**The Pennsylvania State University**

**The Graduate School**

# DEEP REINFORCEMENT LEARNING BASED ENERGY-EFFICIENT HEATING CONTROLLER FOR SMART BUILDINGS

A Thesis in

Data Analytics

by

Anchal Gupta



Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

December 2019

The thesis of Anchal Gupta was reviewed and approved* by the following:

Youakim Badr
Associate Professor of Data Analytics
Thesis Advisor

Guanghua Qiu
Professor of Information Science

Ashkan Negahban
Assistant Professor of Engineering Management

Colin Neill
Director of Engineering Programs

*Signatures are on file in the Graduate School.

# Abstract

Buildings account for 40% of the total energy consumption in the world out of which heating, ventilation, and air conditioning are the major contributors. Traditional heating controllers are inefficient due to lack of adaptability to dynamic conditions such as changing user preferences and extreme outside weather conditions. Therefore, it is necessary to design energy-efficient smart buildings which can improvise user comfort while reducing energy consumption. This research presents a Deep Reinforcement Learning (DRL) based heating controller to improve occupant comfort and minimize energy costs in smart buildings. At first, experiments are performed on one building with consideration of synthetic and real-world outside weather data. The DRL-based smart controller decides in real-time and outperforms the traditional heating controllers by improving user comfort between 15% - 30% and reducing energy costs between 5% - 12% in a simulated environment. Experiments are also performed on multiple zones, each having separate heating equipment. The performance is compared for when the multiple heaters are controlled centrally versus decentralized control, where each heater is controlled independently under various settings. As the number of zones and their complexity increase, decentralized controller performs better than the centralized controller. These results have significant practical implications for heating control in buildings consisting for many similar zones, such as office, classroom and apartment buildings as well as hotels and conference centers.

# Table of Contents

# List of Figures

# List of Tables

# List of Equations

Chapter 1

# Introduction

## 1.1 Context

Emerging technologies such as Artificial Intelligence (AI) and Internet of Things (IoT) seem to be shaping the future of the world, a world that require minimal human assistance to integrate and connect devices with things from the physical world. Smartphones, self-driving cars, and chatbots are some of the examples of devices that are making human intervention less essential, even irrelevant. With advanced technologies integrating into people's daily activities, their lifestyles are changing dramatically. Enabling automation of building operations with sensors and actuators are turning a building into a "Smart Building." Smart buildings are highly equipped with sensors and collect data for every action within and outside the building environment. The enormous amount of data generated from the sensors needs to be mined to enable better decision making. Leading areas of artificial intelligence like machine learning, computer vision, big data, and

reinforcement learning are becoming significant and accessible technologies to handle such a tremendous amount of data and provide some useful insights. These emerging technologies have the potential to improve occupant comfort and energy consumption in buildings and improve individuals' quality of life.

## 1.2 Challenges

Buildings account for 40% of the total energy consumption in the world [1]. The biggest challenge and concern is to meet global energy needs efficiently. Advancement in building technologies is leading to extensive development of smart buildings to provide economic and environmental benefits. Smart Buildings are an integration of IoT with building components. They are primarily constituted of the five different components [2], which includes sensors and actuators, smart control devices, networking and communication systems, Heating, Ventilation, and Air Conditioning (HVAC) systems, and software platforms shown in Figure 1.1 [2]. This thesis primarily pertains to smart control of heating systems.

Sensors and control devices are plugged throughout the building to track every activity and act on its equipment. A network allows communication of sensors with building controllers. HVAC units are the most complicated and energy-consuming equipment and plays a vital role in maintaining user comfort. The objective of smart building automation systems is to manage the sensors with a single controller. The considerable amount of data collected from sensors has a vast capability to reduce energy consumption.

Supervised and unsupervised machine learning algorithms and big data analytics can discover specific trends and patterns from the data that would not be apparent to humans, but the algorithms require a large volume of useful quality data for

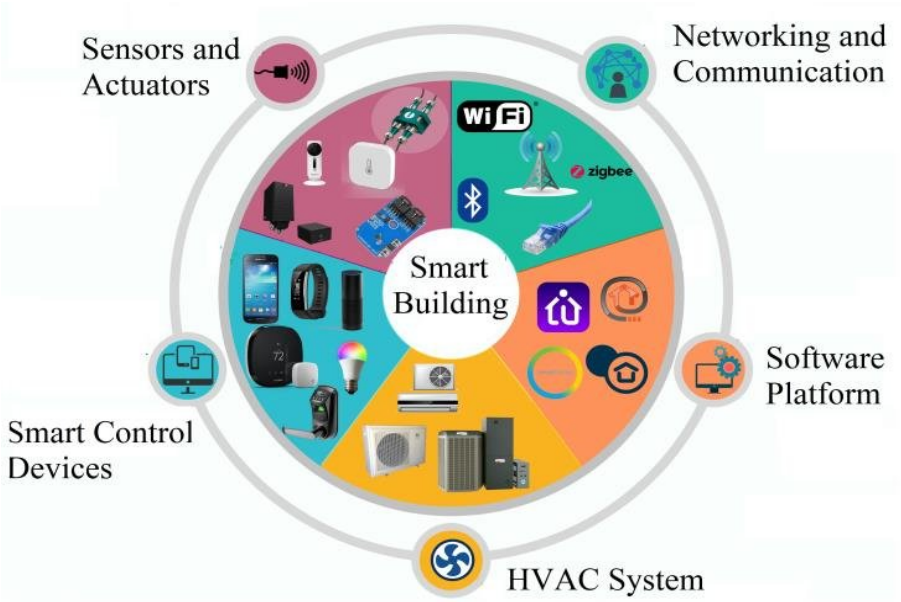



Figure 1.1: Key components of a smart building.

training. Though data acquisition has become accessible by installing sensors in buildings, processing and mining of the data require a lot of computational resources and time. Using small amount of data is highly susceptible to errors and can produce biased results. Reinforcement Learning (RL) is an area of Machine Learning which does not require correct answers/labels for the data and act in the real-time for decision-making. RL requires an environment (often in the form of virtual or simulated) for training and testing of algorithms. Replication of a real-world scenario to a simulated environment and vice-versa is very challenging to avoid biased and inaccurate results.

## 1.3 Research Problem

Automation of building energy management systems has gained significant interest in recent years. Energy consumption in buildings is reported to be a notable portion of the total energy consumed globally. It is one of the primary reasons to

use operational methods for energy optimization. The overall design, insulation properties, and material used for the construction of the building are some of the factors that influence how efficiently can a building use energy resources. The goal of building automated energy management systems is to reduce total energy consumption and costs by using recent advancements in technologies like IoT, automated control and simulation systems, and smart grids.

HVAC is one of the most extensively used and most energy-consuming in buildings. HVAC accounts for 50% of building energy consumption in the US and between 10 - 20% of overall consumption in developed countries [3]. These high percentages result from users prioritizing thermal comfort over the reduction in energy costs [4]. Also, a strong interdependence is noticed between outside weather conditions and energy consumption by HVAC equipment [5]. In regions with extreme cold weather [6] where the temperature drops to -40 degree, people prefer to keep heaters on even if they are outside the building to maintain warm indoor temperatures. Nowadays, the most basic programmable thermostats are used mainly in buildings to automate the heater settings based on a predefined threshold. Due to increasing demand for user comfort, while optimizing energy consumption, there is a need to build automated controllers that can maintain a balance in demands. Recently, smart buildings are supported by various statistical, and machine learning approaches. Machine learning techniques aim to automate the learning process and are capable of providing better results and predictions than traditional statistical approaches. ML techniques can be applied in different styles, including supervised, unsupervised, and semi-supervised learning, where available data is the hero. It becomes challenging to manage a tremendous amount of data generated by sensors and other appliances in smart buildings. Supervised machine learning models and deep neural networks require a massive amount of labeled

data to build control systems and give better results [2]. In case of a change in equipment or users, the historical data becomes obsolete, and the performance of trained machine learning algorithms decreases. Another part of machine learning is reinforcement learning, where learning is the result of the interaction between a model and the environment. This research aims to develop heating controllers which can benefit a smart building by regulating the indoor temperature to maintain user comfort and save energy costs using reinforcement learning techniques. A Building Management System should understand user preferences and behavior to go beyond simple automation strategies. Here, we propose a DRL-based heating controller to automate the decision making in real-time with minimal dependency on historical data.

## 1.4 Overview of Contributions

Reinforcement learning (RL) is a field of machine learning that involves learning of an agent by observing an environment [7]. The agent aims to maximize some reward by exploring the states of the environment by taking specific actions. RL is applied to many areas of building energy management. Deep neural networks combined with RL are also achieving better performance than tabular RL algorithms like Q-learning.

This research aims to develop a smart heating controller using deep reinforcement learning (Deep Q-Networks) that can efficiently improve user comfort and optimize heating costs. Deep Reinforcement Learning is a combination of Deep Learning (that allows to train an Artificial Neural Network (ANN) than can automatically learn complex features) and Reinforcement Learning(that allows learning a policy by interacting with an environment). Deep Q-Networks (DQN) algorithm is

implemented to obtain the optimal policy for the smart heating controller. The algorithm allows the controller to make real-time actions for turning the heater on/off. The Q-values for each action is approximated with a Deep Neural Network (DNN). Previous research on optimizing user comfort and cost in smart buildings and the reason to use DQN over other Artificial Intelligence techniques are discussed in Chapter 2. An overview of DNN and reinforcement learning based algorithms are discussed in Chapter 3.

All experiments are tested in a simulated environment written in MATLAB/Simu -link. The environment is the simulation of a building equipped with a heating system. The simulated environment reflects a physical example of a heating controller in a building. During the simulation, the agent periodically receives data regarding the current state of the building. The data includes the current indoor temperature, the outside temperature, and the instantaneous cost rate. Based on the data, the RL-based controller decides turning the heating system on or off.

A TCP/IP based Application Interface (API) is built to develop communication between RL algorithms (in Python) and simulated environment (in MATLAB/Simulink). The exact implementation details of the heating controller are provided in Chapter 4. The DQN-based heating controller chooses the actions to minimize the difference between the indoor temperature and the setpoint temperature specified by the occupant, and optimize heating costs. The performance of RL and DQN-based controllers are compared to a threshold dependent baseline heating controller.

The experimental results confirm that DQN-based heating controllers outperforms the baseline controllers and improve user comfort while optimizing energy costs. The experimental strategy and statistical results are discussed in Chapter 5. The experiments are performed using both synthetic and real-world outside temper-

atures. The research also extends the idea of implementing DQN-based heating controllers to multiple buildings. Chapter 6 discusses the comparison of one DQN controller managing multiple buildings (centralized control) with each building equipped with a decentralized DQN heating controller.

## 1.5 Conclusion

The smart buildings promise to ease a user's life by providing them the facility to operate everything in a building with one touch of a button. This research proposes a smart heating controller that significantly improves user comfort by regulating the indoor temperature such that fluctuations from the occupant-specified setpoint temperature are minimized, while simultaneously reducing energy consumption and costs.

Chapter 2

# State of the Art

## 2.1 Problem Definition

In the era of Artificial Intelligence (AI) and exponentially growing data, technologies like IoT, big data analytics, machine learning can cost-effectively transform regular buildings into smart buildings with minimum modifications in their infrastructures.

A smart building is an intelligent system that employs ML and networks to communicate between various devices and make the decision to optimize building performance. In smart buildings, smart thermostats that allow controlling the temperature according to user preferences without any manual settings. Occupant's satisfaction is an essential measure to improve building performance because discomfort can force occupants to use resources less efficiently.

The challenge for AI-driven technologies while developing intelligent buildings is to analyze the data available from sensors and learning from user behavior before taking any actions to maintain a balance between user comfort and energy optimization.

## 2.2 Analysis of the Problem

Thermal control in buildings plays an essential role to construct high-quality living and working environments. The indoor temperature and humidity have to stay within the thermal comfort zone for occupants to feel comfortable. However, the surrounding thermal condition may change considerably, which leads to the fluctuation of the indoor temperatures and the discomfort of the occupants. Therefore, it is necessary to build thermal control for maintaining acceptable indoor thermal condition.

Heating, Ventilation, and Air Conditioning (HVAC) systems are generally used to control indoor thermal conditions. These systems consume high energy, which accounts for 20%-40% of the total energy consumption in a building [3] out of which heating systems are less energy-efficient than cooling systems. Because of the inappropriate settings of the setpoints of heating systems, users may feel cold or hot despite more consumption of energy.

Therefore, it is imperative to study how to maintain an occupant's comfort while efficiently using heating control systems and optimizing their energy consumption. This, in turn, helps in reducing environmental pollution and also reduce electricity bills primarily due to the increase in electricity pricing.

## 2.3 Related Works

Recent evolution in AI and ML has led to the immense development of smart buildings. The automation systems are generating environmental and economic benefits for the building management systems. Data analytics and AI techniques are being used to analyze a tremendous amount of data generated from sensors

so that it can be transformed into useful information to make decisions regarding humans and the environment. The subsequent sections list previous works done to build energy-efficient smart buildings using ML, Deep Learning (DL) and RL techniques.

### 2.3.1 Machine Learning & Deep Learning based Research

Buildings equipped with smart sensors and IoT technologies generate a massive amount of data. Data have a scope of valuable information which can be extracted and analyzed. Traditional approaches are unable to handle this vast amount of data and fail to provide useful insights. Machine learning models have been able to overcome this problem by introducing powerful algorithms which can build highly accurate predictive models using the tremendous amount of data produced from various sources. Deep neural networks are also gaining popularity because of the more complex data structure. They are trained on historical data to discover hidden patterns. Predictions are then made based on the learned patterns and data features.

It is essential to study the factors which can contribute to optimizing energy consumption and thermal comfort in a building. Uncertainty/sensitivity analysis has been presented to identify these critical factors [8]. This analysis determines the performance variation in a building and the key factors contributing to its energy consumption. A sparse regression technique Orthogonal Matching Pursuit (OMP) has is compared to conventional least square estimation methods [8]. Modeling error for energy consumption and the average temperature is computed by performing experiments on a simulation building in energy plus tool. The authors show that the OMP requires less training simulation sample to achieve the same

modeling accuracy than the Least Square (LS) method. OMP also significantly reduces the overall modeling cost by 18 times. However, OMP is not very efficient in handling outliers.

Forecasting electricity loads are possible both at the building and equipment level with the installation of sub-meters. Smart buildings have separate meters for HVAC, lighting and other equipment which can keep track of appliance usage. For example, the Support Vector Machine (SVM) method has been proposed to predict the electricity load for the next hour in a public building [9]. A cluster of 24 SVM models has been applied to each hour of the day to forecast hourly electricity load. The model is trained on the data obtained from a building in Shanghai. Also, the methodology outperforms other forecasting models like Auto-Regressive Integrated Moving Average Extended(ARIMAX), Artificial Neural Network (ANN), and Decision Trees. SVM models prove to have the lowest Mean Square Error (MSE), and Root Mean Square Error (RMSE) for total and sub-meters. Forecasting electricity load can help optimizing energy and suggesting users to use equipment efficiently. The SVM model is trained only on the cooling season data and could perform differently when heating conditions are included.

A data-driven framework using Structure Equation Modelling is developed for energy and comfort management [10]. It uses historical data to develop a multivariate model to find the thermal coupling of adjacent zones in HVAC control of large commercial buildings. The proposed framework was validated using real data collected from an airport HVAC system. The focus of the framework was to increase the energy efficiency of HVAC control in the buildings with time-varying occupancy levels and multiple zones. However, only the zone temperature is considered as a control input for experimentation without consideration of preferred temperature by the users.

Analyzing user behavior can help in better understanding of the user comfort. Observing their daily activities and interaction with equipment can help in improving the performance of a building. A similar hybrid technique has been proposed [11], which involves K-pattern clustering for understanding continuous and discontinuous user activities. In this technique, ANN has been applied to predict user activities based on patterns observed. In addition, J48 decision trees are used for feature selection, which improved the results of neural network. The similarity between related activities cause some unsatisfactory results.

Statistical quality control based methods have been proposed to analyze the real state of thermal comfort and energy efficiency [12]. In this context, the quality of HVAC systems in buildings is evaluated using data retrieved from big data web energy platforms. Seasonal ARIMA has been applied to estimate natural variability of temperature and energy consumption in order to detect patterns and system anomalies. In [12], control charts have been used to monitor, control, analyze, maintain, and improve the thermal comfort and energy consumption of HVAC installations quantitatively.

An integrated energy control model, including energy supply model, thermal comfort model, and decision-making model based on AI has been proposed to respond to anti-logic or common sense that can reduce machine's energy-saving effects [13]. The energy supply model controls supply energy to improve user's thermal comfort, whereas the thermal comfort model analyzes these operations and determines the factors that lead to abnormal conditions. The thermal comfort model adjusts the conditions of energy supply by determining the intent of the user. Considering the results of the supply model and thermal model, the decision-making model decides how to optimize the appropriate energy supply to maintain a balance between thermal comfort and energy consumption. The integrated energy control

model results show only 2% - 10% improvement in thermal comfort.

A data-driven approach is proposed to predict thermal comfort level of occupants using environmental and human factors as input [14]. Six supervised machine learning models: SVM, ANN, Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Nearest Neighbour (KNN) and Classification Trees (CT) are used to predicts level of comfort ('cool-discomfort', 'comfort', 'warm-discomfort'). The models are fed with age, gender, air temperature, mean radiant temperature, air velocity, relative humidity, metabolic rate, clothing rate, and outdoor effective temperature as an input. According to the data-driven approach, energy can be saved with the prediction of the thermal comfort of an individual. However, a small dataset (817 data samples) is used for experiments, which is generally considered as a small training dataset for machine learning models.

A modular platform that uses deep neural networks for load monitoring and energy load forecasting is designed to build smart home controllers [15]. The platform uses cloud services to collect and aggregate data from smart environments. In addition, the platform includes a load-monitoring model based on Auto-encoders after testing of several network architectures like Recurrent Neural Networks (RNN) and Long Short-term memory (LSTM) [15]. The load forecasting model was developed using LSTM to detect the significant variance and anomalies of overall energy consumption. The platform helps in detecting energy variation and advise the inhabitants to improve their way of life, which can save energy costs. An evolutionary model combining the Genetic Algorithm (GA) with LSTM is proposed [16] to improve the accuracy of the prediction levels of LSTM.

Authors in [17] introduce a smart thermostat based on Bayesian inference to learn user preferences with limited observations [17]. By learning patterns, an optimal temperature setting schedule can be generated by solving a Stochastic Expected

Value Model (SEVM). User preferences can be tracked dynamically, and the thermostat can automatically update the temperature settings accordingly. The thermostat does not study the effects of output temperatures.

### 2.3.2 Reinforcement Learning-based Research

Traditional Machine learning and Deep learning (DL) models are data-driven and mainly depend on historical patterns to provide future predictions. However, Reinforcement learning is inspired by behaviorist psychology, where an agent takes actions in an environment to maximize long-term rewards. Monte-Carlo methods, Multi-armed bandit, Q-learning [2] are some of the well-known algorithms applied to RL based applications. These algorithms are useful when there is no prior historical data to train supervised learning models for decision making and predictions.

In [18], a review of algorithms and techniques covers the field of RL for demand response applications in a smart grid which includes electric vehicles, HVAC systems, smart appliances, and batteries. According to [18], only a few articles have applied RL algorithms in physical systems, and still, real-world experiments need to be conducted. Similarly, the survey [3] studies the research and practice of DL and RL in smart grids.

A bidding strategy using RL is presented for controlling and coordinating HVAC systems in a double auction market [19]. The authors demonstrate that model-based and model-free algorithms provide better results and conclude that model-free techniques can be used to represent more complex systems

In [20], a Q-learning controller has been developed to operate lights and blinds in a building jointly. This controller provides personalized services as it obtains the

perception of the environment as a feedback signal. Also, a human-machine interface provides the controller with feedback regarding blinds (Draft, noisy, stuffy) and Illuminance (Hot, Dry, and Glare). By such, the Q-learning agent takes action to operate the blinds and lights and improve user satisfaction as well as reduces energy consumption. The controller is trained only during the day time with east-facing blinds.

In [21], an RL-based controller using State-Action-Reward-State-Action (SARSA) is introduced to control the climate inside a vehicle and keep passengers thermally comfortable. The SARSA algorithm outperforms traditional controllers developed with algorithms such as proportional and simple fuzzy logic.

Control of natural ventilation and HVAC operations can together lead to comfort and high amount of reduction in energy consumption. In this context, the control strategy [22] aims to make optimal control decisions for HVAC operations and window systems using RL. The controller proves to lower HVAC system energy consumption between 13% and 23% decrease discomfort degree hours between 62% and 80%Ṫhe proposed RL models rely on numerical simulations, but its application to a real scenario remains questionable.

A smart energy trading approach using fuzzy Q-learning is presented [23]. The fuzzy logic is used to overcome the continuous state space of the scenario. The agent performs two groups of actions: charging and discharging of the battery and a set of trading decisions. As a result, the model makes intelligent decisions to trade surplus energy to maximize the utilization of renewable energy.

In [24], a Q-learning model is developed to learn to when to turn on/off the heater by observing the user behavior. The agent regards the usage of the HVAC and the interaction of the tenant with the system and tries to minimize the HVAC usage by switching it off regularly and turning it on at instances that anticipate the tenant's

action. The models assume that the outside temperature is lower than the user preferred temperature, which makes its scalability difficult to a real scenario.

Standard Q-learning strategies use a Q-table which is not feasible for problems with continuous and ample state space. A spectrum of Deep Reinforcement Learning algorithms are introduced to overcome these limitations, such as Deep Q-networks (DQN) and Deep Deterministic Policy Gradient (DDPG).

An HVAC controller is proposed for buildings with multiple zones [25]. Experiments are performed with real weather and pricing data to significantly reduce energy cost while maintaining the indoor temperature within a range of 19 degrees and 24 degrees Celsius. The authors use different neural networks for buildings with an different number of zones to estimate the Q-values. Features like real-time, zone temperature, ambient temperature, and solar radiance intensity are used as a set of states, and various airflow rates by each HVAC system is considered as an action space. The multi-level control heuristic for multiple zones overcomes the challenge of large action space. It is worth noting that the framework performs less accurately with the high number of zones as compared to buildings with only a few zones.

A Deep neural network with bayesian regularization has been developed to predict thermal comfort ranging from -3 to 3 where -3 is too cold, and +3 refers to too hot, and 0 is neutral. The model is trained using 11164 samples from the ASHRAE RP-884 thermal comfort dataset. A Deep Deterministic Policy Gradient (DDPG) algorithm is applied to learn an optimal policy to maintain thermal comfort [26]. The algorithm takes temperature, humidity, airspeed, metabolic rate, and clothing insulation as input to predict the thermal comfort value. Actions in the Markov decision process seek to set the air temperature and humidity levels of the HVAC system. The strategy shows the advantages of DDPG over DQN and SARSA, but

the dataset used for training is too small.

In [27], authors introduce a strategy for optimal control of space heating in buildings by comparing model-based and model-free DRL algorithms. The indoor temperature is considered as an input to the network and setting the levels of the power of the heat pump (0 to 2000 Watts) as potential actions. A model-free based algorithm called Double Deep Network Fitted Q Iterations (D-DNFQI) algorithms outperform model-based algorithm in reducing energy cost and maintaining user comfort. The experiments focus on scenarios including flat electricity pricing, dual pricing scheme, and real-time pricing and omit essential features such as outside temperature and building dynamics.

Table 2.1 summarizes the state of the art techniques with their advantages and disadvantages. This research is focused on applying deep reinforcement learning techniques to build smart heating controllers that improves occupant comfort while minimizing energy consumption and costs.

## 2.4 Simulation Tools for Energy Domain

Energy simulation software makes it possible to analyze the performance of buildings and optimize energy costs. It helps monitor building equipment, which consequently can be controlled for efficient use based on observations. Bench-marking energy usage through modeling and simulation assist building and facility managers. Since we also used a simulation model to test the appropriateness of our proposed algorithms, we present a brief overview of how simulation tools are used in this context.

In [28], authors use EnergyPlus software to understand the use of energy in residential buildings in Brunei Darussalam. Data is collected from surveys and builders

| Category | Type | Algorithms | Pros | Cons | Applicability in Smart Buildings | Cited |
|---|---|---|---|---|---|---|
| Supervised Learning | Classification | Deep Neural Networks | Does not require feature engineering; Can learn complex non-linear relationships | Requires a lot of data samples for training; increased computational cost | Used for classification, control and automated home appliances, next step/action prediction. | [9][11][14] [15][16] |
| | | SVM | Overfitting can be avoided with appropriate knowledge of kernels and regularization | Computationally expensive; selection of inappropriate kernels can lead to overfitting | Activity recognition, human tracking | [9][14] |
| | | Bayesian Networks | Simple representation; Does not require rich hypothesis | Large dataset for training | Energy management systems and human activity recognition | [17] |
| | | Decision-Trees | Non-parametric model with easy interpretation | Overfitting | Monitoring and feedback | [11][14] |
| | Regression | Orthogonal Matching Pursuit | Requires less training samples | Inaccurate with Outliers | Determine energy efficiency | [8] |
| | Time Series | ARIMA, ARIMAX | Can identify complex temporal relationships | Works well when the data is stationary | Forecasting of energy consumption, weather data | [9][12] |
| Unsupervised learning | Clustering | KNN | Robust to noisy training data. | Slow learner with high computational cost | Human activity recognition. | [14] |
| | | K-pattern clustering | Fast and computationally efficient | Difficult to predict K-Value. | Predict user activities in smart environments. | [11] |
| Reinforcement learning | Temporal Difference Learning | Q-learning, SARSA | Does not require historical data for training | Requires a simulated environment; cannot handle large states and actions | Lighting control services and learning the occupants, preferences of music and lighting services. | [20][21][22] [23][24][25] |
| | Deep Reinforcement Learning | Deep Q-Networks, DDPG | Does not require historical data for training; Can handle continuous states and actions | Requires a lot of experiences to train the agent | To develop controller for improving occupant comfort | [26][27][28] |

Table 2.1: Summary of comparison between State of the Art techniques

for the design of the buildings, outside weather, and occupancy patterns. A simulation model is built in EnergyPlus software using the collected data. The simulation results are compared to the actual power consumption and used to modify building

designs for better energy efficiency.

A case study has been performed using another simulation tool SIMEB [29], software developed by Hydro-Quebec, the electricity provider in Quebec. The tool is used to simulate the building to match the metered data and bills to lower carbon emissions and energy consumption. This software allows estimating the energy consumption of the modeled building, considering the efficiency of energy economy measures, analyzing and comparing with consumption.

Autodesk Revit [30], another simulation software has been used to study the effects of external shading on a building. Various configurations of external shading have been using the software to reduce solar radiations and energy consumption.

A study verifies the use of Energy Recovery Ventilation (ERV) systems on energy consumption by electric heat pumps in a classroom in Korea [31]. Transient System Simulation (TRYNSYS) is used to simulate the environment which calculates the heat supplied to a classroom from the Indoor Unit (IDU), and then it outputs the Power Input (PI) value required for running the Outdoor Unit (ODU). The experiments and the simulation results conclude that energy savings rates of the ERV are lower for buildings with lower occupancy rates.

An air-conditioner model [32] is simulated using MATLAB for a specific room. The results obtained from the simulation model are compared with the results from a measuring instrument. The MATLAB model is used to approximate the energy required to cool a room at a fixed setpoint temperature. Despite its simplicity, the model shows consistent results to the measuring instrument.

## 2.5 Summary

Smart buildings promise to be a world of gadgets which can apprehend occupant. The key to a successful smart building is to integrate various equipment like heating, pumping, lighting, and automate their operations for better performance without human intervention. Though there are many measures been taken and advancement is technology to convert buildings into smart buildings, sensors and actuators installed to collect large volumes of data which provide useful information for decision making and energy consumption optimization. To manage, study, and analyze collected data to provide solutions remain a challenging task. Applying techniques like machine learning and data analytics makes it possible to process and gain insights from large volumes of collected data. Recent advancement in DNN and RL open research opportunities to interact in real-time with the building environment without prior knowledge of historical data. Based on state of the art, it is observable that the concept of smart buildings seems to be feasible, but still, there is a scope to address a variety of challenges that can lead to success in turning this world a smart place to live.

In order to overcome the limitations of maximizing occupant's comfort while minimizing energy consumption and costs, we propose a DQN (a deep reinforcement learning algorithm) based heating controller that maintains a balance between user comfort and optimizing energy cost. The controller uses Deep Q-Networks and MATLAB/Simulink for development. Different from previous work, where thermal comfort is predicted with historical data, our controller does not require any knowledge of historical data and trains itself from the present states of the environment. Also, the previous work is more focused towards optimizing energy consumption than improving user comfort. The heating controller proposed in this

research prioritize occupant's comfort. The controller maximizes user satisfaction which can lead to efficient use of the equipment. Many sophisticated tools like EnergyPlus, TRYNSYS have been used for simulation with a focus on large urban scale buildings, solar systems. We chose MATLAB/Simulink for our simulation environment as it is easy to implement and connect with DRL algorithms written in Python. This research aims to develop a robust heating controller that plugs into any smart building.

Chapter 3

# Theoretical Background

This chapter presents the theoretical knowledge for the contributions, which are required to understand the implementation of the heating controller developed. The theory explains the basic concepts of reinforcement learning, deep learning, deep reinforcement learning, and simulation models. The concepts are required at different stages to build a system which can smartly control the heater by maintaining user comfort and optimizing energy cost.

## 3.1 Reinforcement Learning

### 3.1.1 Introduction

Reinforcement learning (RL) is a goal-oriented branch of machine learning that interacts with an environment for its decision making [7]. It is a process which does not require historical data for learning. Instead, the learner called agent learns from the sequel of its actions. RL is one of the most active and rapidly

growing areas of research in AI. It completely differs from other machine learning paradigms like supervised and unsupervised. Supervised learning models are given labeled historical observations to extrapolate their learning [33]. The learned model can then be applied to an unseen dataset to provide predictions. On the other hand, unsupervised learning models are only provided with input data to discover hidden patterns [34]. RL considers interactions between an agent and an environment which learns an optimal policy by consequences of its actions. The following subsections describe the main concepts used in RL.

#### 3.1.1.1 Representation of Reinforcement Learning

**Agent** : The agent is a decision-maker while training of the RL model [35]. Based on the observations from the environment, the agent performs some actions and receives rewards based on these actions.

**Environment**: The environment is a physical world in which the agent acts as a decision-maker.

**States** : States represent the current situation of the environment.

**Reward**: Reward is the feedback to the agent from the environment on taking action.

A typical scenario of RL is represented in Figure 3.1. The agent acts in an environment by observing a state. Based on the action taken, the agent receives a reward and moves to the next state. This process repeats until a terminal state is achieved.

#### 3.1.1.2 Policy

The policy denoted by $\pi$ defines the behavior of an agent when interacting with its environment [36]. The policy is typically used to decide which action to take in

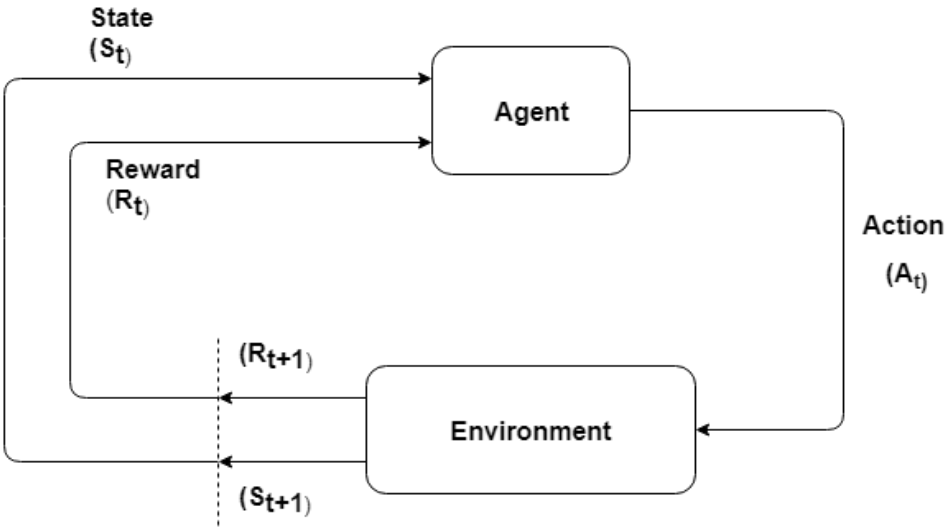



Figure 3.1: Representation of a Reinforcement Learning scenario

a particular state. It can take the form of a complex search operation or a look-up table.

#### 3.1.1.3 Value Function

A value function often denoted by $V_\pi(s)$ is equivalent to the total expected reward received by the agent following a policy $\pi$ starting from the initial state. It indicates how good an agent is in a particular state. An optimal value function is the one with the highest value for all the states.

#### 3.1.1.4 Model

It is the presentation of an agent in an environment and is broadly classified into two categories: model-free learning and model-based learning [37]. In model-based learning, an agent exploits the past learning experiences and only follows the way which has the maximum reward while in model-free learning, the agent explores the environment to maximize the long term reward. The agent performs an appropriate

action through trial and error experiences.

### 3.1.2 Markov Decision Process

According to the Markov property, the future is only dependent on the present state without any consideration of the past. The Markov Chain [38] strictly follows the Markov property and depends solely on the current state to predict the future. The Markov Decision Process (MDP) [39] is an extension of the Markov chain probabilistic model. The Markov Decision Process provides a mathematical framework that is used to model control problems and to study optimization problems like dynamic programming and reinforcement learning. MDP is represented as a tuple of four $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}^{\boldsymbol{a}}_{(\boldsymbol{ss'})}, \boldsymbol{R}^{\boldsymbol{a}}_{(\boldsymbol{ss'})})$ where

$\boldsymbol{S}$: A set of finite states in which an agent is at a particular point of time.

$\boldsymbol{A}$: A set of finite actions which an agent can perform to transit from one state to another

$\boldsymbol{P}^{\boldsymbol{a}}_{\boldsymbol{ss'}}$ : The transition probability that will lead an agent to move from state $s$ (at time t) to $s'$ (time t+1) by performing an action $a$

$\boldsymbol{R}^{\boldsymbol{a}}_{\boldsymbol{ss'}}$: The immediate reward given to an agent for acting in a state $s$ to move to state $s'$

#### 3.1.2.1 The Bellman Equation

There can be two types of value functions:

**State value function ($\boldsymbol{V_\pi}$):** It depicts the goodness of a state [40] by specifying how adequate it is for an agent to be in a state $s$ by following a policy $\pi$. For

MDPs, a $V_\pi$ is defined by

$$V_\pi = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \ , \ for\ all\ s \in S \tag{3.1}$$

where $\mathbb{E}_\pi[.]$ is the expected value of sequence of rewards ($R_{t+k+1}$) with a discount factor of $\gamma$. The **discount factor** determines how much the RL agent cares about the future rewards relative to the immediate rewards.

**State-action value function ($Q_\pi(s, a)$):** Also known as Q-function, it depicts the best of action in a particular state by specifying how adequate it is for an agent to act in a state $s$ by following a policy $\pi$. Value of taking an action $a$ is defined by

$$Q_\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \ , \ for\ all\ s \in S\ and\ a \in A(s) \tag{3.2}$$

where $\mathbb{E}_\pi[.]$ is the expected return starting from state $s$, taking the action $a$. Finding optimal value function and an optimal policy is the ultimate objective MDP. The optimal value function is the one which results in maximum value as compared to other value functions, and the optimal policy is the one having an optimal value function. The Bellman equation [7], named after the American mathematician, Richard Bellman, can be used to solve the objective of MDP. The Bellman Equation to find an optimal value ($V_*(s)$) and an optimal Q-function ($Q_*(s, a)$) is expressed as follows:

$$V_*(s) = \max_\pi V_\pi(s), \ for\ all\ s \in S \tag{3.3}$$

$$Q_*(s,a) = \max_{\pi} Q_\pi(s,a),\ for\ all\ s \in S\ and\ a \in A(s) \tag{3.4}$$

Bellman's equation can be solved with: Model-based (dynamic programming) and Model-free (Monte Carlo and temporal difference) Reinforcement Learning.

### 3.1.3 Dynamic Programming

In Dynamic programming [41], instead of solving a complex problem at once the problem is divided into sub-problems. The solution is computed for each sub-problem. This drastically minimizes computational time by simply reusing the solution if the sub-problem occurs more than once. Dynamic programs can use either value iteration or policy iteration to solve the Bellman equation.

#### 3.1.3.1 Value Iteration

In the Value iteration [42], random values are initialized for each state. Then the Q-function is computed for each state-action pair, and the value function is updated with the maximum value of the Q-function. This is repeated until the change in the value function is minimal.

#### 3.1.3.2 Policy Iteration

A random policy is chosen as the first step in Policy iteration [43]. Then the value function is computed for that policy. If the value function is not an optimal one, then the process is repeated until an optimal value function is obtained, which in turn leads to an optimal policy.

### 3.1.4 Monte Carlo Method

Dynamic programming is used when the model dynamics (transition and reward probabilities) are known, whereas the Monte Carlo method comes into the picture when model parameters are not specified. Monte Carlo method is model-free learning which can be applied only to episodic tasks (which have a well-defined final state). This method uses sampling to approximate the solution. Instead of computing an expected return, the Monte Carlo methods takes the average of the return to approximate the value functions, which is also called a Monte Carlo prediction. The Monte Carlo control algorithm is used to optimize the value function. It follows a closed-loop between the policy evaluation and policy improvement, as shown in Figure 3.2. By such, the policy function always tries to improve itself according to the value function($\pi -> greedy(v)$), and the value function refines itself corresponding to the policy function($v -> v^{\pi}$).

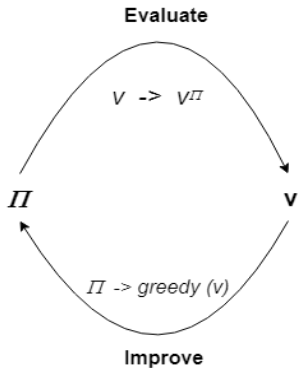



Figure 3.2: A closed-loop representation of Monte Carlo control process.

### 3.1.5 Temporal Difference Learning

An episodic task can last for a long time if there are a large number of states. Temporal Difference (TD) learning algorithms [44] are used to handle this kind of problem. TD learning takes advantage of both Monte Carlo and dynamic programming. It does not require model dynamics like Monte Carlo and can be evaluated at intermediate steps like dynamic programming. Also, TD learning can be applied to a non-episodic task where there is no final state defined. State value is predicted by updating the value of the previous state ($V(S_t)$) with the current state ($V(S_{t+1})$ in TD learning.

$$V(S_t) = V(S_t) + \alpha[R_t + \gamma V(S_{t+1}) - V(S_t)] \tag{3.5}$$

The main goal is to minimize the TD error ($R_t + \gamma V(S_{t+1}) - V(S_t)$), which is the difference between the actual and the current reward in the Equation 3.5. TD control methods are used to optimize the value function ($V(S_t)$). Q-learning and SARSA are generally used for optimizing the estimated value function ($V(S_t)$). Learning rate ($\alpha$) is the step size for convergence.

#### 3.1.5.1 Q-learning

Th ultimate goal of the Q-learning algorithm [20] explained in Algorithm 1, is to learn a policy that can guide an agent to perform an action with a certain set of states. State values are not of much importance in the control algorithm and therefore state-action values are computed. The algorithm has a Q-function that

determines the quality of a state-action pair give by:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1,a}) - Q(S_t, A_t)] \tag{3.6}$$

where $R_t$ is the reward received for moving from state($S_t$) to a new state($S_{t+1}$), and $\alpha$ is the learning rate. At the start, Q is initialized with a set of random values. An action is selected with an *epsilon-greedy policy* [45] in a state $S$. The agent has an exploration-exploitation dilemma where it cannot decide whether to traverse all the possible combinations or act greedy and pick the one that has the maximum return. An epsilon-greedy method is introduced to overcome this difficulty. In this method, different actions are explored randomly with a probability epsilon, and maximum value actions are chosen with a probability 1-epsilon. The agent moves to the next state ($S_{t+1}$) by performing the chosen action and receives a reward ($R_t$). The Q-value is then updated according to Equation 3.6. The episode ends when a final or terminal state is encountered.

**Result:** Estimation of an optimal policy

Initialize $Q(s, a)$ randomly for all $s \in S$ and $a \in A(s)$, and $M$ to the total number of episodes;

**for** *episode = 1 to M* **do**

  Initialize S ;

  **for** *each step in episode until Terminal State* **do**

    Choose action $A_t$ using e-greedy policy ;

    Take action $A_t$, observe $R_t$ and $S_{t+1}$ ;

    $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, A_t) - Q(S_t, A_t)]$ ;

    $S_t \leftarrow S_{t+1}$ ;

  **end**

**end**

**Algorithm 1:** A pseudocode for Q-learning: An off-policy TD control algorithm for estimating an optimal policy($\pi_*$).

#### 3.1.5.2 State-Action-Reward-State-Action(SARSA)

The SARSA [46] algorithm explained in Algorithm 2, is an on-policy TD control algorithm. The Q-values are initialized to some arbitrary values. An action is chosen using epsilon-greedy policy in a state $S_t$.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.7}$$

The chosen action is then performed by the agent to move to the next state ($S_{t+1}$). Instead of updating the Q-value with the maximum value of the next state-action, the next action is picked using the epsilon-greedy policy. This process is

repeated until the last state is achieved.

## 3.2 Deep Neural Networks

Deep learning [47] is a branch of Machine Learning based on Artificial Neural Networks that has multiple layers to extract useful information/feature out of the raw data. Deep Neural Networks(DNN) are being applied in various fields like computer visions, speech recognition, natural language processing, medical imaging analysis, and board game programs, and in many cases, it is shown to perform better than human experts [48]. DNN are also gaining interest, especially due to exponentially growing data.

### 3.2.1 Overview of DNN

A DNN consists of multiple layers between inputs and the outputs. Each layer consists of multiple neurons. The neural network moves through these layers by calculating the probability of each output to find a correct mathematical computation that can map an input to an output. Due to complex mathematical manipulations and many layers, the neural network is termed as "Deep Neural Network." Detailed explanation and working of each component have been provided in further sections.

**Result:** Estimation of an optimal Q-value

Initialize $Q(s, a)$ randomly for all $s \in S$ and $a \in A(s)$, and M to total number of episodes ;

**for** *episode = 1 to M* **do**

  Initialize $S_t$ ;

  Choose action $A_t$ from $S_t$ using e-greedy policy ;

  **for** *each step in episode until Terminal State* **do**

    Take action A,observe R and S' ;

    Choose action $A_{t+1}$ from $S_{t+1}$ using e-greedy policy ;

    $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1})) - Q(S_t, A_t)]$ ;

    $S_t \leftarrow S_{t+1}$ ;

  **end**

**end**

**Algorithm 2:** A psuedocode for the SARSA algorithm: An on-policy TD control algorithm for estimating optimal Q-value ($q_*$)

### 3.2.2 Components

#### 3.2.2.1 Neurons

Neurons [49] are capable of performing basic computation, and combining multiple of them builds a block and can accomplish complex tasks. A neuron can take multiple inputs $x_1, x_2, .....x_n$ and multiplies each of them by their respective weight $w_1, w_2, .....w_n$. Weights [50] are introduced to depict the importance of one input over the other. The multiple inputs and outputs make a neuron different from a perceptron which can only take a single input and can compute one output. There is also a bias value [50] assigned to each layer and added to the sum computed by neurons, as shown in Figure 3.3. Equation 3.8 of the neuron is similar to a linear

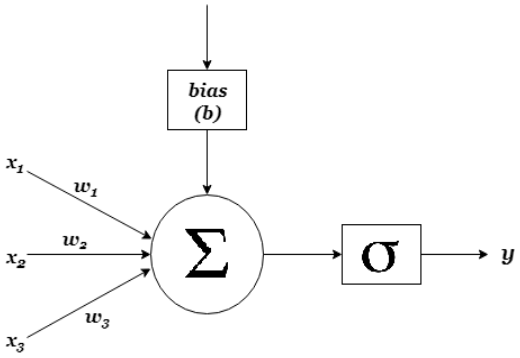



Figure 3.3: Representation of the operation performed by a neuron.

regression model where m is the weights (coefficient), b is the bias (intercept) and x in the input. However, in a neural network, non-linearity is added by applying an activation function ($\sigma$) to the output (Equation 3.9).

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{3.8}$$

$$y = \sigma(z) \tag{3.9}$$

#### 3.2.2.2 Layers

A single neuron is not capable of performing complex classification or prediction tasks. Hence multiple of them are arranged in layers to handle complicated problems. Each layer is connected to every other subsequent layer. The information is distributed throughout the network passing from one layer to another. A neural network typically consists of three types of layers: the input layer, output layer, and hidden layers.

The input layer feeds as an input to the network. The number of neurons in this layer is equivalent to the number of input features. Each input is assigned a weight

and influences to predict the output. The summation of the product of each input feature with their respective weights is computed and added to the bias. There can be only one input layer for each network.

Any layer between the input and output layers is referred to as a hidden layer. There can be zero, one, or more than one hidden layers. The job of each layer is to discover the hidden patterns in the data by processing the input received from the input layer and performing some complex operations. The output of each hidden layer is passed as an input to the consecutive layer. More hidden layers tend to work better for complex problems. For example, in the case of object detection in a video stream, there are hundreds of features to learn by the network. Hence more hidden layers may be needed to extract the vital information [51].

After passing the information through the input and hidden layers, there is an output layer which gives the predicted output. The number of neurons in this layer is dependent on the problem to be solved. If the problem is a classification problem with the multi-class label, then the neurons are equivalent to the number of classes. In regression problems such as the price prediction of a house, a single neuron is present in the output layer. Figure 3.4 shows an example of a fully connected neural network with 1 input layer, 2 hidden layers, and an output layer. The input layer has 2 neurons which represent two input features. The first hidden layer has 4 neurons and the second hidden layer has 3 neurons. The output layer consists of two neurons, representing the two output variables.

#### 3.2.2.3 Activation Functions

An activation function [52] is used to add non-linearity to the network. They are applied to the output of the neurons of each layer. There is a variety of activation functions, but the most commonly used are ReLU, Sigmoid, and Softmax.

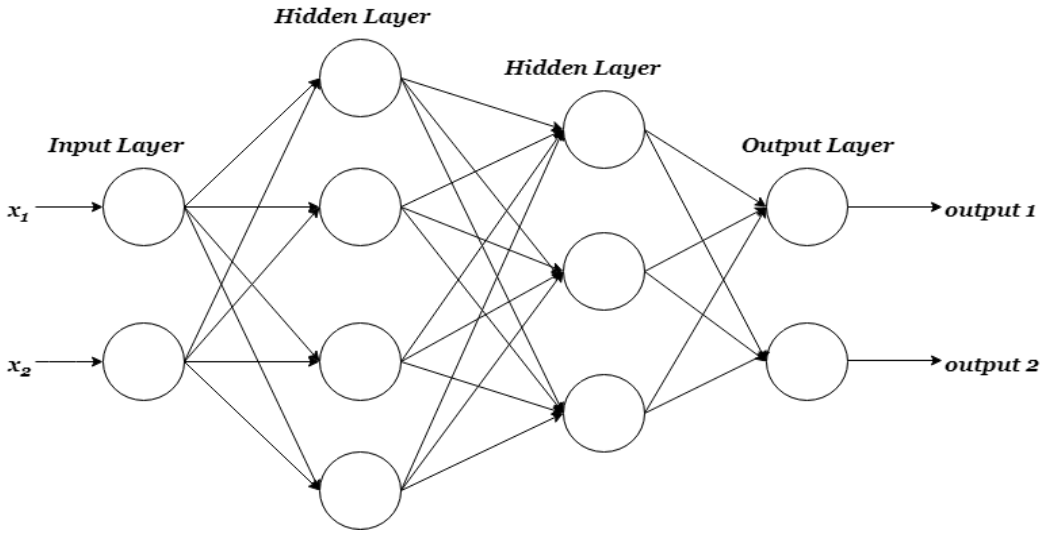



Figure 3.4: An example of a Fully Connected Neural Network

The ReLU activation function [53] stands for Rectified Linear Unit which is graphically represented in Figure 3.5 and mathematically defined as:

$$f(z) = max(0, z) \tag{3.10}$$

where f(z) is 0 when z is less than 0 and equal to z when z is greater than or equal to 0.

A sigmoid function [54] represented in Figure 3.6 scales the value between 0 and 1 and is defined as follows:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{3.11}$$

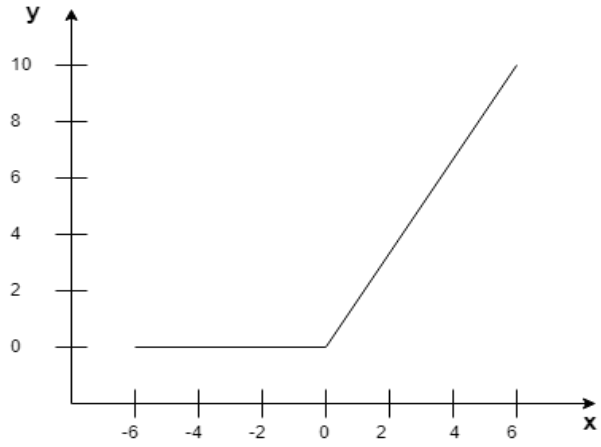



Figure 3.5: A graphical representation of the ReLU activation function

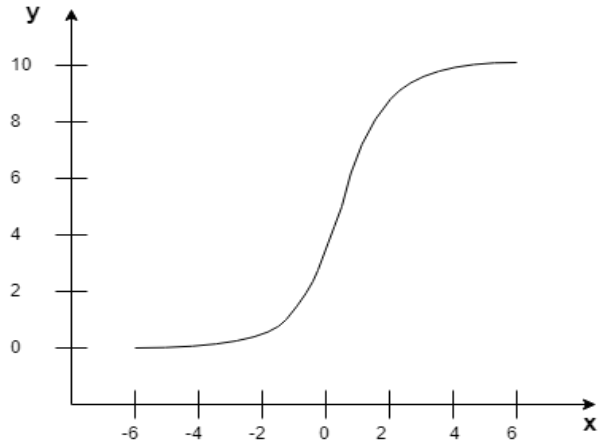



Figure 3.6: A graphical representation of Sigmoid activation function

For the multi-class problems, a softmax function [55] is applied to the final layer of the network to output probabilities of each class. The sum of the probabilities given by softmax is always equal to one. A threshold can be imposed to decide the class of the input. It is defined as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.12}$$

### 3.2.3 Computation and Loss functions

The forward propagation is the combination of all the components explained above. The input is fed through the input layer. Each input is assigned a weight which is randomly initialized at the start. These weights are then multiplied and added together with the bias assigned to each layer. It is always advised to initialize the weights to a random number rather than zero to enhance effective learning of the network. An activation function is then applied to predict the final output. This output (predicted by the neural network) is then compared to the actual output. The difference between the actual and the predicted output is termed as the error function, also known as a loss function. Different loss functions exist and the choice depends on the type of problem being solved. The two most commonly used loss functions are the Mean Squared Error (MSE) [56] and the Cross-Entropy (CE) [56] given by Equation 3.13 and 3.14 respectively.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{3.13}$$

$$CE = \frac{-1}{N}\sum_{i=1}^{N}[y_n log(\hat{y_n}) - (1 - y_n) log(1 - \hat{y_n})] \tag{3.14}$$

The objective of the network is to minimize this loss so that the predicted output value is close to the original label.

**Gradient Descent**

To minimize the loss, randomly initialized weights and biases are updated with backward propagation [57]. Gradient descent algorithm is explained in Algorithm

3. The cost function can be represented as a convex function where the objective is to reach the lowest point, i.e., the point where the cost is minimum, as shown in Figure 3.7. The x-axis represents the value of weights, and the y-axis represents the value of loss function. The goal is to find the particular value of weights for which the loss is minimum.

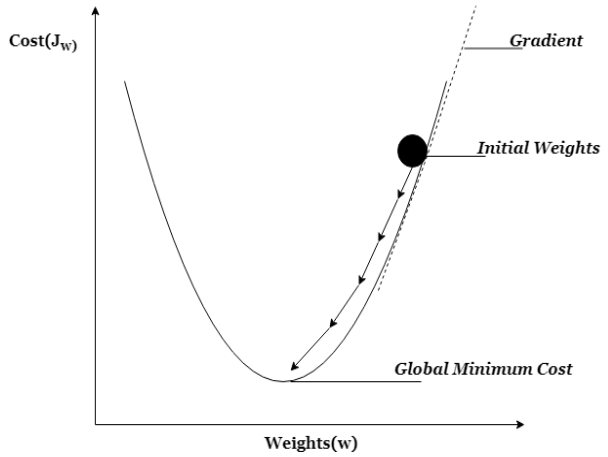



Figure 3.7: Representation of a loss function

Starting from the initial point, the values of weights and biases are updated with each step taken in the direction towards the global minimum. The updating is performed by computing the partial derivative of the cost function with respect to both weight and bias. The Gradient Descent can be defined by the following set of Equations 3.15 and 3.16.

$$w = w - \alpha \frac{\partial J}{\partial w} \tag{3.15}$$

$$b = b - \alpha \frac{\partial J}{\partial b} \tag{3.16}$$

where $\alpha$ in the equation is the learning rate and refers to the step size being taken to reach towards global minimum.

**Result:** A trained neural network to extrapolate the output for a given set of input

Initialize the weights and biases randomly, and training length to the total number of episodes;

**for** *episode = 1 to training length* **do**

  Perform forward propagation to compute the output ;

  Evaluate the error between actual and the predicted output ;

  Compute the gradient descent using backward propagation ;

  Update the weights and biases ;

**end**

**Algorithm 3:** A psuedocode for training a neural network

## 3.3 Deep Reinforcement Learning

The Deep Reinforcement Learning [58] combines neural networks with reinforcement learning principles to create efficient algorithms than the RL algorithms.

**Deep Q-Network**

Building a Q-table in case of a large number of states and actions is very time-consuming and increases the computational time to traverse through every action in each state. Deep Q-Networks (DQN) [59] are used to overcome these problems by approximating the Q-function with a neural network. DQN algorithm is explained in Algorithm 4.

**Neural Network:** A neural network can be of any type (feed forward neural network, convolution neural network, or recurrent neural network). States are passed

as an input to the network, and the output of the neural network is the approximated Q-value for each action. The output layer has neurons equivalent to the total number of actions.

**Experience Replay:** A replay buffer or an experience replay [60] stores the transition information of an agent moving from current state ($S_t$) to next state ($S_{t+1}$) by performing an action $A_t$. This information is also known as the agent experience. A fixed number of recent experiences are stored in a queue, and a batch of them are randomly sampled to train the Q-network.

**Target Network:** The squared difference between the predicted and the target value is used as the loss function given by the following function.

$$Loss = (R_t + \gamma max_a Q(S_{t+1}, A_t; \theta^{'}) - Q(S_t, A_t; \theta))^2 \tag{3.17}$$

where $R_t$ is the reward obtained on taking action $A_t$ in the state $S_t$ to move to state $S_{t+1}$, $\theta$ and $\theta^{'}$ represent the weights of Q-network and the target Q-network respectively. Two separate neural networks are used to calculate the target and the predicted value to avoid divergence. The parameters are updated with the gradient descent algorithm.

## 3.4 The Need for Simulation Models to Train Learning Algorithms

Real systems can be replicated to a model using Simulation Modeling [61]. Simulation models reproduce the behavior and logic of a real system using statistical descriptions of the activities involved. A simulation model has two parts: entities

and tasks. Entities represent the machines, material, and people involved, whereas tasks refer to activities such as actions, processing, and transporting [62]. Each activity is governed by logic. The simulation model runs for a certain period complying with the relevant rules that have been set up. Rules help to analyze the functionality of a system. It can also be tested for any addition and manipulation of logic before deploying it to an actual system. Simulations can also lead to a reduction in failures and save a significant amount of cost.

Simulation modeling is being used from decades in daily practical use and consists of a vast body of real-world examples while DRL is still considered as a research topic and has a large room for development in the area of AI. Simulation modeling involves taking the simulation model and running experiments to replicate the real-world scenario and improve decision making before actual deployment.

OpenAI [63] gym provides simulated environments to allow users to implement different styles of the agent interface. In [64], authors create a 3D simulated environment to train an agent for object detection. They create scenes using the Unity 3D game engine and feed them as an input to the DRL algorithm. In [65], a framework is proposed for autonomous cars based on DRL. Historical driving data are fed into a simulation environment for agent's interactions to find an optimal policy that can estimate the relative speed between a lead and following vehicle. Agents have demonstrated the capability of extracting useful insights and decisions from simulated environments. An RL agent requires a lot of exploration to train itself. Training the RL agent in real-time will require high computational costs and time. Therefore integrating simulation modeling with AI is useful, especially as interest moves away from gaming challenges and towards business-oriented objectives.

**Result:** Q action value function (to obtain policy and select action)

Initialize replay memory $D$ ;

Initialize action-value function $Q$ with random weight $\theta$ ;

Initialize target action-value function $\hat{Q}$ with weights $\theta^{'} = \theta$ ;

Initialize $M$ to the total numbers of episodes and $K$ to the number of iterations ;

**for** *episode = 1 to M* **do**

  Feed the pre-processed game screen to the network ;

  **for** *t = 1 do K* **do**

    Select a random action a with probability epsilon or an action that has maximum Q-value ($A_t = argmax(Q(S_t, A_t, \theta))$) ;

    Perform the action in state $S_t$ to move to next state $S_{t+1}$ and receive reward $R_t$ Store the transition information in D as $< S_t, A_t, R_t, S_{t+1} >$ ;

    Sample a random batch of transition and compute the $Loss = (R_{t+1} + \gamma max_a Q(S_{t+1}, A_t; \theta^{'}) - Q(S_t, A_t; \theta))^2$ ;

    Perform gradient descent w.r.t $\theta$ ;

  **end**

  Copy actual network weights $\theta$ to the target network weights $\theta^{'}$ ;

**end**

**Algorithm 4:** A pseudocode of the Deep Q-Network algorithm

Chapter 4

# A Reinforcement Learning based Heating Controller

Reinforcement learning considers the interaction of an agent with its environment. The agent observes the states of the environment and learns to act according to the modified behavior of its environment. A smart heating controller can automatically adjust the indoor temperature in a building according to occupant's preference. For building a smart heating controller, a simulated environment is developed in MATLAB/Simulink to train the RL agent and study the effects of its actions to optimize energy costs and improvise user comfort.

## 4.1 Thermal Model of a House

A house heating system is provided by MATLAB/Simulink [66], which is used as a baseline model to compare the performance of the proposed RL-based heating controllers (shown in Figure 4.1). The simulated environment is implemented in Simulink and calculates the heating cost for a generic house. The building of the

house consists of one floor and six windows with an area of 3229.17 square feet and a height of 4 meters(m). Each wall of the house is insulated with 0.2 meters thick glass wool. Figure 4.1 [66] represents the simulated environment which models the outdoor temperature, the thermal characteristics of a house and heating systems to calculate the heating costs.

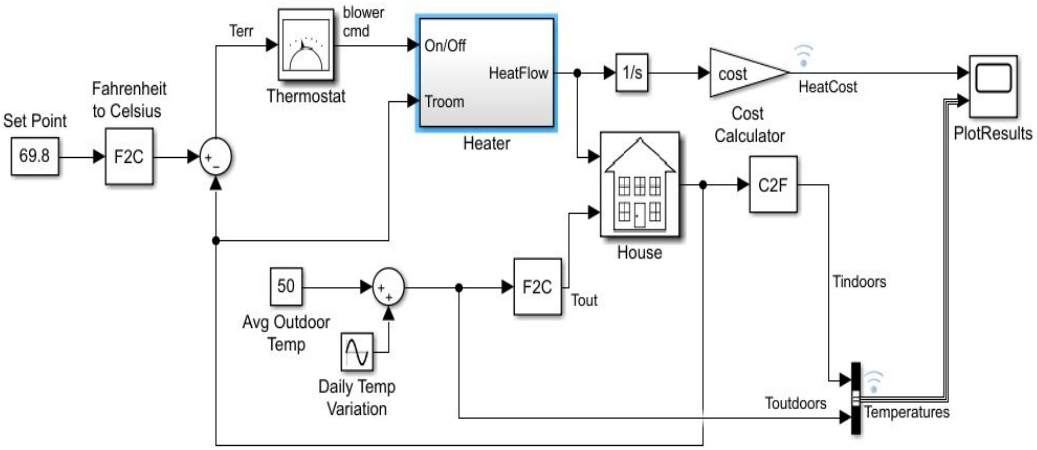



Figure 4.1: Thermal model of a house

### 4.1.1 Set Point

Setpoint is a constant block which specifies an ideal indoor temperature which should be maintained. It is initialized to 69.8°F (21°C), as a perfect ambient temperature for occupants [67]. The setpoint is converted to degree Celsius for further calculations.

### 4.1.2 Thermostat

The thermostat is a subsystem which consists of a relay block and takes the difference of indoor temperature and setpoint (Terr) as an input and allows a fluctuation

of ±5 degrees. If the indoor temperature reaches 75°F or more, the heater is turned off by the thermostat. If the indoor temperature drops below 65°F the heater is turned on. The command of turning on/off the heater is sent to the heater subsystem using a blower switch as shown in Figure 4.2.

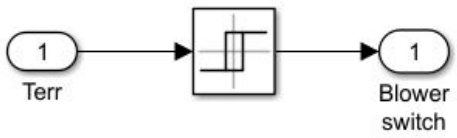



Figure 4.2: A thermostat subsystem

### 4.1.3 Heater

The heater is a subsystem that has a constant airflow rate. When the heater is turned on by the thermostat, it blows hot air at a constant temperature ($THeater$) of 50°C and a constant flow rate of Mdot (1kg/sec = 3600kg/hr). Figure 4.3 represents a heater subsystem that calculates the heat flow from the heater into the room given by the following Equation 4.1:

$$\frac{dQ}{dt} = (THeater - Troom) \cdot Mdot \cdot c \tag{4.1}$$

where $c$ is the heat capacity of air at constant pressure with value 1005.4 J/kg-K and $Troom$ is the indoor temperature.

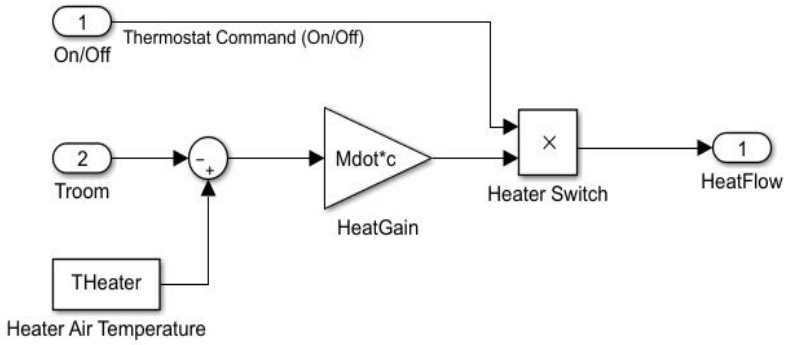



Figure 4.3: Heater subsystem

### 4.1.4 Cost Calculator

Cost Calculator is a gain block which integrates the heat flow and multiplies it by the energy cost rate which is set to 0.09 per $3.6e^6$ J by default.

### 4.1.5 Average Outdoor Temperature and Daily Variation

In Figure 4.1, the average outdoor temperature is a constant block set to 10 °C. The daily temperature variation block is a sine wave used to generate synthetic outdoor temperature.

### 4.1.6 House

House is a subsystem that considers the heat flow by the heater and the heat losses to the environment in order to calculate the variations in indoor temperatures as shown in Figure 4.4. The heat loss and and variations in indoor temperature is calculated using the Equations 4.2 and 4.3 respectively.

$$(\frac{dQ}{dt})_{losses} = \frac{T_{room} - T_{out}}{R_{eq}} \tag{4.2}$$

$$\frac{dT_{room}}{dt} = \frac{1}{M_{air} \cdot c} \cdot (\frac{dQ_{heater}}{dt} - \frac{dQ_{losses}}{dt}) \tag{4.3}$$

where $M_{air}$ is the mass of air inside the house and $R_{eq}$ is the equivalent thermal resistance of the house.

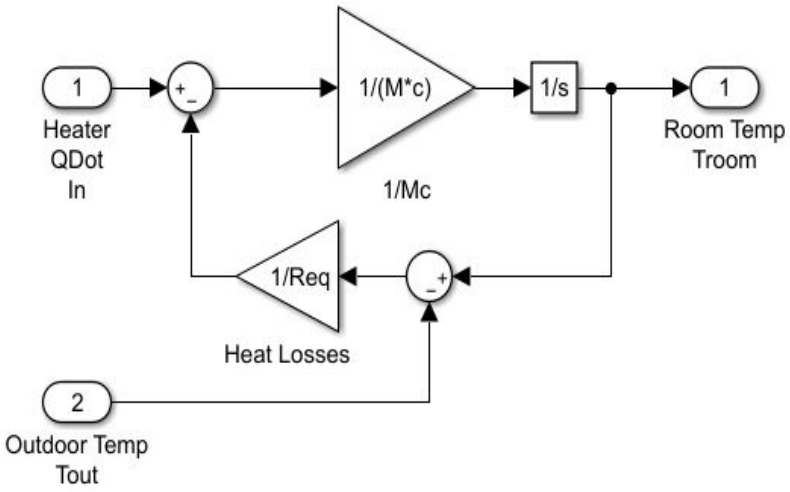



Figure 4.4: Thermodynamic model of the house

### 4.1.7 Plot Results

Plot results is a scope block used to visualize the cumulative heat cost along with the varying indoor and outdoor temperatures as shown in Figure 4.5

## 4.2 Features of Reinforcement Learning

The most important task is to decide upon the various features required to implement RL algorithms. The correct definition of these features plays a vital role in

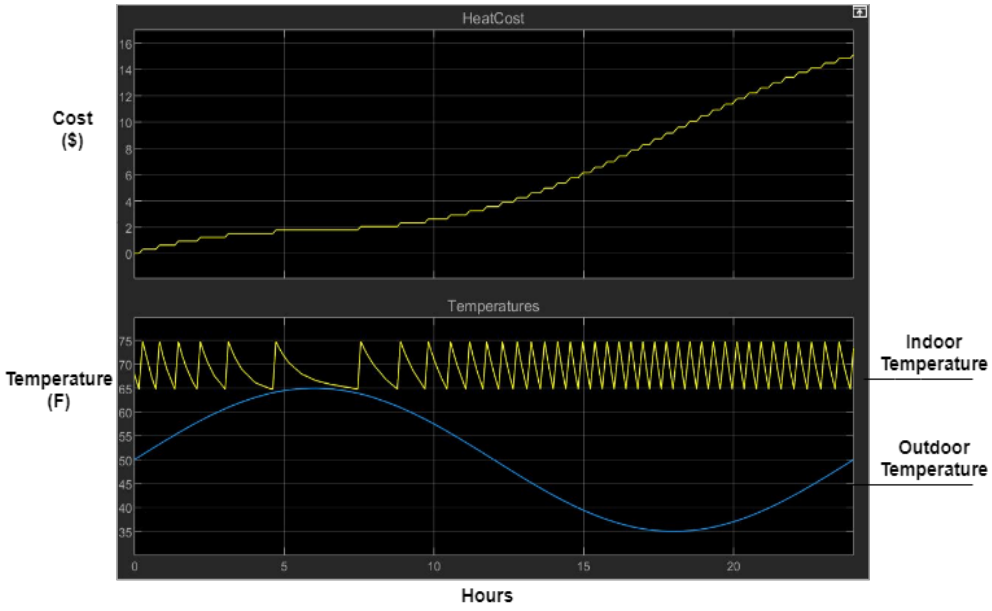



Figure 4.5: Simulation Results

building an efficient RL algorithm. Minor changes in the attributes can make a significant difference in the results.

### 4.2.1 Actions

The **baseline controller** turns on/off the heater system based on a threshold. The MATLAB/Simulink environment [66] sets the threshold to ±5. The baseline controller is replaced by the RL-based controller which decides whether to switch on/off the heater based on the states defined in the next section. The action is a set of two values:{0,1} where 0 is for turning on, and 1 is for turning off the heater respectively.

$$Actions = \{ON = 1, OFF = 0\} \tag{4.4}$$

### 4.2.2 States

As discussed in section 3.1.1.1, states are the representation of an environment. An agent optimizes its actions based on the observed states. In our environment, a state is a set of outside temperature and the difference in the setpoint and indoor temperature.

$$State = \{ T_{out}, T_{diff} \} \tag{4.5}$$

**Outside temperature ($T_{out}$):** The outside temperature conditions highly influence the inside temperature of the building. If the outside temperature is much lower than the room temperature, the heater needs to produce more heat to maintain a comfortable ambient for occupants. On the other hand, if the outside temperature is too high, the controller has to manage to keep the heater off to maintain occupant's comfort. The heating controller should be able to learn a policy that fits the outside weather conditions.

**The difference between the indoor and setpoint temperature ($T_{diff}$):** Setpoint is the temperature preferred by the occupant, and indoor temperature is the inside temperature of the house. Based on the difference between the two, the controller should be able to decide whether to turn on or off the heater.

### 4.2.3 Reward Function

The main goal of the RL-based learning algorithm is that the agent should follow a behavior (policy) to maximize the reward received for every action being taken to move from one state to another. We introduce two reward functions for RL-based controllers:

1) **Comfort-driven:** Since our goal is to maintain user comfort within the house,

we formulate a reward function that maintains an indoor temperature close to the temperature preferred by the user (i.e., setpoint). Equation 4.6 illustrates the comfort-driven reward function.

$$Reward = -(T_{diff})^2 \tag{4.6}$$

By making the temperature difference negative and quadratic, we impose a hefty penalty on the controller as a small error is also reflected as large fluctuations. As a result, the controller can optimize its actions and work more efficiently.

2) **Integrated Cost and comfort-driven:** While maintaining user comfort, it is also essential to consider the costs associated with it. Turning heater on and off consumes an ample amount of energy and thus needs to be optimized along with the user comfort. To overcome this, we formulate a reward function in the hope to maintain can maintain a balance between energy cost and user comfort. The cost and comfort-driven reward function is formulated in Equation 4.7.

$$Reward = -\beta(cost) - (1-\beta)(T_{diff})^2 \tag{4.7}$$

Cost in the Equation 4.7 is equivalent to the instantaneous cost calculated by the simulated environment on turning on/off the heater. The $\beta$ coefficient is introduced to give different weights to cost and the difference in indoor and setpoint temperatures ($T_{diff}$). Also, *cost* and $T_{diff}$ have different scales and to avoid biased

results, we normalize both the values between [0,1] using min-max normalization.

$$z = \frac{x - min(x)}{max(x) - min(x)} \tag{4.8}$$

where $x$ can represent *cost* or $T_{diff}$.

## 4.3 Implementation

To develop a heating controller which can optimize energy and maintain user comfort, we replace the thermostat subsystem with an RL-based controller represented as action by the agent in Figure 4.6. This block acts as the agent of the algorithm, which takes action to turn on or off the heater.

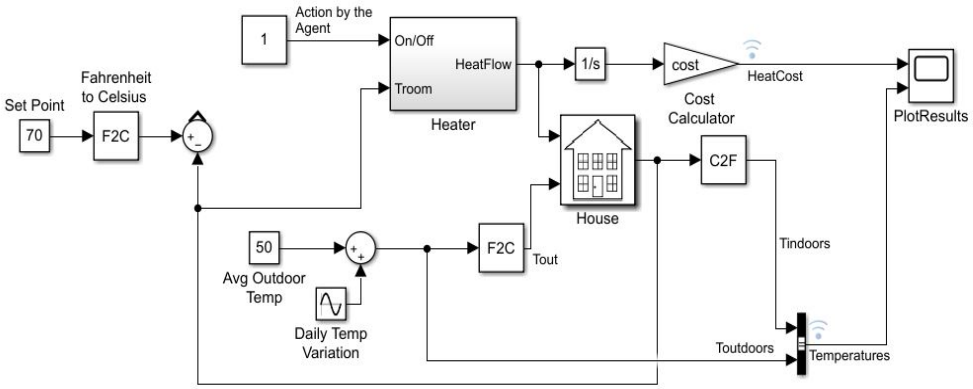



Figure 4.6: Simulated environment for reinforcement learning based controller

### 4.3.1 Reinforcement Learning-based Heating Controller

To build an RL-based heating controller, we implement the Q-learning algorithm, which is a type of temporal difference technique. The algorithm does not require model dynamics and can be computed for intermediate steps.

The algorithm is written in Python and has an API to connect with the simulation environment running in Simulink. The Q-table (Figure 4.8) is initially populated with all possible combinations of discretized states (Equation 4.5) and actions (Equation 4.4). Values of the states are discretized by rounding off to the nearest integer to reduce the number of states. There are a total of 400 states with $T_{out}$ in the range [0,20]and $T_{diff}$ in the range [-10,10]. Each state has two possible actions $\{ON = 1, OFF = 0\}$. The Q-values for these actions is initialized at 0.

The Q-learning algorithm [7] receives data every 5 min of simulation time from the Simulink model. Based on the state received, the controller decides whether to turn on or off the heater following an epsilon-greedy policy. The action is then sent to the simulated environment to receive the next state. The controller's action is rewarded, and the Q-value is calculated based on the present and the next state. The Q-table is then updated with the calculated Q-value for the individual state action taken. The process is repeated until the reward reaches a constant value, and the Q-values are minimized. An architecture of Q-learning algorithm is shown in Figure 4.7.

### 4.3.2 Deep Q-Network based Controller

In problems like controlling the heater in buildings, there can be fluctuations in indoor and outdoor temperatures. Feeding all temperature values in the Q-table can lead to a large table and difficult to handle. This also increases the compu-

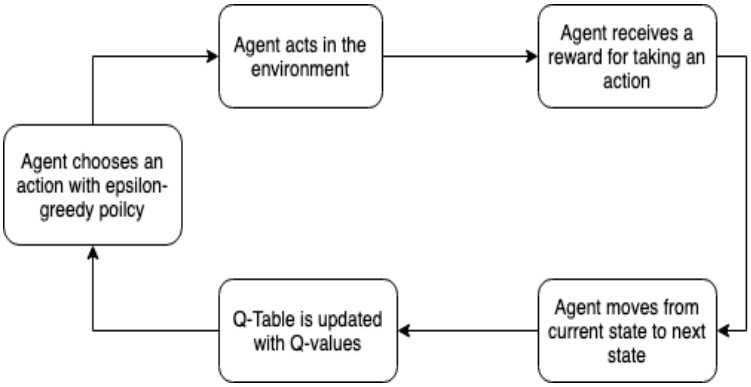



Figure 4.7: Q-learning model architecture

| Q-Table | | | Actions | |
|---|---|---|---|---|
| | | | ON = 1 | OFF = 0 |
| States | $T_{out}$ | $T_{diff}$ | | |
| | 14 | 5 | 0 | 0 |
| | 31 | 2 | 0 | 0 |
| | 23 | 3 | 0 | 0 |

Figure 4.8: A snapshot of Q-Table

tational cost and requires a larger memory space. To overcome this problem and consider continuous states, we decided to implement Deep Q-Network [68], which is a DRL algorithm that does not require a lookup Q-table.

We implement the DQN algorithm in Python using Tensorflow. It receives data every 5 minutes of simulation time. The state is sent to the Deep Q-network, as shown in Figure 4.9 and the action is predicted using a fully connected deep neural network following the epsilon-greedy policy.

The fully connected deep neural network has been constructed to predict the action and Q-values of the actions. Figure 4.9 explains the architecture of a DQN algorithm. The network takes $T_{out}$ and $T_{diff}$ as an input, and the output layer

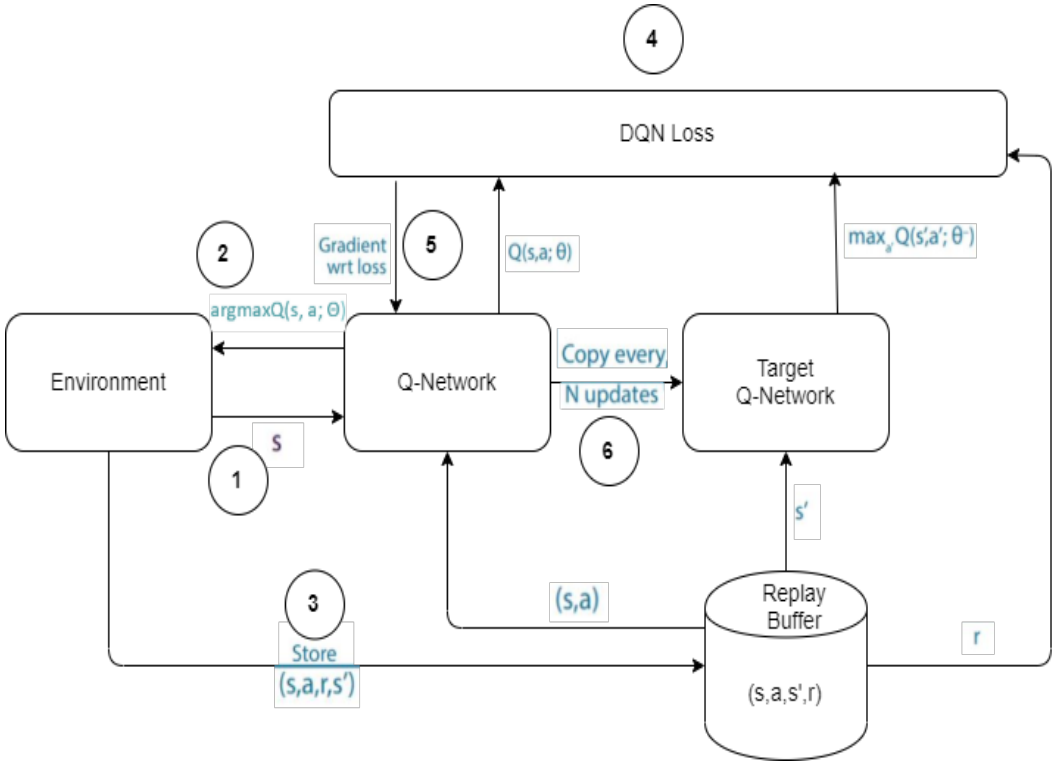



Figure 4.9: Deep Q-Network architecture

has two neurons, each dedicated to predicting the Q-values for each action (step 1). The action selected with the maximum Q-value is then sent to the simulated environment to receive the next state (step 2). The information of state, action, reward and next state ($< S_t, A_t, R_t, S_{t+1} >$) together are then stored in a deque which is used as a replay buffer (step 3). After a certain number of steps, a batch of recent experiences is chosen from the experience replay buffer. These values from the replay buffer are used to train the Q-Network to predict the Q-values for the actions (step 4 and 5). The target Q-Network is updated with weights of Q-Networks after every N steps (step 6). The goal is to minimize the DQN loss illustrated in Equation 3.17.

### 4.3.3 Python Simulink Interface

As the simulated environment is written in MATLAB/Simulink, and algorithms for the heating controller are implemented in Python, we need a solution to connect them in order to send and receive states and actions. The different methods for inter-process communication are discussed in the following sections with additional details.

#### 4.3.3.1 MATLAB/Python API Engine

Our first choice was to use MATLAB/Python API engine [69] to connect Simulink to Python programs. The MATLAB API for Python provides a package for Python to use MATLAB as a computational engine. In the case of simulation models, the package exchanges information only after a simulation ends. Hence we decide to run the simulation model for every 5 minutes. The next simulation run continues from the last state and runs for 5 minutes. This process continues until the entire duration of interest is simulated. This integration has a significant high execution time and memory space complexity due to memory management between Python and Simulink, therefore MATLAB/Python API integration proved to be not suitable for experiments.

#### 4.3.3.2 Transmission Control Protocol/Internet Protocol (TCP/IP) Message Passing

The Transmission Control Protocol [70] allows developing a communication link between Simulink and Python. The TCP/IP block is used to send and receive data, respectively, as shown in Figure 4.10. The TCP/IP message passing is much faster than the MATLAB/Python API engine as it does not require breaking down

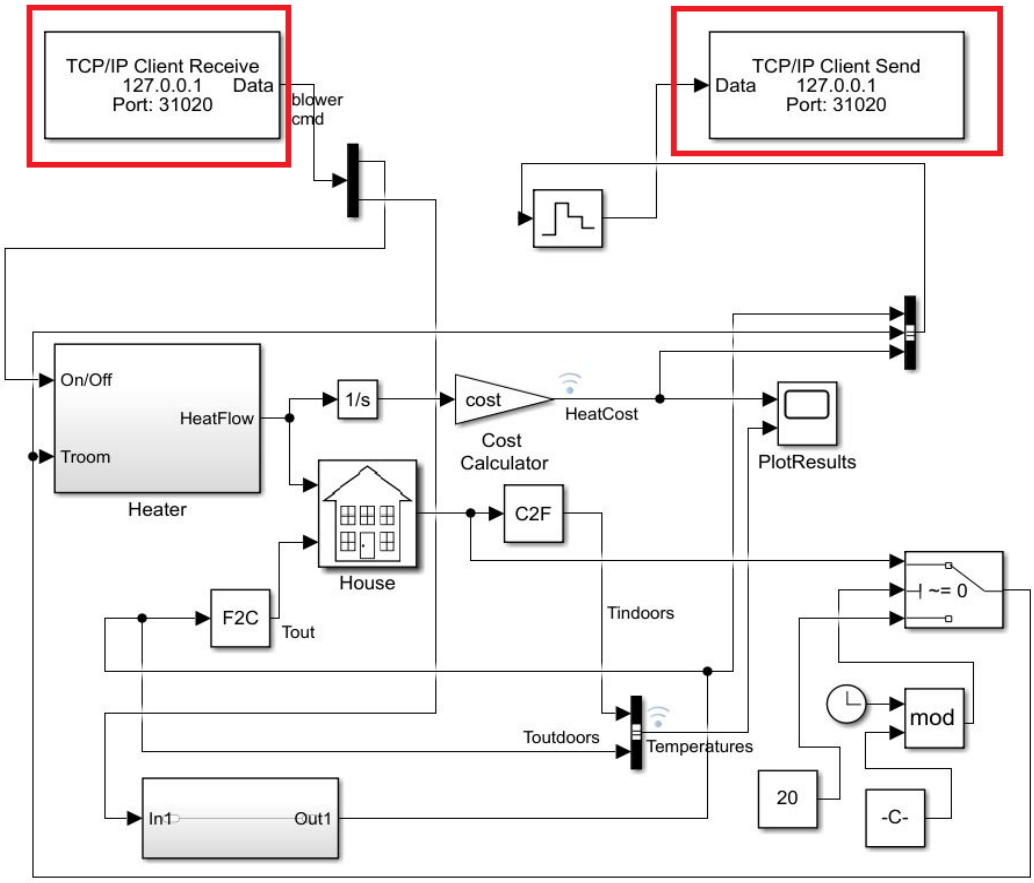



Figure 4.10: Thermal Model of the house with TCP/IP blocks

every run into separate 5-minute runs and provides consistent performance in experimentation.

The RL-based controllers are tested for various parameters and scenarios to increase efficiency, as discussed in the next chapter.

Chapter 5

# Experimental Results

In the previous chapter, we introduced the methods for building a heating controller based on RL-based algorithms. To this end, we compare the RL-based heating controllers with a baseline controller based on a threshold. The RL based controllers are trained in two simulated environments: synthetic and real-world outside temperatures. This chapter looks into the results for comparing baseline controllers with RL-based controllers. An overview of the experimental strategy is illustrated in Table 5.1.

## 5.1 Environments

The RL-based heating controllers are tested in two different simulated environments explained in further subsections.

**Outside Weather Conditions**

The outside temperature is one of the essential parameters which affects the indoor temperature of a building. Firstly, we generate synthetic outdoor temperature data to test the RL-based heating controllers. After computing the performance on syn-

| **Controller** | **Reward Functions** | **Outside Weather Data** | **Training = 1 Day Test = Average of last 50 episodes of training** | **Training = 22 days Validation = 6 days Test = 3 days** |
|---|---|---|---|---|
| Baseline with ±3 threshold (MATLAB/Simulink rule) | Comfort-driven | Synthetic | ☑ | |
| | | Real-World | | |
| | Cost and Comfort-driven | Synthetic | | |
| | | Real-World | | |
| Baseline with ±2 threshold (ASHRAE's rule) | Comfort-driven | Synthetic | ☑ | |
| | | Real-World | | |
| | Cost and Comfort-driven | Synthetic | ☑ | |
| | | Real-World | ☑ | ☑ |
| Q-learning (RL-based) | Comfort-driven | Synthetic | ☑ | |
| | | Real-World | | |
| | Cost and Comfort-driven | Synthetic | | |
| | | Real-World | | |
| Deep Q-Networks (DRL-based) | Comfort-driven | Synthetic | ☑ | |
| | | Real-World | | |
| | Cost and Comfort-driven | Synthetic | ☑ | |
| | | Real-World | ☑ | ☑ |

Table 5.1: Experimental Strategy

thetic data, we collect real-world data to test the RL-based heating controllers and compare their experimental results.

### Synthetic Data

The synthetic data generation for outside temperature data was the first choice to test the heating controllers. The rationale behind synthetic data is to ensure that Q-learning and Deep Q-learning techniques can be applied to the heating controllers. Also, the baseline controller running on MATLAB/Simulink uses synthetic data to show the functionality and behavior of a heating controller.

A sine wave with an amplitude 15 and a frequency of $pi/24$ radians per second is

given as an input to the outdoor temperature. This function is shown in Figure 5.1 and is a convenient way to model the behavior of outside temperature.

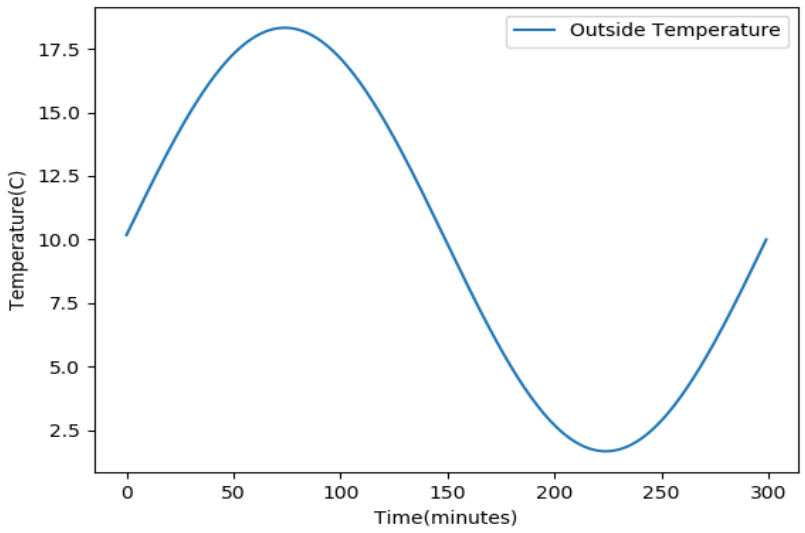



Figure 5.1: Synthetically generated sine wave used as outside temperature data for experiments. Figure shows 5 minutes interval data for 24 hours (1 day).

**Real-World Dataset**

The real-world weather data [71] is collected to evaluate the performance of the heating controller in real-time cases. The dataset is drawn from the Philadelphia region. The temperature data is collected for March of the year 2019 (March $1^{st}$ to March $31^{st}$) on hourly intervals. The real-world weather data shows high-temperature variations and requires a heating controller to manage the heater settings efficiently. Figure 5.2 shows hourly data for 31 days of March 2019. The data is interpolated between two-hour data points to make it a 5-minute interval series as shown in Figure 5.3.

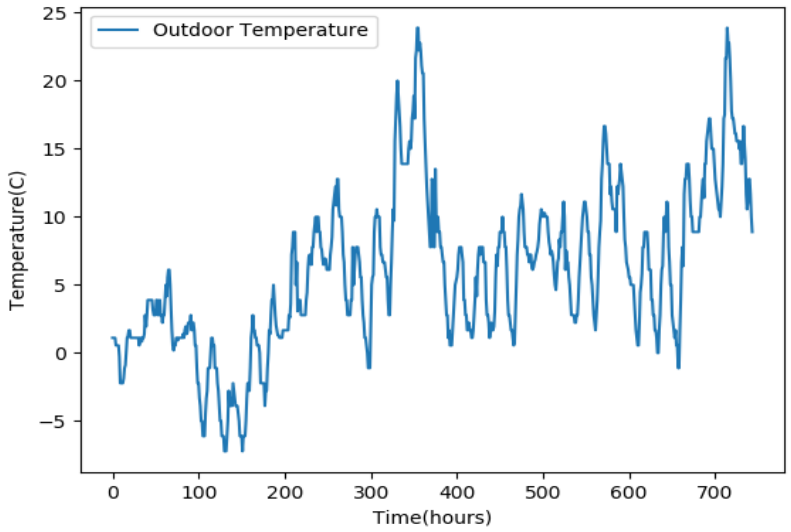



Figure 5.2: March Outside Temperature

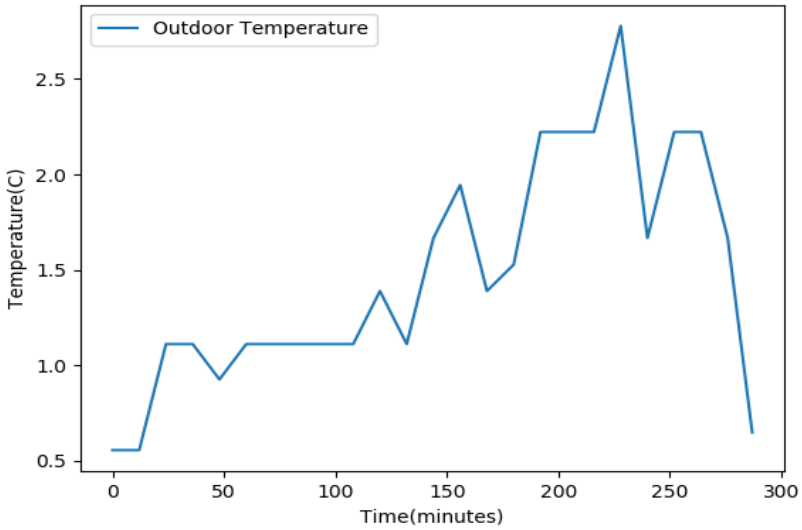



Figure 5.3: Interpolated outiside temperature data for March 4.

## 5.2 Hyperparameters

A Hyperparameter is a configuration that is external to the model. Roughly speaking, the hyperparameter values cannot be estimated by the model or the data used

in its training. Practitioners usually use heuristics to set their values. Hyperparameters affect the performance of a learning algorithm and usually used in predictive modeling for optimization problems. This section describes the hyperparameters used for RL and DRL. Table 5.2 illustrates the initial set of hyperparameters used for experiments.

**Learning Rate:** This parameter decides the step size at each iteration to reach the minimum of the loss function.

**Batch Size:** Data can be too large to feed into a neural network at once and can increase the computational time if passed one at a time. Therefore data points are passed in batches to the neural network.

**Hidden Layers:** The number of hidden layers and the neurons in each layer describes the complexity of the network. A large number of layers and neurons are used to solve complex problems and takes a significant amount of training time.

**Training length:** Training length is the total number of times the data is passed through a neural network for the learning process. Each pass of the total dataset is called an episode. It lets the agent explore and the neural network to improve its performance. Although sometimes more training can lead to overfitting. Overfitting is the case when a model performs substantially more accurately on training data than on validation data.

**Activation Function:** It plays a vital role while updating the weights during backward propagation in a neural network. This is explained in section 3.2.2.3.

**Discount factor:** The value of the discount factor can be between 0 and 1. It decides the significance of future rewards. By setting its value to 0, the agent ignores the knowledge of prior experiences and reacts based on the current states. While setting it to 1 makes the environment's history infinitely long.

**Buffer size:** Some recent experiences of the agent are stored into a replay buffer

in deep reinforcement learning for training of the neural network.

**Exploration step:** Q-learning and Deep Q-Networks follow the epsilon-greedy policy to select an action. It is the rate set for an agent to explore its environment. It can either be set to a fixed value or can be decreased at every iteration of the agent.

**Exploration final:** If the exploration step is set to a decreasing value, the exploration final value is the minimum value for the exploration. The value cannot decay beyond this value.

| **Hyperparameter** | **Value** |
|---|---|
| Learning Rate | 0.001 |
| Batch Size | 32 |
| Hidden Layers | 2 |
| Training Length | 800 |
| Activation Function | ReLU |
| Discount Factor | 0.95 |
| Buffer Size | 10000 |
| Exploration Step | 1 episode |
| Exploration Final | 0.01 |

Table 5.2: Hyperparameters used for training of Q-learning and DQN-based heating controllers

## 5.3 Statistical Measures for Result Comparison

An action is taken by the RL-based heating controller after every 5 minutes for one day to move from one state to another. Each episode runs for 24 hours which consists of a total of 288 data points (1 episode). In our experiments, we focus on three statistical measures: mean, standard deviation, and the cumulative cost, which are computed to compare the performance of the RL-based controllers to the baseline controller. This section describes the steps used to calculate the three

statistical measures.

**Mean:** Every 5 minutes, the absolute difference between the setpoint temperature and the indoor temperature is calculated. At the end of each episode, an average is calculated of all the differences computed every 5 minutes illustrated in Equation 5.1, where N is the size of an episode.

$$Mean = \frac{\sum_{i=1}^{N} abs(T_{(diff)(i)})}{N} \tag{5.1}$$

**Standard Deviation(Std):** A lower value of the standard deviation indicates that the data points are closer to the mean, whereas a higher value indicates a more extensive spread. The standard deviation of the absolute $T_{diff}$ is calculated at the end of each episode.

$$Std = \frac{\sum(abs(T_{diff}) - Mean)^2}{N - 1} \tag{5.2}$$

**Cumulative Cost:** Instantaneous cost is returned every 5 minutes by the simulation model. Total cost at the end of each episode is equivalent to the accumulated cost for every 5 minutes.

$$Cumulative\ Cost = Cost + instantaneous\ cost \tag{5.3}$$

The higher value of mean and standard deviation signifies a more significant deviation from the setpoint temperature, which is equivalent to less user's thermal comfort, whereas cumulative cost describes the measure to compare energy cost savings.

## 5.4 Q-learning Results

The Q-table is initialized with a total of 400 rows and 4 columns, as discussed in section 4.3.1 and shown in Figure 4.8. State values and cost consumption are received for every 5 minutes from the simulated environment for one day (288 different samples per day). The Q-learning based heating controller is tested on the synthetic data and runs for a total of 800 episodes. The results are discussed in the following subsections.

### 5.4.1 Comfort-driven Reward Analysis

The comfort-driven reward function discussed in Equation 4.6 is used for training the heating controller. Figure 5.4 shows the total reward for each episode by the baseline controller and the Q-learning based controller. The total reward is the sum of rewards obtained for taking action in every state. The main goal of the RL-based algorithms is to maximize the reward.

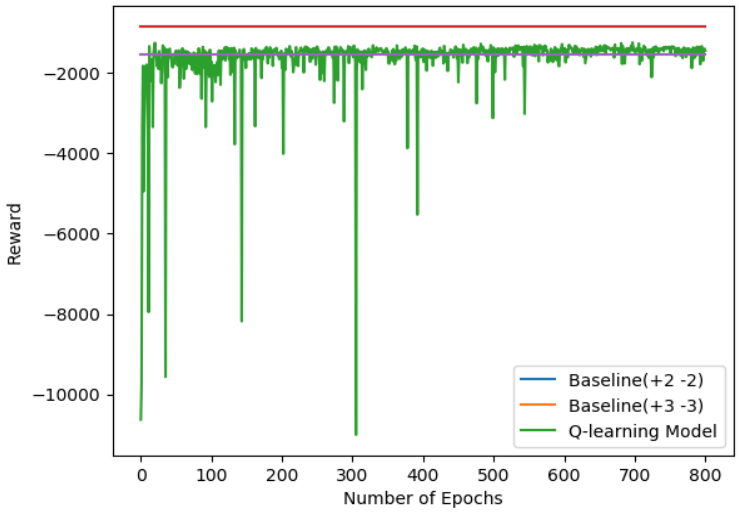



Figure 5.4: The total reward gained by the agent is plotted over 800 epochs.

**Case 1:** According to the baseline controller in MATLAB/Simulink, the heater

is turned on when the difference between the setpoint temperature and indoor temperature is less than or equal to -3 and is turned off when the difference is equal to or more than 3. The total reward for this baseline controller is equivalent to -1520.

**Case 2:** However, according to ASHRAE [67], user comfort is maintained if the indoor temperature is between 19°C and 23°C. Hence, the MATLAB's controller is modified, and the heater is turned on when the difference between the setpoint temperature (21°C) and the indoor temperature drops below -2 and is turned off when difference rises above 2. The reward obtained in this case is -833.

**Case 3:** The reward of Q-learning model-based heating controller increases from -10633.4 to -1435.8 over 800 epochs.

Figure 5.4 shows that both the baseline controllers (case 1 and case 2) have a constant reward as the actions are fixed due to programming based on a threshold. The Q-learning based controller shows variation in rewards due to random actions taken at the beginning because of the high probability of exploration. The reward gets constant after a few epochs with an increasing rate of exploitation. The results illustrated in Table 5.3 confirm that Q-learning based-controller performs better when compared to the baseline controller set to a threshold of $\pm 3$ .

| **Controller** | **Comfort-driven Reward** |
|---|---|
| Baseline(+3 -3) | -1520 |
| Baseline(+2 -2) | -833 |
| Q-learning | -1454.82 |

Table 5.3: Average of reward over the last 50 episodes.

### 5.4.2 Statistical Analysis based on Comfort-driven Reward

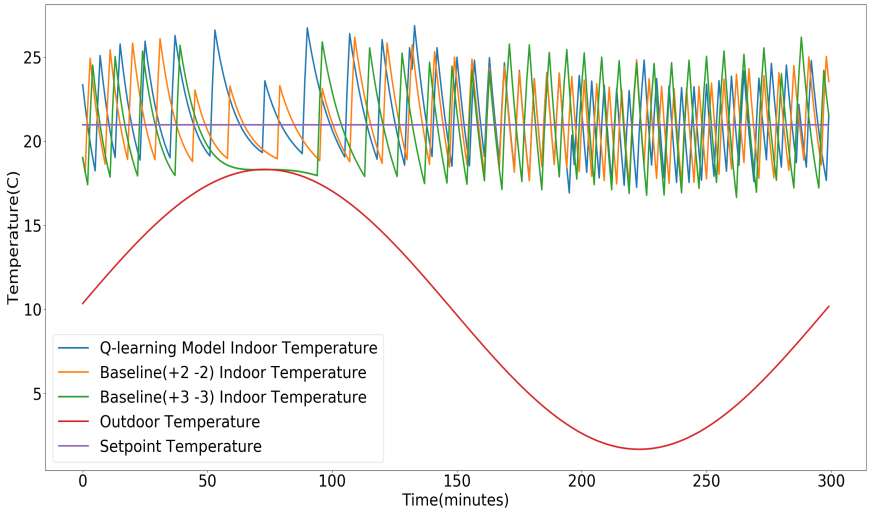
Figure 5.5(a) shows the variations of indoor temperature due to actions taken by the Q-learning based controller and the two baseline controllers with thresholds ±2 and ±3. Whereas Figure 5.5(b) shows the cumulative cost calculated over one day (300 iterations). Mean, standard deviation, and the cumulative cost of Q-learning based controller is computed by taking an average of last 50 episodes of training. Table 5.3 compares the statistical computations of the Q-learning based controller to the baseline controller.

| **Controller** | **Mean(°C)** | **Std(°C)** | **Cost($)** |
|---|---|---|---|
| Baseline(+2 -2) | **1.64** | **1.14** | 15.86 |
| Baseline(+3 -3) | 2.09 | 1.23 | **15.16** |
| Q-learning | 1.78 | 1.29 | 16.18 |

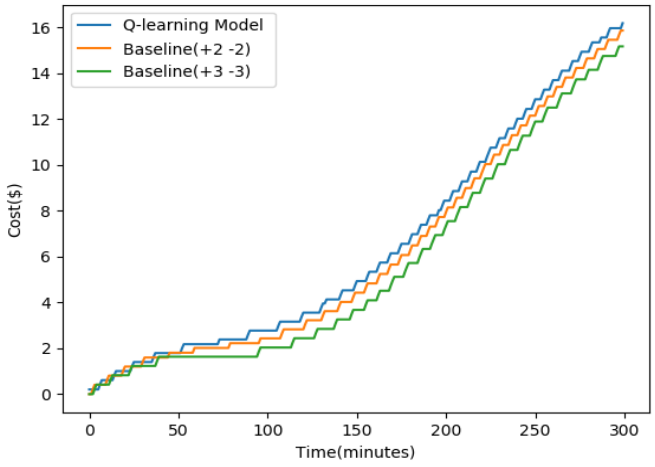
Table 5.4: A statistical comparison of Q-learning controller with baseline controllers.

The Q-learning based heating controller improves user comfort if compared to the baseline controller (threshold ±3) by 17.4% in mean temperature difference. However, it does not performs well when compared to the ASHRAE's rules(threshold ±2). To conclude, baseline controller with threshold ±2 will be preferred due to its better performance than the Q-learning controller and the baseline controller with threshold ±3.

Inconsistent results cannot be ignored while designing efficient heating controllers. The main agenda of the research is to optimize energy consumption and user comfort in all cases. Also, an increasing number of continuous states is challenging to store in the Q-table and requires too many search operations. It increases computational time and costs. A neural network which can estimate the Q-values seems




(a) Indoor Temperature Variations




(b) Cumulative cost

Figure 5.5: Comparison of variations in indoor temperature and cumulative cost after training for 800 episodes

to be more appropriate in case of continuous states. Combining reinforcement learning with deep neural networks can potentially yield better results, which is discussed next.

## 5.5 Deep Reinforcement Learning Results

Data received every 5 minutes is stored in the replay buffer. After the end of every episode (288 iterations), a neural network with the configuration of 2 hidden layers with 24 neurons each is trained. The controller is run for 800 episodes for two types of outside temperature: synthetic and real-world, discussed in the following sections.

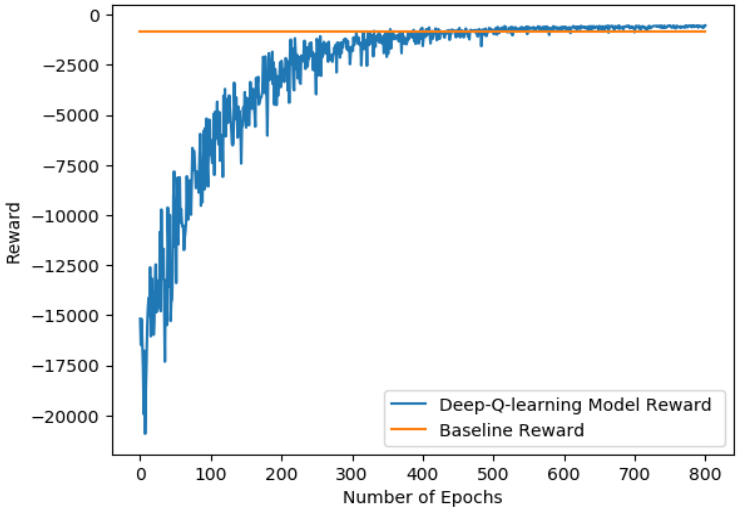
### 5.5.1 Synthetic Data Analysis

Initially, the Deep Q-Network (DQN) based controller is trained on outside temperature data generated by the sine function discussed in section 5.1.1.1

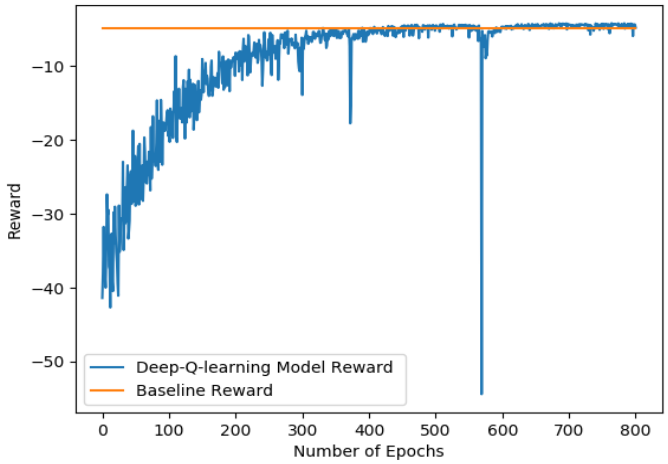
#### 5.5.1.1 Reward Analysis for DQN

The comfort-driven reward function discussed in Equation 4.6, is applied to train the DQN-based heating controller. Figure 5.6(a) shows the comfort-driven reward given to the controllers over 800 epochs. It shows an increasing pattern rising from -21000 to -1450. The reward values for DQN-based controller start to become smooth after 400 episodes and shows a flattening curve after 700 episodes. The DQN-based heating controller receives a higher reward of -1460 and outperforms the baseline controller based on ASHRAE's rules, which has a constant reward of -1520.

Next, we use the integrated cost and comfort reward function from Equation 4.7 to train the controller. Figure 5.6(b) shows the cost and comfort-driven reward given to the controllers based on its consideration of staying close to the setpoint temperature and optimizing cost. The graph shows an overall increasing pattern but some significant dips around episode 350 and 580 for the DQN-based controller.




(a) Comfort-driven reward comparison of Deep Q-Learning model(DQN-based heating controller) with Baseline controller




(b) Cost and comfort-driven reward comparison of Deep Q-Learning model(DQN-based heating controller) with Baseline controller

Figure 5.6: Comparison of different reward functions for DQN and baseline heating controllers

It is due to the exploration of the controller to maximize the long-term reward. The DQN-based controller observes a higher value of the reward that implies a better performance with the increasing number of episodes.

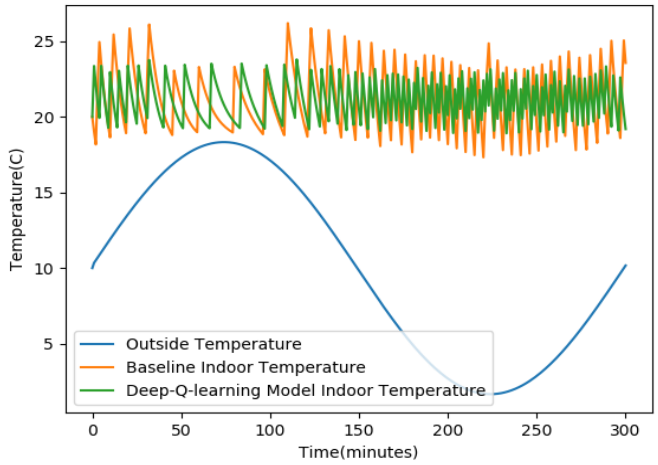
#### 5.5.1.2 Statistical Analysis for DQN

Statistical measures related to user comfort and energy costs are summarized in Table 5.5.

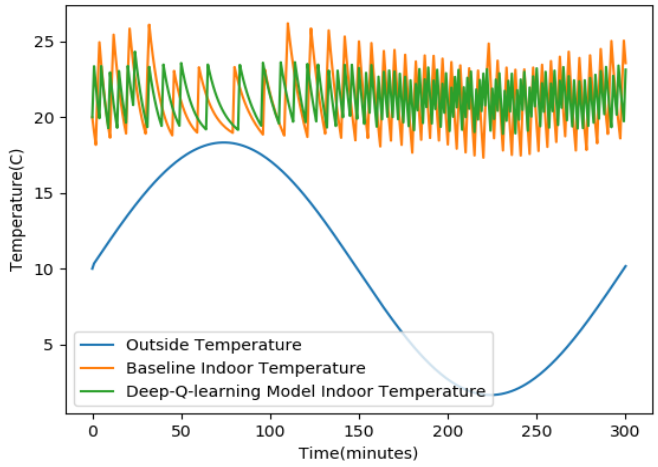
| **Controller** | **Mean(°C)** | **Std(°C)** | **Cost($)** |
|---|---|---|---|
| Baseline | 1.64 | 1.14 | 15.86 |
| DQN with comfort-driven reward | **1.19** | **0.73** | 15.64 |
| DQN with cost and comfort-driven reward | 1.34 | 0.81 | **15.01** |

Table 5.5: Statistical comparison of the baseline controller with DQN-based controller while considering different reward functions and synthetic outside temperature.

Figure 5.7(a) shows the indoor temperature variations due to actions taken by the Baseline controller and DQN-based controller with comfort-driven reward function. Figure 5.7(b) shows the indoor temperature variations due to actions taken by the Baseline controller and DQN-based controller with the integrated cost and comfort driven reward function. The graphs are the results of training the controller for 800 episodes. DQN-based heating controller with comfort-driven reward observes a decrease of 27% in mean and 35.96% in standard deviation. A decrease of 18.2% and 19% in mean and standard deviation is observed with integrated cost and comfort driven reward function. An interesting observation is that even though the comfort-driven DQN controller does not consider cost at all, it still performs slightly better than the baseline controller in terms of energy cost.




(a) Indoor temperature variation when the controller is rewarded for improving comfort

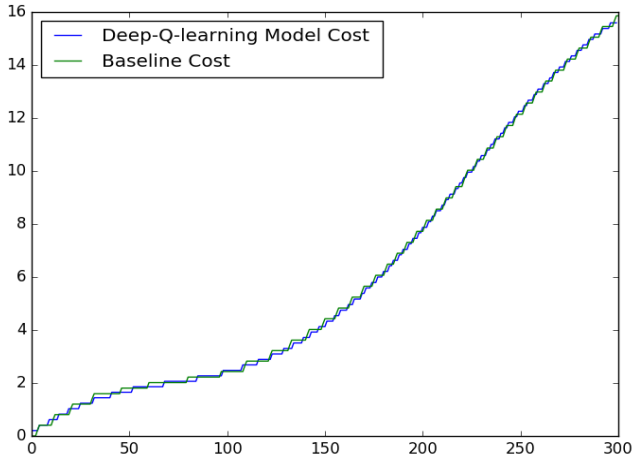



(b) Indoor temperature variation when the controller is rewarded for improving comfort and optimizing cost

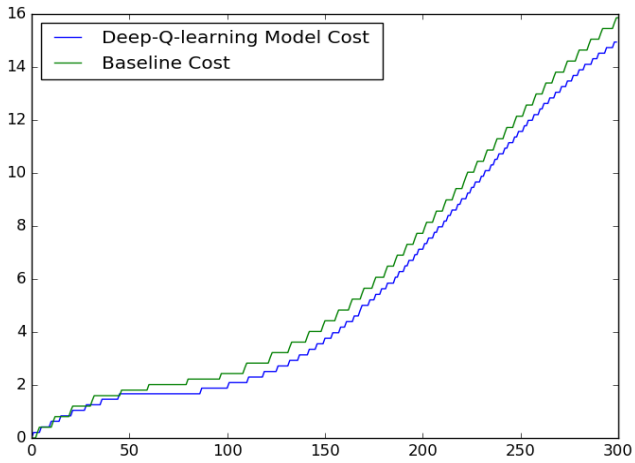
Figure 5.7: Comparison of the indoor temperature variations caused by baseline controller and DQN-based controller with different reward functions.

Figure 5.8(a) and 5.8(b) compares the cost of DQN-based controllers with the Baseline controller considering Equation 4.6 reward function and Equation 4.7 reward function, respectively. The graph shows the cumulative cost for one day after training for 800 epochs. A drop of 5% is observed when a factor of cost

is added to the reward function. While only 1% decrease is observed when user comfort is the only goal of the controller.




(a) Cumulative cost when the controller is rewarded for maintaining comfort




(b) Cumulative cost when the controller is rewarded for maintaining comfort and saving energy costs

Figure 5.8: Comparison of cumulative cost of the Baseline controller with DQN-based controller considering different reward functions.

The results confirm that deep reinforcement learning is a better approach than reinforcement learning for designing heating controllers with continuous and large

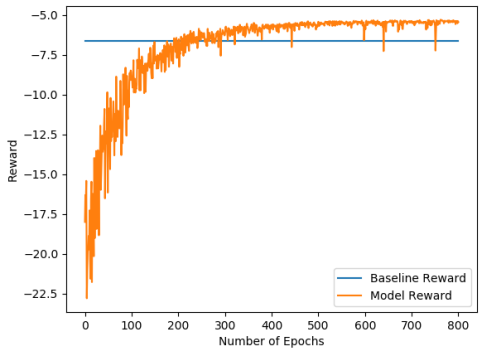
state space.

### 5.5.2 Analysis using Real-World Outdoor Temperature Data

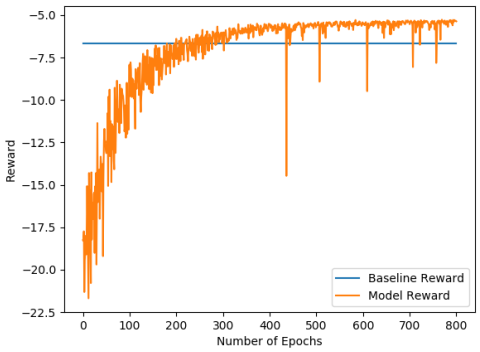
Results discussed in the previous section demonstrate the better performance of DQN-based controller than baseline controllers (based on a threshold) when synthetic data for outside temperature is considered. In order to study the performance of the proposed DQN-based heating controller in real-word cases, we collected weather temperature, as discussed in section 5.1.1.2 for a further set of experiments. The DQN-based controller is trained for each day of March 2019. Instead of passing a sine wave to generate synthetic data, a comma-separated file containing the weather temperature data is passed to the simulated environment. The data is sampled on an hourly basis and interpolated between one-hour time slot to convert into twelve 5-minute interval data.

#### 5.5.2.1 Integrated Cost and Comfort driven Reward Analysis under Real-World Outdoor Temperature

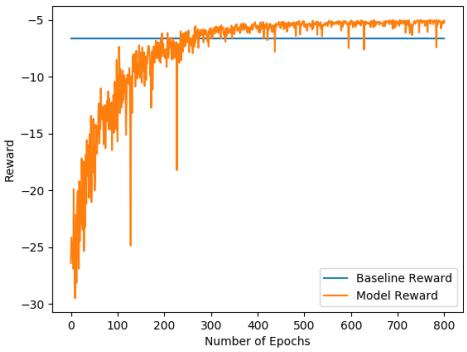
Figure 5.9 shows the rewards obtained by the DQN-based heating controllers for six days over a training length of 800 episodes. The reward values for Baseline controller are termed as Baseline reward while the reward values for DQN-based controller are termed as a model reward in Figure 5.9. The cost and temperature difference are normalized between 0 and 1 to keep them on the same scale and avoid biased results.



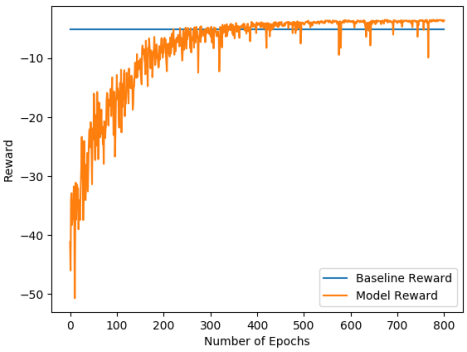

(a) March 26



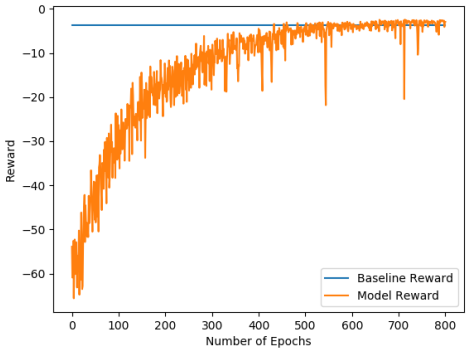

(b) March 27



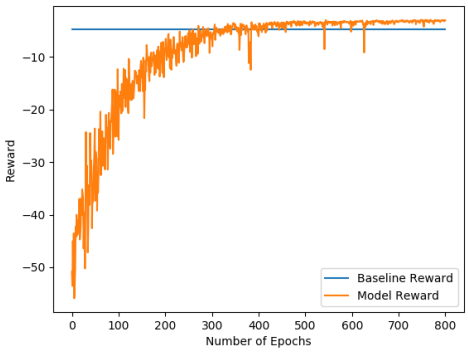

(c) March 28




(d) March 29




(e) March 30



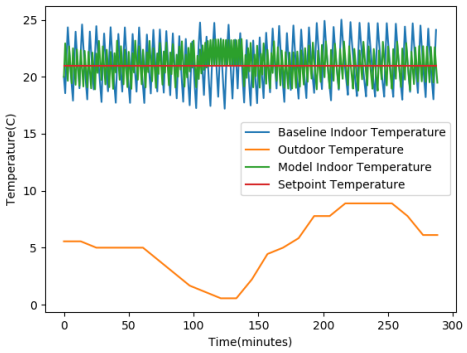

(f) March 31

Figure 5.9: Comparison of the cost and comfort-driven reward of DQN-based heating controller (Model Reward) the baseline controller (Baseline Reward) for six representative days of March 2019.

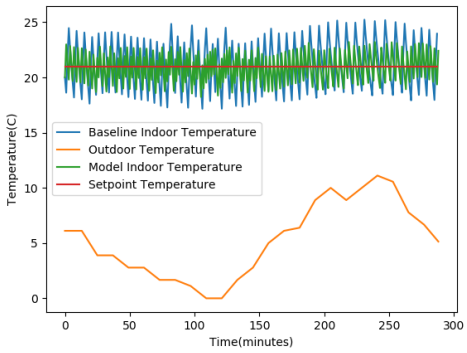
For all the days, the DQN-based controller shows increasing reward over training period and higher values than the baseline controller.

#### 5.5.2.2 Statistical Analysis of DQN under Real-World Outdoor Temperature

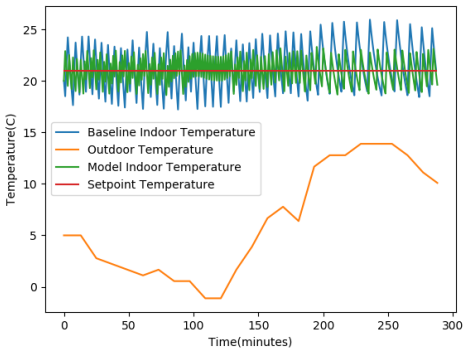
Figure 5.10 shows the indoor temperature variations with different outdoor temperature patterns for six representative days of March and illustrate the closeness of indoor temperature to the setpoint temperature. The blue (baseline controller) oscillates farther from the setpoint and that green (DQN-based controller) stays closer to the setpoint. AN interesting scenario is observed in Figure 5.10(e) where the outdoor temperature exceeds the setpoint temperature. The DQN controller worked well even under this case which shows that it is well-trained and can handle different situations.



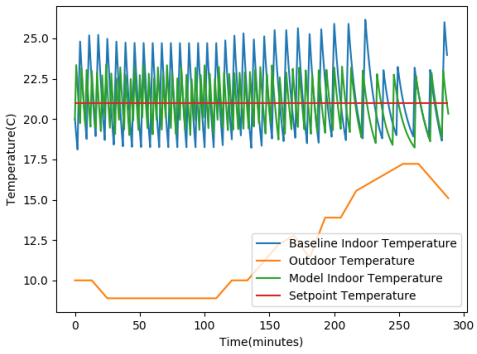

(a) March 26



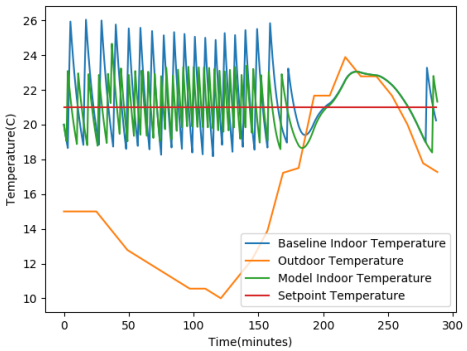

(b) March 27



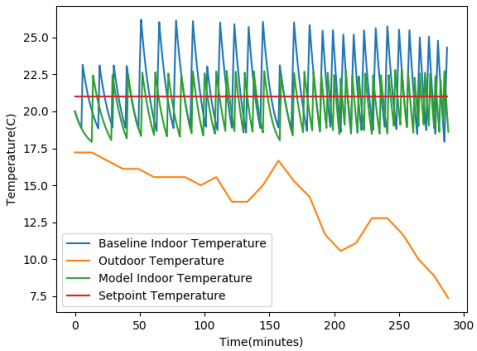

(c) March 28




(d) March 29



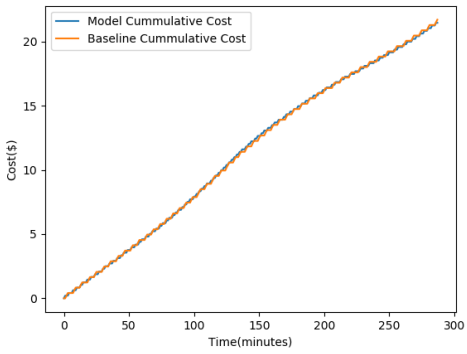

(e) March 30



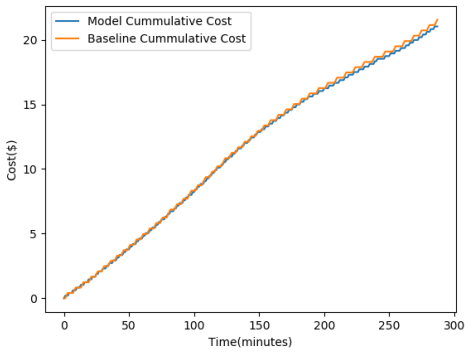

(f) March 31

Figure 5.10: Comparison of indoor temperature variations of DQN-based controller(Model Indoor Temperature) with the baseline controller(Bas -eline Indoor Temperature) for six days of March 2019.

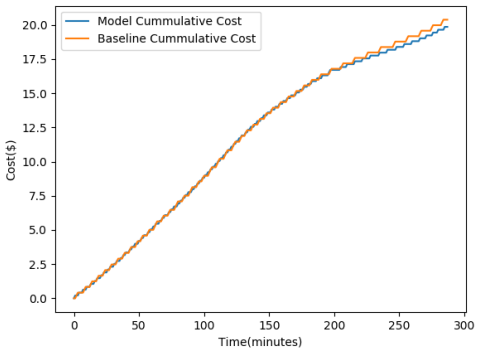
Figure 5.11 shows the cumulative cost computed over the same six days shown above. A small decrease in cost can be observed for March 26 to March 29, but a significant drop in cost can be observed for March 30 and March 31. One possible explanation is the higher outdoor temperatures for the last two days of the month, and therefore the less heating is required compared to other days.



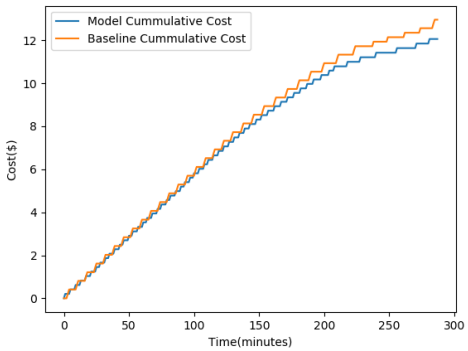

(a) March 26



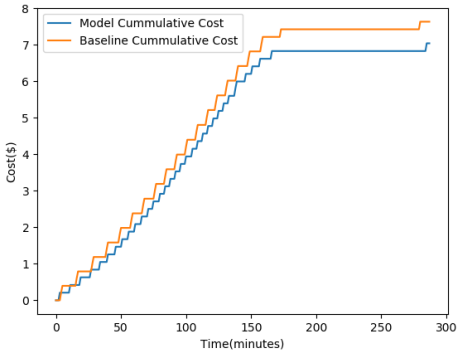

(b) March 27



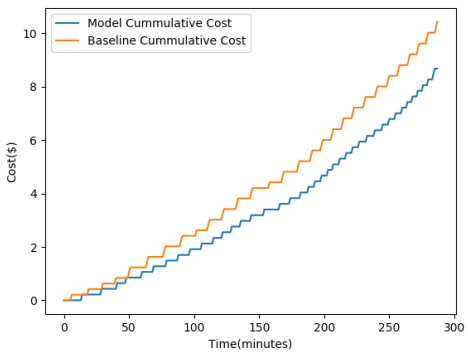

(c) March 28




(d) March 29




(e) March 30




(f) March 31

Figure 5.11: Comparison of cumulative cost of DQN-based controller (Model Cumulative cost) with the baseline controller (Baseline Cumulative cost) for six days of March 2019.

The mean, standard deviation, and the cumulative cost are averaged over the last 50 epochs to evaluate the performance of the DQN-based controller with the baseline controller (illustrated in Table 5.6) for six representative days. There is an average decrease of 25% in mean and 30% in standard deviation for the whole month (results for other days are illustrated in Appendix:Table A.1) while the drop in cost varies from 2% - 12% .

| **Day** | **Baseline Controller** | | | **DQN-based Controller** | | |
|---|---|---|---|---|---|---|
| | Cost($) | Mean(°C) | Std(°C) | Cost($) | Mean(°C) | Std(°C) |
| March 26 | 21.49 | 1.67 | 1.12 | 20.77 | 1.28 | 0.76 |
| March 27 | 21.35 | 1.68 | 1.15 | 20.83 | 1.28 | 0.78 |
| March 28 | 20.38 | 1.75 | 1.19 | 19.56 | 1.28 | 0.77 |
| March 29 | 12.96 | 1.75 | 1.16 | 11.94 | 1.29 | 0.78 |
| March 30 | 7.63 | 1.59 | 1.10 | 6.74 | 1.49 | 0.95 |
| March 31 | 10.24 | 1.77 | 1.22 | 9.15 | 1.29 | 0.79 |
| Average | 15.675 | 1.71 | 1.15 | 14.83 | 1.31 | 0.81 |

Table 5.6: Statistical comparison of DQN-based controller with baseline controller

## 5.6 Validating and Testing DQN-based Heating Controllers

In the previous set of experiments, the DQN-based controller is trained considering each day of March separately. However, we wanted to test the algorithm under realistic outdoor temperature patterns. Therefore, we decided to split the March 2019 weather data into 70% , 20% and 10% for training, validation and testing of the DQN-based controller respectively. The training data is used to train the controller; the validation data is used for hyperparameter tuning, whereas the testing data is used to test the tuned controller on unseen data.

The DQN-based controller is trained on data combined from 22 days, which is randomly picked from a pool of 31 days of March 2019. The controller is trained for 800 episodes. The trained controlled is validated on six days and tested on three days. We are essentially trying to replicate a situation where the DQN-based controller is implemented in a real building. This is equivalent to letting the controller train in the first 22 days of the month and then evaluating its performance for the rest of the month.

### 5.6.1 Cost and Comfort driven Reward Analysis for Validating DQN-based Controllers

The reward given to the DQN-based controller for taking action shows a rise with increasing training length in Figure 5.12. The curve starts to stabilize after 350 episodes of training and shows a small variation after 700 episodes. It outperforms the baseline controller with a higher reward of -141.

### 5.6.2 Statistical Analysis for Validation DQN-based Controllers

The subsections describe the statistical measures computed for training, validation, and testing of the DQN-based controller.

#### 5.6.2.1 Training

The DQN-based heating controller is trained on 22 days (selected randomly from 31 days of Mach 2019) of March 2019. After 800 episodes of training, the controller is run on the for 1 episode (shown in Figure 5.12) to compute the performance on the training dataset. The controller shows 27.6% and 29.2% decrease in mean and

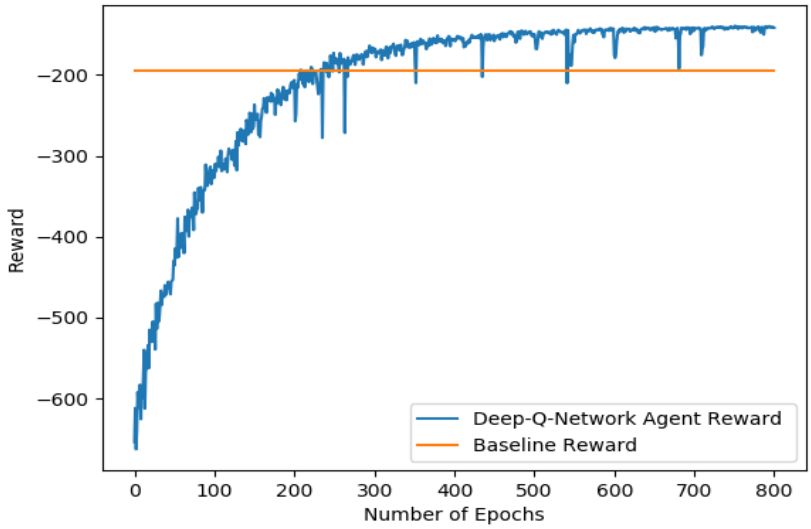
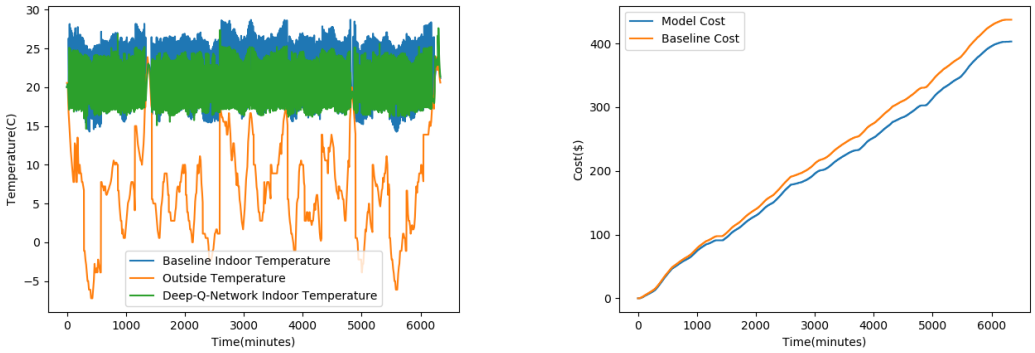



Figure 5.12: Reward comparison of DQN-based controller with Baseline controller for training dataset(22 days of March 2019)

standard deviation, respectively and a drop of 3.7% in total cost illustrated in Table 5.7 and shown in Figure 5.13.




(a) Indoor temperature variations

(b) Cumulative cost

Figure 5.13: Comparison of Indoor temperature variations and cost of DQN-based controller with baseline controller on training dataset (22 days).

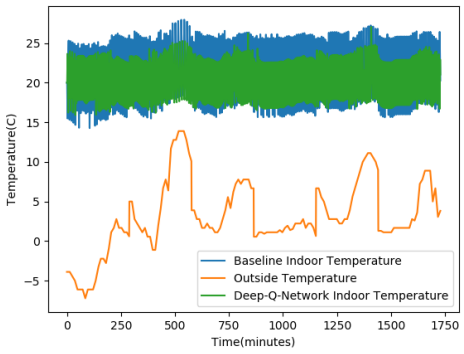
| DQN-based Controller | | | Baseline Controller | | |
|---|---|---|---|---|---|
| Mean(°C) | Std(°C) | Cost($) | Mean(°C) | Std(°C) | Cost($) |
| 1.75 | 1.14 | 421.41 | 2.42 | 1.61 | 437.42 |

Table 5.7: Statistical comparison of DQN-based controller with Baseline controller on training dataset(22 days).

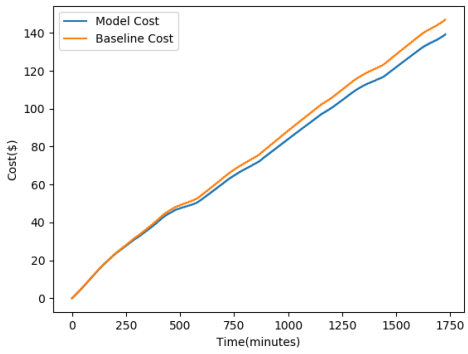
#### 5.6.2.2 Validation

Validation dataset is used to hypertune the training length of the DQN algorithm. Six days are chosen randomly from the 9 remaining days in that month that were not included in the training set. The reward of the DQN controller discussed in the previous section shows less variations after episode number 650. It signifies the stability in the training of the controller. Therefore, after 650 episodes of training, the trained model is validated after every 25 episodes on the validation outside temperature dataset (shown in Figure 5.14). The difference in the upper and lower bounds of green and blue in Figure 5.14 shows the range of indoor temperature variations. The narrower this range, the better the performance in terms of staying close to the setpoint (which we treat as a measure of occupant comfort here).

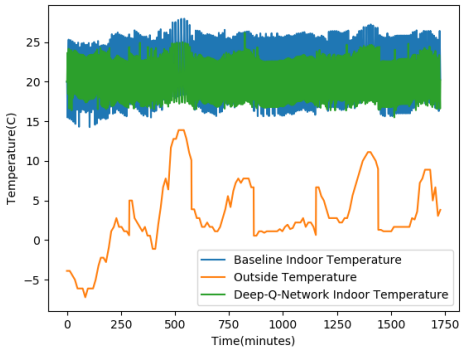
The overall performance of the trained DQN controller is better than the Baseline controller on the validation dataset. A 5.8% drop in energy cost can be observed after training for 775 episodes. Therefore, the controller trained till 775 episodes is considered best for testing. Training length is a hyperparameter, as discussed in section 5.2. We tune the training length from 800 to 775 episodes to get the best performance on test data. This is also called early stopping. Table 5.8 compares the mean, standard deviation, and cumulative cost of DQN-based controller with baseline controller for every 25 episodes after training for 650 episodes.



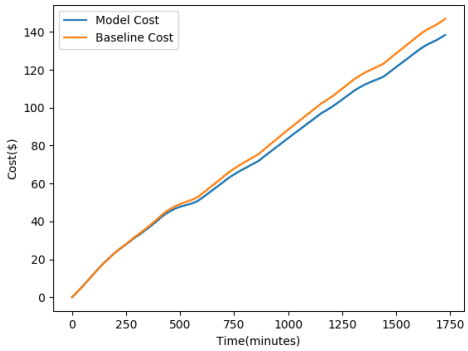

(a) Indoor temperature variations after training for 650 episodes



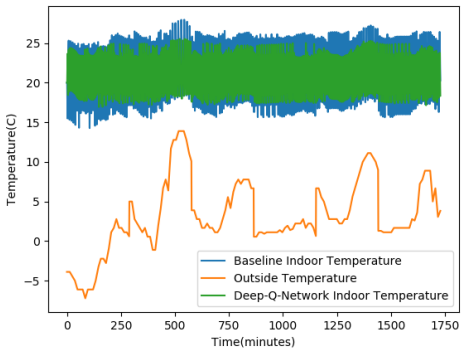

(b) Cumulative costs after training for 650 episodes



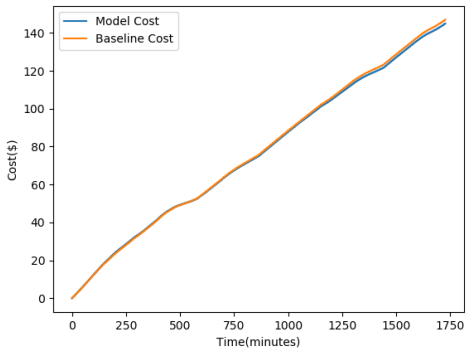

(c) Indoor temperature variations after training for 775 episodes




(d) Cumulative costs after training for 775 episodes




(e) Indoor temperature variations after training for 800 episodes




(f) Cumulative costs after training for 800 episodes

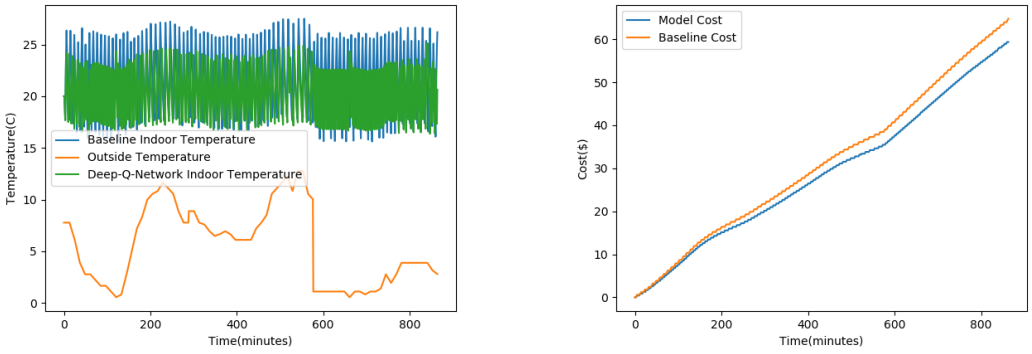
Figure 5.14: Comparison of indoor temperature variations and cumulative cost of DQN controller with baseline controller on validation data (6 days).

| Episode | DQN-based Controller | | | Baseline Controller | | |
|---|---|---|---|---|---|---|
| | Mean(°C) | Std(°C) | Cost($) | Mean(°C) | Std(°C) | Cost($) |
| 650 | 1.76 | 1.26 | 139.27 | 2.51 | 1.57 | 146.95 |
| 675 | 1.69 | 1.18 | 144.5 | 2.51 | 1.57 | 146.95 |
| 700 | 1.72 | 1.23 | 140.76 | 2.51 | 1.57 | 146.95 |
| 725 | 1.75 | 1.24 | 139.67 | 2.51 | 1.57 | 146.95 |
| 750 | 1.68 | 1.19 | 142.78 | 2.51 | 1.57 | 146.95 |
| **775** | **1.79** | **1.27** | **138.36** | **2.51** | **1.57** | **146.95** |
| 800 | 1.68 | 1.17 | 144.92 | 2.51 | 1.57 | 146.95 |

Table 5.8: Statistical comparison of DQN-based controller with baseline controller on validation data (6 days).

#### 5.6.2.3 Testing

The trained and validated DQN-based controller is tested on three days, all different from training and validation dataset as shown in Figure 5.15. This is done to test the controller on unseen data. The test results illustrated in Table 5.9 show a decrease of 3.2% in cost and 30% and 26.5 % drop in mean and standard deviation, respectively.




(a) Indoor temperature variations

(b) Cumulative cost

Figure 5.15: Comparison of indoor temperature variations and cost of DQN-based controller with baseline controller on test data (3 days).

| **DQN-based Controller** | | | **Baseline Controller** | | |
|---|---|---|---|---|---|
| Mean(°C) | Std(°C) | Cost($) | Mean(°C) | Std(°C) | Cost($) |
| 1.73 | 1.16 | 62.54 | 2.47 | 1.58 | 64.61 |

Table 5.9: Statistical comparison of DQN model with the baseline model on the test dataset (3 days).

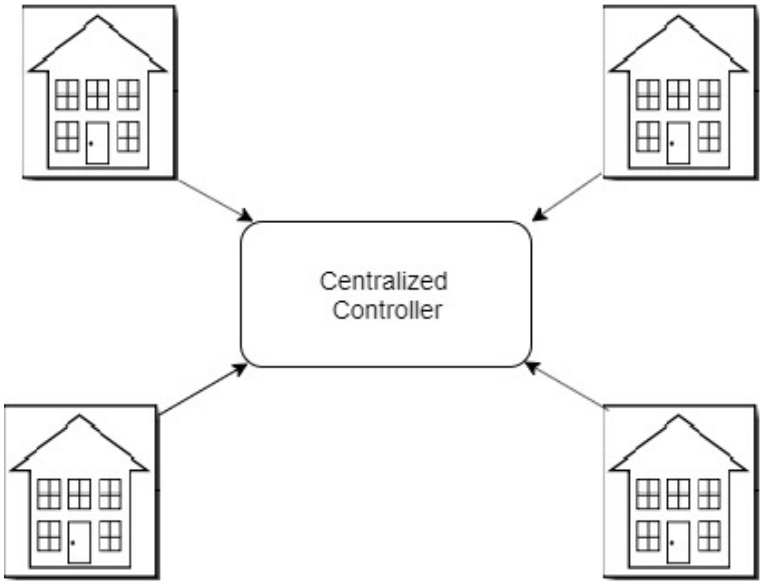
The experimental results show that the DQN-based controller outperforms the baseline controller for synthetic and real-world outside temperature data. It maintains a balance between improving user comfort and optimizing energy costs in all the scenarios. The DQN-based heating controller is a potential fit for buildings like small houses, gas stations, and restaurants, equipped with a single heating unit. The next section describes a heuristic approach that applies the DQN-based controller on multiple smart buildings/zones.

Chapter 6

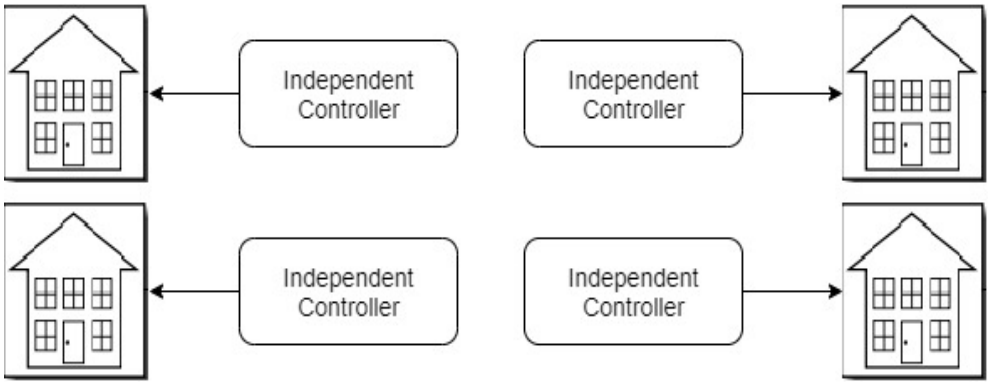
# Heuristics for Multiple Smart Buildings

Deployment of advanced control strategies into smart buildings comes with high implementation cost. Buildings have different types of structure and equipment. Designing specific controllers according to the design of each building can be next to impossible. Extending multiple controllers in the buildings which can interact with different environments can be feasible and also cost-efficient. For buildings like hotels, residential houses a centralized control of the heating systems can be beneficial. This chapter introduces the experiments performed to compare the performance of a centralized DQN-based controller with a decentralized DQN-based controller, when there are multiple buildings/zones to be managed (as shown in Figure 6.1) [72]. A simulated environment has been developed by replicating the building from Figure 4.1. All buildings have the same building material and structure like in residential apartment communities. The experiments are performed by testing different setpoint temperatures in each building. The tests have been performed for the case of 3 buildings and 6 buildings. The results of six buildings

are discussed in following sections and the results for the case of three buildings are provided in Appendix (Table A.2 to Table A.3).




(a) Centralized controller for multiple buildings/zones




(b) Independent controller for multiple buildings/zones

Figure 6.1: Control strategy for multiple buildings

## 6.1 Centralized Controller

A central DQN-based controller controls all of the buildings in the grid. The controller aims to maintain user comfort in each building and save the total cost across all the buildings.

### 6.1.1 Network Architecture of DQN-based Controller for Centralized Control of Multiple Buildings

#### 6.1.1.1 States, Actions and Reward for the Case of 6 Buildings

State in the DQN controller are defined by the difference between indoor and setpoint temperature for each building (6 values) and the outside temperature (1 value, common for all buildings as all buildings are assumed to be located in the same area).

$$State = \{ T^1_{diff}, T^2_{diff} ... T^6_{diff}, T_{out} \} \tag{6.1}$$

A combinatorial action space [73] is defined for the centralized DQN controller where each action is a combination of multiple interdependent sub-actions. The number of actions for $m$ buildings is equivalent to $2^m$ as we are representing actions as a binary number.

The reward function is the sum of the total cost from all the buildings and the sum of differences in the setpoint and indoor temperature in the buildings (given by Equation 6.2).

$$Reward = \beta \sum_{i=1}^{m} Cost^i + (1 - \beta) \sum_{i=1}^{m} T^i_{diff} \tag{6.2}$$

where $i$ is the index for each building (i=1, 2, ..., 6 in this case) and $\beta$ is the parameter defined to give weights to cost and user comfort.

#### 6.1.1.2 Neural Network

The number of neurons in the input layer of the neural network for $m$ buildings is $m+1$ where $m$ neurons are for $T_{diff}$ from each building, and one neuron is for the outside temperature. The output layer has $2^m$ neurons, each neuron representing the Q-value of combined action. In the case of 6 buildings, the neural network has 7 input neurons and 64 output neurons. The neurons in the 2 hidden layers are 64 and 128, respectively. The number of neurons are chosen based on the thumb rule which states that the number of hidden neurons should be 2/3 of the input layer size, plus the size of the output layer.

In DQN algorithm, we have two separate neural networks for Q-network and Target Q-network to avoid divergence in the predicted values (discussed in section 3.3.1.3). While performing experiments for one building, training of the Q-network and updating of the Target Q-network are done after every epoch. However, a divergence in the predicted values is observed, which leads to a sharp decrease in reward as we increase the number of buildings. It shows that the Q-network requires more frequent training and the Target Q-network requires frequent updating. Hence training of the Q-network is done every 100 iterations, and the target network is updated after every 200 iterations.

### 6.1.2 Experimental Results

The behavior of a centralized controller is observed by experimenting with the same, different, and paired setpoints in buildings as illustrated in Table 6.1. The model is trained for 800 episodes on March 2019 outside weather data. The data is divided into train, validate, and test with a ratio of 7:2:1 (similar to experiments when there was only one building, as discussed in section 5.6). The mean absolute

difference in the setpoint temperature and indoor temperature is computed to compare the user comfort maintained by the centralized controller with that of the baseline controller. The cumulative cost is calculated to compare energy costs.

| **Building** | **Case 1 (Same Setpoint) (°C)** | **Case 2 (Paired Setpoint) (°C)** | **Case 3 (Different Setpoint) (°C)** |
|---|---|---|---|
| Building 1 | 21 | 18 | 18 |
| Building 2 | 21 | 18 | 19 |
| Building 3 | 21 | 20 | 20 |
| Building 4 | 21 | 20 | 21 |
| Building 5 | 21 | 23 | 22 |
| Building 6 | 21 | 23 | 23 |

Table 6.1: Experimental strategy for multiple buildings

#### 6.1.2.1 Case 1: Same Setpoint

In the first experiment, all six buildings have a setpoint of 21°C. The results illustrated in Table 6.2 compare the mean and total cost of DQN-based controller with baseline controller for all six buildings Table 6.3 shows that the DQN-based centralized controller outperforms the baseline controller by improving the user comfort between 14% - 31% and decreasing energy costs between 2% - 8%.

#### 6.1.2.2 Case 2: Paired Setpoints

In the second experiment, the set point temperature for building 1 and 2 is 18°C, for building 3 and 4 is 20°C, and for building 5 and 6 is 23 °C. Table 6.4 and Table 6.5 shows that the DQN-based centralized controller outperforms the baseline controller by improving the user comfort between 8% - 26%. The DQN central

| Buildings | Setpoint (°C) | Baseline Controller | | | DQN Centralized Controller | | |
|---|---|---|---|---|---|---|---|
| | | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.86 | 1.75 | 1.81 |
| B2_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.86 | 1.75 | 1.81 |
| B3_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 2.07 | 2.16 | 1.98 |
| B4_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.87 | 1.76 | 1.806 |
| B5_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.88 | 1.74 | 1.81 |
| B6_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.88 | 1.75 | 1.802 |
| Total_Cost($) | | 2624.52 | 881.7 | 387.66 | 2425.58 | 867.81 | 374.6 |

Table 6.2: Mean and total c comparison of DQN-based centralized controller with baseline Controller where all the buildings are set to 21°C.

| Buildings | Setpoint (°C) | Improvements(%) | | |
|---|---|---|---|---|
| | | Train | Val | Test |
| B1_Mean(°C) | 21 | 23 | 30 | 27 |
| B2_Mean(°C) | 21 | 23 | 30 | 27 |
| B3_Mean(°C) | 21 | 14 | 14 | 20 |
| B4_Mean(°C) | 21 | 22 | 30 | 27 |
| B5_Mean(°C) | 21 | 22 | 31 | 27 |
| B6_Mean(°C) | 21 | 22 | 31 | 27 |
| Total_Cost($) | | 8 | 2 | 3 |

Table 6.3: Improvements in total cost and mean by DQN-based controller when compared to baseline controller for multiple buildings with same setpoint.

controller also saves energy costs between 4% - 5%. It can also be observed that buildings with the same setpoints have similar behavior.

#### 6.1.2.3 Case 3: Different Setpoints

In the third experiment, the buildings have different setpoints. The setpoint temperature for building 1 to 6 is 18°C, 19°C, 20°C, 21°C, 22°C, 23°C, respectively. Table 6.6 and Table 6.7 shows that the DQN-based centralized controller outper-

| **Buildings** | **Setpoint (°C)** | **Baseline Controller** | | | **DQN Centralized Controller** | | |
|---|---|---|---|---|---|---|---|
| | | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 18 | 2.59 | 2.61 | 2.67 | 2.11 | 1.93 | 2.03 |
| B2_Mean(°C) | 18 | 2.59 | 2.61 | 2.67 | 2.11 | 1.93 | 2.03 |
| B3_Mean(°C) | 20 | 2.47 | 2.55 | 2.55 | 2.11 | 2.35 | 2.17 |
| B4_Mean(°C) | 20 | 2.47 | 2.55 | 2.55 | 2.11 | 2.35 | 2.17 |
| B5_Mean(°C) | 23 | 2.28 | 2.33 | 2.33 | 2.03 | 1.93 | 1.92 |
| B6_Mean(°C) | 23 | 2.28 | 2.33 | 2.33 | 2.03 | 1.92 | 1.92 |
| Total_Cost($) | | 2160.6 | 728.87 | 320.28 | 2044.3 | 693.55 | 301.98 |

Table 6.4: Mean and total cost comparison of DQN-based centralized controller with baseline Controller where building 1 and 2 are set to 18, building 3 and 4 set 20, and building 5 and 6 are set to 23°C.

| **Buildings** | **Setpoint (°C)** | **Improvements(%)** | | |
|---|---|---|---|---|
| | | Train | Val | Test |
| B1_Mean(°C) | 18 | 19 | 26 | 24 |
| B2_Mean(°C) | 18 | 19 | 26 | 24 |
| B3_Mean(°C) | 20 | 15 | 8 | 15 |
| B4_Mean(°C) | 20 | 15 | 8 | 15 |
| B5_Mean(°C) | 23 | 11 | 17 | 18 |
| B6_Mean(°C) | 23 | 11 | 18 | 18 |
| Total_Cost($) | | 5 | 5 | 5 |

Table 6.5: Improvements in total cost and mean by DQN-based controller when compared to baseline controller for multiple buildings with paired setpoint.

forms the baseline controller by improving the user comfort between 5% - 26%. The central controller also saves energy costs between 4% - 5%.

## 6.2 Independent Controllers

Independent controllers are deployed in each building. The neural network architecture, states, actions, and reward function of each controller is same as the

| **Buildings** | **Setpoint (°C)** | **Baseline Controller** | | | **DQN Centralized Controller** | | |
|---|---|---|---|---|---|---|---|
| | | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 18 | 2.59 | 2.61 | 2.67 | 2.33 | 2.15 | 2.24 |
| B2_Mean(°C) | 19 | 2.53 | 2.5 | 2.58 | 1.98 | 1.89 | 1.93 |
| B3_Mean(°C) | 20 | 2.47 | 2.55 | 2.56 | 1.93 | 1.92 | 1.904 |
| B4_Mean(°C) | 21 | 2.42 | 2.51 | 2.47 | 2.2 | 2.32 | 2.23 |
| B5_Mean(°C) | 22 | 2.32 | 2.36 | 2.34 | 2.05 | 1.99 | 2.042 |
| B6_Mean(°C) | 23 | 2.28 | 2.33 | 2.33 | 2.17 | 2.18 | 2.202 |
| Total_Cost($) | | 2547.97 | 861.4 | 377.51 | 2433.34 | 826.5 | 358.4 |

Table 6.6: Mean and total cost comparison of DQN-based centralized controller with baseline controller where building 1 to 6 are set to range of 18-23°C respectively.

| **Buildings** | **Setpoint (°C)** | **Improvements(%)** | | |
|---|---|---|---|---|
| | | Train | Val | Test |
| B1_Mean(°C) | 18 | 18 | 26 | 16 |
| B2_Mean(°C) | 19 | 22 | 24 | 25 |
| B3_Mean(°C) | 20 | 22 | 25 | 26 |
| B4_Mean(°C) | 21 | 9 | 8 | 10 |
| B5_Mean(°C) | 22 | 12 | 16 | 13 |
| B6_Mean(°C) | 23 | 5 | 6 | 5 |
| Total_Cost($) | | 4 | 4 | 5 |

Table 6.7: Improvements in total cost and mean by DQN-based controller when compared to baseline controller for multiple buildings with different setpoint.

configurations for the case of a single building (discussed in chapter 4 and 5). Each controller aims to maintain user comfort and save cost in their buildings. In other words, we are training 6 independent DQN controllers, one for each building.

### 6.2.1 Experimental Results

The behavior of DQN-based independent controllers are observed when they do not share the reward function and have independent actions. Each controller is entirely independent of one another and does not have the aim to save the total cost in all buildings, but rather aim at optimizing comfort level and cost for their corresponding building.

#### 6.2.1.1 Case 1: Same Setpoints

In the first scenario, all buildings have the same setpoint temperature as 21°C. Tbale 6.8 and Table 6.9 shows that the independent controllers are able to outperform the baseline controller by improving user comfort between 27% - 33% and saving the energy costs between 1% - 3%.

| **Buildings** | **Setpoint (°C)** | **Baseline Controller** | | | **DQN Independent Controller** | | |
|---|---|---|---|---|---|---|---|
| | | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.76 | 1.69 | 1.72 |
| B2_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.77 | 1.71 | 1.72 |
| B3_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.77 | 1.71 | 1.75 |
| B4_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.74 | 1.68 | 1.723 |
| B5_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.75 | 1.72 | 1.74 |
| B6_Mean(°C) | 21 | 2.41 | 2.51 | 2.47 | 1.75 | 1.73 | 1.74 |
| Total_Cost($) | | 2624.52 | 881.7 | 387.66 | 2550.41 | 868.6 | 378.63 |

Table 6.8: Mean and total cost comparison of DQN-based independent controller with baseline controller where all the buildings are set to 21°C.

#### 6.2.1.2 Case 2: Paired Setpoints

In the second case, the setpoint for building 1 and 2 is set to 18°C, building 3 and 4 is set to 20°Celsius and building 5 and 6 is set to 21°C. Table 6.10 and Table

| **Buildings** | **Setpoint (°C)** | **Improvements(%)** | | |
|---|---|---|---|---|
| | | Train | Val | Test |
| B1_Mean(°C) | 21 | 27 | 32 | 30 |
| B2_Mean(°C) | 21 | 27 | 32 | 30 |
| B3_Mean(°C) | 21 | 27 | 32 | 30 |
| B4_Mean(°C) | 21 | 28 | 33 | 29 |
| B5_Mean(°C) | 21 | 27 | 31 | 30 |
| B6_Mean(°C) | 21 | 27 | 31 | 30 |
| Total_Cost($) | | 3 | 1 | 2 |

Table 6.9: Improvements in total cost and mean by DQN-based independent controllers when compared to baseline controller for multiple buildings with different setpoint.

6.11 shows that the independent controllers are able to improve the user comfort from 22% - 31% and save the cost from 3% - 4% .

| **Buildings** | **Setpoint (°C)** | **Baseline Controller** | | | **DQN Independent Controller** | | |
|---|---|---|---|---|---|---|---|
| | | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 18 | 2.59 | 2.61 | 2.67 | 2.03 | 1.97 | 2.03 |
| B2_Mean(°C) | 18 | 2.59 | 2.61 | 2.67 | 2.01 | 1.91 | 1.98 |
| B3_Mean(°C) | 20 | 2.47 | 2.55 | 2.55 | 1.88 | 1.76 | 1.78 |
| B4_Mean(°C) | 20 | 2.47 | 2.55 | 2.55 | 1.84 | 1.76 | 1.793 |
| B5_Mean(°C) | 23 | 2.28 | 2.33 | 2.33 | 1.73 | 1.79 | 1.79 |
| B6_Mean(°C) | 23 | 2.28 | 2.33 | 2.33 | 1.69 | 1.76 | 1.67 |
| Total_Cost($) | | 2522.75 | 855.86 | 375.1 | 2421.3 | 829.97 | 359.33 |

Table 6.10: Mean and total cost comparison of DQN-based independent controller with baseline controller where building 1 and 2 are set to 18°C, building 3 and 4 set 20°C, and building 5 and 6 are set to 23°C.

| **Buildings** | **Setpoint (°C)** | **Improvements(%)** | | |
|---|---|---|---|---|
| | | Train | Val | Test |
| B1_Mean(°C) | 18 | 22 | 25 | 24 |
| B2_Mean(°C) | 18 | 22 | 27 | 26 |
| B3_Mean(°C) | 20 | 24 | 31 | 30 |
| B4_Mean(°C) | 20 | 26 | 31 | 30 |
| B5_Mean(°C) | 23 | 24 | 33 | 23 |
| B6_Mean(°C) | 23 | 27 | 24 | 28 |
| Total_Cost($) | | 4 | 3 | 4 |

Table 6.11: Improvements in total cost and mean by DQN-based independent controllers when compared to baseline controller for multiple buildings with paired setpoint.

#### 6.2.1.3 Case 3: Different Setpoints

In the third scenario, the DQN-based independent controllers are assigned to buildings having different setpoint ranging from 18 to 23. Table 6.12 and Table 6.13 shows that independent controllers are able to improve the user comfort between 7% - 49% and save the cost between 4% - 7%.

| **Buildings** | **Setpoint (°C)** | **Baseline Controller** | | | **DQN Independent Controller** | | |
|---|---|---|---|---|---|---|---|
| | | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 18 | 2.59 | 2.61 | 2.67 | 2.02 | 1.92 | 1.97 |
| B2_Mean(°C) | 19 | 2.53 | 2.5 | 2.58 | 2.12 | 1.95 | 1.98 |
| B3_Mean(°C) | 20 | 2.47 | 2.55 | 2.56 | 1.26 | 1.75 | 1.784 |
| B4_Mean(°C) | 21 | 2.42 | 2.51 | 2.47 | 1.79 | 1.69 | 1.72 |
| B5_Mean(°C) | 22 | 2.32 | 2.36 | 2.34 | 2.15 | 1.96 | 1.97 |
| B6_Mean(C) | 23 | 2.28 | 2.33 | 2.33 | 1.65 | 1.71 | 1.67 |
| Total_Cost($) | | 2547.97 | 861.4 | 377.51 | 2378.63 | 825.5 | 356.67 |

Table 6.12: Mean and total cost comparison of DQN-based independent controller with baseline Controller where building 1 to 6 are set to range of 18-23°C respectively.

| **Buildings** | **Setpoint (°C)** | **Improvements(%)** | | |
|---|---|---|---|---|
| | | Train | Val | Test |
| B1_Mean(°C) | 18 | 22 | 26 | 26 |
| B2_Mean(°C) | 19 | 16 | 22 | 23 |
| B3_Mean(°C) | 20 | 49 | 31 | 30 |
| B4_Mean(°C) | 21 | 26 | 33 | 30 |
| B5_Mean(°C) | 22 | 7 | 17 | 16 |
| B6_Mean(°C) | 23 | 28 | 27 | 28 |
| Total_Cost($) | | 7 | 4 | 6 |

Table 6.13: Improvements in total cost and mean by DQN-based independent controllers when compared to baseline controller for multiple buildings with different setpoint.

## 6.3 Centralized Controller vs Independent Controllers

The DQN-based centralized and the independent controllers both outperform the baseline controller. This section compares the performance of the DQN-based centralized controller with independent controllers. The user comfort and the total cost is compared for both the controllers.

### 6.3.1 Case 1: Same Setpoint

On comparing central controller with a independent controller for buildings with the same setpoint of 21°C, the independent controllers are more efficient in improving user comfort across all the buildings. However, while comparing the total cost for all the buildings, the centralized controller outperforms the independent controllers. The Total_Cost in Figure 6.2 shows that the centralized controller saves energy costs between 2% - 7%. B3_Mean in Figure 6.2 i.e. mean absolute $T_{diff}$ of building 3 is lower as compared to other five buildings due to a combinatorial

action space for centralized controller.

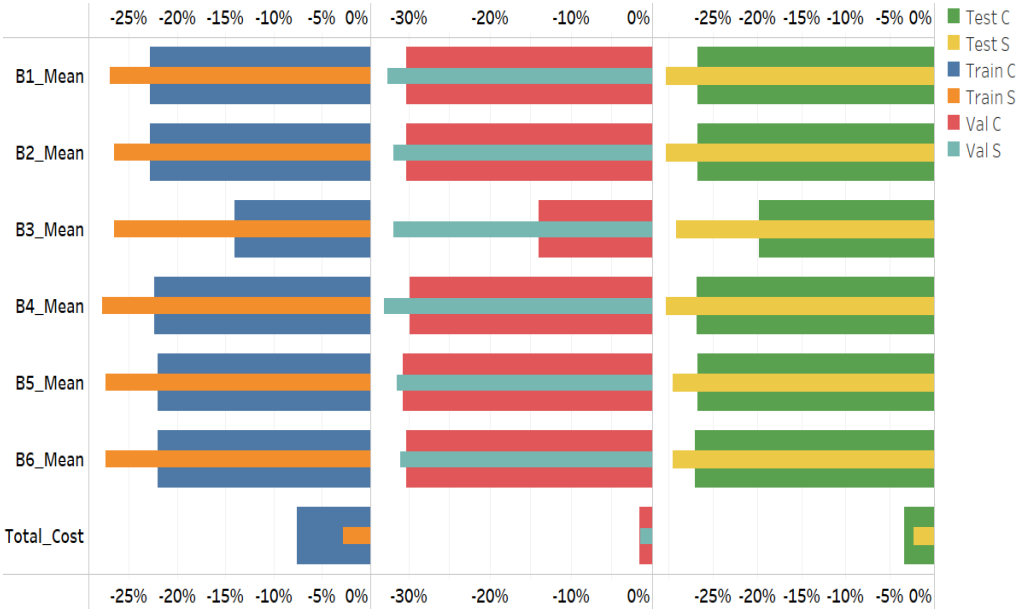



Figure 6.2: Comparison of mean and total cost for training, validation and test results of a central controller with independent controllers when all the buildings are set to 21°C setpoint.

### 6.3.2 Case 2: Paired Setpoints

For the case when the setpoint temperature of building 1 and 2 is 18°C, building 3 and 4 is set 20°C and building 5 and 6 is 23°C, the independent controllers still performs better in improving user comfort but the centralized controller outperforms in total cost saving between 4% - 5% as shown in Figure 6.3. The independent controllers shows a high percentage of improvement in user comfort for building 3 and building 4.

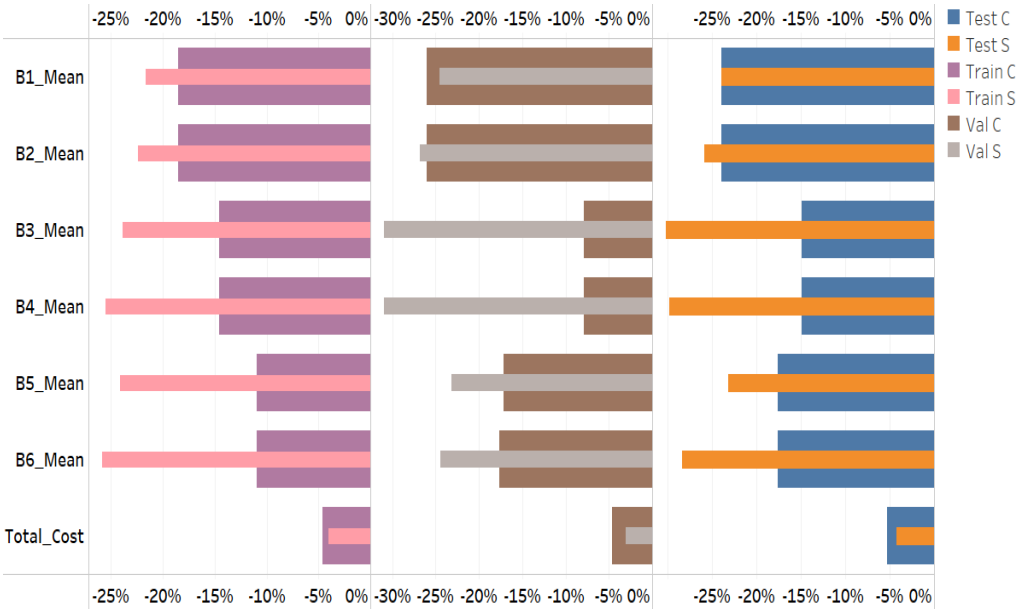



Figure 6.3: Comparison of mean and total cost for training, validation and test results of a central controller with independent controllers with paired setpoints.

### 6.3.3 Case 3: Different Setpoints

When all the buildings are set to different setpoints with a range from 18°C to 23°C respectively, the central controller is observed to maintain more user comfort in building 2 than the independent controller. However, the independent controllers still dominates to improve user comfort in most of the cases. Both controllers have comparable performance in saving cost. The independent controller could save between 4% - 7% while centralized controller could save between 4% - 5% as shown in Figure 6.4. The independent controller shows an improvement of 49% for training dataset in building 3 having setpoint of 20°C.

With all the above experiments, a trade-off is observed between the number of buildings, their settings, and performance. As the building configurations move from simple to complex i.e., from all buildings with the same setpoints to different setpoints, the independent controller performs better as compared to a centralized

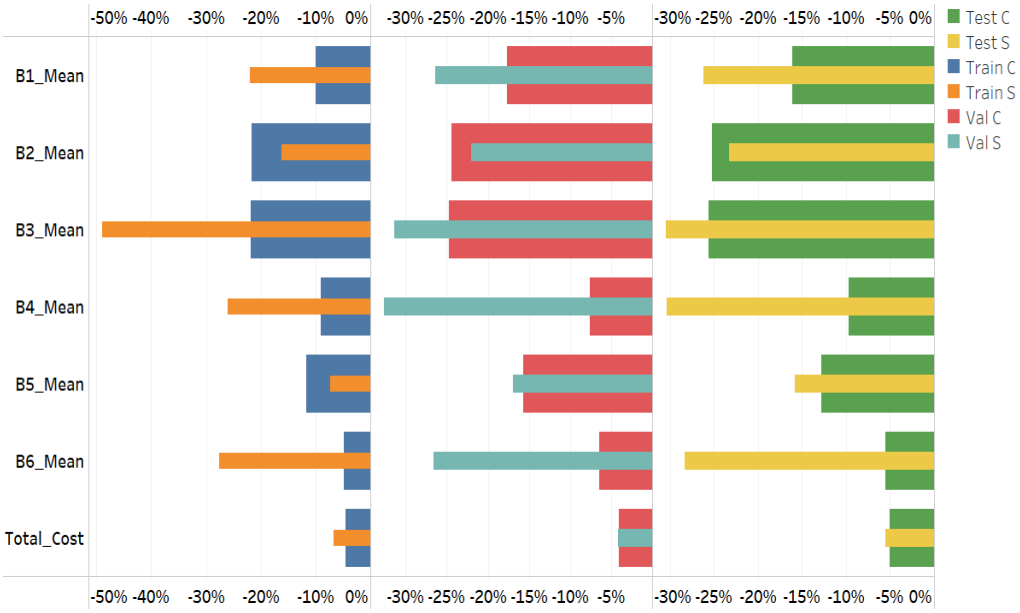



Figure 6.4: Comparison of Mean and Total Cost for Training, Validation and Test results for a central controller with independent controllers when building 1 to 6 are set to range of 18-23 degree Celsius respectively

controller. One potential reason of independent controllers dominating centralized controllers is the goal optimizing comfort and cost separately for each building. Though the independent controllers save energy costs in some of the cases when buildings have different setpoints but the central controller dominates to save total cost in all other cases. A centralized DQN controller can be a potential fit for buildings such as schools with multiple classrooms, shopping malls with multiple zones. Independent controllers will be a better choice for buildings which are equipped with multiple heating units but changing user preferences in different buildings/zones such as hotels, and residential apartments.

Chapter 7

# Conclusion & Future Work

This research presents a novel heating controller for optimizing performance in smart buildings using deep reinforcement learning. The main goal of the smart heating controller is to regulate the temperature inside the building so that occupant comfort is improved and energy costs are reduced. The DQN-based controller implemented is expected to be a good representation of the real-world situation. However, it does not consider the complex interactions of occupants such as changing setpoint temperature preferences with time. The results obtained from reinforcement and deep reinforcement learning controllers are compared to the baseline controllers, which depend on a threshold for their actions. The outcomes provide the upper and lower bounds for the performance achievable with DQN-based heating controllers. Based on the experiments performed in the research, the DQN-based heating controllers outperforms the regular baseline controllers in all case studies.

The research also extends the idea of comparing a centralized controller for multiple buildings with independent controller deployed in each building. Both the centralized and independent controllers outperforms the baseline controllers but

with an increasing number of buildings and variations in setpoints, separate controllers seem to be preferable and feasible.

For future research, we plan to test the controllers on complex simulated environments that consider complex interactions of occupants and aim to solve problems like high energy consumption during peak hours [73]. The complex interactions of users with equipment may include feedback on their satisfaction. The complex simulation models can include various equipment such as lights, air conditioners [73]. This would require designing of multi-objective reward function where the DRL controller will manage the actions of various equipments while considering user feedback and peak hours. The main goal of the DRL controller will be to maximize user comfort while minimizing energy costs. This way, the DRL controller can also learn the patterns of equipment usage by the occupants and can suggest to them the efficient use of devices. We will also study the effects of different outdoor conditions like wind speed, humidity, and building structure with different insulation properties on DQN-based heating controllers [74]. These parameters can significantly affect the results of DRL-based controllers. A systematic study in this aspect will help to select the right design. Different heating systems may have dynamic characteristics, such as decreasing efficiency due to equipment aging, changing operational schedules. Therefore, it is necessary to study how DRL-based controllers can adapt to changing system characteristics. Also, more experiments will be performed for the centralized and separate controllers, which can give a clear picture of the trade-off between the number of buildings and control mechanisms.

To conclude, this research is an attempt to provide a real-time decision-making heating controller for smart buildings using deep reinforcement Learning. The proposed controller enables effective real-time control without the need to ana-

lyze huge amounts of historical data. The DQN-based controller surpasses the traditional threshold-based controllers by maintaining a balance to improve user comfort and reduce electricity bills.

# Appendix

## A.1 Test Results for Different Days of March

The results in Table A.1 refer to section 5.5.2.1 and illustrates the results for different days for March 2019.

| **Day** | **Baseline Controller** | | | **DQN based Controller** | | |
|---|---|---|---|---|---|---|
| | Cost($) | Mean(°C) | Std(°C) | Cost($) | Mean(°C) | Std(°C) |
| March 1 | 28.04 | 1.71 | 1.19 | 27.75 | 1.23 | 0.65 |
| March 2 | 25.52 | 1.65 | 1.148 | 25.055 | 1.26 | 0.75 |
| March 3 | 23.76 | 1.57 | 1.104 | 23.39 | 1.25 | 0.73 |
| March 4 | 25.97 | 1.64 | 1.22 | 25.75 | 1.25 | 0.701 |
| March 5 | 31.35 | 1.62 | 1.15 | 30.83 | 1.22 | 0.71 |
| March 21 | 17.93 | 1.75 | 1.12 | 17.21 | 1.23 | 0.76 |
| March 22 | 18.36 | 1.74 | 1.11 | 17.51 | 1.24 | 0.75 |
| March 23 | 20.26 | 1.66 | 1.13 | 19.73 | 1.25 | 0.75 |
| March 24 | 16.66 | 1.65 | 1.12 | 15.87 | 1.29 | 0.81 |
| March 25 | 14.53 | 1.85 | 1.201 | 13.61 | 1.29 | 0.78 |

Table A.1: Statistical comparison of baseline controller with DQN based controller for different days of March 2019.

## A.2 Results for 3 Buildings

Results illustrated in Table A.2 to Table A.4 are reference to section 6.1 and section 6.2 respectively. Table A.2 and A.3 illustrate the results for a centralized DQN controller for 3 buildings. Table A.4 and A.5 illustrate the results for a independent DQN controller for 3 buildings.

| **Buildings** | **Baseline Controller** | | | **DQN Centralized Controller** | | |
|---|---|---|---|---|---|---|
| | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 2.41 | 2.51 | 2.47 | 1.95 | 1.97 | 1.89 |
| B2_Mean(°C) | 2.41 | 2.51 | 2.47 | 1.98 | 1.87 | 1.78 |
| B3_Mean(°C) | 2.41 | 2.51 | 2.47 | 1.97 | 2.04 | 1.87 |
| Total_Cost($) | 1312.26 | 440.88 | 193.84 | 1251.295 | 425.28 | 185.71 |

Table A.2: Statistical comparison of baseline controller with DQN-based Centralized Controller for 3 buildings (B1,B2 and B3) set to 21°C setpoint temperature.

| **Buildings** | **Baseline Controller** | | | **DQN Centralized Controller** | | |
|---|---|---|---|---|---|---|
| | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 2.59 | 2.61 | 2.67 | 2.09 | 2.07 | 2.05 |
| B2_Mean(°C) | 2.417 | 2.51 | 2.47 | 1.9 | 1.85 | 1.79 |
| B3_Mean(°C) | 2.27 | 2.32 | 2.3 | 1.8 | 1.74 | 1.71 |
| Total_Cost($) | 1312.482 | 441.64 | 194 | 1256.72 | 425.72 | 183.14 |

Table A.3: Statistical comparison of baseline controller with DQN-based centralized controller for 3 buildings (B1,B2 and B3) set to 18°C, 21°C and 23°C setpoint temperature. respectively.

| **Buildings** | **Baseline Controller** | | | **DQN Separate Controller** | | |
|---|---|---|---|---|---|---|
| | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 2.41 | 2.51 | 2.47 | 1.76 | 1.69 | 1.72 |
| B2_Mean(°C) | 2.41 | 2.51 | 2.47 | 1.77 | 1.71 | 1.72 |
| B3_Mean(°C) | 2.41 | 2.51 | 2.47 | 1.77 | 1.71 | 1.75 |
| Total_Cost($) | 1312.2 | 440.856 | 193.83 | 1275.63 | 433.91 | 189.5 |

Table A.4: Statistical comparison of baseline controller with DQN-based separate controllers for 3 buildings (B1,B2 and B3) set to 21°C setpoint temperature.

| **Buildings** | **Baseline Controller** | | | **DQN Separate Controller** | | |
|---|---|---|---|---|---|---|
| | Train | Val | Test | Train | Val | Test |
| B1_Mean(°C) | 2.59 | 2.61 | 2.67 | 2.03 | 1.97 | 2.03 |
| B2_Mean(°C) | 2.42 | 2.51 | 2.47 | 1.79 | 1.69 | 1.72 |
| B3_Mean(°C) | 2.28 | 2.33 | 2.33 | 1.73 | 1.79 | 1.79 |
| Total_Cost($) | 1287.09 | 434.62 | 190.74 | 1228.66 | 421.81 | 182.49 |

Table A.5: Statistical comparison of baseline controller with DQN-based separate controller for 3 buildings (B1,B2 and B3) set to 18°C, 21°C and 23°C setpoint temperature respectively.

# Bibliography


[1] P. Nejat, F. Jomehzadeh, M. M. Taheri, M. Gohari, and M. Z. Abd. Majid, "A global review of energy consumption, CO2 emissions and policy in the residential sector," *Renewable and Sustainable Energy Reviews*, vol. 43, no. C, pp. 843–862, 2015.

[2] B. Qolomany, A. Al-Fuqaha, A. Gupta, D. Benhaddou, S. Alwajidi, J. Qadir, and A. C. Fong, "Leveraging machine learning and big data for smart buildings: A comprehensive survey," *Computing Research Repository (CoRR)*, 2019.

[3] L. Pérez-Lombard, J. Ortiz, and C. Pout, "A review on buildings energy consumption information," *Energy and Buildings*, vol. 40, no. 3, pp. 394–398, 2008.

[4] L. Yang, H. Yan, and J. C. Lam, "Thermal comfort and building energy consumption implications – A review," *Applied Energy*, vol. 115, pp. 164–173, 2014.

[5] D. H. W. Li, L. Yang, and J. C. Lam, "Impact of climate change on energy use in the built environment in different climate zones – A review," *Energy*, vol. 42, no. 1, pp. 103–112, 2012.

[6] D. Coumou and S. Rahmstorf, "A decade of weather extremes," *Nature Climate Change*, vol. 2, no. 7, pp. 491–496, 2012.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, Cambridge, Mass: MIT Press, 1998.

[8] X. Chen, X. Li, and S. X.-D. Tan, "From robust chip to smart building: CAD algorithms and methodologies for uncertainty analysis of building performance," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 457–464, IEEE, 2015.

[9] Y. Fu, Z. Li, H. Zhang, and P. Xu, "Using support vector machine to predict next day electricity load of public buildings with sub-metering devices," *Procedia Engineering*, vol. 121, pp. 1016–1022, 2015.

[10] K. Mary Reena, A. T. Mathew, and L. Jacob, "A flexible control strategy for energy and comfort aware HVAC in large buildings," *Building and Environment*, vol. 145, pp. 330–342, 2018.

[11] S. Bourobou and Y. Yoo, "User activity recognition in smart homes using pattern clustering applied to temporal ANN algorithm," *Sensors*, vol. 15, no. 5, pp. 11953–11971, 2015.

[12] I. Barbeito, S. Zaragoza, J. Tarrío-Saavedra, and S. Naya, "Assessing thermal comfort and energy efficiency in buildings by statistical quality control for autocomputing research repository (corr)elated data," *Applied Energy*, vol. 190, pp. 1–17, 2017.

[13] J. Ahn and S. Cho, "Anti-logic or common sense that can hinder machine's energy performance: Energy and comfort control models based on artificial intelligence responding to abnormal indoor environments," *Applied Energy*, vol. 204, pp. 117–130, 2017.

[14] T. Chaudhuri, Y. C. Soh, H. Li, and L. Xie, "Machine learning based prediction of thermal comfort in buildings of equatorial Singapore," in *IEEE International Conference on Smart Grid and Smart Cities (ICSGSC)*, pp. 72–77, 2017.

[15] D. Popa, F. Pop, C. Serbanescu, and A. Castiglione, "Deep learning model for home automation and energy reduction in a smart home environment platform," *Neural Computing and Applications*, vol. 31, no. 5, pp. 1317–1337, 2019.

[16] A. Almalaq and J. J. Zhang, "Evolutionary deep learning-based energy consumption prediction for buildings," *IEEE Access*, vol. 7, pp. 1520–1531, 2019.

[17] Y. Li, Z. Yan, S. Chen, X. Xu, and C. Kang, "Operation strategy of smart thermostats that self-learn user preferences," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5770–5780, 2019.

[18] J. R. Vázquez-Canteli and Z. Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied Energy*, vol. 235, pp. 1072–1089, 2019.

[19] Y. Sun, A. Somani, and T. E. Carroll, "Learning based bidding strategy for HVAC systems in double auction retail energy markets," in *2015 American Control Conference (ACC)*, pp. 2912–2917, IEEE, 2015.

[20] Z. Cheng, Q. Zhao, F. Wang, Y. Jiang, L. Xia, and J. Ding, "Satisfaction based Q-learning for integrated lighting and blind control," *Energy and Buildings*, vol. 127, pp. 43–55, 2016.

[21] J. Brusey, D. Hintea, E. Gaura, and N. Beloe, "Reinforcement learning-based thermal comfort control for vehicle cabins," *Mechatronics*, vol. 50, pp. 413–421, 2018.

[22] Y. Chen, L. Norford, H. Samuelson, and A. Malkawi, "Optimal control of HVAC and window systems for natural ventilation through reinforcement learning," *Energy and Buildings*, vol. 169, pp. 195–205, 2018.

[23] S. Zhou, Z. Hu, W. Gu, M. Jiang, and X. Zhang, "Artificial intelligence based smart energy community management: A reinforcement learning approach," *CSEE Journal of Power and Energy Systems*, vol. 5, no. 1, pp. 1–10, 2019.

[24] P. Fazenda, K. Veeramachaneni, P. Lima, and U.-M. O'Reilly, "Using reinforcement learning to optimize occupant comfort and energy usage in HVAC systems," *J. Ambient Intell. Smart Environ.*, vol. 6, no. 6, pp. 675–690, 2014.

[25] T. Wei, Y. Wang, and Q. Zhu, "Deep reinforcement learning for building HVAC control," in *Proceedings of the 54th Annual Design Automation Conference*, pp. 1–6, ACM Press, 2017.

[26] G. Gao, J. Li, and Y. Wen, "Energy-efficient thermal comfort control in smart buildings via deep reinforcement learning," *Computing Research Repository (CoRR)*, 2019.

[27] A. Nagy, H. Kazmi, F. Cheaib, and J. Driesen, "Deep reinforcement learning for optimal control of space heating," *Computing Research Repository (CoRR)*, 2018.

[28] V. Shabunko, C. M. Lim, and S. Mathew, "EnergyPlus models for the benchmarking of residential buildings in Brunei Darussalam," *Energy and Buildings*, vol. 169, pp. 507–516, 2018.

[29] A. Charles, W. Maref, and C. M. Ouellet-Plamondon, "Case study of the upgrade of an existing office building for low energy consumption and low carbon emissions," *Energy and Buildings*, vol. 183, pp. 151–160, 2019.

[30] M. S. Shahdan, S. S. Ahmad, and M. A. Hussin, "External shading devices for energy efficient building," *Earth and Environmental Science*, vol. 117, p. 012034, 2018.

[31] J. S. Choi, S. H. Park, B. K. Jeon, and E. J. Kim, "Simulation analysis of energy saving effect of ERV on EHP heating energy consumption in a classroom," *Earth and Environmental Science*, vol. 238, p. 012061, 2019.

[32] J. Dejvises and N. Tanthanuch, "A simplified air-conditioning systems model with energy management," *Procedia Computer Science*, vol. 86, pp. 361–364, 2016.

[33] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction, Second Edition.* Springer Series in Statistics, Springer-Verlag, 2 ed., 2009.

[34] J. Lapalu, K. Bouchard, A. Bouzouane, B. Bouchard, and S. Giroux, "Unsupervised mining of activities for smart home prediction," *Procedia Computer Science*, vol. 19, pp. 503–510, 2013.

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[36] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: A modern approach.* Prentice hall series in artificial intelligence, Upper Saddle River: Prentice Hall, 3rd ed., 2010.

[37] M. F. Balcan and K. Q. Weinberger, *33rd International conference on machine learning (ICML).* Curran Associates, Inc, 2016.

[38] D. Huang, J.-B. Tristan, and G. Morrisett, "Compiling markov chain monte carlo algorithms for probabilistic modeling," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 111–125, ACM, 2017.

[39] S. Misra, A. Mondal, S. Banik, M. Khatua, S. Bera, and M. S. Obaidat, "Residential energy management in smart grid: A markov decision process-based approach," in *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pp. 1152–1157, 2013.

[40] J. Zhu, "A feedback optimal control by Hamilton–Jacobi–Bellman equation," *European Journal of Control*, vol. 37, pp. 70–74, 2017.

[41] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press, 2017.

[42] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems 29*, pp. 2154–2162, Curran Associates, Inc., 2016.

[43] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," in *Advances in Neural Information Processing Systems 30*, pp. 2775–2785, Curran Associates, Inc., 2017.

[44] M. L. Littman, "Reinforcement learning improves behaviour from evaluative feedback," *Nature*, vol. 521, no. 7553, pp. 445–451, 2015.

[45] D. Arumugam, D. Abel, K. Asadi, N. Gopalan, C. Grimm, J. K. Lee, L. Lehnert, and M. L. Littman, "Mitigating planner overfitting in model-based reinforcement learning," *Computing Research Repository (CoRR)*, 2018.

[46] S. Rahili, B. Riviere, S. Olivier, and S. Chung, "Optimal routing for autonomous taxis using distributed reinforcement learning," in *IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 556–563, 2018.

[47] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[48] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[49] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Advances in Neural Information Processing Systems 29*, pp. 2270–2278, Curran Associates, Inc., 2016.

[50] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[51] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling, "Visualizing deep neural network decisions: Prediction difference analysis," *Computing Research Repository (CoRR)*, 2017.

[52] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *Computing Research Repository (CoRR)*, 2017.

[53] J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with ReLU activation function," *Computing Research Repository (CoRR)*, 2017.

[54] C. Zhang and P. C. Woodland, "DNN speaker adaptation using parameterised sigmoid and ReLU hidden activation functions," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5300–5304, 2016.

[55] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *Computing Research Repository (CoRR)*, 2018.

[56] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2009.

[57] S. Ruder, "An overview of gradient descent optimization algorithms," *Computing Research Repository (CoRR)*, 2016.

[58] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[59] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the Deep Q-Network," *Computing Research Repository (CoRR)*, 2017.

[60] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, "Experience replay for continual learning," *Computing Research Repository (CoRR)*, 2018.

[61] N. R. Nielsen, "Application of artificial intelligence techniques to simulation," in *Knowledge-Based Simulation: Methodology and Application*, Advances in Simulation, pp. 1–19, Springer New York, 1991.

[62] L. Monostori, B. Kádár, Z. Viharos, I. Mezgár, and P. Stefán, "AI and ML techniques combined with simulation for designing and controlling manufacturing processes and systems," *IFAC Proceedings Volumes*, vol. 33, no. 20, pp. 181–186, 2000.

[63] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *Computing Research Repository (CoRR)*, 2016.

[64] E. Teng and B. Iannucci, "Learning to learn in simulation," *Computing Research Repository (CoRR)*, 2019.

[65] M. Zhu, X. Wang, and Y. Wang, "Human-like autonomous car-following model with deep reinforcement learning," *Transportation Research Part C: Emerging Technologies*, vol. 97, pp. 348–368, 2018.

[66] "Thermal Model of a House - MATLAB & Simulink." Available at `https://www.mathworks.com/help/simulink/slref/thermal-model-of-a-house.html`. Accessed: 2019-10-20.

[67] R. Durham, *Thermal Environmental Conditions for Human Occupancy*. Ashrae, 2004.

[68] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," *Computing Research Repository (CoRR)*, 2015.

[69] "Calling MATLAB from Python - MATLAB & Simulink." Available at `https://www.mathworks.com/help/matlab/matlab-engine-for-python.html`. Accessed: 2019-10-20.

[70] "TCP/IP Communication - MATLAB & Simulink." Available at `https://www.mathworks.com/help/matlab/tcpip-communication.html`. Accessed: 2019-10-20.

[71] "Pennsylvania State Climatologist." Available at `http://www.climate.psu.edu/`. Accessed: 2019-10-20.

[72] X. Zhou, D. Yan, and X. Shi, "Comparative research on different air conditioning systems for residential buildings," *Frontiers of Architectural Research*, vol. 6, no. 1, pp. 42–52, 2017.

[73] E. Mocanu, D. C. Mocanu, P. H. Nguyen, A. Liotta, M. E. Webber, M. Gibescu, and J. G. Slootweg, "On-line building energy optimization using deep reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 3698–3708, 2019.

[74] Z. Zhang, A. Chong, Y. Pan, C. Zhang, and K. P. Lam, "Whole building energy model for HVAC optimal control: A practical framework based on deep reinforcement learning," *Energy and Buildings*, vol. 199, pp. 472–490, 2019.