DEVIL'S DGA

A Thesis by

Brian C. Akins

Bachelor of Science, Oklahoma State University, 2012

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2021

DEVIL'S DGA

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Sergio Salinas, Committee Chair

Saideep Nannapaneni, Committee Member

Ajita Rattani, Committee Member

KC Dukka, Committee Member

DEDICATION

I would like to dedicate this thesis to my wife Anna, my two sons Eric and Ayden, my mom, my dad, my friends, and my colleagues. Throughout my life and especially now, each of you have provided me with the support necessary for me to make this accomplishment.

"Pursue what is meaningful, not what is expedient " - Jordan Peterson

ACKNOWLEDGEMENTS

I would like to thank Dr. Sergio Salinas for agreeing to be my advisor, helping me find the path to the ideas presented in this thesis, and for his helpful comments on the text. I would also like to thank Endgame Inc., specifically J. Woodbridge et al., for writing the paper [1] and providing the source code for their detection algorithm used as inspiration in this thesis.

I would also like to thank the administrators of the Beoshock High Performance Computer (HPC) for providing access for me to run my experiments on their hardware, specifically David Turner who supplied advice on batch job setup.

I would like to thank my Wife, Anna for the love and support she provided while I completed this work. While not involved in the technical aspects of completing this work, she did make sacrifices and worked to help make this project a reality.

And last, but not least I would like to thank my mom and dad who provided the guidance, and the room to grow that I needed to grow up to be who I am today.

ABSTRACT

Botnets are a collection of devices that have been infected with malware to grant control to a cyber attacker over that device. Botnets are primarily operated over the internet, where the devices can contact the cyber attacker to receive instructions. The existence of these botnets can cause massive issues for not only the owner of the device, but for anyone who uses the internet. Network detection algorithms have shown promise in previous works as a possible solution for combating botnets. These detection algorithms search for common botnet activity like domain generation algorithms (DGA) which botnets use to find the web address to contact the cyber attacker. In this paper we will contest one of these network detection algorithms. Our proposal is based on the idea of a Devil's Advocate. A Devil's Advocate is a person, or machine learning algorithm in this case, that contests an idea in order to test the strength of the idea. Thus, we propose *Devil's DGA*, a Deep Reinforcement Algorithm that searches for a possible modification the attacker can make to avoid a network detection algorithm. *Devil's DGA* takes a known DGA and modifies the algorithm's features to create a new DGA that can avoid detection algorithms. We show that *Devil's DGA* is also able to achieve detection rates as low as 1.1% *Devil's DGA* is also able to find a means to avoid detection in less than 28 seconds.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

α	Alpha
$\in$	Element of Set
%	Percent
γ	Photon/Gamma
π	Pi
$\sum$	Sum

CHAPTER I

INTRODUCTION

It's estimated there are about 21.5 billion internet of things (IoT) devices connected today. [2] The estimated number of devices by the end of 2021 is 46 billion. [2] Each of these devices poses a possible security risk. It's estimated that there were 34.9 million malware attacks against IoT devices in 2020. [3] A 4.8% increase over 2019. [3]

One of the most common malware applications are botnets. A botnet is a collection of bots working together to perform a task. A bot is a device with malware allowing a cyber attacker to remotely control the device.

The most common attack launched by botnets is a denial of service attack (Dos). A DoS attack is when a botnet sends a large number of requests to a server on the Internet. The goal of this attack is to slow down or completely block the service on the server. Besides launching a DoS attack, the attacker can use botnet malware to retrieve data on the victim devices and disrupt their operation. Therefore, detecting devices compromised with botnet malware is crucial to security on the internet.

Detection of devices compromised with botnet malware can be accomplished by observing their communications with the botnet master. Specifically, a bot must contact a server on the internet that is controlled by the attacker, called the command and control center (C&C). The C&C provides bots with instructions about what tasks to perform. To allow the bots to communicate with the C&C, the attacker needs to share with its bots the Uniform Resource Locator (URL) were the C&C is hosted. There are several ways to share the URL with the bots.

First, the attacker can simply host the server on a static URL and store the URL in the botnet malware. When a botnet needs to contact the C&C, it generates a Domain Name System (DNS) request to retrieve the Internet Protocol (IP) address of the C&C, and then establishes a connection to said IP address. However, after security researchers

identify the URL as a C&C, the URL is added to blacklists that are used by system administrators to block malicious traffic from their devices. Once a URL is added to a blacklist, the attacker loses control of its botnets.

To avoid detection, the attacker avoids using a static URL, and instead uses a domain generation algorithm (DGA) to choose dynamic URLs. In particular, The DGA pseudo-randomly generates a list of possible URLs for the C&C. The attacker runs the DGA to obtain the list of possible URLs and chooses one of them to host its C&C. The bots also generate the list of possible URLs and attempt to contact the C&C for each of the URLs on the list until it chooses the URL that is hosting the C&C. In this way if a URL is added to the blacklists, the attacker simply registers its server on a different URL on the list and the bots eventually choose the new URL to connect to the C&C.

Although DGAs allow attackers to move their C&Cs to different URLs, they also generate traffic with certain characteristics that can be used to detect bots within a network. Since bots attempt to connect to many domain names that are not registered with the DNS servers before finding the C&C. These requests include the domain names, which the firewall of the network can log. By analyzing these URLs we can detect bots in a network. Fig. 1 includes a network firewall that analyzes the network traffic of its devices to detect bots.

Machine learning algorithms have been shown to be effective in identifying URLs generated by DGAs. In the past few years, researchers have successfully classified URLs from real-world DNS requests into benign URLs and those generated by a DGA. [1] [4] [5] The machine learning approaches in the literature include Woodbridge et al. [1], who propose a Long Short-Term Memory algorithm that trains on known DGAs and achieves about a 90% detection rate. S. Schuppen et al. [4] propose a Random Forest algorithm to train on known DGAs and achieves about a 99% detection rate. L. Sidi et al. [5] propose an Autoencoder algorithm to train on known DGAs and achieves about a 72% detection rate. Unfortunately, machine learning approaches have a flaw. They are trained with the URLs generated by specific DGAs and are not able to detect URLs generated by unknown DGAs.

To help defenders improve the detection performance of existing machine learning algorithms, there have been efforts to design DGAs using machine learning algorithms specifically designed to bypass existing detection algorithms. The idea is to simulate the best possible DGA that an attacker may design and then design a detection approach that can detect its URLs. Specifically, researchers such as Spooren et al. [6] modify 33 different features of DGAs reducing FANCI [4] from 93.8% to 59.9% and J. Peck et al. [7] create an algorithm that takes known valid domains and modifies two characters and reduce FANCI [4] from 93.7% to 3.05%.

Unfortunately, existing machine learning approaches to design a DGA assume that the attacker has access to the detection algorithms. This is usually known as the white-box assumption. In practice, however, the attacker often does not have enough information about the DGA detection algorithms to launch a white-box attack. Instead, attackers usually only have access to the results of the detection algorithm, we call this the black-box assumption. In a black-box attack, the attacker can only observe if the URL generated by their DGA was classified as benign or malicious.

In this work, we propose a DGA based on deep-reinforcement learning called *Devil's DGA*. We take the name from the concept of a Devil's Advocate. A Devil's Advocate is a person, or machine learning algorithm in this case, that contests an idea in order to test the strength of the idea. *Devil's DGA* iteratively modifies the features of a DGA until it finds values that generate URLs that bypass the target detection algorithm. Our extensive experiment results show that *Devil's DGA* can reduce the detection accuracy of a target detection algorithm to 1.1% within 17 minutes and 54 seconds. *Devil's DGA* was also able to reduce the value to less than 50% detection rate within 6 seconds.

CHAPTER II

Related Works

There is a rich literature on using machine learning algorithms to detect botnets. More recently, researchers have also proposed machine learning to generate domain names that can bypass machine learning detection mechanisms. In this section, we summarize the state-of-the-art in both detection and generation of URLs for botnet operations.

2.1 Machine Learning Algorithms to Detect URLs Generated by DGAs

There are several machine learning algorithms that have been proposed to detect botnets by inspecting the URLs that potentially compromised computers request from DNS servers.

Woodbridge et al. [1] implement an LSTM that can differentiate in real-time domains requested by botnets from domains requested by legitimate users. To determine if a URL is legitimate or malicious, the LSTM takes as input a string representing a domain name, and outputs a percentage chance that that the domain is a legitimate domain. The LSTM [1] works by first learning the most probable sequence on characters that appear in legitimate URLs. Then, as it reads the characters of an input URL, it predicts the next characters according to the learned probabilities. If the predicted characters differ above a threshold from the observed ones, the URL is labeled as malicious. The Woodbridge et al. [1] LSTM achieves a 0.9993 area under the receiver operating characteristic (ROC) curve and about a 90% detection rate. In our experiments, we evaluate our proposed DGA algorithms against the LSTM in [1].

S. Schuppen et al. [4] propose to detect detect URLs generated by DGAs using random forests. Random forests are a series of decision tree layers that decide which path of decisions to take based on the features each layer receives. This algorithm detects botnets by examining non-existent domain name responses that devices receive on a network. The authors evaluate the detection performance of FANCI using 59 different

domain generation algorithms. They find that it can detect with 99% accuracy the URLs generated by most of the test DGAs. The authors also observe that FANCI can detect URLs generated by DGAs that were not used during its training step.

H.S. Anderson et al. [8] propose to improve the detection performance of random forests by using generative adversarial network (GAN) to increase the benign dataset to train a random forest classifier.

L. Sidi et al. [5] propose Helix, a DGA detector that uses an autoencoder combined with an LSTM to detect DGAs. By extracting patterns with the autoencoder and then using an LSTM to categorize those patterns, Helix is able to identify separate DGAs, or even just separate botnets using the same code, but installed with different seeds used to generate the domains.

GANs use a generator and a discriminator neural networks. The generator network creates new samples that follow the same distribution as the sample in a dataset. The discriminator examines those samples and labels them as a sample from the dataset or as a sample created by the generator. The label from the discriminiator is used by the generator to improve its sample in an iterative manner. Once the generator can create samples indistinguishable from the samples in the dataset, the authors use the generator to generate a larger dataset to train a random forest classifier. When comparing to a random forest classifier, the DGA detection from H.S. Anderson et al. [8] had two domain generation algorithms drop by 2% and 3% detection accuracy when using their method. However, the other eight domain generation algorithms had an equal or better, three of them as much as 4% increased detection rate.

2.2 Domain Generation Algorithms Designed to Bypass Detection Mechanisms

Besides designing detection algorithms, researchers have proposed domain generation algorithms to demonstrate how an attacker could modify its botnet malware to bypass existing detection mechanisms. In particular, Spooren et al. [6] model both the botnet master, who controls the parameters of the DGA, and the defender, who needs to analyze

the URLs generated by the devices in its network to detect botnets. To model the defender, they implement both the random forest-based FANCI [4] as well as the LSTM-detection [1]. To model the attacker, the authors propose modifying 33 different features, such as vowel ratio, character probability distribution, domain length, and many others, of a domain generation algorithm to match the characteristics of the Alexa top sites list.

The domain generation algorithm proposed by Spooren et al. reduced FANCI's [4] accuracy from 93.8% to only 59.9%, and the accuracy of Woodbridge et al. classifier [1] from 98.7% down to 85.5%. The DGA in Dual Arms race presented by J. Spooren et al. [6] can reduce the accuracy of existing detection mechanisms, they need to manually create their domain generation algorithm and assume that the attacker has full knowledge of the DGA detection mechanism to design their attack algorithm.

J. Peck et al. [7] propose Charbot a domain generation algorithm that modifies domains from the Alexa list by changing two characters in the domain name with a random equal distribution across the valid characters. The authors were able to reduce the detection rate of the LSTM [1] from 98.3% down to 15.5% and FANCI [4] from 93.7% down to 3.05%. The authors also retrained the LSTM with their Charbot and received an 81.08% detection rate, and a 22.26% detection rate after FANCI was trained with Charbot.

Z. Fang et al. [9] propose Deep Q-network to evade anti-malware engines (DQEAF), a deep reinforcement learning algorithm that modifies malicious software to evade anti-malware defense algorithms. The DQEAF modifies existing malware binary code by adding bytes, modifying specific bytes, or removing segments all together. The authors were able to achieve a 75% success rate in evading certain types of anti-malware engines in their experiment.

2.3 Deep Reinforcement Learning

Y. Li [10] and R. Sutton et al. [11] present in-depth reviews of the research on deep reinforcement learning. The deep reinforcement learning topics that are presented Deep Reinforcement Learning: An overview by Y. Li [10] and Reinforcement Learning: An

Introduction by R.S. Sutton and A.G. Barto [11] are used by this paper to design *Devil's DGA*.

CHAPTER III

Problem Formulation

In this section, we describe the considered problem and our assumptions about the attacker and the defender.

3.1 System Architecture

We consider a network formed by a set of devices that connect to the Internet through the same gateway device. The DNS requests from the devices are resolved by a domain name system (DNS) server. A system administrator logs the DNS traffic of the devices. In particular, it uses the gateway device to capture `NXDOMAIN` replies and logs the URL requested by the device. The system administrator runs intrusion detection systems that monitor the network traffic of the devices. The intrusion detection of the devices are trained to detect botnet traffic, including DNS requests and communications with a command and control center.

We assume a botnet master has successfully recruited devices in the network to its botnet. To communicate with the bots, the botnet master operates a command and control center (C&C). The IP address of the C&C is linked to a domain that is in the list of domains created by the bot's DGA.

The botnet master employs several techniques to prevent the system administrator from detecting the bots in its network. First, the botnet master continuously changes the IP address and domain name of its command and control center. By doing so, it prevents the system administrator from easily noticing anomalous traffic to the command and control center and blocking the bots' communications. Since the bots need to also know the domain name to communicate with the command and control center, the botnet master chooses its next domain name according to a domain generation algorithm (DGA) that is also loaded on botnet malware.

Second, the botnet master designs the DGA in such a way that it avoids DGA detection algorithms. Specifically, we assume the botnet master has offline black-box access to the algorithms the system administrator is using to detect DNS traffic generated by DGAs. Thus, the botnet master iteratively design a DGA that bypasses the intrusion detection of the system operator.

In this work, we investigate how the attacker can designs a DGA that can evade machine learning detection schemes.

3.2 Botnet Architecture

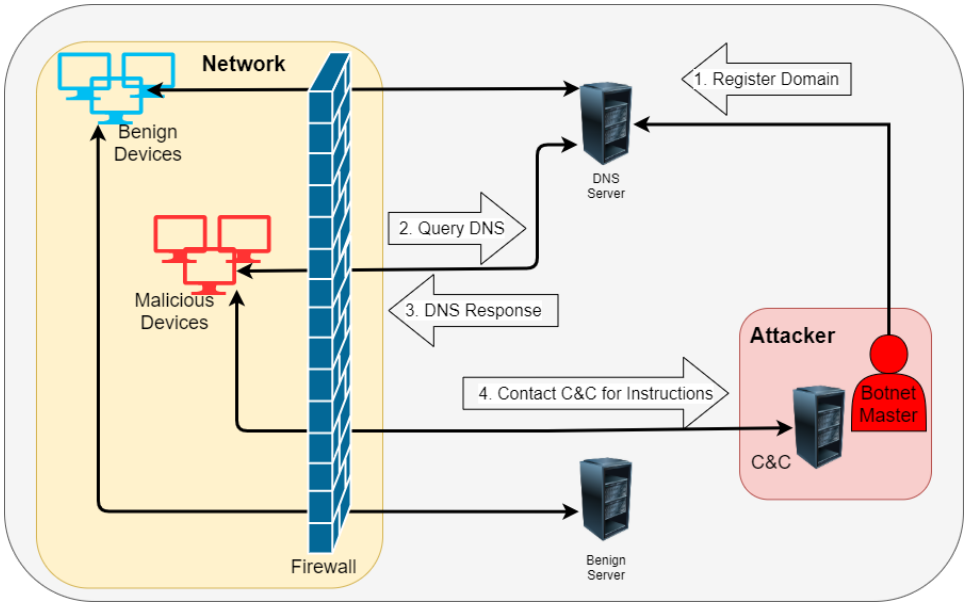

Figure 1: Botnet Architecture

We assume that the botnet master operates in the following manner as outlined in Figure 1. Let $\mathcal{G}(s)$ be the DGA that the botnet master uses to generate URLs. The URLs are generated according to the m-dimensional parameter vector $s \in \mathbb{R}^m$. The set of URLs generated by the DGA is denoted by $\mathcal{U} = \{u_1, u_1, ...u_D\}$. The botnet master chooses a URL $u_c \in \mathcal{U}$ uniformly at random and registers the IP Address of its C&C with u_c on the DNS server. To contact the C&C, a bot generate a DNS query with URL $u_t \in \mathcal{U}$. If $u_t \neq u_c$, the DNS server responds with a non-existent domain (NXDOMAIN) error message, which is

logged by the firewall. If $u_t = u_c$ the DNS Server responds with the IP address of the C&C to establish a connection. In practice, the URL u_t can be uniformly chosen at random by the bots or in sequence as it uses its local DGA to generate possible URLs.

CHAPTER IV

Deep Reinforcement Learning for DGA Detection Avoidance

In this section, we describe our proposed DGA based on deep reinforcement learning called *Devil's DGA*. Instead of directly generating URLs, *Devil's DGA* modifies an existing DGA and finds the optimal parameters of the DGA to generate URLS that can bypass a particular detection mechanism. Since existing detection mechanisms are trained with URLs generated by DGAs using constant parameter values, it is possible to modify DGAs to generate URLs with characteristics that will result in a false negative at the detection algorithm. To find the best parameter values, *Devil's DGA* uses a two-step algorithm. First, the algorithm uses the chosen DGA to generate a set of URLs using the current set of parameter values and feeds it to the target detection algorithm. Then, based on the ratio of URLs that bypass detection, it adjusts the parameters of the chosen DGA. The iteration continues granting a higher reward when more of the generated URLs avoid detection. We show *Devil's DGA* operation in Fig. 2.

4.1 A Markov Decision Process for DGA Evasion

Before delving into details about *Devil's DGA*, we first model the interactions between *Devil's DGA* and the target detection algorithm as a Markov Decision Process (MDP) [10, 11]. An MDP is defined by an action set, a state set, and a reward function. Specifically, let $\mathcal{A}$ be the set of actions that *Devil's DGA* applies to modify the parameters of the chosen DGA. At each iteration t, *Devil's DGA* chooses an action $\mathbf{a}_t \in \mathcal{A}$, where a_t is an n-dimensional vector, such that $\mathbf{a}_t \in \mathbb{R}^n$ (for all t).

We denote the set of states by $\mathcal{S}$, where each state $\mathbf{s}_t \in \mathcal{S}$ indicates the values of the chosen DGA's parameters. Each state is an m-dimensional vector such that $\mathbf{s}_t \in \mathbb{R}^m$. After *Devil's DGA* takes action a_t, the modified DGA advances from state s_t to state s_{t+1}. Once it reaches s_{t+1}, the modified DGA produces the set of URLs $\mathcal{U}(s_{t+1})_t = \mathcal{U}^k = \{u_1^k, u_2^k, ...u_D^k\}$.

Where D is a number of domains that are submitted at each t. The set $\mathcal{U}_t$ is fed by *Devil's DGA* to the target detection algorithm. The detection algorithm outputs the percentage $p_t \in [0, 1]$ of URLs in $\mathcal{U}_t$ that are classified as being produced by a DGA.

The rewards of *Devil's DGA* at each iteration are defined as the number of URLs in set U_t generated by the chosen DGA that are accepted as legitimate by the target detection algorithm. We denote the rewards at each step by $r_a(s_t, s_{t+1})$. The target p for a success is set as p_s. We set V_s as the success value received if $p_t \leq p_s$. We set V_f as the value received if $p_t > p_s$. These values are defined as $V_f < V_s$ and $V_f = - \in \mathbb{R}$.

The total return received by *Devil's DGA* at the end of one time step is shown in Equation 1.

$$r_t = \begin{cases} p_t <= p_s, & V_s \cdot (1 - p_t) \\ p_t > p_s, & V_f \cdot p_t \end{cases} \tag{1}$$

After the end of each t, *Devil's DGA* updates the Q Networks used to calculate a_t. *Devil's DGA* consists of a number of episodes e which consist of a run of t steps T times. After each e *Devil's DGA* resets s_0 to random valid values according to Algorithm 1 and 2 which are further described later in this work. *Devil's DGA* runs a number of e equal to E.

The goal of *Devil's DGA* is to learn a policy $\pi : S \rightarrow A$ which maximizes its expected episode reward, which is denoted by $\mathbb{E}[r_1]$.

To find the policy π, we first define the state-value function, which models the expected rewards of *Devil's DGA* when it starts from a given state s, and then takes action a following π thereafter. Specifically, the state-action value function is $Q^\pi(s_1, a_1) = \mathbb{E}[r_1 | s_1 = s, a_1 = a, \pi]$.

The state-value function Q^π can be expressed in a recursive manner using the Bellman equation shown in Equation 2.

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) Q^\pi(s_{t+1}, \pi(s_{t+1})) \tag{2}$$

4.2 A Deep Reinforcement Learning Architecture for tuning Domain Generation Algorithms

Devil's DGA architecture is outlined in Figure 2 and described in the following text.

Devil's DGA begins with a randomized s_0 that is fed to the modified DGA, specifically a modified Ramnit in this experiment. The Modified DGA produces u_t that is given to the DGA Detector. The DGA Detector then passes p_t back to *Devil's DGA*.

The *Devil's DGA* calculates the r_t from Equation 1. Then *Devil's DGA* passes r_t, s_t to the Deep Reinforcement algorithm Q Networks.

Devil's DGA uses two Q Networks as the Deep Reinforcement Learning Algorithm. One of these Q Networks is referred to as the Fast Q Network. The Fast Q Network trains with r_t, s_t each t. The other Q Network *Devil's DGA* uses is the Slow Q Network. At every interval I the Slow Q Network is updated with the w_t from the Fast Q Network. Values w_t are the weights of the values for the Fast Q Network. The value i counts up with each t step. When $i \geq I$ the Slow Q Network w_t is updated with w_t and i is set to 0.

Training the Fast Q Network happens at each t and is a process where Fast Q Network is optimized to find the lowest expected p for each action. This training process is defined by Equation 3.

$$Q^{\text{new}}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot max_a Q(s_{t+1}, a)) \tag{3}$$

At each t, a_t is calculated from the Slow Q Network. This a_t is performed on the Modified DGA before it transfers u_t again.

Devil's DGA continues this loop of t time steps until the T interval is reached. Once this interval is reached *Devil's DGA* ends the episode e and begins a new one. In the next e a set of s_0 is generated randomly and $t \leftarrow 0$. These e are run a number of times equal to interval E.

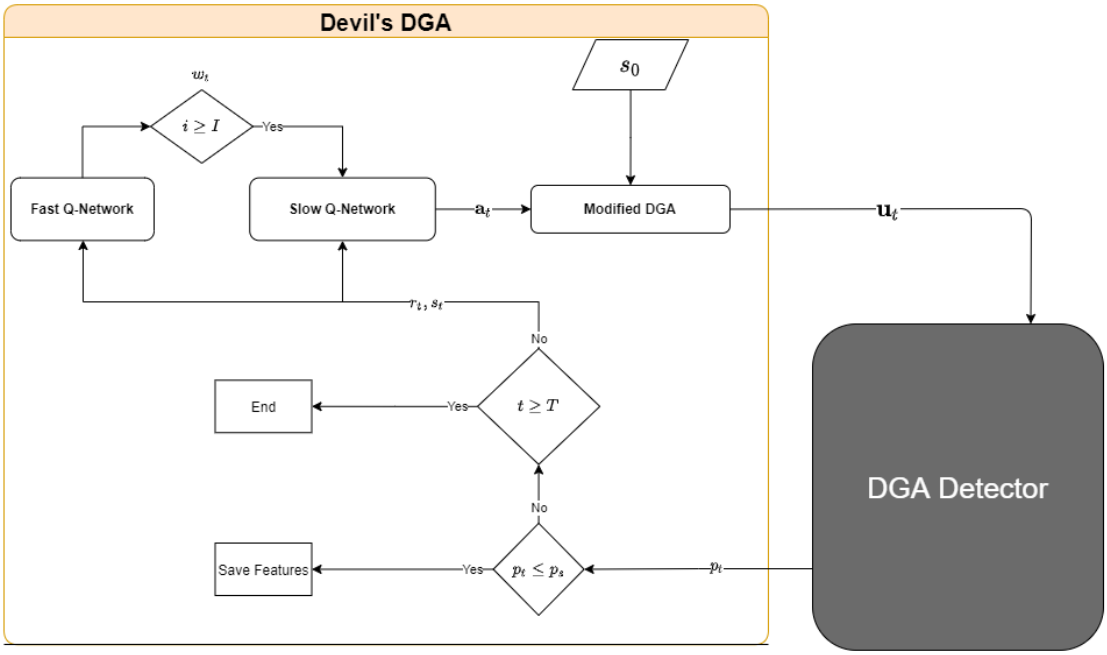

Figure 2: Devil's Advocate DGA

CHAPTER V

Experiments

In this section, we describe the experiment setup used to evaluate *Devil's DGA* and the experiment results.

5.1 Experiment Setup

We implement all components of *Devil's DGA* using Python. The deep reinforcement learning algorithm and neural networks are implemented using the TensorFlow library. We used a high performance computer with the specifications listed in Table 1.

Table 1: Hardware Specifications

Device	Specifications
Processor	Intel Xeon Gold 6240 (2.60 GHz)
RAM	32GB
Hard Drive	446 GB SSD
GPU	nVidia Tesla V100

We use DGA LSTM detection algorithm proposed by Woodbridge et al. [1] as the target detection algorithm, i.e., the detection algorithm that *Devil's DGA* aims to evade.

5.2 Overview of Experiment

To evaluate the performance of *Devil's DGA*, we implement it as follows. First, we obtain a set of domains from our Devil's DGA modified Ramnit and use them as input to the LSTM detection algorithm. The LSTM detection algorithm labels each URL either as benign or malicious. Based on the labels from the detection algorithm, we calculate the performance detector as a percentage of those domains that the detector predicts are malicious. This percentage is used as the reward to the deep reinforcement learning algorithm. Based on the reward and s_t, the deep reinforcement learning algorithm calculates the action to take at the next time step as follows: $a_t = argmax_a Q(s_t, a; \theta)$,

where a_t is the vector of actions that *Devil's DGA* applies to the modified Ramnit. We then update Ramnit based on a_t, and generate a new set of domains $\mathcal{U}_{t+1}$. The iteration continues until *Devil's DGA* reaches its maximum number of steps and episodes.

5.3 The actions *Devil's DGA* applies to Ramnit

Devil's DGA finds the best parameters to evade the target detection mechanism given a DGA. For our experiments, we use Ramnit [1], which is a DGA we customized to allow us to easily change the parameters that define the generated URLs. In our experiments, we use *Devil's DGA* to vary the percentage of vowels, constonants and numbers as well as the total length of the URLs generated by Ramnit. In the following, we explain in details how *Devil's DGA* sets the parameters of Ramnit.

5.3.1 Character Ratios

To control the proportion of vowels, consonants, and numbers, we use our proposed *Devil's DGA* to vary the probability Ramnit uses to choose a character. The original Ramnit uniformly at random chooses a character from the set of allowable characters. Our proposed *Devil's DGA* uses the observations of its rewards to set the probability that the next character chosen by Ramnit is a vowel, a consonant, or a number. After the modified Ramnit chooses the type of character, it will uniformly choose at random from the chosen subset of characters.

Specifically, we denote the *Devil's DGA* actions that set the character probabilities as follows: Let c_t be the probability of choosing consonants in the URLs at time t. Then, *Devil's DGA*, decides whether to increase or decrease this probability. Specifically, *Devil's DGA* action with respect to c_t at time t is $a_t^c \in \{C, -C\}$, where $C \in [0.1, 1]$. Hence, the resulting consonant probability used at time t by the modified DGA is given by $c_t = c_{t-1} + a_t^c$.

Similarly, *Devil's DGA* chooses the probability of choosing vowels in the URLs by increasing or decreasing the probability at the previous step. Let v_t be the probability of

choosing vowels at time t. Let *Devil's DGA* action with respect to the vowel probability be $a_t^v \in \{V, -V\}$, where $V \in [0.1, 1]$. Thus, the probability of choosing vowels at time t is given by $v_t = v_{t-1} + a_t^v$.

Consequently, the probability of choosing numbers in the URLs is given by $n_t = 1 - c_t - v_t$. However, as the sum of the consonant probability and the vowel probability may exceed one, *Devil's DGA* performs the operation identified in Procedure 1 to ensure that $n_t \geq 0$.

Algorithm 1 Control Character Ratios

```
if v_t ≤ 0.1 then
    v_t ← 0.1
end if
if c_t ≤ 0.1 then
    c_t ← 0.1
end if
if v_t ≥ 1.0 then
    v_t ← 1.0
end if
if c_t ≥ 1.0 then
    c_t ← 1.0
end if
if v_t > c_t then
    v_t ← c_t
else if c_t < v_t then
    c_t ← v_t
end if
```

For example, if the vowel probability is 0.5 and the consonant probability is 0.5 then the number probability is zero as shown in Figure 3.

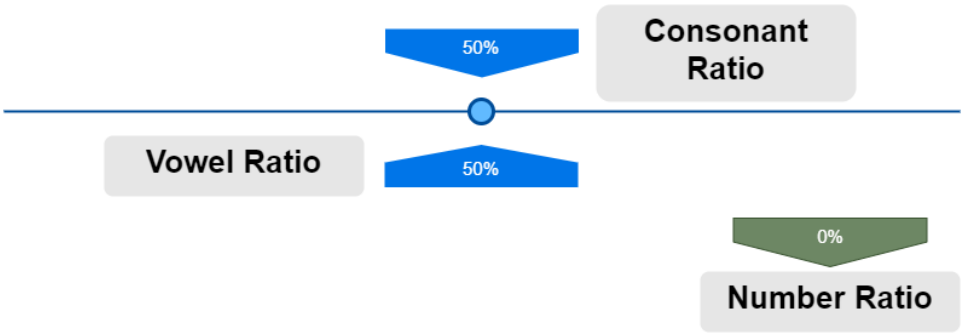

Figure 3: Zero Percent Chance of a number

Moreover, if the vowel probability is 0.4 and the consonant probability is 0.5 then

the number probability is 0.1 as shown in Figure 4.

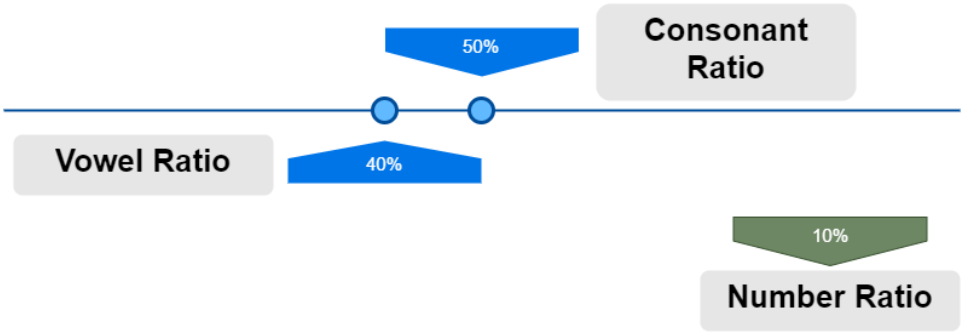

Figure 4: Ten Percent Chance of a number

5.3.2 URL Length

Devil's DGA also changes the URL length chosen by our modified Ramnit. The original Ramnit chooses the length of the URL uniformly at random from a constant range. To help Ramnit avoid detection, *Devil's DGA* chooses the range of URL lengths according to its reward observations.

Formally, we define the actions of *Devil's DGA* to set the length as follows: Let x_t be the maximum length of the URLs generated by Ramnit at time t. Then, the action that *Devil's DGA* chooses at time t is given by $a_t^x \in \{X, -X\}$, where X is a positive integer. Hence, the maximum length of URLs at time t is given by $x_t = x_{t-1} + X$.

Similarly, let the minimum length be y_t and the action of *Devil's DGA* be $a_t^y \in \{Y, -Y\}$, where Y is a positive integer. Then, the minimum length of generated URLs at time t is $y_t = y_{t-1} + Y$

To ensure that the maximum length of the URLs is longer than the minimum length, we perform Procedure 2 after finding x_t and y_t. Procedure 1 also ensures that $x_t \geq 1$ and $y_t \geq 2$ so that valid domains will always be created.

The parameters of the character ratios and the domain length variables are all initialized randomly at the beginning of every episode.

Algorithm 2 Control Domain Length

1: **if** $y_t \leq 1$ **then**
2: $y_t \leftarrow 1$
3: **end if**
4: **if** $x_t \geq 2$ **then**
5: $x_t \leftarrow 2$
6: **end if**
7: **if** $y_t > x_t$ **then**
8: $y_t \leftarrow x_t$
9: **else if** $x_t < y_t$ **then**
10: $x_t \leftarrow y_t$
11: **end if**

5.4 Data Sets

The J. Woodbridge et al. [1] LSTM-based detection algorithm was initially trained with the Alexa 1M dataset as the set of benign samples. The set of malicious samples were generated by using nine known domain generation algorithms, including Ramnit. By doing so we can closely replicate real-world detectors that are trained to detect URLs generated by multiple DGAs. During training, we generated 10,000 malicious domains from each of the domain generation algorithms, and extracted an equal number of benign domains from the Alexa 1M dataset. By using an equal number of benign and malicious samples we avoid biases in the URL labeling of the detection algorithm.

5.5 Experiment Results

For our results we tested Devil's DGA with multiple neural network configurations. We tested out neural networks with no hidden layers, 1 hidden layer, and 3 hidden layers. Each of these scenarios we ran with 24 nodes for each layer. We also ran a scenario with 3 hidden layers where there were 48 nodes in each layer as well. These scenarios are described further below. Table 2 describes the variables settings for each of these scenarios. Table 3 includes the run times for each of these scenarios.

Table 2: Devil's DGA Settings

Math Notation	Description	Setting
T	Time Steps	100
E	Episodes	20
I	Slow Q-Network Update Rate	5
D	# of Domains for Detector	1,000
γ	Discount Factor	0.99
α	Learning Rate	0.99

Table 3: Results Scenarios

Scenario #	# of Layers	# of Nodes per Layer	Run Time
1	None	N/A	24 Minutes and 13 Seconds
2	1	24	23 Minutes and 57 Seconds
3	3	24	26 Minutes and 2 Seconds
4	3	48	25 Minutes and 43 Seconds

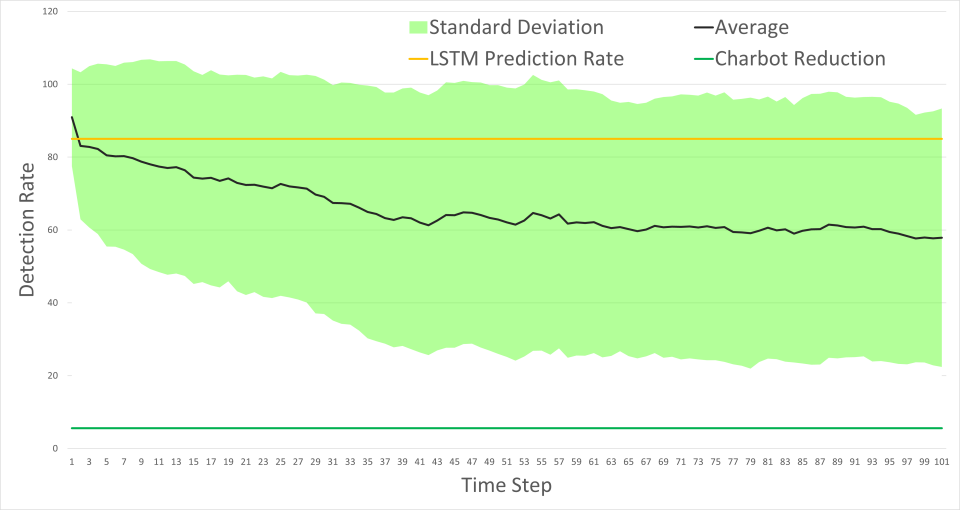

Figure 5: Scenario 1

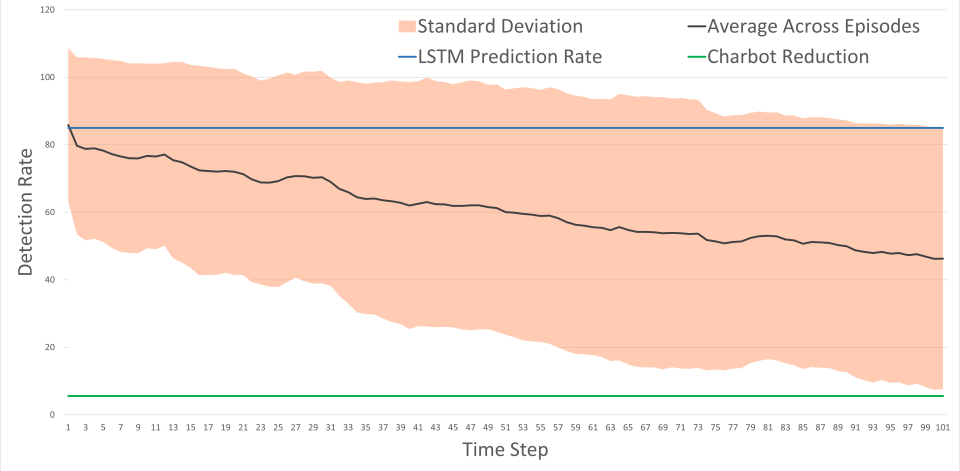

Figure 6: Scenario 2

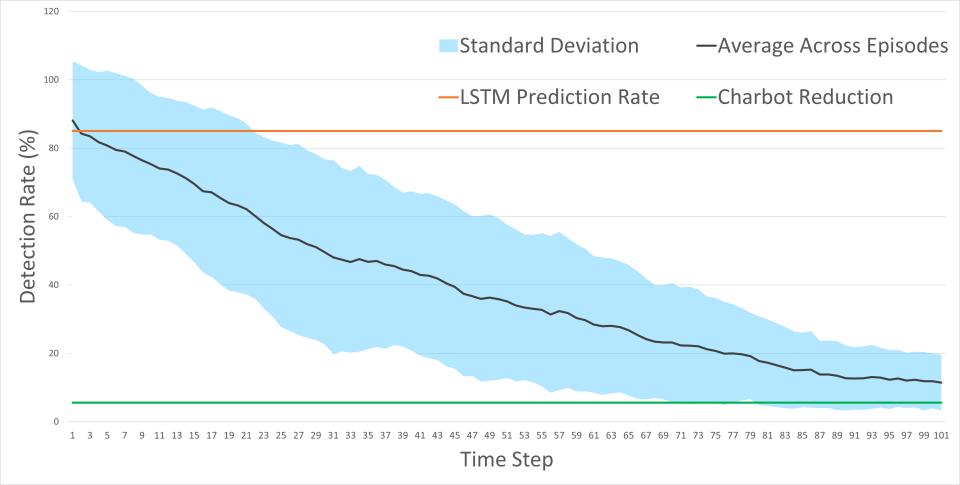

Figure 7: Scenario 3

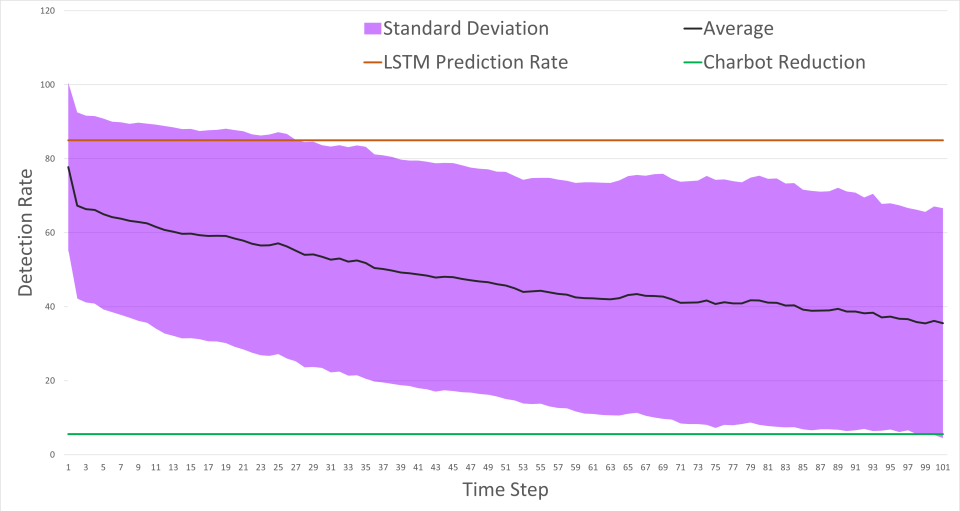

Figure 8: Scenario 4

In Fig. 5, we show Scenario 1 with the p of the *Devil's DGA* which ran in the time shown in Table 3. We also provide for reference the p the DGA predictor received in [1] and the p Charbot reduced the same DGA predictor to [7]. The model used in this run used no hidden layers, just the input and output layers of a network, effectively making this run not a Deep Learning Algorithm. We see that *Devil's DGA* is able to reduce the detection rate to an average of 57.9%. We also see that the lowest achieved detection rate during the entire run was 1.9%. The first value that *Devil's DGA* achieves less than the target value of 50% is 46.2% approximately 2 minutes and 13 seconds into *Devil's DGA* operation. The reason for these results is that there is effectively no deep learning applied to this algorithm. However, the algorithm can still learn to decrease the detection rate with the 1.9% being found. Later results will show that deep learning can improve on these results.

In Fig. 6, we show Scenario 2 with the p of the *Devil's DGA* which ran in the time shown in Table 3. We also provide for reference the p the DGA predictor received in [1] and the p Charbot reduced the same DGA predictor to [7]. The model used in this run used 1 hidden layer with 24 nodes. We see that *Devil's DGA* is able to reduce the detection rate to an average of 46.26%. We also see that the lowest achieved detection rate during the entire run was 1.4%. The first value that *Devil's DGA* achieves less than the target value of 50% is 49.8% approximately 6 seconds into *Devil's DGA* operation. The reason for

these results is that the single hidden layer introduces a rudimentary Deep Learning Algorithm which is able to improve on the results shown without hidden layers.

In Fig. 7, we show Scenario 3 with the p of the *Devil's DGA* which ran in the time shown in Table 3. We also provide for reference the p the DGA predictor received in [1] and the p Charbot reduced the same DGA predictor to [7]. The model used in this run used 3 hidden layers with 24 nodes to each layer. We see that *Devil's DGA* is able to reduce the detection rate to an average of 11.4%. We also see that the lowest achieved detection rate during the entire run was 1.5%. The first value that *Devil's DGA* achieves less than the target value of 50% is 44.7% approximately 27 seconds into *Devil's DGA* operation. The reason for these results is that the increased amount of hidden layers allows *Devil's DGA* to use Deep Reinforcement Learning to its advantage and improve on the previous versions of *Devil's DGA*.

In Fig. 8, we show Scenario 4 with the p of the *Devil's DGA* which ran in the time shown in Table 3. We also provide for reference the p the DGA predictor received in [1] and the p Charbot reduced the same DGA predictor to [7]. The model used in this run used 3 hidden layers with 48 nodes to each layer. We see that *Devil's DGA* is able to reduce the detection rate to an average of 35.8%. We also see that the lowest achieved detection rate during the entire run was 1.1%. The first value that *Devil's DGA* achieves less than the target value of 50% is 47.7% approximately 1 minute and 33 seconds into *Devil's DGA* operation. The reason for these results is that the increased amount of nodes prevents *Devil's DGA* from improving as quickly as the 3 Hidden nodes with 24 nodes at each layer. The 3 hidden layers, 48 nodes run at the end of the 100 steps still has a large deviation showing that it hasn't learned as quickly as the 24 node version.

CHAPTER VI

CONCLUSION

The previous work from J. Woodbridge et al. [1] was able to achieve an 85% detection rate against Ramnit when it was originally trained. *Devil's DGA* was able to decrease this detection rate using a deep reinforcement learning approach to as low as 1.1% and reduced to less than 50% in as little as 6 seconds. *Devil's DGA* successfully functions as a devil's advocate against the DGA Detector. Our experiments show that by implementing a deep reinforcement learning algorithm to modify features of a DGA, an attack could avoid detection from a machine learning based DGA Detector.

6.1 Future Works

The next step will be to use the features found by *Devil's DGA* to improve the DGA Detection algorithm. *Devil's DGA* creates a feature set for the Modified DGA that can be used to create new training data for the DGA Detector. This training data we expect could improve the DGA Detector and reduce the effectiveness of *Devil's DGA*.

We would like to see *Devil's DGA* applied to more DGA Detection algorithms. We believe that *Devil's DGA* will be able to find similar results against other detection algorithms.

We would also like to see *Devil's DGA* applied to more known DGAs. This work used Ramnit as the basis for its modified DGA, however it is likely that other DGAs could also be used in the same manner.

G. Dulac-Arnold et al. [12] discuss a method for using deep reinforcement learning with a large number of discrete action spaces. Current methods become very unwieldy for applications that require a large number of actions and the work of G. Dulac-Arnold et al. may be useful for expanding *Devil's DGA*.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," *arXiv preprint arXiv:1611.00791*, 2016.

[2] N. G., "How many iot devices are there in 2021? [all you need to know]," *Techjury*, 2021.

[3] S. Cook, "Malware statistics and facts for 2021," *Comparitech*, 2021.

[4] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer, "{FANCI}: Feature-based automated nxdomain classification and intelligence," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1165–1181, 2018.

[5] L. Sidi, Y. Mirsky, A. Nadler, Y. Elovici, and A. Shabtai, "Helix: Dga domain embeddings for tracking and exploring botnets," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2741–2748, 2020.

[6] J. Spooren, D. Preuveneers, L. Desmet, P. Janssen, and W. Joosen, "Detection of algorithmically generated domain names used by botnets: a dual arms race," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 1916–1923, 2019.

[7] J. Peck, C. Nie, R. Sivaguru, C. Grumer, F. Olumofin, B. Yu, A. Nascimento, and M. De Cock, "Charbot: A simple and effective method for evading dga classifiers," *IEEE Access*, vol. 7, pp. 91759–91771, 2019.

[8] H. S. Anderson, J. Woodbridge, and B. Filar, "Deepdga: Adversarially-tuned domain generation and detection," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pp. 13–21, 2016.

[9] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, and H. Huang, "Evading anti-malware engines with deep reinforcement learning," *IEEE Access*, vol. 7, pp. 48867–48879, 2019.

[10] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

APPENDIXES

APPENDIX A

Background on Deep Learning

A neural network contains several layers of features, where each layer breaks down information in a way that the algorithm can understand. The first layer of the neural network takes in the features of the item that the neural network is learning about. Each successive layer keeps some of the features from the previous layer and stores them for subsequent layers. The final layer of the neural network's features will be the output of the neural network. This output from the neural network is a percentage of some answer or response. For instance, a neural network can be used to identify pictures of cats versus pictures of dogs. The input layer would take the pixel values of the picture and then at the end there would be two output features, the percentage likelihood that the picture is of a cat and the percentage likelihood that the picture is of a dog.

Deep reinforcement learning in particular is a machine learning technique that creates a neural network with a policy of actions. Deep reinforcement learning focuses on a trial and error approach where it attempts to provide a prediction of which action to take would provide the best outcome. The neural network learns how to best respond to the situation it is presented with by receiving a reward for having achieved the correct prediction.

Reinforcement learning has been around for many years, however recent advances in deep learning have caused a resurgence in interest and applications for reinforcement learning. Deep learning allows the neural network to automatically train itself without human input by using gradient descent. Deep learning uses gradient descent to start at a random location and then identify the first step to take towards the optimal solution. Using elements of deep learning and reinforcement learning there has been enormous advances made in machine learning techniques in recent years.

A Machine Learning Training

Machine learning techniques have two phases. The training phase where the neural network(s) is trained, and then the implementation phase. The implementation phase is the normal running phase for the machine learning algorithm. It is also the phase that the algorithm is tested in. The training phase however, is where the algorithm is created and learns. This is the important step that defines how well the algorithm will work at its intended function. Training is not the same for every machine learning technique, but each of them have a training phase. The neural network begins with random settings, and then is updated to improve the results with each training iteration. Often the training phase includes thousands of iterations.