Small Smart Grid for The Power Utilization System

by

Majed Shakir

Bachelor of Science in Electrical Engineering, UNB, 2015

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Masters of Science in Engineering

In the Graduate Academic Unit of Electrical and Computer Engineering

Supervisor:	Yevgen Biletskiy, PhD, Electrical and Computer Engineering
Examining Board:	Saleh Saleh, PhD, Electrical and Computer Engineering
	Wei Song , PhD, Computer Science

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April 2019

©Majed Shakir, 2019

Abstract

The wide proliferation of renewable energy and deregulation of power grid systems require small power utilization systems to deploy intelligent methods of adjustment to the user power demand. In order to accomplish this goal, the smart power demand forecasting and power consumption optimization methods and algorithms need to be developed. For this purpose, small power utilization systems can benefit from the techniques developed for the smart grid in general.

This thesis is devoted to the development of the forecasting model based on the Long Short-Term Memory (**LSTM**) method and the optimization model based on Genetic Algorithm (**GA**) adopted for the use in home energy management systems (**HEMS**). The thesis work presents the small smart grid architecture, which describes the roles of the **LSTM** and **GA** in small smart grid. The system also includes the load identification subsystem based on the Newton-phaselet algorithm.

The **LSTM** method and **GA** are tested using Matlab simulation, as well as implemented using Python programming/scripting language with the PyTorch library. The forecasting LSTM algorithm is tested with a number of interruptible appliances to predict power demand over 24 hour segments. The experiments demonstrate that the developed algorithm generates a stable pattern of daily power demand; therefore, it has a high predictive power. The scheduling algorithm based on the developed optimization model is tested using photovoltaic power and stored battery power. The use of the developed algorithm allows automated shifting of power to achieve the lowest price without sacrificing user's comfort.

Acknowledgments

I would like to express my deep gratitude to Professor Yevgen Biletskiy my research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to express my very great appreciation to Dr. Eugene Hill, Dr. Maryhelen Stevenson, and Dr. Saleh Saleh for their valuable and constructive suggestions during the planning and development of this research work. Their willingness to give their time so generously has been very much appreciated.

I would also like to extend my thanks to the technicians at the Electrical Engineering department, especially Mr. Ryan Jennings, for their help in offering me the resources for my experiments. I would also like to extend my thanks the Electrical Engineering department staff. Especially, Shelley Cormier for her unwavering support.

I wish to thank Mr. Kieran Lea for his contribution to this research. Especially, in his help to develop the software.

Finally, I wish to thank all my family for their support, encouragement, and finical assistance throughout my study.

Contents

Abstract **ii**

Acknowledgments **iii**

Contents **iv**

List of Figures **vii**

Glossary **ix**

1 Introduction **1**

1.1 General 1

1.2 Thesis Objectives 2

2 Literature Review **4**

2.1 General 4

2.2 Load Identifier and Forecasting 5

2.3 Machine Learning 8

2.3.1 Artificial Neural Network 8

2.3.2 Recurrent Neural Network and Long Short-Term Memory . . 9

2.3.3 Scheduling Genetic Algorithm 15

2.4 Summary 18

3 System Design **19**

3.1 Forecasting Model 20
3.2 Load Identifying Model 20
3.3 Optimization Model 21

4 System Implementation **23**
4.1 Long Short-Term Memory 23
4.2 Genetic Algorithm 28
4.3 Graphic User Interface 31

5 Simulation and Results **37**
5.1 Simulation 37
5.1.1 Forecaster 37
5.1.2 Optimizer 40
5.2 Implementation 43
5.2.1 Forecaster 43
5.2.2 Optimizer 46
5.3 Limitations 48

6 Conclusion and Future Work **49**
6.1 Conclusion 49
6.2 Future Work 50

Bibliography **51**

Appendices **57**

Appendix A MATLAB Training Code **58**

Appendix B Forecaster Python Code **60**

Appendix C Appliances Python Code **68**

Appendix D Genetic Algorithm Python Code **69**

Appendix E Main Python Code **82**

Curriculum Vitae

List of Figures

2.1 Basic Artificial Neural Network Architecture 9

2.2 Unrolled Recurrent Neural Network **RNN** 10

2.3 Recurrent Neural Network **RNN** Unit 11

2.4 Long Short-Term Memory (**LSTM**) Unit 12

2.5 Long Short-Term Memory (**LSTM**) Unit 12

2.6 Long Short-Term Memory (**LSTM**) Step One 13

2.7 Long Short-Term Memory (**LSTM**) Step Two 13

2.8 Long Short-Term Memory (**LSTM**) Step Three 13

2.9 Long Short-Term Memory (**LSTM**) Step Four 14

2.10 Genetic Algorithm Flowchart 17

3.1 System Architecture 19

4.1 Dropout is Only Applied to Feed-forward Connections in The **RNN** (**LSTM**) 26

4.2 Graphic User Interface (**GUI**) Page 1 31

4.3 Graphic User Interface (**GUI**) Page 2 33

4.4 Results Example 33

4.5 System Time-of-Day Price, Photovoltaic (**PV**) support, and Batteries Support 34

4.6 System Schedule with Photovoltaic (**PV**) Example 35

4.7 System Schedule With No Photovoltaic (**PV**) Example 36

5.1 MATLAB Neural Network Toolbox Results With Validation Checks . 38
5.2 MATLAB Neural Network Toolbox Results Without Validation Checks 39
5.3 MATLAB Neural Network Error V.S. Epoch 39
5.4 MATLAB Scheduling Genetic Algorithm Results 41
5.5 MATLAB Scheduling Genetic Algorithm Temperature Results 42
5.6 Results Example 44
5.7 Implementation Forecaster Training Data 45
5.8 System Schedule with no Interruptible Appliances In Use Battery Coverage Example 46
5.9 System Schedule With No Interruptible Appliances In Use Example . 47

Glossary

ADMD After Diversity Maximum Demand.

AI Artificial Intelligence.

ANN Artificial Neural Network.

DF Diversity Factor.

DLC Direct Load Control.

DR Demand Response.

GA Genetics Algorithm.

GUI Graphic User Interface.

HEMS Home Energy Management Systems.

HVAC Heating, Ventilation, and Air Conditioning.

IoT Internet of Things.

ISO Independent System Operator.

KW Kilowatts.

LSTM Long short-term memory.

MSE Mean Squared Error.

PBC Price-Based Control.

PV Photovoltaic.

RNN Recurrent Neural Network.

RSCL Residential and Small Commercial Loads.

SVM Support Vector Machine.

tanh Hyperbolic Tangent.

Chapter 1

Introduction

1.1 General

Since the introduction of deregulated power grid systems, there has been a push for an efficient grid. This push has resulted in the wide proliferation of the smart grid concept and its functions. The original power grid systems are incorporating such aspects as artificial intelligence, renewable energy technologies, and storage systems. Those innovations contribute in the grid system's evolution into the smart grid system. The main purpose of the smart grid is to establish a balance between the generation and load. Research in this field aims to solve issues plaguing grid systems such as power shifting in peak hours, power transmission congestion, and the injection of renewable energy into the system [1]. These smart systems are now spearheading the industry as these issues are important to sustain the grid.

Small utilization systems can benefit from the techniques developed for the smart grid. These techniques can help to turn the power utilization systems into a small smart grid. Techniques, including machine learning and other artificial intelligence can adaptively adjust to the user power demand. These algorithms can ensure the efficiency of the system and reduce wasted energy.

Machine learning approaches, such as an artificial neural network (**ANN**), can accurately and robustly predict classification and regression results. The results of an **ANN**-based algorithm can predict the future power demand for utilization sys-

tem. These results can then be applied to an optimizer, such as scheduling genetic algorithm, to regulate and schedule utilization system for optimal energy management. For instance, heating, ventilation, and air conditioning (**HVAC**) systems consume energy on a regular basis even when it is not essential to the users. In this case, a predictive algorithm can be used to learn the user's behaviour to regulate power consumption for optimal usages.

The research focus of the power industry has mostly been in the generation and distribution side of the smart grid. However, the power industry does incentivise utilization with models such as time-of-day pricing [2]. This research bridges the gap between the power industry and the utilization systems to reduce consumption during peak demand.

Converting the utilization system into an adaptive small smart grid helps both the user and the grid system. The goal of applying artificial intelligence algorithms is to optimize power consumption without impacting user comfort level and experience.

1.2 Thesis Objectives

The smart grid system has been well developed for the segments of power generation, transmission, and distribution, but not developed enough for power utilization. The main objective of the proposed work is to create the framework for a small smart grid for the power utilization system. This involves artificial intelligence automation including Long Short-Term Memory (**LSTM**) and Genetic Algorithm (**GA**).

The secondary objective is to investigate the scheduling of power storage elements to improve the scheduling of regulated loads, and to offset deferrable load pricing. This relaxes the optimization settings when power is drawn from the stor-

age elements. Furthermore, loads that are forced to defer due to the time-of-day pricing system, are not forced to defer if the loads can run on the stored power.

Thesis outline is organized as fallows. Chapter 2 reviews related literature. Chapter 3 describes the design of the system under study. Chapter 4 is devoted to the details of the system implementation. Chapter 5 demonstrate the simulated and the experimental result. Chapter 6 concludes the thesis.

Chapter 2

Literature Review

2.1 General

In power utilization systems, loads can be categorized as fixed loads, regulated (schedulable) loads, and deferrable loads. A forecasting algorithm can develop a method to predict scheduling strategies for the regulated load. Normally, forecasting uses engineering methods, statistical methods, and artificial intelligence methods [3]. The most commonly implemented methods in the field of artificial intelligence are Artificial Neural Network (**ANN**) and Support Vector Machine (**SVM**) [4, 5]. An optimizing algorithm can be deployed to minimize the overall cost and energy consumption [6]. Consequently, a house with a home energy management system (**HEMS**) plays an important role in enhancing efficiency, reliability, economy and energy conservation for distribution systems.

The **HEMS** role is to monitor and schedule appliances in real-time. In addition, user preferences are extracted via the human-machine interface with the appliances. A model is then developed to reduce the cost and improve energy utilization [7].

The grid framework is based on the demand response (**DR**) model [8, 9]. The **DR** is classified into the Direct Load Control (**DLC**) and Price-Based Control (**PBC**). In the **DLC**, the power utilities control particular appliances during peak hours. Generally, this control manages short-term urgent situations, such as a high frequency deviation in the power system [10, 11, 12]. The **PBC**, however, incen-

tivise users to be active articipants in the demand response program and appliance scheduling [13, 14]. Depending on the development of scheduling algorithms, the **PBC** methods are real-time pricing [15] and day-ahead pricing [6]. Both artificial neural networks and support vector machines are widely used in forecasting, and it is difficult to conclude which is the best forecasting model to use [3].

2.2 Load Identifier and Forecasting

The standard approach to model residential and small commercial loads (**RSCLs**) is to establish patterns in the energy consumption. One of the early methods was the bottoms-up method [16]. The bottoms-up structure starts at the individual appliances level and ends at the household level. This model offers an accurate prediction of individual loads if all appliances maintain a cyclic behavior over similar hours in similar seasons. The disadvantage of the bottoms-up model is the large amount of historical data. The data includes, but is not limited to, multiple households' energy consumption data, demographic data, and statistical probability of appliances usage [16, 17]. Another limitation of the bottoms-up model is that any changes in the appliances degrade the accuracy of the algorithm.

The gamma distribution is another algorithm that identifies and forecasts load demand. The algorithm is based on the After Diversity Maximum Demand (**ADMD**) concept [18]. The **ADMD** is the maximum demand, per customer, as the number of customers connected to the network approaches infinity. This is derived from the maximum yearly nodal demand on the transmission network divided by the number of customers served by that network. The **ADMD** is multiplied by a diversity factor (**DF**) that increases the demand per customer as the size of the group decreases [18, 19]. There are multiple forms that formulate the **ADMD**, as shown in Equation 2.1 - Equation 2.3 [18]:

$$MD = N * DF * ADMD, \tag{2.1}$$

$$ADMD = \lim_{x \to \infty} \frac{1}{N} \sum_{n=1}^{N} P_n, \tag{2.2}$$

$$DF = 1 + \frac{k}{N}, \tag{2.3}$$

where:

MD: Maximum demand of the customer group

DF: Diversity factor, also defined as the ratio of non-coincident maximum demand of the customer group to the coincident maximum demand of the group

$ADMD$: After Diversity Maximum Demand per customer

P_n: demand of customer at the time of maximum system demand.

N: Number of customers

k: Empirically determined constant

The gamma distribution represents the demand by sampling power meter readings. Basically, the algorithm establishes the forecasting by creating a utilization profile/load profile. The limitation is the need for a large statistical data set. There is also a need for a stochastic process that demands continuous sampling to obtain correct values [17, 18].

A popular machine learning method is the artificial neural network (**ANN**), *which is covered in detail in section 2.3: Machine Learning.* The purpose of deploying **ANN** is to learn patterns without explicit knowledge in the overall model or profile [20]. The major concern with the algorithm is the need for a large amount of historical data. The generalization of **ANN** and its ability to forecast beyond its

data space can also be questionable.

The Newton-phaselet algorithm is another method constructed to identify the ON-OFF status of energy storage appliances in residential and small commercial loads (**RSCL**) based on the ZIP model. The ZIP model is based on the generic load class constant power, constant current and constant power impedance loads. The model is used in power system analysis, planning, and design. The general ZIP model is characterized in terms of active and reactive power demands as inEquation 2.4 and Equation 2.5 [17]:

$$P = P_o(P_Z(\frac{V}{V_0})^2 + P_I(\frac{V}{V_0}) + P_S)(1 + K_{pf}\Delta f), \tag{2.4}$$

$$Q = Q_o(Q_Z(\frac{V}{V_0})^2 + Q_I(\frac{V}{V_0}) + Q_S)(1 + K_{qf}\Delta f), \tag{2.5}$$

where:

V: The measured voltage

V_o: Rated voltage of the model load

P_o: Active power rating of the model load

Q_o: Reactive power rating of the model load

P_Z: Active power demand by the constant impedance

Q_Z: Reactive power demand by the constant impedance

P_I: Active power demand by the constant current

Q_I: Reactive power demand by the constant current

P_S: Active power demand by the constant power

Q_S: Reactive power demand by the constant power

K_{pf}: The frequency sensitivity to variations in active power

K_{qf}: The frequency sensitivity to variations in reactive power

f: The operating frequency

Instantaneous changes in the active and reactive powers in the ZIP model can compose the demands of different appliances. Then, appliances can be mapped into constant impedance current and load. This can be easily decomposed by the ZIP model [21]. In application, the power meter provides only active power readings; therefore, numerical methods can be used to determine the reactive power [17].

The author in [17] adopts Newton-phaselet iteration methods based on the ZIP model, where the measured active power is fed to the algorithm. The ZIP model also standardizes values for K_{pF} and K_{qF}. The Newton-phaselet algorithm can detect the appliances active power value by varying the K_{pF} and K_{qF} values corresponding to each device.

The strength of the Newton-phaselet algorithm is that (unlike most other algorithms) it eliminates the need for data training or modeling. Newton-phaselet frame struggles in the need to fine tune K_{pF} and K_{qF} standardized values. Nonetheless, the Newton-phaselet has demonstrated that it can produce accurate results with only the active power without the need for data training or model building.

2.3 Machine Learning

2.3.1 Artificial Neural Network

An artificial neural network (**ANN**) is inspired by the biological learning process of building a complex web of interconnected neurons in which the human brain can identify objects, complete tasks, reason solutions, and perform other functions [22]. An **ANN**, similar to the biological process, creates neurons with an input

layer, hidden layer and output layer as indicated Figure 2.1. The figure also shows a possible example of **ANN** architecture. Multiple layers or different connections can be applied to the neural network making **ANN** one of the most powerful learning methods used to find patterns in data. The algorithm helps devices to adaptively change without explicit representation of the changes.

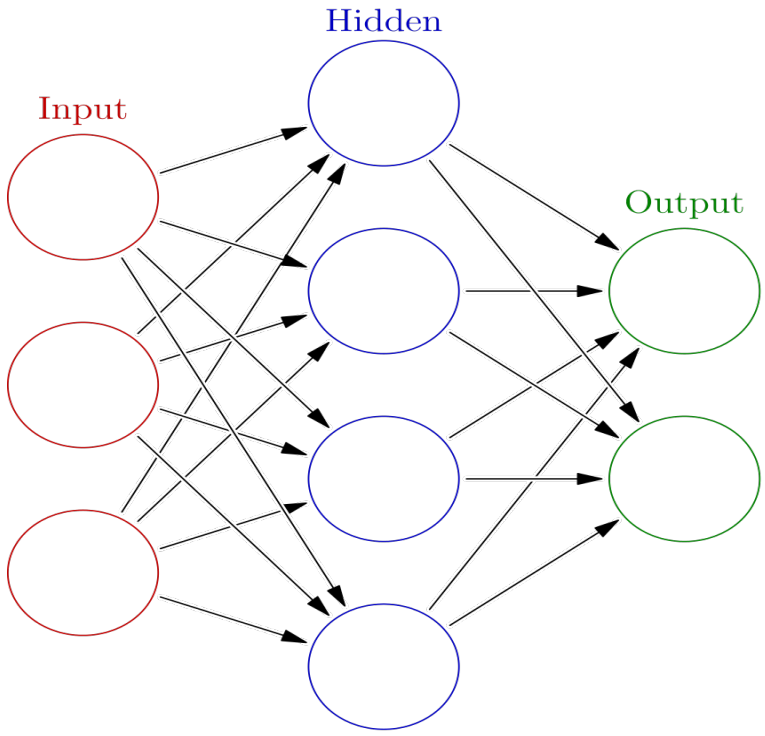


Figure 2.1: Basic Artificial Neural Network Architecture [23]

2.3.2 Recurrent Neural Network and Long Short-Term Memory

A recurrent neural network **RNN** is a deep learning class of **ANN**. Designed to learn sequential or time-varying patterns. An **RNN** builds on the **ANN** architecture by adding a feedback loop allowing the **RNN** to act as a forecasting engine. An **RNN** is essentially optimized for a forecasting application, whereas an **ANN** works best in classification or regression applications [24]. In addition,

RNN requires a large memory and CPU time that hinders its use. It also suffers from a vanishing gradient problem. Since an **RNN** is a large network, small changes in backpropagation of the output layer would be caused by the extremely small changes in the input layer. This is especially true when using a hyperbolic tangent **tanh** or Sigmoid function because they are bounded between $[-1, 1]$ and $[0, 1]$ respectively. Consequently, a reasonable gradient would never be computed [24, 25, 26]. The work [26] argued that gradient descent is inadequate to train tasks involving long-term dependencies. There are multiple solutions for the vanishing gradient problem. The most popular solutions are: (1) gradient clipping in which the gradient is cut with an identified threshold [26]; (2) specific designed optimizer, such as backpropagation through time [24]; and (3) gating, such as Long Short-Term Memory **LSTM** [27]. The technique most commonly used to produce the best results is gating, specifically **LSTM** [28, 29].

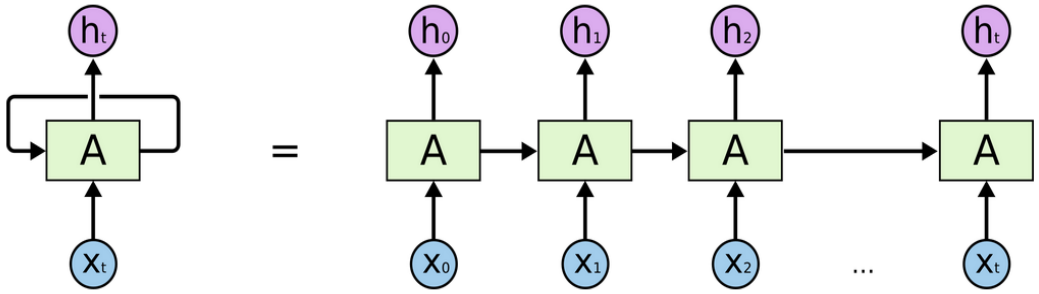


Figure 2.2: Unrolled Recurrent Neural Network **RNN** [30]

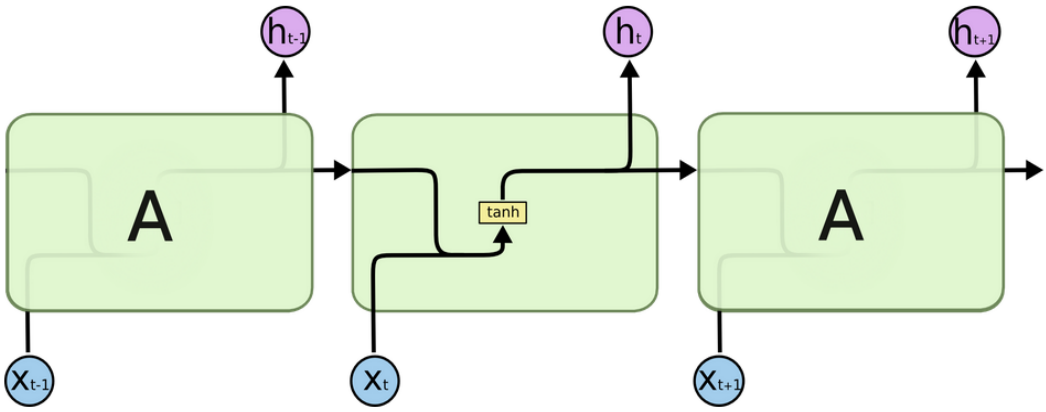


Figure 2.3: Recurrent Neural Network **RNN** Unit [30]

The **LSTM** is designed with an architecture that considers long-term dependencies. The architecture solves the **RNN** vanishing gradient problem, where the backpropagation error either blows up or decays exponentially [29]. Basically, the layers in the **LSTM** are a set of recurrently connected blocks (Figure 2.4). The overall connection is similar to the **RNN** recurrent block connections in Figure 2.3. Both **RNN** and **LSTM** connections resemble the memory chip in a digital computer [24, 27, 29]. Each **LSTM** unit contains an input gate, output gate, and forget gate (Figure 2.5). Using the memory chip analogy, the gates are comparable to the read, write, and reset functions, respectively [29].

In the **LSTM** unit (Figure 2.6), the use of function in Equation 2.6 and Equation 2.7 decides whether to forget/ignore or remember the input data. Figure 2.7 shows another Sigmoid function in Equation 2.8 that resolves which weights to update. Meanwhile, the hyperbolic tangent Equation 2.9 trains the weights. A clear contrast is demonstrated in this particular step between **LSTM** and **RNN**. The **RNN** unit in Figure 2.3 trains the weights using an equation similar to Equation 2.9. However, the hyperbolic tangent function trains all the data, creating a vanishing gradient problem. Techniques such as dropout can help with this problem by only addressing the input stage. In contrast, **LSTM** training addresses all

stages of the network with multiple Sigmoid functions to decide what to forget or remember. Equation 2.10 clearly presents how **LSTM** forgets or remembers input data. Figure 2.9, Equation 2.11, and Equation 2.12 show how the output layer and hidden layer are finally updated.

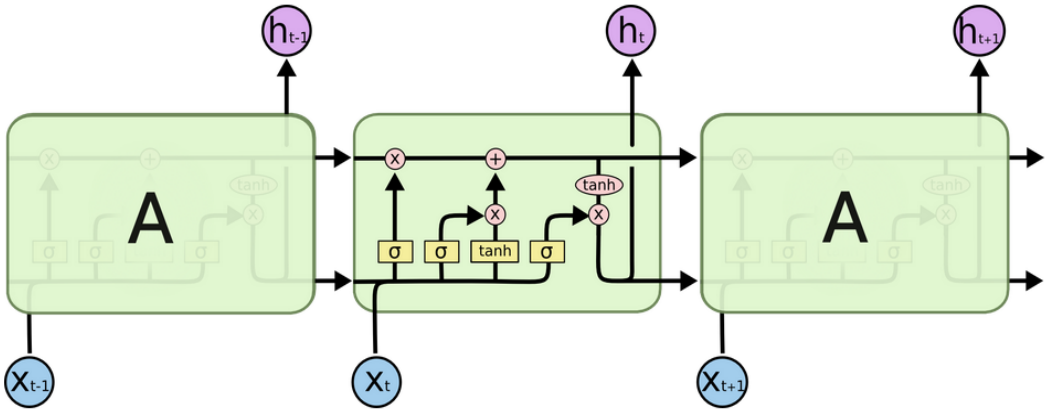


Figure 2.4: Long Short-Term Memory (**LSTM**) Unit [30]

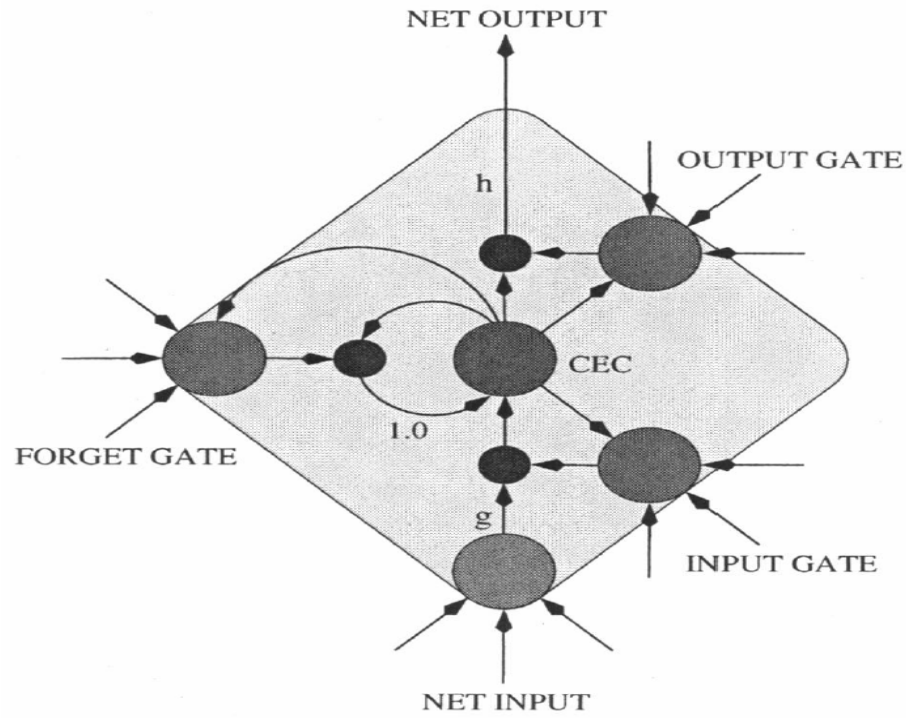


Figure 2.5: Long Short-Term Memory (**LSTM**) Unit [29]

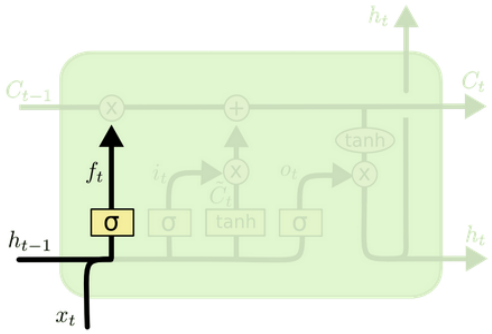


Figure 2.6: Long Short-Term Memory (**LSTM**) Step One [30]

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$f_t = \sigma(W_f.[h_{(t-1)}, x_t] + b_f) \tag{2.7}$$

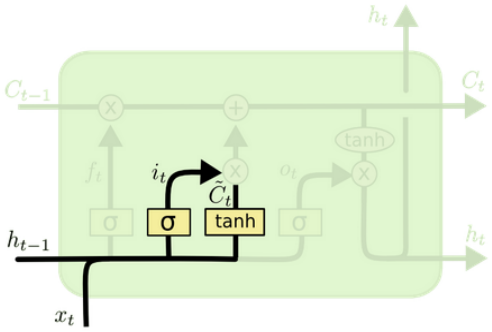


Figure 2.7: Long Short-Term Memory (**LSTM**) Step Two [30]

$$i_t = \sigma(W_i.[h_{(t-1)}, x_t] + b_i) \tag{2.8}$$

$$\widetilde{C}_t = tanh(W_c.[h_{(t-1)}, x_t] + b_c) \tag{2.9}$$

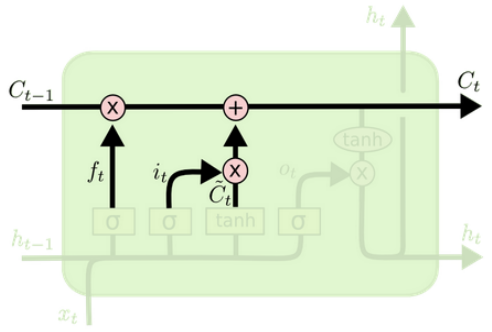
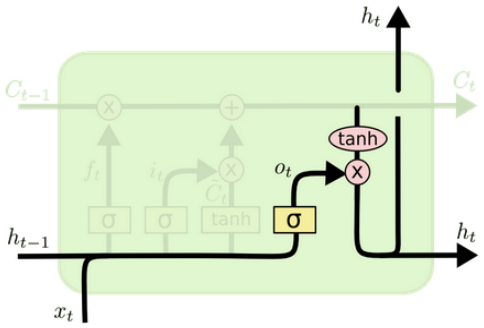


Figure 2.8: Long Short-Term Memory (**LSTM**) Step Three [30]

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{2.10}$$



$$o_t = \sigma(W_o.[h_{(t-1)}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

Figure 2.9: Long Short-Term Memory

(**LSTM**) Step Four [30]

In a comparison between an **ANN** and an **RNN** [31], the **RNN** is more accurate. The authors also demonstrated that an **ANN** is reliable for the operating points covered in the training set of the data. However, points, which are extrapolated outside the training set, have resulted in significant errors. Neural networks are uncomplicated to implement in comparison to **RNN**s. An **ANN** is computationally less expensive and can be applied to wide range of buildings due to its control accuracy and robustness. In comparing an **RNN** and **LSTM**, the authors in [28] found that **LSTM** performed better with fewer iterations than **RNN**. Even though **LSTM** is more computationally expensive than an **RNN** (due to the addition of the forget gate) its overall computation is comparable. Since **LSTM** requires fewer iterations to converge on a solution, the addition of the forget gate is computationally less significant.

To conclude, **LSTM** has been chosen for the present research for its robustness and accuracy over other machine learning algorithms. The **LSTM** is also capable of handling the dynamics of a residential or commercial building where new equipment is added frequently. There has also been a lack of research exploring **LSTM** for forecasting building power consumption.

2.3.3 Scheduling Genetic Algorithm

In its basic form, artificial intelligence (**AI**) is inspired by human intelligence. The puzzle is that **AI** strives to solve how does a brain, biological or electronic, perceives, understands, and solves complex problems [32]. Such as, the use of **AI** in this research is to systematize and automate the appliances energy management system. The algorithms adapted in **AI** can find solutions to complex optimization problems with no knowledge of the problem space [33].

A genetic algorithm mimics the natural evolution of organisms. Organisms that are ill-suited for the environment die, whereas organisms that can adapt (mutate) to the environment prosper and reproduce. Consequently, what works well for nature also works well for artificial intelligence [32]. These trends have been adopted to develop multiple algorithms such as artificial neural network and genetic algorithms.

A scheduling genetic algorithm, similar to a general/common genetic algorithm, starts by creating a random number of population sets. Each population set contains a number of individuals that represents the solution for the variables. This is called the chromosome [34]. The population set is then evaluated by a fitness function to determine each population success, which is the ability for each population to survive in the problem environment. The fitness function helps select the best candidate for the cross-over, where the traits for the best of the population are randomly passed down to their children. The children will undergo random mutation to their traits. The operation can continue until a reasonable fitness is reached or to a preset iteration number.

In the selection stage Figure 2.10, two parents are chosen from the population for crossing. The use of this method is to randomly nominate chromosomes from the population according to the evaluation function. Methods include roulette

wheel selection, random selection, and tournament selection. The higher the fitness function, the higher the chance that an individual has of being selected. The selection pressure is defined as the degree to which the better individuals are favored [35]. The research is focussed on tournament selection. According to [35], tournament selection is an ideal selection strategy. This strategy uses a tournament with all of the population to select an individual with the highest fitness. The tournament winners are inserted into the mating pool. This method drives genetic algorithms (**GA**) to improve the fitness of the succeeding genes. Consequently, the method is more efficient and leads to a more optimal solution Algorithm 1 [35].

In the crossover stage Figure 2.10, the two selected parents from the selection stage produce a child. The operation is applied to the mating pool in the hope that it creates a better offspring. The crossover works by selecting two individual strings for the mutation at random. A cross site (section) is selected along the string length at random. Finally, the position values are swapped between the two strings following the cross site Algorithm 1.

In the mutation stage Figure 2.10, the main role is to prevent the algorithm from being trapped on a local minimal, and it also keeps the gene pool stocked ensuring ergodicity. Ergodic is a non-zero probability of generating any solution from any population state [35]. Mutation plays a role in randomly distributing genetic material (information). Basically, the stage introduces new genetic material to the population by randomly reshaping its building blocks Algorithm 1.

The strength of scheduling genetic algorithm is that it does not require knowledge of the problem space. Therefore, the algorithm can operate on a population of solutions that does not require a gradient or other problem-specific information to produce the optimal solution. A **GA** is also resistant to becoming trapped in local optima [35]. Furthermore, a **GA** is capable in handling non-linear and noisy optimization problems [34, 35]. The weakness of genetic algorithms (which is also

the weakness of all heuristic solution methods) is that the heuristic solutions do not guarantee the best optimal solutions (if one exists). However, heuristic methods find useful optimal solutions [33]. Specifically, in scheduling heuristic solutions, the complexity occurs in defining the constraints of the optimization problem.

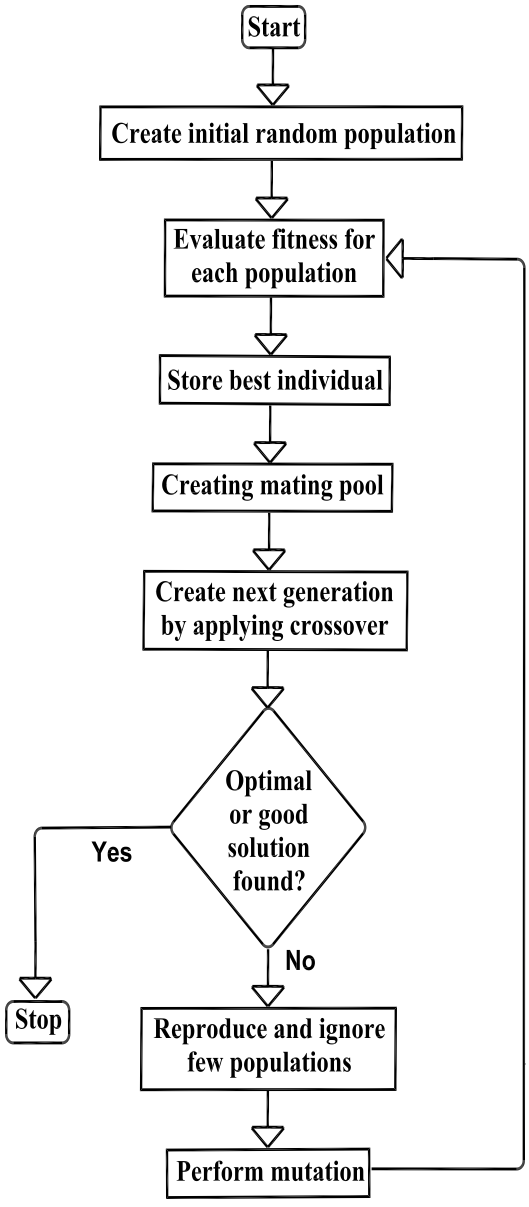


Figure 2.10: Genetic Algorithm Flowchart

Algorithm 1 Genetic Algorithm [35]

1: Start // Randomly populate n chromosomes
2: **for** $i = 0$ to $iteration.length$ **do**
3: Fitness // Evaluate the fitness $f(x)$ of each chromosome x in the population
4: Selection // Select two parent chromosomes from a population according to their fitness (the better fitness, the greater chance to get selected)
5: Crossover // Crossover the parents to form new offspring (children). If no crossover was performed, the offspring is an exact copy of the parents
6: Mutation // Mutate new offspring at each locus (position in chromosome)
7: Accepting // Place new offspring in the new population
8: **if** End condition is satisfied **then return** The best solution in the current population
9: **break**
10: **end if**
11: **end for**

2.4 Summary

The system utilizes machine learning, specifically **LSTM**, and its inherent forecasting capability. Sections 2.3.1 and 2.3.2 presented the essential properties of **LSTM** to forecast. A **GA** is deployed to optimize the forecasting results to reduce power consumption. A **GA** uses the forecasting results to optimize consumption to the time-of-day pricing, which ensures reducing energy usage and cost.

To ensure that the forecasting is updated to changes in both human and appliances behaviors, data is feedback to the forecaster. The Newton-phaselet identifies power consumption from the power meter.

Chapter 3 details how the subsystems that implement the algorithms described in Chapter 2, are integrated with each other.

Chapter 3

System Design

An automated home energy management system design requires general understanding of current regulations and potential regulations, and of the ethics in collecting the building appliances data. The system design needs power data from the appliance, and the data should be handled carefully. When the data is stored, as shown in Figure 3.1, most of the data collected and generated are stored in a database. This is implemented to track the algorithm accuracy and potential improvements. None of the stored data is then disrupted or sold for other purposes.

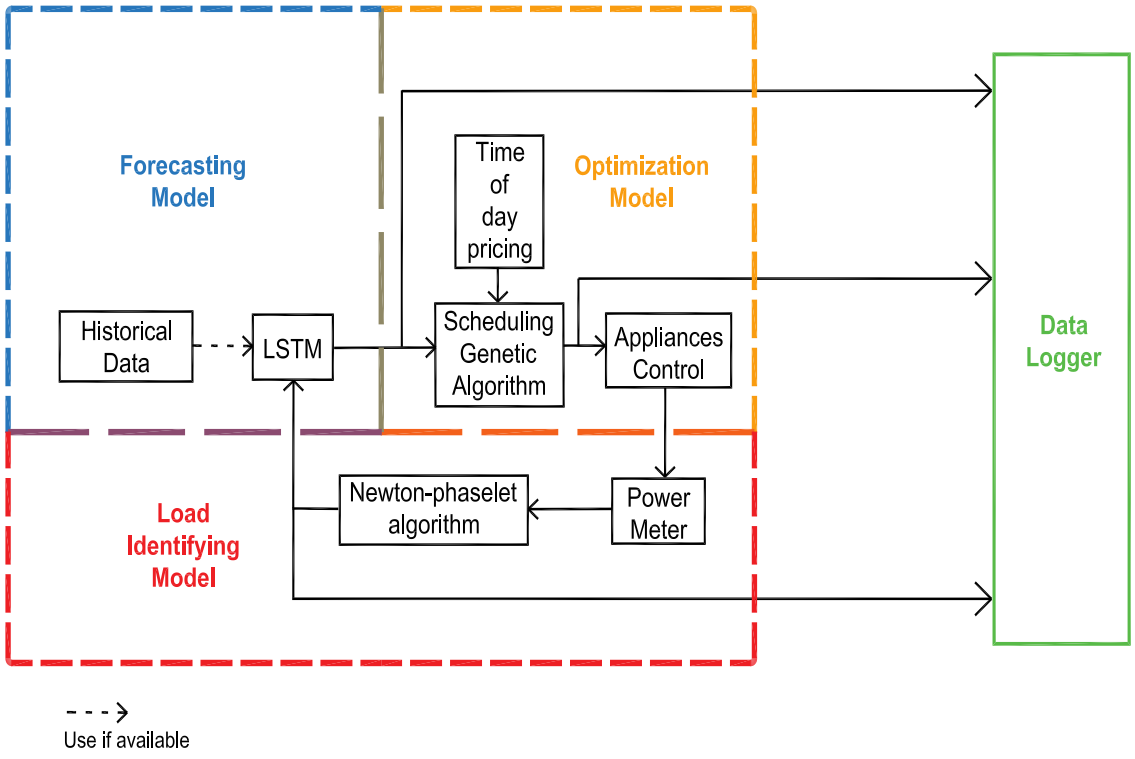


Figure 3.1: System Architecture

3.1 Forecasting Model

The forecasting model begins with Long Short-Term Memory (**LSTM**). This is an important part of the model because forecasting inherently has all the advantages and disadvantages of the algorithm. The reasons to choose a machine learning algorithm is accuracy, the ability to manage changing parameters, and the ability to deal with over-fitting and real-time training. The **LSTM** is a perfect candidate to deal with these issues. If historical data has been collected, **LSTM** can learn patterns. Notably, it is unrealistic to recognize all patterns because most houses do not store historical data of their power usage. Even if historical data is stored and available, it often does not reflect changes in the appliances. However, historical data can be used to extract general patterns about how appliances have been used.

The power of a **LSTM** algorithm can be utilized to learn patterns and behaviours in real-time. The algorithm's learning can occur with or without historical data. The disadvantage of an algorithm being employed without historical data available is how long it can take to perform on an adequate level. Furthermore, the architecture of **LSTM** allows the algorithm to manage new appliances. As technologies develop, appliances will become more energy efficient. Therefore, the algorithm should be capable of managing those changes to forecast accurately.

3.2 Load Identifying Model

Multiple methods can obtain the data to train **LSTM**, including sensors and profile based algorithms. Sensors are limited in use because of their installation cost and privacy regulations. Accordingly, methods that can identify the load with the power meter reads are preferred. The two types of algorithms that identify the load with the meter readings are building-profile-dependent and building-profile-

independent. As it is assumed, the building profile changes because of the changes in appliances. It is also inconvenient to keep changing the profile in the algorithm to match the building configuration. It is a challenging task to create a profile for each building. For example, not every building has a dishwasher. Thus, a profile that includes a dishwasher could detect a non-existent dishwasher because the algorithm is designed to detect one. The forecasting algorithm also would be forced to learn an appliance that does not exist. The building profile independent algorithm finds the signature patterns of each appliance without the need of a pre-defined profile.

A Newton-phaselet algorithm works independently to identify the load. In the present design, the algorithm detects all the On-Off cycles of each appliance. This information is then fed to the forecasting model to learn the patterns. The load identifying model works as feedback to the forecasting model. The Newton-phaselet finds the actual power used by the building. Therefore, if **LSTM** has inaccuracies in forecasting the future power, it can correct its train by the output of Newton-phaselet. After **LSTM** learns historical data, the load identifying model expands the training. When appliances and power demand changes, the forecasting model adapts the changes using the information detected by the Newton-phaselet. Most utilization systems do not have historical data. Newton-phaselet offers the only method to obtain data directly from the utilization system.

3.3 Optimization Model

In the optimization model, the proposed genetic algorithm (**GA**) schedules all the regulated and deferrable loads. The model forecasts the next day power consumption. The **GA** optimizes the time-of-day price set by the Independent System Operators (**ISO**). The regulated and deferrable appliances are scheduled to mini-

mize cost and preserve user's comfort. The **GA** is a heuristic algorithm, which provides multiple suitable scheduling solutions. This fact is needed because the optimization is conducted to a forecast solution.

Chapter 4

System Implementation

4.1 Long Short-Term Memory

The software library PyTorch [36] is used to implement the **LSTM**. The PyTorch is am open source platform based on the Tensor library [36, 37] that uses Python programming language to ease the neural network algorithm's implementation. The PyTorch package has a low overhead that speeds the **LSTM** training. The snapshot of the **LSTM** setup function is presented below. The program is commented using (#) comment statements, which explain the statements of the program in plain language.

```
def _setup_rnn(self, learning_rate=1e-4, momentum=0.25, nesterov=
    True, batch_first=True, dropout=0.5):
        # Save data for re-use
        _rnn_data = 'lstm.dump'
        _hidden_data = 'hidlstm.dump'

        if os.path.isfile(_hidden_data):
            # Load our saved parameters
            with open(_hidden_data, 'rb') as dump_file:
                _h_block = pickle.load(dump_file)
                (hidden, c0) = _h_block
        else:
            # No saved data, create new
            hidden = Variable(torch.randn(self.layer_number, self
    .hours, self.H_layer))
            c0 = Variable(torch.randn(self.layer_number, self.
    hours, self.H_layer))

        # Create or load the RNN
        if os.path.isfile(_rnn_data):
            # Load our saved parameters
            with open(_rnn_data, 'rb') as dump_file:
                self.rnn = pickle.load(dump_file)
        else:
            # No saved data, create new
            # Create the LSTM Network
```

```
24              self.rnn = torch.nn.LSTM(input_size=self.inputSize,
    hidden_size=self.H_layer, num_layers=self.layer_number,
25                                       bias=True,
26                                       batch_first=batch_first,
    bidirectional=False, dropout=dropout)
27
28          # Implements stochastic gradient descent with Nesterov
    momentum.
29          self.optimizer = torch.optim.SGD(self.rnn.parameters(),
    lr=learning_rate, momentum=momentum, nesterov=nesterov)
30
31          # Reduce learning rate when a mean squared error has
    stopped improving
32          self.scheduler = ReduceLROnPlateau(self.optimizer, 'min',
     patience=0, threshold=1e-1,
33                                             factor=0.90, min_lr=1e
    -25, verbose=False)
34          self.hidden = hidden
35          self.c0 = c0
36          # Create a Mean Squared Error to be the loss function
37          self.loss_fn = torch.nn.MSELoss(size_average=False)
```

Listing 4.1: **LSTM** Function Setup

In the setup for **LSTM** function (line 24, Listing 4.1), it starts the network by initiating the input size, hidden layer size, recurrent layer size, and other parameters. Moreover, it sets the error method to a mean squared error (**MSE**) function in Equation 4.1. This enables the **LSTM** network to train.

$$l_n = (x_n - y_n)^2, \tag{4.1}$$

$$\ell(x, y) = \begin{cases} \langle \sum_{a=1}^{a=n} l_a \rangle, & \text{Within a batch,} \\ \sum_{a=1}^{a=n} l_a, & \text{When all batchs are done,} \end{cases}$$

- where mean squared error (**MSE**) parameters:
  - x_n: Normal real (giving) output
  - y_n: Normal predicted output

LSTM network includes [36]:

- **Bias**: Adds a bias layer to the network.
- **Batch first**: Enables batch training in the network [38].
- **Dropout**: Introduces a dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to dropout [39].
- **Bidirectional**: The network becomes a bidirectional LSTM [40].

The batch training method divides the input into small batches, significantly reducing the training speed. Nonetheless, it might affect the generalization of the overall training since only the small batches are being trained at the time. The batching method is prone to over-fitting [22, 25, 38, 41]. Although, generalized training is a concern, batching is implemented in the design because the training occurs every 24 hours. A large set of data mends the generalizing issue. A dropout technique can also help generalize [39, 41].

The dropout technique establishes a layer, within the hidden layer, on the feed-forward connection of the **LSTM** network in Figure 4.1. It uses $H^{test} = \rho h$ to determine if the node (neurons) is dropped from the calculations of the output layer. The method improves the results and the generalization of **LSTM**. Dropout ensures that over-fitting does not occur or at least reduces the chance that over-fitting occurs [39, 41].

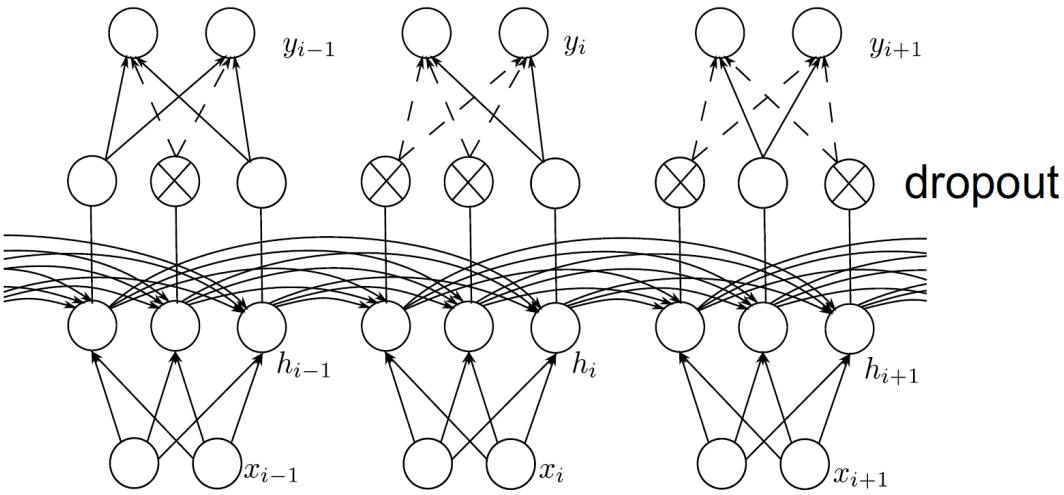


Figure 4.1: Dropout is Only Applied to Feed-forward Connections in The **RNN** (**LSTM**) [39]

Listing 4.1 line 27 sets up the training method. A stochastic gradient descent with momentum because it is effective with regression based problems [42]. The stochastic gradient descent uses a variation of gradient descent that can manage a noisy gradient [43]. This method is especially effective in deep networks [42]. Furthermore, the Nesterov's momentum acts to accelerate gradient descent that accumulates a velocity vector in Equation 4.2 and Equation 4.3. While the momentum concept might be ignored for simple neural networks, that is not the case in deep neural networks. Momentum is required to achieve or reach an optimized deep network training in a timely fashion [42]. Therefore, **LSTM** benefits the stochastic gradient descent with momentum to:

1. Manage noisy gradient, which is common in deep neural networks.

2. Accelerate/achieve the learning process for the deep network.

$$v_{t+1} = \mu v_t + \nabla f(\theta_t + \mu v_t) \tag{4.2}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{4.3}$$

The scheduler method in Listing 4.1 line 30 determines if stochastic gradient descent parameters require changing. While the training of the network is occurring, the learning rate parameter may require modifications. The scheduler method evaluates the gradient descent error. If the error is stuck or moving slowly, the learning rate will change accordingly. The method checks if the gradient is at local minimum. Especially in a deep network like **LSTM**, local minimum should be avoided as it materializes more often. Below is the LSTM training function.

```
1  def _train(self, errors, inputs, outputs, num_iter):
2
3          global y_pred, h_n
4          for epoch in range(num_iter):
5              # Forward pass: compute predicted y_pred, the next
       hidden layer h_n
6              # by passing inputs, hidden and C0 to the model.
7              y_pred, h_n = self.rnn(inputs, (self.hidden, self.c0)
       )
8
9              # Getting the last output of the predicted y_pred
10             y_pred = y_pred[-1]
11             # Calculate the mean square error between the real
       output and the predicted output
12             loss = self.loss_fn(y_pred, outputs)
13             # Decompose the hidden layer h_n
14             (self.hidden, self.c0) = h_n
15
16             prev_lr = self.optimizer.param_groups[0]['lr']
17             # Before the backward pass, use the optimizer object
       to zero all of the
18             # gradients for the variables it will update (which
       are the learnable weights of the model)
19             self.optimizer.zero_grad()
20
21             # Backward pass: compute gradient of the loss with
       respect to model parameters
22             loss.backward(retain_graph=True)
23
24             # Calling the step function on an Optimizer makes an
       update to its parameters
25             self.scheduler.step(loss.item(), epoch)
26
27             # Fine control the learning rate
28             new_lr = self.optimizer.param_groups[0]['lr']
29
```

```
30              if prev_lr == new_lr:
31                  self.optimizer.param_groups[0]['lr'] = new_lr *
    1.0001
32
33              self.optimizer.step()
34
35              # Print epoch for debugging
36              if epoch % 5 == 0:
37                  print(epoch, loss.item())
38
39              # Keep a list of errors for plotting later
40              errors.append(loss.item())
41
42          return y_pred, h_n
```

Listing 4.2: **LSTM** train function

The Listing 4.2 demonstrates how the **LSTM** is trained in the model. In line 8, by passing the inputs, hidden layer, and the memory cell state (forget gate), the functions initiates the **LSTM** in Listing 4.1. The results from the **LSTM** are saved along with the hidden layer and memory cell state. This is done to pass the data to the next iteration or the next training. Keeping a consistent hidden layer and memory state is essential to the network. The prediction output is then evaluated with real output using the loss (mean squared error) function. The loss is passed to the scheduler to adjust the parameters, such as the learning rate of the stochastic gradient descent. This ensures that the networking is learning properly. It also ensures that over-fitting does not occur.

The results of the forecasting stage are then saved and passed to the next stage. In the next stage the optimizer (genetic algorithm) uses the forecasting date to optimize the appliances to the time-of-day pricing.

4.2 Genetic Algorithm

The **GA** obtains the forecasting data and time-of-day pricing to start the process of Figure 3.1. The purpose of the **GA** is to minimize the cost, and by doing so, to reduce the power usage. The snapshot of the first stage of the **GA**, generating

population, is below (the program is commented using #).

```
1  def _generate_sols(self):
2          ## Genetic Algorithm
3
4          # Generate Solutions
5          cost = np.zeros(self.sols)
6
7          solution = []
8          for i in range(self.sols):
9              schedule = self._generatePop()  # generating initial
       population using generatePop()
10             cost[i] = self._costfunction(schedule)  # calculating
        cost of each schedule in population
11             solution.append(schedule)
12         initial_cost = cost[0]
13
14         data = []
15         cost_data = []
16         data.append(solution)
17         cost_data.append(cost)
18
19         # Crossover & Mutation
20         for it in range(self.tot_it):
21             cost1 = np.copy(cost_data[it])
22             probabilities = 1 / cost1  # probability(to become
       parent) is inversely proportional to cost
23             new_population = []
24             temp_cost = np.copy(cost1)
25             for j in range(self.sols):
26                 # parents = crossover(probabilities) # 2 parents
       are selected probabilistically by crossover()
27                 parents = _crossover_tourment_selection(temp_cost
       ,
28                                                         5)  # 2
       parents are selected probabilistically by crossover()
29                 feasible_child = self._generateChild(parents,
30                                                      solution)  #
        generateChild() and mutation() generate a feasible child of
       the parents
31                 cost1[j] = self._costfunction(feasible_child)  #
       cost of each child is calculated
32                 new_population.append(feasible_child)  # each
       child is stored in new population
33
34             solution = copy.copy(new_population)
35             data.append(new_population)
36             cost_data.append(cost1)
37
38         return initial_cost, cost_data, data
```

Listing 4.3: **GA** Generating Population (Solution)

Listing 4.3 presents how the **GA** works. The algorithm generates a random population (line 9). Therefore, it creates random appliance schedules. The algo-

rithm then creates children from the best parents in the population and evaluates the children. This process is repeated in specific number of iterations. The evaluation (crossover) of the parents is the tournament selection, where the population competes for the fittest parents. In this case, it means the best appliance schedule with the lowest cost.

Experiments found that running the cost function results in an undesirable schedule. Therefore, the cost function is modified to penalize the high and medium price producing desirable results. The medium price is further multiplied by 14 and the high price by 50. The multiplying factor is resulted from by trial and error Listing 4.4 below.

```
def _costfunction(self, schedule):
        # this function calculates the total cost in $ for the
    schedule using the price vector of the power company
        cost_vector = np.zeros(self.M)
        n_1 = np.where(schedule[self.N, :] > 0)[0]
        total_watts, ww = self._watts(schedule)
        power = self._checking(schedule)
        total_watts[n_1] = total_watts[n_1] + power * self.ts
        factor = np.ones(len(total_watts))

        ## Penalty for extra-wattages during costly hours ##
        high_price = np.where(self.price > (np.mean(self.price) *
     1.1))[0]
        medium_price = np.where(self.price > (np.mean(self.price)
    ))[0]
        factor[medium_price] = 14
        factor[high_price] = 50
        ##
        for i in range(self.M):
            cost_vector[i] = (self.price[i] * (total_watts[i]) *
    (1.0 / self.ts)) * (1.0 - schedule[self.N + 1, i]) * \
                             factor[i]
        cost = sum(cost_vector)
        return cost
```

Listing 4.4: **GA** Cost Function

4.3 Graphic User Interface

The Graphic User Interface **GUI** objective is to ease the interaction with the algorithms. By changing the input files and the optimization parameters, the results can easily be obtained. Figure 4.2 demonstrates how the **GUI** accepts the input matrices and enters the affected appliances. The default appliances are:

1. Dryer
2. Washer
3. Dishwasher
4. Heat pump
5. Electric vehicle battery
6. Vacuum cleaner
7. Iron
8. HVAC 1
9. HVAC 2
10. Refrigerator
11. LED lights 1
12. LED lights 2

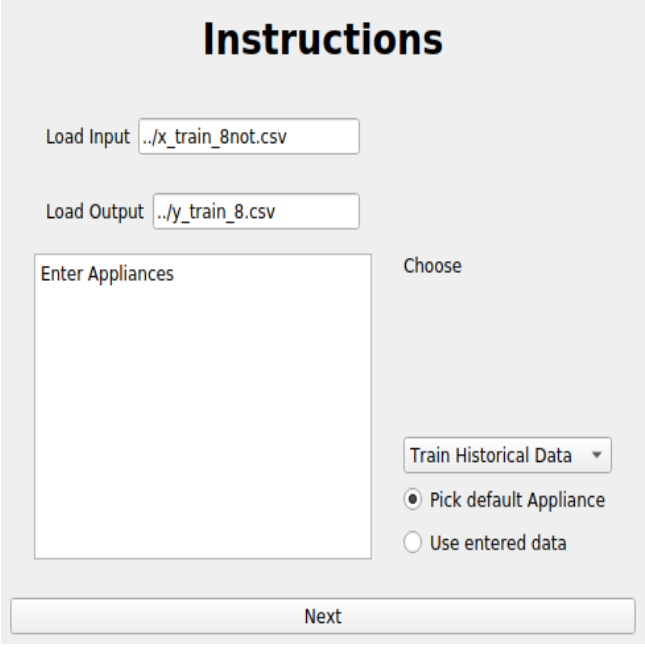


Figure 4.2: Graphic User Interface (**GUI**) Page 1

Figure 4.3 shows the optimization parameters page. On this page, the parameters for the forecaster (**LSTM**) or the optimizer (**GA**) can be fine-tuned or used as default. The intended user of this page is an expert in artificial intelligence. The default settings otherwise are sufficient to obtain satisfactory results. The parameters include:

- **LSTM**
  - **Input side:** The input size for the network;
  - **Number of layers:** The number of memory cells in the network;
  - **Hidden layer:** The number hidden layer for the network .
- **GA**
  - **Total time slots:** The number of time slots to optimize for with every 30 minutes as the default;
  - **Initial no.of solution in GA:** The number of population the **GA** generates;
  - **Total iterations in GA:** The number of iterations the **GA** generates;
  - **Comfort:** This is the level of comfort the **GA** strives to achieve. (In a maximum system, the cost saving might not be positive since the system schedules to the user comfort. In a minimum system, cost is prioritized rather than user preferences. Medium comforts, attempts to strike a balance between comfort and cost);
  - **Time slot period:** The period to schedule the appliances;
  - **Peak Photovoltaic (PV) power in kilowatts (KW):** The maximum power that can be provided by Photovoltaic (**PV**) (solar power);
  - **Power that the batteries are holding in KW:** The power held in the utilization batteries.

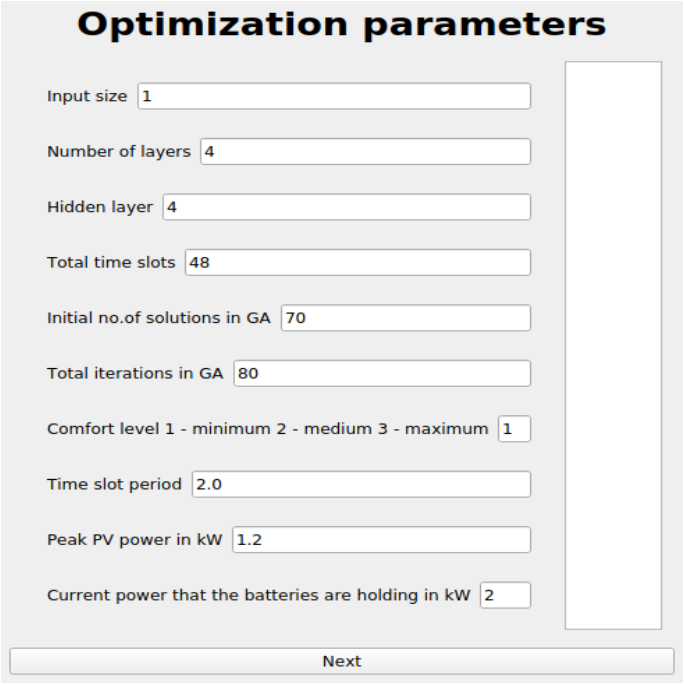


Figure 4.3: Graphic User Interface (**GUI**) Page 2

Figures 4.4 to 4.7 demonstrate examples of the expected results. Figure 4.6 and Figure 4.7 reveal how the difference in comfort and **PV** availability can change the result.

name	kW	type	time	LL	UL
Dryer	1.5	0	1.02	0	24
Washer	0.8	0	4.46	8	24
Dishwasher	1	0	0.97	0	24
Heat Pump	1.0	3	4.72	0	24
Electric Vehicle Battery	3	0	2.5	0	24
Vacuum Cleaner	0.65	0	1	7	19
Iron	1	0	1	10	20
HVAC 1	1	24	0	24	0.2
HVAC 2	2	1	24	4	2.5
Refrigerator	0.25	1	24	0	24
LED Lights 1	0.01	1	24	0	24
LED Lights 2	0.05	2	2.5	16	24

Figure 4.4: Results Example

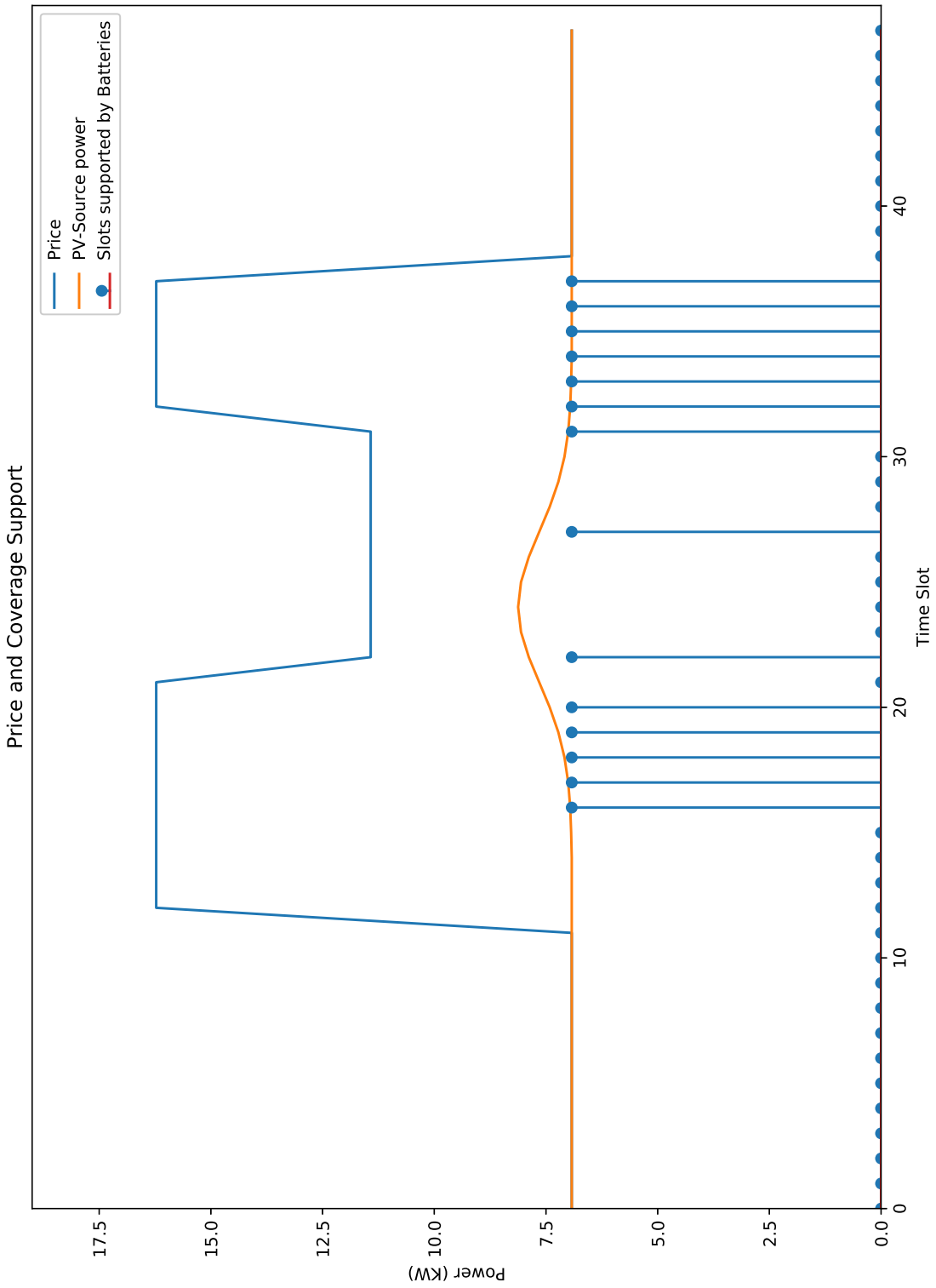


Figure 4.5: System Time-of-Day Price, Photovoltaic (**PV**) support, and Batteries Support

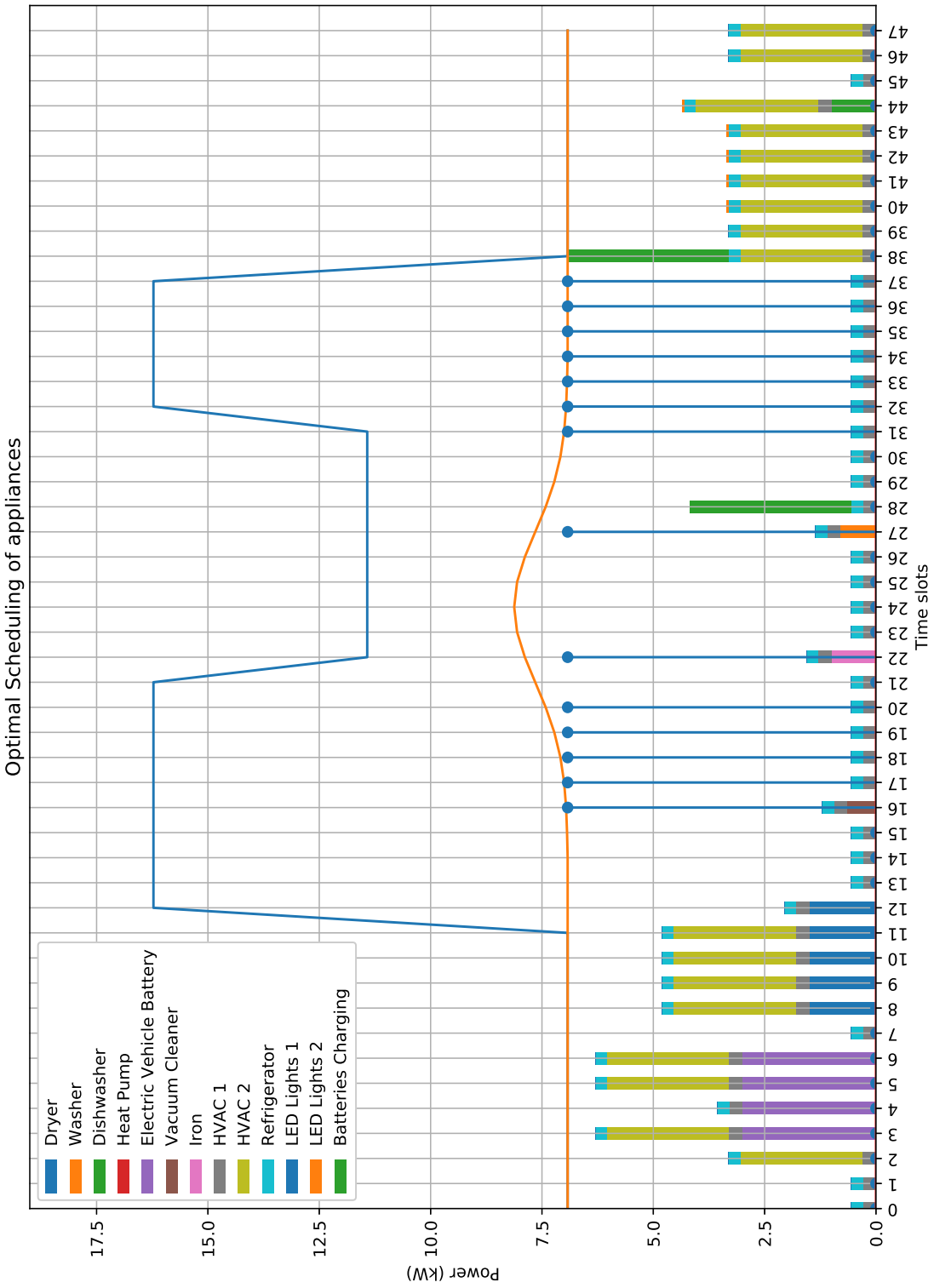


Figure 4.6: System Schedule with Photovoltaic (**PV**) Example

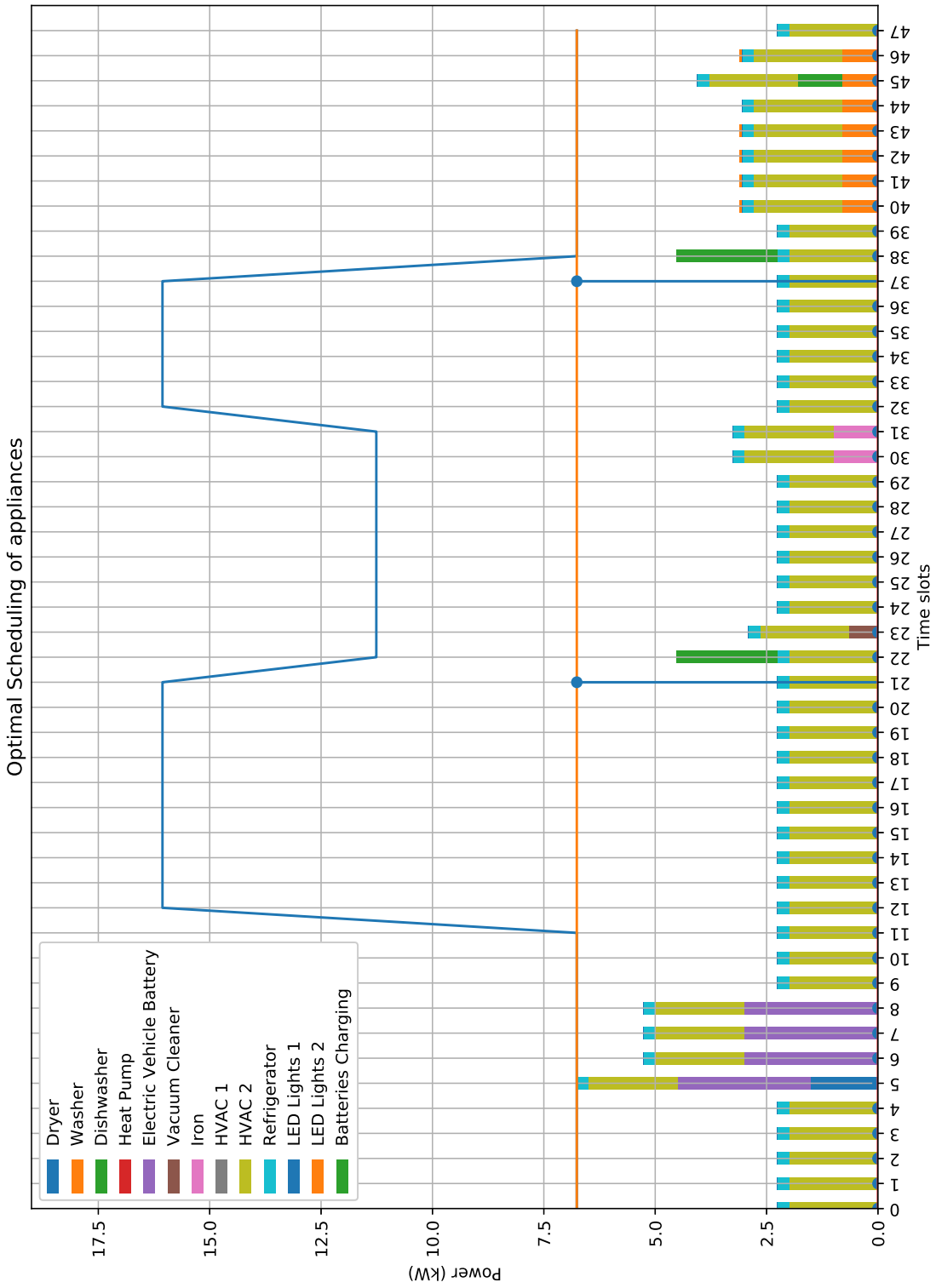


Figure 4.7: System Schedule With No Photovoltaic (**PV**) Example

Chapter 5

Simulation and Results

5.1 Simulation

5.1.1 Forecaster

The forecaster is built using the MathWorks MATLAB Neural Network toolbox (later renamed the Deep Learning toolbox). The toolbox has an easy interface with a high level of functionality while hiding the neural networks details. The tool's main purpose is to verify that a network can be contacted successfully.

The MATLAB's Levenberg-Marquardt backpropagation [44, 45] is used to train the data to obtain $performance = 2.6735$ and $test - performance = 2.1575$. The performance is evaluated using the Mean Square Error Equation 4.1, as shown in Figure 5.1 and 5.2 . The training performance confirms that a network can be established with effective results. This paved the way to design the optimizer.

Figure 5.3 displays how the MATLAB's toolbox manages training. During training, the algorithm tracks how often changes occur in the error and whether the error remains steady for a number of validation checks. The number of checks is arbitrary and user defined. Experiments found that if the number is large (more than five or six checks), the algorithm will take the full epoch length (Figure 5.2). The figure also shows that the validation check was set at 10 checks. When there are six checks, there are 18 iterations Figure 5.1. This limits the strain on the CPU and is an advantage of the neural network toolbox.

Notably, low validation check number (one or two checks) is ineffective, as it

might miss global or lower gradient points. However, a large number will take the full length of the epoch since the gradient will bounce around the minimum point. Figure 5.3 illustrate where the minimum point was located. Figure 5.1 and Figure 5.2, prove that the performance is minimal, and greater processing power cannot improve the results significantly. Furthermore, large iteration causes over-fitting issues and the training cannot generalize to the problem space.

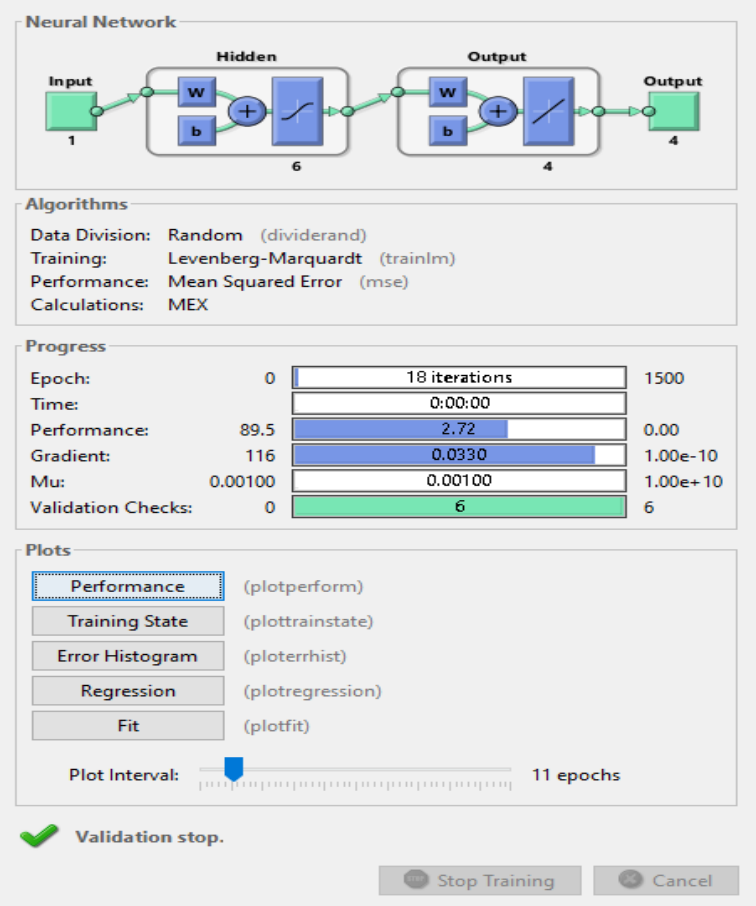


Figure 5.1: MATLAB Neural Network Toolbox Results With Validation Checks

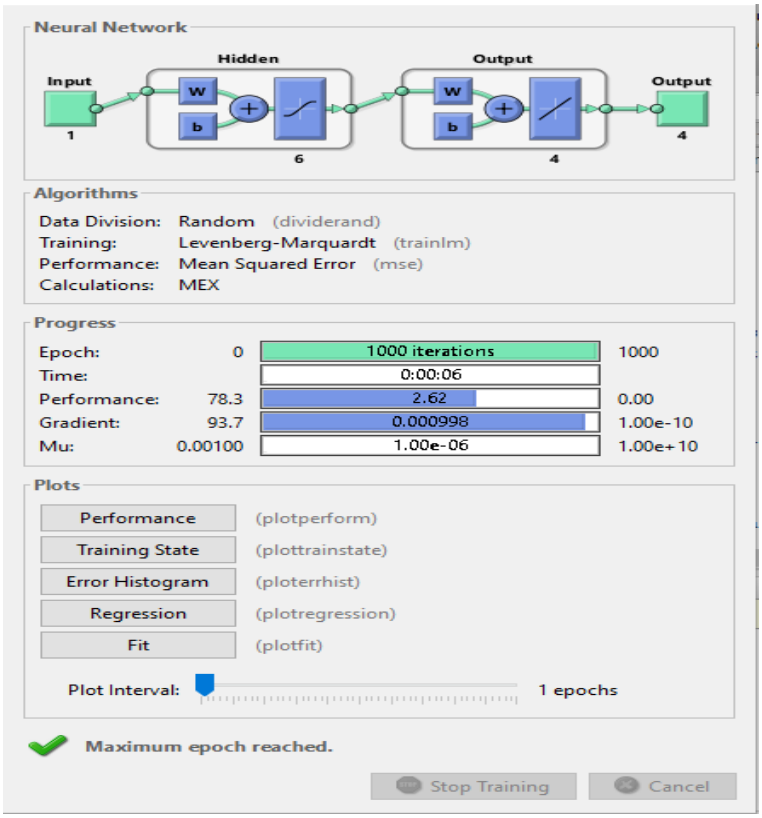


Figure 5.2: MATLAB Neural Network Toolbox Results Without Validation Checks

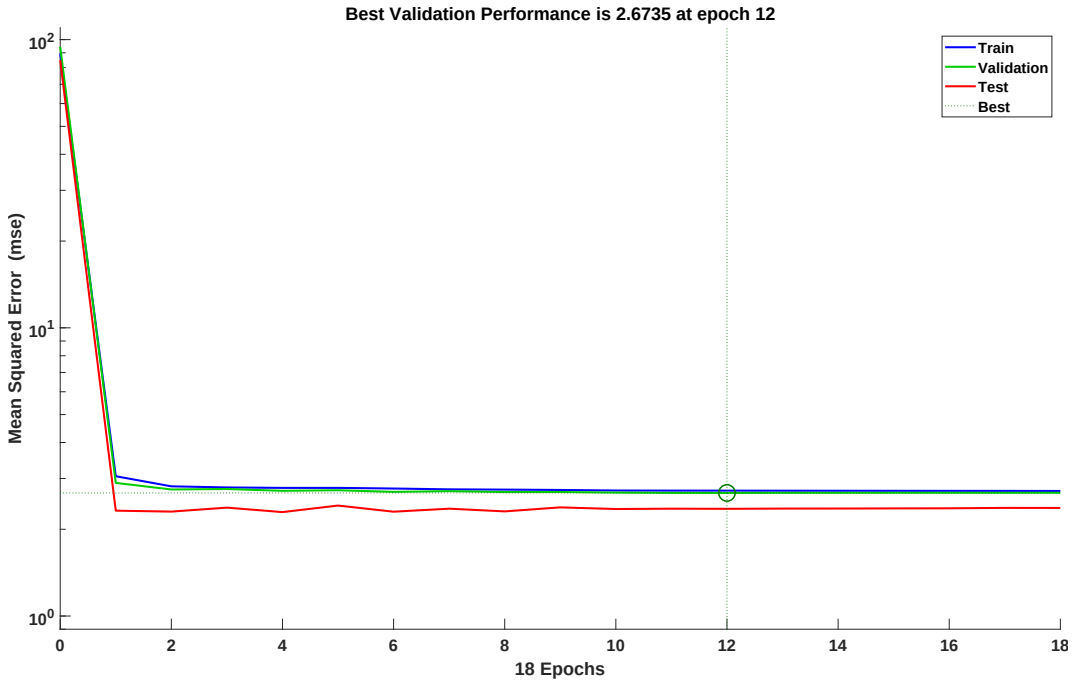


Figure 5.3: MATLAB Neural Network Error V.S. Epoch

5.1.2 Optimizer

The optimizer is designed without using the MATLAB genetic tool because the tool is not optimized for the scheduling genetic algorithm. The design requires forecaster data to schedule the appliances. The scheduling also requires the **PV** power and stored battery power. The algorithm covers the power requirement for appliances with **PV** or battery power when available. The algorithm also attempts to charge the batteries when **PV** power is available, as shown in Figure 5.4, thus covering all the battery power from **PV**. **GA** covers the **HVAC** system differently when it controls the thermostat settings depending on time-of-day pricing and comfort (Figure 5.5).

It is crucial to charge the battery with **PV** power or when the grid power is inexpensive. The battery's main purpose is to alleviate time-of-day pricing constraint by using differing loads in peak hours. Moreover, a non-interrupt load, such as HVAC system, can run consistently without driving up the price. This can be seen in Figure 5.5. Categorizing an HVAC system as an interruptible or non-interrupt system largely depends on the region, but the effect of the HVAC system is apparent. The thermostat settings do shoot up when it is expansive, and yet this is when the HVAC system is sorely needed is some regions. This area needs further investigation.

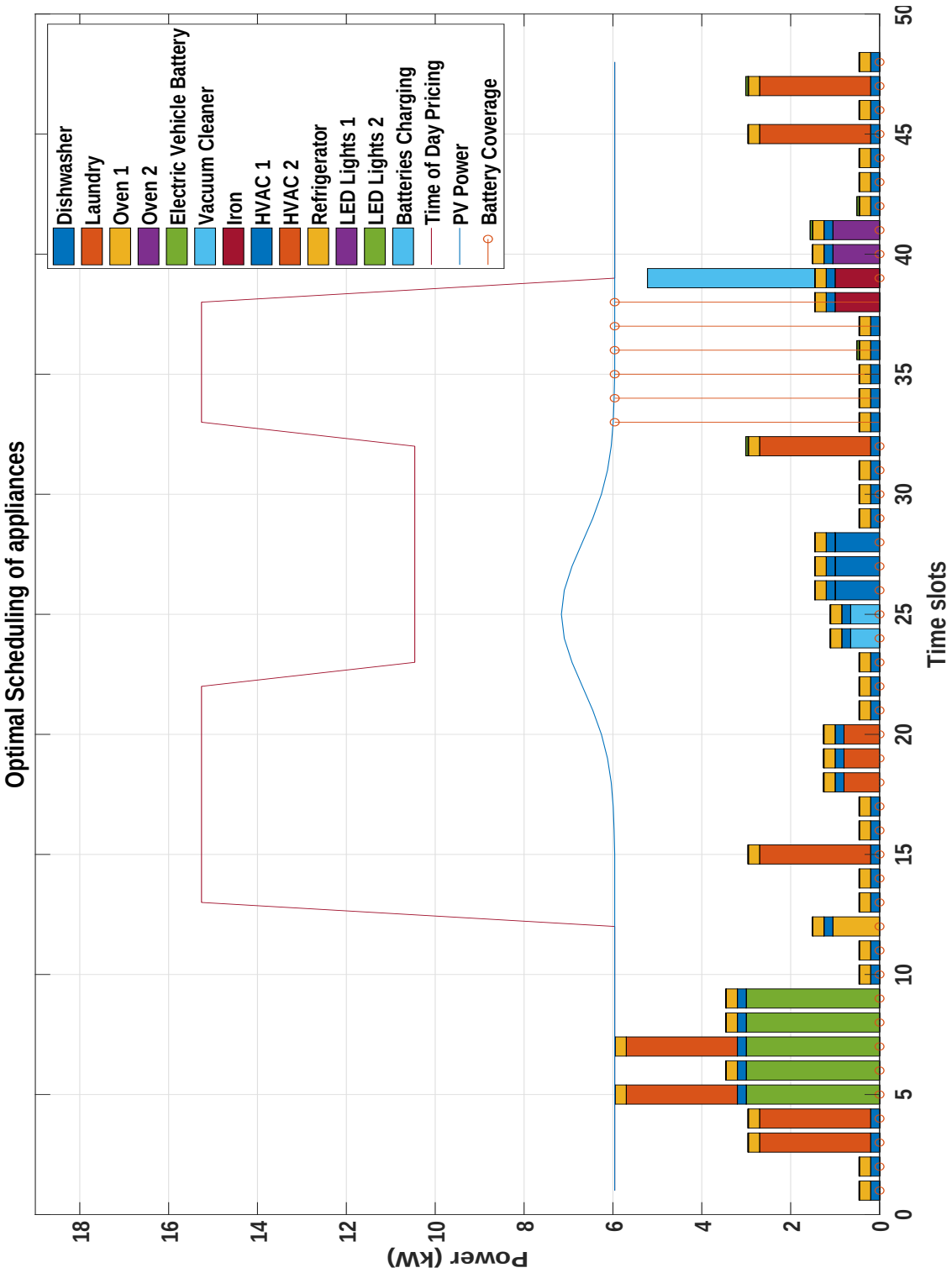


Figure 5.4: MATLAB Scheduling Genetic Algorithm Results

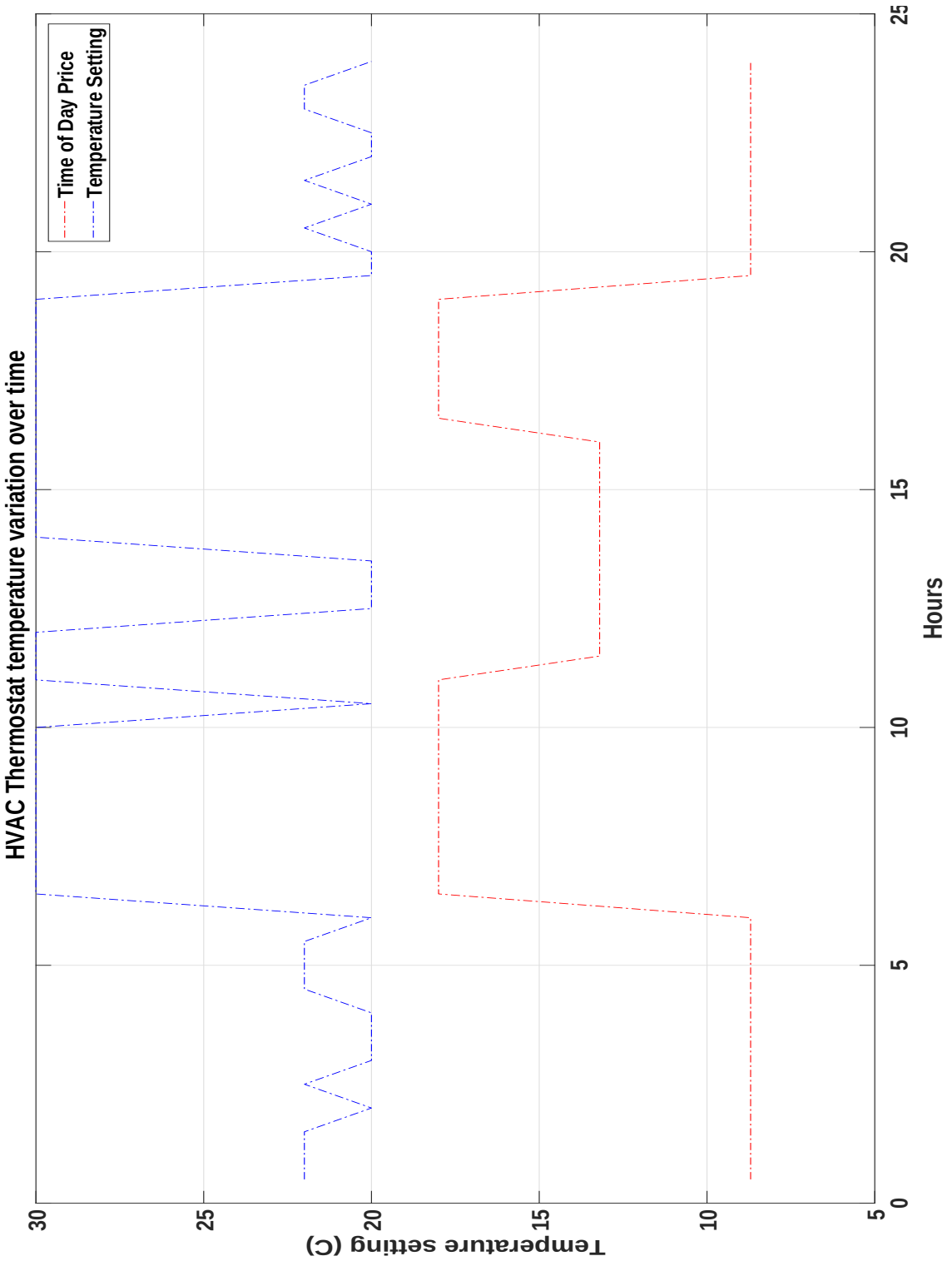


Figure 5.5: MATLAB Scheduling Genetic Algorithm Temperature Results

5.2 Implementation

The implementation has been conducted by using the Python language, PyTorch library and the Python **GUI**. The software design builds from the knowledge acquired in the simulation.

5.2.1 Forecaster

The forecaster is implemented with the knowledge that most house holds do not store past data. The software trains 24 hours of stored data, predicts and schedules the next 24 hours of data. Issues occur in the forecaster training because 24 hours does not represent seasonal changes or weekday vs. weekend patterns. The **LSTM** is a powerful tool because the algorithm can distinguish those patterns in the future.

In the implementation experiment, when the data was divided into 24 hours data segments and trained using an **ANN**, the results are not promising. Small patterns tended to disappear. The weekday results are proven to be more accurate than the weekends, and some seasons had better results than others. In contrast, the **LSTM** training results show more consistency across the weeks and seasons. The training occurs over two years' worth of data. A larger data set is required to prove that the **LSTM** training can predict most patterns and to confirm that an **ANN** prediction will not improve with a large set. However, the experiment demonstrates that for the small data, **LSTM** training accuracy outperforms **ANN**.

The algorithm is focused on the HVAC, heater (heat pump), dishwasher, washer and dryer because they are interruptable and deferrable. All non-interruptable appliances are not included in the training set.

Figure 5.7 shows the training of the **LSTM** software, where the data is introduced in 24 hour segments. The figure reveals that the **LSTM** is learning from

every segment added to the trainer. In the earlier iterations, the errors rate is increasing. Slowly the algorithm starts to identify patters by the end of the training. Figure 4.4 displays an example of how the algorithm predicts the time that the appliances are working, prompting the genetic algorithm to schedule the appliances in off-peak times. Whereas Figure 4.4 is an example of the days the appliances are in use, Figure 5.6 is an example of the days the deferrable appliances are not in use.

name	kW	type	time	LL	UL
Dryer	1.5	0	0	0	24
Washer	0.8	0	0	8	24
Dishwasher	1	0	0	0	24
Heat Pump	1.0	3	4.86	0	24
Electric Vehicle Battery	3	0	2.5	0	24
Vacuum Cleaner	0.65	0	1	7	19
Iron	1	0	1	10	20
HVAC 1	0.3	1	2	0	24
HVAC 2	2.75	2	8.5	1	24
Refrigerator	0.25	1	24	0	24
LED Lights 1	0.01	1	24	0	24
LED Lights 2	0.05	2	2.5	16	24

Figure 5.6: Results Example

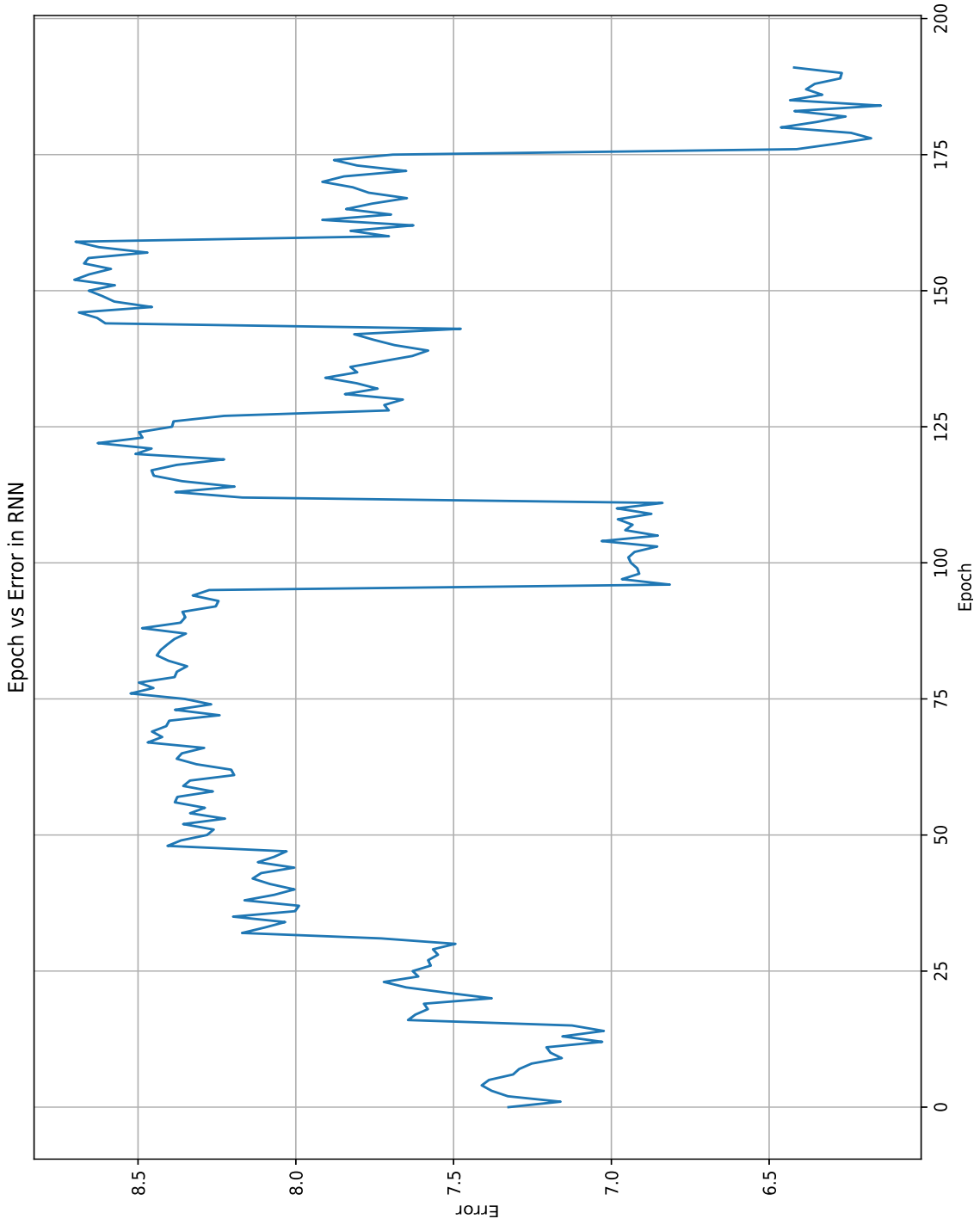


Figure 5.7: Implementation Forecaster Training Data

5.2.2 Optimizer

There are minimum changes in the **GA** algorithm from Section 5.1.2. The algorithm has been rewritten in Python. The results from the software can be seen in Figure 4.6 and Figure 4.7. These scheduling results are when the appliances are in use. Figure 5.9 shows the scheduling results when the deferrable loads are not in use. It also demonstrates the ability of the battery storage to cover the expensive areas when power intensive appliances are not running.

A comparison of Figure 4.5 and Figure 5.8 reveals the power required to offset non-interruptible loads. The algorithm optimizes the stored power to manage non-interruptible but deferrable load. When these loads are used in the expensive areas, batteries can offset them by covering the energy that the appliances require via stored power or by differing them to less expensive time slots.

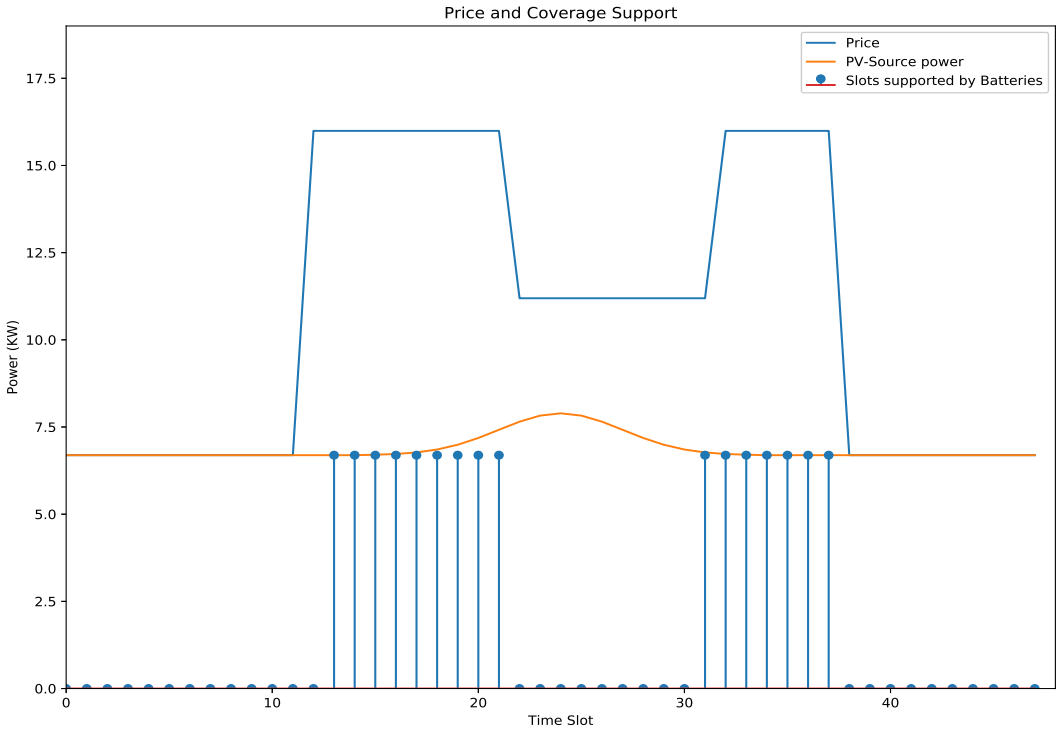


Figure 5.8: System Schedule with no Interruptible Appliances In Use Battery Coverage Example

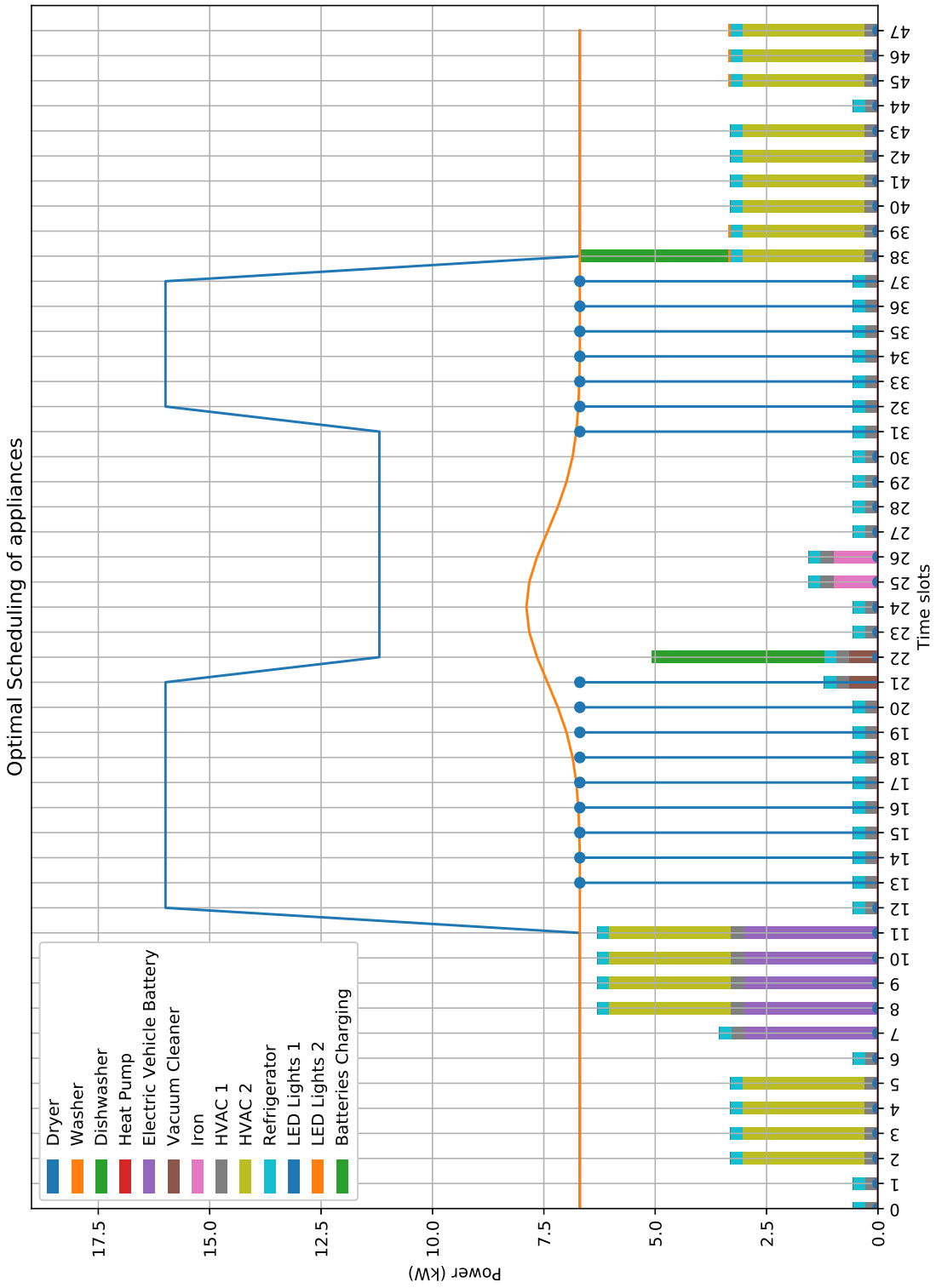


Figure 5.9: System Schedule With No Interruptible Appliances In Use Example

5.3 Limitations

The system struggles to overcome some limitations. It requires a large data set. Although the current system has been fed with two years' worth of data, some patterns still elude the forecaster. This highlights the need for the system to run for an extended time to achieve excellent forecasting.

The forecaster also does not include appliances settings. Another deficiency of the system is that the most appliances have major differences in their settings. Although the algorithm is capable of scheduling appliances at the peak time if battery or **PV** power is available, it could schedule more appliances at the same time with their lower settings. This forces management to all appliances by a single manufacturer. A complete system can mange differences settings.

The software does not have built in **PV** power prediction, thus limiting its coverage accuracy. The current software is built only with historical data. It was difficult to retrieve historical weather data because the data location is confidential. Weather data can be fed to the forecaster to improve **PV** coverage in the future.

Data cannot be extracted directly from the power meter, but must be processed through the Newton-phaselet. Even though the algorithm is designed to identify appliances it is not perfect.

The simulation and the implementation are not proven at an actual site. Consequently, the forecaster network does not train the changing patterns occurring from the optimizer.

A communication standard needs to be developed to allow appliances and devices from different manufactures to be scheduled. With the rapid development of the Internet of Things (**IoT**), there should be a communication protocol that can deliver the scheduling results to the appliances.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The present forecasting model in small smart grid utilizes the Long Short-Term algorithm to learn patterns and behavior in real-time with or without historical data. The main advantage of this forecasting model is the possibility to adapt to changes in home appliances configuration, as well as to changes in the appliance's technologies. Although there are limitations related to the short interval for data collection, the experiments demonstrate that the developed algorithm produces an accurate forecasting of power demand during working weekdays.

The simulation and implementation reveal that artificial intelligence (**AI**) algorithm can assist in forecasting and scheduling appliances at the residential level. The forecaster can utilize the processed power meter data to predict future energy usage. The **AI** algorithms can then use the predictions to schedule appliances to time-of-day prices for the lowest price. This achieves automated shifting of power even when the price changes daily. The purpose for the residents is to learn and adapt their behavior, while the developed software delivers the lowest price without sacrificing user's comfort. In addition to maximizing savings and comfort, the software incorporates photovoltaic power generation and power storage. Thus, the main contribution of the thesis is the development and implementation of forecaster and optimizer algorithms.

6.2 Future Work

To improve the algorithm developed, the limitations described in Section 5.3 need to be overcome. First of all, the system requires larger data sets, actually more extensive than the current data set collected during two years. Secondly, although the present research demonstrates that the real-world data simulations are capable of scheduling loads, the system could benefit in inclusion of real-world variables. This can be achieved through the implementation and validation of the system at an actual site. Furthermore, a communication protocol standard needs to be investigated to deliver the **AI** results to different appliances manufacturers. Weather data also should be injected to the forecaster to predict how much renewable energy can be used for each day.

Bibliography

[1] J. Momoh, *Smart grid: fundamentals of design and analysis*. John Wiley & Sons, 2012, vol. 63.

[2] W. J. Hausman and J. L. Neufeld, "Time-of-day pricing in the us electric power industry at the turn of the century," *The RAND Journal of Economics*, pp. 116–126, 1984.

[3] A. Ahmad, M. Hassan, M. Abdullah, H. Rahman, F. Hussin, H. Abdullah, and R. Saidur, "A review on applications of ann and svm for building electrical energy consumption forecasting," *Renewable and Sustainable Energy Reviews*, vol. 33, pp. 102–109, 2014.

[4] H.-x. Zhao and F. Magoulès, "A review on the prediction of building energy consumption," *Renewable and Sustainable Energy Reviews*, vol. 16, no. 6, pp. 3586–3592, 2012.

[5] Y.-S. Lee and L.-I. Tong, "Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming," *Knowledge-Based Systems*, vol. 24, no. 1, pp. 66–72, 2011.

[6] D. Zhang, S. Li, M. Sun, and Z. O'Neill, "An optimal and learning-based demand response and home energy management system," *IEEE Transactions on Smart Grid*, vol. 7, no. 4, pp. 1790–1801, 2016.

[7] B. Zhou, W. Li, K. W. Chan, Y. Cao, Y. Kuang, X. Liu, and X. Wang, "Smart home energy management systems: Concept, configurations, and scheduling strategies," *Renewable and Sustainable Energy Reviews*, vol. 61, pp. 30–40, 2016.

[8] P. Cappers, C. Goldman, and D. Kathan, "Demand response in us electricity markets: Empirical evidence," *Energy*, vol. 35, no. 4, pp. 1526–1535, 2010.

[9] N. A. E. R. Corporation., "Demand Response Discussion for the 2007 Long-Term Reliability Assessment., year = Feb. 2007 (accessed April 10, 2019), how-published = `https://www.naesb.org//pdf2/dsmee052407w4.pdf`."

[10] C.-M. Chu, T.-L. Jong, and Y.-W. Huang, "A direct load control of air-conditioning loads with thermal comfort control," in *IEEE Power Engineering Society General Meeting, 2005.* IEEE, 2005, pp. 664–669.

[11] A. Gomes, C. Antunes, and A. Martins, "A multiple objective approach to direct load control using an interactive evolutionary algorithm," *IEEE Transactions on Power Systems*, vol. 22, no. 3, pp. 1004–1011, 2007.

[12] N. Ruiz, I. Cobelo, and J. Oyarzabal, "A direct load control model for virtual power plant management," *IEEE Transactions on Power Systems*, vol. 24, no. 2, pp. 959–966, 2009.

[13] C. Triki and A. Violi, "Dynamic pricing of electricity in retail markets," *4OR*, vol. 7, no. 1, pp. 21–36, 2009.

[14] S. Caron and G. Kesidis, "Incentive-based energy consumption scheduling algorithms for the smart grid," in *Smart grid communications (SmartGridComm), 2010 First IEEE international conference on.* IEEE, 2010, pp. 391–396.

[15] J. H. Yoon, R. Baldick, and A. Novoselac, "Dynamic demand response controller based on real-time retail price for residential buildings," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 121–129, 2014.

[16] A. Capasso, W. Grattieri, R. Lamedica, and A. Prudenzi, "A bottom-up approach to residential load modeling," *IEEE Transactions on Power Systems*,

vol. 9, no. 2, pp. 957–964, 1994.

[17] P. Pijnenburg, "Non-intrusive load modeling using newton-phaselet frames," Ph.D. dissertation, University of New Brunswick, 2015.

[18] D. H. McQueen, P. R. Hyland, and S. J. Watson, "Monte carlo simulation of residential electricity demand for forecasting maximum demand on distribution networks," *IEEE Transactions on power systems*, vol. 19, no. 3, pp. 1685–1689, 2004.

[19] J. Boggis, "Diversity, bias and balance," *Distribution of Electricity*, pp. 357–362, 1953.

[20] S. Arora and J. W. Taylor, "Short-term forecasting of anomalous load using rule-based triple seasonal methods," *IEEE transactions on Power Systems*, vol. 28, no. 3, pp. 3235–3242, 2013.

[21] D. He, T. Habetler, M. J. Mousavi, and N. Kang, "A zip model-based feeder load modeling and forecasting method," in *Power and Energy Society General Meeting (PES), 2013 IEEE*. IEEE, 2013, pp. 1–5.

[22] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[23] Glosser.ca, "Artificial neural network," https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/851px-Colored_neural_network.svg.png, 28 February 2013 (accessed April 10, 2019).

[24] L. Medsker and L. C. Jain, *Recurrent neural networks: design and applications*. CRC press, 1999.

[25] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent

neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[26] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[29] J. Schmidhuber and A. Graves, "Framewise phoneme classification with bidirectional lstm networks," in *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 4. IEEE, 2005, pp. 2047–2052.

[30] C. Olah, "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 2015 (accessed April 10, 2019).

[31] A. Javed, H. Larijani, A. Ahmadinia, and R. Emmanuel, "Comparison of the robustness of rnn, mpc and ann controller for residential heating system," in *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*. IEEE, 2014, pp. 604–611.

[32] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[33] M. B. Wall, "A genetic algorithm for resource-constrained scheduling," Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[34] Z. Zhao, W. C. Lee, Y. Shin, and K.-B. Song, "An optimal power scheduling method for demand response in home energy management system," *IEEE Transactions on Smart Grid*, vol. 4, no. 3, pp. 1391–1400, 2013.

[35] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Springer Science & Business Media, 2007.

[36] S. C. G. C. Adam Paszke, Sam Gross, "Pytorch," https://pytorch.org/, 2017 (accessed April 10, 2019).

[37] B. Lorica, "Why ai and machine learning researchers are beginning to embrace pytorch," https://www.oreilly.com/ideas/why-ai-and-machine-learning-researchers-are-beginning-to-embrace-pytorch/, August 3 2017 (accessed April 10, 2019).

[38] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2657–2661.

[39] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014, pp. 285–290.

[40] E. Pei, L. Yang, D. Jiang, and H. Sahli, "Multimodal dimensional affect recognition using deep bidirectional long short-term memory recurrent neural networks," in *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*. IEEE, 2015, pp. 208–214.

[41] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2012.

[42] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[43] C. Darken and J. Moody, "Towards faster stochastic gradient search," in *Advances in neural information processing systems*, 1992, pp. 1009–1016.

[44] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[45] M. I. Lourakis, "A brief description of the levenberg-marquardt algorithm implemented by levmar," *Foundation of Research and Technology*, vol. 4, no. 1, pp. 1–6, 2005.

Appendices

Appendix A

MATLAB Training Code

```
close all
clear;clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% import data
x = importdata('Classification_Power_Day_time_all.csv');
y = importdata('Classification_Classes_Day.csv');
xtime = importdata('Classification_fullData_Day.csv');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X = x.data; %% removing time
Y = y.data; %% removing time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xt = xtime.data(:,1); %% removing time
% convert time from a string to float
t = datetime(xt,'ConvertFrom','posixtime','Format','HH.mm');
S = char(t);
time = str2num(S);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
divdier = 2; %% Data list divdier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
columnSamples = false; % samples are by columns.
cellTime = true;     % time-steps in matrix, not cell array.
hiddenLayerSize = 6; % numnber of hidden layers
net = fitnet(hiddenLayerSize); % construct the network
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.divideParam.trainRatio = 70/100; % the percentage of the data
    being trained
net.divideParam.valRatio = 15/100; % the percentage of the
    valation date
net.divideParam.testRatio = 15/100; % the percentage of the
    testing data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.trainFcn = 'trainlm'; % set the training fuction Levenberg-
    Marquardt backpropagation
net.trainParam.time = inf; % set the time to infinite or until
    the condition are met
net.trainParam.epochs = 1500; % set the max iteration
net.trainParam.mu = 1e-3; % set the learning rate
net.trainParam.mu_dec = 0.1; % set the learning rate decay
net.trainParam.mu_inc = 10; % set the learning rate increase
net.trainParam.mu_max = 1e10; % set the maxumam learning rate
net.trainParam.min_grad = 1e-10; % set the minumam learning reate
net.trainParam.goal = 0; % set the the error goal
net.trainParam.max_fail = 6;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Divide the full data deal with data
```

```
X_train=X(1:(end/divdier),2);
Y_train=Y(1:(end/divdier),:);
%%%%%%%%%%%%%%%%%%
X_test = X(((end/divdier)+1):end,2);
Y_test = Y(((end/divdier)+1):end,:);
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert the data to a cell
[XX,wasMatrixX] = tonndata(X_train,columnSamples,cellTime);
[YY,wasMatrixY] = tonndata(Y_train,columnSamples,cellTime);
[XX_test,wasMatrixX] = tonndata(X_test,columnSamples,cellTime);
[YY_test,wasMatrixX] = tonndata(Y_test,columnSamples,cellTime);
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net = configure(net,XX,YY) % configure the network
view(net) % view the network
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[net,tr] = train(net,XX,YY); % train the network
% Test the Network
outputs = net(XX);
%%%%%%%%%%%%
% evaluate the error and preformance
errors = gsubtract(YY,outputs);
plotperform(tr);
performance = perform(net,YY,outputs)
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% use the test data to evalute the training
output_test = net(XX_test);
errors_test = gsubtract(YY_test,output_test);
performance_test = perform(net,YY_test,output_test)
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Output = fromnndata(outputs,wasMatrixY,columnSamples,cellTime); %
    convert back the cell to a matrix
figure(1);
plotregression(Y_train,Output,'Regression') % plot a figure
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Output_test = fromnndata(output_test,wasMatrixY,columnSamples,
   cellTime); % convert back the cell to a matrix
figure(2);
plotregression(Y_test,Output_test,'TestRegression')  % plot a
   figure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trun the test data the actual data in one matrix
T = table(Y_train,round(Output));
TT = table(Y_test,round(Output_test));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% export the data in csv format
writetable(T,'results_matlab_Regression_thesis.csv');
writetable(TT,'test_results_matlab_Regression_thesis.csv');
```

Listing A.1: MATLAB Training Code

Appendix B

Forecaster Python Code

```
import os.path
import pickle
import matplotlib.pyplot as plt
import numpy as np
import pandas
import torch
from torch.autograd import Variable
from torch.optim.lr_scheduler import ReduceLROnPlateau
from Full.dynamicnet import DynamicNet

class Forecasting(object):
  ##
  #Initiate the forecaster
  # Input:
  # 1- hours: identifies the hours ahead to forecast
  # 2- inputSize: the number of input included in the training
  # 3- layer_number: the number of recurrent layers
  # 4- H_layers: number of hidden layers
  # 5-  x_file: the input data location
  # 6-  y_file: the output data location
  # 7- network: type of network false for lstm, true for ann
  # Output:
  # the the trained equation
  ##
    def __init__(self, hours, inputSize, layer_number, H_layer,
    x_file, y_file, network=False):
        self.hours = hours
        self.inputSize = inputSize
        self.layer_number = layer_number
        self.H_layer = H_layer
        self.network = network  # False = LSTM, True = ANN, The
    algorithm defaults to LSTM

        self.x_data = None
        self.y_data = None
        self.hidden = None
        self.c0 = None
        self._setup(x_file, y_file)

    def _setup(self, x_file, y_file,):

        self._setup_rnn()
        self._load_data(x_file, y_file)


    def _load_data(self, x_file, y_file):
```

```
45          if self.network is True:
46              x_csv = pandas.read_csv(x_file)
47              self.x_rows = x_csv.shape[0]
48              x_csv = np.resize(x_csv.values, (self.x_rows, self.
    inputSize))
49              self.x_data = torch.Tensor(x_csv)
50
51              y_csv = pandas.read_csv(y_file)
52              y_csv = np.resize(y_csv.values, (y_csv.shape[0], self
    .H_layer))
53              self.y_data = torch.Tensor(y_csv)
54          else:
55              x_csv = pandas.read_csv(x_file)
56              self.x_rows = x_csv.shape[0]
57              x_csv = np.resize(x_csv.values, (self.x_rows, self.
    hours, self.inputSize))
58              self.x_data = torch.Tensor(x_csv)
59
60              y_csv = pandas.read_csv(y_file)
61              y_csv = np.resize(y_csv.values, (y_csv.shape[0], self
    .H_layer))
62              self.y_data = torch.Tensor(y_csv)
63
64      def _load_chunk(self, chunk):
65
66          if self.network is True:
67              _D_in = chunk * self.hours
68              _D_out = (chunk + 1) * self.hours
69              x_chunk = self.x_data[_D_in:_D_out, :]
70              y_chunk = self.y_data[_D_in:_D_out, :]
71              return x_chunk, y_chunk
72          else:
73              _D_in = chunk * self.hours
74              _D_out = (chunk + 1) * self.hours
75              x_chunk = self.x_data[_D_in:_D_out, :, :]
76              y_chunk = self.y_data[_D_in:_D_out, :]
77              return x_chunk, y_chunk
78
79      def _run_chunk(self, chunk_num, x_lin=False):
80          if self.network is True:
81              x, y = self._load_chunk(chunk_num)
82              return self._linearization(x, y, x_lin)
83          else:
84              x, y = self._load_chunk(chunk_num)
85              return self._linearization(x, y, x_lin)
86
87      def _run_fullChunk(self, x_lin=False):
88          if self.network is True:
89              x = self.x_data
90              y = self.y_data
91              return self._linearization(x, y, x_lin)
92          else:
93              x = self.x_data[:, :, :]
94              y = self.y_data[:, :]
95              return self._linearization(x, y, x_lin)
96
97      def _linearization(self, chunk_x, chunk_y, x_lin=False):
```

```
98
99          if x_lin:
100             inputs = (chunk_x - self.x_data.min()) / (self.x_data
    .max() - self.x_data.min())
101         else:
102             inputs = chunk_x
103
104         outputs = ((chunk_y - self.y_data.min()) / (self.y_data.
    max() - self.y_data.min()))
105
106         return inputs, outputs
107
108     def _delinearization(self, y_predicted):
109
110         y_final = abs(((y_predicted * (self.y_data.max() - self.
    y_data.min())) + self.y_data.min()))
111
112         return y_final
113
114         return inputs, outputs
115
116     def _setup_rnn(self, learning_rate=1e-8, momentum=0.25,
    nesterov=True, batch_first=True, dropout=0.5):
117
118         if self.network is True:
119             self._setup_ann(learning_rate=learning_rate, momentum
    =momentum, nesterov=nesterov)
120         else:
121             # Save data for re-use
122             _rnn_data = 'lstm.dump'
123             _hidden_data = 'hidlstm.dump'
124
125             if os.path.isfile(_hidden_data):
126                 # Load our saved parameters
127                 with open(_hidden_data, 'rb') as dump_file:
128                     _h_block = pickle.load(dump_file)
129                     (hidden, c0) = _h_block
130             else:
131                 # No saved data, create new
132                 hidden = Variable(torch.randn(self.layer_number,
    self.hours, self.H_layer))
133                 c0 = Variable(torch.randn(self.layer_number, self
    .hours, self.H_layer))
134
135             # Create or load the RNN
136             if os.path.isfile(_rnn_data):
137                 # Load our saved parameters
138                 with open(_rnn_data, 'rb') as dump_file:
139                     self.rnn = pickle.load(dump_file)
140             else:
141                 # No saved data, create new
142                 self.rnn = torch.nn.LSTM(input_size=self.
    inputSize, hidden_size=self.H_layer, num_layers=self.
    layer_number,
143                                          bias=True,
144                                          batch_first=batch_first,
     bidirectional=False, dropout=dropout)
145
```

```
146             # Implements stochastic gradient descent with
    Nesterov momentum.
147             self.optimizer = torch.optim.SGD(self.rnn.parameters
    (), lr=learning_rate, momentum=momentum, nesterov=nesterov)
148
149             # Reduce learning rate when a mean squared error has
    stopped improving
150             self.scheduler = ReduceLROnPlateau(self.optimizer, '
    min', patience=0, threshold=1e-1,
151                                                factor=0.90,
    min_lr=1e-25, verbose=False)
152             self.hidden = hidden
153             self.c0 = c0
154             self.loss_fn = torch.nn.MSELoss(size_average=False)
155
156     def _setup_ann(self, learning_rate=1e-4, momentum=0.25,
    nesterov=True):
157
158             # Save data for re-use
159             _ann_data = 'ann.dump'
160
161             # Create or load the RNN
162             if os.path.isfile(_ann_data):
163                 # Load our saved parameters
164                 with open(_ann_data, 'rb') as dump_file:
165                     self.ann = pickle.load(dump_file)
166             else:
167                 # No saved data, create new
168                 self.ann = DynamicNet(self.inputSize, self.
    H_layer, self.layer_number)
169
170             #for param in self.ann.parameters():
171             #    param.data = ((torch.randn(param.data.size()) -
    0.5) * 0.1 * 2)
172
173             # Implements stochastic gradient descent with
    Nesterov momentum.
174             self.optimizer = torch.optim.SGD(self.ann.parameters
    (), lr=learning_rate, momentum=momentum,
175                                              nesterov=nesterov)
176
177             # Reduce learning rate when a mean squared error has
    stopped improving
178             self.scheduler = ReduceLROnPlateau(self.optimizer, '
    min', patience=0, threshold=1e-1,
179                                                factor=0.90,
    min_lr=1e-25, verbose=False)
180
181             self.loss_fn = torch.nn.MSELoss(size_average=False)
182
183
184     def load_forecast(self, row=None):
185
186         if self.network is True:
187             if row is None:
188                 y_pred = self.ann(self.x_data)
189             else:
```

```
                y_pred = self.ann(self.x_data[(row * self.hours)
    :((row + 1) * self.hours), :])

            return abs(((y_pred * (self.y_data.max() - self.
    y_data.min())) + self.y_data.min()))
        else:
            if row is None:
                y_pred, h_n = self.rnn(self.x_data, (self.hidden,
     self.c0))
            else:
                y_pred, h_n = self.rnn(self.x_data[(row * self.
    hours):((row + 1) * self.hours), :, :],
                                       (self.hidden, self.c0))

            with open('hidlstm.dump', 'wb') as f:
                pickle.dump(h_n, f)

            y_pred = y_pred[-1]

            return abs(((y_pred * (self.y_data.max() - self.
    y_data.min())) + self.y_data.min()))

    def run_full(self, iterations=10):
        global y_pred
        errors = []
        if self.network is True:
            inputs, outputs = self._run_fullChunk()
            y_pred = self._train(errors, inputs, outputs,
    iterations)
            self._save_ann()
            # y_final = self._plot(errors, y_pred)
            return self._plot(errors, y_pred)
        else:
            inputs, outputs = self._run_fullChunk()
            y_pred, h_n = self._train(errors, inputs, outputs,
    iterations)
            self._save_rnn(h_n)
            # y_final = self._plot(errors, y_pred)
            return self._plot(errors, y_pred)


    def run_all(self, iterations=10):
        global y_pred
        errors = []
        if self.network is True:
            for chunk_num in range(int((self.x_rows / 2) / self.
    hours)):
                inputs, outputs = self._run_chunk(chunk_num)
                y_pred = self._train(errors, inputs, outputs,
    iterations)
                self._save_ann()
            # y_final = self._plot(errors, y_pred)
            return self._plot(errors, y_pred)
        else:
            for chunk_num in range(int((self.x_rows / 2) / self.
    hours)):
                inputs, outputs = self._run_chunk(chunk_num)
```

```
237                 y_pred, h_n = self._train(errors, inputs, outputs
    , iterations)
238                 self._save_rnn(h_n)
239             # y_final = self._plot(errors, y_pred)
240             return self._plot(errors, y_pred)
241
242     def run(self, iterations=10):
243         errors = []
244
245         if self.network is True:
246             inputs, outputs = self._linearization(self.x_data[0:
    self.hours, :], self.y_data[0:self.hours, :],
247                                                   x_lin=False)
248             y_pred_ann = self._train(errors, inputs, outputs,
    iterations)
249             self._save_ann()
250             self._plot(errors, y_pred_ann)
251             return self._plot(errors, y_pred_ann)
252         else:
253             inputs, outputs = self._linearization(self.x_data[0:
    self.hours, :, :], self.y_data[0:self.hours, :],
254                                                   x_lin=False)
255             y_pred, h_n = self._train(errors, inputs, outputs,
    iterations)
256             self._save_rnn(h_n)
257             self._plot(errors, y_pred)
258             return self._plot(errors, y_pred)
259
260     def _train(self, errors, inputs, outputs, num_iter):
261
262         global y_pred, h_n
263         if self.network is True:
264             for epoch in range(num_iter):
265                 # Forward pass: compute predicted y_pred, the
    next hidden layer h_n
266                 # by passing inputs, hidden and C0 to the model.
267                 y_pred = self.ann(inputs)
268
269                 # Calculate the mean square error between the
    real output and the predicted output
270                 loss = self.loss_fn(self._delinearization(y_pred)
    , self._delinearization(outputs))
271
272                 prev_lr = self.optimizer.param_groups[0]['lr']
273                 # Before the backward pass, use the optimizer
    object to zero all of the
274                 # gradients for the variables it will update (
    which are the learnable weights of the model)
275                 self.optimizer.zero_grad()
276
277                 # Backward pass: compute gradient of the loss
    with respect to model parameters
278                 loss.backward(retain_graph=True)
279
280                 # Calling the step function on an Optimizer makes
     an update to its parameters
281                 self.scheduler.step(loss.item(), epoch)
```

```
282
283                 # Fine control the learning rate
284                 new_lr = self.optimizer.param_groups[0]['lr']
285
286                 if prev_lr == new_lr:
287                     self.optimizer.param_groups[0]['lr'] = new_lr
    * 1.0001
288
289                 self.optimizer.step()
290
291                 # Print epoch for debugging
292                 if epoch % 5 == 0:
293                     print(epoch, loss.item())
294
295                 # Keep a list of errors for plotting later
296                 errors.append(loss.item())
297
298             return y_pred
299         else:
300             for epoch in range(num_iter):
301                 # Forward pass: compute predicted y_pred, the
    next hidden layer h_n
302                 # by passing inputs, hidden and C0 to the model.
303                 y_pred, h_n = self.rnn(inputs, (self.hidden, self
    .c0))
304
305                 # Getting the last output of the predicted y_pred
306                 y_pred = y_pred[-1]
307                 # Calculate the mean square error between the
    real output and the predicted output
308                 #loss = self.loss_fn(self._delinearization(y_pred
    ), self._delinearization(outputs))
309                 loss = self.loss_fn(y_pred, outputs)
310                 # Decompose the hidden layer h_n
311                 (self.hidden, self.c0) = h_n
312
313                 prev_lr = self.optimizer.param_groups[0]['lr']
314                 # Before the backward pass, use the optimizer
    object to zero all of the
315                 # gradients for the variables it will update (
    which are the learnable weights of the model)
316                 self.optimizer.zero_grad()
317
318                 # Backward pass: compute gradient of the loss
    with respect to model parameters
319                 loss.backward(retain_graph=True)
320
321                 # Calling the step function on an Optimizer makes
     an update to its parameters
322                 self.scheduler.step(loss.item(), epoch)
323
324                 # Fine control the learning rate
325                 new_lr = self.optimizer.param_groups[0]['lr']
326
327                 if prev_lr == new_lr:
328                     self.optimizer.param_groups[0]['lr'] = new_lr
    * 1.0001
329
```

```
                    self.optimizer.step()

                    # Print epoch for debugging
                    if epoch % 5 == 0:
                        print(epoch, loss.item())

                    # Keep a list of errors for plotting later
                    errors.append(loss.item())

                return y_pred, h_n

    def _save_rnn(self, h_n):
        # Save data for re-use
        with open('lstm.dump', 'wb') as f:
            pickle.dump(self.rnn, f)

        with open('hidlstm.dump', 'wb') as f:
            pickle.dump(h_n, f)

    def _save_ann(self):
        # Save data for re-use
        with open('ann.dump', 'wb') as f:
            pickle.dump(self.ann, f)

    def _plot(self, errors, y_pred):

        y_final = abs(((y_pred * (self.y_data.max() - self.y_data
    .min())) + self.y_data.min()))

        # Plotting
        plt.figure()
        plt.plot(errors)
        plt.xlabel('Epoch')
        plt.ylabel('Error')
        plt.title('Epoch vs Error in RNN')
        plt.grid(True)
        return y_final
```

Listing B.1: Forecaster Python Code

Appendix C

Appliances Python Code

```
1
2 class Appliance(object):
3
4     def __init__(self, name, kW, type, time, LL, UL):
5         TODO::param name::param kW::param type::param time::param LL::param
      UL:
6         self.name = name
7         self.kW = kW
8         self.type = type
9         self.time = time
10        self.LL = LL
11        self.UL = UL
```

Listing C.1: Appliances Python Code

Appendix D

Genetic Algorithm Python Code

```
import copy
import random
from random import randint

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from prettytable import PrettyTable

from appliancescheduler.appliance import Appliance


def _calc_min(cost_data):
    # TODO: find function for this
    min_cost = 100000000
    row = 0
    col = 0
    for i in range(len(cost_data)):
        for j in range(cost_data[i].size):
            if min_cost > cost_data[i][j] > 0:
                min_cost = cost_data[i][j]
                row = i
                col = j
    return min_cost, row, col


def _crossover(probabilities):
    # two parents are selected by the roulette wheel method, the schedule which
    # has a lower cost has more probability of becoming a parent, parents are
    # replacable, i.e. the same schedule might become parent more than once.
    parents = np.zeros(2)
    par = 0
    while par < 2:
        r = 200
        sum_probabilities = sum(probabilities)

        while (r > sum_probabilities):
            r = random.uniform(0, 1)
        j = 0
        p = probabilities[j]

        while p < r:
            j = j + 1
```

```
44              p = p + probabilities[j]
45          parents[par] = j
46          par = par + 1
47      return parents
48
49
50  def _crossover_tourment_selection(cost, k):
51      parents = np.zeros(2)
52      i = 0
53      while i < 2:
54          samples = random.sample(range(len(cost)), k)
55          sub_cost = cost[samples]
56          pa = samples[np.argsort(sub_cost)[0]]
57          if i == 0:
58              parents[i] = pa
59              i = i + 1
60          elif pa != parents[0]:
61              parents[i] = pa
62              i = i + 1
63      return parents
64
65
66  class GeneticAlgorithm(object):
67
68      def __init__(self, appliances, price, ts, PV_power, bp, M,
    sols, total_it, pvPower=False):
69          self.appliance = appliances
70          self.N = len(self.appliance)
71          self.price = price
72          self.ts = ts
73          self.PV_power = PV_power
74          self.bp = bp
75          self.M = M
76          self.sols = sols
77          self.tot_it = total_it
78          self.PV = self._calc_PV(pvPower)
79
80      def __init__(self, appliances, price, PV_power, bp, M, sols,
    total_it, pvPower=False):
81          self.appliance = appliances
82          self.N = len(self.appliance)
83          self.price = price
84          self.PV_power = PV_power
85          self.bp = bp
86          self.M = M
87          self.ts = self._calc_ts()
88          self.sols = sols
89          self.tot_it = total_it
90          self.PV = self._calc_PV(pvPower)
91
92      def _calc_PV(self, pvPower):
93          PV = [GeneticAlgorithm.normpdf(i, round(12.30 * self.ts),
    3) for i in range(1, self.M + 1)]
94          PV = np.array(PV) * (self.PV_power / max(PV))
95          PV[np.where(PV < 0.01)[0]] = 0
96          if not pvPower:
97              PV[:] = 0
98          return PV
```

```
99
100     def _calc_ts(self):
101         ts = self.M / 24
102         return ts
103
104     @staticmethod
105     def initialization(comfort):
106
107         # Dryer Washer Dishwasher Heat_Pump
108         appliance = list()
109
110         appliance.append(Appliance('Dryer', 1.50, 0, 0.5, 0, 24))
111         appliance.append(Appliance('Washer', 0.8, 0, 1.50, 8, 24)
    )
112         appliance.append(Appliance('Dishwasher', 1, 0, 1.50, 0,
    24))
113         appliance.append(Appliance('Heat Pump', 1.0, 3, 1, 0, 24)
    )
114         appliance.append(Appliance('Electric Vehicle Battery', 3,
     0, 2.5, 0, 24))
115         appliance.append(Appliance('Vacuum Cleaner', 0.65, 0, 1,
    7, 19))
116         appliance.append(Appliance('Iron', 1, 0, 1, 10, 20))
117
118         if comfort == 1:
119             kW = 0.20
120         elif comfort == 2:
121             kW = 0.25
122         else:
123             kW = 0.30
124
125         appliance.append(Appliance('HVAC 1', kW, 1, 2, 0, 24))
126
127         if comfort == 1:
128             time = 4
129             kW = 2.5
130         elif comfort == 2:
131             time = 6
132             kW = 2.6
133         else:
134             time = 8.5
135             kW = 2.75
136
137         appliance.append(Appliance('HVAC 2', kW, 2, time, 1, 24))
138         appliance.append(Appliance('Refrigerator', 0.25, 1, 24,
    0, 24))
139         appliance.append(Appliance('LED Lights 1', 0.01, 1, 24,
    0, 24))
140         appliance.append(Appliance('LED Lights 2', 0.05, 2, 2.5,
    16, 24))
141
142         return appliance
143
144     def _temperature(self, final_solution):
145         temp = np.zeros(self.M)
146         matrix = final_solution[np.array([7, 8]), :]
147         wattage = np.zeros((2, self.M))
```

```
148         wattage[0, :] = self._get_appliance('HVAC 1').kW * matrix
    [0, :]
149         wattage[1, :] = self._get_appliance('HVAC 2').kW * matrix
    [1, :]
150         total_watt = np.sum(wattage, 0) - self.PV
151         temp[np.where(total_watt > 3)[0]] = 22
152         temp[np.where(total_watt[np.where(total_watt >= 2.85)[0]]
     <= 3)[0]] = 23
153         temp[np.where(total_watt[np.where(total_watt >= 2.7)[0]]
    < 2.85)[0]] = 24
154         temp[np.where(total_watt[np.where(total_watt > 0.25)[0]]
    < 0.5)[0]] = 26
155         temp[np.where(total_watt[np.where(total_watt > 0.20)[0]]
    <= 0.25)[0]] = 27
156         temp[np.where(total_watt <= 0.2)[0]] = 28
157         temp = temp * (self.price / 10.5)
158         temp[np.where(temp < 20)[0]] = 20
159         temp[np.where(temp > 30)[0]] = 30
160         temp = np.round(temp)
161
162         t = np.array(range(1, 49)) * 0.5
163         plt.figure()
164         plt.plot(t, temp, b, linestyle='dashed', dashes=[4, 2, 1,
    2])
165         plt.title('HVAC Thermostat temperature variation over
    time')
166         plt.xlabel(Hours)
167         plt.ylabel('Temperature setting (C)')
168         plt.xlim([0, 24])
169         plt.ylim([15, 40])
170         plt.grid()
171
172         plt.figure()
173         plt.plot(t, self.price, r, linestyle='dashed', dashes=[4,
    2, 1, 2], label='Price')
174         plt.plot(t, temp, b, linestyle='dashed', dashes=[4, 2, 1,
    2], label='Temperature Setting')
175         plt.legend(loc='best')
176         plt.grid()
177
178     def _costfunction2(self, schedule):
179
180         cost_vector = np.zeros(self.M)
181         n_1 = np.where(schedule[self.N, :])[0]
182         [total_watts, w] = self._watts(schedule)
183         power = self._checking(schedule)
184         total_watts[n_1] = total_watts[n_1] + power * self.ts
185         factor = np.ones(len(total_watts))
186         for i in range(self.M):
187             cost_vector[i] = (self.price[i] * (total_watts[i]) *
    (1.0 / self.ts)) * (1 - schedule[self.N + 1, i]) * \
188                              factor[i]
189         cost = np.sum(cost_vector)
190         return cost
191
192     def _mutation(self, child):
```

```
193         child[self.N: self.N + 2, :] = 0
194         for i in range(self.N):
195             type = self.appliance[i].type
196             length = len(np.where(child[i, :] > 0)[0])
197             total = round(self.ts * self.appliance[i].time)
198             if type == 1 or length == total:
199                 continue
200             if type == 0:
201                 if length < total:
202                     if length > 0:
203                         difference = total - length
204                         ind = np.where(child[i, :] > 0)[0]
205                         if ind.size > 0:
206                             child[i, ind[-1]: ind[-1] + int(
    difference)] = 1
207                     else:
208                         lower = self.ts * self.appliance[i].LL
209                         if lower == 0:
210                             lower = 1
211                         upper = self.ts * self.appliance[i].UL
212                         _randval = randint(lower, upper - total)
213                         child[i, int(_randval): int(_randval +
    total)] = np.ones(int(total))
214                 elif length > total:
215                     ind = np.where(child[i, :] > 0)[0]
216                     child[i, :] = 0
217                     child[i, int(ind[0]): int(ind[0] + total) -
    1] = 1
218             elif type == 2:
219                 difference = abs(total - length)
220                 ind = np.where(child[i, :] > 0)[0]
221                 if length < total:
222                     lower = self.ts * self.appliance[i].LL
223                     if lower == 0:
224                         lower = 1
225                     upper = self.ts * self.appliance[i].UL
226                     check = 0
227                     while check == 0:
228                         check2 = 0
229                         while check2 == 0:
230                             _randval = [randint(lower, upper - 1)
     for ii in range(int(difference))]
231                             if len(np.unique(_randval)) ==
    difference:
232                                 check2 = 1
233                         var = np.zeros(len(_randval))
234                         for k in range(len(_randval)):
235                             var[k] = len(np.where(_randval[k] ==
    ind)[0])
236                         if sum(var) == 0:
237                             check = 1
238                     child[i, _randval] = 1
239                     if len(np.where(child[i, :] > 0)[0]) < total:
240                         print('Error 1')
241                 elif length > total:
242                     num = [randint(1, len(ind)) for ii in range(
    int(difference))]
```

```
                    child[i, ind[np.array(num) - 1]] = 0
                    if len(np.where(child[i, :] > 0)[0]) < total:
                        print('Error 2')

        [total_watts, total_watts2] = self._watts(child)
        ccc = 0
        while ccc == 0:
            power = []
            index = []
            ind2 = []

            child[self.N: self.N + 2, :] = 0
            if self.bp <= 0:
                return child
            check = np.zeros(self.M)
            check3 = np.zeros(self.M)
            n = randint(1, 4)

            ind2 = np.zeros(n)
            for ii in range(n):
                batt = self.bp
                index = randint(1, self.M - 5)
                subind = np.where(check > 0)[0]
                c = 1
                if ii > 0:
                    check2 = 0
                    while check2 == 0:
                        if subind.size == 0:
                            c = 20
                            break
                        index = randint(1, self.M - 5)
                        if index < subind[0] - 5 or index > subind[-1] + 5:
                            check2 = 1
                        c = c + 1
                        if c > 15:
                            break
                if c > 15:
                    break
                ind2[ii] = index
                c5 = 0
                while batt > min(total_watts2) / self.ts:
                    if (total_watts[index] / self.ts) < batt and check[index] != 1 and check3[index] != 1 and \
                            total_watts[index] != 0:
                        child[self.N + 1, index] = 1
                        batt = batt - total_watts[index] / self.ts
                        check[index] = 1
                        c5 = 1

                    index = index + 1
                    if index == self.M or index + 1 in ind2[:ii - 1]:
                        break

                if c5 == 1:
                    if index == child.shape[1]:
```

```
                        index = 47
                    child[self.N, index] = 1
                    check3[index] = 1

            power = self._checking(child)
            A = np.where(child[self.N, :] > 0)[0]
            B = np.where(child[self.N + 1, :] > 0)[0]
            C = A[np.in1d(A, B)]
            if power.size > 0:
                if max(power) < self.bp and C.size == 0:
                    ccc = 1
        return child

    def _generateChild(self, parents, solution):
        # this function crosses over the two parents selected in crossover to
        # create a new child; however in order to ensure that the new child fulfils
        # all constraints of the system, mutation function is called which makes
        # minimum amendments in the child to make it a feasible child.
        crosspoint = randint(0, self.M)
        A = solution[int(parents[0])][:, 0: crosspoint]
        aa = A[:, :]
        A = aa
        B = solution[int(parents[1])][:, crosspoint:]
        bb = B[:, :]
        B = bb
        child = np.hstack((A, B))
        feasible_child = self._mutation(child)

        return feasible_child

    def _costfunction(self, schedule):
        # this function calculates the total cost in $ for the schedule using the price vector of the power company
        cost_vector = np.zeros(self.M)
        n_1 = np.where(schedule[self.N, :] > 0)[0]
        total_watts, ww = self._watts(schedule)
        power = self._checking(schedule)
        total_watts[n_1] = total_watts[n_1] + power * self.ts
        factor = np.ones(len(total_watts))

        ## Penalty for extra-wattages during costly hours ##
        high_price = np.where(self.price > (np.mean(self.price) * 1.1))[0]
        medium_price = np.where(self.price > (np.mean(self.price)))[0]
        factor[medium_price] = 14
        factor[high_price] = 50
        ##
        for i in range(self.M):
            cost_vector[i] = (self.price[i] * (total_watts[i]) * (1.0 / self.ts)) * (1.0 - schedule[self.N + 1, i]) * \
                             factor[
                                 i]
```

```
346         cost = sum(cost_vector)
347         return cost
348
349     def _checking(self, child):
350         # This function calculates the total power in kW that
    battery supports (& consumes in charging) as
351         # shown in the (self.N+2)th row (& (self.N+1)th row )of
    the schedule (or child)
352         total_watts, w = self._watts(child)
353         n_1 = np.where(child[self.N, :] > 0)[0]
354         n_2 = np.where(child[self.N + 1, :] > 0)[0]
355         nn = len(np.where(child[self.N, :] > 0)[0])
356         chunk = np.zeros((nn, n_2.size + 1))
357         power = np.zeros(nn)
358         c = np.zeros(nn)
359         for ii in range(nn):
360             if ii != 0:
361                 try:
362                     c[ii] = len(n_2[np.where(n_2[np.where(n_2 >
    chunk[ii - 1, int(c[ii - 1] - 1)])[0]] < n_1[ii])[0]])
363                     chunk[ii, 0: int(c[ii])] = n_2[
364                         np.where(n_2[np.where(n_2 > chunk[ii - 1,
     int(c[ii - 1] - 1)])[0]] < n_1[ii])[0]]
365                 except:
366                     power = 5e8
367                     return power
368             else:
369                 c[ii] = len(n_2[np.where(n_2 < n_1[ii])[0]])
370                 chunk[ii, 0: int(c[ii])] = n_2[np.where(n_2 < n_1
    [ii])[0]]
371
372             chunk = chunk.astype(np.int)
373             power[ii] = (np.sum(total_watts[chunk[ii, np.where(
    chunk[ii, :] > 0)[0]]])) / self.ts
374         return power
375
376     def _watts(self, schedule):
377         # TODO: check this function
378         wattage = np.zeros(self.N)
379         total_watts = np.zeros(self.M)
380         total_watts2 = np.zeros(self.M)
381
382         for j in range(self.N):
383             wattage[j] = self.appliance[j].kW
384
385         for i in range(self.M):
386             total_watts[i] = (wattage.dot(schedule[0: self.N, i])
    ) - self.PV[i]
387
388         for ii in range(self.M):
389             total_watts2[ii] = (wattage.dot(schedule[0: self.N,
    ii]))
390
391         total_watts[np.where(total_watts < 0)[0]] = 0
392         return total_watts, total_watts2
393
394     def _generatePop(self):
```

```
395         # This function generates population, that is random but
    contains feasible
396         # solutions within the given constraints of the system
397         schedule = np.zeros((self.N + 2, self.M))
398         for i in range(self.N):
399             lower = self.ts * self.appliance[i].LL
400             if lower < 0:
401                 lower = 0
402             upper = self.ts * self.appliance[i].UL
403             if upper > self.M - 1:
404                 upper = self.M - 1
405             total = int(self.ts * self.appliance[i].time)
406 
407             if self.appliance[i].type == 0:
408                 rand_val = randint(lower, ((upper - total) + 1))
409                 schedule[i, rand_val: rand_val + total] = np.ones
    (total)
410             elif self.appliance[i].type == 1:
411                 schedule[i, :] = 1
412             elif self.appliance[i].type == 2:
413                 check = 0
414                 while check == 0:
415                     rand_val = [randint(lower, upper) for ii in
    range(total)]
416                     if len(np.unique(rand_val)) == total:
417                         check = 1
418                 schedule[i, rand_val] = 1
419             ##
420         if self.bp <= 0:
421             return
422         [total_watts, total_watts2] = self._watts(schedule)
423         ccc = 0
424         while ccc == 0:
425             power = []
426             index = []
427             schedule[self.N: self.N + 2, :] = 0
428             check = np.zeros(self.M)
429             check3 = np.zeros(self.M)
430             n = randint(1, 4)
431             ind2 = np.zeros(n)
432             for ii in range(n):
433                 batt = self.bp
434                 index = randint(1, self.M - 5)
435                 subind = np.where(check > 0)[0]
436                 c = 1
437                 if ii != 0:
438                     check2 = 0
439                     while check2 == 0:
440                         if subind.size == 0:
441                             c = 20
442                             break
443                         index = randint(0, self.M - 6)
444                         if index < subind[0] - 5 or index >
    subind[-1] + 5:
445                             check2 = 1
446                         c = c + 1
447                         if c > 15:
448                             break
```

```
449                 if c > 15:
450                     break
451                 ind2[ii] = index
452                 c5 = 0
453                 while batt > (min(total_watts2) / self.ts):
454                     if (total_watts[index] / self.ts) < batt and
    check[index] != 1 and check3[index] != 1 and \
455                             total_watts[index] != 0:
456                         schedule[self.N + 1, index] = 1
457                         batt = batt - total_watts[index] / self.
    ts
458                         check[index] = 1
459                         c5 = 1
460                     index = index + 1
461                     if index == self.M - 1 or index + 1 == (ind2
    [0: ii - 1]).all():
462                         break
463                 if c5 == 1:
464                     schedule[self.N, index] = 1
465                     check3[index] = 1
466                 # if schedule[self.N, index - 1] == 1:
467                 #     disp('error')
468             power = self._checking(schedule)
469             A = np.where(schedule[self.N, :] > 0)[0]
470             B = np.where(schedule[self.N + 1, :] > 0)[0]
471             C = A[np.in1d(A, B)]
472             if power.size > 0:
473                 if np.max(power) < self.bp and C.size == 0:
474                     ccc = 1
475         return schedule
476
477     def _get_appliance(self, name):
478         for appliance in self.appliance:
479             if appliance.name == name:
480                 return appliance
481         return None
482
483     @staticmethod
484     def normpdf(x, mu=0, sigma=1):
485         u = float((x - mu) / abs(sigma))
486         return np.exp(-u * u / 2) / (np.sqrt(2 * np.pi) * abs(
    sigma))
487
488     def _print_table(self):
489         ################
490         tab = PrettyTable(['name', 'kW', 'type', 'time', 'LL', '
    UL'])
491         for i in range(12):
492             tab.add_row([self.appliance[i].name, self.appliance[i
    ].kW, self.appliance[i].type, self.appliance[i].time,
493                          self.appliance[i].LL,
494                          self.appliance[i].UL])
495         print(tab)
496
497     def _initial_plot(self):
498         ## Price Function
```

```
499         # This is the definition of pricing function for the
    power supply company
500         self.price = np.hstack(
501             (np.ones(12) * 8.7, np.ones(10) * 18.0, np.ones(10) *
     13.2, np.ones(6) * 18.0, np.ones(10) * 8.7))
502         hours = np.array(range(1, 49)) * 0.5
503         plt.figure()
504         plt.plot(hours, self.price, r, linestyle='dashed', dashes
    =[4, 2, 1, 2])
505         plt.plot(hours, self.PV, b, linestyle='dashed', dashes=[4,
     2, 1, 2])
506         plt.xlabel(Hours)
507         plt.ylabel(Electricity Price (cents/kWh))
508         plt.grid()
509
510     def _generate_sols(self):
511         ## Genetic Algorithm
512
513         # Generate Solutions
514         cost = np.zeros(self.sols)
515
516         solution = []
517         for i in range(self.sols):
518             schedule = self._generatePop()  # generating initial
    population using generatePop()
519             cost[i] = self._costfunction(schedule)  # calculating
     cost of each schedule in population
520             solution.append(schedule)
521         initial_cost = cost[0]
522
523         data = []
524         cost_data = []
525         data.append(solution)
526         cost_data.append(cost)
527
528         # Crossover & Mutation
529         for it in range(self.tot_it):
530             cost1 = np.copy(cost_data[it])
531             probabilities = 1 / cost1  # probability(to become
    parent) is inversely proportional to cost
532             new_population = []
533             temp_cost = np.copy(cost1)
534             for j in range(self.sols):
535                 # parents = crossover(probabilities) # 2 parents
    are selected probabilistically by crossover()
536                 parents = _crossover_tourment_selection(temp_cost
    ,
537                                                         5)  # 2
    parents are selected probabilistically by crossover()
538                 feasible_child = self._generateChild(parents,
539                                                      solution)  #
     generateChild() and mutation() generate a feasible child of
    the parents
540                 cost1[j] = self._costfunction(feasible_child)  #
    cost of each child is calculated
541                 new_population.append(feasible_child)  # each
    child is stored in new population
```

```
542
543             solution = copy.copy(new_population)
544             data.append(new_population)
545             cost_data.append(cost1)
546
547         return initial_cost, cost_data, data
548
549     def _final_figure(self, final_solution):
550
551         self._temperature(final_solution)
552
553         n_2 = np.zeros(self.M)
554
555         power = self._checking(final_solution)
556
557         GU = np.copy(final_solution[: self.N + 1, :].transpose())
558         for j in range(self.N):
559             GU[:, j] = GU[:, j] * self.appliance[j].kW
560         GU[np.where(GU[:, self.N])[0], self.N] = (power * self.ts
    )
561
562         c = []
563         for i in range(12):
564             c.append(self.appliance[i].name)
565
566         c.append('Batteries Charging')
567         df = pd.DataFrame(GU)
568         ax = df.plot(kind='bar', stacked=True)
569         ax.legend(c, loc='best')
570         plt.title('Optimal Scheduling of appliances')
571         plt.xlabel('Time slots')
572         plt.ylabel('Power (kW)')
573         plt.grid()
574
575         plt.plot(self.price - abs(max(np.sum(GU, 1)) - min(self.
    price)))
576         plt.xlim([0, 48])
577         plt.ylim([0, max(self.price) + 1])
578         plt.plot(self.PV + max(np.sum(GU, 1)))
579         n_2[np.where(final_solution[self.N + 1, :] > 0)[0]] = max
    (np.sum(GU, 1))
580         plt.stem(n_2)
581
582         plt.figure()
583         plt.plot(self.price - abs(max(np.sum(GU, 1)) - min(self.
    price)), label='Price')
584         plt.xlim([0, 48])
585         plt.ylim([0, max(self.price) + 1])
586         plt.plot(self.PV + max(np.sum(GU, 1)), label='PV-Source
    power')
587         n_2[np.where(final_solution[self.N + 1, :] > 0)[0]] = max
    (np.sum(GU, 1))
588         plt.stem(n_2, label='Slots supported by Batteries')
589         plt.legend(loc='best')
590         plt.show()
591
592     def final_results(self, initial_cost, final_cost):
```

```
593         ## Price Function
594         # This is the definition of pricing function for the
    power supply company
595         initial_cost = initial_cost / 1000.0
596         final_cost = final_cost / 100.0
597 
598         return initial_cost, final_cost
599 
600     def run(self):
601         self._print_table()
602 
603         self._initial_plot()
604 
605         initial_cost, cost_data, data = self._generate_sols()
606 
607         min_cost, row, col = _calc_min(cost_data)
608 
609         final_solution = data[row][col][:, :]  # final solution
    is the schedule with minimum cost
610         final_cost = self._costfunction(final_solution)
611 
612         print('The initial price is $', initial_cost / 1000.0, '
    for 24 hours')
613         print('The cost is $', final_cost / 100.0, ' for 24 hours
    ')
614 
615         self._final_figure(final_solution)
```

Listing D.1: Genetic Algorithm Python Code

Appendix E

Main Python Code

```
import torch

from Full.forecasting import Forecasting
from Full.geneticalgorithm import GeneticAlgorithm
import numpy as np

def main():
    inputSize = 1
    layer_number = 4
    H_layer = 4
    hours = 30

    M = 48  # total time slots
    sols = 70  # initial no.of solutions in GA
    tot_it = 80  # total iterations in GA
    comfort = 3  # comfort level 1 - minimum 2 - medium 3 - maximum
    #ts = M / 24  # time slot period
    PV_power = 1.2  # peak PV power in kW
    bp = 2  # Current power that the batteries are holding in kW
    price = 0

    fc = Forecasting(hours, inputSize, layer_number, H_layer, '../x_train_8not.csv', '../y_train_8.csv', False)
    #y_final = fc.run(11)#3
    #y_final = fc.load_forecast(14)
    y_final = fc.run_all(36)
    #y_final = fc.run_full(16)

    times = y_final.data.numpy()
    times = np.around(times,2)

    appliance = GeneticAlgorithm.initialization(comfort)
    for i in range(hours):
        Time = times[i]
        for ii in range(4):
            appliance[ii].time = round(100 * float(Time[ii])) / 100.0

        ga = GeneticAlgorithm(appliance, price, PV_power, bp, M, sols, tot_it, True)
        ga.run()

if __name__ == '__main__':
```

```
41        main()
```

Listing E.1: Main Python Code

Curriculum Vitae

Majed Shakir

Education

Academic Qualifications

- **Masters In Science in Electrical Engineering**
  University of New Brunswick *2015–2019*
  IEEE Member
  - Thesis: Small Smart Grid for Utilization System
  - Focus:
    - Artificial Intelligence
    - Embedded Systems
    - Machine Learning
    - Smart Grid
    - Software Design

- **Bachelors In Science in Electrical Engineering**
  University of New Brunswick *2011–2015*
  IEEE Member

Relevant Courses

- Computer Organization
- Digital System Design
- Embedded Systems Design
- Foundations of Artificial Intelligence
- Machine Learning and Data Mining
- Power Electronics
- Power System Optimization
- Power System Stability
- Realtime System
- Realtime System II (*Advance Realtime System*)
- Robotics
- Signals
- Systems and Control
- Topics Artificial Neural Networks
- VLSI System Design

Publishing

- Author: Majed Shakir, Yevgen Biletskiy. Forecasting in Small Smart Grid. IEEE International Symposium on Industrial Electronics.