DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2021

Deep Reinforcement Learning for Temperature Control in Buildings and Adversarial Attacks

KEVIN AMMOURI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Deep Reinforcement Learning for Temperature Control in Buildings and Adversarial Attacks

KEVIN AMMOURI

Master's Programme, Computer Science, 120 credits
Date: June 18, 2021
Supervisor: Alessio Russo, Ph.D.
Examiner: Prof. Carlo Fischione
School of Electrical Engineering and Computer Science

Deep Reinforcement Learning for Temperature Control in Buildings and Adversarial Attacks

Abstract

Heating, Ventilation and Air Conditioning (HVAC) systems in buildings are energy consuming and traditional methods used for building control results in energy losses. The methods cannot account for non-linear dependencies in the thermal behaviour. Deep Reinforcement Learning (DRL) is a powerful method for reaching optimal control in many different control environments. DRL utilizes neural networks to approximate the optimal actions to take given that the system is in a given state. Therefore, DRL is a promising method for building control and this fact is highlighted by several studies. However, neural network polices are known to be vulnerable to adversarial attacks, which are small, indistinguishable changes to the input, which make the network choose a sub-optimal action. Two of the main approaches to attack DRL policies are: (1) the Fast Gradient Sign Method, which uses the gradients of the control agent's network to conduct the attack; (2) to train a a DRL-agent with the goal to minimize performance of control agents.

The aim of this thesis is to investigate different strategies for solving the building control problem with DRL using the building simulator IDA ICE. This thesis is also going to use the concept of adversarial machine learning by applying the attacks on the agents controlling the temperature inside the building.

We first built a DRL architecture to learn how to efficiently control temperature in a building. Experiments demonstrate that exploration of the agent plays a crucial role in the training of the building control agent, and one needs to fine-tune the exploration strategy in order to achieve satisfactory performance. Finally, we tested the susceptibility of the trained DRL controllers to adversarial attacks. These tests showed, on average, that attacks trained using DRL methods have a larger impact on building control than those using FGSM, while random perturbation have almost null impact.

Keywords

Deep Reinforcement Learning, Adversarial Attacks, Optimal Attacks, Building Control, Optimal Control, Energy Efficiency

Sammanfattning

Ventilationssystem i byggnader är energiförbrukande och traditionella metoder som används för byggnadskontroll resulterar i förlust av energisparande. Dessa metoder kan inte ta hänsyn till icke-linjära beroenden i termisk beteenden. Djup förstärkande inlärning (DRL) är en kraftfull metod för att uppnå optimal kontroll i många kontrollmiljöer. DRL använder sig av neurala nätverk för att approximera optimala val som kan tas givet att systemet befinner sig i en viss stadie. Därför är DRL en lovande metod för byggnadskontroll och detta faktumet är markerat av flera studier. Likväl, neurala nätverk i allmänhet är kända för att vara svaga mot adversarial attacker, vilket är små ändringar i inmatningen, som gör att neurala nätverket väljer en åtgärd som är suboptimal.

Syftet med denna anvhandling är att undersöka olika strategier för att lösa byggnadskontroll-problemet med DRL genom att använda sig av byggnads-simulatorn IDA ICE. Denna avhandling kommer också att använda konceptet av adversarial machine learning för att attackera agenterna som kontrollerar temperaturen i byggnaden. Det finns två olika sätt att attackera neurala nätverk: (1) Fast Gradient Sign Method, som använder gradienterna av kontrollagentens nätverk för att utföra sin attack; (2) träna en inlärningsagent med DRL med målet att minimera kontrollagenternas prestanda.

Först byggde vi en DRL-arkitektur som lärde sig kontrollera temperaturen i en byggad. Experimenten visar att utforskning av agenten är en grundläggande faktor för träningen av kontrollagenten och man måste finjustera utforskningen av agenten för att nå tillfredsställande prestanda. Slutligen testade vi känsligheten av de tränade DRL-agenterna till adversarial attacker. Dessa test visade att i genomsnitt har det större påverkan på kontrollagenterna att använda DRL metoder än att använda sig av FGSM medans att attackera helt slumpmässigt har nästan ingen påverkan.

Nyckelord

Djup förstärkande inlärning, Adversarial Attacker, Optimala Attacker, Byggnadskontroll, Optimal Kontroll, Energieffektivitet

Acknowledgments

I would like to express my graditude to Alessio Russo (Ph.D) who has been an excellent supervisor throughout my thesis project and his expertise never cease to amaze me. Russo has shown great knowledge of adversarial attacks and without his previous papers on optimal attacks this project would not have been possible. I would also want to thank Prof. Carlo Fiscione for being my examiner and allowing the administrative measures to go very smoothly.

Stockholm, June 2021
Kevin Ammouri

Contents

List of Figures

List of Tables

List of Algorithms

List of acronyms and abbreviations

AHU Air Handling Unit

CA Control Agent

DDPG Deep Deterministic Policy Gradient

DDQN Double Deep Q-Network

DQN Deep Q-Network

DRL Deep Reinforcement Learning

FC Fully-Connected

HVAC Heating Ventilation and Air Conditioning

MDP Markov Decision Process

PID Proportional–Integral–Derivative

PRBS Pseudorandom binary sequence

ReLU Rectified Linear Unit

RL Reinforcement Learning

VRFT Virtual Reference Feedback Tuning

Chapter 1

Introduction

Deep Reinforcement Learning (DRL) has shown to be promising in crafting optimal policies for complex system dynamics. Notable examples such as playing Atari games [3, 4], playing Go [5] and accomplishing excellent performance in other control settings with continuous state and action spaces [6]. Deep reinforcement learning has also been applied to Heating, Ventilation and Air Conditioning (HVAC) systems for building control, and promising results have been brought forward by several studies. By installing sensors in buildings for measuring the different thermal behaviours, data can be utilized by DRL algorithms to optimally control the building temperature and reduce energy consumption overall. The goal for the agent is to maximize the comfort of the occupants in the building while considering the energy consumption of the system given the thermal input states [7]. There are various methods proposed in solving the control and energy optimization such as Model Predictive Control (MPC), Fuzzy Control [8, 9] and other traditional methods. However, non-linear dependencies in the thermal behaviour can result in energy losses when using these conventional techniques [2].

In optimal control, reinforcement learning is used to optimally control discrete-time dynamical systems. When working with environments where the state space and/or the action space is continuous, different approximation methods can be used to derive the optimal policy. For example, one can combine reinforcement learning deep networks (a.k.a Deep Reinforcement learning). However, control policies parametrized using deep neural networks have shown to be vulerable to very small perturbations of the input. These perturbations are minimal, indistinguishable changes to the input, making the DRL system regard the state as a completely different one and take a sub-optimal action with high confidence. This could have devastating

effects in some applications where DRL systems are implmented. The small perturbations to the input are referred to as adversarial attacks, which is also the focus in this thesis.

This thesis investigates both white- and black-box attacks in the context of adversarial attacks. Black-box attacks are when the adversary (the attacker) is not familiar with the system's structure and can only observe the rewards of the agent controlling the environment. In contrast, white-box attacks consider cases where the adversary can access the whole system, specifically the architecture and parameters. Hence, attacks that concern themselves with the internal structure of the system being attacked, are only possible in a white-box setting.

First, we consider the problem of designing a DRL agent for temperature control in buildings. Next, we investigate the effects of adversarial attacks on the control performance by making slight changes to the measurements sent from the sensors. Although there is no specific way to mitigate the impact of adversarial attacks completely, different approaches can be taken to make the control agent more robust to these attacks, and the ability to detect the attacks. Implementation of detection methods and increasing robustness of the agents is left as an interesting direction for future work.

To summarize, this thesis provides the following contributions:

- Show the effect of adversarial attacks in a simple control environment.
- The design of a deep reinforcement learning agent for HVAC system control to optimally control the temperature in a building given the thermal state inputs.
- Show the effects on the agent's performance controlling the building by applying different adversarial attacks.

1.1 Problem

The measurable objectives of this thesis are the following:

- Can we use deep reinforcement learning to perform temperature control in buildings compared to current adopted methods?
- How can a hacker optimally attack the system, given that it has access to the sensors, by changing the measurements of the input signal?

1.2 Purpose

When creating a system that uses neural networks, it is important to consider the vulnerability of them. Minimal changes to the original input can make the neural network output a suboptimal action with high confidence. For instance, consider a setting where a DRL agent controls a vehicle, and the images it captures along the road are the states for which it decides what action to take. There can be some configuration of the paint along the road that makes the agent regard the image as a different state than what it is and take a sub-optimal action, which could have devastating effects. Therefore, when implementing such a system, these adversarial perturbations need to be considered to make the agent overall more robust. Thus, this thesis aims to show that these attacks can be easily constructed, and can be used in any environment. Therefore, different measurements have to be taken to ensure that these systems can be trusted and are secure.

1.3 Goal

The goal of this thesis is to achieve satisfactory results in optimally controlling HVAC systems with DRL and show that these systems can be subjected to adversarial attacks.

1.4 Research Methodology

The research process in this thesis is conducted in two different parts. The first part consists of qualitative research regarding different adversarial attacks, how they are implemented, and why they work in theory. Moreover, adversarial attacks, both white- and black-box attacks, are evaluated to distinguish which attacks are more powerful than the others on environments in the OpenAI's Gym library [10]. The second part also consists of a qualitative research about DRL-HVAC systems and what to take into consideration when implementing such a system for building control. This includes data pre-processing, reward functions, interpreting the system dynamics and finding a suitable network architecture.

1.5 Ethics and Sustainability

The sustainability aspect of this thesis is the fact of highlighting the drawbacks of implementing deep reinforcement learning agents for building control. People who consider implementing DRL for any system has to take these attacks into consideration. Furthermore, the ethics is debateable if this were to be implemented in a real-world setting. However, since this is a simulated setting with no harm to the outside world, the ethics can be disregarded. If one were to implement a DRL-HVAC system in a real building then the ethical responsibilities cannot be overlooked. Nevertheless, it is important to consider investigating these concepts in a real-world setting, because of the systems' vulnerabilities to these attacks.

1.6 Structure of the thesis

Chapter 2 of this thesis presents the background knowledge required to understand the different algorithms implemented in this work, and the fundamentals of reinforcement learning. Chapter 3 provides the state of the art methods of DRL for building control and adversarial attacks together with the relevant related work for these concepts. Chapter 4 describes the approach taken to conduct the experiments for both the building control problem and the adversarial attacks. Chapter 5 presents the results and discussion of the experiments. Finally, chapter 6 gives the conclusion for this work and ideas for future work.

Chapter 2

Background

In this chapter, the reinforcement learning paradigms are introduced and thoroughly described by starting with the building block of RL, which is the Markov Decision Process. The last section describes the topic of deep reinforcement learning and is vital for understanding the work conducted in this thesis.

2.1 Markov Decision Process

In optimal control, discrete-time stochastic processes can be modeled using the Markov Decision Process (MDP) framework [11]. This model is important since it can be used to frame optimal control problems. An MDP can be described as follows:

$$\mathcal{M} = \{\mathcal{S}, (\mathcal{A}_s, s \in \mathcal{S}), P, R\},$$

where $\mathcal{S}$ is defined as the state-space, and $\mathcal{A}_s$ is the set of actions to take in a given state s. P represents the state transition probabilities of the system, thus, $P(s'|s, a)$ is the probability of transitioning to state s' given that the agent is in state s and chose action a, with $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}_s$. Moreover, $R(s, a, s')$ represents the expected immediate reward, when action a is taken in state s, leading to a transition to the next state s'.

The objective of the MDP $\mathcal{M}$ is to find a policy π that maximizes the expected total discount reward for the agent

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t-1} R(s_t, a_t, s_{t+1}) | s_0 = s\right] \quad s \in \mathcal{S}, \tag{2.1}$$

with discount factor $\gamma \in (0, 1)$. The discount factor determines how myopic the agent is to future rewards. If the discount factor is 0, the agent only cares about the immediate rewards. Furthermore, if the discount factor is strictly larger than 0 and strictly less than 1, one can guarantee convergence of the infinite sum series for MDPs with infinite time horizons [12].

A policy π, is a mapping between states and actions and can be deterministic or stochastic. Deterministic policies map each state in $\mathcal{S}$ to a specific action in $\mathcal{A}$, $\mathcal{S} \to \mathcal{A}$, whereas stochastic policies map each state to a probability distribution over actions $\mathcal{S} \to \omega_{\mathcal{A}}$. Hence, for stochastic policies, $\pi(\cdot|s)$ is a probability distribution over $\mathcal{A}_s$ [13]. The policy describes how the agent should proceed in the environment given the current state s by choosing action suggested by $\pi(s)$.

Solving MDPs amounts to finding a policy that maximizes the total collected reward. In the infinite-time horizon case, with discounted rewards, it can be solved by means of traditional algorithms, such as value iteration, policy iteration and approximated dynamic programming [14]. Specifically, the goal is to compute the following quantity $V^\star(s)$ for all s:

$$V^\star(s) = \max_{a \in \mathcal{A}_s} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^\star(s') \right]. \tag{2.2}$$

For finite time horizon, the equation can be solved through dynamic programming. For all terminal states s_T, $u_T^\star(s_T) = \max_a r_T(s_T, a)$, and for all timesteps from $T - 1$ to 0 the equation can be solved through

$$u_t^\star(s_t) = \max_{a \in \mathcal{A}_{s_t}} \left[R_t(s_t, a) + \sum_{s' \in \mathcal{S}} P_t(s'|s_t, a) u_{t+1}^\star(s') \right], \tag{2.3}$$

hence, the optimal value function at time T is $V_T^\star = u_0^\star$, since $u^\star$ is computed with backward recursion. However if the state-space is too large, this computation can be too exhausting and one has to resort to approximating the optimal value function, which can be done in several ways and is showcased sections 2.2 and 2.3 [12].

2.2 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning where the goal of the agent is to maximize the cumulative reward collected in the environment. Unlike supervised learning, the agent is not bound to some training dataset;

instead, it learns from the experiences it observes while traversing the environment. Moreover, reinforcement learning is used when the underlying MDP is unknown. RL uses online learning to solve the Bellman's equation, that is, to find the optimal policy using the data/experiences gathered from the environment. In the following section we describe some general reinforcement learning paradigms.

2.2.1 Q-learning

Q-learning is a model-free reinforcement learning algorithm, meaning that it does not need a model to learn. The function used by the Q-learning algorithm computes the value describing the quality of each (state, action) pair $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

The Q-learning update function at timestep t can be written as

$$Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha_t \left[r_t + \gamma \max_{b \in \mathcal{A}} Q^{(t)}\left(s_{t+1}, b\right) - Q^{(t)}\left(s_t, a_t\right) \right]. \tag{2.4}$$

Under the Q-learning algorithm, if $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$, and for discount factor $\gamma \in (0, 1)$, as well as ensuring that the state-action pairs are visited infinitely often then

$$\lim_{t \to \infty} Q^{(t)} = Q^{\star}. \tag{2.5}$$

$Q^{\star}$ can be used to derive the optimal policy for the environment by always choosing the greedy action in $Q^{\star}(s_i)$ for all $s_i \in \mathcal{S}$. [15]

ε-greedy policy

The agent learning a particular task can follow a behaviour policy for updating their state-action value pair, and use that policy for control. For instance, the Q-learning algorithm discussed in the previous section does not explicitly follow a greedy policy when updating the state-value action pairs, hence, it is an off-policy algorithm. For on-policy algorithms, the state-action value pairs are updated each step given the next state s' and the policy action a_π rather than the greedy action as in the Q-learning algorithm.

The ε-greedy policy defines a probability distribution over a set of actions given the current state. The probability of an action $a_t \in \mathcal{A}_{s_t}$ being selected

by the agent in state s_t in round t is [15]:

$$a_t = \begin{cases} \text{uniform}(\mathcal{A}_{s_t}) & \text{w.p. } \varepsilon \\ \arg\max_{b \in \mathcal{A}_{s_t}} Q^{(t)}(s_t, b) & \text{w.p. } 1 - \varepsilon \end{cases} \tag{2.6}$$

where $\varepsilon \in [0, 1]$. This type of policy favors exploration of the environment, since there is always a positive probability of picking any action. The deep Q-network used in this thesis, seen in section 2.3.1, makes use of the ε-greedy policy.

2.3 Deep Reinforcement Learning

As mentioned in the previous sections, when dealing with a large or continuous state-space, it is usually more efficient to approximate the $Q^\star$ function through different generalization methods, such as randomized trees, kernel-based methods and neural networks [16]. Deep reinforcement learning combines the idea of reinforcement learning and deep learning; hence, it uses deep neural networks in order to approximate the $Q^\star$ function. There are different neural network structures and algorithms, for constructing deep RL agents. Algorithms for training deep neural networks include Deep Deterministic Policy Gradient (DDPG) [6], Proximal Policy Optimization (PPO) [17], and Deep Q-Networks (DQN) [18], to name a few. Standard DQN and DDPG algorithms are suited only for environments with discrete and continuous action-spaces, respectively, while PPO can be applied for either type. In this section, however, only the DQN and a modified version of the algorithm called Double DQN [19] is covered as these are relevant to the thesis.

2.3.1 Deep Q-Networks

Deep Q-Networks (or DQN) is used for approximating the $Q^\star$ using the deep Q-function algorithm, using the ε-greedy policy mentioned in section 2.2.1, as seen in Algo. 1. ε should decrease over time when using DQN, making the function favor the greedy action because the agent gets better at determining which actions to take in a specific state. The exploration rate ε can either be decreased linearly or exponentially for each episode. Let T denote the total number of episodes then ε can be decreased exponentially by

$$\varepsilon_t = \max\left(\varepsilon_{\min}, \varepsilon_{\max} - \frac{(\varepsilon_{\max} - \varepsilon_{\min})\,(t-1)}{T-1}\right), \quad t = 1, 2, \ldots \tag{2.7}$$

and for a linearly decreasing exploration rate

$$\varepsilon_t = \max\left(\varepsilon_{\min}, \varepsilon_{\max}\left(\frac{\varepsilon_{\min}}{\varepsilon_{\max}}\right)^{\frac{t-1}{T-1}}\right), \quad t = 1, 2, ... \tag{2.8}$$

where $\varepsilon_{\max}$ and $\varepsilon_{\min}$ correspond to values for the initial exploration rate and the last episode's exploration rate.

DQN is made up of three components; a main network, a target network and the experience replay buffer. The main network is used for generating the current Q-values that are used for deciding which action to take. More specifically, the action generated on line 9 in Algo. 1 depends on the output of the main network. The target network is essential for the stability of DQN because the parameters of the main network are updated using a batch of state-action value pairs, while the regular Q-learning algorithm seen in section 2.2.1 updates the pairs one at a time. This results in the next states having a different value since they were updated with the current state. To keep the pairs stable, one utilizes a target network that is updated every C steps as seen in the algorithm. Finally, the experience replay buffer $\mathcal{B}$ (also seen in Algo. 1) stores experiences generated from the environment. By using an experience replay buffer we can decorrelate the samples extracted from the buffer, which is a source of instability in off-policy learning with function approximation.

The target y is computed on line 13 of Algo. 1, as one can see, this calculation is similar to the Q-learning algorithm when updating the value of a state-action pair. However, if the next state is a terminal state, only the reward is considered as a target. In line 14 of Algo. 1 the mean square error loss between the target and the prediction of the main network is calculated, and finally, the parameters of the main network is copied over to the target network if C steps have passed since the last copy.

Double DQN

Hasselt et al. [19] introduces a different way to compute the target values because according to [19], the Q-function tends to overestimate the action values and they introduce double DQN as a solution to this problem. The estimation of the target is done by the target network for the next states and the actions are chosen by the main network depending on these states. This procedure is different from the general DQN algorithm, as the DQN algorithm computes the target only with respect to the output of the target network. For

Algorithm 1 Episodic DQN with ε-greedy exploration

Input Discount factor γ; buffer size L; number of episodes T; target network frequency update C; Training batch size N; exploration rate $\{\varepsilon_k\}_{k=1}^{T}$
procedure
    Initialize network and target network $Q_\theta, Q_{\theta'}$
    Initialize replay buffer $\mathcal{B}$ with maximum size L, and fill with random experiences.
    for episodes $k = 1, 2, ..., T$ **do**
        Reset environment and read initial state s_0
        $t \leftarrow 0$
        while Episode k not done **do**
            Take ε-greedy action a_t ▷ w.r.t. $Q_\theta(s_t, a)$
            Execute action a_t in the environment and observe (r_t, s_{t+1})
            Append (s_t, a_t, r_t, s_{t+1}) to $\mathcal{B}$
            Randomly sample a batch $\mathcal{D} \sim \mathcal{B}$ of size N
            For each experience (s_i, a_i, r_i, s_{i+1}) in $\mathcal{D}$

$$y_i = \begin{cases} r_i + \gamma \max_b Q_{\theta'}(s_{i+1}, b) & \text{if } s_{i+1} \text{ is not terminal} \\ r_i & \text{otherwise} \end{cases}$$

            Update θ by minimizing the MSE loss $\frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (y_i - Q_\theta(s_i, a_i))^2$
            if t mod C = 0 **then**
                $\theta' \leftarrow \theta$
            $t \leftarrow t + 1$

double DQN, the target is computed as

$$y_i = \begin{cases} r_i + \gamma \max_a Q_{\theta'}\left(s_{i+1}, \arg\max_a Q_\theta(s_{i+1}, a)\right) & \text{if } s_{i+1} \text{ is not terminal} \\ r_i & \text{otherwise.} \end{cases} \tag{2.9}$$

Chapter 3

State of the art

This chapter presents different state of the art methods, and related work, regarding DRL-HVAC systems and adversarial attacks. Section 3.1 covers deep reinforcement learning for building control and section 3.2 presents attacks on reinforcement learning policies. Results and examples of the aforementioned subjects are showcased at the end of each section.

3.1 Deep Reinforcement Learning for Building Control

DRL-based HVAC Control has gained more attention in recent years. Conventional methods used for building control regarding energy efficiency results in energy losses because the methods cannot account for the complex non-linear dynamics of the system states. The optimization problem can be solved by using a DRL-based control system, which tries to optimally control the system.

3.1.1 HVAC Architecture

Russo et al. [1], shows a model of a HVAC architecture, and this model can be seen in figure 3.1. As seen in the figure, the HVAC is controlled using Virtual Reference Feedback Tuning (VRFT) [20] in the model used by [1]. However, for deep reinforcement learning in building control, the ventilation is controlled using DRL. Nonetheless, the figure shows the entire process from the Air Handling Unit (AHU), receiving external air, to the airflow being propagated into the apartments. Based on the measurements received from the

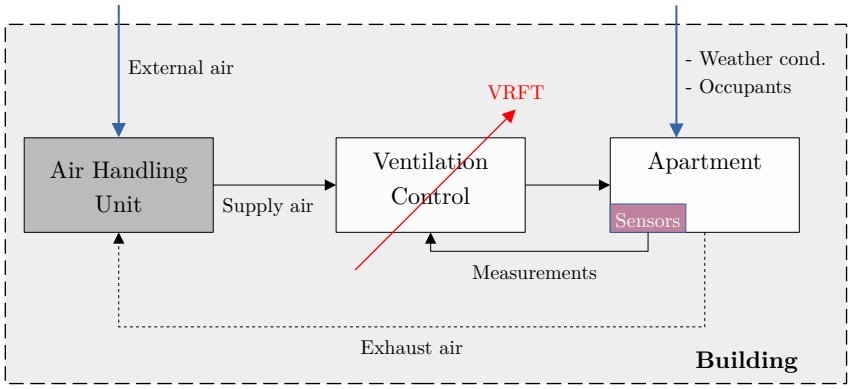

Figure 3.1: HVAC architecture model controlled using Virtual Reference Feedback Tuning (VRFT) [1].

sensors inside the apartment, the system uses this data to optimize the control system and keep the temperature at the desired comfort range.

3.1.2 Deep Reinforcement Learning for Temperature Control

A short paper published by Le Coz et al. [2] implements a control strategy that employs neural network for district heating control in China. First, they trained a recurrent neural network using simulated data in order to predict the air temperature inside each zone, and then used the model to construct the DRL agents. The reward received by the agents is

$$r_t = -|T_t^{in} - T_{target}|, \tag{3.1}$$

for every zone in the building that the agent is operating within, where T_t^{in} is the indoor temperature at timestep t and T_{target} denotes the target temperature. This implies that the agent gets less reward the more the indoor temperature deviates from the target temperature, which was set to $18°$C. However, their reward function does not account for the energy cost when taking a specific action in a given state, which leads to the agent not considering maintaining a low energy cost.

In [7] they implement a DRL-system using DQN and [16] constructed a system using the DDPG algorithm. Both [7] and [16] jointly consider thermal comfort as well as energy cost minimization by using a similar reward function to

$$r_t = -cost(s_t, a_t) - |T_t^{in} - T_{target}|, \tag{3.2}$$

for each zone within the building that the agent is controlling. The cost function is defined for each zone in the building and is composed of both the electricity cost and the air flow rate [16] given the action a_t taken in state s_t.

DRL-HVAC Results

This section intends to highlight the different results achieved in many works where DRL was implemented on HVAC systems.

As mentioned in section 3.1.2, Wei et al. [16] included the electricity cost of taking a specific action in a zone. This in turn allowed them to evaluate how much, in terms of American dollars, DRL systems were able to decrease the energy cost compared with their rule-based baseline control strategy and a basic model-free Q-learning implementation. In comparison with the baseline control method, DRL-HVAC systems managed to reduce the total energy cost by 23.3-71.2% depending on the number of zones that are being controlled and the area of control [16].

Le Coz et al. [2] compares their baseline method with the traditional proportional–integral–derivative (PID) controller and two different RL agents on metrics such as Mean Absolute Error (MAE) from the target temperature, standard deviation, energy gain in %, and how much CO_2 has been saved. The table can be seen in 3.1. The difference between the two DRL agents is that the first agent is trained with a standard DRL strategy while the second agent uses DRL to finetune the baseline control strategy proposed by [2]. Furthermore, RL Agent 2 showed dominance in all categories as seen from table 3.1. However, there are minimal differences between DRL Agent 1 and 2. Therefore, if one were not to implement a baseline control strategy, utilizing the standard algorithms of RL still yields better results than the average control method.

Given the results by [16] and [2], using DRL for building control; can decrease energy cost significantly. It can decrease carbon footprint and is consistent with keeping comfort temperature within the apartments of the buildings, thus, improving energy efficiency overall.

	MAE T_{in} (°C)	**std** T_{in} (°C)	**Energy gain** (%)	**CO_2 saved** (g/m^2)
Baseline	0.599	0.755	0	0
PID	0.584	0.742	0.95	215
Agent 1	0.549	0.699	2.15	486
Agent 2	**0.545**	**0.692**	**2.19**	**495**

Table 3.1: Results from Le Coz et al. [2] regarding optimal district heating control in China.

3.2 Attacks on Reinforcement Learning Policies

In this section we briefly describe adversarial attacks. Adversarial attacks were first brought into attention in 2014 by Szegedy et al. [21] displaying how deep neural networks used to classify images can be fooled by perturbing the original input. The perturbed input is indistinguishable from the original input to the human eye, yet the deep neural networks can misclassify the input with high confidence. Furthermore, an explanation of this phenomenon and the construction of these attacks are highlighted by Goodfellow et al. [22]. The attack proposed by [22] is the Fast Gradient Sign Method (FGSM). FGSM uses the gradient of the loss function with respect to the input to maximize the prediction error while minimizing the difference (in the ℓ_∞ norm) between the perturbed input and the original input.

Since deep reinforcement learning uses deep neural networks, this would suggest that the policies learnt by the networks are also subject to adversarial attacks, as in the deep learning setting. These policies can be attacked, and performance can be heavily affected, as shown by Huang et al. [13] by applying the FGSM attack tailored for reinforcement learning policies. Moreover, an optimized version of the FGSM attack, regarded as Optimized Gradient Attacks in this thesis, was proposed by Pattanaik et al. [23] that does more harm to the policy than FGSM, though the idea is similar. Finally, a paper published by Russo et al. [24] further shows that one can craft optimal attacks against any Markov Decision Process (MDP). This is done by having a deep reinforcement learning adversary (agent) whose objective is to find the optimal perturbation for the control agent in order to minimize its reward, which in turn maximizes the reward for the adversary. It is important to note that the optimal attack method does not require access to the control agent's system and only needs to be able to observe the variables from which the control agent builds

her rewards.

In this section, an investigation of the three different attacks is presented, specifically how they work in theory and implemented in practice. The last part of this chapter is an example showcasing the performance of the attacks on the CartPole environment provided by OpenAI's gym library in Python3 [10]. It is important to note that the action space of the CartPole environment is discrete; hence, the algorithm used for both the control agent and the adversary is the DQN algorithm mentioned in section 2.3.1.

3.2.1 Gradient Based Attacks (White box attacks)

In a deep learning setting, Goodfellow et al. [22] shows how to efficiently generate perturbation for some input given that the adversary has access to the neural network that is being targeted, which is why FGSM is considered a white-box attack. Let θ represent the parameters of the network, x the input, y the true class label for that input, and $J(\theta, x, y)$ represents the loss function of the network; then the perturbation, according to the FGSM, becomes

$$\Delta x = \varepsilon \text{sign}(\nabla_x J(\theta, x, y)). \tag{3.3}$$

The constant $\varepsilon \geq 0$ defines the amplitude of the perturbation. This means that for higher values of ε the perturbed input is more distinguishable from the original input. Finally, the perturbed input that is fed to the neural network is

$$\hat{x} = x + \Delta x. \tag{3.4}$$

The FGSM attack has also been proven to be effective attacks against neural network policies, as showcased in [13]. However, it is not straightforward to define what is an FGSM attack in reinforcement learning, since there is no true class label. Therefore, the loss function is defined as the cross-entropy loss between the output y and the one-hot encoded representation of the highest weighted action in y [13], where y is the vector of Q-values. Hence, the loss function is defined as

$$J(s, \pi^\star) = -\sum_{i=1}^{n} p_i \ln \pi_i^\star, \tag{3.5}$$

where $\pi_i = \pi^\star(a_i|s)$ is the optimal policy for the agent, and $p_i = P(a_i)$ is the adversarial probability distribution, where P is a one-hot encoded representation of the highest weighted action in $\pi_i^\star$. Huang et al. [13] show

results from the FGSM attack against neural network policies in both a white- and black-box setting. The FGSM attack in a black-box setting computes the gradient using a separate network, B, and applies the attack to the targeted network A. In the black-box case for gradient attacks, the gradient becomes $\nabla_x J(\theta', x, y')$ where θ' is the parameters of network B and x is the same input given to both A and B. Although, [13] use the same network architecture for both agents, hence, the attack is not a pure black-box attack. Nonetheless, this is known as adversarial transferability [13].

Moreover, Pattainak et al. [23] proposed a more effective gradient-based attack where the goal is to make the agent always choose the worst action. In [23] they derive a proof for showing how to make the RL agent more effective in choosing the worst possible action (the action with the lowest Q-value) in comparison with FGSM. The main difference between FGSM and Pattanaik's Optimized Gradient Attack is that instead of placing all weight on the highest weighted action, Pattanaik instead places all the weight on the lowest weighted action in $\pi_i^\star$ [23].

3.2.2 Optimal Attacks (Black box attack)

Russo et al. [24] proposed an optimal attack policy against any MDP. The idea is to create a deep reinforcement learning agent that learns how to optimally perturb the input and that minimizes the reward for the control agent. The attack is considered black-box since this method only requires that the adversary is able to observe the rewards, and the states, of the control agent. Hence, the adversary does not need to have full access to the neural network (or the policy) used by the control agent. Russo et al. [24] defines an MDP for the adversary as:

$$\overline{\mathcal{M}} = \left\langle \mathcal{S}, \left(\overline{\mathcal{A}}_s^\varepsilon, s \in \mathcal{S}\right), \bar{P}, \bar{R} \right\rangle. \tag{3.6}$$

The state-space of the adversary is the same as the control agent; however, the rest of the variables differ. In the adversary's case, the action space $\overline{\mathcal{A}}_s^\varepsilon$ is the crafted perturbed states with a perturbation magnitude of ε. $\bar{P}$ describe the transition probabilities, which in this case is the probability of transitioning to the next state s' given that the control agent is in state s and the adversary feeds $\bar{s}$ to the control agent. Because the adversary's objective is to minimize the control agent's reward, the reward for the adversary is

$$\bar{R}(s, \bar{s}) = -\sum_a \pi(a|\bar{s}) R(s, a), \tag{3.7}$$

leading to the adversary being able to maximize its reward by minimizing the reward of the control agent.

The adversary can employ different algorithms to conduct the attack, and depending on the algorithm chosen, the implementation differs. For the examples of adversarial attacks seen in the upcoming section 3.2.3, the adversary is implemented using the DQN algorithm.

3.2.3 Examples of Adversarial Attacks

As part of the work of this thesis, this section intends to showcase examples for the readers of the attacks mentioned in the previous sections. Two agents, denoted as agent A and B, are trained and used to control the CartPole-v1 environment provided by OpenAI's Gym library for Python [10]. The attacks evaluated are the FGSM attack [13], the Optimized GA by Pattanaik [23], and optimal attacks by Russo [24]. The CartPole-v1 environment has a maximum of 500 steps in the environment for each episode and for each step a reward of 1 is awarded to the agent. The longer the agent is able to balance the pole on the cart, the more reward it will receive. A state in the CartPole environment consists of four variables; the position of the cart, the velocity of the cart, the angle of the pole and the velocity of the pole. Based on a state the agent can choose two actions; to move the cart one unit to the left or to the right.

The two control agents trained for this example were implemented using the DQN algorithm (see section 2.3.1). The average reward of the control agents A and B during evaluation was 500 on average, evaluated over 250 episodes; hence, the agents solve the environment optimally.

Furthermore, for the optimal attacks, two adversaries are trained and tailored for each respective control agent. The adversaries for the optimal black-box attacks were also trained using DQN. The settings for the adversary and the control agent, including the neural network architecture and hyperparameters can be found in the Appendix.

The perturbation used for all examples, both the gradient attacks and optimal attacks, perturbs the input with respect to each state component instead of the whole state itself because all the components have different value domains. Hence, instead of using equation 3.4, the perturbed input $\hat{x}$ is computed through

$$\hat{x} = x \odot (1 + \overline{\varepsilon}\Delta x). \tag{3.8}$$

The performance loss of Agent A and Agent B under attack can be seen in figures 3.2 and 3.3 respectively given the value of the perturbation magnitude ε which in this experiment was $0 < \varepsilon \leq 0.2$. Generally, the black-box

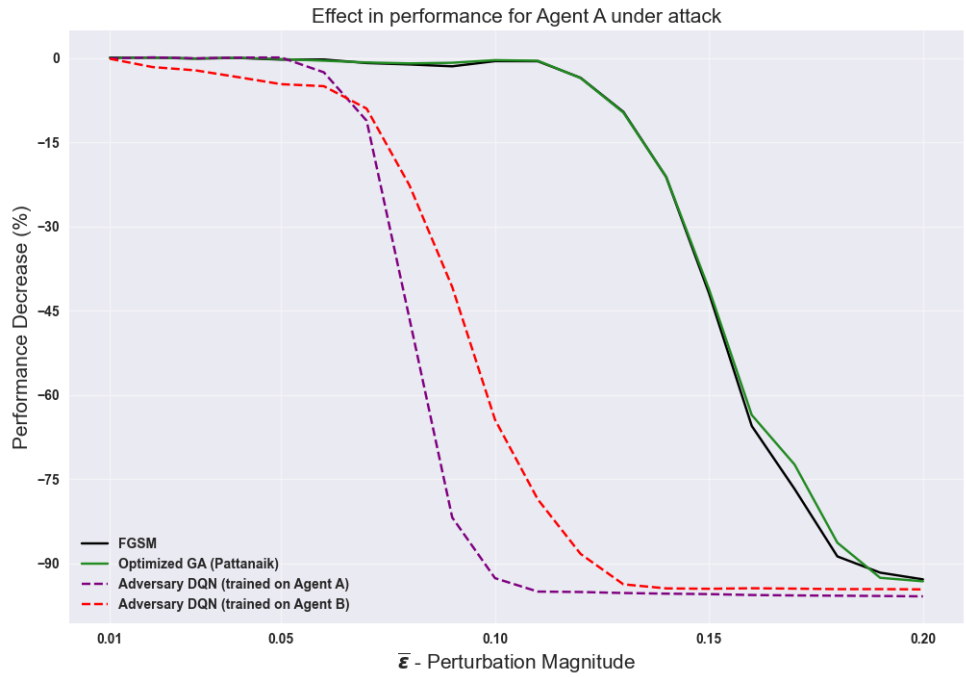

Figure 3.2: Effect in performance for Agent A under attack given the value of the perturbation constant ε.

optimal attack performs better than the white-box gradient attacks for both experiments. Although the white-box attacks utilize the gradients of the neural networks for both agents A and B, training an adversary that learns the optimal strategy to perturb the input does more damage to the agent's performance according to these results. The same conclusion is drawn by Russo [24] regarding optimal attacks. It is also important to observe that the adversary DQN trained on agent B is able to affect the performance of agent A more than the gradient attacks. Further, the robustness of agent B seen in figure 3.3 made it not possible for the adversary trained on agent A to decrease the performance, neither the gradient attacks. However, the adversary tailored specifically for attacks against agent B successfully decreases the performance for $\varepsilon \geq 0.1$. This further proves how powerful using an adversary can be.

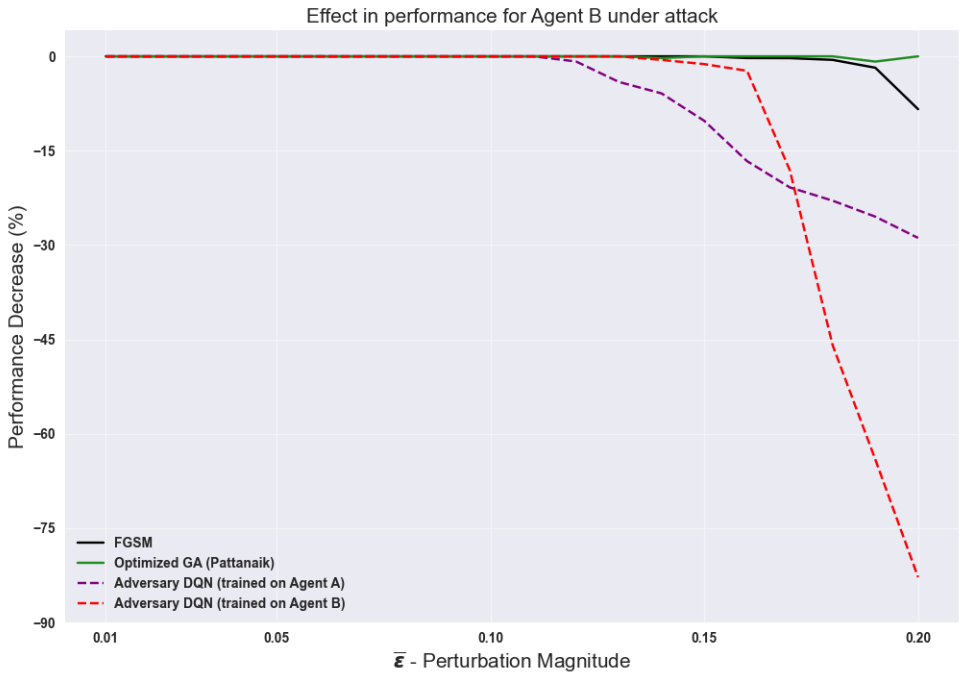

Figure 3.3: Effect in performance for Agent B under attack given the value of the perturbation constant ε.

Chapter 4

Methods

This chapter covers the methods taken to conduct the experiments for both building control and adversarial attacks. The first section includes the description of building control, specifically the building architecture, the heating architecture, MDP for temperature control in buildings, system design, and the training procedure for the control agent. The adversarial attack section describes how the adversaries were built, including the perturbation technique and the training procedure.

4.1 Building Control

4.1.1 Building Description

We simulated the building environment by using the proprietary software IDA ICE, by EQUA [25]. The system simulate over the months of January and February and assumes the the building to be in Stockholm, Sweden. Furthermore, the system can simulate occupants coming in and out of the zone in the apartment. However, this is not part of the data sent by IDA; hence, the agent experiences partial observability if trained on the building model with occupants.

The data generated are the following thermal behaviours

- *External Temperature.* The current outside temperature in Stockholm, measured in celsius °C.
- *Normal Irradiation.* Amount of solar radiation received per unit area of the building, measured in watt per square metre W/m^2.

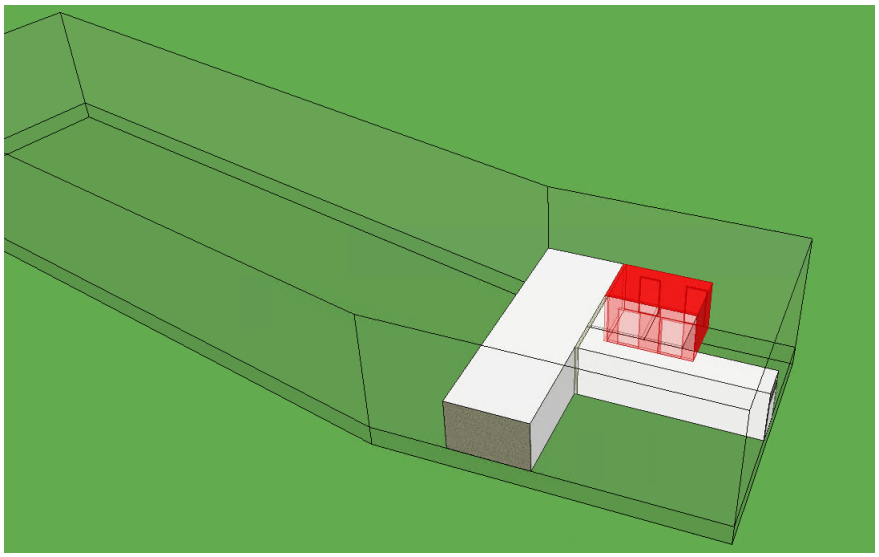

Figure 4.1: Schematic of the apartment inside the building in the IDA ICE environment.

- *Air Temperature*. The current temperature in the given zone of the building, measured in celsius °C.
- *CO_2*. The CO_2 level in the given zone of the building, measured in parts per million (ppm). Safe CO_2 levels are between 400-1000 ppm according to Wisconsin Department of Health Services [26].
- *Relative Humidity*. The indoor humidity in percentage.
- *Air Supply Temperature*. The temperature of the air produced by the Air Handling Unit measured in celsius °C. The architecture of the AHU is described in section 4.1.2.

The schematic of the building in IDA is shown in figure 4.1. This is a simple building with one apartment. The reasoning why not a complex model with multiple apartments was used, is discussed in section 6.2. The red zone in the image denotes the apartment's living room, and this is the zone that the agents control in the experiments. Finally, for training, we had to use the DQN because of the limitations in IDA (see section 6.2).

4.1.2 Heating Architecture

The heating architecture can be seen in figure 4.2, which is similar to the image shown in section 3.1.1. However, the difference lies in the way the

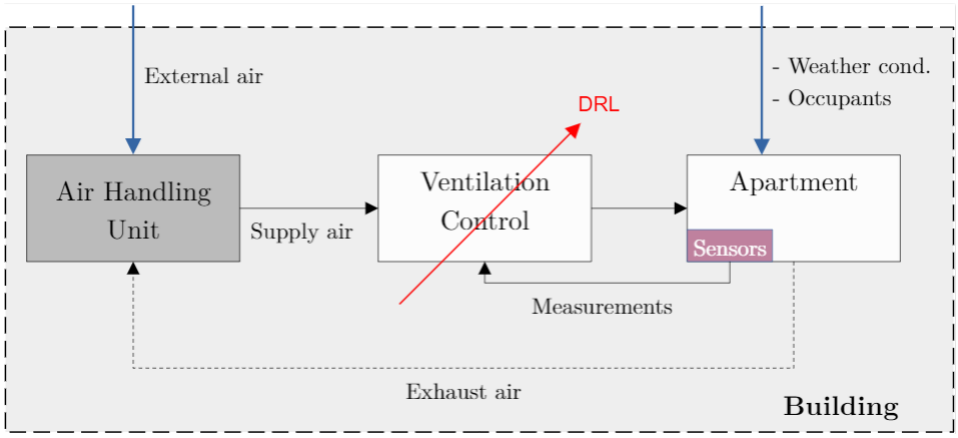

Figure 4.2: HVAC architecture model controlled using DRL.

ventilation system is controlled. The ventilation system is normally controlled with a proportional–integral–derivative controller (PID controller) or Virtual Reference Feedback Tuning (VRFT). However, in this case, DRL is utilized to keep the temperature inside the zone of the apartment at the desired comfort range instead.

The AHU controls the airflow that propagates through the building. In each apartment, a valve allows the flow of warm air to propagate further inside the rooms. The valve can either be fully opened, fully closed, or anywhere in-between, and a number between 0 and 1 denotes how much the valve is open. For instance, the state of the valve is fully opened if the action sent in is 1 and fully closed if the action is 0. Controlling the valve is, hence, a continuous action space domain and is further explained in section 4.1.3.

4.1.3 MDP Formulation

If we consider the building problem as an MDP with the following variables

$$\mathcal{M} = \{\mathcal{S}, (\mathcal{A}_s, s \in \mathcal{S}), P, R\},$$

we can define each component of the MDP. The state-space representation of the MDP is described by the six variables that IDA sends in during the simulation of the building, which is the data sample denoted as $\mathbb{D}$. We normalize each variable received by IDA using the following formula:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \quad x \in \mathbb{D} \wedge x' \in s \tag{4.1}$$

which is inspired by Wei et al. [16]. The normalized values are put in a vector, and this vector represents the state of the system. Furthermore, this procedure results in each component of the data having a value between 0 and 1 before being fed into the network, which stabilizes training. Each component's minimum and maximum values can be found in table 4.1.

State Component	**Min. Value**	**Max. Value**
External Temperature	-30	50
Normal Irradiation	0	1500
Air Temperature	-30	50
CO_2	0	15000
Relative Humidity	0	1
Air Supply Temperature	-30	50

Table 4.1: The minimum and maximum values for each respective state component.

The action-space is a value between 0 and 1, hence, $\mathcal{A} \in [0, 1]$, which indicates the state of the valve as mentioned in section 4.1.2. Note that $\mathcal{A}$ is a continuous action space. By choosing some action $a \in \mathcal{A}$ the system transitions to a state s' and this is described by the variable P in the MDP.

Finally, the reward function R, is an essential component of the system. This tells the agent what the goal is and what is expected from it. Thus, the function must be precise and consistent with the dynamics of the system. In this thesis, the reward function is a combination of thermal comfort and CO_2 level consideration. Specifically, the temperature within the zones of the apartment should be kept at around 22 °C while keeping the CO_2 levels within a reasonable range. This can be achieved in various ways; however, it is essential to know how exactly the agent should be punished given that the CO2 levels are too high or that the temperature is deviating from the target temperature.

We propose three different reward functions and argue for the one chosen for this work. Firstly, we use the absolute reward function, similar to the one highlighted in section 3.1.2 used by different papers, and include the CO_2 level for the building at time step t

$$r_t = \begin{cases} -|T_t^{in} - T_{target}| - k|C_t^{in} - C_{target}| & \text{if } C_t^{in} > C_{target} \\ -|T_t^{in} - T_{target}| & \text{otherwise.} \end{cases} \tag{4.2}$$

The variables T_t^{in} and C_t^{in} represents the temperature and CO_2 level of the

zone in the apartment at time step t, and their corresponding target variables are denoted as T_{target} and C_{target} respectively. The reward for the CO_2 is multiplied by a factor $k \geq 0$ in order to find a balance between the reward of the temperature and the CO_2 reward, if $k = 0$ the agent does not focus on the CO_2 levels of the building and only consider the temperature. Furthermore, the agent does not get punished if the CO_2 level is less than the CO_2 target unlike the reward for the temperature. The reward distribution is shown in a contour plot in figure 4.3 for different values of T_t^{in} and C_t^{in}, given that $T_{target} = 22$ and $C_{target} = 700$. It is important to note that, for the absolute reward function, the reward is normalised for both components as they have different range of values.

Secondly, we have an exponential reward function defined as

$$r_t = \begin{cases} \exp(-(T_t^{in} - T_{target})^2) + \exp(-k(C_t^{in} - C_{target})^2) & \text{if } C_t^{in} > C_{target} \\ \exp(-(T_t^{in} - T_{target})^2) & \text{otherwise,} \end{cases} \tag{4.3}$$

and the corresponding contour plot can be seen in figure 4.4. Finally, the last reward function is logarithmic with the following equation:

$$r_t = \begin{cases} -\ln(|T_t^{in} - T_{target}| + 1) - k\ln(|C_t^{in} - C_{target}| + 1) & \text{if } C_t^{in} > C_{target} \\ -\ln(|T_t^{in} - T_{target}| + 1) & \text{otherwise.} \end{cases} \tag{4.4}$$

Comparing all three reward functions, it is clear that the logarithmic reward distribution overall has better spread according to figure 4.5 and does not deviate quickly from the target as the exponential one. The logarithmic reward function showed to be most effective during the experiments and it is the reward function officially implemented for this work.

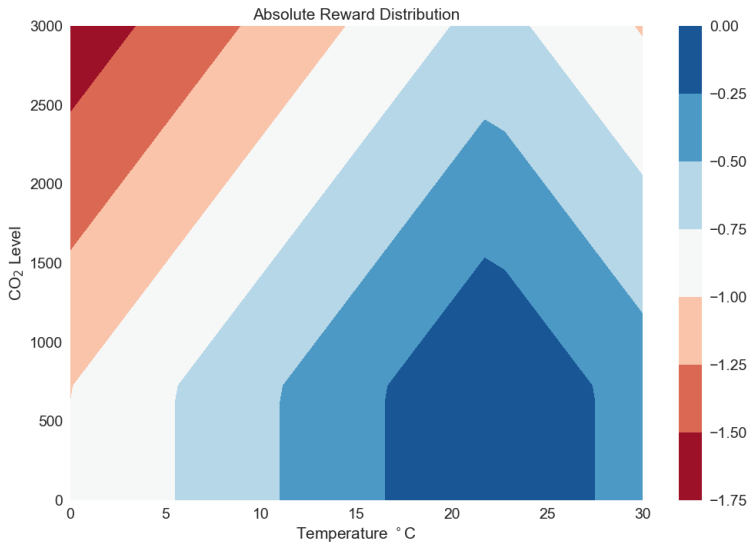

Figure 4.3: Contour plot for the absolute reward function given that the temperature target is set to 22 °C, and the CO_2 target set to 700 ppm.

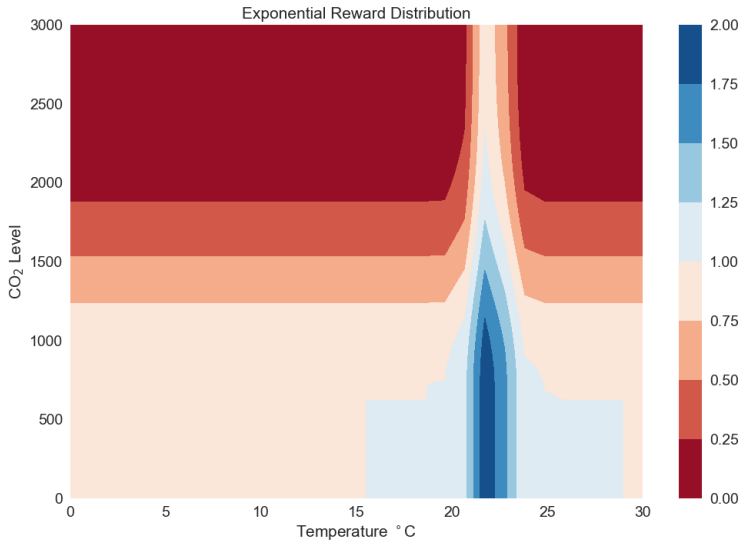

Figure 4.4: Contour plot for the exponential reward function given that the temperature target is set to 22 °C, and the CO_2 target set to 700 ppm.

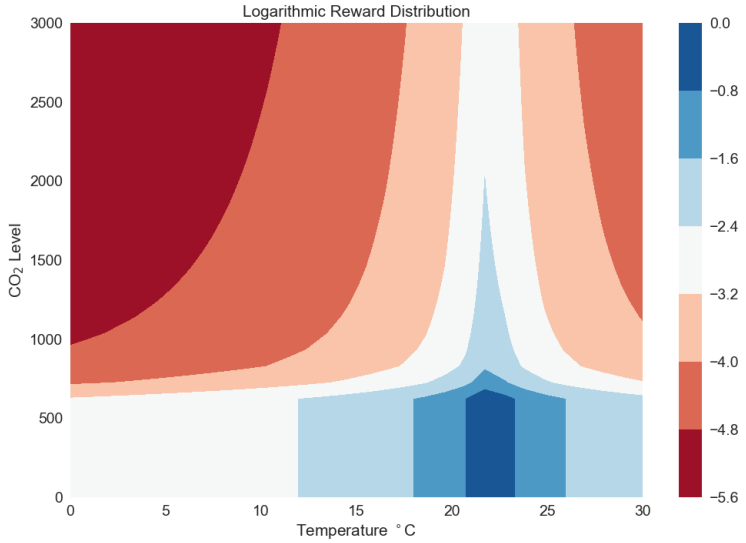

Figure 4.5: Contour plot for the logarithmic reward function given that the temperature target is set to 22 °C, and the CO_2 target set to 700 ppm.

4.1.4 DQN Training Procedure

For the control problem, we decided to make use of the episodic DQN, seen in section 2.3.1. One step in the training process environment can be seen in Algo 2. It is important to note that this training is online, meaning the agent is learning a policy during the simulation. The procedure of the algorithm is executed every S_L (sampling time) virtual seconds. Depending on the episode length T_L, which is set to 24 hours, the episode's status must be derived to find out if the state is terminal or not. When implementing the agent using DQN, the action space need to be discretized as mentioned in section 3.1. Let $A_{disc} \in \{0, 1, 2, .., n\}$ describe the discrete action space, and $A_{cont} \in [0, 1]$ the corresponding continuous action space. Then, the mapping between the discrete and the continuous space is

$$A_{cont} = \frac{1}{n} A_{disc}, \tag{4.5}$$

where $n + 1$ is the number of actions in the discrete action space. Further, if the data received by the simulator is the first data in the segment (a starting state), there is no a_{pre} or s_{pre}; hence, one cannot append an experience $(s_{pre}, a_{pre}, r, s_{curr})$ to the replay buffer, this is why the status codes are essential

for the training process. Moreover, line 15 to 20 is similar to the update to the network parameters as shown in Algo 1 regarding general DQN. As seen in the algorithm, the target is calculated using the Double Q-learning idea as mentioned in section 2.3.1. Furthermore, the update of the target network is done in a soft fashion rather than a hard copy every C steps as mentioned in section 2.3.1. Soft update continuously updates the target network by copying the parameters of the main network. The smoothing factor $0 \leq \rho \leq 1$ determines how large a portion of the main network's parameters should be copied over to the target network at each training step. Finally, the actions selected by the agent, specifically the output of the main network Q_θ, has to be made continuous by using equation 4.5, which is done on line 22 of Algo. 2 before returning the action to take in each zone of the building to the simulator because the actions are represented in a discrete fashion during training. Finally, the training occurs during the month of January. Two of the weeks in January are used as a warm-up of the simulator, and the other two weeks are used for the agent to train on. The policy for the agent is evaluated on the month of February, while January is used for training.

4.1.5 System Design

The flow of the system can be seen in figure 4.6. The manager in the image acts as an intermediator between the building environment and the control agent. By receiving the data $\mathbb{D}$ from the building environment, the manager converts the data into a representable state and sends it to the control agent. The control agent responds with a continuous action based on the state, generated by the main network, which the manager executes in the building environment. Concurrently, the manager adds an experience to the buffer. The control agent extracts a batch of experiences and trains the main network using the target network as long as the buffer is filled with enough experiences.

Algorithm 2 DDQN with ε-greedy exploration for Building Control

Input Simulation time t; episode length in seconds T_L; sampling time in seconds S_L; Data $\mathbb{D}$ for the given timestep; zones in the building Z; main network Q_θ; target network $Q_{\theta'}$; experience replay buffer $\mathcal{B}$; discount factor γ; smooth target update factor ρ; training batch size N; number of episodes T_E; exploration rate $\{\epsilon_k\}_{k=1}^{T_E}$
procedure
  if t mod T_L = 0 **then**
    $status \leftarrow$ Start
  else if $(t + S_L)$ mod T_L = 0 **then**
    $status \leftarrow$ Done
  else
    $status \leftarrow$ Prog
  for zone $\in Z$ **do**
    Normalise and flatten the data $\mathbb{D}$, receive s_{curr}
    Take ϵ-greedy action a_{curr} ▷ w.r.t. $Q_\theta(s_{curr}, a)$
    if $status \neq$ Start **then**
      $r \leftarrow reward(\mathbb{D})$
      Append $(s_{pre}, a_{pre}, r, s_{curr}, d)$ to $\mathcal{B}$
    if $size(\mathcal{B}) \geq N$ **then**
      Randomly sample a batch $\mathcal{D} \sim \mathcal{B}$ of size N
      For each experience (s_i, a_i, r_i, s_{i+1}) in $\mathcal{D}$

$$y_i = \begin{cases} r_i + \gamma \max_a Q_{\theta'}\left(s_{i+1}, \arg\max_a Q_\theta(s_{i+1}, a)\right) & \text{if } s_{i+1} \text{ is not terminal} \\ r_i & \text{otherwise} \end{cases}$$

      Update θ by minimizing the MSE loss $\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left(y_i - Q_\theta\left(s_i, a_i\right)\right)^2$
      $\theta' \leftarrow \rho\theta' + (1-\rho)\theta$
    $a_{pre} \leftarrow a_{curr}$
    $s_{pre} \leftarrow s_{curr}$
  return to_continuous$(\forall a_{curr} \in Z)$

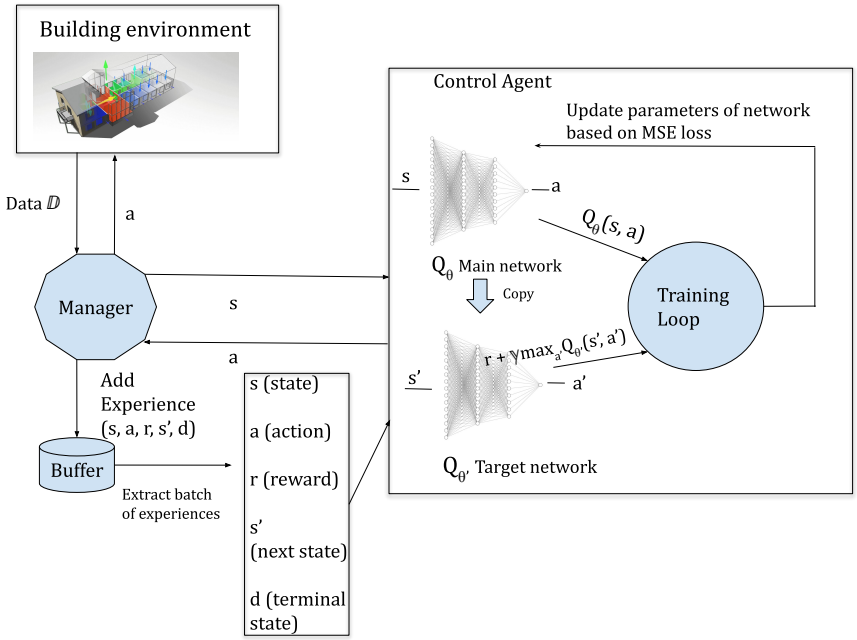

Figure 4.6: System control flow with deep reinforcement learning.

4.1.6 Exploration

Exploration in a continuous domain is a difficult but essential task. Exploration is required for the agent to try a wide variety of actions, leading to a better understanding of the system's behaviour and increased generalization of the agent. In this work, the following exploration methods, and a combination of these methods, are used

- **Uniform**. This type of strategy chooses a completely random action from A_{disc}. Used in the typical episodic DQN algorithm with ε-greedy seen in 2.3.1.

- **Pseudorandom binary sequence**. Pseudo Random Binary Signals (PRBS) are binary signals that randomly switch from 0 to 1. PRBS has a repetition period and is deterministic in this work. The repetition period is denoted with the variable T_{PRBS} and decreases during the learning period of the agents. The signal switches output choice every $\frac{T_{PRBS}}{2}$ calls to PRBS signal function. This method is chosen in order for the agent to test the maximum and minimum action values in the continuous

domain. This, in turn, makes the agent understand the effects these actions have on the environment.

- **Gaussian Noise**. This strategy adds zero-mean Gaussian noise to the action chosen by the network. This is is inspired from the general Deep Deterministic Policy Gradient algorithm suited for continuous action-space environments [27].

- **Greedy**. This strategy chooses the greedy action according to the Q-values learnt by the network.

The exploration procedure used in this work uses a convex combination of these strategies to perform exploration, similarly to the ε-greedy strategy for multiple choices of exploration strategies. If we denote by N_s as the number of strategies, then the greedy strategy is chosen with probability $\varepsilon/N_s + (1-\varepsilon)$ and the additional strategies are chosen with probability ε/N_s respectively, where $0 \leq \varepsilon \leq 1$. As mentioned in section 2.3.1, ε decreases over time, either linearly or exponentially, which in turn makes the probability distribution favor the greedy strategy on the long run. A simple algorithm of this exploration procedure is shown in Algo. 3.

Algorithm 3 Exploration Technique using ε-greedy

Input control agent's main network Q_θ; exploration rate $\{\epsilon_k\}_{k=1}^{T_E}$; discrete step D; thermal input state s; exploration strategies E; current episode ep: gaussian std σ; T_{PRBS}
procedure
    strategy $\leftarrow \begin{cases} e \in E & \text{w.p. } \varepsilon/|E| \\ Greedy & \text{w.p. } (1-\varepsilon) \end{cases}$
    if strategy is $Greedy$ or $Gaussian$ **then**
        $a \leftarrow D \cdot \arg\max_a Q_\theta(s_{curr}, a)$
        if strategy is $Gaussian$ **then**
            $a \leftarrow a + Z$, where $Z \in \mathcal{N}(0, \sigma)$
    else if strategy is $Uniform$ **then**
        $a \leftarrow D \cdot Z$, where $Z \sim \mathcal{U}(\mathcal{A}_s)$
    else if strategy is $PRBS$ **then**
        $a \leftarrow$ *prbs_signal*(ep, T_{PRBS})
return a

4.1.7 Evaluation

Since January month is used for training, February is used for evaluating the agents' performance. In comparison with the training procedure, agents are active during the warm-up time during the evaluation. Since the warm-up period is used just to start the simulation, the agent's performance is evaluated only after warm-up. As a result of many papers already being published on this topic, the idea is not to explore which hyperparameters are considered the best, but rather the exploration of the agents in order to try a variety of actions and find the many possible ways to achieve the desired results. Therefore, the results consist of evaluating different agents with different exploration strategies. The measurements that are evaluated in order to determine the capabilities of the agent are

- The average temperature is close to the desired target temperature.
- The CO_2 level in the building should be within a reasonable level.
- The agent should consider a variety of discrete actions. As choosing too few actions and not changing the actions frequently may suggest that the agent did not learn anything significant.

Finally, during evaluation, there is no source of randomness from the simulations. Instead, we use different exploration methods such as noise during training. Therefore, only one simulation for each agent type is executed and evaluated.

4.2 Attack

4.2.1 Perturbation technique

In this section we describe the attack procedure. In order to find a perturbation technique using the optimal attack method, due to the constraints in using IDA, as mentioned in section 6.2, we had to discretize the action space of the attacker as well. Specifically, for each state component in s we admit two possible perturbations, $x(1+\overline{\varepsilon}\Delta x)$ or $x(1-\overline{\varepsilon}\Delta x)$ where Δx is defined as the perturbation pattern, x is the input, and $\overline{\varepsilon}$ is the perturbation constant, not to be confused with the ε of the previous section. A perturbation pattern is, similar to the sign method in FGSM, a vector in the same length as the state where each element describe the direction of the perturbation of the corresponding

state component. Each element can either be -1 or 1, thus, there are 2^n possible actions for the adversary, where n is the number of state components to perturb.

Due to the action-space being large if one were to perturb all state components, only three out of the possible six state components are going to be perturbed. This makes the agent more efficient at finding the correct perturbation patterns given the state as the action space is now reduced. The components that are perturbed are the ones connected to the temperature of the building, which include, *External Temperature*, *Air Temperature*, and *Air Supply Temperature*. This creates $2^3 = 8$ different perturbation patterns as seen below

- $\Delta x_0 = \begin{bmatrix} -1 & 0 & -1 & 0 & 0 & -1 \end{bmatrix}$
- $\Delta x_1 = \begin{bmatrix} -1 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$
- $\Delta x_2 = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$
- $\Delta x_3 = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$
- $\Delta x_4 = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & -1 \end{bmatrix}$
- $\Delta x_5 = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$
- $\Delta x_6 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$
- $\Delta x_7 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$

Note that $x =$ [*External temperature*, *Normal Irradiation*, *Air Temperature*, *CO_2*, *Relative Humidity*, *Air Supply Temperature*]. Moreover, the adversaries do not focus on perturbing the other state components, and therefore, the corresponding element in the perturbation pattern is set to 0.

Based on the action that the attack agent chooses, the perturbed input that is fed into the control agent is

$$\hat{x} = x \odot (1 + \bar{\varepsilon}\Delta x_a), \quad a \in \overline{\mathcal{A}}^{\bar{\varepsilon}}. \tag{4.6}$$

4.2.2 Training Procedure

The training procedure of the attack agent can be seen in Algo. 4. The procedure is similar to the one used for the control agent, however, with a few modifications. Firstly, since the adversary is doing the training in this case, and the control agent's models are already pre-trained, the adversary follows the ε-greedy policy, and the control agent always chooses the greediest action.

This can be seen on line 11 and 13, respectively, in Algo. 4. It is important to notice that the adversary chooses its action based on the actual state input, and the control agent chooses an action based on the state that the adversary has perturbed. Furthermore, on line 15 of the algorithm, the reward is being multiplied by -1 because the reward of the adversary is the opposite of what the control agent should receive as a reward. This also makes the adversary's goal to minimize the reward of the control agent. Moreover, the tuple that represents the experience is a bit different, as seen on line 17. The experience consists of the previous state, the adversary's action, the adversary's reward and the current state. Once trained on, the experiences makes the adversary better at finding perturbation patterns that yield the most reward, which in turn minimizes the reward of the control agent. Finally, line 18 to 22 of the algorithm is the same training process as the control agent. However, the definition of the variables, such as the actions taken, and the rewards (which are overlined in the algorithm), are different, as previously explained.

4.2.3 Evaluation

The evaluation of the adversaries occurs in almost the same fashion as the example of adversarial attacks shown in section 3.2.3, however, with a few changes. It is important to note that, during simulation, the first two weeks of February are used for the control agents to warm-up and the attack starts once the warm-up is finished. The following steps are taken in order to evaluate the attacks

- Three different control agents are created with different hyperparameters, network architectures and exploration strategies.
- For each created agent, an adversary is created and tailored for that agent, i.e., trained against that agent.
- Additionally, we compare the optimal attack to FGSM and a random adversary.Each agent is attacked by an adversary and FGSM separately for perturbation magnitudes between 0 and 0.2.
- Finally, we measure the performance decrease and the temperature difference of each agent being under attack with the given perturbation magnitude.

Algorithm 4 DDQN Adversary against Building Control Agent

Input Simulation time t; episode length in seconds T_L; sampling time in seconds S_L; Data $\mathbb{D}$ for the given timestep; zones in the building Z; control agent's main network Q_θ; experience replay buffer $\mathcal{B}$; discount factor γ; smooth target update factor ρ; training batch size N; number of episodes T_E; exploration rate $\{\epsilon_k\}_{k=1}^{T_E}$; perturbation patterns Δx; perturbation constant $\overline{\varepsilon}$; adversary main network Q_φ; adversary target network $Q_{\varphi'}$

procedure

  if t mod T_L = 0 **then**

    $status \leftarrow$ Start

  else if $(t + S_L)$ mod T_L = 0 **then**

    $status \leftarrow$ Done

  else

    $status \leftarrow$ Prog

  for zone $\in Z$ **do**

    Normalise and flatten the data $\mathbb{D}$, receive s_{curr}

    Adversary takes ϵ-greedy action $\overline{a}_{curr}$ ▷ w.r.t. $Q_\varphi(s_{curr}, \overline{a})$

    $\overline{s}_{curr} \leftarrow s_{curr} * (1 + \overline{\varepsilon} * \Delta x_{\overline{a}_{curr}})$

    Control agent takes greedy action a_{curr} ▷ w.r.t. $Q_\theta(\overline{s}_{curr}, a)$

    if $status \neq$ Start **then**

      $\overline{r} \leftarrow reward(\mathbb{D}) * -1$

      Append $(s_{pre}, \overline{a}_{pre}, \overline{r}, s_{curr})$ to $\mathcal{B}$

    if $size(\mathcal{B}) \geq N$ **then**

      Randomly sample a batch $\mathcal{D} \sim \mathcal{B}$ of size N

      For each experience $(s_i, \overline{a}_i, \overline{r}_i, s_{i+1})$ in $\mathcal{D}$

$$y_i = \begin{cases} \overline{r}_i + \gamma \max_{\overline{a}} Q_{\varphi'}\left(s_{i+1}, \arg\max_{\overline{a}} Q_\varphi(s_{i+1}, \overline{a})\right) & \text{if } s_{i+1} \text{ is not terminal} \\ \overline{r}_i & \text{otherwise} \end{cases}$$

      Update φ by minimizing the MSE loss $\frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left(y_i - Q_\varphi\left(s_i, \overline{a}_i\right)\right)^2$

      $\varphi' \leftarrow \rho\varphi' + (1 - \rho)\varphi$

    $\overline{a}_{pre} \leftarrow \overline{a}_{curr}$

    $s_{pre} \leftarrow s_{curr}$

  return to_continuous$(\forall a_{curr} \in Z)$

Chapter 5

Results and Discussion

In this chapter, we present the results for temperature control, and adversarial attacks. First, we begin by considering the temperature control problem, and then we consider attacking this control, followed by a discussion of the results for both topics.

5.1 Temperature Control in Buildings

Exploration is an essential task in a continuous domain; hence, different agents with the same hyperparameters and network architecture are investigated. The difference lies only in the exploration strategy used. These results present a combination of the exploration methods mentioned in section 4.1.6. We trained three different networks for each different type of agent and picked the best one.

The first part of this section includes the relevant parameters and architecture used for the agents. Next, we evaluate a random agent to know how the system behaves if no control (or random control) is performed. In the later sections, we evaluate the agents using different exploration strategies. We begin by evaluating an agent that use one exploration method on top of the Greedy method during training. Then, we evaluate with two additional methods and finally one agent that uses all methods described in section 4.1.6 regarding exploration.

Moreover, a summary of the performance of each type of agent regarding temperature and CO_2 control can be found in table 6.3 and figure 6.4 respectively in the Appendix. This includes the average, maximum, minimum and standard deviation for temperature and CO_2.

The results focus mainly on temperature control rather than CO_2 control.

Because on average, the indoor CO_2 level is stable for all agents trained, regardless of their temperature control. Finally, all the agents in this section, except for one denoted with *(OCC)*, trained on a building model with no occupants in the building and all agents are evaluated on a model with occupants.

5.1.1 Training Parameters and Model Architecture

The constant parameters, such as the length of an episode, the sampling time and target temperature, with their respective description, are shown in table 5.1.

The hyperparameters can be found in table 5.2. The values of the hyperparameters are chosen based on [16]. Each agent is trained over 140 episodes with an experience replay buffer size of 15000 and a batch size of 128. The optimizer used is Adam [28] with learning rate 0.008. It is important to note that the number of discrete actions is 11, hence, the actions are only able to take actions corresponding to $A_{cont} \in 0, 0.1, 0.2, .., 1.0$. The noise (if applied) is zero-mean Gaussian noise with a standard deviation of 0.05.

The network architecture of each agent is shown in table 5.3. FC stands for Fully-Connected, and ReLU is an activation function with $max(0, x)$ as its property. Note that the number of output nodes of the network corresponds to the number of actions the agents can take. Hence, each mapped output node to a specific action describes the Q-value for that action.

Constant Parameter	**Value**	**Description**
Simulation Time	1209000 seconds (Two weeks)	Total simulation time.
Warmup Time	1209000 seconds (Two weeks)	Warmup time for the building.
Sampling time	600 seconds	The number of seconds between each data sample from IDA.
Episode Length	86400 seconds (24 hours)	The number of seconds corresponding to a full episode.
Target Temperature	22 °C	The target temperature set for all agents.
Target CO_2	700 ppm	The target CO_2 level set for all agents.
State Dimension	6	The state dimension for all thermal input states.

Table 5.1: The constant parameters used in the training environment.

Hyperparameter	**Value**
Episodes, T_E	140
Buffer size	15000
Batch size, N	128
Learning rate/Optimizer	0.008/*Adam*
Discount factor γ	0.97
Number of actions	11
T_{PRBS}	500 *(with 1% decrease every episode)*
$\varepsilon_{\max}$	0.99
$\varepsilon_{\min}$	0.05
Exploration epsilon decay	*Linear*
Noise	$\mathcal{N}(0, 0.05)$
Smoothing factor ρ	0.001
Reward Function, k	Logarithmic Reward, $k = 0.2$

Table 5.2: Hyperparameters used for the different building control agents.

Network Architecture (Building Control)	
6-dimensional State Input	
Layer 1	FC(512, ReLU)
Layer 2	FC(256, ReLU)
Layer 3	FC(128, ReLU)
Layer 4	FC(64, ReLU)
Layer 5	FC(11)

Table 5.3: Network Architecture for the different building control agents.

5.1.2 Evaluation of Uniform Agent

To get an idea of how much an agent can improve on the system dynamics, we show results of an agent trying to control the temperature in the zone of the building with uniform actions being chosen. This can be seen in figure 5.1. The evaluation shows that the average temperature is at 26.62 °C over 14 days. The agent is not able to stay within the desired temperature by choosing actions uniformly.

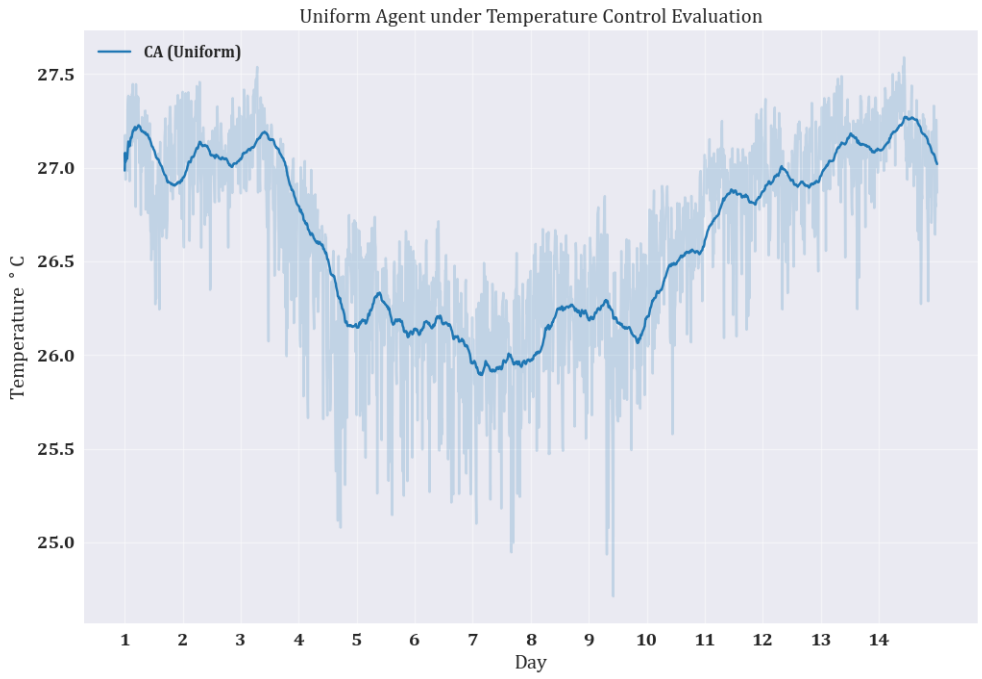

Figure 5.1: Evaluation of an agent choosing random actions. The stronger color in the plot indicates the running average.

5.1.3 Training with one additional exploration method

In figure 5.2 we can see the results for three different control agents utilizing one additional exploration method. In terms of exploration, we can see that applying zero-mean gaussian noise to the actions during training yields better exploration. This is because the agent applying noise reaches the lower temperatures instead of directly settling for the higher temperatures. However, all agents converge towards approximately 22 °C. Figure 5.3 shows the evaluation of these agents during two weeks (different from the weeks trained on). The agent applying only PRBS as an additional exploration method did converge during training, but it did not perform well during evaluation compared to the other agents. Since PRBS outputs 0 and 1, it is not sufficient enough to combine it with greedy as the actions are never noisy. Thus, not allowing the agent to explore the environment thoroughly.

CA *(Greedy, PRBS)* has an average of 24.80 °C during evaluation while the other two agents show a nearly-perfect result of 22.35 °C for CA *(Greedy,*

Uniform) and 22.40 for CA*(Greedy, Gaussian)*.

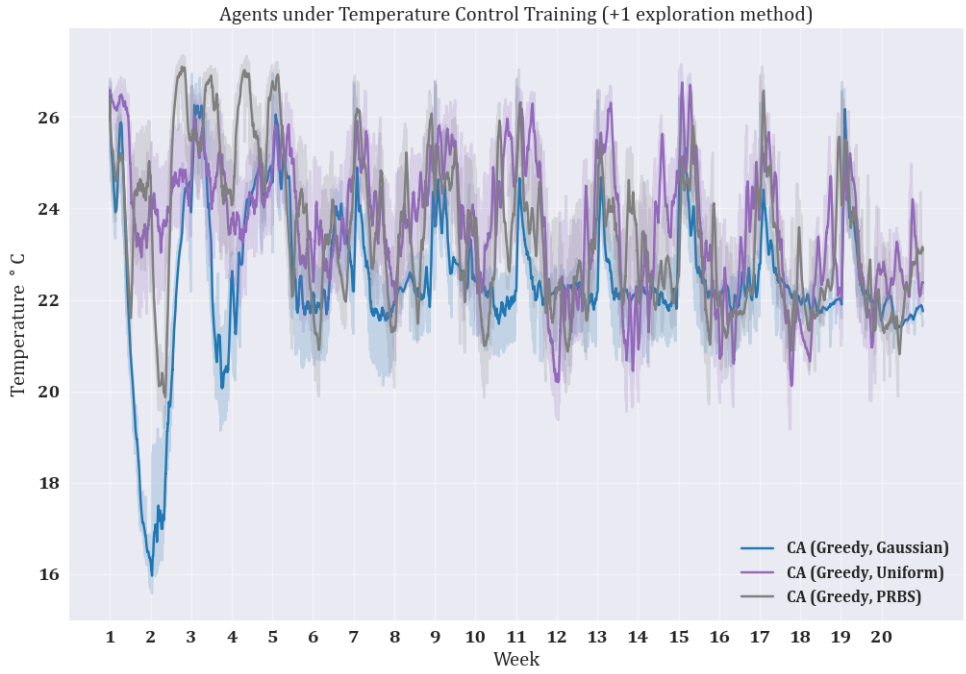

Figure 5.2: Comparison of temperature control during training between agents with one additional exploration method.

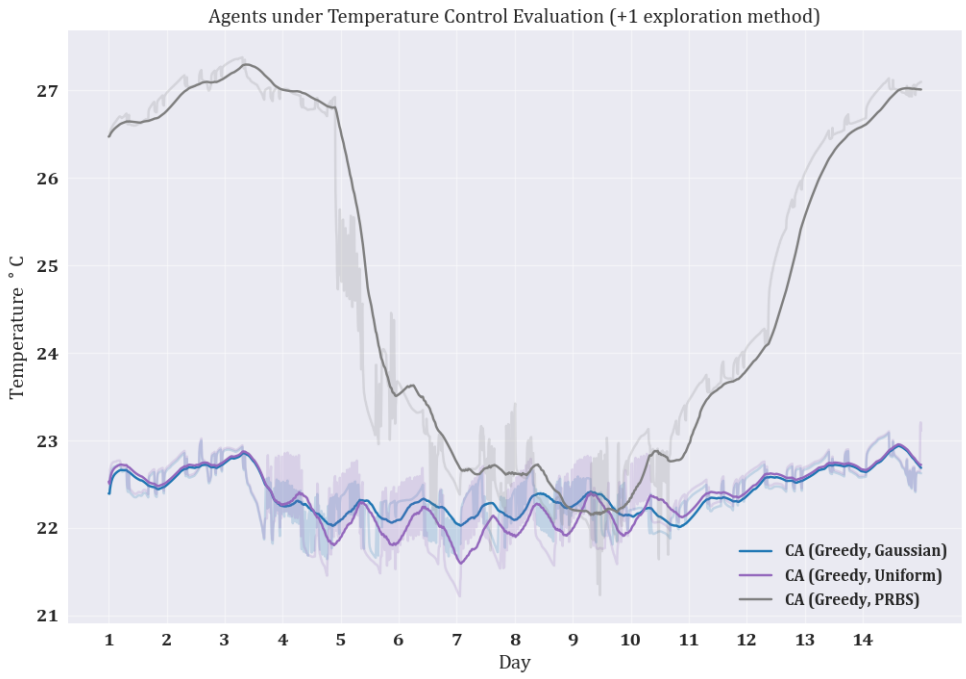

Figure 5.3: Performance of different control agents, trained with one additional exploration method. The stronger color in the plot indicates the running average.

5.1.4 Training with two additional exploration methods

Next, we evaluated using two exploration methods together. Results can be found in figure 5.4. As we can see, the agents using PRBS are not behaving the same as the previous agent using only PRBS. This is because the newly added exploration method allows the agents to try a larger variety of actions during training, and learn from these samples. Moreover, this can also be seen by the length of the spikes in figure 5.4 in comparison with figure 5.3. Furthermore, the agent utilizing both Uniform and PRBS fails to keep the temperature around the target temperature compared to the other two agents. The best combination of using two additional exploration methods seems to be Gaussian noise with PRBS. These results are further discussed in section 5.3.

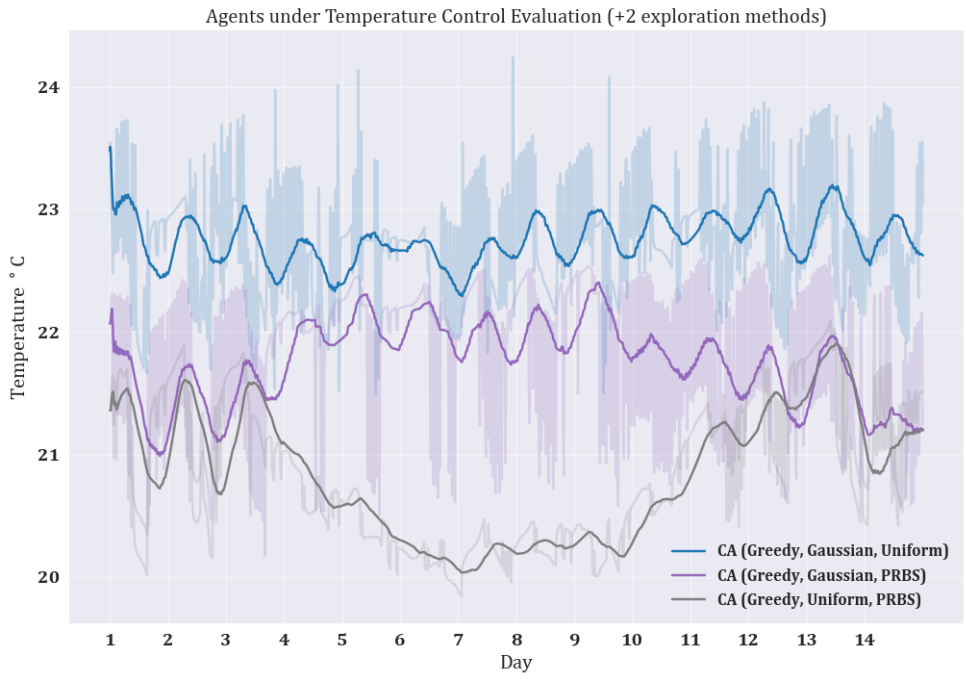

Figure 5.4: Performance of different control agents, trained with two additional exploration methods, during evaluation. The stronger color in the plot indicates the running average.

5.1.5 Training with three additional exploration methods

Finally, using all three exploration methods together with Greedy, seen in figure 5.5, one can see that the desired temperature is not reached. Using too many exploration methods actually yields worse control. This can be due to the actions being too noisy instead, and in turn, leads to the agent not finding a clear path for how to reach the desired comfort temperature.

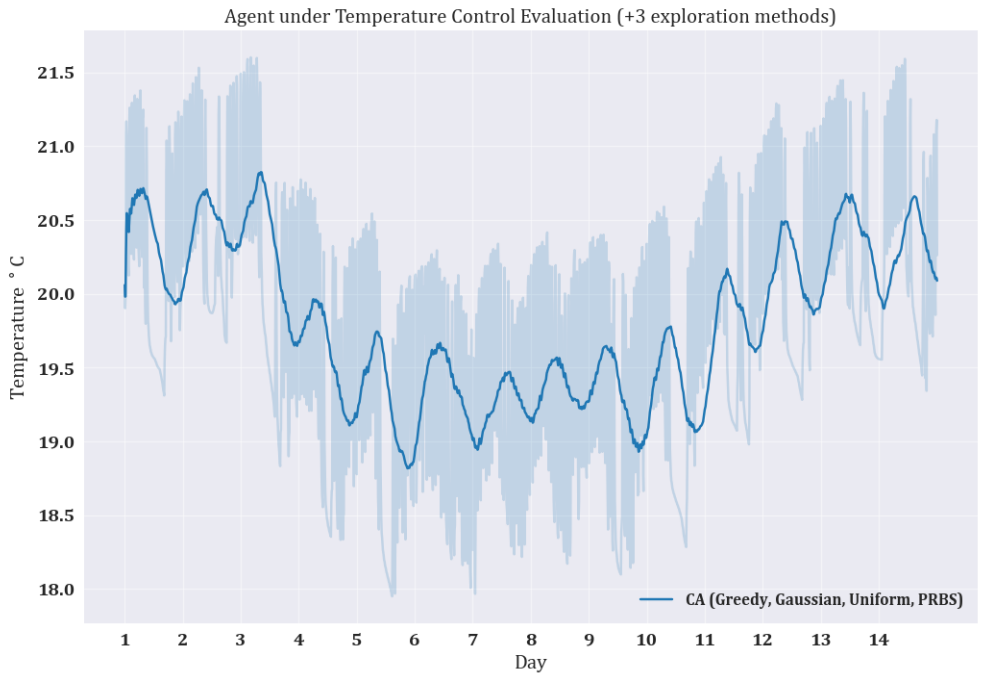

Figure 5.5: Performance of control agent utilizing all exploration methods. The stronger color in the plot indicates the running average.

5.1.6 Training with/without occupants in the building

If the agents are evaluated on an apartment with occupants, it makes more sense to train on a model with occupants. However, this shows to be less effective in this work. Figure 5.6 shows an evaluation of two agents with the same hyperparameters, network architecture and exploration strategies. The only difference between the agents is that one was trained with no occupants in the building, similar to the other agents, while the other was trained with occupants. As we can see from the figure 5.6, the agent trained with occupants in the building performed worse during evaluation than the agent trained without occupants in the building; however, not in terms of CO_2 control as seen in table 6.4. Although the difference in CO_2 control is minimal in comparison with the temperature control. The agent trained with occupants can have performed worse because of the partial observability that may have hampered the training.

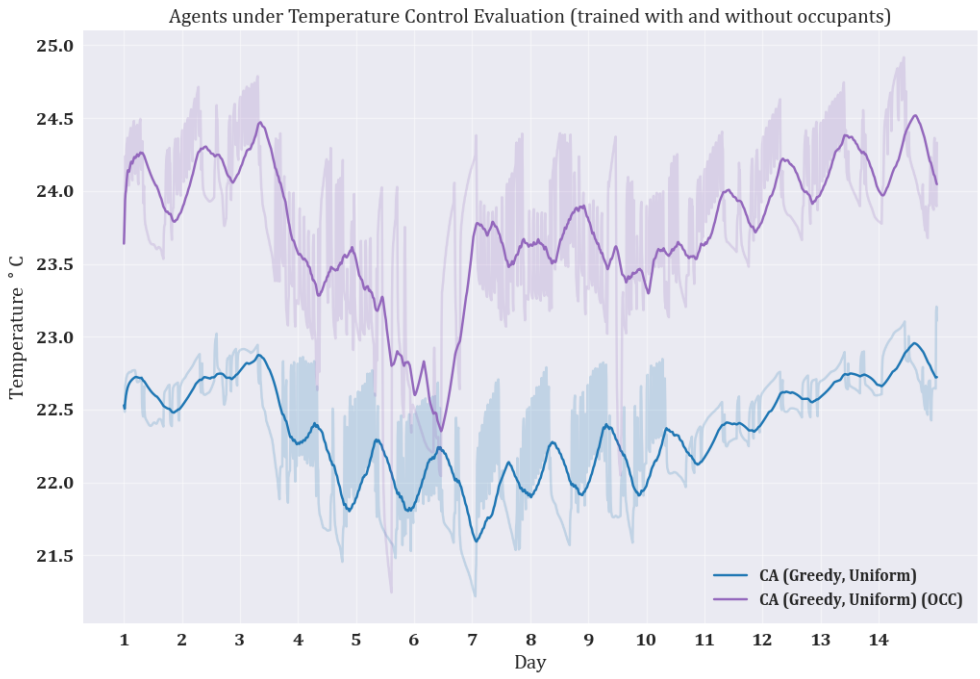

Figure 5.6: Comparison of performance during evaluation for similar agents trained with occupants in the building and with no occupants respectively. The stronger color in the plot indicates the running average.

5.1.7 Q-Value Convergence

In order to determine if the choice of action by the agent is consistent throughout training, five hundred of the most common states were tracked and sent into the network at the end of every episode. This allows for easy tracking of the Q-values outputted by the network. Figure 5.7 shows the average maximum Q-value of the five hundred states over 140 episodes. We can see that the Q-values converge at approximately episode 120 out of the possible 140. Moreover, figure 5.8 shows the mean Q-value for each action of those five hundred states, and we can see that the convergence is similar to the average maximum Q-value for every action. Hence, we see that the network used by the agent is consistent with describing the quality of each action and shows no sign of over- or underestimation for a specific action.

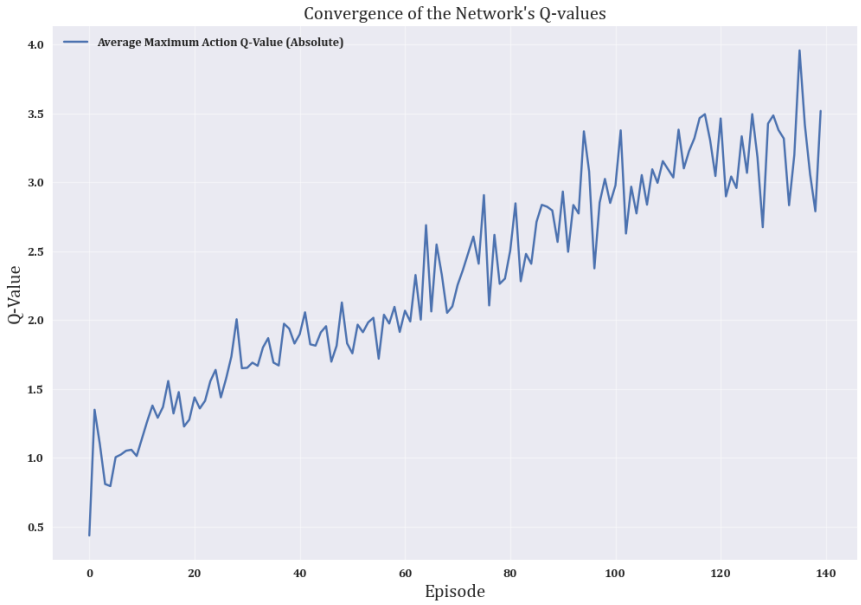

Figure 5.7: The average maximum Q-value as a result from the output of the network during training.

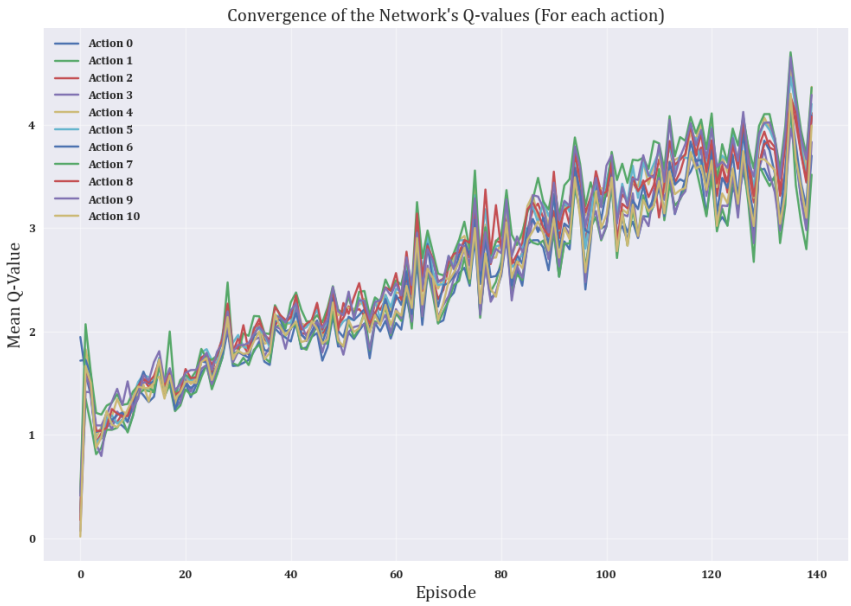

Figure 5.8: The mean Q-value for each action during training of the network.

5.2 Attack Evaluation

In this section, we are going to investigate optimal adversarial attacks against three building control agents, as well as adversarial transferability, which was mentioned in section 3.2.1 (and in [13]). Adversarial transferability measures an adversary's attacks given that the adversary was tailored (trained on) another agent than the one being attacked.

Three control agents were created: agent A, B and C. For each agent, a corresponding adversary is created and tailored for that agent. Thus, adversary A is trained on agent A, adversary B is tailored for agent B, and finally, adversary C is trained using agent C. Because of the randomness in training for an adversary, each adversary is an average of three adversaries of the same kind. This allows for better determination of the adversary's capabilities as training is affected by the random exploration.

Furthermore, the control agents differ in network architecture and hyperparameters. However, all the adversaries have the same structure. Additionally to the adversaries, we use the Fast Gradient Sign Method (FGSM) mentioned in section 3.2.1 and a random adversary. The argument for the random adversary is to showcase what would happen if the measurements were manually changed, which further proves the claim that adversarial attack pose a serious threat to systems enabled by DRL.

5.2.1 Training Parameters and Model Architecture

The hyperparameters, and network architecture for the three control agents that are attacked can be found in table 5.4 and 5.5 respectively. For the adversaries, the information regarding the hyperparameters can be found in table 5.6 and the network architecture in table 5.7. The number of output nodes of the adversary corresponds to the Q-value representation of each action, specifically each perturbation pattern mentioned in section 4.2.1. Moreover, the adversaries are evaluated on all perturbation magnitudes between 0.01 and 0.2; however, during training, they use a perturbation magnitude (constant) of 0.1 and can be seen in the hyperparameters of the adversaries.

Network Architectures (For agents under attack)			
Layers	**Agent** A	**Agent** B	**Agent** C
6-dimensional State Input			
Layer 1	FC(512, ReLU)	FC(256, ReLU)	FC(400, ReLU)
Layer 2	FC(256, ReLU)	FC(128, ReLU)	FC(200, ReLU)
Layer 3	FC(128, ReLU)	FC(64, ReLU)	FC(100, ReLU)
Layer 4	FC(64, ReLU)	FC(11)	FC(11)
Layer 5	FC(11)	-	-

Table 5.5: Network Architecture used for Agent A, B, and C respectively.

Hyperparameter	**Values for agent** A**/agent** B**/agent** C
Episodes, T_E	140/140/140
Buffer size	15K/15K/20K
Batch size, N	128/128/128
Learning rate	0.008/0.002/0.004
Discount factor γ	0.97/0.95/0.99
Number of actions	11/11/11
T_{PRBS}	500/-/750
$\varepsilon_{\max}$	0.99/0.99/0.95
$\varepsilon_{\min}$	0.05/0.05/0.05
Exploration epsilon decay	Linear for all
Noise	$\mathcal{N}(0, 0.05)$/$\mathcal{N}(0, 0.1)$/$\mathcal{N}(0, 0.08)$
Smoothing factor ρ	0.001/0.005/0.001
Exploration Strategies	(Greedy, Gaussian, PRBS)/ (Greedy, Gaussian)/ (Greedy, Gaussian, PRBS)

Table 5.4: Hyperparameters used for Agent A, B, and C respectively.

Hyperparameter	Value for all adversaries
Episodes, T_E	70
Buffer size	15000
Batch size, N	128
Learning rate	0.005
Number of actions	2^3
Discount factor γ	0.99
Perturbation Constant $\overline{\varepsilon}$	0.1
$\varepsilon_{\max}$	0.99
$\varepsilon_{\min}$	0.05
Exploration epsilon decay	Exponential
Smoothing factor ρ	0.001
Exploration Strategies	(Greedy, Uniform)

Table 5.6: Hyperparameters used for all the adversaries.

Network Architecture (Adversary)	
6-dimensional State Input	
Layer 1	FC(512, ReLU)
Layer 2	FC(256, ReLU)
Layer 3	FC(128, ReLU)
Layer 4	FC(64, ReLU)
Layer 5	FC(8)

Table 5.7: Network Architecture for all adversaries.

5.2.2 Attacks and Adversarial Transferability

Attacks against agent A

Figure 5.9 shows the drop in temperature control performance for agent A when attacked by the adversaries, including the adversary choosing a random perturbation as well as FGSM. By observing the results, we can see that adversary B har the largest impact on the the agent; however, at $\overline{\varepsilon} = 0.19$, FGSM finds a local minimum, and it is able to decreases the performance of the agent by 25%. It remains unchanged, however, for $\overline{\varepsilon} = 0.2$. Adversary C does not seem to be able to damage the agent. This might be because, the corresponding figure 6.1 seen in the appendix regarding the temperature difference, shows that the adversary is trying to decrease the temperature while the other attacks are increasing the temperature. This adversary, however, shows to be effective when attacking the agent it is tailored for.

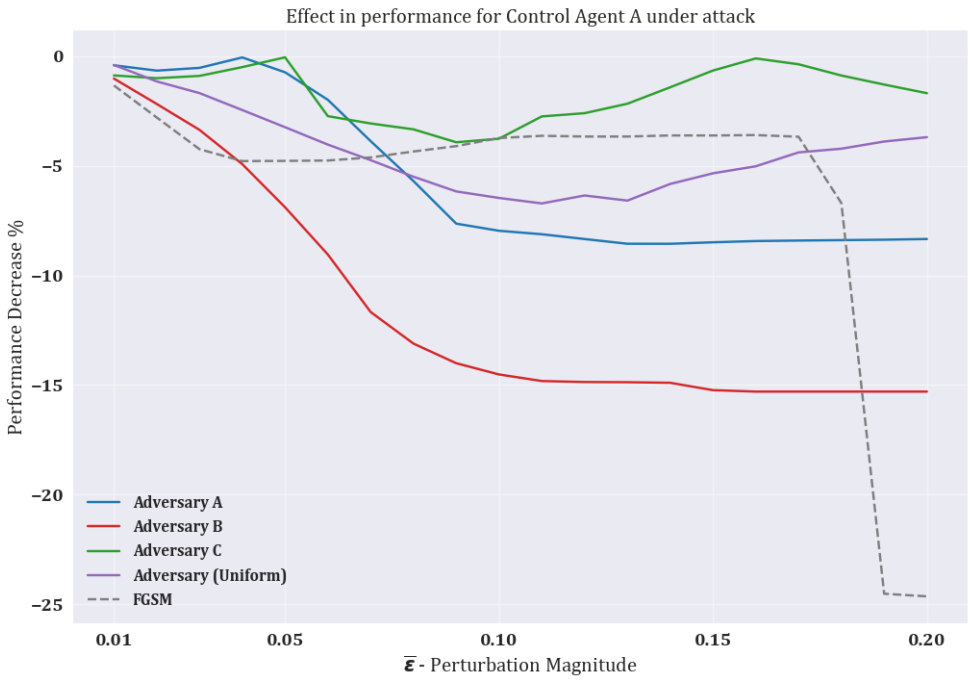

Figure 5.9: Effect in performance for Agent A under attack given the value of the perturbation magnitude $\overline{\varepsilon}$.

Attacks against agent B

In figure 5.10 is shown the performance of agent B under attack. This time, however, FGSM is in the lead for small values of epsilon for doing the most harm to the agent and loses that lead to adversary B after $\overline{\varepsilon} = 0.05$. From figure 5.10 we can deduce that adversaries A and B are quite similar in the way they choose to perturb the input, while adversary C is still not making any progress in harming neither agent A nor B as effectively as the other adversaries. The adversary choosing a random perturbation pattern is attacking the agent more efficiently than adversary C but still not as effective as adversary A, B or FGSM.

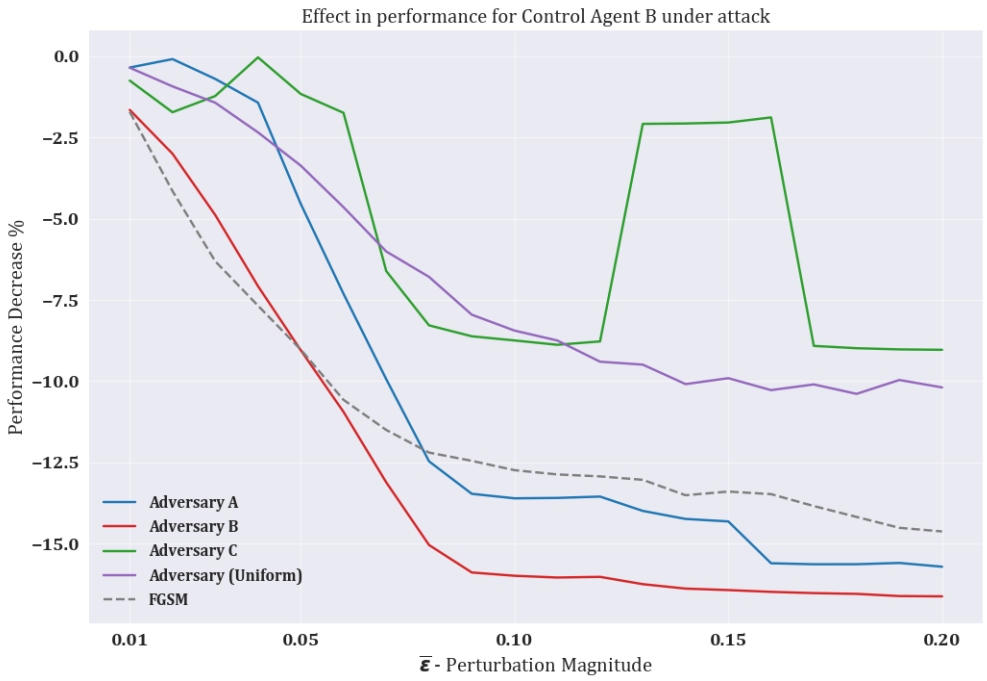

Figure 5.10: Effect in performance for Agent B under attack given the value of the perturbation magnitude $\overline{\varepsilon}$.

Attacks against agent C

Finally, figure 5.11 shows agent C under attack. As expected, adversaries A and B are have a similar behaviour. Note, though, that now adversary C which is control agent C, is able to largely affect the performance of agent C. Surprisingly, FGSM is not effective against agent C, which also goes for the adversary choosing random actions. In terms of adversarial transferability, we can deduce that adversaries A and B are quite consistent in damaging the agent regardless of which agent it is. More about this is discussed in section 5.4.

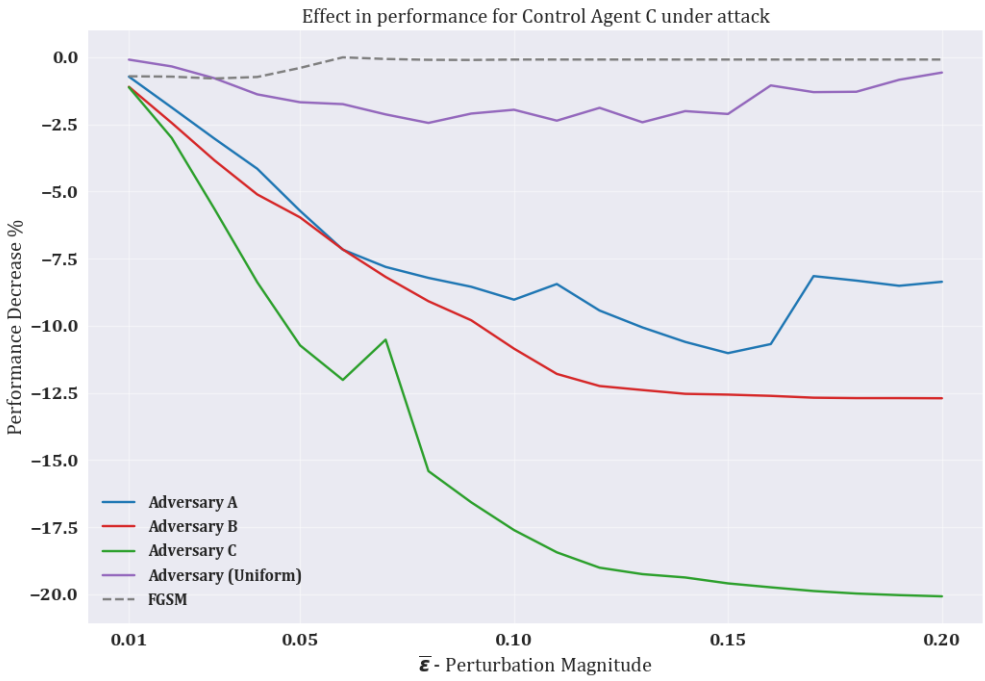

Figure 5.11: Effect in performance for Agent C under attack given the value of the perturbation magnitude $\bar{\varepsilon}$.

5.3 Discussion: Temperature Control in Buildings

Wei et al. [16] define a boundary for comfort temperature for which the agents implemented should strive for. This comfort temperature is between 19 °C, and 24 °C, which is also derived from the ASHRAE standard [29]. By observing figure 5.4, which is showing the agents evaluated after using two additional exploration methods during training, we can see that all agents are between that range. The same goes for the agents that only used one exploration method, except for CA *(Greedy, PRBS)*. Thus, even though we used a strict target temperature, we could make the agents reach a reasonable comfort temperature and stay within that boundary. However, from the results of the building control, we can see that some agents are generally not that good by observing their behaviour during evaluation. For instance, using an excessive amount of exploration methods, as seen in figure 5.5, leads to the agent being too noisy,

and unable to properly control temperature. By observing the multiple spikes, and the length of the spikes, we can deduce that the agent finds a balance of how often it switches between actions and finds a way to make to cyclically control temperature. Nevertheless, it is not achieving the desired comfort temperature.

Figure 5.6 shows the evaluation of two agents with the same hyperparameters, network architecture and exploration strategies, with the only difference being that they were trained in different environments, with and without occupants in the apartment. It is interesting to observe that the one trained without any occupants has better performance, given that the evaluation is a simulation of a building that includes occupants. This may be because the agent trained with occupants has partial observability during training, and the occupant acts as a confounder, confusing the network on what is the true temperature dynamics in the room. This factor needs to be taken into account, since it may hamper training. This argument is also supported by the fact that the plot for CA *(Greedy, Uniform) (OCC)* is shifted approximately by +1 °C in the y-axis in comparison with CA *(Greedy, Uniform)*.

Next we evaluate how often the control network changes action during evaluation. Table 6.5 in the Appendix showcases a summary of all the exploration agents regarding their frequency of changing actions and the number of actions they settled for during evaluation. The agents with one additional exploration method, shown in figure 5.3, have settled for the same amount of actions as CA *(Greedy, Gaussian, PRBS)*. However, their behaviour to the winter weeks is different. Because they only have one method, they have not explored the domain as frequently as CA *(Greedy, Gaussian, PRBS)* and are not able to understand how frequently they should choose the actions they have favored in order to achieve a more generalised solution to the temperature control problem.

Consider the table showcasing the results by Le Coz et al. [2] found in table 3.1. Many of our agents, seen in table 6.3, have similar results in temperature control regarding the standard deviation of temperature and the mean deviation from the target temperature. Depending on the type of exploration strategy the agent used, the results are better, similar or worse in comparison with [2].

5.4 Discussion: Adversarial Attacks

Russo's [24] optimal attacks shows performance loss of nearly 25% of for a DQN agent attacked by a DDPG adversary for perturbation magnitude of 0.07 in the Pong environment provided by OpenAI. By observing the results found for this thesis work, we can clearly see that this is not the case for

the DDQN adversary against a DDQN control agent in a building control environment. Although, the building system is approximated using a set of differential equations, unlike Pong, which in turn make the temperature control for buildings more difficult. Nonetheless, there is only one instance where the performance decrease of any agent reaches 25%, and that is FGSM attacking agent A for $\bar{\varepsilon} \geq 0.19$. This is, however, a high perturbation magnitude and could easily be detected as this changes almost 20% of the original input. Moreover, this might be rare, and the gradient descent step for FGSM happened to reach a good local minimum. In contrast, FGSM could not damage the performance of agent C and was worse than the random adversary. This suggests that it may not be likely for FGSM to find the optimal perturbation.

One may suggest that any perturbation would work if one were to increase or decrease the temperature manually. By observing the random adversary, this shows not to be effective as the other attack methods. The adversary choosing random actions is second-worst in every result. Therefore, training an adversary, or using FGSM, does more damage than if one were to perturb the agents' input manually.

Adversary tailored for control agent C has an interesting behaviour. It was the worst-performing adversary when attacking the other two agents that it was not trained on. When attacking control agent C, it had the most effective attack (regarding optimal attacks) in all of the results shown. It is effective for the low perturbation magnitudes as well, and it succeeded in decreasing the performance of control agent C by 12% for a minor attack magnitude of $0.06 = 6\%$, which ís double the effect. This adversary might cause a large amount of damage for the specific control agent, with the attacks going unnoticed by the detection methods because of the small magnitude.

Regarding adversarial transferability, across all control agents that were attacked, adversary B shows the best transferability and seems to be generalized in perturbing almost any DDQN agent. Adversary B was able to decrease the performance of agent A more effectively than the adversary tailored for agent A and was second-best in damaging the performance for agent C. Furthermore, we can see that adversary C does not show any transferability and can only successfully attack the agent it was tailored for. This might suggest that if an adversary is exceptionally effective at attacking its agent, it might not be generalized, thus, performs worse against other agents. Adversary A shows good transferability but cannot damage the performance of the agents as effective as adversary B, including the agent it was trained on.

Finally, based on the results shown, it is difficult to distinguish if the

hyperparameters or the choice of the network architecture of each respective agent affects the capabilities of the adversaries, or the FGSM attack.

Chapter 6

Conclusion and Future work

In this chapter, the conclusion of this thesis work is presented. The limitations encountered in this work and possible future work are also presented and discussed.

6.1 Conclusions

The goal of this thesis was to achieve satisfactory results in controlling the HVAC systems with DRL, and show that these systems can be subjected to adversarial attacks. By using the methods mentioned in chapter 4, we were able to conduct experiments to reach the goal set, and prove the claim that adversarial attacks pose a threat to these DRL systems.

When it comes to temperature control, it seems that a mixture of zero-mean Gaussian noise and pseudorandom binary sequence, together with tuned hyperparameters, an effective and explorative control agent can be implemented. However, the period of the PRB sequency needs to be fine-tuned in order to achieve a good exploration strategy. Furthermore, adding a few or several exploration methods may cause the agent to overfit or be overgeneralized, respectively, and not keep the temperature at the desired comfort range.

Regarding adversarial attacks on the building control agents, on average, implementing a black-box adversary is more reliable and powerful than the Fast Gradient Sign Method. However, there exists a probability for the FGSM to cause more damage than the average adversary. Additionally, perturbing the input by changing the measurements manually, with similar magnitude as an adversary, does not work as effectively as implementing an adversary or using FGSM. Finally, in terms of adversarial transferability, an adversary with

bad transferability may be extremely powerful against the agent it was tailored for. In contrast, an adversary with good transferability shows to be effective against other control agents and its own, yet, it may perform moderately across all agents, including the one it was tailored for.

6.2 Limitations

During the development of the thesis, we had a major limitation. The limitation is the IDA ICE integration with Python. By having a bad integration between the software and Python, the following problems were stumbled upon:

1. The simulation time is two weeks for both training and evaluation. If one were to extend the simulation time for training or evaluation, IDA would crash by the third week 100% of the time, which limited the simulation time to exactly two weeks.

2. If the code in Python3 is computationally heavy, IDA would crash. Therefore, implementations such as DDPG with a suitable network architecture would not work because of the bad integration between the software and Python. This also introduced another problem:

    (a) Adding plenty of experiences to the experience replay buffer at once would also make the software crash, so there was a limit set on the number of experiences that could be appended at once, which may have hampered the training of the agents.

Finally, the final limitation of this work is regards to the building model. The original intention was to work with a complex building model with multiple apartments. However, during the work, we noticed that this building model had issues and the actions sent to the environment had no effect. Therefore, we had to decide on using the simpler model, which only had one apartment.

6.3 Future work

There is plenty of work that could be further developed regarding building control and adversarial attacks. The following suggestions may be considered for future work:

1. Implementation of better and more effective attacks.

2. Implementation of other RL algorithms, such as DDPG or PPO, for both the adversary and the control agent. It also makes more sense to use DDPG as the control for continuous environments because discretizing the action space creates a limitation for the agents' performance.

3. Implementation of attack detection methods. Although there is no way to mitigate the effects of adversarial attacks completely, detection methods can be implemented.

4. Verify if we can train robust policies. Adversarial training may increase the robustness of the agent, and the idea is also addressed in Russo et al. [24] and Pattanaik et al. [23].

5. Adversarial attacks can be used on a real building utilizing DRL for HVAC systems to measure the effects in a real-world setting instead of a simulating one. However, the ethical issues mentioned in section 1.5 have to be taken into account as this will have effects on the building and the people inside.

References

[1] A. Russo, M. Molinari, and A. Proutiere, "Data-Driven Control and Data-Poisoning attacks in Buildings: the KTH Live-In Lab case study," *arXiv:2103.06208 [cs, eess]*, Mar. 2021, arXiv: 2103.06208. [Online]. Available: http://arxiv.org/abs/2103.06208

[2] A. Le-Coz, T. Nabil, and F. Courtot, "Towards Optimal District Heating Temperature Control in China with Deep Reinforcement Learning," *arXiv:2012.09508 [cs, eess]*, Dec. 2020, arXiv: 2012.09508. [Online]. Available: http://arxiv.org/abs/2012.09508

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013, arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783 [cs]*, Jun. 2016, arXiv: 1602.01783. [Online]. Available: http://arxiv.org/abs/1602.01783

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. doi: 10.1038/nature16961 Number: 7587 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature16961

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement

learning," *arXiv:1509.02971 [cs, stat]*, Jul. 2019, arXiv: 1509.02971. [Online]. Available: http://arxiv.org/abs/1509.02971

[7] G. Gao, J. Li, and Y. Wen, "Energy-Efficient Thermal Comfort Control in Smart Buildings via Deep Reinforcement Learning," *arXiv:1901.04693 [cs]*, Jan. 2019, arXiv: 1901.04693. [Online]. Available: http://arxiv.org/abs/1901.04693

[8] A. B. Shepherd and W. J. Batty, "Fuzzy control strategies to provide cost and energy efficient high quality indoor environments in buildings with high occupant densities," *Building Services Engineering Research and Technology*, vol. 24, no. 1, pp. 35–45, Feb. 2003. doi: 10.1191/0143624403bt059oa Publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1191/0143624403bt059oa

[9] F. Calvino, M. La Gennusa, G. Rizzo, and G. Scaccianoce, "The control of indoor thermal comfort conditions: introducing a fuzzy adaptive controller," *Energy and Buildings*, vol. 36, no. 2, pp. 97–102, Feb. 2004. doi: 10.1016/j.enbuild.2003.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378778803001312

[10] OpenAI, "Gym: A toolkit for developing and comparing reinforcement learning algorithms." [Online]. Available: https://gym.openai.com

[11] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, ser. Wiley series in probability and statistics. Hoboken, NJ: Wiley-Interscience, 2005. ISBN 978-0-471-72782-8 OCLC: 254152847.

[12] A. Proutiere, "Markov Decision Processes," Stockholm, Sweden, Nov. 2020.

[13] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial Attacks on Neural Network Policies," *arXiv:1702.02284 [cs, stat]*, Feb. 2017, arXiv: 1702.02284. [Online]. Available: http://arxiv.org/abs/1702.02284

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN 978-0-262-03924-6

[15] A. Proutiere, "Q-learning and SARSA algorithms," Stockholm, Sweden, Nov. 2020.

[16] T. Wei, Y. Wang, and Q. Zhu, "Deep Reinforcement Learning for Building HVAC Control," in *Proceedings of the 54th Annual Design Automation Conference 2017*. Austin TX USA: ACM, Jun. 2017. doi: 10.1145/3061639.3062224. ISBN 978-1-4503-4927-7 pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/3061639.3062224

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Aug. 2017, arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347

[18] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the Deep Q-Network," *arXiv:1711.07478 [cs]*, Nov. 2017, arXiv: 1711.07478. [Online]. Available: http://arxiv.org/abs/1711.07478

[19] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *arXiv:1509.06461 [cs]*, Dec. 2015, arXiv: 1509.06461. [Online]. Available: http://arxiv.org/abs/1509.06461

[20] M. C. Campi, A. Lecchini, and S. M. Savaresi, "Virtual reference feedback tuning: a direct method for the design of feedback controllers," *Automatica*, vol. 38, no. 8, pp. 1337–1346, Aug. 2002. doi: 10.1016/S0005-1098(02)00032-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109802000328

[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv:1312.6199 [cs]*, Feb. 2014, arXiv: 1312.6199. [Online]. Available: http://arxiv.org/abs/1312.6199

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv:1412.6572 [cs, stat]*, Mar. 2015, arXiv: 1412.6572. [Online]. Available: http://arxiv.org/abs/1412.6572

[23] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust Deep Reinforcement Learning with Adversarial Attacks," *arXiv:1712.03632 [cs]*, Dec. 2017, arXiv: 1712.03632. [Online]. Available: http://arxiv.org/abs/1712.03632

[24] A. Russo and A. Proutiere, "Optimal Attacks on Reinforcement Learning Policies," *arXiv:1907.13548 [cs, stat]*, Jul. 2019, arXiv: 1907.13548. [Online]. Available: http://arxiv.org/abs/1907.13548

[25] "IDA ICE - Simulation Software | EQUA." [Online]. Available: https://www.equa.se/en/ida-ice

[26] "Carbon Dioxide," Jan. 2018. [Online]. Available: https://www.dhs.wisconsin.gov/chemical/carbondioxide.htm

[27] OpenAI, "Deep Deterministic Policy Gradient — Spinning Up documentation." [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ddpg.html

[28] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[29] C. B. Ramspeck, F. E. Jakob, S. D. Kennedy, D. E. Knebel, F. H. Kohloss, M. F. McBride, M. P. Modera, C. H. Nasseri, G. Shavit, D. R. Tree, T. H. Williams, J. E. Woods, R. D. Montgomery, and K. W. Peterson, "ASHRAE STANDARDS COMMITTEE 2003-2004," p. 30.

Appendix

Network Architecture (CartPole-v1)		
Structure	**Agent A & B**	**Adversary DQN**
4-dimensional state input		
Layer 1	FC(256, ReLU)	FC(512, ReLU)
Layer 2	FC(128, ReLU)	FC(256, ReLU)
Layer 3	FC(64, ReLU)	FC(128, ReLU)
Layer 4	FC(2)	FC(16)

Table 6.1: The network architecture used for examples of adversarial attacks

Parameters (CartPole-v1)	**Agent A/Agent B**	**Adversary DQN**
Episodes	500/400	600
Buffer size	12000	30000
Batch size	64	128
Learning Rate	10^{-3}	$4 * 10^{-4}$
Discount Factor	0.97/0.99	0.97
Actions	2	16
Max exploration rate	0.99	0.99
Min exploration rate	0.05	0.05
Perturbation constant	-	0.2
Target Update Frequency	*floor(buffer size/batch size)*	

Table 6.2: The settings used for examples of adversarial attacks

Agent Type	**Avg. Indoor Temperature °C**	**Max./Min./Std. Temperature °C**
CA *(Greedy, Gaussian)*	22.40	23.08/21.64/0.29
CA *(Greedy, Uniform) (OCC)*	23.74	27.47/25.00/0.46
CA *(Greedy, Uniform)*	22.35	23.21/21.22/0.41
CA *(Greedy, PRBS)*	24.80	27.37/21.23/1.94
CA *(Greedy, Gaussian, PRBS)*	21.74	22.62/20.17/0.58
CA *(Greedy Gaussian, Uniform)*	22.75	24.24/21.49/0.46
CA *(Greedy, Uniform, PRBS)*	20.82	22.09/19.84/0.58
CA *(Greedy, Gaussian, Uniform, PRBS)*	19.80	21.6/17.95/0.83
CA *(Uniform)*	26.62	27.58/24.71/0.52

Table 6.3: Evaluation of temperature control for each control agent over the period of two weeks, with the maximum and minimum temperature reached by the agent as well as the standard deviation of the temperature control.

Agent Type	**Avg. Indoor CO_2 Level (ppm)**	**Max./Min./Std. CO_2 Level (ppm)**
CA *(Greedy, Gaussian)*	898	1259/608/238
CA *(Greedy, Uniform) (OCC)*	832	1256/608/192
CA *(Greedy, Uniform)*	886	1259/608/233
CA *(Greedy, PRBS)*	808	1970/608/248
CA *(Greedy, Gaussian, PRBS)*	986	1250/608/225
CA *(Greedy Gaussian, Uniform)*	863	1630/608/223
CA *(Greedy, Uniform, PRBS)*	1059	2634/608/355
CA *(Greedy, Gaussian, Uniform, PRBS)*	1059	1945/608/301
CA *(Uniform)*	700	1056/608/89

Table 6.4: Evaluation of CO_2 control for each control agent over the period of two weeks, with the maximum and minimum value of CO_2 as well as the standard deviation of the CO_2 control.

Agent Type	**Action Change Frequency**	**Number of Unique Actions**
CA *(Greedy, Gaussian)*	10.51	2
CA *(Greedy, Uniform) (OCC)*	6.38	4
CA *(Greedy, Uniform)*	10.62	2
CA *(Greedy, PRBS)*	18.18	5
CA *(Greedy, Gaussian, PRBS)*	5.45	2
CA *(Greedy, Gaussian, Uniform)*	3.36	5
CA *(Greedy, Uniform, PRBS)*	21.47	2
CA *(Greedy, Gaussian, Uniform, PRBS)*	4.34	3
CA *(Uniform)*	1.0	11

Table 6.5: The table describes, for each agent type, how often an action is changed during evaluation. A higher value would indicate a lower frequent of switching between the unique actions.

Avg. Temperature Attack Std. - Matrix	**Agent** A	**Agent** B	**Agent** C
Adversary A	0.64	0.81	0.9
Adversary B	0.55	0.58	0.54
Adversary C	0.63	0.69	1.05
Adversary (Uniform)	0.54	0.79	0.56
FGSM	0.63	0.77	0.62

Table 6.6: A matrix showing the average standard deviation of temperature for attacks against each control agent. This is an average over perturbation magnitude $0 < \bar{\varepsilon} \leq 0.2$.

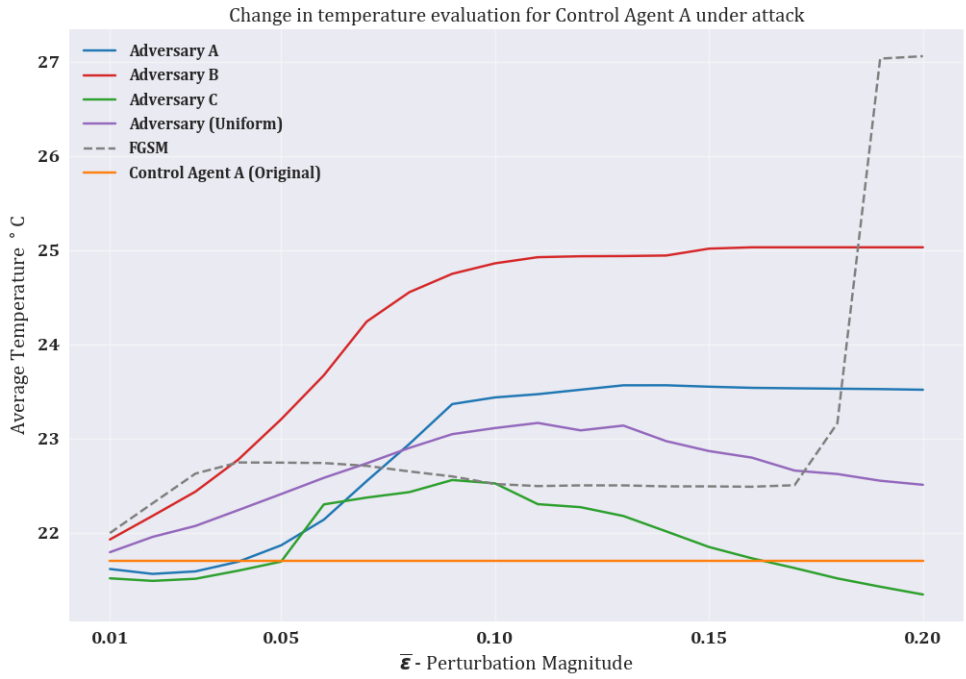

Figure 6.1: Change in temperature for Agent A under attack given the value of the perturbation magnitude $\bar{\varepsilon}$. The original average temperature of Agent A is highlighted in the figure with orange and is not dependent on the x-axis.

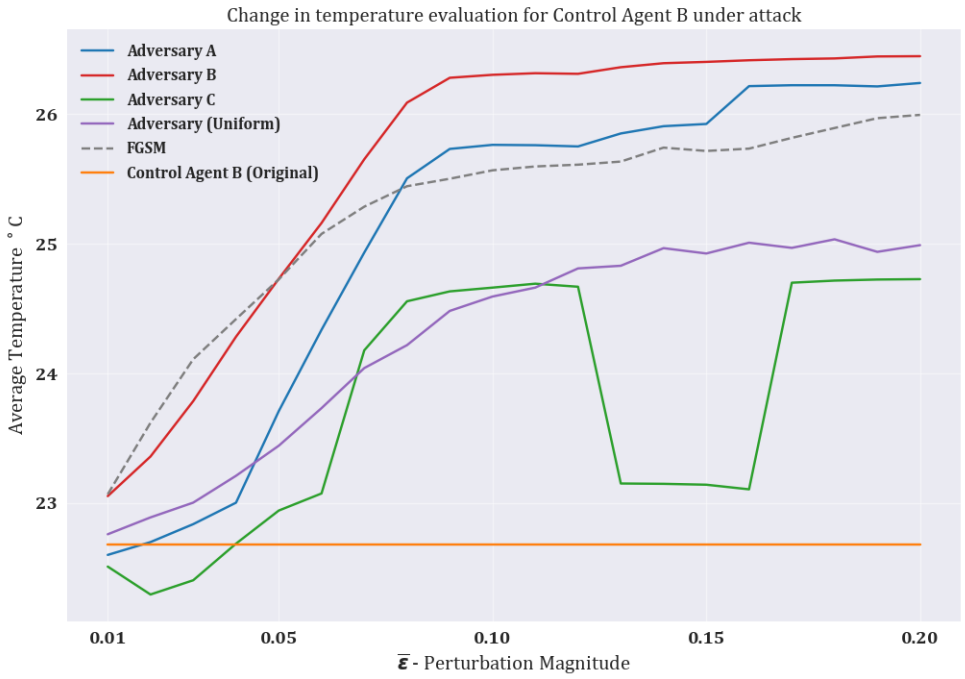

Figure 6.2: Change in temperature for Agent B under attack given the value of the perturbation magnitude $\bar{\varepsilon}$. The original average temperature of Agent B is highlighted in the figure with orange and is not dependent on the x-axis.

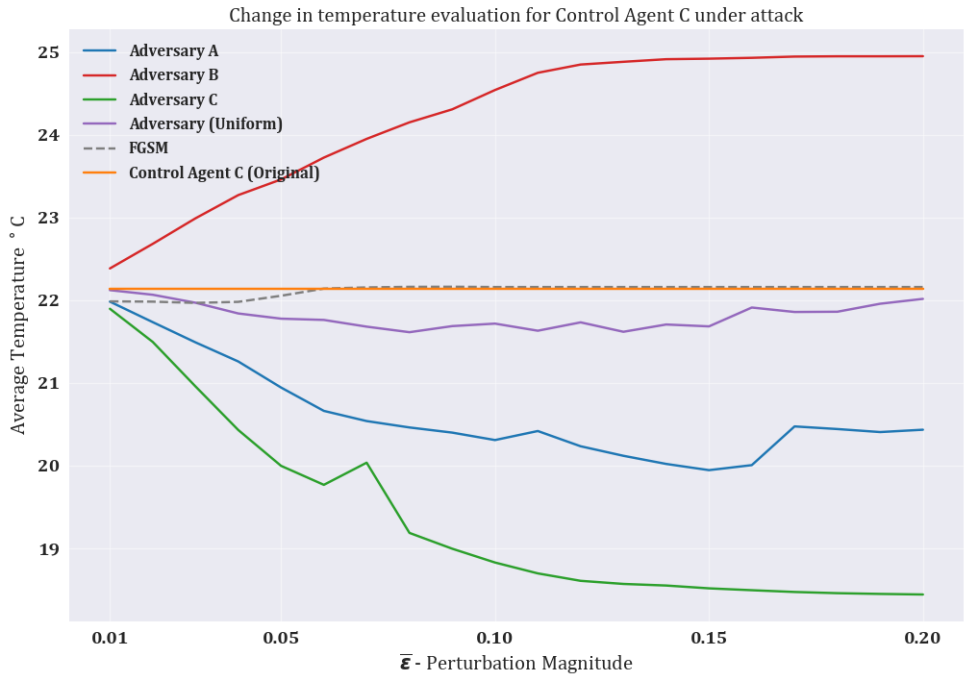

Figure 6.3: Change in temperature for Agent C under attack given the value of the perturbation magnitude $\bar{\varepsilon}$. The original average temperature of Agent C is highlighted in the figure with orange and is not dependent on the x-axis.

For DIVA

{
"Author1": { "name": "Kevin Ammouri"},
"Degree": {"Educational program": "Master's Programme, Computer Science, 120 credits"},
"Title": {
"Main title": "Deep Reinforcement Learning for Temperature Control in Buildings and Adversarial Attacks",
"Language": "eng" },
"Alternative title": {
"Main title": "",
"Language": "swe"
},
"Supervisor1": { "name": "Alessio Russo, Ph.D." },
"Examiner": {
"name": "Prof. Carlo Fischione",
"organisation": {"L1": "School of Electrical Engineering and Computer Science" }
},
"Other information": {
"Year": "2021", "Number of pages": "xiii,75"}
}

TRITA-EECS-EX-2021:344