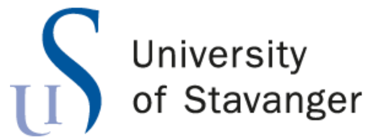

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Reinforcement Learning

Master's Thesis in Computer Science by

Ali Karooni

Supervisor

Dr. Nejm Saadallah

June 30, 2022

“The quickest road to success is to possess an attitude toward failure of 'no fear” Ralph Heath

Abstract

Electricity prices have risen significantly year on year and reducing energy use in homes can save money, improve energy security and reduce pollution from non-renewable energy sources. Whether to lower the monthly electricity bills or be concerned about the home's carbon footprint, reducing energy is helpful. The best way to start saving on electricity costs is to get smart with how electricity is being used.

The goal of this paper is to find an efficient approach to using electricity using machine learning algorithms. To achieve that, this thesis will apply q-learning, DQN with Replay Memory, and Double DQN with Replay Memory of Reinforcement Learning in python. The agent will interact with the Gym environment implemented from data given by Nova Smart company, achieving rewards upon reaching the goal or penalties based on the power price, time of the day, and comfort zone. Numerous studies have been conducted on this subject recently and there has been a lot of research. This work will demonstrate the behavior of the algorithms to meet the main criteria of trajectory design as an alternative solution.

Acknowledgements

I would like to thank my family and friends who have been supporting me and give the courage to tackle all the problems.

I am very grateful to my supervisors, DR. Nejm Saadallah and Terje A.H.Aas from Nova Smart company for providing me with this wonderful opportunity to pursue this thesis under them. Their guidance, ideas, encouragement, and insightful and valuable feedback amidst their busy schedule has been invaluable during the thesis.

Table of Contents

Abstract **1**

Acknowledgements **2**

Abbreviations **6**

Introduction **7**

1.1 Motivation 7

1.2 Problem Definition 7

1.3 Research Questions 8

1.4 Use cases 8

1.5 Challenges 8

1.6 Outline 9

Background **5**

2.1 Technical and Theoretical Background 5

2.1.1 Linear Regression 5

2.1.2 Long Short-Term Memory (LSTM) 5

2.1.3 Gym Environment 6

2.2.1 Machine Learning 6

2.2.2 Reinforcement Learning 7

2.2.3 Q-Learning 7

2.2.4 Deep Q-Learning (DQN) 9

2.2.5 Deep Reinforcement Learning with Replay Memory 9

2.2.6 Double DQN 10

Solution Approach **12**

3.1 Reinforcement Learning in Electric water heater (EWH) 12

3.1.1 Reinforcement Learning Applied to an Electric Water Heater: From Theory to Practice 12

3.1.2 Deep Reinforcement Learning for Autonomous Water Heater Control 13

3.2 Proposed Solution 13

Implementation 14

4.1 Preprocessing data 14

4.1.1 All data dataset 15

4.1.2 1440 data frame: 16

4.1.3 No water usage: 17

4.2 Daily power price: 18

4.3 Temperature predictor engine 19

4.3.1 Linear Regression: 19

4.3.2 Linear Regression fitting issues: 21

4.3.3 LSTM: 22

4.3.4 LSTM fitting issues: 24

4.4 Open AI Gym environment 25

4.5 Reinforcement Learning 29

4.5.1 Q-Learning: 29

4.5.2 DQN with Replay Memory: 30

4.5.3 Double DQN with Replay Memory: 32

Result 33

5.1 Q-Learning results 33

5.1.1 Q-Learning on all Data 34

5.1.2 Q-Learning on no water usage data 35

5.1.3 Q-Learning on 1440 dataset 36

5.2 DQN with Replay Memory results 37

5.2.1 DQN with Replay Memory on all Data 38

5.2.2 DQN with Replay Memory on no water usage dataset 39

5.2.3 DQN with Replay Memory 1440 dataset 40

5.3 Double DQN with Replay Memory results 41

5.3.1 Double DQN with Replay Memory on all data: 42

5.3.2 Double DQN with Replay Memory on no water usage dataset 43

5.3.3 Double DQN with Replay Memory on 1440 dataset 44

Discussion and Conclusion **45**

6.1 Discussion on the results 45

6.2 Limitations 45

6.3 conclusion 45

List of Figures **48**

Appendix A **51**

Downloadable content **51**

A.1 Source code 51

Bibliography **52**

Abbreviations

ML	Machine Learning
RL	Reinforcement Learning
QL	Q Learning
EWH	Electric Water Heater
NN	Neural Network
DQN	Deep Q-Network
LR	Linear Regression
LSTM	Long Sort-Term Memory
RNN	Recurrent Neural Network
AI	Artificial Intelligence

Chapter 1

Introduction

Energy conservation is a process that involves reducing the consumption of energy by making less use of it. This can be done through either using energy more efficiently or by reducing the amount of energy that's been used. By implementing energy conservation measures in buildings, they can help lower their energy costs and increase their environmental quality. They can also help prevent the depletion of resources. This is considered one of the top sustainable energy priorities [1].

1.1 Motivation

As smart houses are increasing, AI and data mining can be of particular benefit to reduce power consumption. An artificial intelligence system monitors and controls energy consumption in factories and buildings. It can analyze and monitor data collected by the system and sensors to learn to control systems more efficiently and reduce power usage. It can also reduce energy usage during peak hours or when there is no need for it [2].

1.2 Problem Definition

Due to the rising energy prices and being concerned about home's carbon footprint, controlling the energy usage of smart houses is very demanding. In smart houses, some sensors can measure the current temperature or indicate a status e.g., if the EWH power switch is On. Additionally, there are switches to turn on/off heaters, lights, appliances, etc. All data would be gathered periodically to be saved in cloud storage e.g., every minute. It turns out that a water heater is among the most power consumption appliance inside a house. It is common to have 70-degree temperature water. However, there are times when there is no need for heat water for example during the night. Also, it is possible to save some energy by reducing water temperature to some extent, especially during peak hours.

Reinforcement learning agent is a good practice to be left in an unknown environment to get to know it and with a bit of uncertainty, handle the situation with a good margin.

Considering this problem, we attempt to make a gym environment from stored data to simulate the water heater environment. Then apply different Reinforcement Learning algorithms to control the electric water heater switch based on different penalties and see how well these algorithms can work.

1.3 Research Questions

Following are the questions for the undertaken research on Investigation and study reinforcement learning for controlling heat water:

1) Is it possible to simulate a water heater using stored data? If so, what are the challenges in this regard, and which approaches generate a better engine to be used in a gym environment?

2) Is it possible to control the electric water heater using RL algorithms?

3) Can Reinforcement Learning optimize the power usage of a water heater?

1.4 Use cases

The related cases from the relevant and highly credible research on similar topics to control electric water using reinforcement learning would be critically analyzed. So, the results of the research can meet the compliance of the given research questions in the best possible way.

1.5 Challenges

The first challenge in this task was stored data. It is discussed with more details in the upcoming sections that what solution has been taken to overcome missing entry point in a day or uncertainty of data during usage of heat water to generate a well perform heat water simulator engine.
Additionally, some techniques have been implemented to find the best water temperature engine to be used in the gym environment.

Moreover, as discussed in [3] Reinforcement Learning techniques do not need expert knowledge in contrast with MPC. They can control policy by interacting with the environment. However, state-space can significantly impact the learning rate of the Reinforcement Learning algorithm. As discussed later, another challenge would be the process of fitting the Reinforcement Learning model due to the big state space of the considering problem.

1.6 Outline

Chapter 2 - Background: The second chapter describes the relevant theory behind the methods implemented in this thesis. In the second part of this chapter, the proposed solution has been provided.

Chapter 3 - Solution Approach: This chapter discusses related works.

Chapter 4 - Implementation: This chapter shows what has been done in this thesis. It starts by discussing the data and temperature predictors engine and then discusses different RL approaches to make a controller for EWH.

Chapter 5 - Results: The results of QL models implemented in Chapter 4 are shown in this chapter.

Chapter 6 - Discussion and Conclusion: This chapter discusses the results shown in Chapter 5.

Chapter 2

Background

As the name suggests, this chapter aims to provide the theory behind the methods and algorithms implemented in this thesis. Due to the use of a Machine Learning Model, terms such as Machine Learning, Reinforcement Learning, and Q-Learning are discussed, as well as more advanced terms such as DQN and Double DQN.

2.1 Technical and Theoretical Background

As mentioned above, this section aims to provide a summary of the technical and/or theoretical knowledge that is required to understand the work that has been done in this thesis.

2.1.1 Linear Regression

The linear regression is a statistical model that can be used to study the relationship between the input and output variables. It combines a set of values with a solution that predicts the output of that set of values. The two sets of values are respectively the input values and the output value. [4]:

$$y = B0 + B1*x$$

Figure 1: The relationship between input and output of Linear Regression

When we have more than one input, the line is referred to as a hyper-plane or plane. This is the representation of the equation that explains the relationship between higher dimensions values [4]:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

Figure 2: The relationship between input and output of Linear Regression in higher dimensions

2.1.2 Long Short-Term Memory (LSTM)

LSTM is a type of neural network that can handle long-term dependencies. It can also perform well when dealing with the vanishing gradient problem in RNN. This type of network is commonly referred to as RNN for persistent memory When processing an input, the user typically stores the previous

information and uses it to perform the next step. However, due to the vanishing gradient, they can't remember the long-term dependencies that were previously stored. LSTM cells are designed to prevent these problems [5]:

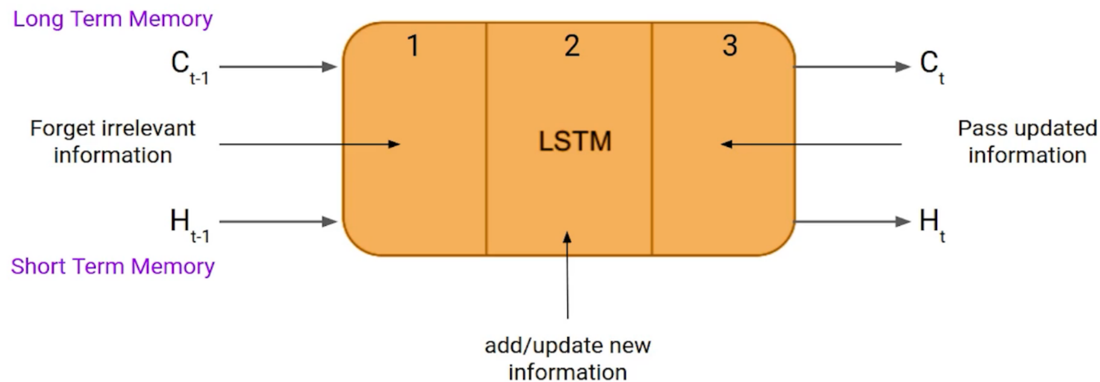

Figure 3: Structure of a LSTM cell

The first gate of an LSTM cell decides if the previous information is relevant or should be forgotten. The second gate then learns the new information and the third gate updates the data to the next time. This type of network also has two hidden states, Long-term Memory and Short-term Memory. Long-term Memory refers to the current time and is known as CT-1, while Short-term Memory is referred to as H. [5].

2.1.3 Gym Environment

Gym is an open-source Python library for developing and comparing reinforcement learning algorithms. It provides a standard API to communicate between learning algorithms and environments. The agent performs some actions in the environment usually by passing some control inputs to the environment and observes how the environment's state changes. One such action-observation exchange is referred to as a *timestep.* To do so, one has to make a class based on the abstraction class of the gym environment and makes the functionality for Reset, Step, and Render functions as well as provide an engine to run in the background of the gym with state and action. The states of an LSTM cell are also carried out along with the current and past timestamps [6].

2.2.1 Machine Learning

Machine learning is a subset of artificial intelligence that focuses on developing systems that can automatically learn from experience. This technology can be used to develop systems that can perform various tasks such as analyzing and predicting the future. Experience can be any type of

data that's inputted into a machine, such as images, videos, and text. Machine learning is a type of AI that aims to teach computers how to think like humans [8].

2.2.2 Reinforcement Learning

One of the main areas of machine learning that focuses on developing systems that can perform various tasks is reinforcement learning. This type of learning is designed to help agents maximize their performance in an environment. It is one of the three main methods that are used in machine learning [9].

Unlike supervised learning, reinforcement learning doesn't require a labeled input/output pair to be presented. Instead, it focuses on finding a balance between exploitation and exploration. This type of learning is designed to help agents perform at their best [9].

2.2.3 Q-Learning

An off-policy method known as q-learning is a type of reinforcement learning that focuses on finding the best action to take based on the current state. It's considered off-policy because it doesn't require a policy to implement. This method aims to learn a policy that maximizes its total reward [10].

The 'q' in q-learning stands for quality. Quality represents how performed action was good to gain some future rewards.

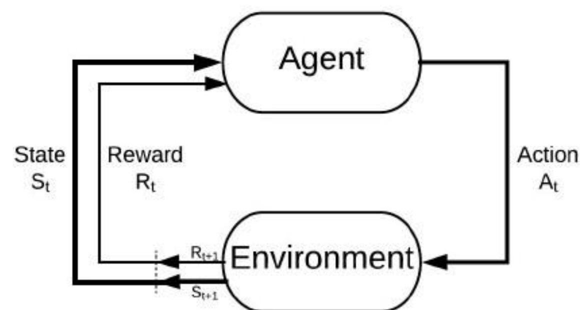

Figure 4: Basic components of Q Learning

The environment reacts to the actions of an agent by generating new observations O(t), rewards R(t). The subsequent action is then determined by history, which is a sequence of actions, observations, and reward signals that happened at the time of the incident.

The states of an environment are the factors that determine what will happen next. There are three types of states that are used to determine what actions will be taken next: the environment state, the agent state, and the information state. The reward function R takes into account the time steps and calculates the potential rewards that can be obtained through the game.

$$\sum_{t=0}^{t=\infty} \gamma^t r(x(t), a(t))$$

Figure 5: Cost/reward function of Q Learning

x represents the state at any given time t, and r represents the reward function for x [11].

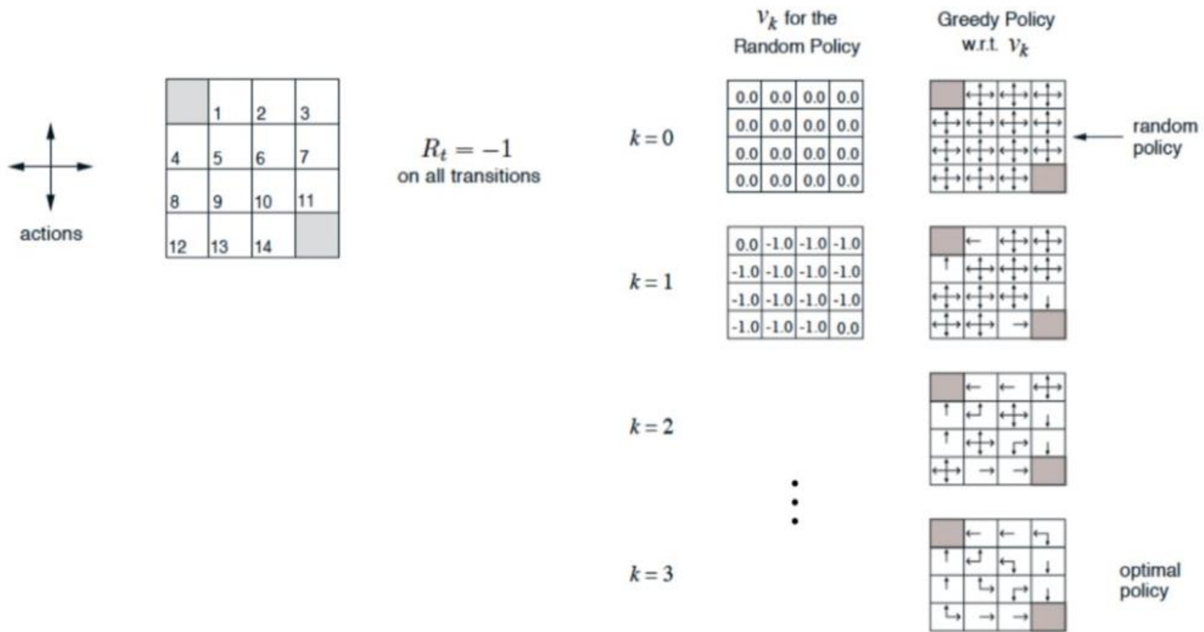

Figure 6: policy iteration of Q Learning

A good policy-iteration method is able to re-define policies after each step and then compute the value of the new policy until it reaches its convergence value. This method can be used to develop systems that require fewer iterations.

The implementation of q-learning involves analyzing and calculating a quantization and approximation functions. This method is commonly used to store information in tables. However, it can be hard to implement due to the increasing number of states and actions involved [11].

With the addition of work approximation, q-learning can be used to perform calculations on larger issues. It can also be used to develop systems that are designed to analyze and solve complex problems. The learning rate, the discount factor, and the initial conditions are some of the factors that influence the performance of q-learning. In most cases, a fast learning rate is ideal. However, if the issue is stochastic, the algorithm can combine under some technical situation to expect the issue to get worse. The Q-function uses the Bellman equation and takes two inputs: state (s) and action (a) [11].

$$Q^{\pi}(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t, a_t]$$

Q-Values for the state given a particular state

Expected discounted cumulative reward

Given the state and action

Figure 7: Bellman Equation in Q Learning [13]

2.2.4 Deep Q-Learning (DQN)

DQN or Deep reinforcement learning is a subfield of machine learning known as deep reinforcement learning. This method combines the capabilities of deep learning and reinforcement learning to provide agents with a more accurate and timely decision-making process. Deep learning can be used to perform decisions from unstructured data [12]. The main idea of Q-learning is it can provide an agent with a function of $Q{:}State \times Action \rightarrow \mathbb{R}$, that can tell the agent what actions will lead to reward. If the value of Q was known, it can then be used to create a policy that maximizes the rewards:

$$\pi(s) = \text{argmax}a\ Q(s,a)$$

Figure 8: The policy of maximum Q value

Unfortunately, in the real world, we don’t have access to full information. This means than we need to come up with a method that can approximate the value of Q. One of the most common approaches to do this is by creating a table that updates the values of Q after each action. However, this method is very slow and doesn't scale to large state and action spaces. [13].

2.2.5 Deep Reinforcement Learning with Replay Memory

One of the most effective ways to do this is by storing the experiences of an agent in a dataset known as Experience Replay. This method can be used to retrieve the data at each time t as e_t:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

Figure 9: Content of a Replay Memory entry

This tuple gives a summary of the agent's *experience* at time t including environment S_t, the action taken from state a_t, the reward r_{t+1} given to the agent at time t+1 as a result of the previous state-action pair (s_t, a_t), and the next state of the environment (s_{t+1}) [14]. In order to prevent the data from being discarded, replay memory can be used to store the previous experiences. When a batch of

experiences is collected from memory, they are then used to train the neural network. The size of the replay determines the number of experiences that the network uses in the update process [13].

2.2.6 Double DQN

Deep Q Learning tends to overestimate the reward, and ends up to low quality policy and unstable training. Let's consider the equation for the Q value:

$$\text{New } Q(s,a) = Q(s,a) + \alpha\,[R(s,a) + \gamma\,\max Q'(s',a') - Q(s,a)]$$

New Q Value for that state and the action
Learning Rate
Reward for taking that action at that state
Current Q Values
Maximum expected future reward given the new state (s') and all possible actions at that new state.
Discount Rate

Figure 10: The equation of the Q value

The last part of the equation takes into account the estimate of the maximum value. This method leads to a systematic overestimation, which can introduce a bias in the training process. To avoid this issue, a new target network can be created. The values from the new network will be taken from the main DQN, which is the representation of the state of the network. However, since it only updates after a certain number of episodes, it doesn't have the same weights. However, adding a target network can slow down the training process because the target network does not update continuously [13].

Chapter 3

Solution Approach

This chapter aims to provide an overview of the previous related works to this project, as well as the main approach that was used in the study.

3.1 Reinforcement Learning in Electric water heater (EWH)

Machine learning has been used in various applications to reduce the energy costs of electric water heaters. One of the main methods that was used in this project was reinforcement learning. This method allows the network to learn and perform on its own.

3.1.1 Reinforcement Learning Applied to an Electric Water Heater: From Theory to Practice

In this paper, the researchers presented a method that was able to store energy without affecting the end-user's comfort. They formulated a sequential decision-making process that was based on a method known as a Markov decision process. They also used an automatic-encoder network to find compact feature representations of the sensor data. They found a control policy that was suitable for an electric water heater by fitting Q-iteration. In a simulation-based test, the proposed method performed better than the full state method. The researchers found that the proposed method was able to find a policy that was more effective than the full state method. They found that the implementation of the proposed method was able to reduce the energy consumption of an electric water heater by 34% and 24%, respectively, by using imbalance prices and day-ahead prices. In a laboratory experiment, they applied the fitted Q-iteration method to an electric water heater. The researchers found that the state vector of the fitted Q-iteration method did not improve its performance. Compared to the control policy, the fitted Q-iteration method was able to reduce the energy consumption by 15% in just 40 days. The researchers concluded that learning in a compact feature space improves the quality of the control policy. They also recommend using the full state vector when there are only a few temperature sensors available. In addition, they found that switching to higher state-space representation when the number of observations increases can improve the performance of the fitted Q-iterating method. They additionally found that the implementation of the method was able to achieve good control policies in just 20 days [3].

3.1.2 Deep Reinforcement Learning for Autonomous Water Heater Control

The paper presented a method that was able to reduce the electricity costs of an electric water heater by implementing a TOU pricing policy. The researchers trained a set of RL agents using the DQN algorithm. They tested their performance against a pair of hot water usage profiles and an unseen pair of prices for 30 days. The results of the study validated the performance of the RL agents against the different control policies. The main advantage of this method was that it was able to perform better than the rule-based controllers when it came to controlling the energy consumption of an electric water heater. Due to the optimization-based controller's longer simulation period, it is more challenging to perform effective optimization. The paper also noted that the proposed method could help reduce the electricity costs of an electric water heater without any prior knowledge about its operation. The results of the study revealed that the RL agents performed better than the rule-based controllers when it came to controlling the energy consumption of an electric water heater. They were also able to save up to 35% of the electricity cost. In addition, the agents were able to achieve a close performance to the optimization-based controllers when it came to analyzing future hot water usage volumes. Although the proposed method was validated on a water heater simulation, the results of the study are still based on a simulation. Another limitation of the study is that it assumes that the future hot water usage volumes will be known. This means that the results of the study are still based on a simulation [15].

3.2 Proposed Solution

With the help of Linear regression and LSTM methods, two predictor models for the gym environment have been made based on the three different datasets reproduced from the raw data. Predictors models have been tested for an acceleration/deceleration speed that has been implemented to check whether they can make a convincing result for the chaotic q-learning fitting process. In the end, three LR methods -q-learning, DQN with Replay Memory, and Double DQN with Replay Memory- were fitted to control EWH simulator efficiently.

Chapter 4

Implementation

The goal of this paper is to find an optimal control policy to reduce power usage on an electric water heater using different machine learning algorithms and find differences among them. In the first section, given data from Nova Smart company has been cleaned up. In the second section, we create different EWH temperature simulator engines using Linear Regression and LSTM to be used in the Gym environment. And in the third section, three different temperature engines will be applied to RL algorithms to manage EWH control.

4.1 Preprocessing data

Nova Smart is a newly instituted company. Their goal is to build smart houses by installing sensors on a different part of a house and reducing power usage. They discovered that an electric water heater is among the most electric consumption devices in a house by using about 15% of energy in a day. Thus, reducing power usage of the electric water heater would be a big step for their start-up.

Nova Smart uses cloud storage to save sensor data every minute. Each entry in the storage includes space that determines the sensor, Unix base timestamp, and value columns. So far, they have smarted 15 houses already and have all data in the same storage. Following API can be used to download data from the start point upward now. It is possible to apply a couple of filters like looking for data of a particular house e.g. Alex house and type of timestamp e.g. Unix standard. An overview of the data looks as follows:

http://ymawyymlkdwzkkjj.hostedmetrics.com:20661/api/v1/export/csv?start=2022-01-01T00:00:00Z&format=__name__,__timestamp__:unix_ms,__value__&match[]={__name__=~"alex.*"}

```
heater_temp.shape, heater_switch.shape
((114119, 3), (114120, 3))
```

	space	timestamp	value
0	RPT1106_Bath_2nd_window	1647996221095	0.0
1	RPT1106_Bath_2nd_window	1647996281095	0.0
2	RPT1106_Bath_2nd_window	1647996341095	0.0
3	RPT1106_Bath_2nd_window	1647996401095	0.0
4	RPT1106_Bath_2nd_window	1647996461095	0.0

Figure 11: The shape of temperature and power switch

4.1.1 All data dataset

There are two spaces in which this paper is interested, EWH temperature (RPT1106_heater_temp) and EWH switch (RPT1106_heater_switch). Looking at the image above it seems that heater temperature has a missing entry, therefore both temperature and switch datasets must be aligned based on the timestamp and an additional row in the heater dataset must be removed. At the same time, the timestamp will be converted to time and both datasets are merged into one table called "dataset of all data". Further, we will have more datasets. Additionally, we are not interested in the date part on timestamp after conversion as it is just the time value that would be useful during the implementation of EWH gym simulator. Finally, we checked datasets for Nans or any missing value and made sure that there are no rows with the same timestamps. The head of the final dataset is shown in the following image:

	temperature	power	counter	time
timestamp				
1647996161095	60.6	0.0	0.0	014200
1647996221095	60.6	0.0	1.0	014300
1647996281095	60.6	0.0	2.0	014400

Figure 12: The head of "all data" dataset

The distribution of temperature and power are shown in the following plots. There are 114,119 rows. Based on the power distribution plot, we assume any power value below 200 as power switch Off and any power value greater than 1600 as power switch if ON. Based on temperature distribution, it seems that most of the time, the temperature value is between 60 to 72 apart from times when heated water is in use. Plus, temperature has never been below 28 degrees or higher than 72 degrees.

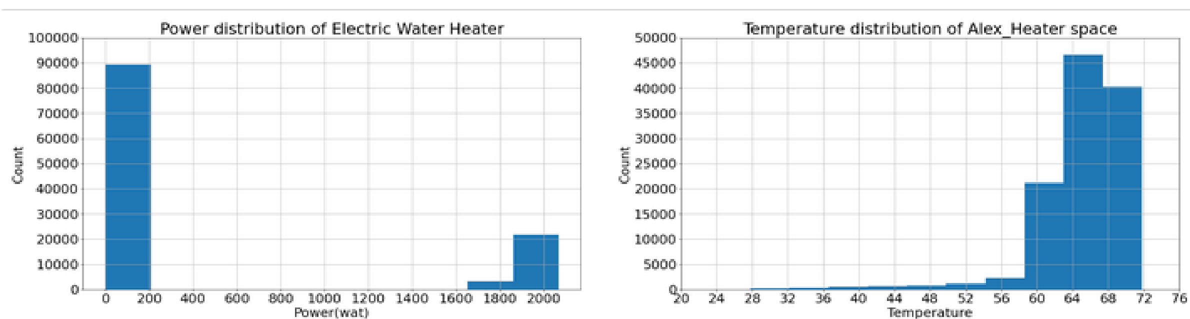

Figure 13: Overview of distribution of power and temperature in raw data

4.1.2 1440 data frame:

Later, we found that even though there were no empty cells (Nan) in the data there are a lot of missing data rows during the day. It is expected to be (60 * 24 =) 1440 rows per day, however, there are just 25 days that fulfill this criterion and have a complete data entry for a day. Thus, those days have been taken out using the following snippet code to make the second dataset called “1440 dataset”.

At first, indexes of beginning time (00:00:00) and end time (23:59:00) of each day have been saved into a list. Then days that fulfill the criteria of having 40 rows more/less than 1440 split and merged together to make the second data frame called “1440 data”.

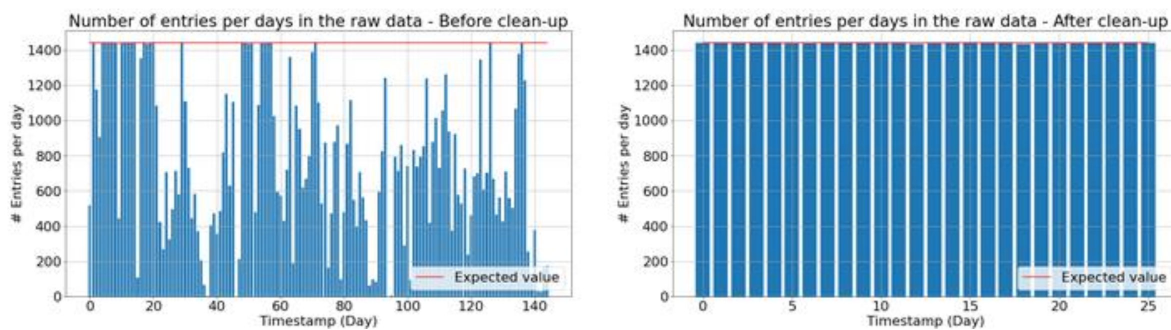

Figure 14: Taking out Days with missing rows

```
# remove days that has less than 1440 entries.
temp_data = data[814:]
lstcount = []
timer = 0
currtime = -1
for index, row in temp_data.iterrows():
    if(float(row.time) > currtime):
        currtime = float(row.time)
        timer+=1
    else:
        currtime = float(row.time)
#         print(currtime)
        lstcount.append(timer)
        timer=1

newdata = temp_data[:0]
lowerbound = 0
for item in lstcount:
    if(abs(1440 - item) < 40):
        newdata = newdata.append(temp_data[lowerbound:lowerbound+item])
    lowerbound += item
```

4.1.3 No water usage:

The temperature of heat water up and down between 60 to 70. However, at times when heated water is in use, it will drop dramatically to a very low degree even though the power switch turns on immediately after temperature is around 60. This behavior can confuse prediction models in two ways. First, when power is on, the predictor model expects an increase in temperature which it does not. Second, it would be difficult for the predictor model to predict deceleration speed of water temperature compare to when the power switch is off. Therefore, following snippet code has been written to find and remove such data points and create the third data frame called “data with no water usage”. The former data frame will not have a “time” column, because tracking time and predicting the next temperature would be confusing after merging sliced data.

```
for index, row in data.iterrows():
    expectedTemp = data[data.index== index-2]
    expectedTemp = expectedTemp.temperature.values[0] if(len(expectedTemp.temperature.values))>0 else 0
    if(expectedTemp - row.temperature > 0.65 ):
        data.loc[index-2, 'water_tap'] = 1
        data.loc[index-1, 'water_tap'] = 1
        data.loc[index, 'water_tap'] = 1
    else:
        data.loc[index, 'water_tap'] = 0
```

```
increasing = True
usedheatwater = False
dct = {}
peakindex = 0
peak = 70

for index, row in data.iterrows():
    if(data.loc[index, 'temperature'] > peak):
        peak = data.loc[index, 'temperature']
        peakindex = index
        usedheatwater = False
        continue
    elif(data.loc[index, 'temperature'] < 70):
        peak = 70
        if(dct.get(peakindex, None) == None):
            dct[peakindex] = False

        if(data.loc[index, 'water_tap'] == 1 and dct[peakindex] == False):
            dct[peakindex] = True

tempdata = data[:0]
lastindex = -1
jump = False

for k, v in dct.items():
    if(jump == True):
        lastindex= k
        jump = v
    else:
        tempdata = tempdata.append(data[lastindex:k])
        lastindex = k
        jump = v
```

When the power switch is off, the temperature will drop slowly every minute. It is found that if the temperature drops more than 0.65 degrees in 2 minutes, it is unusual and assumes that heated water

is in use. Therefore, a new column “water_tap” was added to the dataset to store whether heated water is withdrawing (water_tap=1). Then, starting from peak temperature (70) start index and end index of windows (every time temperature drops from 70 to 60 or lower) having no water usage stored in the dictionary. Finally, windows with unused water were sliced and merged. The following plots show the result of taking data with water_tap=1 in one day.

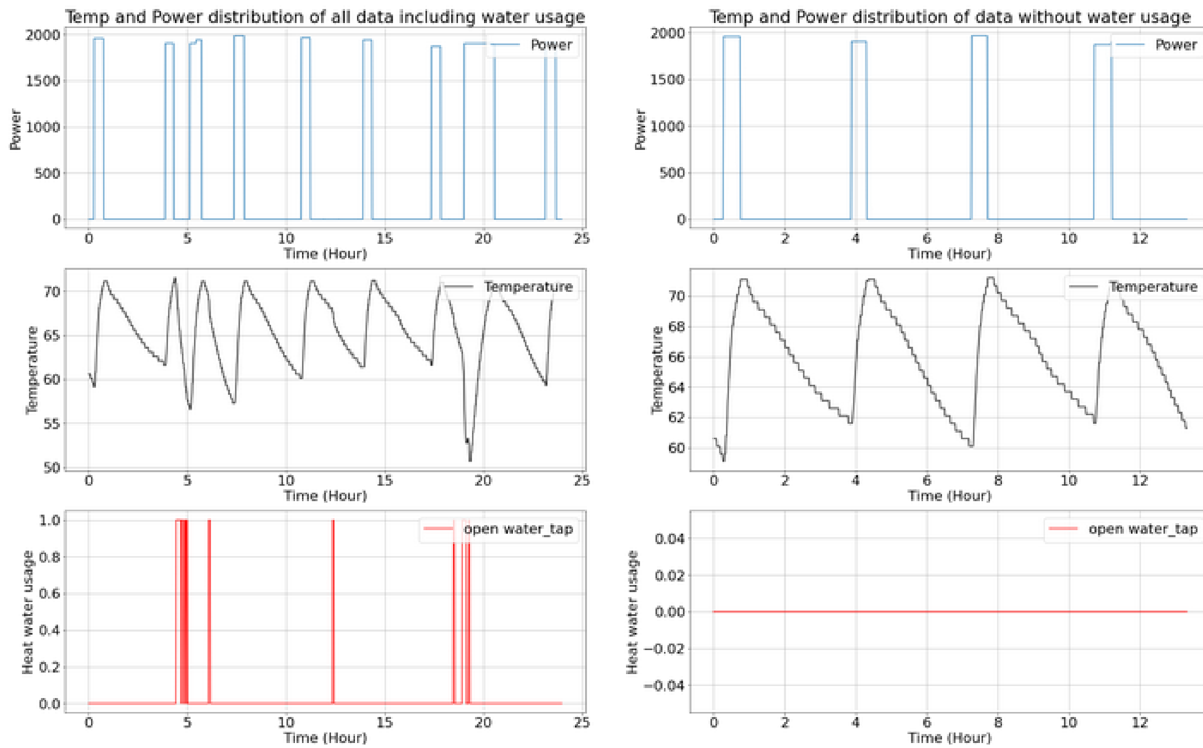

Figure 15: Taking out rows of used water in data

4.2 Daily power price:

The following plot shows the result of querying power prices on 21 May 2022 from Nord Pool [16] APIs. Power prices can go up to 300 or more during rush hour. Power price can affect returned rewards from the gym environment. It will be discussed later.

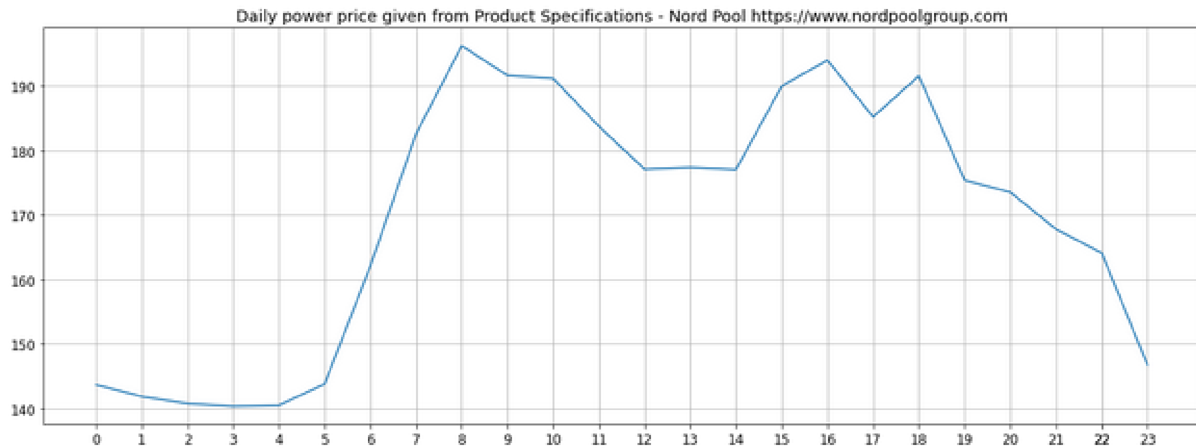

Figure 16: Power price on 21 May 2022

4.3 Temperature predictor engine

The gym environment needs an engine to perform an action on it which is a water temperature predictor in this case. Linear Regression and LSTM are two well-known machine learning algorithms that can make regression models. Other machine learning algorithms like PSINDY have also been used but the result was not convincing. All three datasets were applied on LR and LSTM to generate six predictor models. In the following section, we will explain how these two engines have been implemented.

4.3.1 Linear Regression:

Scikit-learn is a well-known framework for machine learning algorithms. Time, Power, and temperature are chosen columns for all datasets except “1440 dataset” as it does not have the time column. Data is divided into 80% and 20% for training and test part. MinMaxScaler framework will adjust all values in the dataset between -1 to +1 and one more dimension added to the train and test parts to fix dimensionality issues. Linear Regression model fitted with all datasets to show how differently they will behave. The below snippet code shows how this has been done followed by result plots of each dataset.

```
train = data[:int(0.8*(len(data)))]
test = data[int(0.2*(len(data))):]

train = train[['time', 'power','temperature']].to_numpy()
test = test[['time', 'power','temperature']].to_numpy()

# Scale features
s1 = MinMaxScaler(feature_range=(-1,1))
Xs = s1.fit_transform(train)
pickle.dump(s1, open('linearregression_MinMaxScaler.s1', 'wb'))

# # Scale predicted value
s2 = MinMaxScaler(feature_range=(-1,1))
Ys = s2.fit_transform(train[:,[2]])
pickle.dump(s2, open('linearregression_MinMaxScaler.s2', 'wb'))

# Each time step uses last 'window' to predict the next change
window = 1
X = []
Y = []
for i in range(window,len(Xs)):
    X.append(Xs[i-window:i,:])
    Y.append(Ys[i])

# Reshape data to format accepted by LSTM
X, Y = np.array(X), np.array(Y)

model = LinearRegression().fit(X[:, 0], Y[:, 0])
```

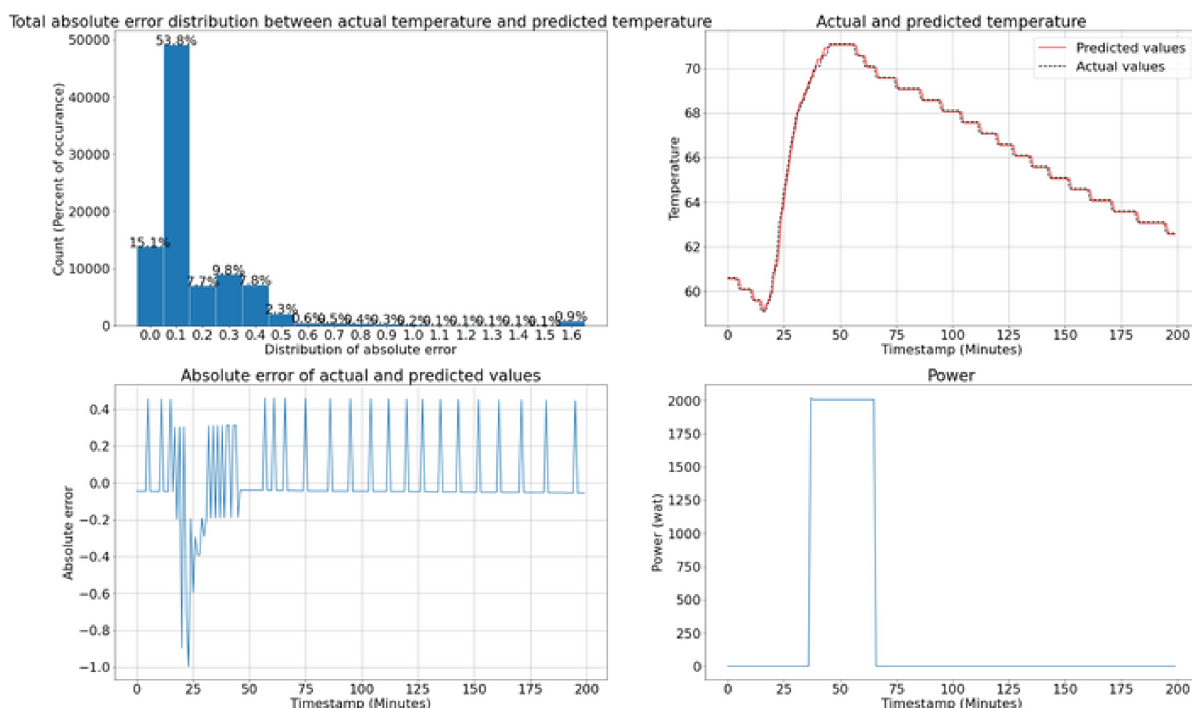

Figure 17: Linear Regression of "all data" dataset

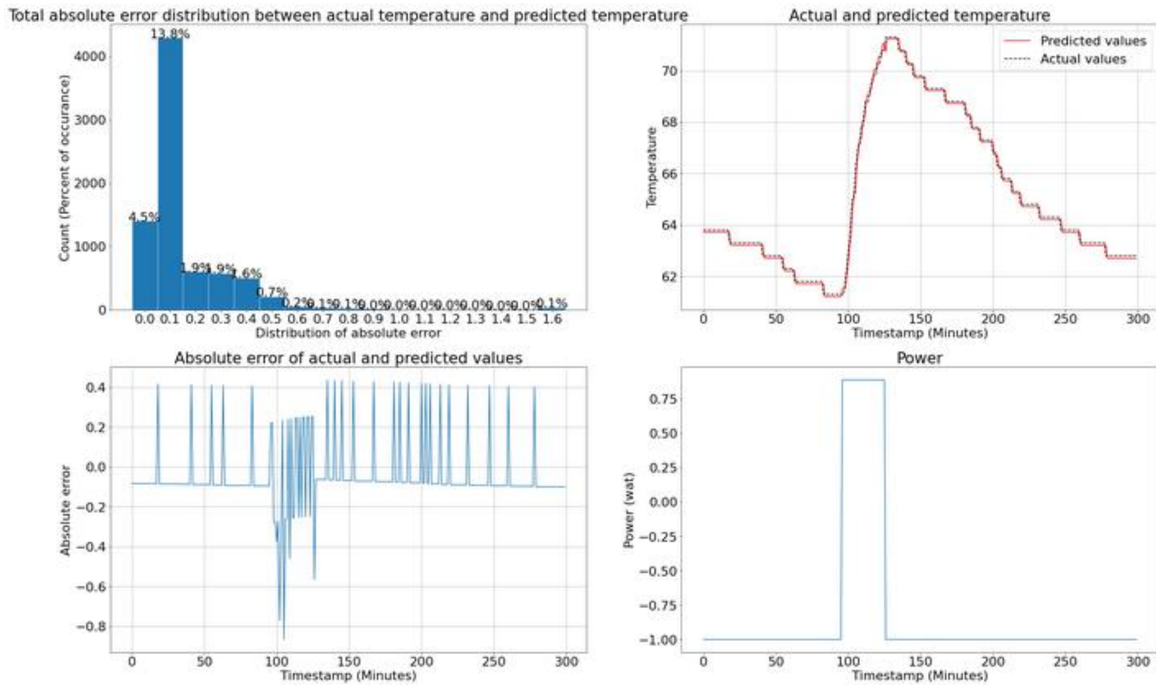

Figure 18: Linear Regression of "1440" dataset

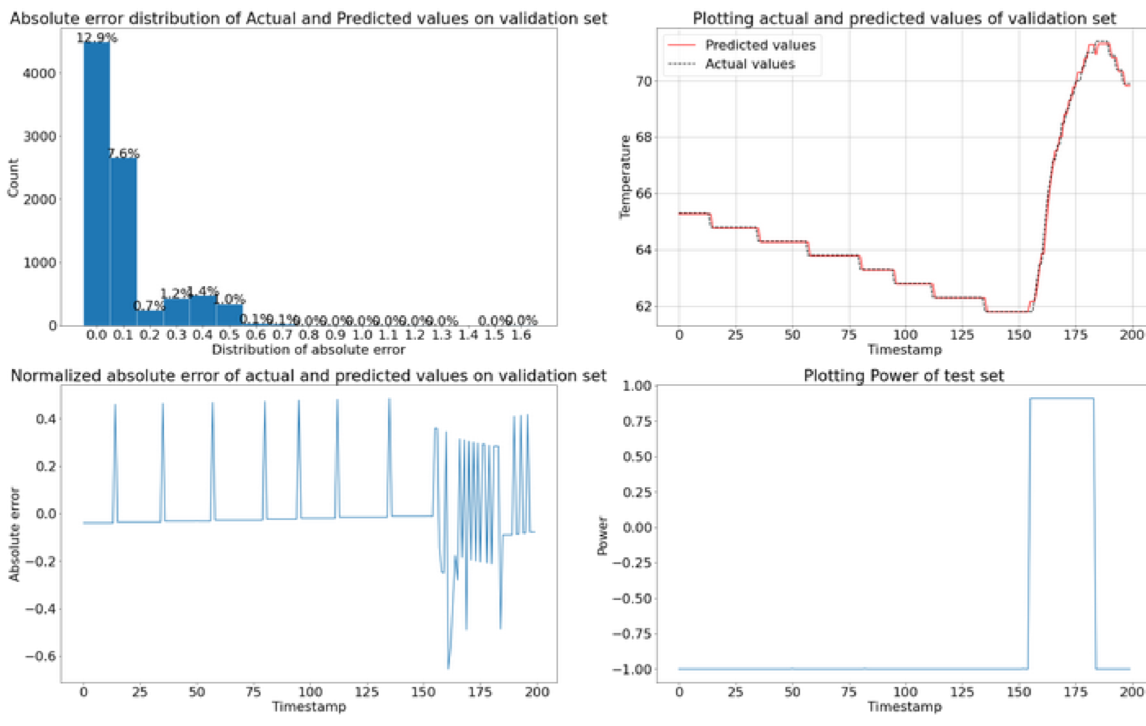

Figure 19: Linear Regression of "no water usage" dataset

4.3.2 Linear Regression fitting issues:

Looking at the following plots, it is shown how linear regression learned patterns of the data. To do that, we applied two tests. One, starting at 80-degree temperature and apply power switch OFF for 12

hours (720 iterations, one per minute), and second, starting at 50-degree temperature and apply power switch On for 12 hours. This is to check how fast temperature will accelerate and decelerate on each model. The reason for this test is that in real-life, temperature is always between certain low and high degrees, in this case 60 to 72, apart from when the heated water is in use and temperature drops very quickly. The fact is, fitted linear regression models for each dataset (and LSTM models as we will discuss later) have almost survived the validation and test part and they seem to have a decent result over test part. However, since q-learning tends to learn the environment by applying unpredictable actions, at some points they may apply a power switch on (Action=1) even though the current temperature is higher than 70 or 75, or on the other hand, they may try power switch Off (Action=0) even though the current temperature is below 60 or 55, thus this test can help to show if these models can go beyond data patterns. It seems that “no water usage” dataset cannot drop the temperature below 60 as all given data in the LR model were around 60 to 70. But the rest datasets seem to be fine even though the acceleration and deceleration speed may vary.

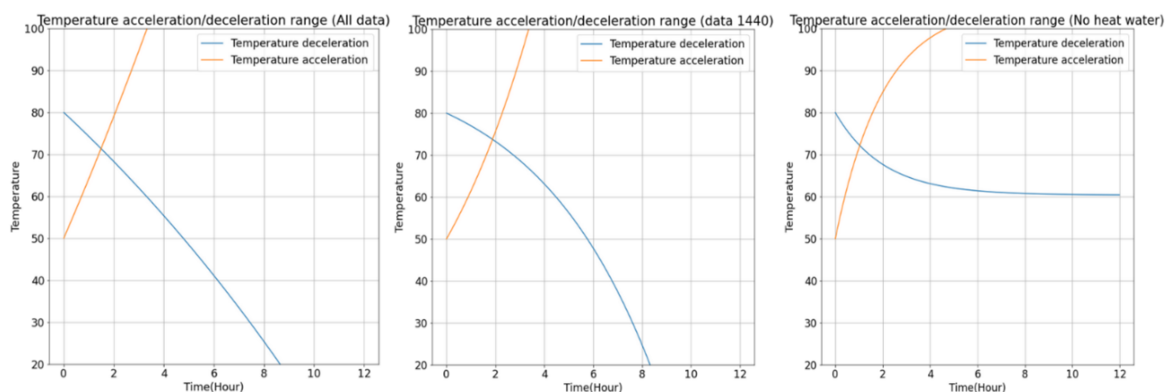

Figure 20: Linear Regression acceleration/deceleration speed

4.3.3 LSTM:

Our datasets are among sequence data as the next temperature depends on the current temperature and power switch value. LSTM is a recurrent neural network that can process an entire sequence of data [16] therefore it is a good choice for such a dataset. The following code snippet will show how LSTM has been implemented. Scikit-Learn framework has LSTM functionality as well. Like Linear Regression, 80% of data will be used to train and 20% to test the fitted NN. MinMaxScaler will adjust values between -1 to +1 and we have windows of 1. Other values for window have been tested but this seems to be the best setting. 70 LSTM units lined up in two LSTM layers and one Dense layer for neural network output and there are no dropouts. Dropouts tends to exclude activations and weights updates to reduce overfitting but, in our scenario, the best was to pass everything to the next unit [17]. Too many or too few epochs can lead to overfitting and underfitting the neural network.

Therefore, EarlyStopping allows setting a large number of epochs while stops the training where there was no performance improving [18].

```
# Initialize LSTM model
model = Sequential()

model.add(LSTM(units=70, return_sequences=True, input_shape=(X.shape[1],X.shape[2])))
model.add(Dropout(0.0))
# model.add(LSTM(units=75, return_sequences=True))
# model.add(Dropout(0.0))
model.add(LSTM(units=70))
model.add(Dropout(0))
model.add(Dense(units=1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error',
              metrics = ['accuracy', 'MeanSquaredError'])
```

```
# Allow for early exit
es = EarlyStopping(monitor='loss',mode='min',verbose=1,patience=10)
```

```
# Fit (and time) LSTM model
t0 = time.time()
history = model.fit(X, Y, epochs = 10, batch_size = 50, callbacks=[es], verbose=1)
t1 = time.time()
print('Runtime: %.2f s' %(t1-t0))
```

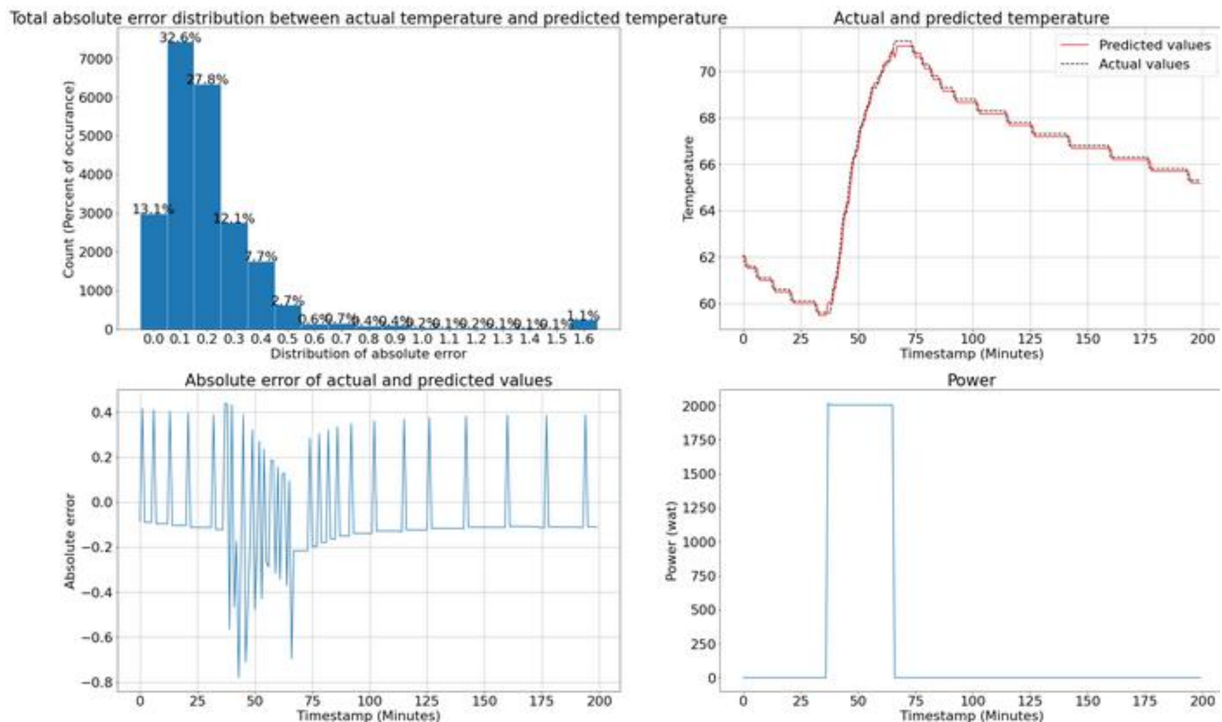

Figure 21: LSTM on “all data” dataset

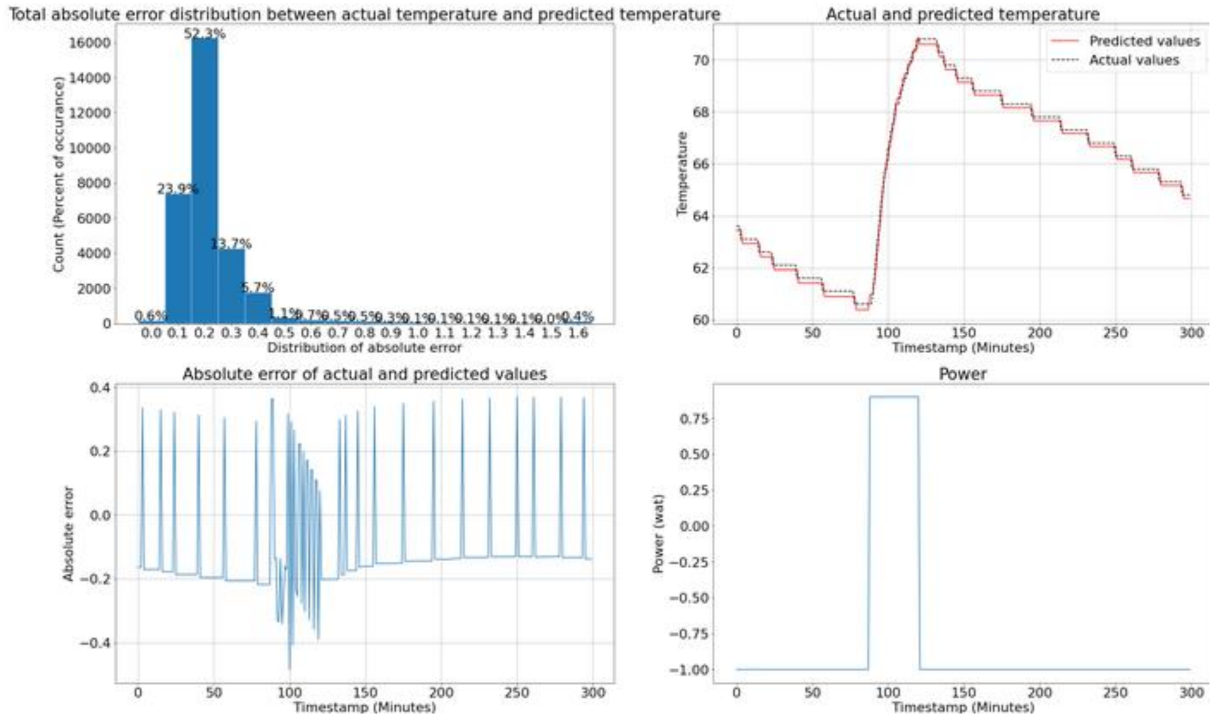

Figure 22: LSTM on "1440" dataset

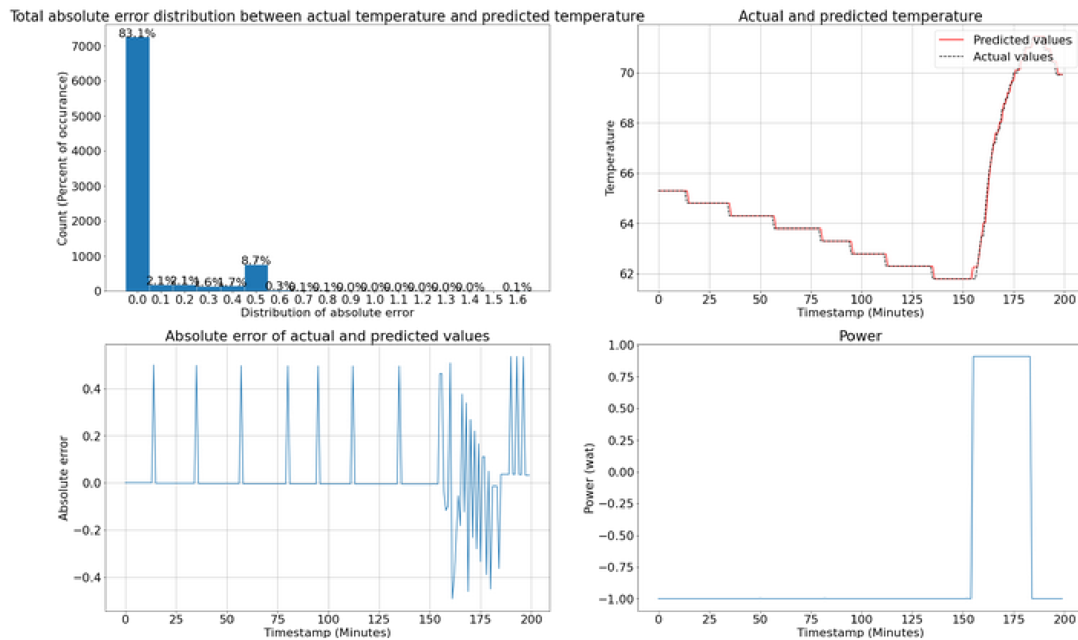

Figure 23: LSTM on "no water usage" dataset

4.3.4 LSTM fitting issues:

As discussed before, the best practice is to test temperature predictor models as q-learning will try different scenarios in the environment. The same test was conducted on the LSTM models and the result are shown in the following plots. It seems that LSTM models result poorly compare to LR. They decrease the temperature until 60-degree as expected and then remain at 60-degree for a while and

then drop again. This is due to the fact that most of the time, water temperature is going up and down around 60 to 70, and then when heated water withdraws, drop to a temperature less than 60. Additionally, temperature will not go beyond 30 and 72 because those are minimum and maximum temperatures values in the whole dataset. LSTM learned data patterns as expected, but q-learning algorithms will try different actions no matter what the temperature is, and it is important to have a predictor engine that can behave beyond data limits.

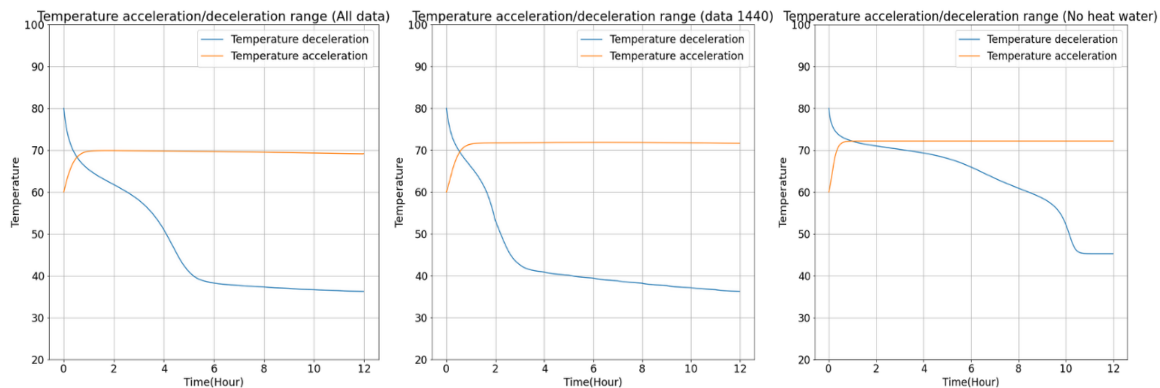

Figure 24: LSTM acceleration/deceleration speed

4.4 Open AI Gym environment

There are a lot of environments to practice RL algorithms in the Open-AI framework. There is also an option to make a customized environment based on the Open AI framework. To do that, a python class should inherit the gym environment abstract class and implement the functionalities of a gym. Gym functionalities include “actions”, “state”, “reset function”, “step function”, “done function”, “render function”, and “reward”. “Penalty” is another member included in the gym which will be explained.

- Actions, defines what can be performed on the environment which is turning On/Off the power switch.
- State, is a tuple of the following parameters (current time, current temperature, ideal EWH temperature, current penalty) that represent the current state of the EWH. The ideal temperature is the preferred temperature of EWH which is 70.
- Reset, defines the initial state of the environment. The current time will be set to “00:00:00”, a random number between 60 to 75 will initiate the current temperature and “isTraining” parameter will be set. The former variable defines whether we are fitting the RL algorithm or simulating the EWH.

- Done, basically is the final state. Some environments like CartPole [cartpole] are episodic. They finish when the player loses the game. EWH environment is not episodic. It starts at time “00:00:00” and finishes at night at “23:59:00”.
- Step, performs an action on the gym having the current state. It returns the answer to the question of what would be the next state if chosen action applied to the gym. However, there is a difference in step functionality whether the gym is in the fitting mode (isTraining=True) or simulation mode (isTraining=False). In simulation mode gym simply performs selected action on the current state and returns the result, but in training mode, the same action will be applied to the gym 10 times. It looks like taking 10 steps simultaneously and returning the final result. The reason is changing temperature is too slow. Setting the power switch to On or Off for just one minute does not affect the temperature too much. As an example, when the temperature is dropping, and the power switch is turned on (action = 1) it will take one or two minutes for the EWH element to warm up and start increasing temperature in the water tank. Therefore, even though the power switch is turned On, the water temperature is still decreasing in the first couple of minutes, and this will confuse RL models. Moreover, this iteration does not affect EWH behavior other than emphasizing the result of taken action. Another implemented feature in the Step function is avoiding the temperature to pass defined “room temperature” and “high temperature”. It is obvious that EWH never drops below room temperature and also cannot pass boiling temperature. This feature also does not affect EWH model behavior since at any given point, taking the wrong action will return a negative reward, and taking the right action will return a positive reward.
- Penalty, is the sum of defined parameters using “x_parameter” function and it will be deduced from the reward. Time, price, and comfort are three penalties than can affect reward function at any point. Plus, they can be changed at any time during q-learning fitting.
    - Time: each day has divided into 4 parts. 1) “00:00:00” - “05:30:00”, 2) “05:30:00” - “09:00:00”, 3) “09:00:00” - “18:00:00”, 4) “18:00:00” - “23:59:00”. Based on NovaSmart experience, heated water would not withdraw at 1st and 3rd part of a day. Thus, one option is to not warm up the water during those hours. It is possible to apply none or 45 penalties at 1st and 3rd part of a day.
    - Power price: Power price can affect reward at any time of the day. Power price value is divided into 3 parts. Below 100, between 100-200 and higher than 200 which can apply 0, 5, 10 penalties on the reward respectively.
    - Comfort: It is possible to have none or 5 penalties on the reward.
- Reward, is the result of the taken action. It defines how good or bad was the action based on the state, taken action, and penalties. If taken action led water temperature to get closer to the ideal temperature, then reward would be the positive value of the difference between ideal temperature and new temperature, otherwise, reward would be the negative difference between ideal temperature and new temperature.

```python
"""Custom Environment that follows gym interface"""
class CustomEnv(gym.Env):
    def __init__(self):
        super(CustomEnv, self).__init__()

        self.currentTime = '000030'
        self.initTemperature = 0
        self.currentTemperature = 0
        self.idealTemperature = 0
        N_DISCRETE_ACTIONS = 2 # 0= Off, 1=On
        self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
        self.currentAction = 0
        self.roomTemp = 29
        self.highTemp = 100
        self.istraining = False

        # load models
        self.LR_model = pickle.load(open('linear_regression_nowaterusage.model', 'rb'))
        self.LR_s1 = pickle.load(open('linearregression_MinMaxScaler_nowaterusage.s1', 'rb'))
        self.LR_s2 = pickle.load(open('linearregression_MinMaxScaler_nowaterusage.s2', 'rb'))

        # penalties
        self.timePenalty = 0
        self.pricePenalty = 0
        self.comfortPenalty = 0

    def step(self, action):
        self.currentAction = action
        power = 1940 if action == 1 else 0.0
        curTemp = self.currentTemperature

        if(self.istraining):
            for i in range(10): # LR
                transform = self.LR_s1.transform([[self.currentTime, power, self.currentTemperature]])
                predictedTemperature = self.LR_model.predict([[transform[0][0], transform[0][1], transform[0][2]]])
                self.currentTemperature = np.round(self.LR_s2.inverse_transform([predictedTemperature])[0][0], 2)
        else: # LR
            transform = self.LR_s1.transform([[ power, self.currentTemperature]]) #self.currentTime
            predictedTemperature = self.LR_model.predict([[transform[0][0], transform[0][1] ]]) #, transform[0][2]
            self.currentTemperature = np.round(self.LR_s2.inverse_transform([predictedTemperature])[0][0], 2)

        newTemp = self.currentTemperature
        self.currentTemperature = curTemp
        reward = self.calculateReward(newTemp)
        self.currentTemperature = newTemp

        # we will not drop water temperature below room temperature
        if(self.currentTemperature < self.roomTemp): self.currentTemperature = self.roomTemp
        # also, we will not increase water temp over highTemp = 100
        if(self.currentTemperature > self.highTemp): self.currentTemperature = self.highTemp

        self.currentTime = self.getnexttime()
        return (self.currentTime, np.round(self.currentTemperature, 0), self.idealTemperature,
                self.get_penalties()), reward, self.done(), []
```

```python
# re-initial the state of the environment
def reset(self, istraining):
    self.istraining = istraining
    self.currentTemperature = randint(60, 75)
    self.currentTime = '000030'
    return (self.currentTime, np.round(self.currentTemperature, 0), self.idealTemperature, self.get_penalties())

# adding one minute to the current time
def getnexttime(self):
    newTime = datetime.strptime(str(date.today()) + ' ' + str(self.currentTime), '%Y-%m-%d %H%M%S') + timedelta(minutes=1)
    return newTime.strftime('%H%M%S')

def done(self):
    return True if(int(self.currentTime) > 235900
                   or self.currentTemperature < 5 or self.currentTemperature > 95) else False

#calculate reward based on currentTemp and idealTemp
def calculateReward(self, newTemp):
    idealTemp = self.idealTemperature - self.get_penalties()
    if(newTemp < idealTemp):
        reward = newTemp - self.currentTemperature
        diff = abs(newTemp - idealTemp)
        coefficient = +1 if reward>0 else -1
        reward += coefficient * diff
    else:
        reward = self.currentTemperature - newTemp
        diff = abs(newTemp - idealTemp)
        coefficient = +1 if reward>0 else -1
        reward += coefficient * diff
    return round(reward, 1)

# Render the environment to the screen
def render(self, mode='human', close=False):
    print(self.currentTime, " - ", self.idealTemperature, self.currentTemperature)

# updating penalty parameters value
def X_parameter(self, ideal_temp, time, price, comfort):
    self.idealTemperature = ideal_temp
    self.timePenalty = time
    self.pricePenalty = price
    self.comfortPenalty = comfort

# calculate penalty for the current state
def get_penalties(self):
    if(int(self.currentTime) >= 220000 or int(self.currentTime) < 53000):
        tim = self.timePenalty[0]
    elif(int(self.currentTime) >= 53000 and int(self.currentTime) < 90000):
        tim = self.timePenalty[1]
    elif(int(self.currentTime) >= 90000 and int(self.currentTime) < 180000):
        tim = self.timePenalty[2]
    elif(int(self.currentTime) >= 180000 and int(self.currentTime) < 220000):
        tim = self.timePenalty[3]

    x_parameter = tim + self.pricePenalty + self.comfortPenalty
    return x_parameter
```

4.5 Reinforcement Learning

Three RL algorithms have been implemented in this paper. Q-Learning, DQN with Replay Memory, and Double DQN with Replay Memory.

4.5.1 Q-Learning:

Q-Learning is a machine learning algorithm that is different from supervised or unsupervised learning. It is basically a dynamic programming approach to observing the environment as much as possible and storing state, taken action, and rewards in storage. And then replay stored data on the environment by selecting the highest rewarded action per state. Linear Regression engines performed better than LSTM, therefore they chose to be the temperature predictor engine for the gym environment. Implemented q-learning algorithm starts with a couple of for-loops to training environment with different settings. Each episode begins from “00:00:00” until 23:59:00 and There is no break during each episode. The learning rate is decreasing slightly every 100 rounds. Since storage has not been initialized, a new entry will be created and filled with random numbers for each new state-action observation. Fitting q-learning by having a learning rate of 0.9 and gamma at 0.05 results the best.

```
def Q_learning(q_table, episodes = 2000, gamma = 0.05, lr = 0.9, timestep = 100, epsilon = 0.1):
    rewards = 0
    steps = 0
    env = CustomEnv()
    result = []
    temp_lr = lr

    for ideal_temp in [70]:
        for time in [[0, 0, 0, 0], [45, 0, 45, 0]]:
            for price in [0, 5, 10]:
                for comfort in [0, 5]:
                    env.X_parameter(ideal_temp, time, price, comfort)

                    lr = temp_lr
                    for episode in range(1,episodes+1):
                        if(episode % timestep == 0):
                            lr*= 0.99

                        current_state = env.reset(istraining=True)
                        done = False

                        while not done:
                            if(q_table.get(current_state, None) == None):
                                q_table[current_state] = [np.round(np.random.uniform(0, 1), 2),
                                                          np.round(np.random.uniform(0, 1), 2)]

                            if (np.random.uniform(0,1) < epsilon):
                                action = env.action_space.sample()
                            else:
                                action = np.argmax(q_table[current_state])

                            observation, reward, done, info = env.step(action)
                            next_state = observation

                            if not done:
                                if(q_table.get(next_state, None) == None):
                                    q_table[next_state] = [np.round(np.random.uniform(0, 1), 2),
                                                           np.round(np.random.uniform(0, 1), 2)]

                                max_future_q = np.max(q_table[next_state])
                                current_q = q_table[current_state][action]
                                new_q = (1-lr)*current_q + lr*(reward) + gamma*max_future_q
                                q_table[current_state][action] = round(new_q, 1)
                                current_state = next_state
                            else:
                                q_table[current_state][action] = round(reward, 1)
                                break
```

4.5.2 DQN with Replay Memory:

The difference between q-learning and DQN is that DQN approximates state and action with a neural network instead of a data table to estimate the q-value per action. A neural network with 3 layers has been implemented. The number of nodes in the first layer is the same as observation state. There are 64*2 nodes at the hidden layer and one node per action (power switch Off/On) in the last layer.

Replay memory stores mini-batches of state, action, and reward to solve overfitting of the network. The network will be updated by a random selection of memory. It helps faster learning and not to forget past transitions. A learning rate of 0.001 and gamma 0.05 was the best setting for DQN with Replay Memory. The Q-Learning function for the Deep Q-Learning algorithm is the same as the standard q-learning function except it fills up replay memory and updates neural network using replayed memory data.

```
class DQN():
    # Deep Q Neural Network
    def __init__(self, state_dim=4, action_dim=2, hidden_dim=64, lr=0.001):
        self.criterion = torch.nn.MSELoss()
        self.model = torch.nn.Sequential(
                        torch.nn.Linear(state_dim, hidden_dim),
                        torch.nn.LeakyReLU(),
                        torch.nn.Linear(hidden_dim, hidden_dim*2),
                        torch.nn.LeakyReLU(),
                        torch.nn.Linear(hidden_dim*2, action_dim)
                )
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr)

    # Update weights of the network given a state and reward
    def update(self, state, y):
        y_pred = self.model(torch.Tensor(state))
        loss = self.criterion(y_pred, Variable(torch.Tensor(y)))
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

    # predict action based on the given state
    def predict(self, state):
        with torch.no_grad():
            return self.model(torch.Tensor(state))
```

```
# Expand DQN class with a replay memory functionality/.
class DQN_replay(DQN):
    def replay(self, memory, size=20, gamma=0.05):
        if len(memory) >= size:
            states = []
            targets = []
            # Sample a batch of experiences from the agent's memory
            batch = random.sample(memory, size)

            #Extract information from the data
            for state, action, next_state, reward, done in batch:
                states.append(state)
                # Predict q_values
                q_values = self.predict(state).tolist()
                if done:
                    q_values[action] = reward
                else:
                    q_values_next = self.predict(next_state)
                    q_values[action] = reward + gamma * torch.max(q_values_next).item()

                targets.append(q_values)
            self.update(states, targets)
```

```python
def q_learning(env, model, episodes, gamma=0.9, epsilon=0.3, eps_decay=0.99, replay=False,
               replay_size=20, title = 'DQL', double=False, n_update=20, soft=False, verbose=True):

    final = []
    memory = []
    episode_i=0
    for episode in range(episodes):
        episode_i+=1

        if double and not soft:
            # Update target network every n_update steps
            if episode % n_update == 0:
                model.target_update()
        if double and soft:
            model.target_update()

        # Reset state
        state = env.reset(istraining=True)
        done = False
        total = 0

        while not done:
            # Implement greedy search policy to explore the state space
            if random.random() < epsilon:
                action = env.action_space.sample()
            else:
                q_values = model.predict(state)
                action = torch.argmax(q_values).item()

            # Take action and add reward to total
            next_state, reward, done, _ = env.step(action)

            # Update total and memory
            total += reward
            memory.append((state, action, next_state, reward, done))
            q_values = model.predict(state).tolist()

            if done:
                if not replay:
                    q_values[action] = reward
                    # Update network weights
                    model.update(state, q_values)
                break

            if replay:
                # Update network weights using replay memory
                model.replay(memory, replay_size, gamma)
            else:
                # Update network weights using the last step only
                q_values_next = model.predict(next_state)
                q_values[action] = reward + gamma * torch.max(q_values_next).item()
                model.update(state, q_values)

            state = next_state

        # Update epsilon
        epsilon = max(epsilon * eps_decay, 0.01)
        final.append(total)
```

4.5.3 Double DQN with Replay Memory:

Calculating Q-value from the same neural network that updates by every step would make reward on a particular action overestimated. Therefore, a second neural network which is an early copy of the main network is used to double-check selected action for the observed state.

```
class DQN_double(DQN):
    def __init__(self, state_dim, action_dim, hidden_dim, lr=0.001):
        super().__init__(state_dim, action_dim, hidden_dim, lr)
        self.target = copy.deepcopy(self.model)

    # makeing predicitons usiug the target network
    def target_predict(self, s):
        with torch.no_grad():
            return self.target(torch.Tensor(s))

    # Update target network with the model weights
    def target_update(self):
        self.target.load_state_dict(self.model.state_dict())

    # the same replay memory functionality
    # except getting q-values from the second network
    def replay(self, memory, size, gamma=.05):
        if len(memory) >= size:
            data = random.sample(memory, size)
            states = []
            targets = []

            for state, action, next_state, reward, done in data:
                states.append(state)
                q_values = self.predict(state).tolist()
                if done:
                    q_values[action] = reward
                else:
                    q_values_next = self.target_predict(next_state)
                    q_values[action] = reward + gamma * torch.max(q_values_next).item()

                targets.append(q_values)
            self.update(states, targets)
```

Chapter 5

Result

This chapter presents the results from the Implementation done in Chapter 4. As discussed, three different datasets were available, all data datasets, 1440 dataset where days with missing data rows were taken out, and no water usage dataset where anytime heated water were all data with replay_tap=1 have been taken out. Additionally, Linear Regression engines perform better than LSTM. Then, we fit q-learning, DQN with Replay Memory, and Double DQN with Replay Memory algorithms on the gym environment to control the electric water heater by applying time, power price, and comfort penalties. The beginning temperature of EWH was chosen randomly which can affect the final result while the ideal temperature of an EWH is 70. Moreover, the acceleration and deceleration speed of LR engines can affect the result significantly and looking at results of “no water usage” dataset, it seems that power usage is significantly lower than other dataset which is due to the fact that fitted Linear Regression engine of this dataset drops temperature slower than other engines. Because, sudden drops of the temperature values have been taken out from this dataset. The rest of this chapter describes the results.

5.1 Q-Learning results

It took 2000 iterations to fit q-learning on the Linear Regression engine. Results will be shown in the following. As mentioned before, produced engines are not able to drop temperature below 60 while fitting on “no water usage” dataset as there is no temperature value below this number. Therefore, time penalties have not applied in any test cases of this dataset, as this penalty will drop temperature below 60-degree.

5.1.1 Q-Learning on all Data

No penalty has been applied apart from the power price. As it has shown at the beginning of chapter 4, the power price for the day was between 140 to 190, therefore 5 penalties perform during fitting q-learning that let the temperature to remained at 65 degrees. EWH power switch was on at 296 minutes a day, compared to 342 minutes a day from the real data. 342 minutes of a day from the raw data.

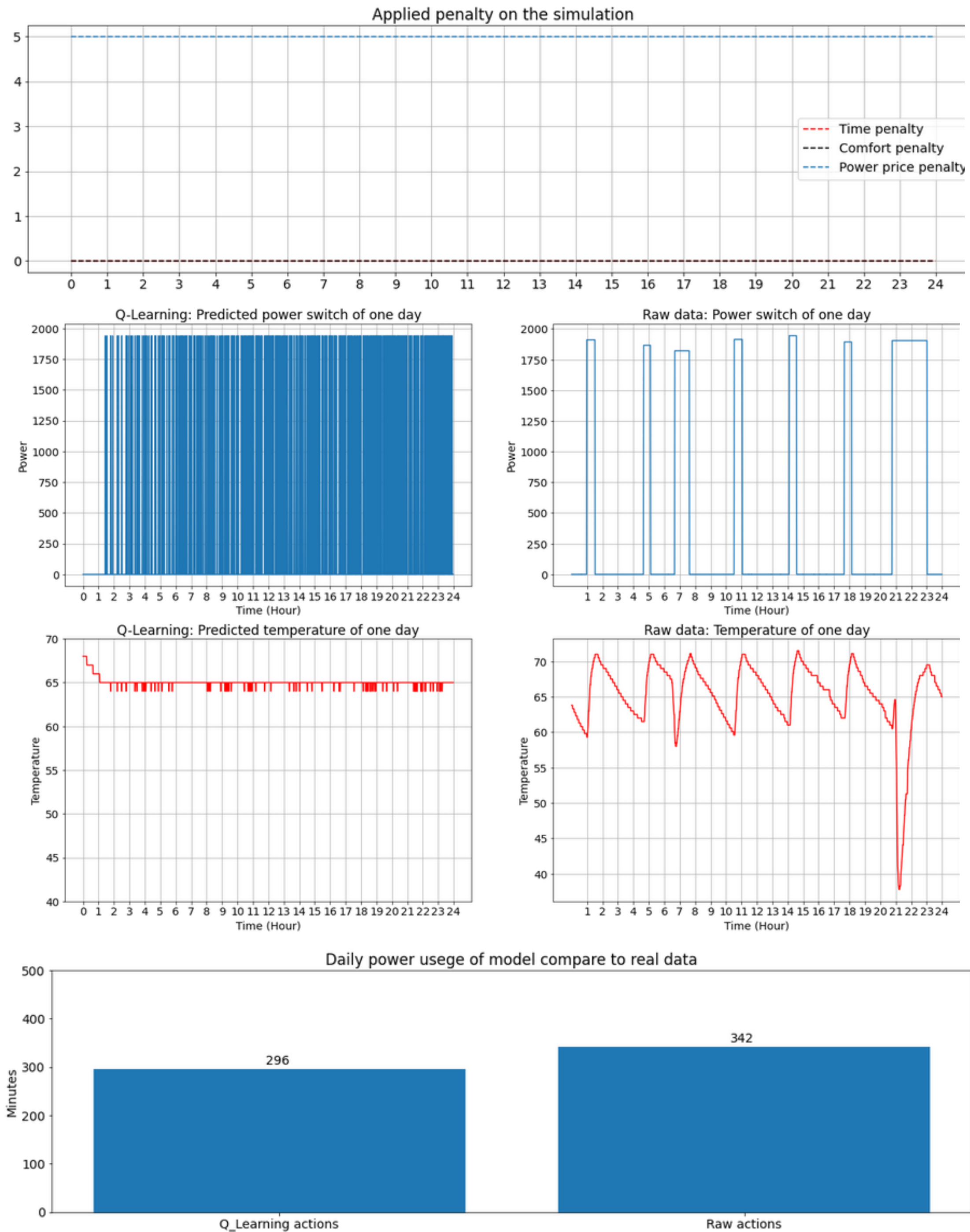

Figure 25: Q-Learning on "all data" dataset results

5.1.2 Q-Learning on no water usage data

EWH power switch was on at 146 minutes of a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 75-degree.

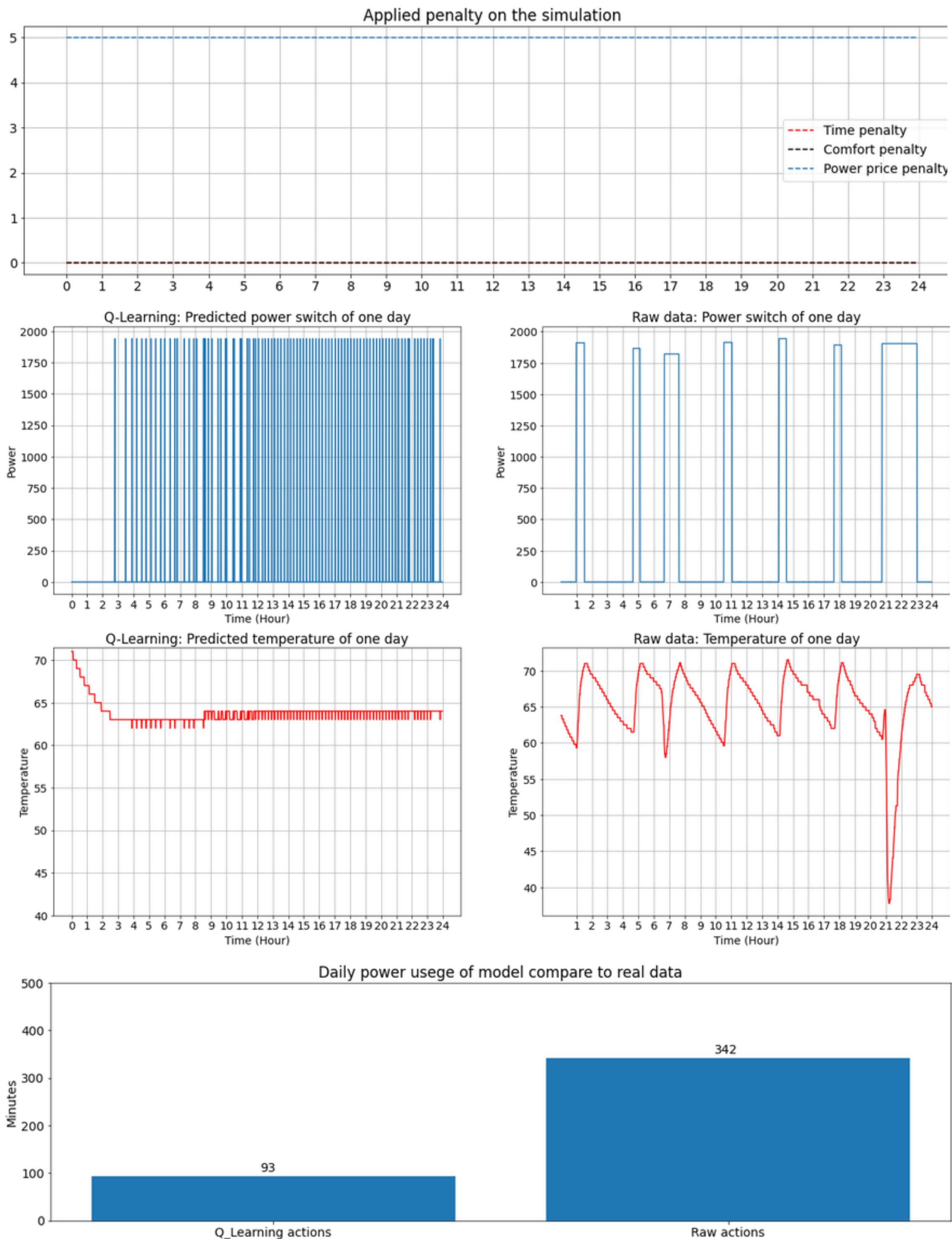

Figure 26: Q-Learning on “no water usage” dataset results

5.1.3 Q-Learning on 1440 dataset

EWH power switch was on at 314 minutes of a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 66-degree.

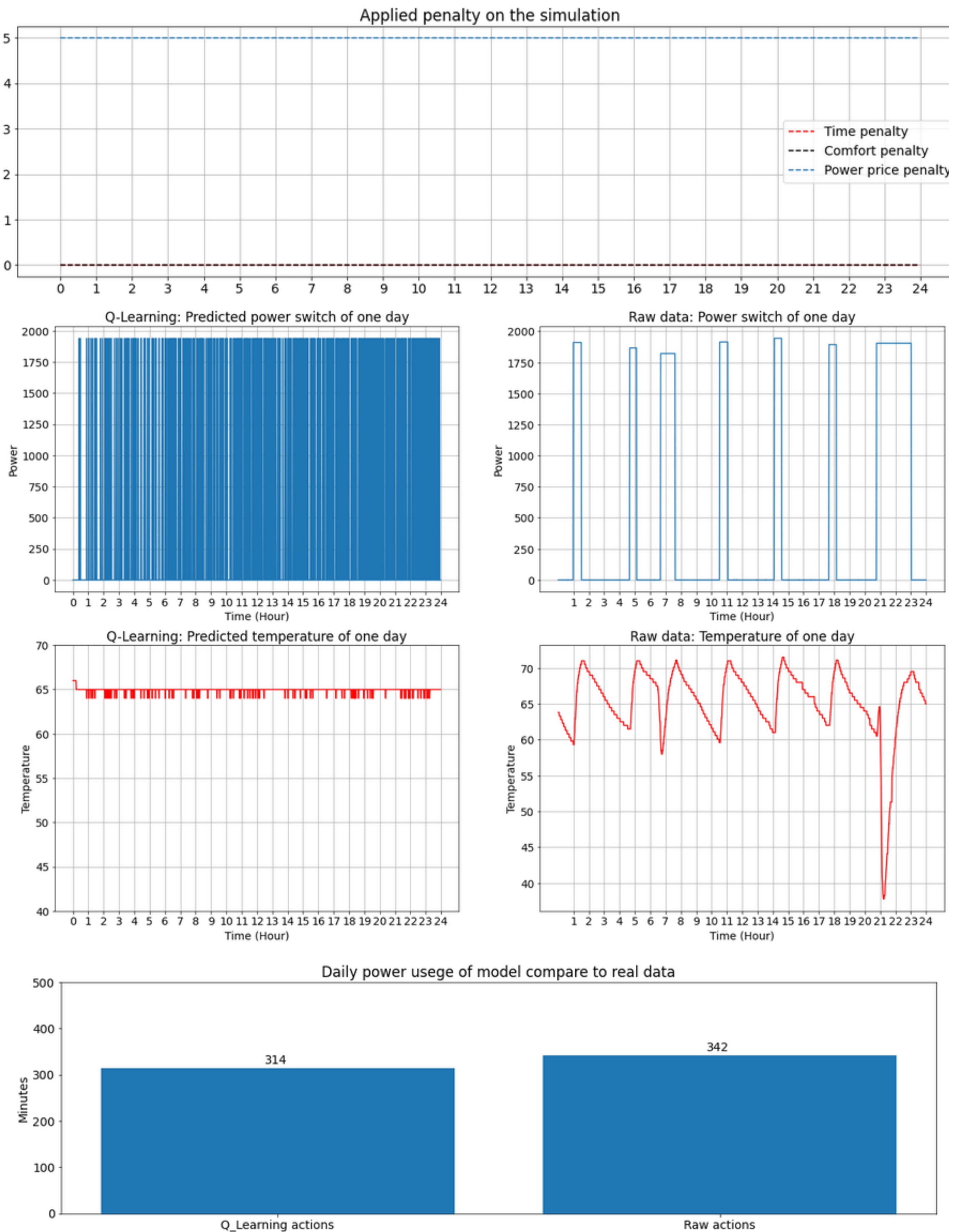

Figure 27: Q-Learning on "1440" dataset results

5.2 DQN with Replay Memory results

5 comfort penalties and 5 power price penalties have been applied during fitting DQN with replay memory. Additionally, 45 penalties were applied during the first and 3rd part of the day. Therefore, there were 10 penalties during 2nd and 4th part of the day and 55 penalties during the first and 3rd part of the day. But because room temperature is 29-degree, temperature did not drop below that.

5.2.1 DQN with Replay Memory on all Data

EWH power switch was on at 241 minutes of a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 63-degree.

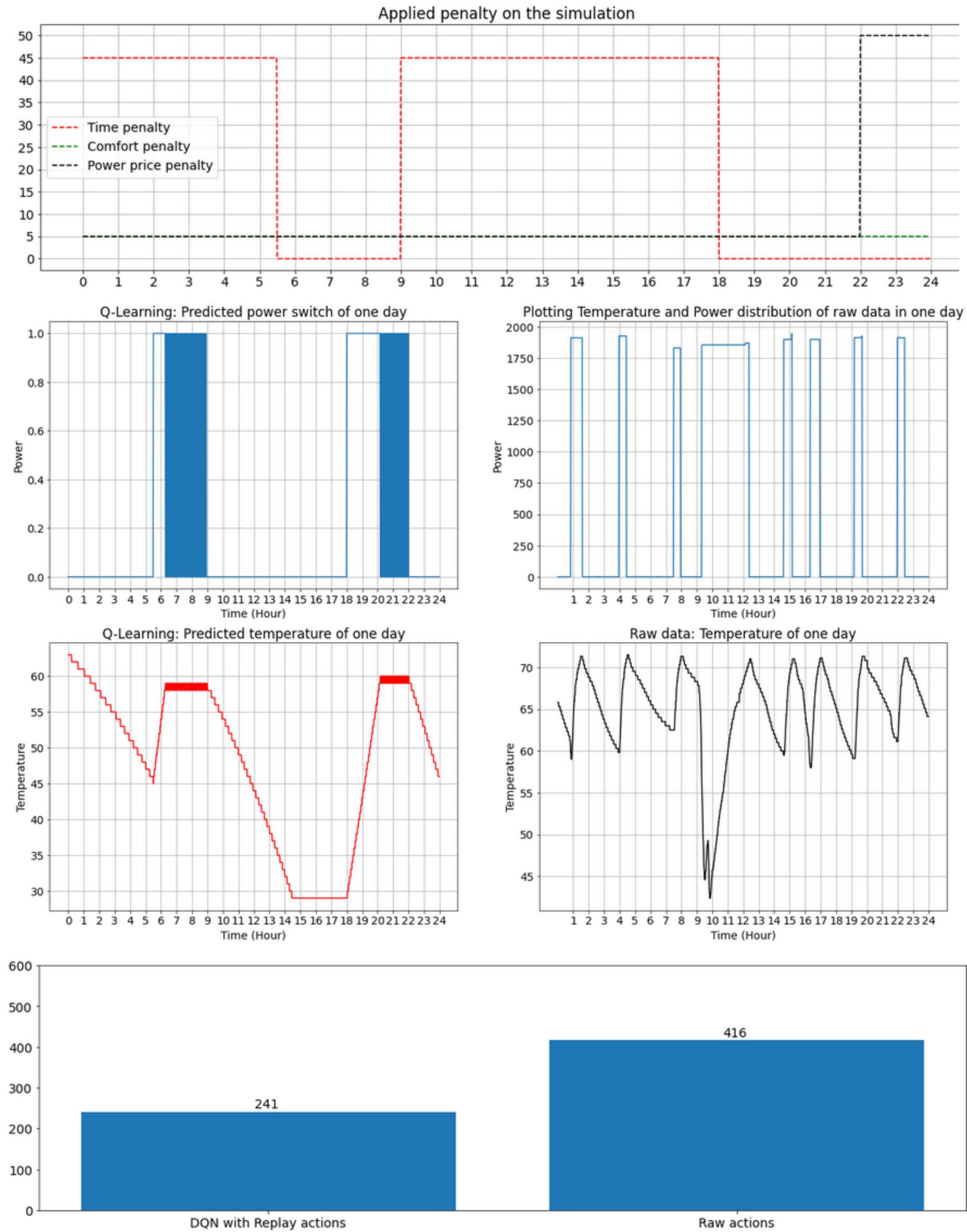

Figure 28: DQN with Replay Memory on "all data" dataset results

5.2.2 DQN with Replay Memory on no water usage dataset

EWH power switch was on at 80 minutes of a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 65-degree.

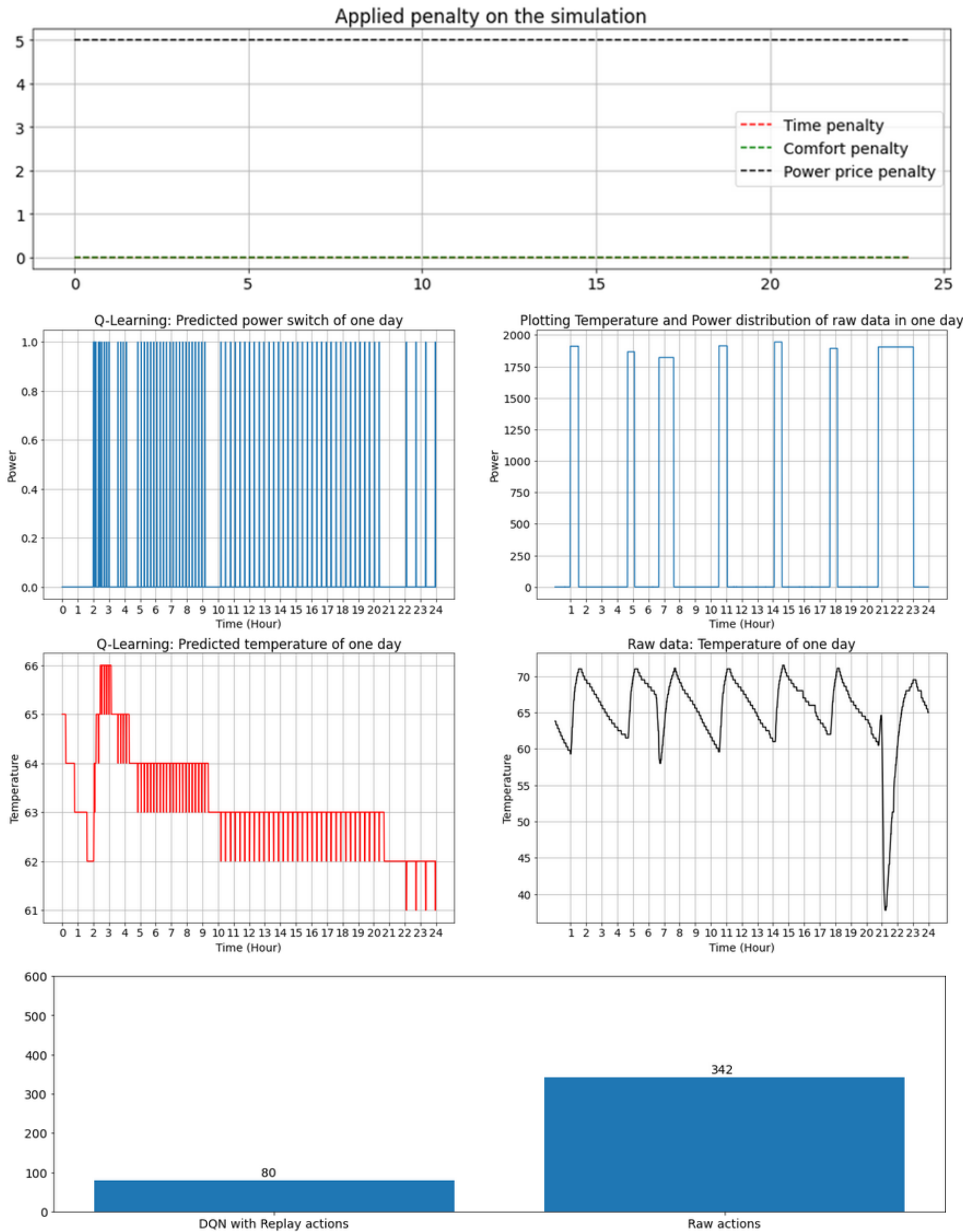

Figure 29: DQN with Replay Memory on “no water usage” dataset results

5.2.3 DQN with Replay Memory 1440 dataset

EWH power switch was on at 303 minutes of a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 67-degree.

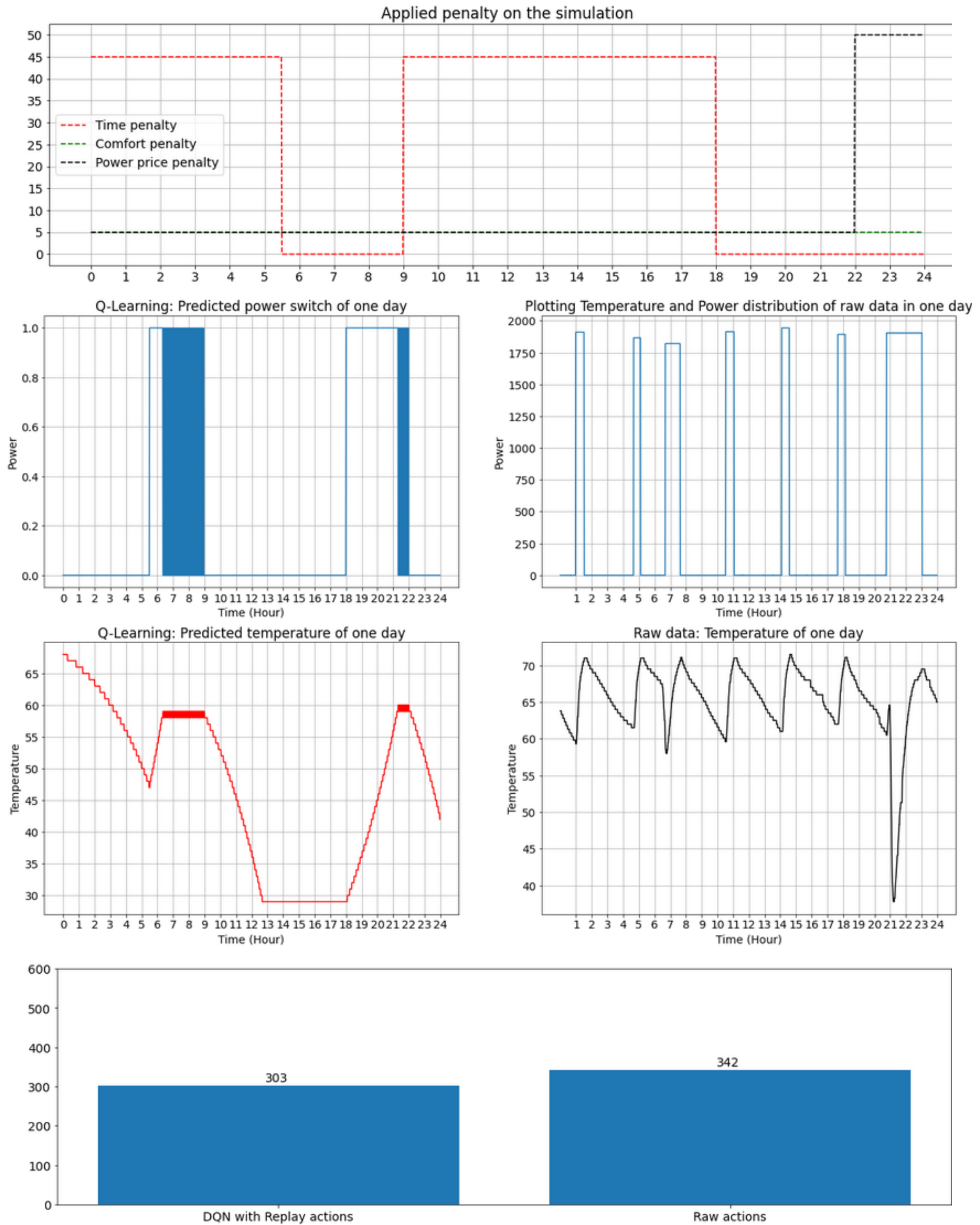

Figure 30: DQN with Replay Memory on "1440" dataset results

5.3 Double DQN with Replay Memory results

The same as DQN with Replay Memory, 5 comfort penalties and 5 power price penalties have applied during fitting DQN with replay memory as well as 45 penalties for the first and 3rd part of the day.

5.3.1 Double DQN with Replay Memory on all data:

EWH power switch was on for 323 minutes a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 66-degree.

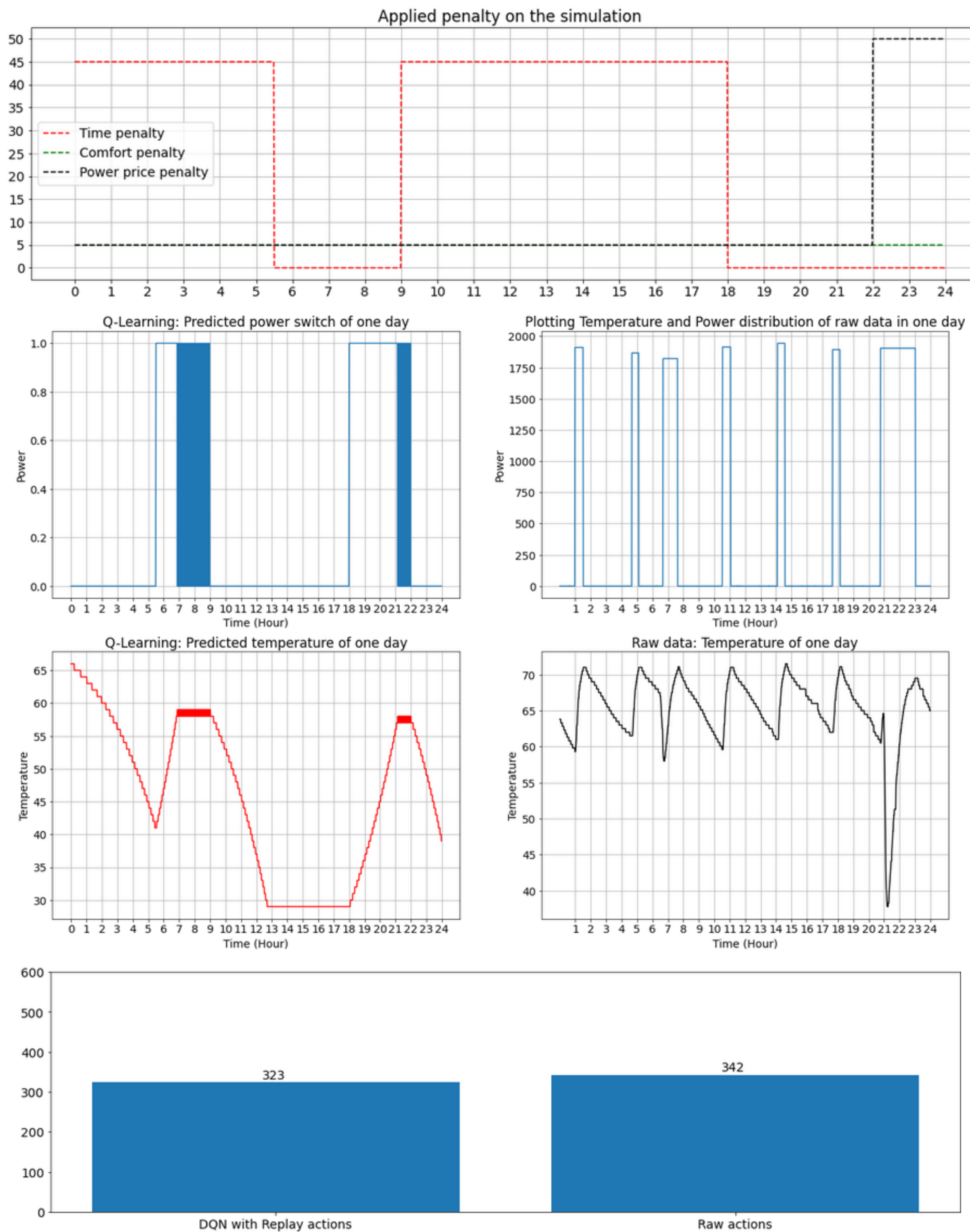

Figure 31: Double DQN with Replay Memory on "all data" dataset results

5.3.2 Double DQN with Replay Memory on no water usage dataset

EWH power switch were on at 96 minutes of a day, compare to 342 minutes of a day from the raw data and beginning temperature was 65-degree.

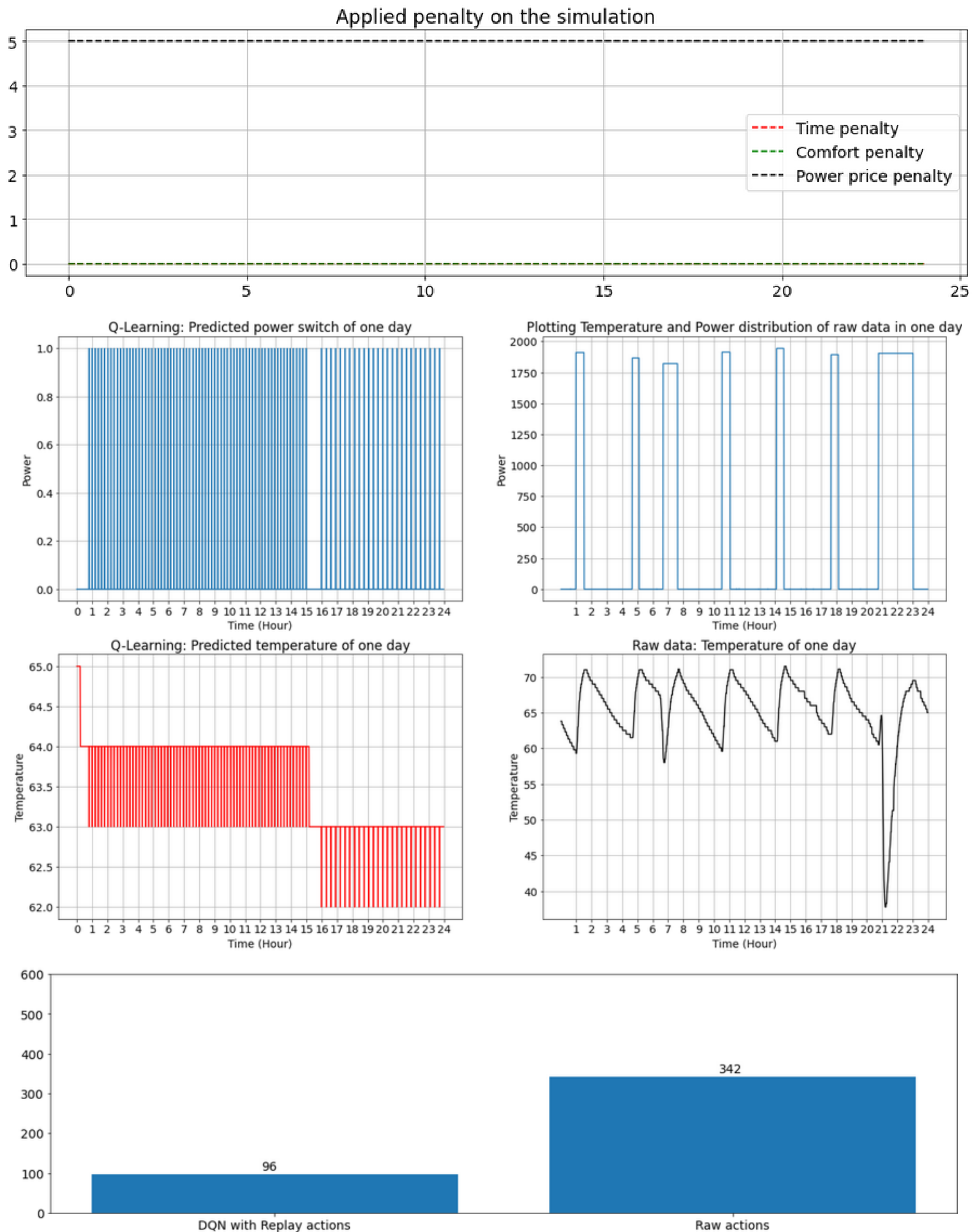

Figure 32: Double DQN with Replay Memory on "no water usage" dataset results

5.3.3 Double DQN with Replay Memory on 1440 dataset

EWH power switch was on at 198 minutes of a day, compared to 342 minutes of a day from the raw data and the beginning temperature was 63-degree.

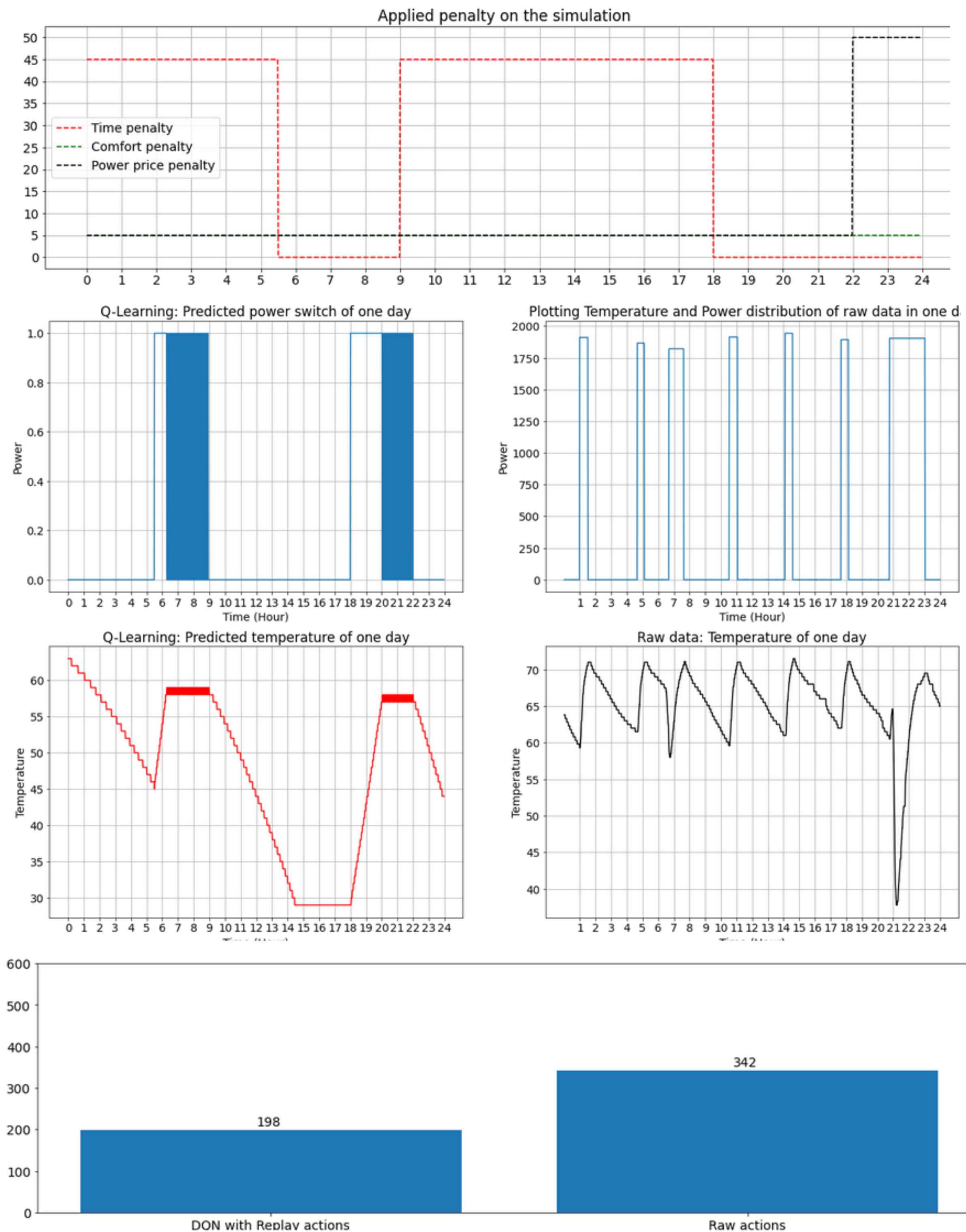

Figure 33: Double DQN with Replay Memory on "1440" dataset results

Chapter 6

Discussion and Conclusion

6.1 Discussion on the results

According to the results presented in Chapter 5, all RL models were able to control EWH. All models that have used the Linear Regression engine of “no water usage” dataset showed less power usage. It seems that other LR engines drop the temperature more quickly due to having temperature droppings in their dataset. Therefore, they used more power to keep the temperature up. Moreover, it took 2000 loops for standard q-learning to fit for each setting, 200 for DQN with Replay Memory, and less than 200 for Double DQN with Replay Memory.

6.2 Limitations

The biggest limitation that affect results a lot was the acceleration and deceleration speed of gym environment engines. The main range of temperature values in the data was 60 to 70, however, the better approach was to have a bigger range of temperature values in the raw data. This would result to have a more accurate prediction model in the gym simulators and more realistic q-learning results.

6.3 conclusion

The objective of this thesis was to investigate and simulate Reinforcement learning to control and optimize EWH. There are similar works to control and optimize EWH using different machine learning algorithms techniques. However, the result shows that Reinforcement learning can be a good option for controlling EWH.

In the very first chapter, in section 1.6, there were some research questions that this thesis was going to answer. The answers are as follows:

1. Is it possible to simulate a water heater using stored data? If so, what are the challenges in this regard, and which approaches generate a better engine to be used in a gym environment?

Ans: Yes. But it would be a better practice to have a bigger range in the temperature values of the raw data to fit a more accurate engine. LSTM shows that it was able to learn the pattern of the data

however, it was not able to pass the acceleration and deceleration speed test as well as Linear Regression. Linear Regression showed that it was able to learn data patterns and it also had a good result in the acceleration/deceleration speed test.

2. Is it possible to control the electric water heater using RL algorithms?

Ans: Yes, q-learning, DQN with Replay Memory, and Double DQN with Replay memory algorithms were all able to control EWH usage and they had almost the same result. However, it took a much longer time to fit q-learning compared to DQN and Double DQN.

3. Can Reinforcement Learning optimize the power usage of a water heater?

Ans: Results show that q-learning algorithms were able to drop the temperature when it is not needed or when the power price is high to a lower degree and save a lot of energy.

List of Figures

Figure 1: The relationship between input and output of Linear Regression 5
Figure 2: The relationship between input and output of Linear Regression in higher dimensions 5
Figure 3: Structure of a LSTM cell 6
Figure 4: Basic components of Q Learning 7
Figure 5: Cost/reward function of Q Learning 8
Figure 6: policy iteration of Q Learning 8
Figure 7: Bellman Equation in Q Learning [13] 9
Figure 8: The policy of maximum Q value... 9
Figure 9: Content of a Replay Memory entry 9
Figure 10: The equation of the Q value 10
Figure 11: The shape of temperature and power switch 14
Figure 12: The head of “all data” dataset 15
Figure 13: Overview of distribution of power and temperature in raw data 15
Figure 14: Taking out Days with missing rows 16
Figure 15: Taking out rows of used water in data 18
Figure 16: Power price on 21 May 2022 19
Figure 17: Linear Regression of “all data” dataset 20
Figure 18: Linear Regression of “1440” dataset 21
Figure 19: Linear Regression of “no water usage” dataset 21
Figure 20: Linear Regression acceleration/deceleration speed 22
Figure 21: LSTM on “all data” dataset 23
Figure 22: LSTM on “1440” dataset 24
Figure 23: LSTM on “no water usage” dataset 24
Figure 24: LSTM acceleration/deceleration speed 25
Figure 25: Q-Learning on “all data” dataset results 34
Figure 26: Q-Learning on “no water usage” dataset results 35
Figure 27: Q-Learning on “1440” dataset results 36
Figure 28: DQN with Replay Memory on “all data” dataset results 38
Figure 29: DQN with Replay Memory on “no water usage” dataset results 39
Figure 30: DQN with Replay Memory on “1440” dataset results 40
Figure 31: Double DQN with Replay Memory on “all data” dataset results 42
Figure 32: Double DQN with Replay Memory on “no water usage” dataset results 43
Figure 33: Double DQN with Replay Memory on “1440” dataset results 44

Appendix A

Downloadable content

A.1 Source code

The SourceCode of the project can be found in a repository at Github.com

Bibliography

[1] Wikipedia, "Energy conservation," [Online]. Available: https://en.wikipedia.org/wiki/Energy_conservation.

[2] "Energy efficiency and Artificial Intelligence," [Online]. Available: https://www.maximpact.com/artificial-intelligence-in-energy-efficiency/.

[3] F. C. B. J. Q. S. D. S. B. B. R. &. B. R. Ruelens, "Reinforcement Learning Applied to an Electric Water Heater: From Theory to Practice," 21 12 2016.

[4] J. Brownlee, "Linear Regression for Machine Learning," 25 03 2016. [Online]. Available: https://machinelearningmastery.com/linear-regression-for-machine-learning/.

[5] S. Saxena, "Introduction to Long Short Term Memory (LSTM)," 16 03 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

[6] G. B. a. V. C. a. L. P. a. J. S. a. J. S. a. J. T. a. W. Zaremba, "OpenAI Gym," p. 06, 05 06 2016.

[7] J. Selig, "What Is Machine Learning? A Definition.," 14 03 2022. [Online]. Available: https://www.expert.ai/blog/machine-learning-definition/.

[8] DataRobot, "How machine learning works," 07 01 2020. [Online]. Available: https://www.datarobot.com/blog/how-machine-learning-works/.

[9] t. f. e. From Wikipedia, "Reinforcement learning," [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&%20oldid=1029168114.

[10] A. Violante, "Simple Reinforcement Learning: Q-learning," 18 03 2019. [Online]. Available: https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56.

[11] S. Bansal, "RL Explained- Reinforcing the Intuition and Math," 17 02 2020. [Online]. Available: https://medium.datadriveninvestor.com/rl-explained-reinforcing-the-intuition-and-math-fd1185369186.

[12] t. f. e. Wikipedia, "Deep reinforcement learning," [Online]. Available: https://en.wikipedia.org/wiki/Deep_reinforcement_learning.

[13] R. Kurban, "Deep Q Learning for the CartPole," 30 12 2019. [Online]. Available: https://towardsdatascience.com/deep-q-learning-for-the-cartpole-44d761085c2f.

[14] D. L. C. 5. o. 6. -. L. A. -. C. a. Mandy, "Reinforcement Learning - Developing Intelligent Agents," [Online]. Available: https://deeplizard.com/learn/video/Bcuj2fTH4_4.

[15] J. M. K. K. T. K. H. Z. Kadir Amasyali, "Deep Reinforcement Learning for Autonomous Water Heater Control," 2021.

[16] t. f. e. Wikipedia, "Long short-term memory," [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory.

[17] J. Brownlee, "Dropout with LSTM Networks for Time Series Forecasting," 28 05 2017. [Online]. Available: https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/.

[18] J. Brownlee, "Use Early Stopping to Halt the Training of Neural Networks At the Right Time," 10 12 2018. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

[19] B. J. C. S. Q. B. D. S. R. B. a. R. B. F. Ruelens, "Reinforcement Learning Applied to an Electric," 2016.

[20] N. Pool. [Online]. Available: https://www.nordpoolgroup.com/en/.