NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOLVING REWARD-COLLECTING PROBLEMS WITH UAVS AGAINST THE STOCHASTIC ADVERSARY THROUGH REINFORCEMENT LEARNING AND ONLINE OPTIMIZATION

by

Yixuan Liu

March 2021

Thesis Advisor: Ruriko Yoshida
Second Reader: Michael P. Atkinson

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE SOLVING REWARD-COLLECTING PROBLEMS WITH UAVS AGAINST TH E STOCHASTIC ADVERSARY THROUGH REINFORCEMENT LEARNING AND ONLINE OPTIMIZATION	**5. FUNDING NUMBERS**
6. AUTHOR(S) Yixuan Liu	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Unmanned autonomous vehicles (UAV) have made significant contributions to reconnaissance and surveillance missions in past U.S. military campaigns. As the prevalence of UAVs increases, there have also been improvements in counter-UAV technology that make it difficult for UAVs to successfully obtain valuable intelligence within an area of interest. Hence, it has become important that modern UAVs can accomplish their missions while maximizing their chances of survival. In this work, we specifically study the problem of identifying a short path from a designated start to a goal, while collecting all rewards and avoiding adversaries that move randomly on the grid. We present a comparison of two methods to solve this problem: a Deep Q-Learning model and an online optimization framework. Our computational experiments, designed using simple grid-world environments with random adversaries, showcase how these approaches work and compare them in terms of performance, accuracy, and computational time.

14. SUBJECT TERMS reinforcement learning, agent-based environment, online optimization, reward-based game, UAV flight pattern, Deep Q-Learning	**15. NUMBER OF PAGES** 55
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SOLVING REWARD-COLLECTING PROBLEMS WITH UAVS AGAINST THE STOCHASTIC ADVERSARY THROUGH REINFORCEMENT LEARNING AND ONLINE OPTIMIZATION

Yixuan Liu
Lieutenant Junior Grade, United States Navy
BS, United States Naval Academy, 2017

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 2021

Approved by: Ruriko Yoshida
Advisor

Michael P. Atkinson
Second Reader

W. Matthew Carlyle
Chair, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned autonomous vehicles (UAV) have made significant contributions to reconnaissance and surveillance missions in past U.S. military campaigns. As the prevalence of UAVs increases, there have also been improvements in counter-UAV technology that make it difficult for UAVs to successfully obtain valuable intelligence within an area of interest. Hence, it has become important that modern UAVs can accomplish their missions while maximizing their chances of survival. In this work, we specifically study the problem of identifying a short path from a designated start to a goal, while collecting all rewards and avoiding adversaries that move randomly on the grid. We present a comparison of two methods to solve this problem: a Deep Q-Learning model and an online optimization framework. Our computational experiments, designed using simple grid-world environments with random adversaries, showcase how these approaches work and compare them in terms of performance, accuracy, and computational time.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

With the increased use of unmanned aerial vehicles (UAV) for surveillance and reconnaissance missions, there has also been an increase in counter-UAV technology. This means it is more important than ever for UAVs to have robust path planning algorithms that can ensure UAV mission success and survivability when faced with adversarial components. This thesis seeks to explore two potential UAV path planning algorithms by developing two simple grid environments and analyzing how the algorithms perform in them. The two methods we explore are Deep Q-Learning and online optimization.

The first environment we developed is a 5×5 grid environment that is composed of a UAV agent, a stationary reward, and a dynamic adversary. The agent's goal is to start in one corner of the grid, reach the reward, and then exit the grid by reaching the opposite corner of the grid using the shortest path available when avoiding the adversary. The goal of the adversary is to capture the agent by being in the same cell as the agent. While the agent can move anywhere in the grid environment, the adversary begins each experiment located in a cell adjacent to the reward and can only move in the cells immediately adjacent to the reward. We set up 3 different types of experiments where the adversary can either only move counterclockwise around the reward, clockwise around the reward, or to either cells adjacent to the adversary's current location with uniform distribution. The grid environment has a point system where the agent receives a negative point for each step it takes, a huge penalty that ends each experiment trial for being captured by the adversary, positive points for reaching the reward, and positive points for completing each experiment trial having captured the reward. After performing experiments in the 5×5 grid environment, we move on to experimenting in a 9×9 grid environment with two rewards and two adversaries located separately.

The first method we use to approach our problem is a Deep Q-Learning algorithm that has no information regarding the grid environment before starting each experiment. The Deep Q-Learning algorithm begins each experiment with random actions and uses a trial and error approach to learn how to best move in the grid environment. By remembering its past actions and the positive or negative results of those actions, the algorithm uses a neural network to develop a policy for the UAV agent to successfully solve our problem.

Our second method is an online optimization algorithm where our agent solves a series of sequential mixed-integer linear programming problems to develop an optimal path to solve our problem. The agent is allowed a certain number of time steps before each experiment starts in order to observe how the adversary moves. Once each experiment begins, the agent solves a mixed-integer linear programming problem at each time step based on its knowledge of the current reward location, the current adversary location, and how the adversary might move based on the agent's initial observations of the adversary to decide the best action it should take.

While it is difficult to fairly compare the two methods due to their intrinsically different nature and input requirements, we were able to glean much insight into the two algorithms' performances with our problem. While both methods perform well in the 5×5 grid environment, Deep Q-Learning begins to under-perform compared to online optimization in the 9×9 grid environment because of its trial and error approach to solving the problem. Since we want the agent to find the shortest path while avoiding the adversary in each run of the experiment, there is a small penalty for each step the agent takes in order to minimize any extra steps. This small penalty causes the agent to have no desire to explore the entire grid since it continually accrues a small penalty for each step it takes. As a result, the Deep-Q algorithm is usually unable to find both rewards in the bigger grid environment. Online optimization performs much better once we scale up size of the grid environment compared to Deep Q-Learning since we allow our online optimization algorithm to know the locations of the rewards and adversaries. Online optimization is also computational faster than Deep Q-Learning, but we did see a great increase in the computational time for both methods once we began to increase the size of the grid environment.

Acknowledgments

I am very thankful towards Professors Ruriko Yoshida, Chrysafis Vogiatzis, Erich Morman, and Michael Atkinson for the countless hours and effort everyone graciously offered in helping me complete this thesis. This thesis took years in the making and I am extremely grateful to everyone for taking the substantial time out of their ever busy schedules to provide essential input, answer my questions, and give me the encouragement needed to finish this thesis.

I am forever thankful towards my wife, Madeleine, for her constant support and help in every aspect of my life to allow me the opportunity to work on this thesis and complete my degree.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Due to the technological advances made in the past decades and numerous other factors, the U.S. military has recognized the increased value of unmanned aerial vehicles (UAV) in modern warfare campaigns (Grier 2009). UAVs are cost-effective, safe for operators, and can provide surveillance and reconnaissance in areas that might otherwise be inaccessible. However, with the increased usage and threat of UAVs, counter-UAV technology has become an increasing concern when considering UAV deployment. The U.S. military, for example, is investing hundreds of millions of dollars to develop jamming, laser, and gun technologies that will serve to defend against UAV threats (Hoehn and Sayler 2020). In this thesis, we explore and compare two potential methods that can be used for UAV path planning for surveillance missions when there is an active adversary trying to capture and destroy our UAVs. We develop a simple grid environment composed of rewards that our UAV agent must capture and adversaries whom our UAV agent must avoid while traversing through the environment in the shortest path possible. We will approach our problem with Deep Q-Learning and online optimization algorithms and analyze the results.

Robotic systems use five main deliberation functions in order to fulfill their missions (Ingrand and Ghallab 2017). The five deliberation functions are planning, acting, monitoring, observing, and learning. This study focuses on exploring the planning function of robotic systems as a probabilistic planning problem. We propose here to investigate the use of reinforcement learning and online optimization as tools to gain insight into better UAV movement patterns by solving a dynamic, stochastic, rewarding-collecting path problem with side constraints. The UAV is sent from a source node s to a terminal node t in a network; its goal is to reach the terminal node in a fast and safe manner. In addition, during its route, it collects useful information by visiting certain nodes of the network. This is the reward-collecting part of the problem. The dynamic and stochastic nature of the problem comes from the presence of adversaries that may move in a deterministic or stochastic fashion through the nodes in an effort to capture our agent.

There are numerous UAV path planning algorithms and studies comparing the numerous

methods have shown that different methods are appropriate depending on the scenario (Radmanesh et al. 2018). The two methods we are comparing in this thesis is between Deep Q-Learning and an online optimization method. Research has been done in recent years on solving UAV path planning and obstacle avoidance problems with Deep Q-Learning methods (Yan et al. 2019; Yan and Xiang 2018). However, it has not been compared to online optimization in an environment where there is a randomly moving adversary. Our goal here is to implement a Deep Q-Learning algorithm on a simple problem and compare its performance to our online optimization method to glean further insight into the two methods.

We begin our experiments with an extremely simple problem and eventually scale up. Our experimentation begin in a 5×5 grid environment with a single agent, reward, and adversary. Our Deep Q-Learning model frames the problem as a Markov decision process for our UAV agent. The algorithm seeks to find an optimal path by having the agent traverse through the grid using a trial and error approach to learn how to solve the problem. As the agent explores more of the grid, it remembers its experiences and trains a neural network with batches of its past experiences. The goal of the agent is to use the neural network to create an optimal policy for moving around the grid environment. Our Deep Q-Learning model is implemented using the `Python` package Keras (Chollet et al. 2015) and TensorFlow (Abadi et al. 2015). After we attempt our problem using our Deep Q-Learning model, we attempt it using our online optimization method and compare the two.

Our online optimization method is a greedy algorithm to estimate the best path over the entire time horizon by solving sequentially mixed-integer linear programming problems at each time step. Since our adversaries are moving with unknown (stochastic or deterministic) patterns, the problem over the entire time horizon is stochastic. However, in our method, we set up a deterministic problem at each time step using the estimated transition probabilities of adversaries. Then, we use mixed-integer linear programming to find an optimal step for the UAV to move during the next time step by minimizing the risk to encounter adversaries and to take an extra step towards the rewards (and the exit) at the same time. This is a sequential optimization problem since the optimal solution at the current time step sets up the mixed-integer linear programming problem in the next time step. Finally, it is a greedy algorithm since we take the optimal solution at the current time step (local optimal solution), but not the global optimal solution in the entire time horizon. Our online optimization method

is implemented in `Python` using the Gurobi optimizer, a commercial optimization solver (Gurobi Optimization 2020).

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

In this chapter we present some of the necessary background and fundamental information behind Deep Q-Learning and online optimization.

2.1 Machine Learning

To understand how Q-Learning works, we must first discuss the machine learning umbrella that Q-Learning falls under. Machine learning is a subfield of computer science that describes the process of solving problems by gathering a relevant dataset and algorithmically using the dataset to build a statistical model (Burkov 2019). Machine learning is generally categorized into three types: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning consists of training a model with a dataset full of labeled examples to be able to categorize inputs into known classes. Unsupervised learning consists of training a model with a dataset full of unlabeled examples and categorizing inputs based off of features from the unlabeled dataset. Reinforcement learning, which Q-Learning is an example of, consists of learning a policy on how an agent should act based on the state of its environment in order to maximize some kind of reward system.

2.2 Reinforcement Learning

We focus on solving our specific problem with a reinforcement learning algorithm in this thesis because of reinforcement learning's ability to solve near-optimally, complex and large-scale Markov decision processes on which classical dynamic programming breaks down (Gosavi 2003). The Markov decision processes in our problem simply refers to the framework of our UAV agent having to make a decision on what action to take based on its current location and the state of the environment. While our current problem is small in size and can be solved with ease through other methods, we investigate solving it through reinforcement learning as real life problems involving UAVs in unfamiliar environments will be much greater in complexity.

Reinforcement learning is an area of machine learning that focuses on how an agent should

behave inside of an environment. Different actions taken by the agent in different states might have positive, neutral, or negative outcomes in the form of rewards. The goal of reinforcement learning is to derive a policy of actions the agent should take to maximize overall reward.

The process of finding an optimal policy through reinforcement learning requires three elements besides an agent and environment: a policy, a reward signal, and a value function (Sutton and Barto 2018). The **policy** π can be a lookup table or a function that allows the agent to know which action will produce the highest reward based on the state of the agent. The **reward signal** tells us whether an action is a "good" action or a "bad" action based on the rewards we receive. The overall goal of reinforcement learning is to maximize the long term rewards earned by our agent through its actions. The **value function** differs from the reward signal in that while the reward signal is the immediate reward gained from taking actions, the value of a state refer to the long term desirability from being in that state and the subsequent states that follow. We can also provide our algorithm a model of the environment, but it is not a necessity as we can conduct reinforcement learning whether or not we have a model of the environment. A model of the environment should allow us to make predictions about future states and rewards based on current states and actions. When we do not have a model of the environment, reinforcement learning takes a completely trial and error approach to learn more about the environment.

2.3 Q-Learning

For our problem, we will use a reinforcement learning algorithm called Q-Learning (Watkins and Dayan 1992), where the Q is understood to mean "quality". Q-Learning proceeds as following: an agent will take an action in a state within the environment. This action will result in a reward or penalty for the agent and also take the agent to a new state within the environment. The agent will judge the action based on the immediate response it receives and also the value of the new state that it arrives in. By repeatedly trying all of the different actions in all of the states, the agent learns which actions are the best (Watkins and Dayan 1992).

Let s represent the state of an environment the agent is in and a be the action the agent takes from state s. The goal of Q-Learning is to be able to calculate a Q value, $Q(s, a)$,

for all state action pairs in the environment, so that an optimal policy can then be found by always picking the action with the highest Q value in every state. Given a policy π, $Q(s, a)$ can be thought of as the immediate reward gained from taking action a in state s plus the discounted reward of following policy π in the future. Calculating the Q values for an environment can be quite complex as it would require taking every possible action in all possible states for a finite Markov decision process. In Q-Learning, we will use Bellman's equation to approximate the Q values for an optimal policy:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a'), \tag{2.1}$$

where $r(s, a)$ is the immediate reward of taking action a in state s, $Q(s', a')$ is the Q value function for the next state s' and next action a', and γ is the discount factor used to ensure future rewards are worth less than the immediate reward. From Bellman's equation, we calculate $Q(s, a)$ by adding the immediate reward from taking action a in state s and the Q value of the next state action pair with the highest Q value. If we create a policy based on always choosing state action pairs with the highest Q values generated with Bellman's equation, it can be proven that the policy will eventually converge to the optimal policy (Watkins and Dayan 1992). Once we have our Q values, we will have our agent choose to either make the move that results in the highest Q value or make a move completely at random. This exploitation versus exploration strategy allows us to constantly seek new paths that might result in higher Q values.

2.4 Deep Q-Learning

Our Deep Q-Learning model utilizes a neural network with more than 1 hidden layer, thus this Q-Learning algorithm is termed Deep Q-Learning. This technique is motivated by a company named Deepmind that has gained a lot of attention in recent years due to the success of their Deep reinforcement learning methods. Deepmind proposed a method called Deep Q Network (DQN) to perform a variant of Q-Learning with convolutional deep neural networks to keep track of future rewards (Mnih et al. 2013). DQN uses a neural network function approximator called a Q-network that can be trained by minimizing a sequence of loss functions by stochastic gradient descent. The motivation behind this method is that traditional methods of iteratively updating the Q values for an environment with Bellman's equation has very high computational complexity. DQN is able to more efficiently generate

Q values by training the neural network with random batches of experience replays that are composed of $e_t = (s_t, a_t, r_t, s_{t+1})$. These experience replays are stored every time the agent takes an action and a random sample of them are used to train the neural network every time step. Their method has achieved exceptional results when playing Atari games and their success led to their acquisition by Google for $500 million (Shu 2014). Deep Q-Learning has been shown to have a variety of applications such as stock market forecasting (Carta et al. 2020), scheduling schemes (Zhang et al. 2017), and handwritten digits recognition (Qiao et al. 2018).

2.5 Online Optimization

The final approach to our problem is through online optimization. While traditional statistical learning methods are robust, they are often unable to solve problems where the data is dynamic and our knowledge of the problem is incomplete. Online optimization provides us with a way to solve sequential problems where we make no probabilistic assumptions of the distributions of our inputs and require only partial knowledge or a partial view of the problem (Bubeck 2011). Online optimization can be used to solve a variety of problems such as network routing (He et al. 2013), e-mail spam filtering (Wang et al. 2006), or stock portfolio selection (Li and Hoi 2014). The general idea behind online optimization is that we are solving a series of sequential problems instead of one big problem. Each problem is solved based on information we obtained from solving previous problems and observing the resulting environments.

A generic online optimization decision process for an optimizing agent can be described below (Belmega et al. 2018).

Given a time horizon $T = \{0, 1, \ldots, |T|\}$, for every time step $t \in T$:

Step 1. The agent chooses an action x_t based on its current knowledge of the problem after solving an optimization problem $LP(t)$.

Step 2. The agent incurs some sort of loss based on the selected action, $l_t(x_t)$.

Step 3. The agent observes the new environment and obtains new knowledge.

In the process described above, the agent's action x_t must always come from a finite amount of possible actions and the loss function l_t can be of any form. The goal of online optimization

is ultimately to minimize the loss function as we proceed through the time steps.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Experimental Setup

In this chapter, we describe the experimental setup and the specific implementation details of the approaches adopted.

3.1 Environment Setup

A simple set up of the a 5×5 grid environment with a single reward and adversary can be seen in Figure 3.1:

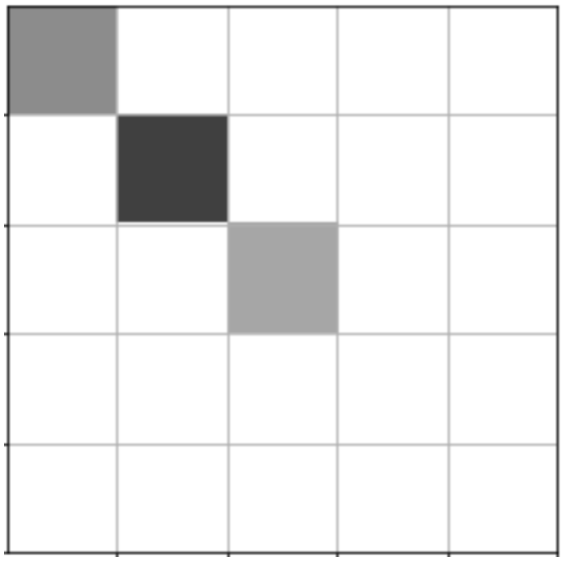

Figure 3.1. 5x5 Grid Environment Setup

Figure 3.1 shows how an experiment trial may be initialized. We have our agent starting in the upper left corner, our stationary reward in the center of the grid, and our dynamic adversary starting in the cell to the upper left of our reward. The adversary can only move in the cells directly surrounding our reward, but our agent can move anywhere in the grid. The goal of the agent is to traverse through the grid and reach the reward in the center of the grid

using the shortest path while simultaneously avoiding the adversary. Once it has reached the reward, it then proceeds to the lower right corner in order to finish the trial in a similar fashion of finding the shortest path and simultaneously avoiding the moving adversary. The reward will remain stationary in all of our experiments.

3.2 Types of Adversaries

The environment for the adversary is fixed and a subset of the entire environment. This means that the adversary can go only a subset of the entire grid and the place where the adversary can go is fixed. Each reward in our environment is surrounded by a moving adversary. The adversary can only move in the cells adjacent to each reward and in one of three different ways each experiment: clockwise around the reward, counterclockwise around the reward, or uniform randomly into adjacent cells. In each time step of our experiment, our agent will take one step, and then our adversary will take one step. At the end of each time step, if our agent and adversary end up in the same node, then we consider the agent to be captured and the experiment to be over.

3.2.1 Adversary Actions

At each time step, the adversary can only take one of four possible moves depending on their location in the environment: moving

- up
- down
- left
- right

in the given subset of the entire grid.

Figure 3.2 shows the possible cells that an adversary can move in reference to a reward. If a reward is located in the blank center grid, then an adversary will be able to move in the cells labelled one through eight directly around it. If the adversary is moving clockwise around the reward, then an adversary starting in cell one will move to cell two, three, five, and so on. If the adversary is moving counterclockwise around the reward, then an adversary starting in cell one will move to cell four, six, seven, and so on. A randomly moving adversary will

move with uniform distribution to either cells adjacent to it. So, an adversary starting in cell one will move to either cell two with a probability of 0.5 or cell 4 with a probability of 0.5.

1	2	3
4		5
6	7	8

Figure 3.2. Adversary Possible Locations

3.3 Deep Q-Learning Method

Our Deep Q-Learning model and code was built directly upon the Tour De Flags code written by Zafrany (2017). Most of the parameters of our reinforcement learning and neural network model remain the same as his as those have already been tuned for a similar experiment.

As described earlier in Chapter 2, our Deep Q-Learning algorithm relies on finding the optimal policy by always choosing state action pairs with the highest Q values generated with Bellman's equation. The trick is being able to generate correct the correct Q values from our agent's experiences exploring the grid environment. The method which we used to do this is to create a model where the target function of our neural network is to match the right side of Bellman's equation. Our goal is to minimize the difference between our neural network's estimation of the state action Q-values generated by the right side of Bellman's equation, and the actual Q-values our agent experiences from traversing through our grid environment.

We train our neural network by allowing our agent to traverse through the environment and keeping a record of the its most recent experiences in a way that we will explain more below. Each time it moves, it will also select a random sample of the record of its most recent experiences and use this information to train our neural network. Once the agent's record of most recent experiences has grown too large, we will delete the older records and only keep the newer ones. This allows our neural network to be trained with newer data as our

agent learns more and more of the environment.

Some of the parameter values that we use in our Deep Q-Learning algorithm are listed below:

- number of epochs - number of attempts our agent is allotted for each experiment trial: 1000;
- max memory - the number of records of recent experiences our agent keeps before deleting older records: 500;
- data size - the number of records of recent experiences used to train our neural network each time the agent moves: 100;
- exploration factor - the percentage of times the agent chooses a completely random move instead of an optimal move: 0.2; and
- discount factor - the γ coefficient in the right side of Bellman's equation used to discount the worth of future rewards - 0.97.

We will conduct reinforcement learning in this thesis in `Python` using Tensorflow and Keras packages. Originally developed by Google, Tensorflow is an open-source platform to facilitate the development and application of machine learning techniques (Abadi et al. 2015). Similarly, Keras is an API to enable deep learning approaches on top of Tensorflow with the goal of fast prototyping (Chollet et al. 2015).

Keras works by processing vectorized and standardized representations of raw data such as NumPy arrays. Each time the agent takes a step, often referred to as an episode, we store data in the following manner: episode = [env_state, action, reward, next_env_state, trial_over]. Each element of the array consists of either numbers or arrays of numbers representing respectively the grid environment before any action takes place, the action the agent takes, the reward that resulted from the respective action, the updated grid environment after the action takes place, and whether or not the experiment trial is still ongoing. Since Keras only works with vectorized representations of raw data, the environment states recorded are compressed 1-dimensional arrays where each element represents a cell in our grid environment. Different values in the arrays allow the neural network to know where the agent and the reward are located.

We use the same neural network that Professor Zafrany uses in his Tour de Flags code,

which is a sequential deep neural network model built with Keras. The neural network model has four fully connected layers composed of one input layer, two hidden layers, and one output layer. Our input layer is the same size as our grid environment, so a 5×5 grid environment will have an input layer of 25. This is because our input to the neural network is a one dimensional array that displays where within the grid environment our agent is currently located. Our output layer is size four, representing the four actions our agent can take and producing a Q-value for each action.

We use LeakyReLu activation layers in our neural network to prevent our neural network from becoming too sparse. While the rectified linear unit has a gradient of zero when not active, the LeakyReLu class in keras always allows at least a small gradient (Maas et al. 2013):

$$f(x) = \begin{cases} \alpha \cdot x, & \text{if } x < 0. \\ x, & \text{if } x >= 0. \end{cases} \tag{3.1}$$

The optimizer we use in our neural network is the Adam optimizer (Kingma and Ba 2014), which uses a stochastic gradient descent method.

3.4 Online Optimization Method

Our online optimization problem is set up where our agent is able to observe the adversary for a certain number of time steps in order to obtain an observed transition probability matrix for the adversary's future moves. At each time step, the agent knows the current location of the adversary, the current location of the reward, and the location of the exit cell of the grid environment. It uses this information to solve a series of mixed-integer linear programming problems to calculate the optimal path to move through the grid environment based on the observed adversary transition probability matrix. Our online optimization model is described below.

Let $G(V, E)$ be an undirected graph/grid with a series of targets on the grid $k \in \mathcal{K}$, where $\mathcal{K} \subseteq V$. Let r_k^t be the reward from reaching target k at time $t \geq t_0$, where t_0 marks the current time; for convenience, we let $r_i^t = 0, \forall i \in V \setminus \mathcal{K}$. We consider one of the locations in the grid the "exit" node, d. When the agent reaches this node, they have effectively exited

the grid, and they no longer need to move.

Moreover, define p_i^t as the probability of seeing an adversary at location i at time $t \geq t_0$. Finally, let ϕ be the "importance" we place in avoiding the adversary.

We are now ready to move to the definition of our decision variables. We define two integer variables, x_{ij}^t and y_{ik}^t, as follows:

$$x_{ij}^t = \begin{cases} 1, & \text{if traversing } (i,j) \in E \text{ at time } t \geq t_0, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_i^t = \begin{cases} 1, & \text{if arriving at node } i \in V \text{ at time } t \geq t_0, \\ 0, & \text{otherwise.} \end{cases}$$

Before we describe the mathematical formulation, we describe the process. At each time step t_0 and starting from $t_0 = 0$, we solve this optimization problem. Then, we observe the movement of the adversary and then move again. In effect, we collect a series of optimal solutions, one for each current time t_0.

Our model's components can be summarized below:

Index Use :

$i, j \in V$	nodes
$(i, j) \in E$	arcs from i to j
t	time steps
t_0	initial time step
d	exit node

Data :

r_i^t	reward from being in node i at time t
p_i^t	probability of adversary being in node i at time t
ϕ	importance in avoiding the adversary

Decision Variables :

$x_{ij}^t \in V$	traversing $(i, j) \in E$ at time t
$y_i^t \in V$	arriving at $i \in V$ at time t

We then have the formulation in (3.2). For each time step $t_0 = 0, 1, \ldots$, we have:

$$\min \quad \sum_{i \in V} \sum_{t \geq t_0} \left(t - r_i^t + \phi \cdot p_i^t\right) \cdot y_i^t \tag{3.2a}$$

$$s.t. \quad \sum_{i \in V \setminus \{d\}} y_i^t \leq 1, \qquad \forall t \geq t_0, \tag{3.2b}$$

$$\sum_{t \geq t_0} y_i^t \leq 1, \qquad \forall i \in V \setminus \{d\}, \tag{3.2c}$$

$$\sum_{j:(i,j) \in E} x_{ij}^{t+1} = y_i^t, \qquad \forall i \in V \setminus \{d\}, \forall t \geq t_0, \tag{3.2d}$$

$$\sum_{j:(j,i) \in E} x_{ji}^t = y_i^{t+1}, \qquad \forall i \in V, \forall t \geq t_0, \tag{3.2e}$$

$$\sum_{t \geq t_0} \sum_{i:(i,d) \in E} x_{id}^t = 1, \tag{3.2f}$$

$$x_{ij}^t \in \{0, 1\}, \qquad \forall (i, j) \in E, \forall t \geq t_0, \tag{3.2g}$$

$$y_i^t \in \{0, 1\}, \qquad \forall i \in V, \forall t \geq t_0. \tag{3.2h}$$

In the formulation, the objective function in (3.2a) minimizes a "risk function" for our agent. The risk function consists of three components, all multiplied by the decision variable of visiting some node i at some future time t:

1. a "negative" reward, as we would like to collect as many rewards as possible;
2. a $\phi \cdot p_i^t$ component, which aims to consider future positions the agent may take; and
3. an addition of t, since we would like the agent to take fewer steps on their way to collecting the rewards and exiting the grid.

Constraints (3.2b) and (3.2c), we simply enforce that our agent is in at most one location at each time; observe how this allows for being in "no location" if the agent has reached the exit point d. Constraints (3.2d) and (3.2e) are flow preservation constraints which state that the agent has to use one of the edges around them at each location they find themselves. Finally, constraint (3.2f) enforces that the agent will have to exit at some point; recall that the objective also makes the agent want to exit faster, if possible. Constraints (3.2g) and (3.2h) are variable restrictions seeing as both of our decision variables are binary.

CHAPTER 4:
Computational Results

For our experiments, we created two environments to experiment with a 5×5 grid environment and a 9×9 grid environment. The results from experiments using our Deep Q-Learning and online optimization methods are discussed in this chapter.

4.1 5 x 5 Grid Environment Results

We begin the section with a description of the initial 5×5 setup. In Figure 3.1, we can see the initial placement of our agent, adversary, and reward. The agent starts in the upper left corner of the grid, the reward is in the center of the grid, and the adversary is adjacent northwest to the reward. The agent must find a short and safe path to capture the reward and reach the lower right corner of the grid.

We begin our experiments with our reinforcement learning algorithm. We set up our experiments for three different types of adversary that move clockwise, counterclockwise, and randomly. We initialize our experiments with a 5×5 grid environment, a single reward in the center of the 5×5 grid, and a single adversary that starts in the upper left cell adjacent to our reward. The reward system for our grid environments are set up as below:

- each step that does not result in capturing a reward or reaching the final destination: -1;
- reaching the reward: +200;
- being captured by the adversary: -1000; and
- reaching the bottom right corner of our 5×5 grid having captured the reward: +100.

It takes a minimum of eight steps for the agent to go from one corner of the grid to another while passing through the center of the grid environment to capture the reward. If the agent knows exactly how the adversary will move, then it will always be able to complete each experiment trial in eight steps as there is no way for the adversary to force the agent to take more than eight steps. Taking eight steps will cause the agent to achieve a final reward of 294 using the reward system described above, so we consider 294 to be the optimal reward

for our 5×5 experiments.

In an effort to make our Deep Q-Learning and online optimization algorithms more comparable, we do not have a training period for our Deep Q-Learning algorithm to train our neural network and then experiment with the trained neural network. We instead have each replication using our Deep Q-Learning algorithm consist of our agent being allowed 1000 attempts to capture the reward and exit the grid. If the agent successfully completes its tasks once, we consider the replication a success and end the replication. If the agent cannot successfully complete its tasks within 1000 epochs, then we consider the replication to be a failure. We set up our experimentation this way because each of our online optimization replications only allow our agent one attempt. If we allow our reinforcement learning algorithm to have a training period, then we are allowing the agent to potentially successfully complete its tasks multiple times in the training period and thus unfairly comparing the two methods. While it is impossible to make comparisons between the two methods completely fair as they are intrinsically different algorithms and require different input, we attempt to make the two algorithms more comparable by stopping each Deep Q-Learning replication once the agent successfully its tasks once.

Successfully completing an experiment trial requires the agent to capture the reward, avoid the adversary, and reach the lower right corner of our environment. If the agent is captured by the adversary at any point, then we consider the replication to be a failure. An example of the path traveled by the agent from a successful trial is displayed in Figure 4.1.

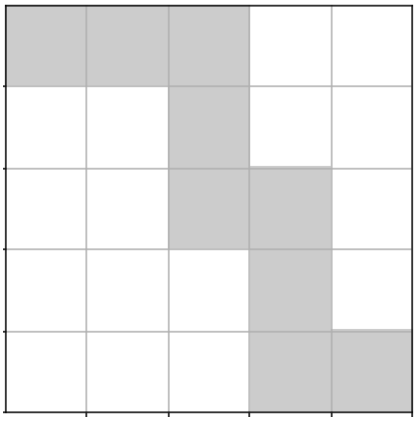

Figure 4.1. 5×5 DQN Successful Path Example

We can see from Figure 4.1 that the agent is able to reach the center of the grid during its path and capture the reward. It then proceeds to exit the grid environment by reaching the lower right corner. This path is considered to be optimal because the agent takes eight steps to complete its tasks, which is the shortest possible amount of steps required.

For our experiments, we replicated each type of adversarial movement scenario for 50 trials. The average reward and regret are calculated only from replications that are successful in that the agent is able to capture the reward and reach the lower right corner of the grid without being captured by the adversary.

From Table 4.1, we can see that Deep Q-Learning performs well for the 5×5 grid environment with a single adversary regardless of adversarial movement. The algorithm is able to successfully complete all 50 replications for each type of adversary movement. One thing to note is that it does seem to take longer for the algorithm to complete 50 replications when the adversary moves randomly.

Table 4.1. 5×5 Environment: Deep Q-Learning Results

5×5 Environment - Deep Q-Learning Results			
Adversary Movement	Clockwise	Counterclockwise	Random
# of Successes	50	50	50
Average Reward	292.60	292.56	292.60
Average Regret	1.40	1.44	1.40
Time(secs)	173.74	198.60	312.74

Now we will attempt to solve the problem using online optimization. For our online optimization method, our experiment is simply a traditional optimization problem when the adversary moves clockwise or counterclockwise. As long as our agent can observe our adversary for 8 time steps before starting to move, it is able to know exactly how the adversary will move as it takes 8 steps for the adversary to complete its route and return to its starting position. Experiments involving randomly moving adversaries are more complicated and will be described later in this chapter.

The online optimization results for the clockwise and counterclockwise moving adversary are displayed in Table 4.2.

Table 4.2. 5×5 Environment: Deterministic Adversary Online Optimization Results

5×5 Environment - Deterministic Adversary Online Optimization Results		
Adversary Movement	Clockwise	Counterclockwise
# of Successes	50	50
Average Reward	294	294
Average Regret	0	0
Time(secs)	48.89	47.32

From the results in Table 4.2, we can see that the online optimization framework works

perfectly and efficiently when the adversary moves deterministically. When the adversary moves randomly, we are unable to succeed one hundred percent of the time with as little as 8 time step observations. We need to increase the number of observations before allowing our agent to begin the experiments in order to ensure success. This is because with a small amount of time step observations, our agent is unable to accurately capture the correct transition probability matrix the adversary is using to move around the reward.

Using the labelling system in Figure 3.2, we are able to generate the transition probability matrix for a random adversary in Figure 4.2. Each transition probability is 0.5 because the adversary has an equal chance to move either cell adjacent to it.

$$\mathbf{P} = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 2 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 4 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 5 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 \\ 6 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 8 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \end{array}$$

Figure 4.2. Random Adversary True Transition Probability Matrix

When the agent is only able to observe the randomly moving adversary for 10 time steps, we get the observed transition probability matrix in Figure 4.3.

$$\mathbf{P_{10}} = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0.4 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 4.3. Random Adversary 10 Observations Transition Probability Matrix

We can see from Figure 4.3 that when we are only able to observe the adversary for 10 time steps, our observed transition probability matrix is far from accurate. Since the adversary moves according to the true transition probability matrix, and our agent solves its optimization problems using the observed transition probability matrix, it is obvious that our agent will be unable to accurate predict the movement of the adversary due to the discrepancy between the two transition probability matrices. When we can observe the adversary for 75 time steps, we can see from Figure 4.4 that the observed transition probability matrix is far more accurate.

$$\mathbf{P_{75}} = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 0.44 & 0 & 0.56 & 0 & 0 & 0 & 0 \\ 2 & 0.47 & 0 & 0.53 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0.75 & 0 & 0 & 0.25 & 0 & 0 & 0 \\ 4 & 0.57 & 0 & 0 & 0 & 0 & 0.43 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0.56 & 0 & 0 & 0.44 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0 & 0.25 \\ 8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Figure 4.4. Random Adversary 75 Observations Transition Probability Matrix

In Figure 4.5, we see an example of what a successful run in the 5x5 grid environment

looks like using the online optimization approach. The top nine images show the path that the agent traveled, while the bottom nine images show the respective movement of the adversary. We can see that the adversary initially moves downward and stays to the left of the reward. The agent takes advantage of this by going right to completely avoid ever being captured by the adversary.

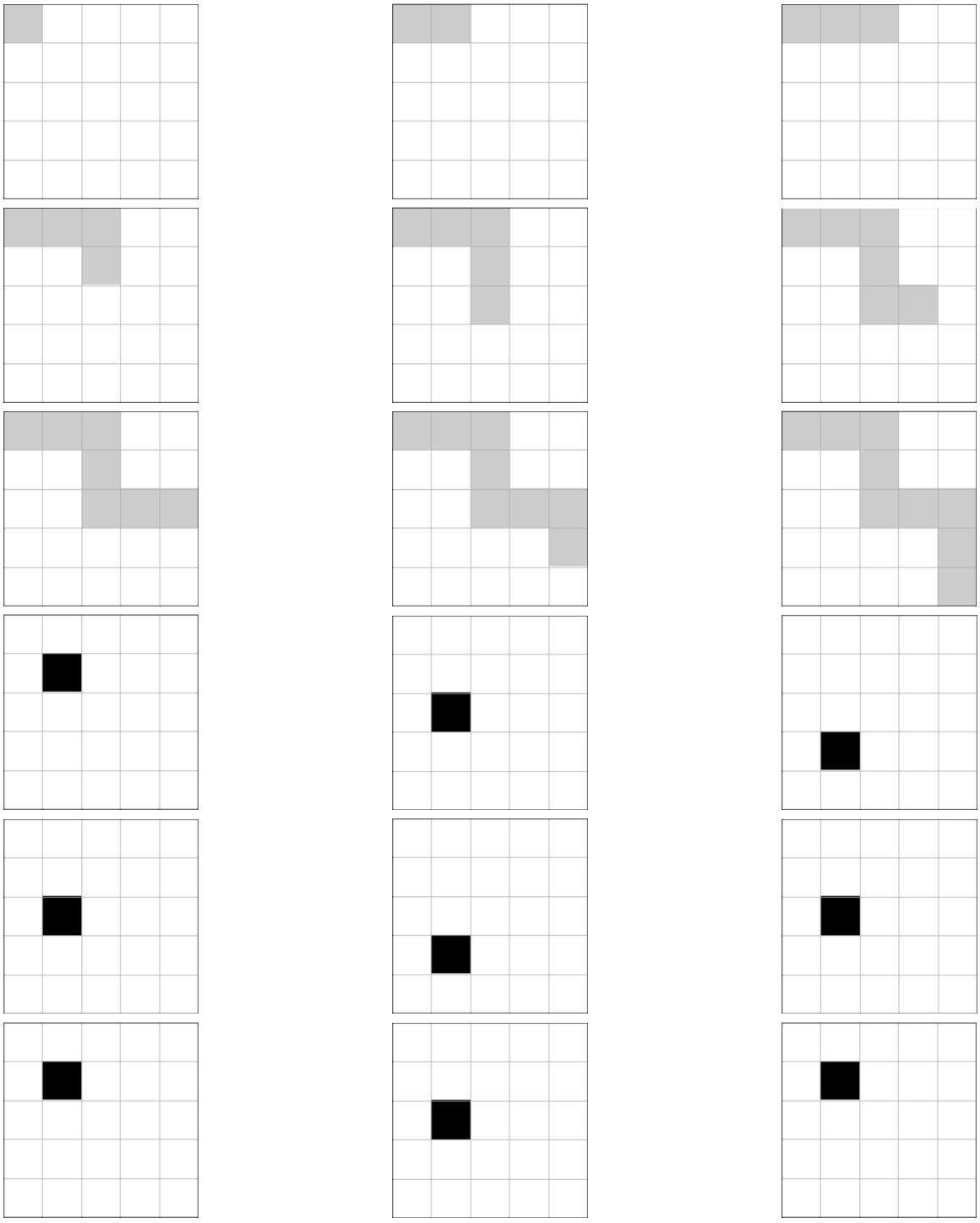

Figure 4.5. Online Optimization Successful Example

The results in Table 4.3 show the online optimization results after collecting 10, 25, 50, and 75 observations on a randomly moving adversary.

Table 4.3. 5×5 Environment: Random Adversary Online Optimization Results

5×5 Environment - Random Adversary Online Optimization Results				
Adversary Movement	Random	Random	Random	Random
# of Observations	10	25	50	75
# of Successes	18	27	36	50
Average Reward	294	293.04	290.61	278.88
Average Regret	0	.96	3.39	15.12
Time(secs)	29.27	39.71	69.74	69.69

We can see from Table 4.3 that we do not successfully complete all 50 runs with online optimization until we are able to observe the adversary for 75 time steps. Another thing to note is that while success rate is low with fewer observations, there appears to be higher reward. This is due to the fact that when the agent does not have accurate knowledge of how the adversary will move, it speeds through the grid using the shortest path and often gets captured. When the agent is able to avoid the adversary by chance, it is able to complete the experiment using the shortest path and achieve a high reward. As the agent has more accurate knowledge of how the adversary will move, it will take more steps in order to avoid the adversary and therefore take a longer path to complete each experiment, resulting in a lower reward. A line chart plotting the number of observations against the probability of success is shown in Figure 4.6.

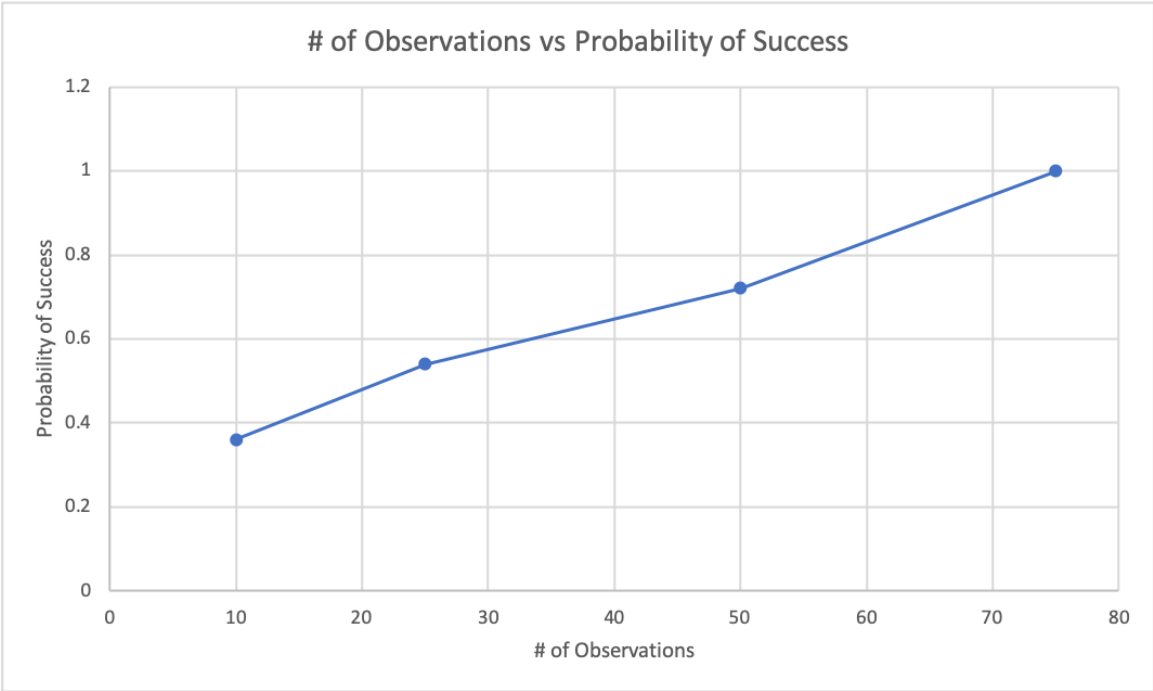

Figure 4.6. 5×5 Online Optimization Randomly Moving Adversary Results

4.2 9 x 9 Grid Environment Results

Now, let us expand our experiments by increasing the grid size to 9×9 and allowing there to be 2 rewards and 2 adversaries. We will place the adversaries far from each other and start the adversaries again in the upper left cell adjacent to each reward. With a bigger environment and two rewards, the optimal reward now is 475 as it will take a minimum of 28 steps to complete each experiment. Figure 4.7 shows the set up of the grid environment.

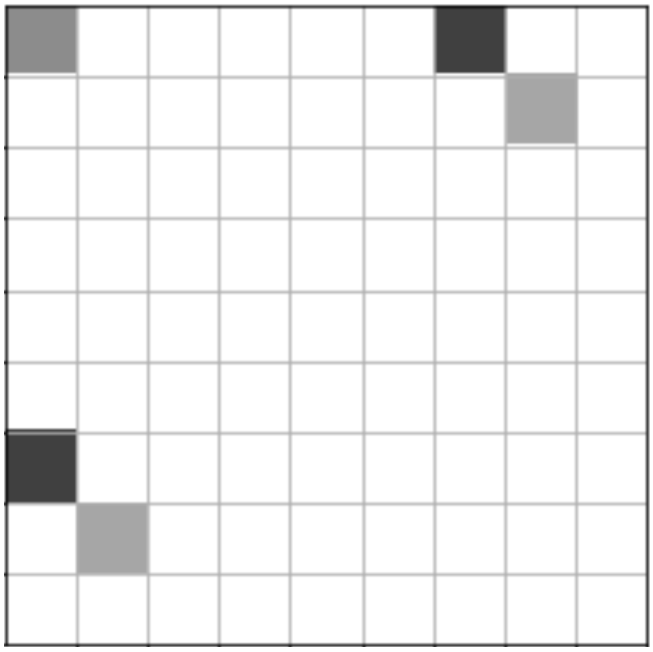

Figure 4.7. 9×9 Grid Environment Setup

We ran our experiments first with Deep Q-Learning and received much different results when the environment size has increased. From Table 4.4, we can see that the algorithm took much longer time to run and achieved minimal success. This is due to the fact that our agent has no idea where the rewards are located. Once our agent has reached one reward and received a positive response, it has little incentive to experiment crossing the entire grid and continually accrue a small penalty for each step in order find another reward. The only reason that the Deep Q-Learning algorithm has any success is due to the exploratory nature of the algorithm. Even with a 1000 epochs, or 1000 tries, for each of the 50 times we ran our Deep Q-Learning algorithm, our agent was still unable to successfully complete the experiments a single time when the adversary moved randomly.

Table 4.4. 9×9 Environment: Deep Q-Learning Results

9×9 Environment - Deep Q-Learning Results			
Adversary Movement	Clockwise	Counterclockwise	Random
# of Successes	7	4	0
Average Reward	322.43	342.50	0
Average Regret	152.57	132.50	475
Time (days)	8.75	9	1.69

When we tried completing the 9×9 grid environment using online optimization, we saw similar results to the 5×5 grid environment, but with much longer computational time. For the clockwise and counterclockwise moving adversaries, 8 time step observations was once again enough to ensure success as we can see in Table 4.5.

Table 4.5. 9×9 Environment: Deterministic Adversary Online Optimization Results

9×9 Environment - Deterministic Adversary Online Optimization Results		
Adversary Movement	Clockwise	Counterclockwise
# of Successes	50	50
Average Reward	475	475
Average Regret	0	0
Time(secs)	1472.28	2250.28

We can see that with 8 time step observations, online optimization is able to achieve the optimal reward and always succeed when the adversary moves deterministically. The online optimization algorithm is also able to complete the experiments in much faster time than Deep Q-Learning. When the adversary moves randomly, we see similar results to the 5×5 case, in that success is not ensured until after collecting observations for 75 time steps. From Table 4.6, we see very similar trends between the 9×9 and the 5×5 case. While the probability of success increases with more time step observations, the average reward of successful runs go down.

Table 4.6. 9×9 Environment: Random Adversary Online Optimization Results

9×9 Environment - Random Adversary Online Optimization Results				
Adversary Movement	Random	Random	Random	Random
# of Observations	10	25	50	75
# of Successes	11	17	24	50
Average Reward	475	472.88	465.50	453.40
Average Regret	0	2.12	9.50	21.60
Time(hours)	0.26	0.33	5.79	8.75

A line chart plotting the number of observations against the probability of success is shown in Figure 4.8.

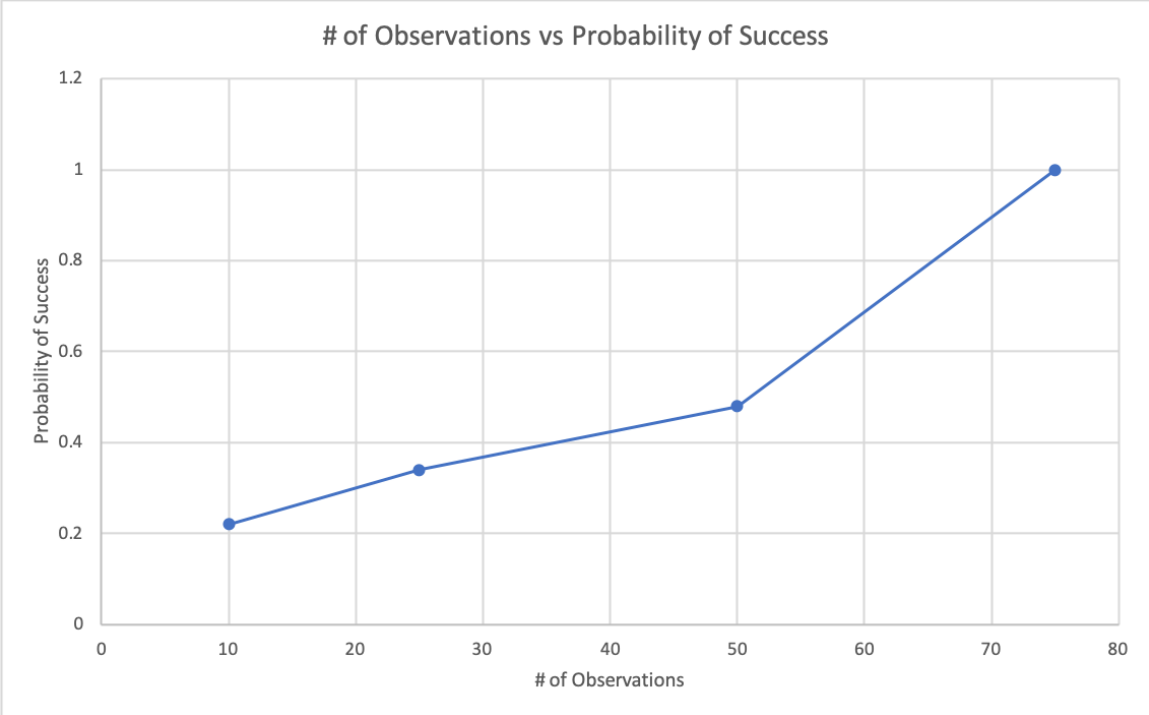

Figure 4.8. 9×9 Online Optimization Randomly Moving Adversary Results

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusion

In this thesis, we were able to learn a great deal on how the two algorithms perform when faced with our specific problem setup. In the 5×5 Grid environment, the Deep Q-Learning and online optimization algorithms performed relatively well with quick computational times and low regret. However, once we increased the grid environment size and placed two sets of rewards and adversaries far from each other, Deep Q-Learning performed much worse than online optimization. This is due to the exploratory nature of our Deep Q-Learning algorithm as it has zero knowledge of the environment and learn everything through trial and error. Our online optimization algorithm may not know initially how the adversary moves, but it is given the chance to observe the adversary for a certain number of time steps and has complete knowledge of the reward and adversary locations. While it is unfair to compare the two algorithms based on our results because they begin with different amounts of knowledge of their operating environment, we were still able to gain a lot of insight into the two algorithms in regards to simple grid environments.

When we increase the size of the grid environment and the number of rewards, it appears that the Deep Q-Learning algorithm has trouble locating all of the rewards and completing the experiments as there is a penalty for each additional step it must take. Once it has found one reward, it has little motivation to keep exploring further and further in hopes of finding an additional reward. Online optimization works well in our experiments, but the computational time increases significantly when we increase our grid environment as the algorithm has much more computations. It is interesting to note that in both our smaller and bigger experiment grids, it took 75 time steps before our algorithm was able to successfully predict the movement of the adversaries and successfully complete the experiments every time when the adversary moves randomly.

The experiments we ran were initial experiments, and there are much more additional experiments worth exploring. We did not scale past the 9×9 grid environments due to extremely long computational times. Some future work may involve adjusting the parameters of our Deep Q-Learning algorithm and online optimization algorithms to increase efficiency

and effectiveness, incorporating parallel processing into our codes to decrease computational time, and doing more experiments with different grid environments, rewards, and adversaries to learn more about the two algorithms.

List of References

Abadi M, et al. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Accessed January 17, 2020, https://www.tensorflow.org/.

Belmega V, Mertikopoulos P, Negrel R, Sanguinetti L (2018) Online convex optimization and no-regret learning: Algorithms, guarantees and applications. *arXiv preprint* arXiv:1804.04529, https://arxiv.org/abs/1804.04529.

Bubeck S (2011) Introduction to online optimization. Lecture, December 14, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ.

Burkov A (2019) *The hundred-page machine learning book*, volume 1 (Andriy Burkov, Quebec City, Canada).

Carta S, Ferreira A, Podda AS, Recupero DR, Sanna A (2020) Multi-DQN: An ensemble of deep q-learning agents for stock market forecasting. *Expert Systems with Applications* 164:113820.

Chollet F, et al. (2015) Keras. Accessed January 17, 2020, https://github.com/keras-team/keras.

Gosavi A (2003) *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning* (Kluwer Academic Publishers, Boston, MA).

Grier P (2009) Drone aircraft in a stepped-up war in afghanistan and pakistan (December 11), https://www.csmonitor.com/USA/Military/2009/1211/Drone-aircraft-in-a-stepped-up-war-in-Afghanistan-and-Pakistan.

Gurobi Optimization L (2020) Gurobi optimizer reference manual. Accessed January 17, 2020, http://www.gurobi.com.

He T, Goeckel D, Raghavendra R, Towsley D (2013) Endhost-based shortest path routing in dynamic networks: An online learning approach. *2013 Proceedings IEEE INFOCOM*, 2202–2210 (IEEE).

Hoehn JR, Sayler KM (2020) Department of defense counter-unmanned aircraft systems. https://fas.org/sgp/crs/weapons/IF11426.pdf.

Ingrand F, Ghallab M (2017) Deliberation for autonomous robots: A survey. *Artificial Intelligence* 247:10–44.

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li B, Hoi SC (2014) Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)* 46(3):1–36.

Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. *Proc. imcl*.

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller A (2013) Playing atari with deep reinforcement learning. *CoRR* abs/1312.5602, http://arxiv.org/abs/1312.5602.

Qiao J, Wang G, Li W, Chen M (2018) An adaptive deep q-learning strategy for handwritten digit recognition. *Neural Networks* 107:61–71.

Radmanesh M, Kumar M, Guentert P, Sarim M (2018) Overview of path-planning and obstacle avoidance algorithms for UAVs: A comparative study. *Unmanned systems* 6(02):95–118.

Shu C (2014) Google acquires artificial intelligence startup deepmind for more than $500m. Tech Crunch. Accessed January 17, 2021, https://techcrunch.com/2014/01/26/google-deepmind/.

Sutton R, Barto A (2018) *Reinforcement learning an introduction* (The MIT Press, Cambridge, MA).

Wang Q, Guan Y, Wang X (2006) SVM-based spam filter with active and online learning. *TREC* (Citeseer).

Watkins C, Dayan P (1992) Technical note. *Reinforcement Learning* 55–68, doi:10.1007/978-1-4615-3618-5_4.

Yan C, Xiang X (2018) A path planning algorithm for UAV based on improved q-learning. *2018 2nd International Conference on Robotics and Automation Sciences (ICRAS)*, 1–5 (IEEE).

Yan C, Xiang X, Wang C (2019) Towards real-time path planning through deep reinforcement learning for a UAV in dynamic environments. *Journal of Intelligent & Robotic Systems* 1–13.

Zafrany S (2017) Deep reinforcement learning the tour de flags test case. Accessed January 18, 2021, https://www.samyzaf.com/ML/tdf/tdf.html.

Zhang Q, Lin M, Yang LT, Chen Z, Li P (2017) Energy-efficient scheduling for real-time systems based on deep q-learning model. *IEEE transactions on sustainable computing* 4(1):132–141.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California