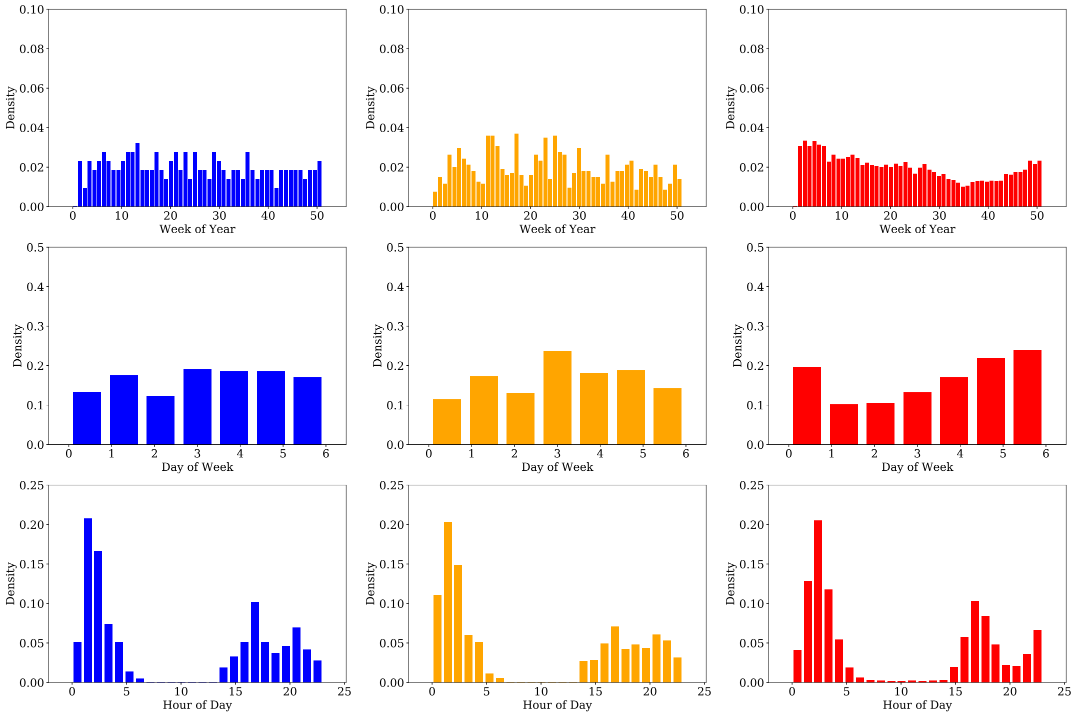
Figure 2.26: Correlation among engineered features for all loads.

Figure 2.31 shows that MMD for each load is decreasing along the training steps. This quality metric shows that CGAN-Habits successfully approached the target distribution of the



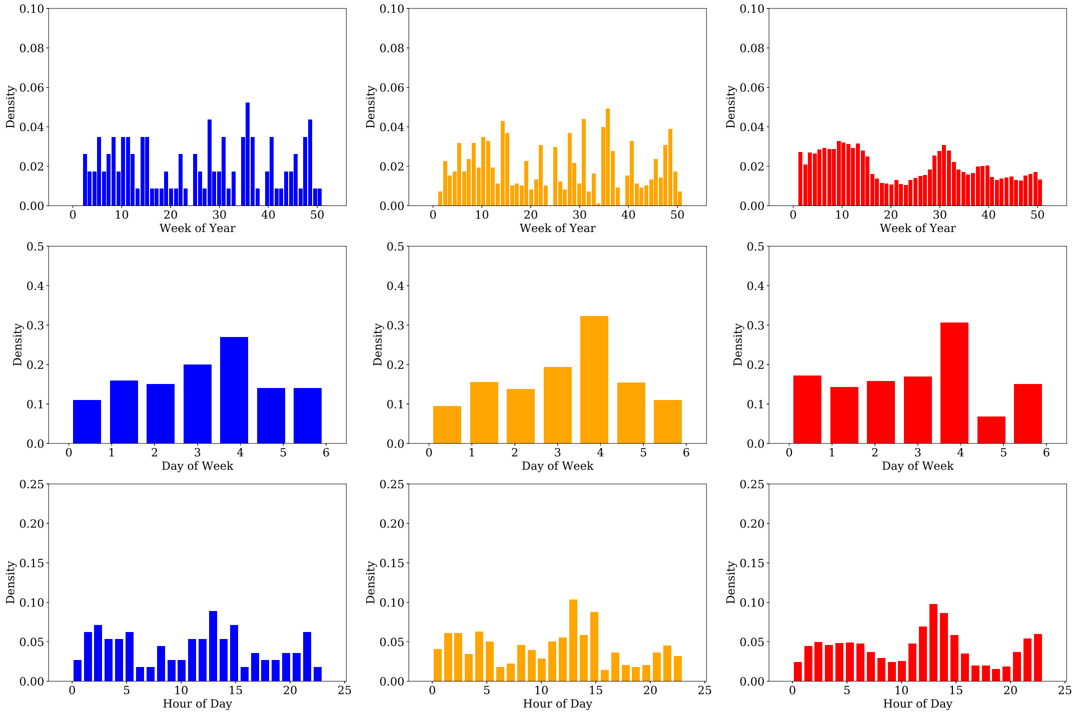
(a) Real (b) KDE (c) GAN

Figure 2.27: Histograms for usage habits of Cloth Dryer (CDE).



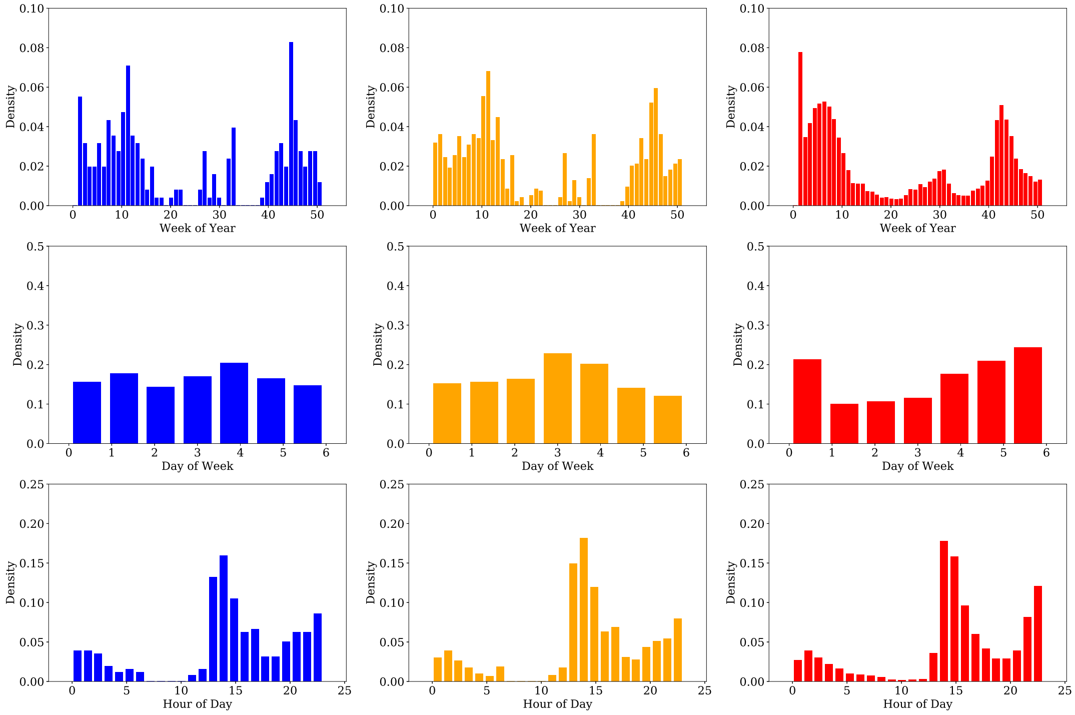
(a) Real (b) KDE (c) GAN

Figure 2.28: Histograms for usage habits of Dishwasher (DWE).



(a) Real (b) KDE (c) GAN

Figure 2.29: Histograms for usage habits of Fridge (FGE).



(a) Real (b) KDE (c) GAN

Figure 2.30: Histograms for usage habits of Heat pump (HPE).

real samples.

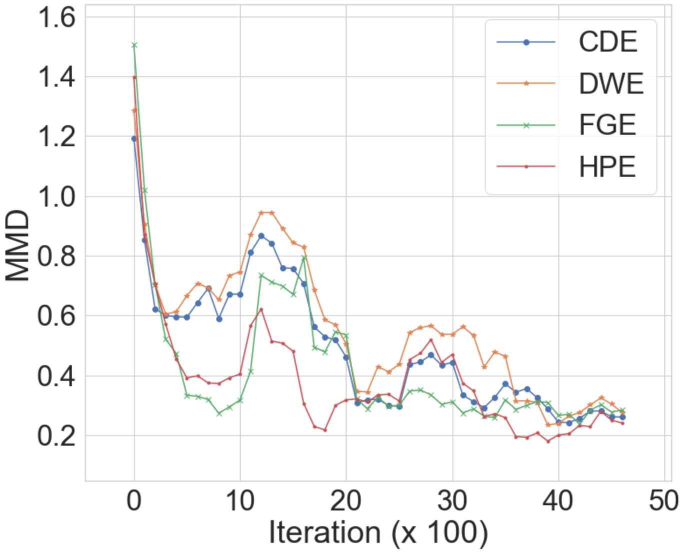


Figure 2.31: MMD for training CGAN-Habits

Finally, per load MMDs are calculated for both KDE and CGAN-Patterns. Table 2.5 summarizes the results. KDE surpasses CGAN-Habits in all four loads. However, KDE inherits the limitation of a non-parametric estimator. Hence, for each load, the training dataset shall be always saved to estimate the target distribution in real-time as per equation 2.35. On the other hand, for CGAN-Habits only the model's optimized need to be saved and a simple feed-forward operation takes place whenever a synthetic sample needs to be generated.

	Cloth Dryer	**Dishwasher**	**Fridge**	**Heat Pump**
GAN	0.08	0.13	0.20	0.10
KDE	0.03	0.04	0.06	0.03

Table 2.5: MMD comparison of usage habit synthesis.

2.5 Summary

In this chapter, we used the latest advancements in generative modelling to develop a novel framework for simulating residential electrical loads. The developed framework is a valuable tool for demand-side studies. Traditionally, simulating the power consumption of electrical loads depends on physics-driven models. In our case, data-driven models are used. In specific, generative models learn from historical measurements their underlying distributions. Once a distribution is learnt, synthetic data is generated by simply sampling from that distribution.

In the off-line training mode, the framework incorporates three stages: pre-processing the raw power measurements, training the generative models and finally evaluating the results. Once trained, the framework is used to simulate both power consumption patterns and usage habits of the modelled electrical loads.

The framework included conditional Generative Adversarial Network (GAN) as a major building block in addition to Kernel Density Estimator (KDE). Other novel contributions included the automatic extraction of training samples using matched filter and the elimination of GAN divergence problem by inverting the vanilla loss function.

The experimental study was conducted for four electrical loads. The framework successfully learnt the target distributions for both habits and patterns.

Chapter 3

Non-intrusive load monitoring

This chapter first discusses the basic theoretical background of the ML methods used in developing our proposed framework. These methods include hidden Markov model (HMM) and k-Means clustering. HMM is a generative modelling technique mainly used to model a sequence of random variables including time series. In specific, our discussion is limited to Markov models with discrete time steps and first-order Markov property as will be explained later. k-Means clustering is an unsupervised learning technique where no labelled data is present. We only provide general discussion about both HMM and k-Means without discussing the details of their underlying algorithms.

Next, the problem is defined while highlighting the computational complexity involved and assumptions that are usually made about emission distributions. Such assumptions are unlikely to hold for various types of loads and states. In the next section, we tackle these shortfalls by proposing a framework that takes advantage of ML techniques to bypass these assumptions. Finally, the numerical results are presented and the chapter concludes with a summary.

3.1 Background

3.1.1 Hidden Markov model

Hidden Markov Models (HMMs) is a widely used technique in sequence modelling. It has many applications. For instance, a sequence may be a set of words comprising a sentence or a times series data such as the active power consumption recorded by a smart meter.

A sequence is generated by a random process. Unlike random variables, where we may model the variable by a probability distribution, a sequence carries dependency among its successive instances. Each instance is considered as a random variable, i.e *state*, that is dependent on all previous states. Markov property tightens that dependency by assuming that the distribution of the current state depends only on the distribution of the previous state. Let S_t be the state at time t, then Markov property may expressed as $P(S_t|S_{t-1}, S_{t-2}, ..., S_0) = P(S_t|S_{t-1})$. The is called a first-order Markov chain. If the current state distribution depends on the previous two states' distributions, it is called a second-order Markov chain and so forth. To model the whole sequence of states in a first-order Markov chain, we can simply use the chain rule to find their joint distribution.

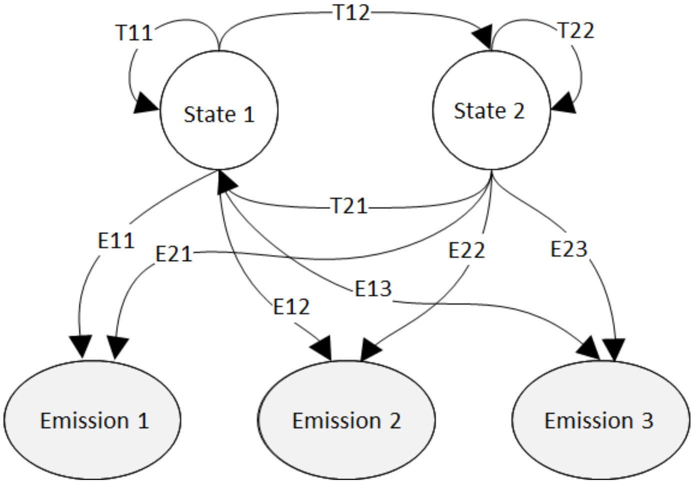


Figure 3.1: Hidden Markov Model

In *Hidden* Markov Models, states are not observable. Rather, we observe a sequence of emissions or observations caused by these hidden states. For example, for Part-of-Speech tagging, we can think about the sequence of words as the emissions and the underlying part-of-speech (i.e. noun, verb, etc.) as the hidden states.

Consider figure 3.1, the system assumes two states. At any time instance, we can only observe one of three discrete emissions. The probability of transitioning from State 1 to State 2 is T12. The probability of observing Emission 2 while in State 2 is E22 and so forth. Extending the number of states to N and the number of discrete emissions to M, we define [94]:

- $S = \{S_1, S_2, \ldots, S_N\}$ is the set of hidden states that may be assumed by the system
- $V = \{v_1, v_2, \ldots, v_M\}$ is the set of emissions that can be observed
- $\mathbf{q} = \{q_1 q_2 \cdots q_T\}$ is the sequence of states assumed by the system at time instances $1, 2, ..., T$
- $\mathbf{o} = \{o_1 o_2 \cdots o_T\}$ is the sequence of observations at time instances $1, 2, ..., T$

Accordingly, HMM involves two levels of randomness: probabilities associated with state-to-state transition and probabilities associated with state-to-observation emission. If we know the initial probabilities of the states, we can fully describe the model by :

$$\lambda = (A, B, \pi) \tag{3.1}$$

where:

- $B = \left[b_j(m)\right]$ is $N \times M$ emission probability matrix where $b_j(m) \equiv P\left(o_t = v_m | q_t = S_j\right)$ is probability of observing v_m at time t given that the system is in state S_j
- $A = \left[a_{ij}\right]$ is $N \times N$ transition probability matrix where $a_{ij} \equiv P\left(q_{t+1} = S_j | q_t = S_i\right)$ is probability of assuming state S_j at time $t+1$ given that the system is currently in state S_i
- $\pi = [\pi_i]$ is initial state probability vector of N dimension where $\pi_i \equiv P(q_1 = S_i)$

Based on the above, we can define the three common problems typically encountered in HMM [94]:

- Problem #1: Given an emission sequence $\mathbf{o}$ and the model's parameters λ, find the probability of that given sequence i.e. $P(\mathbf{o}|\lambda)$. This is typically solved using forward-backward algorithm which has a complexity of $O\left(N^2T\right)$
- Problem #2: Given an emission sequence $\mathbf{o}$ and the model's parameters λ. find the most likely underlying sequence of hidden states, i.e. find $\mathbf{q}^*$ that maximizes $P(\mathbf{q}/O, \lambda)$. Viterbi algorithm is typically used to solve this problem which also has a complexity of $O\left(N^2T\right)$.
- Problem #3: Find the model's parameters λ by fitting it to K training examples of emission sequences $\mathcal{X} = \{\mathbf{o}^1, \mathbf{o}^2, \ldots, \mathbf{o}^K\}$, i.e. find λ^* that maximizes $P(\mathcal{X}|\lambda)$. A variation of expectation-maximization algorithm, namely Baum-Welch algorithm, is used to solve this problem.

3.1.2 k-means clustering

k-Means is a clustering algorithm used in unsupervised learning when no labelled data is available. For example, while analyzing customer's spending behaviours, we may be interested to classify a customer as either a low spender or a high spender. In this case, we do not need historical labelled data classifying a customer as a low or high spender. Rather, we need to detect similarities in shopping habits among all the customers in the sample. k-Means clustering can be used to detect such similarities.

Assume a sample X with N sample points $X = \{\mathbf{x}_t\}_{t=1}^N$. Let us pick k reference vectors or centroids, $\mathbf{m}^j, j = 1, \ldots, k$. Note that $\mathbf{m}^j$ has the same dimension of $\mathbf{x}_t$. We can measure the distance (e.g Euclidean distance) between each of the picked reference vectors and every sample point. Each sample point will be allocated to the cluster that minimizes $||\mathbf{x}_t - \mathbf{m}^j||$. We need to find the optimized values of $\mathbf{m}_j$ across all the sampling points. The best centroids will be the ones that minimize the reconstruction error defined as [94]:

$$E\left(\left\{\mathbf{m}^i\right\}_{i=1}^k \mid X\right) = \sum_t \sum_i b_t^i \left\|\mathbf{x}_t - \mathbf{m}^i\right\|^2 \tag{3.2}$$

where $i : 1, \ldots, k$ is the index for reference vectors and $t : 1, \ldots, N$ is the index for sample points and b_t^i are the estimated labels given as:

$$b_t^i = \begin{cases} 1 & \text{if } \left\|\mathbf{x}_t - \mathbf{m}^i\right\| = \min_j \left\|\boldsymbol{x}_t - \mathbf{m}^j\right\| \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The first line in 3.3 means that a sample point is allocated to the centroid $\mathbf{m}^\mathbf{i}$ which is nearest to that sample point among all available k centroids $\mathbf{m}^j\ \forall j : 1..k$. The optimization problem given in 3.2 is solved iteratively. First, we set the values of $\mathbf{m}^i$ randomly and calculate b_t^i for all sample points. Finally, the algorithm converges to an optimized k number of centroids $\mathbf{m}^i$ to which each set of sample points are allocated. The returned centroid is the center point of the cluster.

In summary, if the input of the k-Means algorithm is sequence vector $\mathbf{o} = \{o_1 o_2 o_t \ldots\}$, we can obtain the corresponding sequence vector of estimated labels $\mathbf{y} = \{y_1 y_2 y_t \ldots\}$ such that $\forall t,\ y_t \in \{1, 2, \ldots, k\}$.

3.1.3 Quality metrics

Accuracy is a widely used metric to evaluate the performance of a classifier. For instance, a classifier may be used to predict if a certain load is 'on' or 'off'. This is a binary classier. Usually, in supervised machine learning algorithms, a labelled dataset is provided. The dataset is divided into training and testing. The data samples used for training are not used for testing. After constructing the classifier, the *true* labels provided in the testing dataset are compared with labels *predicted* by the classifier. Since classification is binary, we may define one label as positive (e.g. 1 or 'on' state of an electrical load) and the other label as negative (e.g. 0 or 'off' state). If a sample is correctly labelled as positive, we count this as true positive (TP). If a sample is incorrectly labelled as positive, we count this as (FP). Similarly, if a sample is correctly labelled as negative, we count this as true negative (TN). If a sample is incorrectly labelled as negative, we count this as false negative (FN). Accordingly, the model accuracy can be calculated using equation 3.4 [100].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.4}$$

When the dataset includes unbalanced samples from both labels, accuracy becomes insufficient to judge the soundness of the model. The two metrics given in equations 3.5 and 3.6 expose that problem.

$$Precision = \frac{TP}{TP + FP} \tag{3.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.6}$$

Usually, any improvement in the model's precision will be at the expense of recall and vice versa. As such, a single metric (F_1) score is used to describe the classifier's performance. The F_1 score is given in 3.7.

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.7}$$

3.2 Problem definition

In general, an electricity meter records the aggregate energy consumption of a household in kW.h. Smart meters make such readings available at fixed time intervals with relatively high granularity. For instance, smart meters installed by London Hydro may provide aggregate energy consumption of a household every 4 minutes. Given that power is the rate of change of energy, we can convert energy readings into power at fixed sampling intervals. Let ΔE be the kW.h change in energy in the sampling interval T_s in seconds. The estimated power in Watts at T_s is $\Delta E \times 1000 \times 3600 \times \frac{1}{T_s}$. Accordingly, given the energy readings at fixed time intervals T_s, we can obtain a sequence of power values for the aggregate load of a household. Consider:

$\mathbf{o}^A = \left\{o_1^A o_2^A \cdots o_T^A\right\}$: the observed aggregate power measurements at time instances $1, 2, \ldots, T$. Alternatively, this may be the estimated power from the recorded energy samples as discussed above.

$\mathbf{q}^l = \left\{q_1^l q_2^l \cdots q_T^l\right\}$: the sequence of states assumed by the electrical load l at time instances $1, 2, ..., T$.

L: number of connected electrical loads to the smart meter.

At any point in time, a load is operating in one of its states. For instance, if load l is characterized by K states, then $q_t^l \in \{S_1, S_2, \ldots, S_K\}$. In general, different loads may have a different

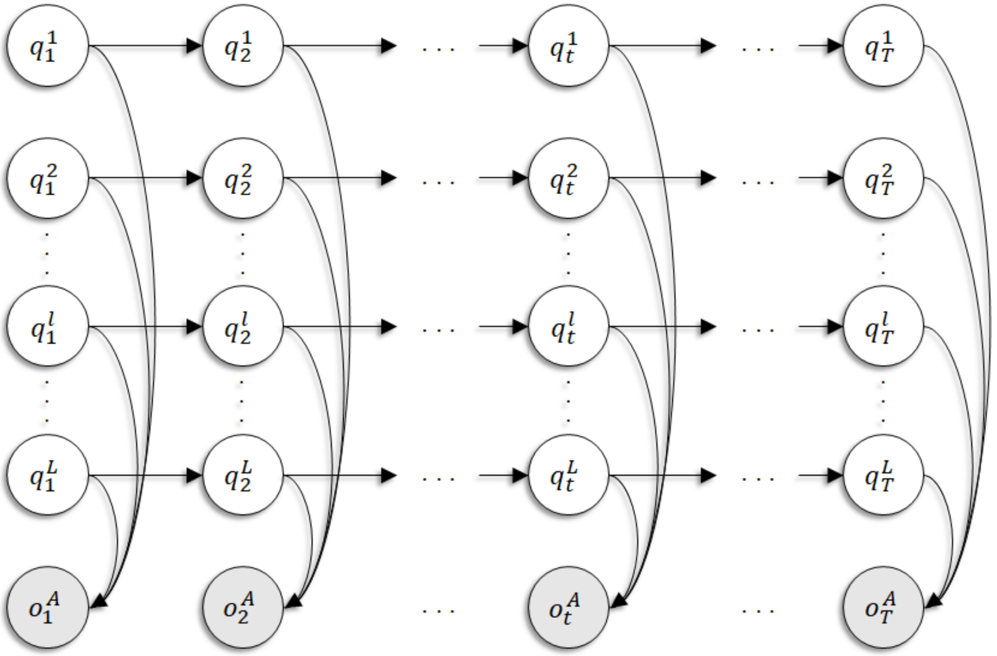


Figure 3.2: Factorial hidden Markov model where the aggregate power o_t^A at any point in time t is dependent on the internal states of all the connected loads at that time instance.

number of states. This is slightly different than what was explained in 3.1.1 where the observation herein is dependent on a collection of states of independent systems (i.e. loads) instead of being dependent on the states of only one system or chain. This is called Factorial Hidden Markov Model (FHMM) and is illustrated in figure 3.2. In FHMM, we may think about

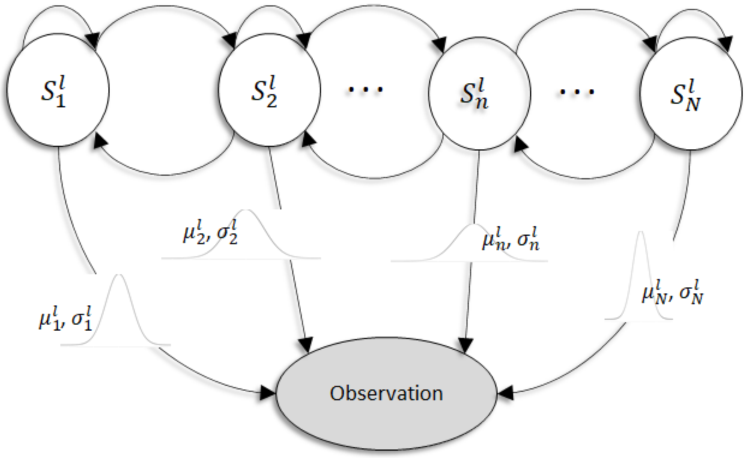


Figure 3.3: Hidden Markov chain for load l with univariate emission distribution such for observed power.

the combination of loads as one single virtual load. The virtual load will have a number of states which is equal to the sum of all states associated with the individual loads. For instance, consider an aggregate power observation comprising only two loads. If load #1 has 3 states and load #2 has 5 states, then the virtual load may be considered as having a total of 8 states. However, the transition probabilities associated with transitions between the states of the individual loads are all set to zero. In the other words, transitions are only allowed between states belonging to the same individual load.

We can generalize further if we assume $\mathbf{o}^A$ to be multivariate. For instance, the smart meter may be sampled for both the active power and power factor. In this case, o_t^A will have a dimension of two, each impacted by the same combination of states at time t.

The objective is to find the underlying states for all connected loads. First, the model parameters need to be found (problem #3 in 3.1.1). However, a certain emission distribution needs to be assumed. Consider the hidden Markov chain for load l in figure 3.3. Given the training dataset, transition probabilities among all states from 1 to N shall be estimated. Further, emission distribution parameters conditioned on each state shall be estimated. For that purpose, the emission is assumed to follow a certain distribution (e.g. Gaussian in figure 3.3). Then the corresponding parameters conditioned on each state shall be estimated. Note that if an observation is multivariate (e.g. both active and reactive power), a covariance matrix conditioned on each state rather than variance needs to be estimated. In the FHMM set-up (figure 3.2), the parameters shall be estimated for all loads. The assumption that all emission distributions follow the same type introduces errors in the model. Further, considering all loads and their underlying states, the estimation process becomes computationally expensive. In our proposal, we try to simplify the modelling procedure by eliminating the assumption about the type of emission distribution. Another question we answer is that how many states shall we consider for each load. Indeed, this will impact the complexity of the Markov chain and reflect on the computational complexity.

Once the parameters are estimated, Viterbi algorithm (problem #2 in 3.1.1) is used to decode the hidden states.

3.3 Proposed framework

The proposed framework in this chapter is related to the author's work published in reference [75]. As discussed in 3.2, exiting HMM-based solutions for the NILM problem have high computational complexity. Besides, they make assumptions about emission distributions conditioned on various states.

Typically, in FHMM, given the emission sequence, we can find the underlying model (equation 3.1) using Baum-Welch algorithm. However, we eliminate that by proposing a hybrid model that combines both HMM and k-Means clustering to estimate the model parameters.

In our proposed framework, we focus on simplicity. As such, we make the assumption that for any individual load l, only two states are considered: low power (S^0) and high power (S^1). This is a reasonable assumption since our interest in NILM is just to know if a load is OFF or ON at any instant. Let $\mathbf{q}^l = \left\{q_1^l q_2^t \cdots q_T^l\right\}$ be the sequence of hidden states for load l for a duration of T. For any sequence instant q_t^l, we can state our assumption as: $q_t^l \in \{S^0, S^1\}$. We further assume that our observed aggregate quantity is univariate (e.g only active power). If $\mathbf{o}^l = \left\{o_1^A o_2^A \cdots o_T^A\right\}$ is the sequence of observed aggregate signal for a duration of T, then for any sequence instant o_t^A, we can state our assumption as: $o_t^A \in \mathcal{R}$. These two assumptions will substantially reduce computational complexity since it is dependent on the number of states as explained earlier in 3.1.1. On the other hand, we do not make any assumptions about the emission distributions that are conditioned on various load states.

Figure 3.4 shows the main blocks of the proposal. First, physical measurements of individual electrical loads are used for training. Consider L number of loads with physical measurement samples for a duration of T such that $\mathbf{p}^l = \{p_1^l, p_2^l, \ldots, p_T^l\}\ \forall\ l : 1, \ldots, L$. The physical measurements are cleaned and preprocessed by removing noise and outliers to obtain the corresponding observation sequences $\mathbf{o}^l = \{o_1^l, o_2^l, \ldots, o_T^l\}\ \forall\ l : 1, \ldots, L$. The proposed Hybrid Training Algorithm (HTA) uses the input observations to estimate the parameters of the FHMM model. The parameters include the transition matrix of the aggregate signal (A^A) and the emission matrix of the aggregates signal (B^A). Further, the resulting centroids for the aggregate model are passed for clustering the aggregate observations $\mathbf{o}^A$ into a sequence of labels $\mathbf{y}^A$.

During production, the input aggregate signal ($\mathbf{p}^A$), typically acquired by the smart meter, is first processed to eliminate any outliers or temporal inconsistency in the sampling interval. The inconsistent sampling rate can be easily treated using interpolation. The obtained clean sequence of observations of the aggregate signal ($\mathbf{o}^A$) is converted into a sequence of labels ($\mathbf{y}^A$). This is achieved by measuring the Euclidean distance between each observation point and each centroid calculated by HTA. The observation point is assigned to the label of the centroid with the nearest distance. Equation 3.8 describes the process of converting the sequence of observations to a sequence of labels.

$$y_t^A = \underset{j}{\operatorname{argmin}} \left\| o_t^A - c_j^A \right\| \tag{3.8}$$

where:

o_t^A is an observation point (e.g.clean active power) at instant t in the observation sequence $\mathbf{o}^A = \{o_1^A \ldots o_t^A \ldots o_T^A\}$.

y_t^A is a label assigned to the observation point o_t^A at instant t in the label sequence $\mathbf{y}^A =$

$\{y_1^A \ldots, y_t^A \ldots y_T^A\}$.

c_j^A is the j^{th} component of the centroid vector $\mathbf{c}^A$ corresponding to the aggregate signal as calculated by HTA such that $\mathbf{c}^A \in \mathcal{R}^{k^A}$.

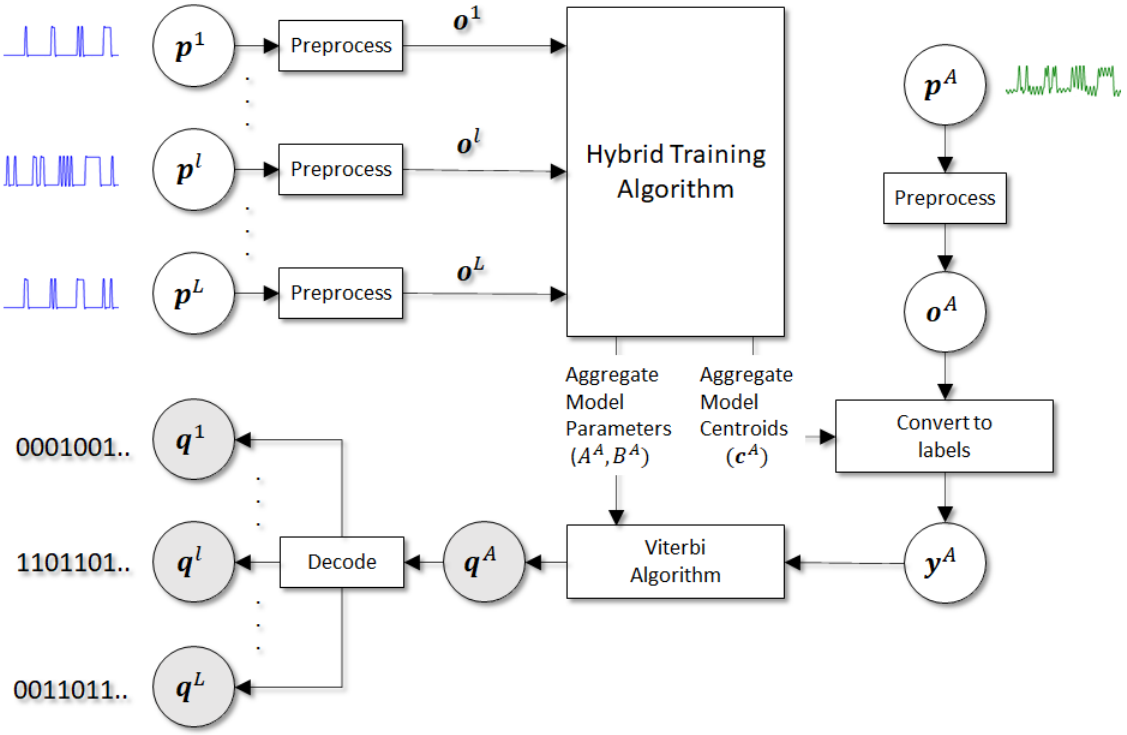


Figure 3.4: Hybrid Algorithm for NILM

The number of centroids k^A calculated by HTA is dependent on the number of centroids associated with each individual model. Let the number of centroids associated with the l^{th} individual model be k^l, then for L individual models comprising the aggregate signal, the number of centroids corresponding to the aggregate signal is calculated as:

$$k^A = \prod_{l=1}^{L} k^l \tag{3.9}$$

The Viterbi algorithm in figure 3.4 uses the saved model parameters (A^A, B^A) to process the input sequence of labels $\mathbf{y}^A$. The algorithm predicts $\mathbf{q}^A$ which is the likely underlying hidden state sequence of the aggregate signal. Further decoding is necessary to obtain the sequence of hidden states for every load i.e. $\mathbf{Q}^l \; \forall \; l : 1, \ldots, L$. The number of hidden states for the aggregate signal is a combination of the states in the individual loads. Since each individual load is assumed to have two states only, the number of states in the aggregate signal is 2^L where L is the number of individual loads. In general, the number of states in the aggregate signal equals the product of the number of states for the individual loads. Hence, we can write:

$$N^A = \prod_{l=1}^{L} N^l \tag{3.10}$$

where:

L: no. of individual loads

N^l: No. of states of the l^{th} load.

N^A: No. of states of the aggregate load.

Our proposed HTA is described in algorithm 2. As input, the algorithm receives clean discrete sequences ($\mathbf{o}^l = \{o^l_1 \dots o^l_t \dots o^l_T\} \; \forall l \in \{1, \dots, L\}$) representing the processed measured active power values at fixed time intervals throughout a period of T for various loads $l : 1, \dots, L$ connected in a house. The received observations are clean and have consistent sampling intervals. The input signals are used by HTA to estimate the transition and emission matrices (A^A, B^A) for the aggregate clean observation sequence ($\mathbf{o}^A$) as shown in figure 3.4. Further HTA returns the centroid vector $\mathbf{c}^A$ that is needed during the real-time disaggregation process to convert the aggregate signal $\mathbf{o}^A$ to the corresponding labelled sequence $\mathbf{y}^A$ as per equation 3.8.

The algorithm starts by initializing a transition matrix A^l for each load l with a size of 2×2. The size stems from the fact that we only consider in our model two states corresponding to low power S_0 (first row in the emission matrix) and high power S_1 (second row in the emission matrix) as explained earlier. The transition matrix will be filled with transition probabilities between states as explained in 3.1. The algorithm also initializes an emission matrix B^l for each load l with a size of $2 \times k^l$. The emission matrix summarizes the distributions conditioned on every single state. The first row corresponds to emission probabilities conditioned on the low power state S_0 while the second row is conditioned on the high power state S_1.

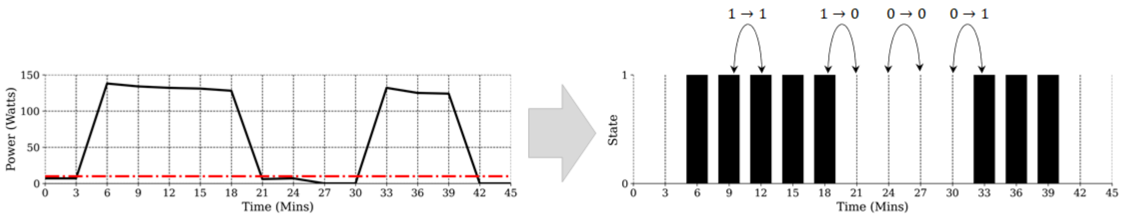


Figure 3.5: Illustration of steps 4 and 5 in HTA algorithm. Clean observations $\mathbf{o^l}$ (left) for load l are converted to sequence of states $\mathbf{q^l}$ (right) by applying a threshold τ (red dotted line). The four transition frequencies between states are calculated and the load's transition matrix A^l is updated accordingly.

Once A^l and B^l are initialized, $\mathbf{o}^l$ is converted to a sequence of states $\mathbf{q}^l$ by applying a threshold (τ). The threshold is very low power value (e.g. 10 Watts) to distinguish $\mathbf{S_0}$ states

Algorithm 2 Hybrid Training Algorithm

```
Input: $\mathbf{o}^l \; \forall l : 1, \ldots, L$ where $L$ is no. of loads
    $k^l$: No. of clusters $\forall l : 1, \ldots, L$
Output: $A^A = [a^A_{i,j}]$: Transition matrix for aggregate signal
    $B^A = [b^A_{i,j}]$: Emission matrix for aggregate signal
    $\mathbf{c}^A$: Centroid vector for aggregate signal
 1: for $l$=1 to $L$ do
 2:     Initialize zero transition matrix $A^l = [a^l_{i,j}]$ for load $l$ with size $2 \times 2$
 3:     Initialize zero emission matrix $B^l = [b^l_{i,j}]$ for load $l$ with size $2 \times k^l$
 4:     $\mathbf{q}^l \leftarrow$ convert $\mathbf{o}^l$ to state sequence by applying a threshold $\tau$
 5:     $a^l_{0,0}, a^l_{0,1}, a^l_{1,0}, a^l_{1,1} \leftarrow$ calculate frequency of transitions between states in $\mathbf{q}^l$
 6:     $\mathbf{c}^l, \mathbf{y}^l \leftarrow$ apply $k^l$-Means clustering to $\mathbf{o}^l$ and obtain corresponding centroids and labelled sequence
 7:     $\tilde{c}, \tilde{k} \leftarrow \min(\mathbf{c}^l)$: find lowest power centroid $\tilde{c}$ and its label $\tilde{k}$
 8:     $b^l_{1,\tilde{k}} \leftarrow 1$: update element corresponding to lowest power in the first row which corresponds to the low power state
 9:     for $k$=1 to $k^l$ do
10:         $b^l_{2,k} \leftarrow$ calculate the frequency of label $k$ in $\mathbf{y}^l$
11:     end for
12:     $b^l_{2,\tilde{k}} \leftarrow 0$: update element corresponding to lowest power in the second row which corresponds to the high power state
13:     $b^l_{2,k} \leftarrow \frac{b^l_{2,k}}{\sum_{k=1}^{k^l} b^l_{2,k}}$: normalize second row
14: end for
15: for $l$=1 to $L$ do
16:     $A^A \leftarrow \bigotimes_{l=1}^{L} A^l$: Kronecker product
17:     $B^A \leftarrow \bigotimes_{l=1}^{L} B^l$: Kronecker product
18:     $\mathbf{c}^A \leftarrow \bigoplus_{l=1}^{L} \mathbf{c}^l$: Outer summation
19: end for
```

from $\mathbf{S_1}$ states. Then transitions between different states are counted and divided by total number of samples to calculate the corresponding frequencies. These will be the estimated transition probabilities and the A^l is updated accordingly. Figure 3.5 illustrates lines 4 and 5 in algorithm 2.

The sequence of observations $\mathbf{o}^l$ is passed to the k-Means algorithm. The number of clusters is determined by the input k^l. k-Means basically maps each observation sample to the label corresponding to the nearest centroid (i.e. center of cluster) to that observation as explained in

3.1.2. Hence, the sequence of power observations for each load $\mathbf{o}^l$ is converted into a sequence of labels $\mathbf{y}^l$ and the set of centroids are returned as the centroid vector $\mathbf{c}^l$ (line 6 in algorithm 2).

The low power state can *not* be assumed to emit zero emissions. Such an assumption will result in the cancellation of many load combinations due to multiplication by zero. Hence, we assign a probability of 1 to be associated with the centroid having the lowest power (line 8 in the algorithm 2). This probability is forced to zero (line 12 in the algorithm 2) when it is conditioned on the high power state (row 2). The value and index of the centroid that resulted in the lowest power is obtained in line 7.

The rest of emission probabilities conditioned on S_1 are obtained by finding the frequency pertaining to each cluster or label (line 10 in algorithm 2). Finding frequencies is equivalent to finding the histogram of $\mathbf{y}^l$ as depicted by figure 3.6. However, since the probability pertaining to the lowest power and conditioned on S_1 is forced to be zero, the rest of the probabilities conditioned on S_1 shall be normalized so they sum up to 1 (line 13).

Finally, the transition matrix A^A for the aggregate signal is found by Kronecker multiplication of the component transition matrices. The same applies for the emission matrix B^A as depicted by lines 16 and 17 in algorithm 2. The centroids for the aggregate signal $\mathbf{c}^A$ are calculated using outer summation of all the individual loads. As an example, consider load 1 with centroid vector $\mathbf{c}^1 = \{c^1_1, c^1_2, c^1_3\}$ and load 2 with centroid vector $\mathbf{c}^2 = \{c^2_1, c^2_2\}$, then $\mathbf{c}^1 \bigoplus \mathbf{c}^2 = \{c^1_1 + c^2_1, c^1_1 + c^2_2, c^1_2 + c^2_1, c^1_2 + c^2_2, c^1_3 + c^2_1, c^1_3 + c^2_2\}$.

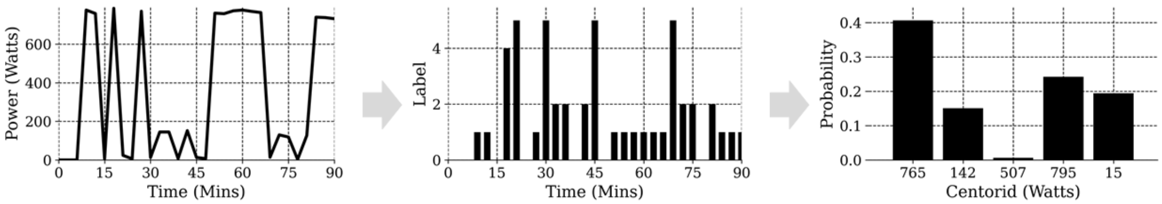


Figure 3.6: Illustration of step 10 in HTA algorithm. Clean observations $\mathbf{o}^\mathbf{l}$ (left) for load l are converted to sequence of lables $\mathbf{y}^\mathbf{l}$ (middle) using k^l-Means algorithm.The distribution of data is found (right) and the load's emission matrix B^l is updated accordingly.

3.4 Experimental studies

We use the dataset described in appendix A for our experiment. Given the aggregate whole house power signal (WHE), we define our objective to disaggregate the signal and identify the status of three appliances namely: the cloth dryer (CDE), the fridge (FGE) and the heat pump (HPE). Figure 3.7 shows the aggregate active power signal for 450 hours.

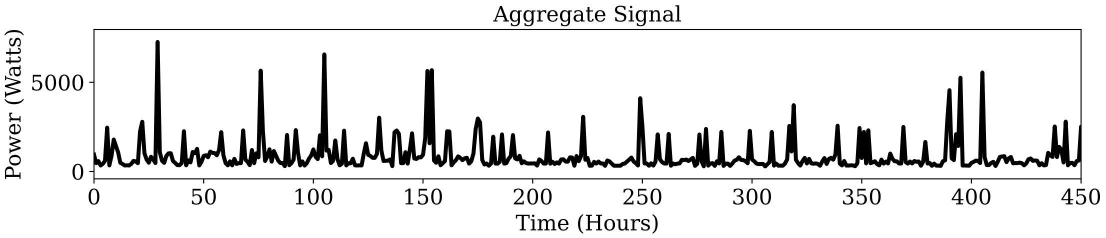


Figure 3.7: Raw whole house active power signal

For training purposes, we use the physical measurement of power for the three appliances under consideration i.e CDE, FGE and HPE. Data is provided at a granularity of 1 minute. The physical measurements are relatively clean, so only basic detection of outliers is performed. We detect outliers by excluding samples that are substantially above the standard deviation. For both HPE and CDE no outliers were detected while 80 outliers were detected for FGE. Figure 3.8 shows three signals.

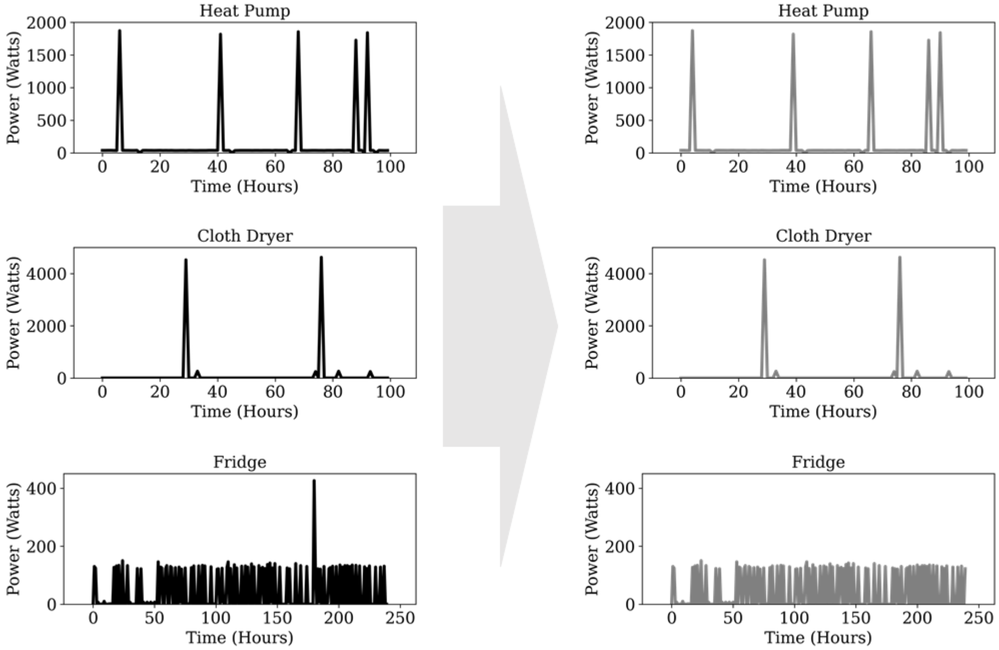


Figure 3.8: Removing outliers from training signals

It is unlikely that utilities provide customers' data at that granularity. Hence, the three signals are down-sampled to 3 mins intervals to resemble real-life situations. The three signals are ingested to the hybrid training algorithm (HTA) shown in figure 3.4.

HTP converts the signals to sequences of states. Figure 3.9 shows the resulting sequences for the three individual loads. The red dotted line indicates the applied threshold. Note that

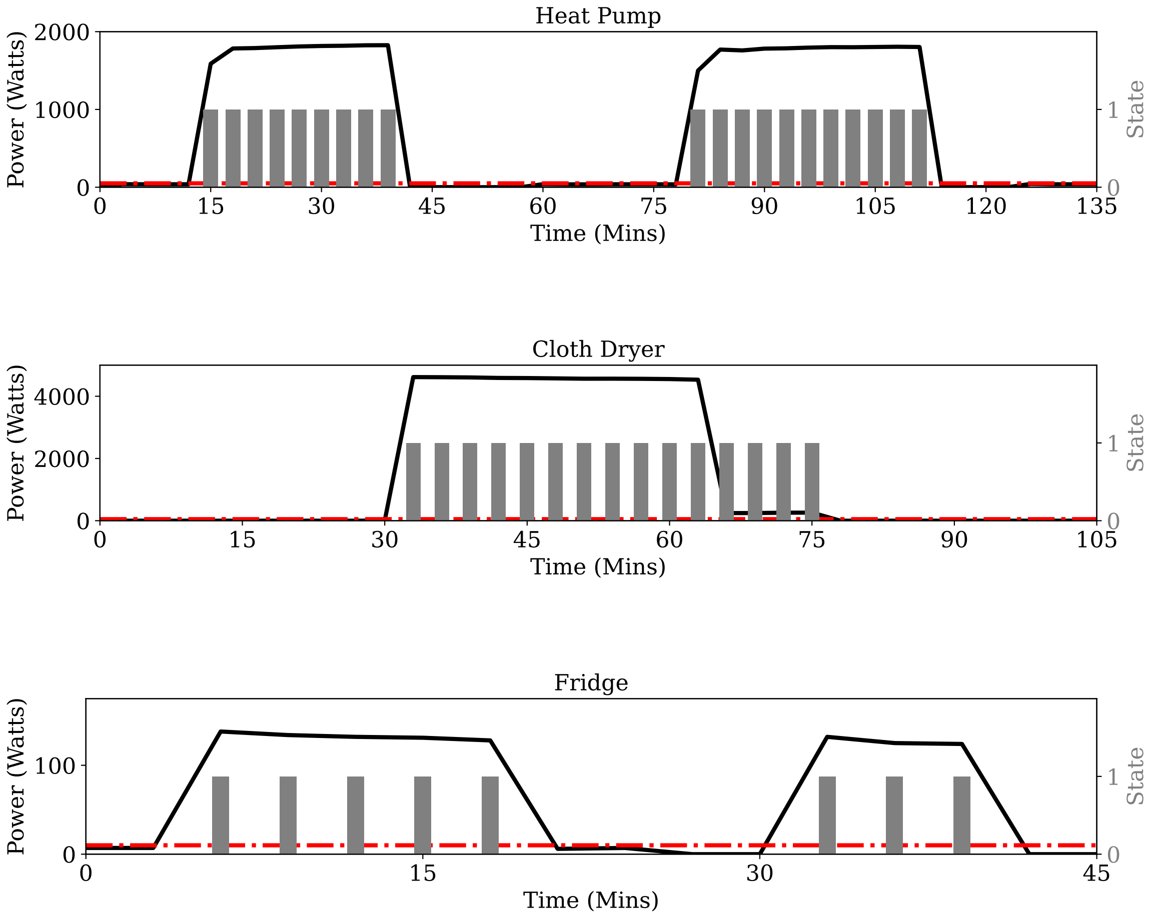


Figure 3.9: Conversion of observed power values for various appliances to two-state sequence.

for the cloth dryer (CDE), its low power consumption (trailing edge) is converted to a state of ’1’ since the load is still operational. As discussed earlier, our objective is not to identify the internal states of a load, rather it is to identify whether that load is on or off. The transition matrix for each load is obtained by counting the $0 - 1$ transitions as explained in 3.3.

Next, HTP performs k-Means clustering on observed sequences and converts them to label sequences as depicted by figure 3.10. The figures on the left represent the observation sequences (cleaned active power) for the three loads. The observations are converted to sequences of discrete labels (figures in the middle). Labels are indices of centroids. The labels and their respective centroids are given in tabular format on the right top of each figure. While we clustered each load into three clusters (i.e. $k = 3$ for all loads, we show only two centroids. This is because in our algorithm 2, we dedicate the centroid with the lowest power with S_0 and here we are interested only with the clusters pertaining to S_1, i.e. the state that emits higher power. Note that these centroids are not sorted. As such, a smaller label may be associated with a higher value centroid. For instance, for a cloth dryer, if a sample point is labelled ’1’, it means that the point has a relatively higher power value than a point labelled ’2’, since label ’1’ is associated with a power of 4727 Watts while label ’2’ is associated with a power of just 266 Watts. Accordingly, the sequences of labels (figures on the middle) do *not* follow the same waveforms of observations (figures on the left). The column on the right represents the distribution of centroids across the observation sequences. For instance, the heat pump has most of the observed points clustered around 1778 Watts and so forth.

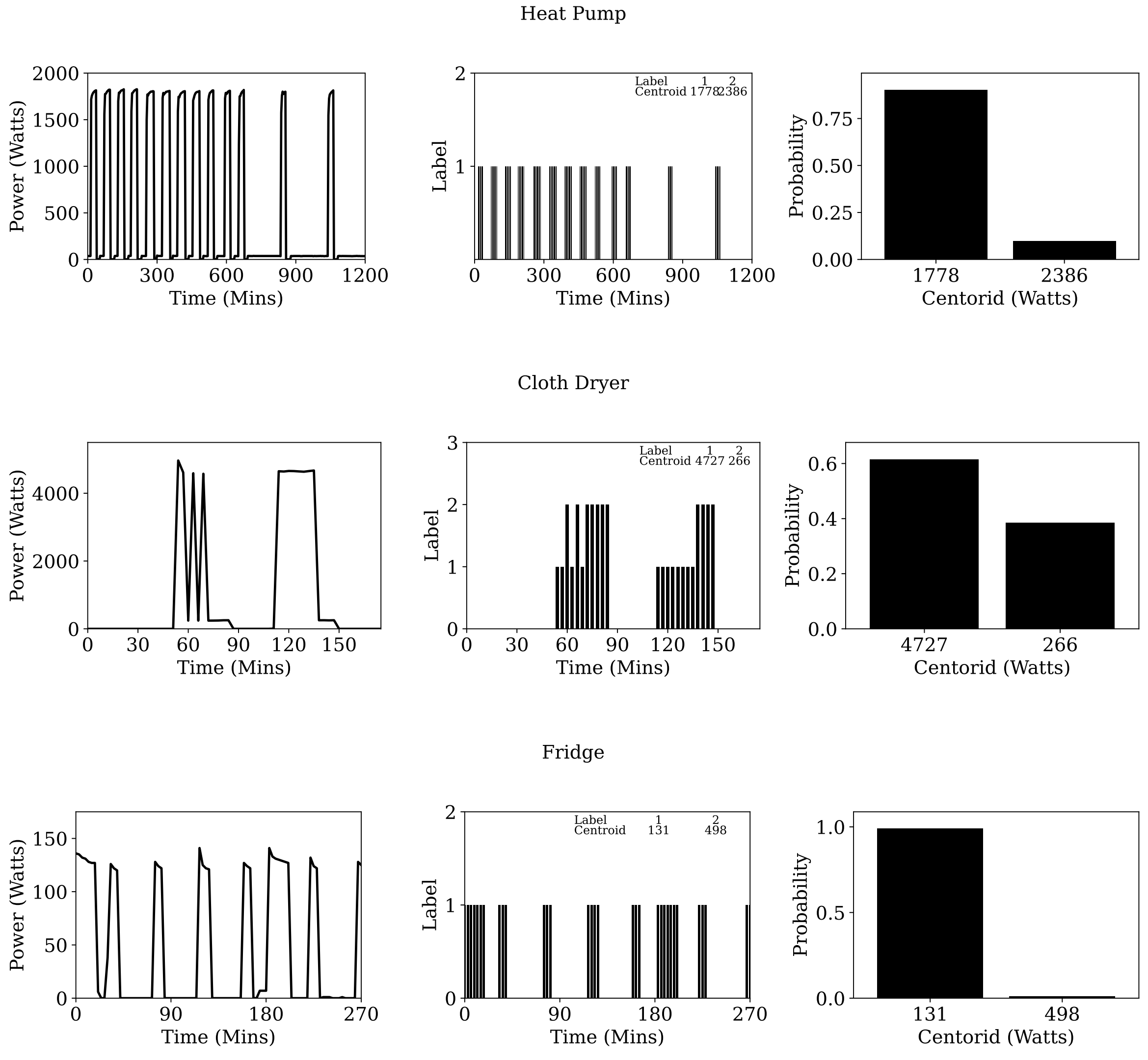


Figure 3.10: k-Means clustering of various load signals.

Once the model is trained according to algorithm 2, it is implemented to disaggregate the smart meter signal. The aggregate observations (figure 3.11) is converted to sequence of labels as explained in 3.3. Since we have three loads and each is clustered into three clusters, we end up with $3^3 = 27$ centroids that are used to convert the aggregate signal to a sequence of labels (equation 3.9). Given both aggregate transition and emission matrices as estimated by HTA, Viterbi algorithm is used to predict the hidden states of the aggregate signal. Since each load has two hidden states, the number of states for the aggregate signal is $2^3 = 8$ as per equation 3.10. These represent the combination of states of the three loads as shown on the y-axis of figure 3.13.

The inferred state sequence for the aggregate signal is further decoded to identify the hidden state sequence for each individual load. The resulting state sequence train for load has a duration that is equal to the ingested aggregate signal as depicted by figure 3.10. This concludes the disaggregation process.

The AMPd dataset provides physically measured data for both the whole house signal (i.e. aggregate) and the individual loads. This allows us to split the data into training and testing datasets. We use the training dataset for constructing the model parameters for the aggregate signal using HTA. Once trained, the model's performance is tested using the testing dataset. The resulting individual states are compared to those given in the testing dataset as per AMPd. Table 3.1 summarizes both accuracy and F_1 for each load. The cloth dryer scored the lowest F_1 score, yet the value is considered satisfactory in the NILM literature. For instance, [101] reports an enhanced F_1 score of 83.5% for one of the tested appliances compared with 66.7% when using Bayesian classifier.

Load	No. of Centroids	Accuracy (%)	F_1 (%)
Cloth Dryer (CDE)	3	99.2	81.7
Fridge (FGE)	3	98.1	97.4
Heat Pump	3	99.9	99.4

Table 3.1: Results of proposed non-intrusive load monitoring showing remarkable disaggregation power.

3.5 Summary

In this chapter, a framework to perform non-intrusive load monitoring is proposed. At its core, the framework adopts hidden Markov chains, being one of the widely used generative modelling techniques. Further, a novel hybrid training algorithm (HTA) is proposed. HTA eliminates the need to make assumptions about the emission distribution associated with various electrical loads by incorporating k-Means unsupervised learning algorithm. Instead of restricting its output to both transition and emission distributions for the aggregate signal, HTA

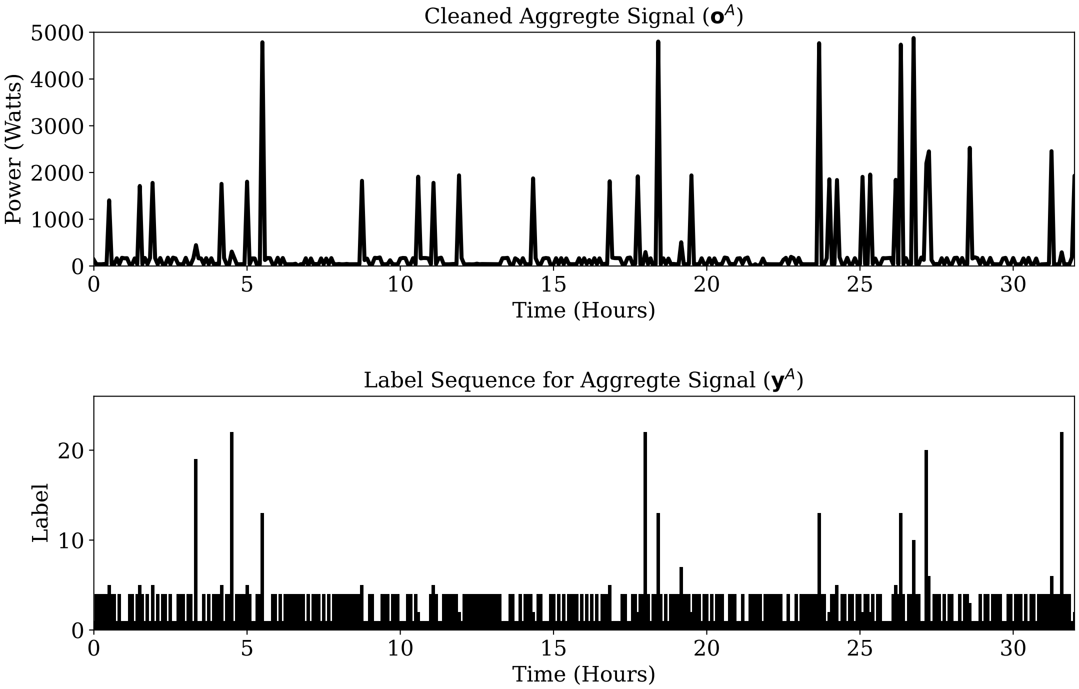


Figure 3.11: Converting aggregate power signal to labels

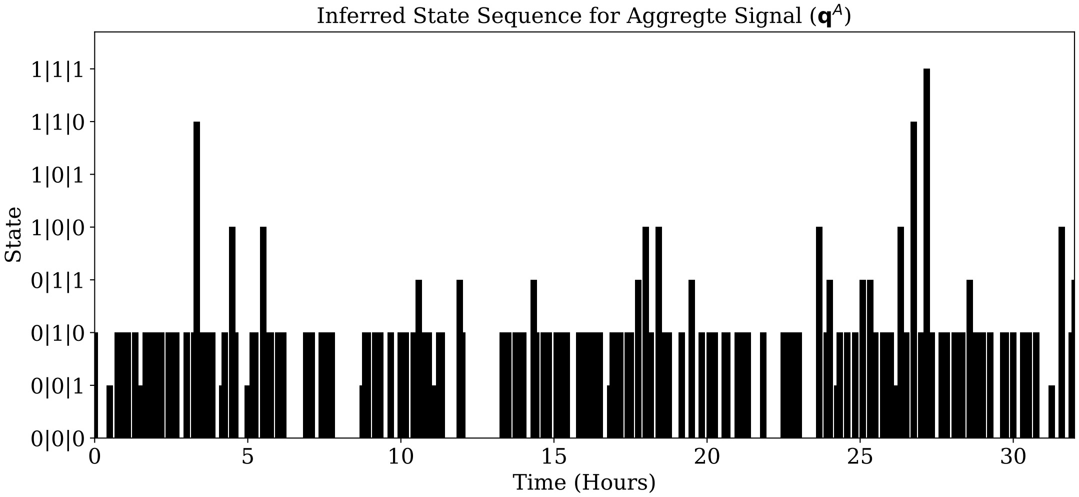
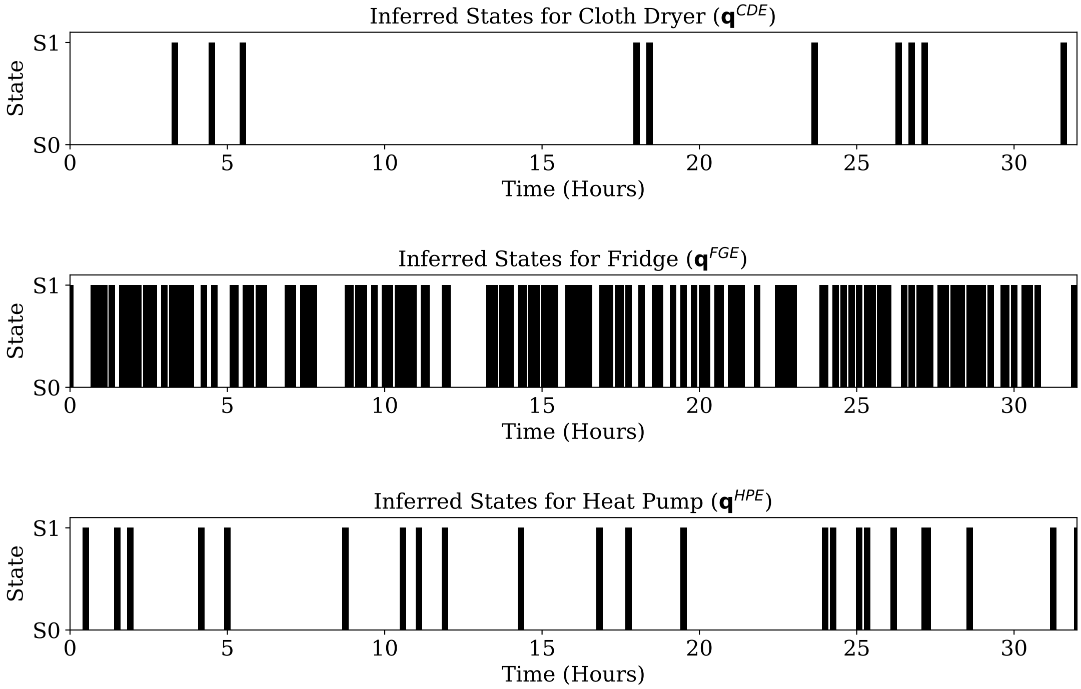


Figure 3.12: Inferring states of aggregate signal

outputs a list of all centroids that can be used to convert the aggregate signal to a sequence of labels. This eliminates the need to use Baum-Welch algorithm to solve for HMM's parameters. When implementing and tested using the dataset described in A, the model scored remarkably



Inferred States for Cloth Dryer ($\mathbf{q}^{CDE}$)

Inferred States for Fridge ($\mathbf{q}^{FGE}$)

Inferred States for Heat Pump ($\mathbf{q}^{HPE}$)

Figure 3.13: Decoding states of aggregate signal into individual loads

high accuracies.

Chapter 4

Power state estimation

This chapter focuses on the application of generative models to Power System State Estimation (PSSE). First, the necessary background for estimating power states is presented using the widely adopted method namely Weighted Least Squares (WLS). In addition to state estimation, traditional bad data detection (BDD) is discussed in a separate section. This is followed by a background about cycle Generative Adversarial Networks which is based on the material discussed in 2.1.1, 2.1.2 and 3.1.3.

Once the necessary background is discussed, the problem is stated followed by our proposed generative data-driven framework for state estimation and bad data detection.

Finally, experimental results using IEEE-118 test case are discussed and followed by the chapter's summary.

4.1 Background

4.1.1 Power state estimation

Power state estimation refers to the statistical estimation of voltage phasors at all buses using a redundant set of measurements. The IEEE task force [102] defines two distinct operating conditions of a power grid: quasi-state and transient. In quasi-steady-state operating conditions, the system operating point changes due to slow and smooth load/renewable generation changes. For that purpose, a Static State Estimator (SSE) that processes measurement snapshots is used. In SSE, conventional measurements that are acquired by the remote terminal units in SCADA system are sufficient. Unlike, dynamic state estimators, synchrophasor measurements (like these acquired by phasor measurement units) are not necessary for SSE. In general, system dynamics are not tracked by SSE. It has no memory of the states in the previous time steps. It requires that all states are observable given the measurement vector. In this chapter, we focus on SSE.

Weighted least-squares (WLS) is a commonly used criterion for SSE. Consider a system of N buses, the number of unknown states n is equal to $2N - 1$. This includes all voltage magnitudes and phase angles excluding the reference bus whose phase angle is typically set to zero. A non-linear measurement model is described in [80], [54] and [53] as per equation 4.1.

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) + \mathbf{e} \tag{4.1}$$

where: $\mathbf{y}^T = [y_1, y_2, \ldots, y_m]$ is the vector of *observed* set of measurements.
$\mathbf{x}^T = [x_1, x_2, \ldots, x_n]$ is the vector of *true* system states which we need to estimate.
$\mathbf{h}^T(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_m(\mathbf{x})]$ is the *measurement function* where $h_i(\mathbf{x})$ is a scalar function relating the *ith* measurement y_i to all real states $\mathbf{x}$.
$\mathbf{e}^T = [e_1, e_2, \ldots, e_m]$ is the vector of measurement errors.
The error given in 4.1 is usually weighted to reflect the accuracy associated with each measuing instrument. WLS provides an estimate of $\mathbf{x}$, namely $\hat{\mathbf{x}}$, by minimizing the sum of weighted error given below:

$$J(\mathbf{x}) = \sum_{i=1}^{m} w_i e_i^2 \tag{4.2}$$

where w_i is weight associated with the *ith* measurement. The weight is typically taken to be the inverse of the variance that reflect the accuracy of the corresponding measuring instrument i.e. $w_i = 1/\sigma_i^2$. Further, the following assumptions are commonly made regarding the error:

- Any measurement error has an expected value of zero, i.e. $\mathrm{E}(e_i) = 0, \quad i = 1, \ldots, m$
- Measurement errors are independent, i.e. $\mathrm{E}\left[e_i e_j\right] = 0$ for $i \neq j$.

Defining the measurement error covariance matrix as $R = \mathrm{E}[\mathbf{e}.\mathbf{e}^T]$, then based on the above assumptions, $R = \mathrm{diag}\left\{\sigma_1^2, \sigma_2^2, \cdots, \sigma_m^2\right\}$. Accordingly, we can write the objective function (4.2) in matrix form:

$$J(\mathbf{x}) = [\mathbf{y} - \mathbf{h}(\mathbf{x})]^T R^{-1} [\mathbf{y} - \mathbf{h}(\mathbf{x})] \tag{4.3}$$

Applying the first-order optimality condition, the iterative solution based on Gauss-Newton method is given as [80], [103]:

$$\left[G\left(\mathbf{x}^k\right)\right] \Delta \mathbf{x}^k = H^T\left(\mathbf{x}^k\right) R^{-1} \left[\mathbf{y} - \mathbf{h}\left(\mathbf{x}^k\right)\right] \tag{4.4}$$

where:

- k is the iteration index

- $\mathbf{x}^k$ is the solution vector at iteration k
- $\Delta\mathbf{x}^k = \mathbf{x}^{k+1} - \mathbf{x}^k$
- $H(\mathbf{x}) = \left[\frac{\partial\mathbf{h}(\mathbf{x})}{\partial\mathbf{x}}\right]$ which is $m \times n$ matrix called the measurement Jacobian.
- $G\left(\mathbf{x}^k\right) = H^T\left(\mathbf{x}^k\right)R^{-1}H\left(\mathbf{x}^k\right)$ which is $n \times n$ sparse matrix called the gain matrix.

Based on the above, we note the following:

- Network topology and parameters need to be known in order to calculate the measurement function $\mathbf{h}(\mathbf{x})$. First, various components (e.g. transmission lines, transformers...etc.) of the network are properly modelled in similar fashion to power flow analysis. Then power flow equations that relate all states with available observed measurements are obtained.
- It is necessary to calculate the measurement Jacobian and gain matrices for every iteration. This increases computational complexity and estimation time.
- In practice, not all measurements are obtained simultaneously, so for real-time estimation, the availability of time-tagged measurements is necessary[53].
- Convergence of Gauss-Newton algorithm is not guaranteed. Ill-conditioning may occur for different reasons such as widely different weighting factors and others [103], [53].
- Assumptions are made about the statistical distribution of error.

A DC approximation of the system described in eq. 4.1 is obtained by assuming that all voltage magnitudes are known to be equal to 1 per unit and neglecting all shunt elements and branch resistances. Consider N bus system, the DC approximated model is expressed as:

$$\mathbf{y} = H\mathbf{x} + \mathbf{e} \tag{4.5}$$

where:

- $\mathbf{y} \in \mathcal{R}^m$ is the vector of observed measurements and m is the number of measurements. .
- H is $m \times N$ measurement matrix relating states to measurements.
- $\mathbf{x} \in \mathcal{R}^N$ is the vector of bus phase angles. One of the buses is typically considered as a reference with its phase angle (i.e. state) set to 0.
- $\mathbf{e} \in \mathcal{R}^m$ is the vector of measurement errors.

Let bus indices $i, j \in \aleph = \{1, 2, \ldots, N\}$, then a branch ij may represent a transmission line or transformer with branch reactance X_{ij} in per unit.

In DC approximation, observed measurements are restricted to selected real power flows and power injections. Power flow from bus i to bus j is approximated by [80]:

$$y_{ij} = \frac{1}{X_{ij}}[x_i - x_j] + e \tag{4.6}$$

where x_i is phase angle at bus i and x_j is phase angle at bus j while e is the measurement error associated with the measured power injection y_{ij}. Further, power injection at bus i is the sum of power flows in the set of branches connected to that bus and denoted as $\aleph_i \subseteq \aleph$. Hence, the measured power injection at i can be written as:

$$y_i = \sum_{j \in \aleph_i} \frac{1}{X_{ij}}[x_i - x_j] + e \tag{4.7}$$

Note that e in eq. 4.7 is the error associated with measuring the power injection which is not the same as the error associated with measuring the power flow given in eq. 4.6. The measurement matrix H can be calculated form both equations 4.6 and 4.7.

A closed form solution of equation for equation 4.5 can be expressed as:

$$\hat{\mathbf{x}} = G^{-1} H^T R^{-1} \mathbf{y} \tag{4.8}$$

where $G = H^T R^{-1} H$ and R is defined earlier.

4.1.2 Bad data detection

Bad data detection (BDD) is an important aspect of any SSE. BDD refers to the determination of the measurement vector that includes any bad data. When bad data is detected, a flag is raised. On the other hand, bad or tampered data identification (TDI) refers to the procedure used to find the specific measurements that were compromised.

Chi-squares test is used for BDD. This arises from the assumption that measurement errors are normally distributed. Hence, from equation 4.2, $J(\mathbf{x})$ will have χ^2 distribution with at most $(m - n)$ degrees of freedom where m is the number of measurements and n is the number of states. Hence, a procedure to detect data would follow the following steps [80]:

- Once states are estimated solve for J in 4.2
- Consider certain detection confidence (e.g. 95%), look up the value from the Chi-squares distribution with $(m - n)$ degrees of freedom.

- If the value of J is larger than the value obtained from the Chi-squares distribution, then bad data is detected and a flag is raised.

A more accurate procedure to detect bad data is by calculating the *normalized residuals*. Recall that the measurement errors are independent and normally distributed, i.e. $\mathbf{e} \sim \mathcal{N}(0, R)$ where R is a diagonal measurement error covariance matrix. Consider the linear formulation, the estimated measurement values $\tilde{\mathbf{y}}$ can be found by multiplying the estimated states by the measurement matrix as per equation 4.10 below.

$$\tilde{\mathbf{y}} = H\hat{\mathbf{x}} \tag{4.9}$$

Substituting equation 4.8 in equation 4.10, we obtain:

$$\tilde{\mathbf{y}} = K\mathbf{y} \tag{4.10}$$

where $K = HG^{-1}H^TR^{-1}$. Th measurement residuals are defined as:

$$\mathbf{r} = \mathbf{y} - \tilde{\mathbf{y}} = (I - K)\mathbf{y} \tag{4.11}$$

It is proven that the matrix K has the property that $(I - K).H = 0$ [80], hence from equations 4.11 and 4.5, we obtain:

$$\mathbf{r} = (I - K)\mathbf{e} = S\,\mathbf{e} \tag{4.12}$$

S is called the *residual sensitivity matrix* and it not symmetric unless all diagonal entries of R are equal. Further, S has the property that $SRS^T = SR$ [80]. Knowing that $\mathbf{e} \sim \mathcal{N}(0, R)$ and from equation 4.12, we can find the mean and covariance of residuals as:

$$\mathrm{E}(\mathbf{r}) = S.\mathrm{E}(\mathbf{e}) = 0 \tag{4.13}$$

$$\begin{aligned} \mathrm{Cov}(\mathbf{r}) = \Omega &= \mathrm{E}\left[\mathbf{r}\mathbf{r}^T\right] \\ &= S \cdot \mathrm{E}\left[\mathbf{e}\mathbf{e}^T\right] \cdot S^T = SRS^T = SR \end{aligned} \tag{4.14}$$

The *residual covariance matrix* Ω is very important. The normalized residual for measurement i can be calculated by dividing the absolute value of the measurement's residual by the corresponding diagonal entry of the residual covariance matrix. i.e. $r_i^N = |r_i|/\sqrt{\Omega_{ii}}$. The measurement with the largest r_i^N is simply compared against a threshold to decide the presence of any bad data. If the normalized residual is larger than the threshold, the corresponding measurement is identified as being compromised and eliminated. State estimation is repeated, normalized residuals are calculated for all measurements, the tampered measurement with the largest normalized residual exceeding the threshold is identified and eliminated and the process is repeated until no measurement has a normalized residual larger than the threshold. This process achieves both BDD and TDI.

4.1.3 Cycle GAN

The concept of conditional GAN discussed in 2.1.2 is further extended by [104] to image-to-image translation where the GAN is conditioned on input image rather than a mere label. A naive alternative to accomplish the translation would be by minimizing a certain measure of distance between prediction and ground truth. However, finding the loss function that satisfies the desired output requires expert knowledge. For instance, generating images by minimizing the Euclidean distance results in blurred images. Instead, a min-max game using GAN allows learning the appropriate loss function to satisfy the goal. In such cases, blurred images will not be tolerated as they will be easily identified by the discriminator as fake images [104]. A typical application includes converting a specific photo into paint, or a B&W image to a coloured one.

One limitation of image-to-image translation is the necessity to have paired training samples. For instance, if photos need to be translated into paints of Claude Monet, then for every single photo in the training dataset a corresponding paint by Monet has to be produced. Clearly, this is impossible. Cycle GAN, proposed by [105], provides a solution for that by allowing training using *unpaired* sets of samples.

The cycle GAN includes two generators and two discriminators and incorporates cycle consistency loss in addition to the adversarial loss usually incorporated in GANs. Mapping is accomplished in both forward and reverse directions. In figure 4.1, there are two domains: x and y. For instance, domain y consists of random photos of nature and domain x consists of random paints by Claude Monet. The cycle GAN is trained to covert any new photo into paint of Monet. y^{real} consists of a set of real photos used for training while x^{real} consists of a set of real paints of Monet. The provided training samples of photos and paints are not paired. The *forward* generator G is trained to convert photos to paints while the backward generator F learns to convert paints to photos. In other words $G : y^{real} \to x^{fake}$ and $F : x^{real} \to y^{fake}$. The generator G is trained by playing a min-max game with its adversary D_x while the generator F plays the min-max game with D_y. This follows the same logic discussed earlier in 2.1.2. To allow for *unpaired* training samples, the concept of cycling is introduced. In this case, the synthetic (i.e. fake) samples generated by G are cycled by F and the synthetic samples generated by G are cycled by G. In other words, $F : x^{fake} \to y^{cycled}$ and $G : y^{fake} \to x^{cycled}$.

Accordingly, we define the following objective functions for cycle GAN.

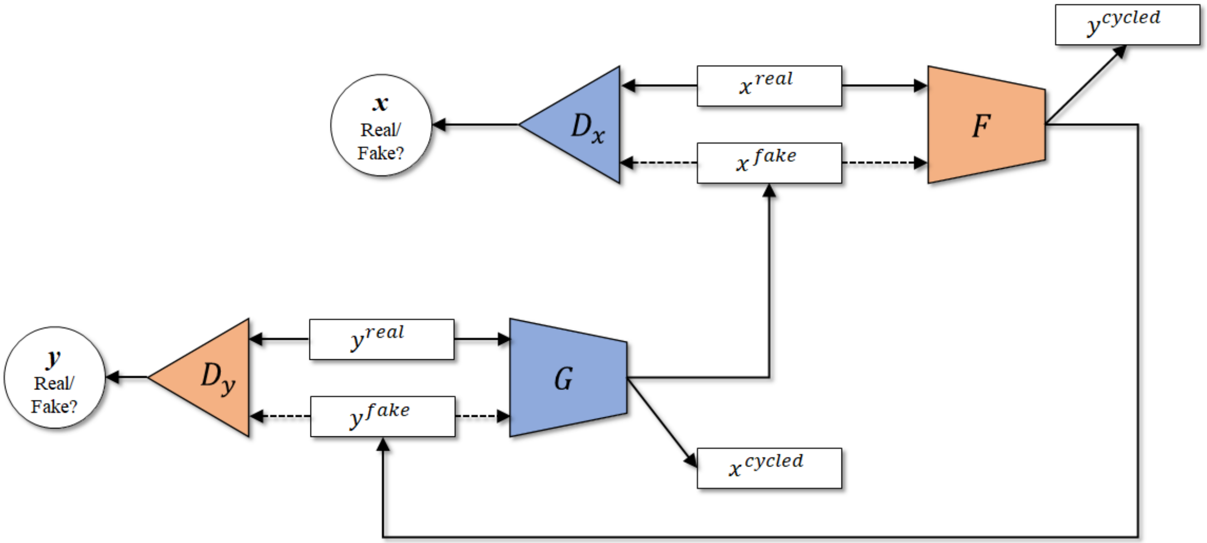


Figure 4.1: Main blocks of a cycle GAN.

Adversarial loss between state generator G and state discriminator D_X:

$$\min_{G}\max_{D_X}\mathcal{L}_{\text{GAN}}\left(G, D_X, Y, X\right) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D_X(x)\right] \quad +\mathbb{E}_{y\sim p_{\text{data}}(y)}\left[\log\left(1 - D_X(G(y))\right]\right. \tag{4.15}$$

Adversarial loss between measurement generator F and measurement discriminator D_Y:

$$\min_{F}\max_{D_Y}\mathcal{L}_{\text{GAN}}\left(F, D_Y, X, Y\right) = \mathbb{E}_{y\sim p_{\text{data}}(y)}\left[\log D_Y(y)\right] \quad +\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log\left(1 - D_Y(F(x))\right]\right. \tag{4.16}$$

Cycle-consistency loss for both G and F:

$$\min_{G,F}\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{y\sim p_{\text{data}}(y)}\left[\|F(G(y)) - y\|_1\right] \quad +\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\|G(F(x)) - x\|_1\right] \tag{4.17}$$

4.1.4 Embedding

One of the most common techniques to deal with a categorical variable is to encode it using one-hot-encoding, which is a unit vector representation of labels or classes as explained earlier in section 2.3. For example, for a set of three classes, one class is represented by the unit vector [0 0 1], while the other class is represented by [0 1 0] and so forth. When the number of classes becomes large, one-hot-encoding causes a substantial increase in data dimensionality and becomes infeasible. Another drawback of one-hot-encoding is the necessity to know all classes a priori, so no new classes are allowed to appear in the test set [106]. Embedding is used to overcome these problems. Embedding is widely used in the domain of natural language processing (NLP).

In embedding, each class is mapped to a distinct vector. The vector's components are learnt by the neural network. By minimizing the loss, the learning process maps the space of classes into a vector space that reflects similarity among classes. For instance, in NLP, consider a sequence of s words that are drawn from a repository of w words. Each word needs to be mapped to a vector of length v. In this case, the input of the embedding layer is the number of classes w, the dimension of the target vector representation v and the input length s. The output of the embedding layer is a matrix of dimension $s \times v$. Vector elements are learnt during the training process.

4.2 Problem definition

Consider $(n+1)/2$ bus system, we use the traditional estimator given in equation 4.4 to estimate states $\hat{\mathbf{x}} \in \mathcal{R}^n$ from the observed set of measurements $\mathbf{y} \in \mathcal{R}^m$ where $m > n$. The iteration index k is set to 0 and the vector $\mathbf{x}^k$ is initialized, then at each subsequent and until convergence, the following needs to be calculated:

- The measurement function $\mathbf{h}((x)^k)$ This is an m number of functions with n independent variables each.
- Estimate the $m \times n$ Jacobian $H(\mathbf{x}) = \left[\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\right]$
- Estimate the $n \times n$ gain matrix $G\left(\mathbf{x}^k\right) = H^T\left(\mathbf{x}^k\right) R^{-1} H\left(\mathbf{x}^k\right)$. Typically, decomposition techniques are used for that purpose [80].

The set of measurement functions are dictated by the set of corresponding power equations. The parameters of the power grid shall be known in order to derive the measurement functions and the measurement Jacobian. The calculation of matrices at every iteration is computationally expensive. Typically, the G matrix is less sparse than the bus admittance matrix and as such power estimation problem has higher computational complexity than power flow analysis. The gain matrix may even become singular in some cases e.g. in the case of a large proportion of injection measurements, large weighting factors and others [80]. Since power state estimation shall be carried out online for monitoring purposes, several techniques are proposed to reduce the computational complexity. Given the high computational complexity involved in traditional state estimators, we propose a data-driven framework to shift the burden of iteration and optimization to offline training. Once trained, online state estimation becomes a one-step matrix multiplication process. However, data-driven models need historical datasets for training and testing. Accordingly, we define our problem as follows:

- Obtain dataset for training and testing. The dataset shall include snapshots (i.e. load scenarios) of historical measurements and historical states. Measurements and states are not necessarily paired. In other words, for each set of observed measurements at a given load scenario i, the obtained set of states belong to the same or different load scenario j. For experimental purposes, we generate these load scenarios along with the associated measurements and states using load flow calculations at various load settings.

- Develop a data-driven generative framework that, once trained, will estimate states given the current set of observed measurements. The system is assumed to be fully observable. Observability is beyond the scope of this research.

- Devise a method to evaluate the estimation accuracy of the proposed framework.

- The developed framework shall be capable of detecting bad measurements and identifying the specific tampered measurements.

The hyperparameters of the developed framework shall be optimized for the experimented test case. Besides, any necessary adjustment of the vanilla objective functions shall be addressed.

4.3 Proposed framework

4.3.1 Vanilla cycle GAN

Inspired by the work done by [105] related to image processing, we apply vanilla GAN to the state estimation problem. Figure 4.2 shows the main blocks of the proposed cycle GAN. Once trained, the forward generator neural network G is used to estimate states from the observed measurements at any time instance. The backward generator neural network (F) is used to generate cycled measurements from the estimated states. The cycled measurements are utilized to identify tampered measurements as will be discussed later. Both discriminators (D_x) and (D_y) are neural networks that are necessary for the training process. Define:

- $\mathbf{x}_i$: Sample vector of historical real states recorded at arbitrary load scenario i where $\mathbf{x}_i \in \mathcal{R}^n$ and n is the number of states.

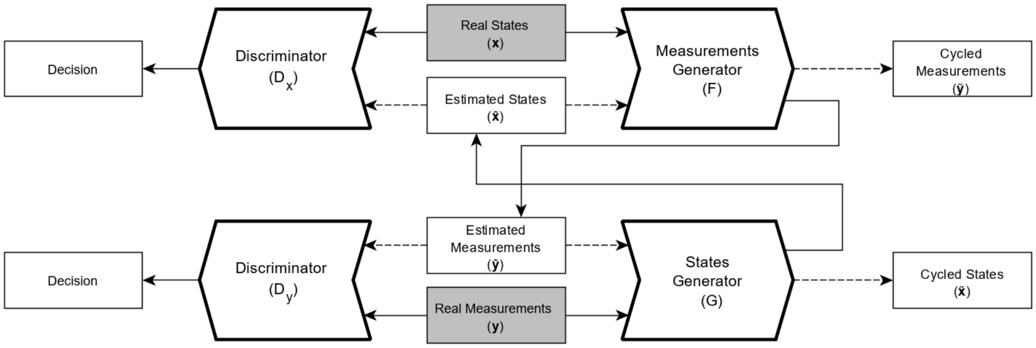


Figure 4.2: Cycle GAN setup for state estimation. Load scenario indices *i* and *j* are omitted for clarity.

- $\mathbf{y}_j$: Sample vector of historical real measurements observed at arbitrary load scenario j where $\mathbf{y}_j \in \mathcal{R}^m$ and m is the number of measurements.
- G: State estimator such that $G : \mathcal{R}^m \to \mathcal{R}^n$.
- $\hat{\mathbf{x}}_j$: Estimated states vector where $\hat{\mathbf{x}}_j = G(\mathbf{y}_j)$.
- F: Measurement estimator such that $F : \mathcal{R}^n \to \mathcal{R}^m$.
- $\hat{\mathbf{y}}_i$: Estimated measurements vector where $\hat{\mathbf{y}}_i = F(\mathbf{x}_i)$.
- D_x: Discriminator that aims to distinguish *any* real states sample $\mathbf{x}$ from *any* estimated states sample $\hat{\mathbf{x}}$. D_x plays a min-max game with G such that $D_x : \mathcal{R}^n \to \mathcal{R}$. The scalar output of D_x represents a probability.
- D_y: Discriminator that aims to distinguish *any* real measurements sample $\mathbf{y}$ from *any* estimated measurement sample $\hat{\mathbf{y}}$. D_y plays a min-max game with F such that $D_y : \mathcal{R}^m \to \mathcal{R}$. The scalar output of D_y represents a probability.
- $\tilde{\mathbf{x}}_i$: Cycled states vector where $\tilde{\mathbf{x}}_i = G(\hat{\mathbf{y}}_i)$.
- $\tilde{\mathbf{y}}_j$: Cycled measurements vector where $\tilde{\mathbf{y}}_j = F(\hat{\mathbf{x}}_j)$.

In the above model, observe the following:

- A states sample $\mathbf{x}_i$ and a measurements sample $\mathbf{y}_j$ are not necessarily paired i.e. i is not necessarily equal to j. This eliminates the need to obtain time-tagged historical readings for training purposes. However, historical readings should reflect the healthy operation of the power grid.

- Both discriminators need to distinguish whether their input samples are real or fake (i.e. estimated) regardless of the load scenario index.
- It is always true that the load scenario index of a cycled sample matches that of the input sample. This allows for defining an additional loss function, namely cycle consistency loss.

Below, we fully define the proposed training scheme by first defining the necessary losses, followed by the objective function and finally we define the overall optimization problem that is used in training the neural networks shown in the proposed framework (figure 4.2).

– Adversarial Loss

$$\begin{aligned}\mathcal{L}_{\mathrm{GAN}}(G, D_x) &= \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\log D_x(\mathbf{x})\right] \\ &+ \mathbb{E}_{\mathbf{y}\sim p_{\mathrm{data}}(\mathbf{y})}\left[\log\left(1 - D_x(G(\mathbf{y}))\right]\end{aligned} \tag{4.18}$$

$$\begin{aligned}\mathcal{L}_{\mathrm{GAN}}(F, D_y) &= \mathbb{E}_{\mathbf{y}\sim p_{\mathrm{data}}(\mathbf{y})}\left[\log D_y(\mathbf{y})\right] \\ &+ \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[\log\left(1 - D_y(F(\mathbf{x}))\right]\end{aligned} \tag{4.19}$$

For measurement distribution $\mathbf{y} \sim p_{\mathrm{data}}(\mathbf{y})$, G tries to estimate states $G(\mathbf{y})$ that are similar to real states $\mathbf{x} \sim p_{\mathrm{data}}(\mathbf{x})$, while D_x tries to distinguish between real states $\mathbf{x}$ and estimated states $G(\mathbf{y})$. Hence, G is optimized to minimize adversarial loss while D_x is optimized to maximize that loss, i.e. the distinction between real and estimated states. The same rationale applies to F and its adversary D_y.

– Cycle Consistency Loss

$$\begin{aligned}\mathcal{L}_{\mathrm{cyc}}(G, F) &= \mathbb{E}_{\mathbf{y}\sim p_{\mathrm{data}}(\mathbf{y})}\left[||F(G(\mathbf{y})) - \mathbf{y}||_1\right] \\ &+ \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}(\mathbf{x})}\left[||G(F(\mathbf{x})) - \mathbf{x}||_1\right]\end{aligned} \tag{4.20}$$

While adversarial loss learns back and forth mapping functions between measurements and states, there is no guarantee that a set of measurements $\mathbf{y}_i$ at a certain load scenario i will map to the corresponding states $\mathbf{x}_i$. This is ensured by introducing the cycle consistency loss for both forward and backward mapping functions G and F. G is optimized by minimizing the distance between real measurements $\mathbf{y}$ and their cycled version $F(G(\mathbf{y}))$. The same rationale applies to F.

– Objective Function

$$\begin{aligned}\mathcal{L}\left(G, F, D_x, D_y\right) &= \mathcal{L}_{\mathrm{GAN}}(G, D_x) + \mathcal{L}_{\mathrm{GAN}}(F, D_y) \\ &+ \lambda\mathcal{L}_{\mathrm{cyc}}(G, F)\end{aligned} \tag{4.21}$$

where λ is a hyperparameter that is used to control the relative importance of each loss.

– Optimization Problem

$$\min_{G,F}\max_{D_x,D_y}\mathcal{L}\Big(G,F,D_x,D_y\Big) \tag{4.22}$$

4.3.2 Grid-aware cycle GAN

Typically, the parameters of the power grid (e.g. reactances of transmission lines..etc.) are readily available. Hence, we propose and test a novel grid-aware cycle GAN to estimate state and identify bad measurements. Recall that a DC approximation of the system as described in eq. 4.5 is obtained by assuming that all voltage magnitudes are known to be equal to 1 per unit and neglecting all shunt elements and branch resistances. To stress that the measurement matrix H is associated with DC formulation, we will denote H in equation 4.5 as H_{DC}.

Equations 4.6 and 4.7 can be used to easily construct H_{DC}. For instance, if the k^{th} measurement corresponds to a power flow measurement in a transmission line extending from the i^{th} bus to the j^{th} bus and with a reactance of X_{ij}, the H_{DC} entries are updated as follows: $H_{DC}(k,i) \leftarrow 1/X_{ij}$ and $H_{DC}(k,j) \leftarrow -1/X_{ij}$. If the i^{th} measurement corresponds to a power injection, then i^{th} row, $H_{DC}(i,:)$, is updated with the sum of all reactances associated with the power flows into that bus, i.e. $H_{DC}(i,:) \leftarrow -\sum_{j\in\aleph_i} H_{DC}(j,:)$ where $\aleph_i$ is the set of branches connected to the i^{th} bus (considering a convention of positive power flowing into the bus).

Once the H_{DC} matrix is constructed, we drop the column that corresponds to its reference bus. We use H_{DC} in the cycle GAN by defining additional loss function that captures the physical relationship between the states and measurements as follows:

$$\mathcal{L}_{\mathrm{H_{DC}}}(G) = \mathbb{E}_{\mathbf{y}\sim p_{\mathrm{data}}(\mathbf{y})}\Big[\|H_{DC}^T\cdot\mathbf{y} - G_{angles}(\mathbf{y})\|_1\Big] \tag{4.23}$$

where G_{angles} is the output of G after dropping all states corresponding to voltage magnitudes. Clearly, G is optimized to minimize the distance given in 4.23.

In theory, since H_{DC} is a tall rectangular matrix, getting an estimation of $\mathbf{x}$ typically requires the computation of the pseudo-inverse of H_{DC} which is $(H_{DC}^T H_{DC})^{-1}H_{DC}^T$. However, if $H_{DC}^T H_{DC}$ is ill-conditioned, then this inverse cannot be computed and this is the case in practice. To avoid this issue, the transpose operator is used instead (i.e. $\mathbf{x} \approx H_{DC}^T\mathbf{y}$). Rationale based on singular value decomposition (SVD) is provided in the following to justify why the transpose operation captures important attributes pertaining to the pseudo-inverse. Suppose that the SVD of H_{DC} is:

$$H_{DC} = U\Sigma V^T \tag{4.24}$$

where $U \in \mathcal{R}^{m\times m}$ and $V \in \mathcal{R}^{n\times n}$ are orthogonal matrices that are the left and right singular vectors of H_{DC} and Σ is the diagonal matrix composed of the singular values of H_{DC}. The linear mapping H_{DC} can be interpreted as a series of three transformations where the first involves a rotation/ reflection defined by U, then scaling with Σ and then finally a rotation/ reflection based on V^T. It is important to note that the transpose of an orthogonal matrix is the same as the inverse of the matrix. For the transpose of H_{DC}, the SVD is now $V\Sigma^T U^T$ where the first transformation is the inverse rotation/reflection defined by V (this is the inverse of the last rotation/reflection of the forward mapping). Then, the scaling operation defined by Σ^T follows. The final rotation/reflection U^T is the inverse of the first rotation/reflection of the forward mapping. It is evident now that the transpose operation preserves directional attributes of the inverse mapping from grid measurements to states as the transpose and inverse operations differ by the scaling element Σ.

Hence, our choice of the grid-aware loss component seeks to preserve the directional attributes pertaining to the mapping from grid measurements to states. As the transpose operator is used, it is not affected by ill-conditioning. Furthermore, H_{DC} matrix is simple to compute and needs to be computed only once. The proposed machine learning model will learn the remaining non-linearities that are not captured by H_{DC} via the adversarial and cycle consistency loss components.

The objective function given in 4.21 is replaced by:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_X)\\ +\mathcal{L}_{\text{GAN}}(F, D_Y) + \lambda\mathcal{L}_{\text{cyc}}(G, F) + \mathcal{L}_{H_{DC}}(G)\end{aligned} \tag{4.25}$$

4.3.3 Evaluating cycle GAN for state estimation

We propose a benchmark multi-layer perceptron (MLP) neural network (figure 4.3) to evaluate the performance of the cycle GAN as a state estimator. The MLP is trained to map the input measurements to the corresponding outputs states. Since measurements dataset represent different load scenarios, it is necessary to identify measurements and states in pairs, otherwise, an input measurement vector will be mapped to a state vector that is associated with another load scenario.

The unique ID that identifies each load scenario is a categorical variable. We simulate a large number of load scenarios. Further, some of the simulated load scenarios in the test set may not be encountered in the train set. Accordingly, instead of using one-hot-encoding, we use learned embedding where each class is mapped to a distinct vector as discussed in 4.1.4. The vector representation is further combined with its corresponding input measurements using point-wise multiplication. As such, it necessary that the vector representation have the same dimension as the measurement vector. The output of the MLP will be the complete states vector.

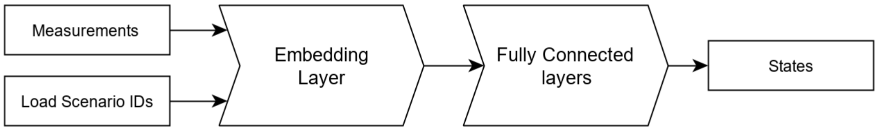


Figure 4.3: A benchmark MLP is used to evaluate the cycle GAN accuracy for estimating states. Unlike cycle GAN, labelled input data is used and predicted states are measured against the paired ground truth states.

Let the number of load scenarios be $T \in \mathcal{Z}$ and each load scenario is identified by a unique ID $t \in \mathcal{Z}$ and $\forall t : t \leq T$. The input of the benchmark MLP is both t and its corresponding vector of measurements $\mathbf{y}_t \in \mathcal{R}^m$. The benchmark MLP (figure 4.3) embeds t as vector $\mathbf{z}_t \in \mathcal{R}^m$. Then the element-wise product $\mathbf{z}_t \odot \mathbf{y}_t$ is passed to the subsequent fully connected layers. The output $\hat{\mathbf{x}}_t \in \mathcal{R}^n$ is the corresponding estimated states. Both m and n are defined earlier in 4.3.1. Let the respective ground truth states be: $\mathbf{x}_t \in \mathcal{R}^n$, the MLP is optimized by minimizing the mean squared-error $\mathbb{E}[\mathbf{x}_t - \hat{\mathbf{x}}_t]^2$.

4.3.4 Tampered data identification (TDI)

The proposed cycle GAN can be used for identifying tampered measurements. Figure 4.4 shows the cycle GAN in inference mode. For measurement input vector $\mathbf{y}_i$ corresponding to load scenario i, a pre-trained G estimates the state vector $\hat{\mathbf{x}}_i = G(\mathbf{y}_i)$. The estimated state vector is further processed by F to generate the corresponding cycled measurement vector $\tilde{\mathbf{y}}_i = F(\hat{\mathbf{x}}_i)$. The residual vector $\mathbf{r}_i =| \mathbf{y}_i - \tilde{\mathbf{y}}_i |$ is calculated. Any measurement in $\mathbf{r}_i$ is identified as tampered measurement if its residual exceeds a threshold. The value of the threshold decides the sensitivity of the detector.

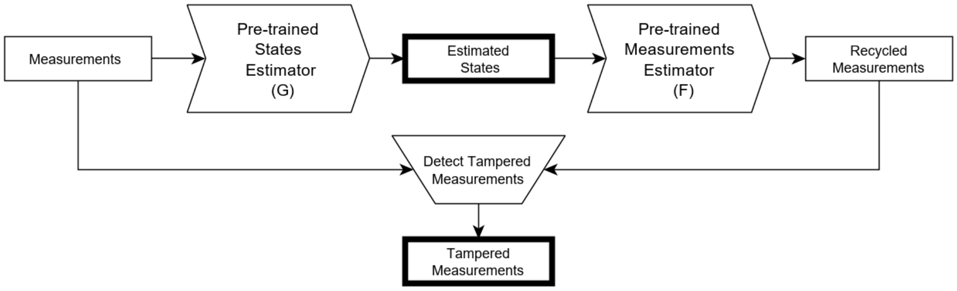

Measurements
Pre-trained States Estimator (G)
Estimated States
Pre-trained Measurements Estimator (F)
Recycled Measurements
Detect Tampered Measurements
Tampered Measurements

Figure 4.4: Proposed cycle GAN in inference mode used fr tampered data identification

4.4 Experimental studies

IEEE-118 case study [107], [108] is used to conduct the experiment. Pandapower [109] and Google's Tensorflow [110] are used to construct necessary modules. First, different load scenarios are generated. Power flow analysis is conducted and the resulting dataset is aggregated in the proper format. Second, cycle GAN is constructed and trained. Further, a benchmark MLP that maps measurements to states is constructed to evaluate the performance of the cycle GAN. Finally, scenarios with tampered measurements are generated and applied to the trained cycle GAN. The following subsections describe each module in detail.

4.4.1 Generating load scenarios

As illustrated in figure 4.5, loads in IEEE-118 case study are randomly varied according to the uniform distribution. A total of 6000 load scenarios are generated. Power flow analysis is performed for each load scenario to obtain the grid's electrical quantities such as complex power flows, power injections and voltages. A total of 759 measurements are selected at various locations to resemble observed measurements. Gaussian noise that represents the error in measuring instruments is added to each of the selected measurements ending up with a total of 120,000 samples of noisy measurement vectors. The measurements comprise voltage magnitudes, active and reactive power flows and power injections. The generated measurement-state dataset is used for training and testing the proposed cycle GAN estimator in addition to the benchmark MLP.

To compare computational complexity, WLS method is used to estimate the states for each load scenario using Pandapower's engine for power state estimation. For each measurement, the variance associated with the measuring instrument is fed to WLS algorithm. It is imperative that WLS assumes a Gaussian distribution for measurement error. The same variance is used when generating the noisy scenarios indicated above. We record both estimation time and the number of load scenarios that failed to converge using WLS.

Each load scenario has a unique ID (e.g. timestamp) which is used for training the benchmark MLP. Contrarily, the cycle GAN does *not* need a paired measurement-state dataset.

4.4.2 Training cycle GAN

The dataset generated in 4.4.1 includes both measurements and corresponding states. The dataset is divided into training and testing sets. The unique ID is *not* used as an input to the cycle GAN. However, we keep track of the unique ID for each training sample in order to calculate MSE against the ground-truth states. Measurements and states used for training are

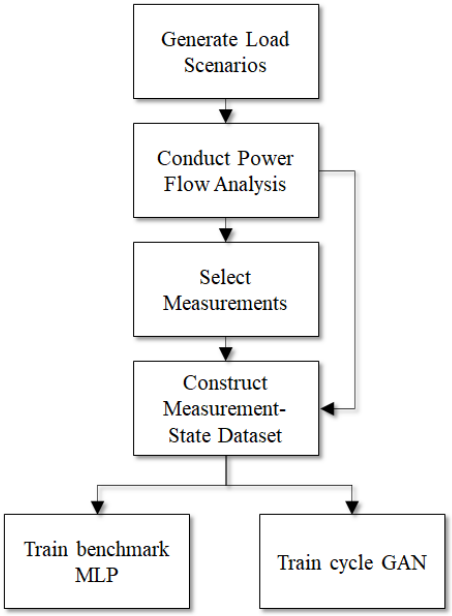


Figure 4.5: Constructing Training and Testing Datasets for cycle GAN

separately shuffled. Tables 4.1 shows the optimized architecture of the cycle GAN. The same architecture is used to test both vanilla and grids-aware loss functions as given in equations 4.21 and 4.25 respectively.

Figure 4.6 shows the adversarial losses for both forward and backward generators (G, F) and their corresponding discriminators (D_y, D_x) while training the vanilla cycle GAN with objective function given in eq. 4.21. Figure 4.7 shows the losses while training the grid-aware cycle GAN with objective function given in eq. 4.25.

4.4.3 Benchmarking

A benchmark MLP is constructed to evaluate the accuracy of the cycle GAN. The dataset generated in 4.4.1 is divided into train and testing datasets. Each unique ID representing a load scenario is encoded by mapping it to a vector of continuous numbers using embedding. The embedding shall have the same dimension of the measurement vector to allow for point-wise multiplication. Table 4.2 shows the optimized architecture of network.

For a labelled input of measurements, the MLP is optimized by minimizing the *mean squared-error* between the predicted states and the corresponding ground truth states that are identified by the ingested label. Figure 4.8 shows the progression of loss while training the benchmark MLP.

After training, the test set is ingested to all three networks, i.e. the benchmark MLP, the

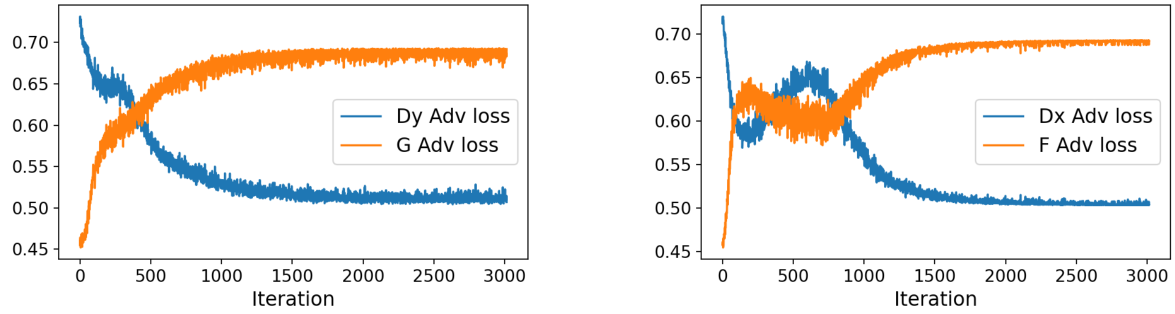
State Generator - G	
Nodes	Input: 759 L1:512, L2:1024, L3:2048, L4:1024, L5:512 Output: 235
Activation	relu, relu, relu, relu, relu, tanh
State Discriminator - D_x	
Nodes	Input: 235 L1:512, L2:1024, L3:256, L4:64 Output: 1
Activation	relu, relu, relu, relu, sigmoid
Measurement Generator - F	
Nodes	Input: 235 L1:512, L2:1024, L3:2048, L4:1024, L5:512 Output: 759
Activation	relu, relu, relu, relu, relu, tanh
Measurement Discriminator - D_y	
Nodes	Input: 759 L1:512, L2:1024, L3:256, L4:64 Output: 1
Activation	relu, relu, relu, relu, sigmoid

Table 4.1: Cycle GAN Architecture

<table>
<tr><td rowspan="6">Nodes</td><td>Input1
(load scenario unique ID): 1</td><td rowspan="2">Input2
(measurements): 759</td></tr>
<tr><td>L1 (Embedding):759</td></tr>
<tr><td colspan="2">L2 (pointwise multiplication): 759</td></tr>
<tr><td colspan="2">L3:512, L4:1024, L5:2048, L6:1024, L7:512</td></tr>
<tr><td colspan="2">Output: 235</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Activation</td><td colspan="2">relu, relu, relu, relu, relu, tanh</td></tr>
</table>

Table 4.2: Benchmark MLP Architecture

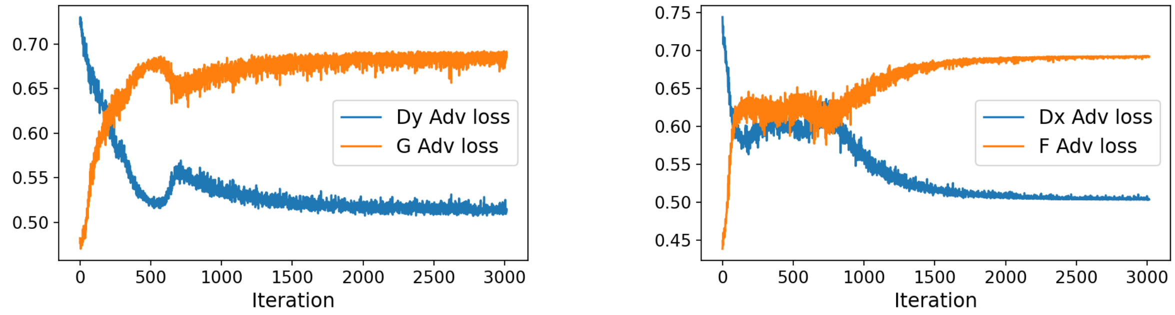
vanilla cycle GAN and the grid-aware cycle GAN and the mean squared error is calculated. Table 4.3 summarizes results. Clearly, the grid-aware cycle GAN exhibits remarkable accuracy in estimating the power states. Recall that the benchmark MLP needs a labelled dataset during



(a) Losses for estimating states.

(b) Losses for recycling measurements.

Figure 4.6: Adversarial losses for vanilla cycle GAN. The forward generator (G) and its adversary D_y are trained to estimate states, while the backward generator (F) and its adversary (D_x) are trained to cycle measurements.



(a) Losses for estimating states.

(b) Losses for recycling measurements.

Figure 4.7: Adversarial losses for grid-aware cycle GAN.

training. Embedding was used to label the 6000 scenarios. No labelled data is needed for training the cycle GAN. Further, MLP can not be used for BDD or TDI. All in all, using MLP as a state estimator is not practical and its function is restricted to benchmarking the estimation accuracy in the proposed cycle GAN.

Benchmark MLP	**Vanilla cycle GAN**	**Grid-aware cycle GAN**
0.00331	0.00550	0.00321

Table 4.3: Mean squared-error for various models where the MLP is used only for benchmarking purposes.

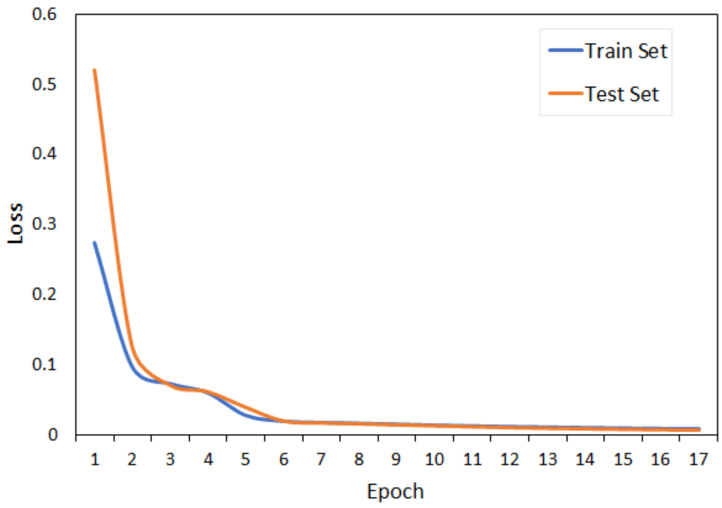


Figure 4.8: Loss of benchmark MLP for both train and test datasets

4.4.4 Identifying bad measurements

We define three attack scenarios and apply them to both vanilla and grid-aware cycle GANs. The attack scenarios are defined as follows:

- Attack 1: In this attack, only one measurement is tampered. The measurement of active power flow from bus 106 to bus 105 is tampered by increasing its authentic value by three times its authentic value.
- Attack 2: In this attack, three measurements are tampered. These are active power flow from bus 95 to bus 81, active power flow from bus 106 to bus 105 and active power flow from bus 53 to bus 48. The measurements are tampered by increasing their authentic values by 3, 15 and 0.01 times their authentic values respectively.
- Attack 3: In this attack, a linear combination of the columns of H_{DC} is constructed. This is known as false data injection (FDI) as stated by [57]. It is proven that FDI bypasses the traditional state estimator in its linear formulation.

We generate 200 random load scenarios with tampered measurements pertaining to each attack scenario. The generated tampered scenarios are processed by the optimized forward generator (G). The corresponding outputs (i.e. estimated states) are processed by the backward generator (F) to obtain the cycled measurements. The tampered measurements are then identified by the trained cycle GAN as explained in 4.3.4.

As explained earlier in 4.3.4, the residual for each measurement is calculated from both the tampered measurement input vector and the cycled measurement vector. This applies to any load scenario. All measurements with residuals exceeding a selected threshold are identified as tampered. The threshold impacts the identification sensitivity, so a high threshold may fail to capture tampered measurements, while a lower threshold may incorrectly classify an authentic measurement as being tampered.

Typically, any observed measurement may be either tampered (positive) or authentic (negative). If a tampered measurement is correctly identified by the cycle GAN as being tampered, it is counted as a true positive. If the tampered measurement is incorrectly identified as being authentic, it is counted as a false positive. This can be extended to the other classes ending up with four possibilities, i.e. true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Hence, the quality metrics discussed in 3.1.3 are used to evaluate the performance of the cycle GAN to identify bad measurements.

Table 4.4 summarizes the results for both vanilla and grid-aware cycle GANs against each attack scenario. Both cycle GANs achieve remarkable results when identifying tampered measurements. The results are very close. In terms of both accuracy and TDI, the grid-aware cycle GAN is a favourable choice. Note that, unlike traditional state estimators, the proposed framework can easily detect FDI. The second attack, where three measurements were tampered and one of them with a very small factor, is the most challenging scenario to be detected in terms of F1 score.

Attack Scenario	**Loss Type**	**Accuracy**	**F1**
1	Vanilla	0.9997	0.8270
2	Vanilla	0.9989	0.7964
3	Vanilla	0.9722	0.9636
1	Grid-aware	0.9997	0.8270
2	Grid-aware	0.9989	0.7976
3	Grid-aware	0.9719	0.9633

Table 4.4: Accuracy and F1 scores for various types of losses against different attacks.

As indicated earlier, the threshold plays an important role in TDI. If historical data about various attacks is available or such data can be obtained by simulation, a reasonable value of the threshold can be determined. For instance. for the grid-aware cycle GAN, we plotted the evaluation metrics against possible threshold values. The value that maximizes the metric of interest (e.g. F1) is selected (figure 4.9).

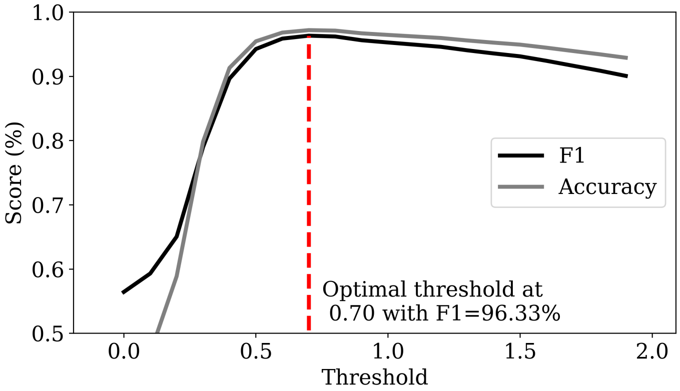


Figure 4.9: F1 and Accuracy scores against threshold in grid-aware cycle GAN.

4.5 Summary

Data-driven models are recently employed to tackle 'static power state estimation' and 'bad data detection' as two separate problems. In this chapter, we proposed a novel data-driven generative framework to handle both power state estimation and bad data identification. At the core of the proposed framework is the cycle GAN. Cycle GAN is a generative model the eliminates the need for paired training datasets. We improved the accuracy of the model by incorporating a grid-aware term in the objective function.

The experiment is designed by generating load scenarios using load flow analysis. The load scenarios are generated to reflect a normal operation of the power grid. A benchmark network is constructed to evaluate the estimation performance. The proposed framework recorded remarkable estimation accuracy.

Once the model is trained, measurements are deliberately tampered and ingested into the pre-trained network. Measurements are cycled and residuals are calculated to identify the tampered measurements. With a proper setting of a threshold, the proposed frameworks achieved high accuracy and F1-score.

Chapter 5

Conclusions

The electrical power grid has grown into a complex smart grid associated with digital technology, two-way communication, distributed generation and self-monitoring. It is even expected to grow further into a neural grid with ubiquitous connectivity accompanied by cloud-based artificial intelligence. With this high level of connectivity and complexity, physics-based models become inadequate. Probabilistic generative models stand as inevitable replacements of rule-based models. Yet, generative models are flexible enough to incorporate grid-aware objective functions. While discriminative models are limited to discriminating the target value (i.e. label) for a given observation, generative models can learn the underlying distribution of observations even if they are conditioned on targets (i.e. conditions). Generative models in our research included conditional GAN, kernel density estimator (KDE), hidden Markov model (HMM) and finally cycle GAN.

5.1 Summary

In this thesis, we introduced various types of generative models in both areas of power transmission and power demand. While simulating residential electrical loads, both conditional generative adversarial (GAN) networks and kernel density estimators (KDE) were used. Conditional GANs were used to learn the distribution associated with the power waveform of each experimented residential load. Instead of building four GANs for the four experimented loads, only one conditional GAN was used to learn the underlying distribution conditioned on the corresponding load. In other words, if a random variable x represents any load's waveform, then for each load l, the conditional GAN (CGAN-Patterns in figure 2.11) learnt the distribution $p(x|l)$. Once learnt, we managed to sample from that distribution to obtain synthetic patterns for a given load. Similarly, the synthetic habits were generated (using CGAN-habits in figure 2.11) after converting the timestamps associated with patterns to engineered features.

In addition, for each load, a kernel density estimator was construed. The performance of each model was evaluated using maximum mean discrepancy. In practice, an inverted form of the vanilla objective function was proposed to improve stability.

For load disaggregation (i.e. NILM), HMM's model was constructed by estimating the transition and emission distributions using the proposed hybrid training algorithm. In the algorithm, transition probabilities were estimated by simply counting the on-off transitions in the training sequence for each load. Emission distribution is approximated using k-means clustering for each load. Individual loads were combined into a factorial HMM model for the aggregate signal (i.e. smart meter's active power measurements). The aggregate signal is converted into a discrete sequence of labels based on the centroids produced by the hybrid algorithm. Viterbi algorithm is used to disaggregate the input sequence of labels into the combined states which are then converted to the individual states. The proposed NILM model may be easily adopted by electrical utilities and integrated into their demand-side programs that enable their customers to cut back on their energy costs. However, it is imperative that a larger training dataset will result in a better model. A larger dataset may include more loads with their real power consumption being recorded for longer periods of time.

For state estimation, a framework mainly based on cycle GAN is used to map the set of redundant measurements into the hidden states. The cycle GAN allowed for an unpaired training set of measurements and states. Different load scenarios representing the normal operation of the power grid are generated using load flow analysis. These scenarios are split into training and testing datasets. To improve accuracy, an additional grid-aware loss is introduced. Further, the cycle GAN successfully detected tampered measurements, thanks to the backward GAN (F in figure 4.2) that produced the cycled measurements. By comparing the observed measurements with the cycled measurements, individual residuals are calculated. If a residual exceeds the threshold, the respective measurement is considered tampered. To evaluate the estimation performance, a benchmark neural network is constructed and the mean squared error of the cycle GAN is compared with that of the benchmark network. Bad data identification is evaluated using F1 score besides accuracy.

In NILM problem, we used Hidden Markov Models. In the area of power transmission, we fully exploited cycle GAN to estimate power states and identify bad data.

5.2 Contributions

In each researched topic, several novel contributions are made. The contributions of chapter 2 are summarized as follows:

1. We developed a flexible framework to generate synthetic load patterns and consumers'

usage habits in a household. By utilizing a conditional generative adversarial network (GAN), we eliminated the need to build a model for each specific load. We open-sourced our code[1] for researchers and industry.

2. We utilized a signal processing technique, namely, matched filter, in pre-processing the available public time series and converting it to two sets of training examples representing patterns and habits.
3. We stabilized the training process by inverting the existing loss function that is used in vanilla GAN.
4. We conducted comparative studies for learning habits between two generative models, namely, conditional GAN and Kernel Density Estimators.

In chapter 3, we applied hidden Markov model (HMM) to the NILM problem. While HMM is a commonly used generative method, assumptions are typically made regarding the probability distribution of the observed sequence of power samples (i.e. emissions). We introduced a novel hybrid framework that utilizes k-means clustering to eliminate the assumptions made about the probability distribution of emissions.

The novel contributions of chapter 4 are summarized as follows:

1. We developed a semi-supervised generative framework using cycle GAN to estimate power states from a redundant set of measurements. Unlike other data-driven solutions, ours does not require paired datasets of measurements and states. The only requirement is to ensure that the training data is acquired during the grid's normal operation.
2. The developed framework naturally supports tampered data identification in one integrated package. This is a major contribution over existing discriminative models. Unlike existing solutions, our model eliminates the need for training using tampered data, which is usually rare and difficult to predict. Further, our model identifies the exact tampered measurement rather than just raising a flag that bad data is detected.
3. We optimized our model and introduced a novel grid-aware loss to enhance estimation accuracy.
4. We developed a benchmark neural network to evaluate the accuracy of the proposed framework. The benchmark neural network employs label embedding to identify a large number of load scenarios in the mapping process. The embedding concept is widely applied in the literature of natural language processing and is successfully applied to our application.

---

[1] Code is available at `https://github.com/skababji/ElecLoads`

5.3 Future works

For residential load simulation, our work can be extended as follows:

- Automate the process of extracting the load's template from the raw input time series.
- In addition to maximum mean discrepancy, incorporate further metrics to measure the distance between the real and learnt distributions. Evaluation of GANs remains a topic that is widely researched in various disciplines. While some measures are well-established in the field of image processing, further research is needed in other fields.
- The proposed framework allows for different sampling rates and, hence, additional public datasets may be used for training. The additional training datasets may include different types of loads as well as different levels of occupancy and usage habits. This improves the model's versatility and expands its usability.
- The modelled habits in the framework are functions of occupants' availabilities, proclivities (i.e. tendencies to operate a load) and loads' internal cycles. Both availability and proclivity are impacted by external factors such as the weather condition, the holiday season, the region..etc. Accordingly, the proposed model for simulating habits may be improved by adding more features. For instance, using the time stamp provided with the training dataset, the corresponding historical temperature may be fetched from service providers (e.g. `https://www.weatherstats.ca/`) and added as a feature to train habits.

For Non-intrusive load monitoring, our work can be extended as follows:

- In our proposed hybrid training model, conduct a comparative study if k-means clustering is replaced by other clustering techniques such as Gaussian mixture models.
- The Viterbi algorithm in the proposed framework has high computational complexity. This is greatly impacted by the size of the aggregate model matrices (figure 3.4). In algorithm 2, the first row of the emission matrix for each load is highly sparse. This can be researched further to reduce the overall disaggregation complexity.
- The model may be improved by incorporating real-time feedback from the occupants. For instance, when the model incorrectly identifies a load to be in the 'on' state, an occupant may send an error message and the model will adjust its parameters accordingly.

For state estimations, our research may be extended to several important areas. This can be summarized as follows:

- The proposed framework needs to be investigated for its applicability to the distribution network.
- The sensitivity of residuals to various types of errors and attacks needs to be thoroughly studied.
- The state estimation performance of the proposed framework needs to be thoroughly investigated while varying the number, types and location of the measuring instruments. The impact of observability needs to be researched and compared with traditional estimation methods.

Bibliography

[1] G. Fadaie, “The influence of classification on world view and epistemology,” in *I n SITE 2008: Informing Science+ IT Education Conference*, vol. 8. Citeseer, 2008.

[2] S. J. Gershman, “The generative adversarial brain,” *Frontiers in Artificial Intelligence*, vol. 2, p. 18, 2019.

[3] A. Gopstein, C. Nguyen, C. O’Fallon, N. Hastings, and D. Wollman, “Nist framework and roadmap for smart grid interoperability standards, release 4.0,” 2021-02-18 2021.

[4] S. E. Kababji and P. Srikantha, “A data-driven approach for generating synthetic load patterns and usage habits,” *IEEE Transactions on Smart Grid*, vol. 11, no. 6, pp. 4984–4995, 11 2020.

[5] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” 2016.

[6] E. Wallner, “Turning Design Mockups Into Code With Deep Learning,” *FloydHub Blog*, May 2020. [Online]. Available: https://blog.floydhub.com/turning-design-mockups-into-code-with-deep-learning

[7] G. Insights, “From Smart Grid to Neural Grid,” Jun 2021, [Online; accessed 27. Jun. 2021]. [Online]. Available: https://guidehouseinsights.com/reports/from-smart-grid-to-neural-grid

[8] A. Géron, “Hands-on machine learning with scikit-learn and tensorflow: Concepts,” *Tools, and Techniques to build intelligent systems*, 2017.

[9] S. Zainul, “Rule-Based V/s AI-Based: Approaches to Building AI - Applozic Blog,” *Applozic Blog*, Feb 2020, [Online; accessed 18. Aug. 2021]. [Online]. Available: https://www.applozic.com/blog/rule-based-v-s-ai-based-approaches-to-building-ai

[10] “Web API Reference | Spotify for Developers,” Jun 2021, [Online; accessed 28. Jun. 2021]. [Online]. Available: https://developer.spotify.com/documentation/web-api/reference/#endpoint-get-audio-features

[11] E. Hossain, I. Khan, F. Un-Noor, S. Sikander, and M. S. Sunny, “Application of big data and machine learning in smart grid, and associated security concerns: A review,” *IEEE Access*, vol. PP, pp. 1–1, 01 2019.

[12] D. Sidorov, F. Liu, and Y. Sun, “Machine learning for energy systems,” *Energies*, vol. 13, p. 4708, 09 2020.

[13] K. Günel and A. R. Ekti, “Exploiting machine learning applications for smart grids,” in *2019 16th International Multi-Conference on Systems, Signals Devices (SSD)*, 2019, pp. 679–685.

[14] A. Arif, Z. Wang, J. Wang, B. Mather, H. Bashualdo, and D. Zhao, “Load modeling—a review,” *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 5986–5999, May 2017.

[15] C. F. Walker, “A RESIDENTIAL ELECTRICAL LOAD MODEL,” Ph.D. dissertation, University of New Hampshire, 1982. [Online]. Available: https://scholars.unh.edu/dissertation/1350

[16] J. M. G. López, E. Pouresmaeil, C. A. Cañizares, K. Bhattacharya, A. Mosaddegh, and B. V. Solanki, “Smart Residential Load Simulator for Energy Management in Smart Grids,” *IEEE Trans. Ind. Electron.*, vol. 66, no. 2, pp. 1443–1452, Feb 2019.

[17] R. Stamminger, G. Broil, C. Pakula, H. Jungbecker, M. Braun, I. Rüdenauer, and C. Wendker, “Synergy potential of smart appliances,” *Report of the Smart-A project*, pp. 1949–3053, Nov 2008.

[18] W. Guo and T. Ullah, “Deployment of a load simulator in simulating residential household appliances,” in *2015 IEEE 12th International Conference on Networking, Sensing and Control*. IEEE, 2015, pp. 570–575.

[19] M. Trčka and J. L. Hensen, “Overview of hvac system simulation,” *Automation in Construction*, vol. 19, no. 2, pp. 93–99, 2010.

[20] J. Jeyakumar and D. Devaraj, “Load profile generation for dr program,” in *2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*. IEEE, 04 2019, pp. 1–5.

[21] M. Ouassaid, M. Maaroufi *et al.*, “Smart home appliances modeling and simulation for energy consumption profile development: Application to moroccan real environment case study,” in *2016 International Renewable and Sustainable Energy Conference (IRSEC)*. IEEE, 2016, pp. 1050–1055.

[22] G. Valverde, A. Saric, and V. Terzija, “Probabilistic load flow with non-gaussian correlated random variables using gaussian mixture models,” *IET generation, transmission & distribution*, vol. 6, no. 7, pp. 701–709, Jul 2012.

[23] W. Labeeuw and G. Deconinck, “Residential electrical load model based on mixture model clustering and markov models,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1561–1569, Jan 2013.

[24] A. Keyhani, W. Lu, and G. T. Heydt, “Composite neural network load models for power system stability analysis,” in *IEEE PES Power Systems Conference and Exposition, 2004*. IEEE, Oct 2004, pp. 1159–1163.

[25] A. Al-Wakeel, J. Wu, and N. Jenkins, “k-means based load estimation of domestic smart meter measurements,” *Applied Energy*, Jun 2016.

[26] K.-J. Park and S.-Y. Son, “A novel load image profile-based electricity load clustering methodology,” *IEEE Access*, vol. 7, pp. 59 048–59 058, May 2019.

[27] C. Zhang, S. R. Kuppannagari, R. Kannan, and V. K. Prasanna, “Generative adversarial network for synthetic time series data generation in smart grids,” in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2018, pp. 1–6.

[28] Y. Gu, Q. Chen, K. Liu, L. Xie, and C. Kang, “Gan-based model for residential load generation considering typical consumption patterns,” in *2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*. IEEE, 2019, pp. 1–5.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[30] D. Mwiti, “Introduction to Generative Adversarial Networks (GANs): Types, and Applications, and Implementation,” May 2019. [Online]. Available: https://heartbeat.fritz.ai/introduction-to-generative-adversarial-networks-gans-35ef44f21193

[31] K. Shmelkov, C. Schmid, and K. Alahari, “How good is my gan?” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 213–229.

[32] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, “A kernel two-sample test,” *Journal of Machine Learning Research*, vol. 13, no. Mar, pp. 723–773, 2012.

[33] M. E. Berges, E. Goldman, H. S. Matthews, and L. Soibelman, “Enhancing electricity audits in residential buildings with nonintrusive load monitoring,” *Jour. of Ind. Eco.*, vol. 14, no. 5, pp. 844–858, 2010.

[34] (2017, Dec) Datasets | NILM wiki. [Online]. [Online]. Available: http://wiki.nilm.eu/datasets.html

[35] J. R. Herrero, Á. L. Murciego, A. L. Barriuso, D. H. de la Iglesia, G. V. González, J. M. C. Rodríguez, and R. Carreira, “Non intrusive load monitoring (nilm): A state of the art,” in *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer, 2017, pp. 125–138.

[36] Z. Wang and G. Zheng, “Residential appliances identification and monitoring by a non-intrusive method,” *IEEE transactions on Smart Grid*, vol. 3, no. 1, pp. 80–92, 2012.

[37] T. Zia, D. Bruckner, and A. Zaidi, “A hidden markov model based procedure for identifying household electric loads,” in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2011, pp. 3218–3223.

[38] K. Basu, A. Hably, V. Debusschere, S. Bacha, G. J. Driven, and A. Ovalle, "A comparative study of low sampling non intrusive load dis-aggregation," in *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2016, pp. 5137–5142.

[39] S. Makonin, F. Popowich, I. V. Bajic, B. Gill, and L. Bartram, "Exploiting hmm sparsity to perform online real-time nonintrusive load monitoring," *IEEE Trans. on Sm. Grid*, vol. 7, no. 6, pp. 2575–2586, 2016.

[40] J. Z. Kolter and T. Jaakkola, "Unsupervised disaggregation of low frequency power measurements," *SIAM International Conference on Data Mining*, pp. 747–758, 2011.

[41] J. Cho, Z. Hu, and M. Sartipi, "Non-intrusive a/c load disaggregation using deep learning," 04 2018, pp. 1–5.

[42] L. Mauch and B. Yang, "A novel dnn-hmm-based approach for extracting single loads from aggregate power signals," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2384–2388.

[43] (2021, Jun) Redd. [Online; accessed 23. Jun. 2021]. [Online]. Available: http://redd.csail.mit.edu

[44] M. Z. A. Bhotto, S. Makonin, and I. V. Bajić, "Load disaggregation based on aided linear integer programming," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 7, pp. 792–796, 2017.

[45] J. Kelly and W. Knottenbelt, "Neural nilm: Deep neural networks applied to energy disaggregation," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. ACM, 2015, pp. 55–64.

[46] J. Kim, T.-T.-H. Le, and H. Kim, "Nonintrusive load monitoring based on advanced deep learning and novel signature," *Computational intelligence and neuroscience*, vol. 2017, 2017.

[47] J. Wang, S. El Kababji, C. Graham, and P. Srikantha, "Ensemble-based deep learning model for non-intrusive load monitoring," in *2019 IEEE Electrical Power and Energy Conference (EPEC)*, 2019, pp. 1–6.

[48] S. M. Tabatabaei, S. Dick, and W. Xu, "Toward non-intrusive load monitoring via multi-label classification," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 26–40, 2017.

[49] G. Wang, G. B. Giannakis, J. Chen, and J. Sun, "Distribution system state estimation: An overview of recent developments," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 1, pp. 4–17, Jan 2019.

[50] T. Dy Liacco, "The role of state estimation in power system operation," *IFAC Proceedings Volumes*, vol. 15, no. 4, pp. 1531–1533, 1982, 6th IFAC Symposium on Identification and System Parameter Estimation, Washington USA, 7-11 June. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017632166

[51] A. Abur and A. G. Exposito, *Power system state estimation: theory and implementation*. CRC press, 2004, ch. 3.

[52] R. E. Larson, W. F. Tinney, and J. Peschon, "State estimation in power systems part i: Theory and feasibility," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-89, no. 3, pp. 345–352, 1970.

[53] A. Monticelli, "Electric power system state estimation," *Proceedings of the IEEE*, vol. 88, no. 2, pp. 262–282, 03 2000.

[54] A. J. Wood, B. F. Wollenberg, and G. B. Sheblé, *Power generation, operation, and control*. John Wiley & Sons, 2013, ch. 12.

[55] F. C. Schweppe and D. B. Rom, "Power system static-state estimation, part ii: Approximate model," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-89, no. 1, pp. 125–130, 1970.

[56] D. Van Hertem, J. Verboomen, K. Purchala, R. Belmans, and W. L. Kling, "Usefulness of dc power flow for active power flow analysis with flow controlling devices," in *The 8th IEE International Conference on AC and DC Power Transmission*, 2006, pp. 58–62.

[57] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 1–33, 2011.

[58] O. Ivanov and M. Garvrilaş, "State estimation for power systems with multilayer perceptron neural networks," in *11th Symposium on Neural Network Applications in Electrical Engineering*. IEEE, 2012, pp. 243–246.

[59] O. Ivanov and M. Gavrilaş, "State estimation with neural networks and pmu voltage measurements," in *2014 International Conference and Exposition on Electrical and Power Engineering (EPE)*. IEEE, 2014, pp. 983–988.

[60] H. Mosbah and M. El-Hawary, "Multilayer artificial neural networks for real time power system state estimation," in *2015 IEEE Electrical Power and Energy Conference (EPEC)*, 10 2015, pp. 344–351.

[61] L. Zhang, G. Wang, and G. B. Giannakis, "Real-time power system state estimation and forecasting via deep unrolled neural networks," *IEEE Transactions on Signal Processing*, vol. 67, no. 15, pp. 4069–4077, Jul 2019.

[62] G. Wang, G. B. Giannakis, and J. Chen, "Robust and scalable power system state estimation via composite optimization," *IEEE Transactions on Smart Grid*, vol. 10, no. 6, pp. 6137–6147, 08 2019.

[63] L. Zhang, G. Wang, and G. B. Giannakis, "Distribution system state estimation via data-driven and physics-aware deep neural networks," in *2019 IEEE Data Science Workshop (DSW)*, 2019, pp. 258–262.

[64] A. S. Zamzam and N. D. Sidiropoulos, “Physics-aware neural networks for distribution system state estimation,” *IEEE Transactions on Power Systems*, vol. 35, no. 6, pp. 4347–4356, 2020.

[65] F. Aeiad, W. Gao, and J. Momoh, “Bad data detection for smart grid state estimation,” in *2016 North American Power Symposium (NAPS)*, 2016, pp. 1–6.

[66] M. Esmalifalak, L. Liu, N. Nguyen, R. Zheng, and Z. Han, “Detecting stealthy false data injection using machine learning in smart grid,” *IEEE Systems Journal*, vol. 11, no. 3, pp. 1644–1652, 2017.

[67] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. V. Poor, “Machine learning methods for attack detection in the smart grid,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 8, pp. 1773–1786, 2016.

[68] R. D. Trevizan, C. Ruben, K. Nagaraj, L. L. Ibukun, A. C. Starke, A. S. Bretas, J. McNair, and A. Zare, “Data-driven physics-based solution for false data injection diagnosis in smart grids,” in *2019 IEEE Power Energy Society General Meeting (PESGM)*, 2019, pp. 1–5.

[69] L. YAVUZ, A. SORAN, A. ÖNEN, and S. M. MUYEEN, “Machine learning algorithms against hacking attack and detection success comparison,” in *2020 2nd International Conference on Smart Power Internet Energy Systems (SPIES)*, 2020, pp. 258–262.

[70] Y. Zhang, J. Wang, and B. Chen, “Detecting false data injection attacks in smart grids: A semi-supervised deep learning approach,” *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 623–634, 2021.

[71] W. Qiu, Q. Tang, K. Zhu, W. Wang, Y. Liu, and W. Yao, “Detection of synchrophasor false data injection attack using feature interactive network,” *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 659–670, 2021.

[72] L. Wu, S. You, X. Zhang, Y. Cui, Y. Liu, and Y. Liu, “Statistical analysis of the fnet/grideye-detected inter-area oscillations in eastern interconnection (ei),” in *2017 IEEE Power Energy Society General Meeting*, 2017, pp. 1–5.

[73] T. Wu, W. Xue, H. Wang, C. Chung, G. Wang, J. Peng, and Q. Yang, “Extreme learning machine-based state reconstruction for automatic attack filtering in cyber physical power system,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1892–1904, 2020.

[74] “London hydro unveils trickl app,” CFRL, Mar 2018. [Online]. Available: https://www.cfrlradio.com/syn/202/71446/london-hydro-unveils-trickl-app/

[75] S. E. Kababji and P. Srikantha, “Power Appliance Disaggregation Framework Via Hybrid Hidden Markov Model,” *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, pp. 1–5, May 2018.

[76] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, “Nilmtk: an open source toolkit for non-intrusive load monitoring,” in *Proceedings of the 5th international conference on Future energy systems*. ACM, 2014, pp. 265–276.

[77] F. Nuha and A. Afiahayati, “Training dataset reduction on generative adversarial network,” *Procedia computer science*, vol. 144, pp. 133–139, 01 2018.

[78] C. Dilmegani, “The Ultimate Guide to Synthetic Data in 2021,” *AIMultiple*, Aug 2021, [Online; accessed 17. Aug. 2021]. [Online]. Available: https://research.aimultiple.com/synthetic-data

[79] “Ieee standard for scada and automation systems,” *IEEE Std C37.1-2007 (Revision of IEEE Std C37.1-1994)*, pp. 1–143, 2008.

[80] A. Abur and A. G. Exposito, *Power system state estimation: theory and implementation*. CRC press, 2004, ch. 2,5.

[81] Y.-F. Huang, S. Werner, J. Huang, N. Kashyap, and V. Gupta, “State estimation in electric power grids: Meeting new challenges presented by the requirements of the future grid,” *Signal Processing Magazine, IEEE*, vol. 29, pp. 33–43, 09 2012.

[82] R. A. M. van Amerongen, “On convergence analysis and convergence enhancement of power system least-squares state estimators,” *IEEE Transactions on Power Systems*, vol. 10, no. 4, pp. 2038–2044, 1995.

[83] A. Monticelli, “Electric power system state estimation,” *Proceedings of the IEEE*, vol. 88, no. 2, pp. 262–282, 2000.

[84] A. Meliopoulos, B. Fardanesh, and S. Zelingher, “Power system state estimation: Modeling error effects and impact on system operation.” 01 2001.

[85] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. [Online]. Available: https://books.google.ca/books?id=0jbxwQEACAAJ

[86] D. Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O’Reilly Media, Incorporated, 2019. [Online]. Available: https://books.google.ca/books?id=BQFhwQEACAAJ

[87] D. Mwiti, “Introduction to Generative Adversarial Networks (GANs): Types, and Applications, and Implementation,” *Medium*, Apr 2021. [Online]. Available: https://heartbeat.fritz.ai/introduction-to-generative-adversarial-networks-gans-35ef44f21193

[88] D. MacKay, D. Kay, and C. U. Press, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. [Online]. Available: https://books.google.ca/books?id=AKuMj4PN_EMC

[89] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.

[90] Y. Qin, N. Mitra, and P. Wonka, "Do GAN Loss Functions Really Matter?" *arXiv*, Nov 2018. [Online]. Available: https://arxiv.org/abs/1811.09567

[91] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[92] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2794–2802.

[93] Y. Yazici, C.-S. Foo, S. Winkler, K.-H. Yap, and V. Chandrasekhar, "Empirical analysis of overfitting and mode drop in gan training," in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 1651–1655.

[94] E. Alpaydin, *Introduction to machine learning*, 3rd ed. MIT press, 2014.

[95] A. Tsybakov, *Introduction to Nonparametric Estimation*, ser. Springer Series in Statistics. Springer New York, 2008. [Online]. Available: https://books.google.ca/books?id=mwB8rUBsbqoC

[96] C. Euler, C. T. Lin, B. Juarez, and M. Flores, "Real-time activity classification by matched filtering using body-worn accelerometers," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2016, pp. 1059–1062.

[97] M. Elgenedy, M. Sayed, M. Mokhtar, M. Abdallah, and N. Al-Dhahir, "Interference mitigation techniques for narrowband powerline smart grid communications," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2015, pp. 368–373.

[98] D. Erdogmus, R. Agrawal, and J. C. Principe, "A mutual information extension to the matched filter," *Signal Processing*, vol. 85, no. 5, pp. 927–935, 2005.

[99] H. K. Iqbal, F. H. Malik, A. Muhammad, M. A. Qureshi, M. N. Abbasi, and A. R. Chishti, "A critical review of state-of-the-art non-intrusive load monitoring datasets," *Electric Power Systems Research*, vol. 192, p. 106921, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779620307197

[100] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2012.

[101] M. Zeifman, "Disaggregation of home energy display data using probabilistic approach," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 23–31, 2012.

[102] J. Zhao, A. Gómez-Expósito, M. Netto, L. Mili, A. Abur, V. Terzija, I. Kamwa, B. Pal, A. K. Singh, J. Qi, Z. Huang, and A. P. S. Meliopoulos, "Power system dynamic state estimation: Motivations, definitions, methodologies, and future work," *IEEE Transactions on Power Systems*, vol. 34, no. 4, pp. 3188–3198, 2019.

[103] R. Amerongen, “On convergence analysis and convergence enhancement of power system least-squares state estimators,” *Power Systems, IEEE Transactions on*, vol. 10, pp. 2038 – 2044, 12 1995.

[104] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 07 2017, pp. 5967–5976.

[105] J. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 10 2017, pp. 2242–2251.

[106] P. Cerda and G. Varoquaux, “Encoding high-cardinality string categorical variables,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 5 2020.

[107] (2021, May) Electric grid test cases. [Online; accessed 19. Jun. 2021]. [Online]. Available: https://electricgrids.engr.tamu.edu/electric-grid-test-cases

[108] “Power Systems Test Case Archive - UWEE,” Jan 2021, [Online; accessed 29. Jan. 2021]. [Online]. Available: https://labs.ece.uw.edu/pstca/index.html

[109] L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, “pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems,” *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, 11 2018.

[110] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[111] S. Makonin, F. Popowich, L. Bartram, B. Gill, and I. V. Bajić, “Ampds: A public dataset for load disaggregation and eco-feedback research,” in *2013 IEEE Electrical Power & Energy Conference*. IEEE, 2013, pp. 1–6.

[112] S. Makonin, B. Ellert, I. V. Bajic, and F. Popowich, “Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014,” *Scientific Data*, vol. 3, no. 160037, pp. 1–12, 2016.

[113] S. Makonin, “RAE: The Rainforest Automation Energy Dataset,” 2017. [Online]. Available: https://doi.org/10.7910/DVN/ZJW4LC

Appendix A

Public datasets

AMPd dataset

The Almanac of Minutely Power dataset (AMPds) is presented by [111, 112] and is available on-line at `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/FIE0S4`.

The dataset consists of series of time-stamped real power measurements for each individual load. The measurements are physically recorded throughout two years (from April 2012 to March 2014) at a sampling rate of 1 sample per minute. In addition to real power, voltage, current, frequency power factor and other electrical quantities are recorded. The aggregate consumption recorded by the smart meter is monitored and saved at the same sampling rate. The aggregate consumption is denoted WHE and named 'Whole-House meter'.

The monitored individual loads are given unique IDs. Occasionally, a branch circuit rather than an individual load is monitored. However, the following loads were individually monitored

CDE- Clothes Dryer
CWE- Clothes Washer
DWE- Dishwasher
FGE- Kitchen Fridge
FRE- HVAC/Furnace
HPE- Heat Pump
HTP- Instant Hot Water Unit
WOE- Wall oven

RAE dataset

The Rainforest Automation Energy (RAE) dataset is presented by [113]. The data is available on-line at `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi%3A10.7910/DVN/ZJW4LC`.

Measurements are physically acquired using sub-meters at a rate of 1Hz for two houses, both located in Burnaby, BC, Canada. For the first house, data is acquired in February and March 2016. Individual loads monitored included clothes dryer, Kitchen dishwasher, heat pump and others. For the second house, data is acquired in September and October 2017. The second house includes similar loads to the first one.

Curriculum Vitae

Name:	Samer El Kababji
Education and Degrees:	2017-2018 M.Eng. Electrical & Computer Eng. Western University
	1992-1995 M.Sc. Industrial Eng. 1986-1991 B.Sc. Electrical Eng. The University of Jordan
Awards	2019, 2020 Ontario Graduate Scholarship 2019, 2021 Outstanding Graduate Symposium Presentation

Publications:

- S. El Kababji and P. Srikantha, ”Power Appliance Disaggregation Framework Via Hybrid Hidden Markov Model,” 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), 2018, pp. 1-5, doi: 10.1109/CCECE.2018.8447822.

- J. Wang, S. El Kababji, C. Graham and P. Srikantha, ”Ensemble-Based Deep Learning Model for Non-Intrusive Load Monitoring,” 2019 IEEE Electrical Power and Energy Conference (EPEC), 2019, pp. 1-6, doi: 10.1109/EPEC47565.2019.9074816.

- S. E. Kababji and P. Srikantha, ”A Data-Driven Approach for Generating Synthetic Load Patterns and Usage Habits,” in IEEE Transactions on Smart Grid, vol. 11, no. 6, pp. 4984-4995, Nov. 2020, doi: 10.1109/TSG.2020.3007984.

Open source codes:

Generating Patterns and Habits of Electrical Loads using GANs
`https://github.com/skababji/ElecLoads`

In preparation:

Subject: A modified cycle GAN for bad data detection and identification
Target journal: IEEE Transactions on Smart Grid

Work experience:

- JAN 2021 - APR 2021 Intern
  Filament AI
- SEP 2019 - APR 2020 Teaching Assistant
  Western University
- 2015 - 2017 Founder
  Smartegrators Ltd
- 2013 - 2015 General Manager
  Specialised Gulf Welding Co.
- 2005 - 2013 Country Manager
  Illinois Tool Works
- 2001 - 2003 Assistant General Manager
  Mona Trading
- 1997 - 2001 Marketing Manager
  EDGO Group
- 1994 - 1997 Maintenance Manager
  Alnejma Bulk Pharmaceutical Co.
- 1992 - 1994 Maintenance Eng.
  Arab Drip Irrigation Systems