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

STOCKHOLM, SWEDEN 2021

Bayesian Reinforcement Learning Methods for Network Intrusion Prevention

ANTONIO FREDERICO NESTI LOPES

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

Bayesian Reinforcement Learning Methods for Network Intrusion Prevention

ANTONIO FREDERICO Nesti Lopes

Master's Programme, Machine Learning, 120 credits
Date: October 11, 2021

Supervisor: Kim Hammar
Examiner: Dr. Rolf Stadler
School of Electrical Engineering and Computer Science

© 2021 Antonio Frederico Nesti Lopes

Abstract

A growing problem in network security stems from the fact that both attack methods and target systems constantly evolve. This problem makes it difficult for human operators to keep up and manage the security problem. To deal with this challenge, a promising approach is to use reinforcement learning to adapt security policies to a changing environment. However, a drawback of this approach is that traditional reinforcement learning methods require a large amount of data in order to learn effective policies, which can be both costly and difficult to obtain. To address this problem, this thesis investigates ways to incorporate prior knowledge in learning systems for network security. Our goal is to be able to learn security policies with less data compared to traditional reinforcement learning algorithms. To investigate this question, we take a Bayesian approach and consider Bayesian reinforcement learning methods as a complement to current algorithms in reinforcement learning. Specifically, in this work, we study the following algorithms: Bayesian Q-learning, Bayesian REINFORCE, and Bayesian Actor-Critic. To evaluate our approach, we have implemented the mentioned algorithms and techniques and applied them to different simulation scenarios of intrusion prevention. Our results demonstrate that the Bayesian reinforcement learning algorithms are able to learn more efficiently compared to their non-Bayesian counterparts but that the Bayesian approach is more computationally demanding. Further, we find that the choice of prior and the kernel function have a large impact on the performance of the algorithms.

Keywords

Network Security, Reinforcement Learning, Bayesian Q-Learning, Bayesian Policy Gradient, Bayesian Actor-Critic, Markov Security Games

Sammanfattning

Ett växande problem inom cybersäkerhet är att både attackmetoder samt system är i en konstant förändring och utveckling: å ena sidan så blir attackmetoder mer och mer sofistikerade, och å andra sidan så utvecklas system via innovationer samt uppgraderingar. Detta problem gör det svårt för mänskliga operatörer att hantera säkerhetsproblemet. En lovande metod för att hantera denna utmaning är förstärkningslärande. Med förstärkningslärande kan en autonom agent automatiskt lära sig att anpassa säkerhetsstrategier till en föränderlig miljö. En utmaning med detta tillvägagångsätt är dock att traditionella förstärkningsinlärningsmetoder kräver en stor mängd data för att lära sig effektiva strategier, vilket kan vara både kostsamt och svårt att erskaffa. För att lösa detta problem så undersöker denna avhandling Bayesiska metoder för att inkorporera förkunskaper i inlärningsalgoritmen, vilket kan möjliggöra lärande med mindre data. Specifikt så studerar vi följande Bayesiska algoritmer: Bayesian Q-learning, Bayesian REINFORCE och Bayesian Actor-Critic. För att utvärdera vårt tillvägagångssätt har vi implementerat de nämnda algoritmerna och utvärderat deras prestanda i olika simuleringsscenarier för intrångsförebyggande samt analyserat deras komplexitet. Våra resultat visar att de Bayesiska förstärkningsinlärningsalgoritmerna kan användas för att lära sig strategier med mindre data än vad som kravs vid användande av icke-Bayesiska motsvarigheter, men att den Bayesiska metoden är mer beräkningskrävande. Vidare finner vi att metoden för att inkorporera förkunskap i inlärningsalgoritmen, samt val av kernelfunktion, har stor inverkan på algoritmernas prestanda.

Nyckelord

Nätverkssäkerhet, förstärkningslärande, Bayesian Q-Learning, Bayesian Policy Gradient, Bayesian Actor-Critic, Markov Security Games

Acknowledgments

I would like to thank the examiner Dr. Rolf Stadler for the shared expertise in research and all the feedback brought to this work. I would like to thank my supervisor Kim Hammar for the numerous inputs and clarifications made to the concretion of this work, in particular, with respect to the Bayesian Q-learning implementation and all the technical support. I would like also to thank the School of Electrical Engineering and Computer Science of KTH, and in particular to the Division of Network and Systems Engineering, where the resources for running the experiments were used.

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

A2C Advantage Actor-Critic

AC Actor-Critic

ALD Approximate Linear Dependent

BAC Bayesian Actor-Critic

Ber Represents the Bernoulli distribution

BPG Bayesian Policy Gradient

BQL Bayesian Q-Learning

CVaR Conditional Value at Risk

DBQPO Deep Bayesian Quadrature Policy Optimization

DDoS Distributed Denial of Service

DQN Deep Q Network

GPTD Gaussian Process Temporal Difference

HMM Hidden Markov Model

IDS Intrusion Detection System

idsgame Intrusion Detection System Game

IPS Intrusion Prevention System

IT Information Technology

KL Kullback-Leibler

MDP Markov Decision Process

PG Policy Gradient

POMDP Partially Observed Markovian Decision Process

PPO Proximal Policy Optimization

PPO-AR Proximal Policy Optimization Auto regressive

RBF Radius Basis Function

RL Reinforcement Learning

SDG Sustainable Development Goal

SGD Stochastic Gradient Descent

SKI Structured Kernel Interpolation

SVM Support Vector Machine

TD Temporal Difference or Time Difference

TRPO Trust Region Policy Optimization

UN United Nations

VaR Value at Risk

Chapter 1

Introduction

While the requirements for network security—confidentiality, integrity, and availability—might appear straightforward to the novice, the mechanisms to achieve those requirements are often complex [1]. Moreover, evolving attack vectors make network security an endless challenge. From the evolving dynamics between attackers and defenders, an artificial arms race can emerge. In the arms race, new network attacks are developed in response to improved network defense.

Reinforcement learning is the science of sequential decision making under uncertainty [2]. In recent times, reinforcement learning has achieved super-human performance in games [3, 4]. As a consequence of the contemporary breakthroughs in reinforcement learning, there is a growing interest to use reinforcement learning in practical domains, such as robotics control [5], networking [6], and network security [7].

One can think of many applications in network security that could benefit from reinforcement learning. For example, the evolving nature of network attacks testifies to a need for adaptive defense policies. Moreover, many applications in security could gain from decision making in computer time-scales—milliseconds or seconds—rather than human time-scales—minutes or hours. Consequently, although still in its infancy, the study of autonomy and self-learning in network security is an active area of research (Fig. 1.1) [8, 9].

However, a drawback of current state-of-the-art reinforcement learning methods is the large data requirement. To give an example, the AlphaGo Zero system used over 4 million games of self-play to reach superhuman performance in the game of Go [10]. For this reason, reinforcement learning is generally run in simulated environments where data can be gathered in a scalable manner. While the use of abstract simulations is appealing as it

takes little effort to simulate a wide range of scenarios, this approach does not, however, provide a reasonable indication of the validity of the method in industrial settings. Consequently, to be able to use reinforcement learning for practical intrusion prevention use cases, we must first develop reinforcement learning methods that can work with smaller amounts of data, which is the topic of this thesis.

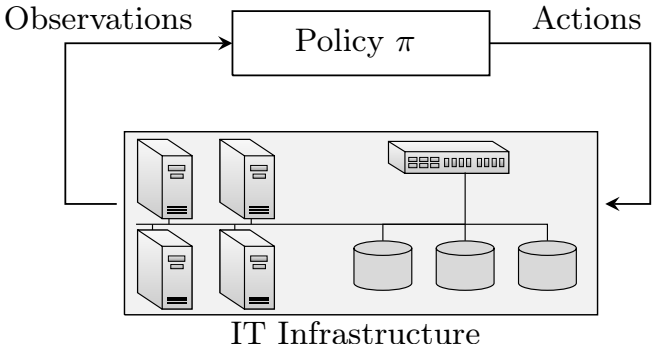


Figure 1.1 – A computer infrastructure that is controlled by an automatic policy that takes control actions based on measured observations from the infrastructure.

1.1 Use Case

We consider the use case of intrusion prevention. This use case considers a defender that owns an infrastructure that consists of a set of connected components (e.g. a communication network), and an attacker that seeks to intrude on the infrastructure. The defender's objective is to protect the infrastructure from intrusions by monitoring the network and patching vulnerabilities. Conversely, the attacker's goal is to compromise the infrastructure and gain access to a critical component. To achieve this, the attacker must explore the infrastructure through reconnaissance and attack components along the path to the critical component.

1.2 Research Questions

Automatic intrusion prevention involves a policy that uses infrastructure measurements to take actions for preventing network intrusions (Fig. 1.1). To find such policies, one approach is reinforcement learning. Reinforcement

learning is a general technique to produce effective control policies for different tasks by learning from data. Consequently, reinforcement learning systems are dependent on data to produce effective policies. To collect this data, a reinforcement learning system needs to take actions in the environment and evaluate the outcomes of the actions. This process of taking actions and evaluating the outcomes is called *exploration*. In the domain of intrusion prevention, the exploration process of reinforcement learning can be both costly and difficult to execute. For example, a single action (e.g. a network scan) can take up to an hour to run on a computing infrastructure. This presents an important challenge for intrusion prevention systems that are based on reinforcement learning.

One approach for reducing the amount of data required by reinforcement learning methods is the Bayesian approach where prior knowledge is combined with data collected through the exploration process. This approach contrasts with traditional reinforcement learning methods which do not use any prior knowledge. By using prior knowledge, the amount of data that needs to be collected with exploration can be reduced, which can allow to find effective intrusion prevention policies through reinforcement learning with less data. However, the Bayesian approach requires new methods for combining the prior knowledge about intrusion prevention with traditional reinforcement learning algorithms in an effective way, which is the topic of this thesis. Specifically, we aim to answer the following research questions:

- Is it possible to improve the efficiency of exploration in reinforcement learning algorithms by incorporating prior knowledge in the algorithm with a Bayesian approach and learn intrusion prevention policies with less data? To answer this question, we implement and evaluate three Bayesian reinforcement learning algorithms for intrusion prevention.
- How should prior knowledge of intrusion prevention be encoded in reinforcement learning algorithms to make exploration more efficient? We evaluate this question by developing several priors for intrusion prevention that we evaluate through extensive simulations.

1.3 Approach

Currently, most organizations' intrusion prevention rules are defined manually or by static rules or models, which puts a large burden on human operators. In this thesis, we investigate a learning-based approach to intrusion prevention.

Specifically, we model intrusion prevention as a game and use reinforcement learning to develop automated strategies that can co-evolve with a changing environment. We study Bayesian reinforcement learning methods where prior knowledge can be incorporated into learning-based intrusion prevention systems to learn strategies with less data.

1.4 Contributions

With this thesis, we make three main contributions. First, we implement three Bayesian RL algorithms and apply them to an intrusion prevention use case—Bayesian Q-Learning (BQL), Bayesian Policy Gradient (BPG) and Bayesian Actor-Critic (BAC). Second, we design two priors for both attacker and defender that can be incorporated with the BQL algorithm, four linear attacker priors and one risk-averse defender prior that can be incorporated with the BPG algorithm, as well as three attacker priors and one defender prior for the BAC algorithm. Third, we present extensive simulation results that evaluate the implemented algorithms and priors. Our results demonstrate that, by using the Bayesian reinforcement learning algorithms, we are able to learn more efficiently compared to the non-Bayesian counterparts (such as Q-Learning, Policy Gradient and Actor-Critic) and that the Bayesian approach is more computationally demanding per iteration. Further, we find that the choice of prior and the kernel function have a large impact on the efficiency of the algorithms.

Replication and Open Source: The implementation of the studied algorithms as well as the code for running the simulation used to produce the results in this thesis is open source and publicly available. In particular, we have constructed a Bayesian parity for the models in the repository `github.com/Limmen/gym-idsgame` which is available in the following GitHub repository: `github.com/FredericoNesti/idsgame`.

1.5 Structure of the Thesis

This thesis is organized as follows. Chapter 2 presents the theoretical background on reinforcement learning and Bayesian reinforcement learning. Related literature to this work is introduced and discussed in Chapter 3. Chapter 4 presents our formal game model of intrusion prevention. Chapters 5, 6 and 7 present our evaluation of three different Bayesian reinforcement

learning algorithms for the use case of network intrusion prevention. Chapter 8 presents an evaluation of our results to address and discuss the research questions. Finally, Chapter 9 presents our conclusions and future work as well as experiences and limitations from the implementation and ethical considerations.

Chapter 2

Background

This chapter covers the necessary theoretical background on network security, reinforcement learning, and Bayesian reinforcement learning.

2.1 Network Security

Cyber security can be divided into several sub-fields: (1) computer security, (2) network security, (3) web security, (4) software and system security, and (5) cryptography. Network security refers to the branch of cyber security that is concerned with ensuring the *confidentiality* (assures that confidential data is not disclosed), *integrity* (assures that data is only changed in a specified and authorized manner), and *availability* (assures that systems work promptly and service is not denied to authorized users) of networked systems and applications [1]. There is no single approach to achieve confidentiality, integrity, and availability, rather it depends on the type of network security problem. Common mechanisms used in network security are: firewalls, access control policies, network monitoring, intrusion detection systems, and intrusion prevention systems.

2.1.1 Intrusion Detection Systems

The purpose of intrusion detection systems is to "identify, preferably in real time, unauthorized use, misuse and abuse of computer systems by both system insiders and external penetrators" (Mukherjee et al. (1994)). Specifically, an Intrusion Detection System (IDS) serves as a first line of defense against network attacks. An IDS uses operational data such as log files and network traffic to diagnose the health of the system. Given the operational data, two

general methods are used to detect intrusions: signature-based detection and anomaly-based detection [7, 11].

Signature-based detection

Signature-based detection, (also called *misuse detection*) detects attack by so-called *attacks signatures*, i.e., patterns of network traffic or activity that could indicate a possible malicious intent [12]. For example, a pattern of bits in an IP packet (that indicates a buffer overflow attack), or the number of recent failed login attempts (related to identification and authentication problems) are examples of attack signatures.

Anomaly-based detection

Anomaly-based detection (also referred as to statistical anomaly detection or behaviour-based detection), uses statistical techniques to detect intrusions. This approach establishes the normal operation of the system and triggers an alarm whenever the system diverges from normal operation.

2.1.2 Intrusion Prevention Systems

Intrusion Prevention Systems (IPSs) extend IDSs by adding functionality to respond to detected intrusions, e.g. throttling a suspected distributed denial of service (DDoS) attack. Moreover, IPSs are often built around IDSs. For example, it is common that an IPS is a combination of an access control component (e.g. a firewall) and an IDS [13, 14].

2.1.3 Machine Learning Problems in Network Security

The application of machine learning to network security problems can be divided in two categories: detection problems and control problems. The first problem is usually tackled by means of supervised or unsupervised learning, while the latter is usually tackled with a reinforcement learning approach.

There are two major types of detection problems in network security, namely detection of network intrusions and detection of malware [11]. In both of these detection problems, there are *patterns* that can be learned and used to detect network intrusions and malware. Thus, standard machine learning techniques based on pattern recognition can be applied. Even though there has been a lot of research on applying machine learning to detection

problems in network security [15, 16, 17, 18, 19, 20, 21, 22], many intrusion detection systems and anti-malware programs are still rule-based [23, 24]. One reason for this is that using machine learning for security is inherently more difficult than using machine learning in other domains. This difficulty in applying machine learning to network security is due to the existence of an adversary. For example, an adversary could deliberately adjust the intrusion method to evade an intrusion classifier. This breaks the assumption of machine learning models that the training data and the operational data share the same data distribution [25]. In addition, another challenge with applying machine learning for security is the trade-off between false-positives and false-negatives. A machine learning system that has a high false-positive rate will quickly become ignored [11]. On the other hand, a system that has a high false-negative rate will be useless in its intended purpose [11].

In addition to detection problems in network security, there are also *control problems*. For example, the problem of intrusion prevention, which is the problem studied in this thesis. In the problem of intrusion prevention it is not sufficient to just detect an attack but it is also required to take active decisions, such as updating firewall policies or introducing rate-limiting. Another type of control problem in network security is the problem of automatically patching a found vulnerability in a system [26, 27]. To approach these problems with machine learning, the main method is reinforcement learning. The research on reinforcement learning methods for control problems in network security is in its infancy, some early works are [8, 28, 29].

2.2 Reinforcement Learning

In this section, we cover the theoretical background on Markov Decision Processes (MDPs), Markov games, and traditional reinforcement learning methods. We start by introducing the framework of MDPs, which provide a formal framework to define reinforcement learning problems.

2.2.1 Reinforcement Learning Preliminaries

This section defines the reinforcement learning objective and the formal model used to model reinforcement learning problems.

Markov Decision Processes

A Markov Decision process (MDP) is a controlled Markov reward process. In other words, consider a stochastic process* $\{S_t\}_t$ through time with stationary transition probabilities dependent only on the last realization, i.e., it obeys the *Markov property* $\mathbf{P}(S_{t+1}|S_t, S_{t-1}, ..., S_0) = \mathbf{P}(S_{t+1}|S_t)$. Further, associated with each state transition $S_t \rightarrow S_{t+1}$ is a reward R_{t+1}.

MDPs are defined by a tuple $\{\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}\}$, that is, a state space $\mathbf{S}$, a action space $\mathbf{A}$, a transition probability distribution $\mathbf{P} = \mathbf{P}(s_t \in \mathbf{S}|s_{t-1} \in \mathbf{S}, a_{t-1} \in \mathbf{A})$, which represent the controlled transition probabilities of the Markov reward process, and a reward function $\mathbf{R} : S \times A \; or \; S^2 \times A \rightarrow \mathcal{R}$.

The objective is to maximize the expected reward under a given policy π. For this goal formulation, there are three different classes of MDPs, depending on the time horizon of the problem. These are: finite-time horizon MDPs (Eq. 2.1), stationary MDPs with terminal state (Eq. 2.2) or infinite-time horizon MDPs (Eq. 2.3).

$$max \quad \sum_{t=1}^{T} \mathbb{E}[R_t(s_t^\pi, a_t^\pi)] \tag{2.1}$$

$$max \quad \mathbb{E}[\sum_{t=1}^{T_\emptyset - 1} R_t(s_t^\pi, a_t^\pi)] \tag{2.2}$$

$$max \quad \mathbb{E}[\sum_{t \geq 1} \gamma^{t-1} R_t(s_t^\pi, a_t^\pi)] \tag{2.3}$$

Here $T_\emptyset$ represents the time necessary to reach the terminal state. In addition, any MDP with a terminal state can be generalized into an infinite-time horizon MDP by redesigning some of the specified elements, such as the reward function. For the infinite-time horizon case, a discount factor γ is used to keep the sum of rewards finite and to give more importance to earlier rewards.

The Reinforcement Learning Problem

Reinforcement Learning (RL) problems are those concerned of finding a policy π^* that can maximize the expected cumulative (and possibly discounted) reward over a time-horizon T, that can be either finite or infinite. Formally, this problem can be modeled as the problem of finding a policy π^*

* A sequence of random variables.

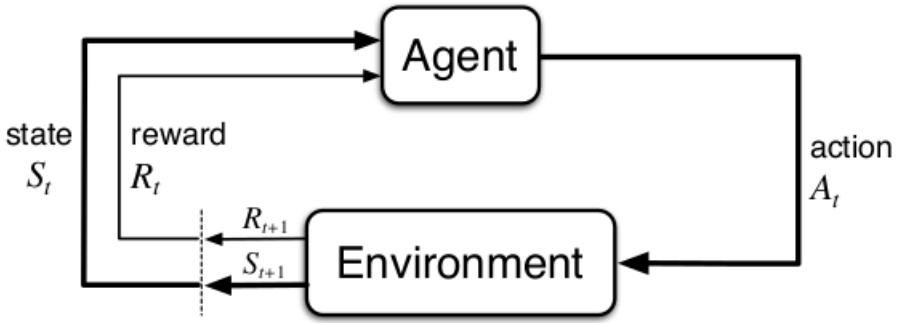


Figure 2.1 – Agent-Environment Interface; the image expresses the order for the dynamics of information flow for the state, action and rewards with respect to the Agent and the Environment. Image extracted from [2].

that maximizes the expected reward in an MDP:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ \mathbf{E}_\pi[\sum_{t=0}^{T} \gamma^t R_{t+1}] \tag{2.4}$$

where π^* is the optimal policy, R_{t+1} is the reward at time-step $t+1$, $\mathbf{E}_\pi$ is the expectation under policy π, and γ is the discount factor*. In this context, a policy is a function $\pi : \mathbf{S} \rightarrow \mathbf{A}$ that controls the MDP and that associates an action from the action space $\mathbf{A}$ of the MDP to a state from the state space $\mathbf{S}$ of the MDP. The optimal policy π^* is a policy that is better than or equal to any other policy with respect to the objective in Eq.2.4.

Figure 2.1 describes the main components of a RL problem. Starting from the Agent's node, which is currently in state S_t with reward R_t, it takes an action A_t that leads to a new state and reward S_{t+1}, R_{t+1}, respectively. Through interaction with the environment, the agent can learn by experience to reinforce sequences of actions that lead to larger rewards.

Depending on how the loop in Fig. 2.1 is realized, reinforcement learning algorithms can be categorized as on-policy, off-policy, model-based, or model-free. On-policy reinforcement learning algorithms are algorithms where the learning agent is updating the exact same policy as is used to take actions and trigger state-transitions in the environment. Conversely, off-policy reinforcement learning algorithms are algorithms where the policy used to take actions in the environment (the behavior policy) differs from the policy that is learned.

* Depending on the time-horizon and the formulation of the problem this discount factor could either be present or not.

Model-based algorithms are reinforcement learning algorithms that use a dynamics model of the environment to learn a policy. Model-free algorithms, on the other hand, are reinforcement learning algorithms that do not use an explicit model of the environment but rather learns through a trial-and-error method.

Markov Games: A Generalization of the MDP Model for Multi-Agent Reinforcement Learning Littman proposes *Markov games* as a model to generalize reinforcement learning problems to a multi-agent framework [30]. As opposed to MDPs that only contain one action space, Markov games include one action space for each agent, i.e. $\mathcal{A} = \{\mathcal{A}_i\}_{i<k}$. Moreover, while MDPs contain only one reward function, Markov games include one reward function per agent, i.e. $\mathcal{R} = \{\mathcal{R}_i\}_{i<k}$, in particular $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_k \rightarrow \mathbf{R}$.

Formally, a Markov game can be defined as a generalization of an MDP. Considering the case with two agents—A and D—a Markov game is defined by the tuple $\{\mathcal{A}_A, \mathcal{A}_D, \mathcal{T}, \mathcal{S}, \mathcal{R}_A, \mathcal{R}_D\}$, where $\mathcal{A}_A$ is the action space of agent A and $\mathcal{A}_D$ is the action space of agent D. Similarly, $\mathcal{R}_A$, is the reward function of agent A and $\mathcal{R}_D$ is the reward function of agent D.

2.2.2 Self-play

In the context of reinforcement learning, self-play is a method where two reinforcement learning agents learn by interacting with each other in a game. Specifically, in self-play, two agents play a game against each other with their current strategies and based on the outcome of the game, both agents update their strategies using a reinforcement learning method. This process of playing games and updating the agents' policies continues until both policies sufficiently converge. An example of the use of Self-play is AlphaGo Zero [31], where an agent is trained to reach superhuman performance in the game of Go by using self-play. Further, an example of using self-play for learning intrusion prevention strategies can be found in [8].

Although self-play often converges in practice, as reported in [31], [8] and [32], no formal guarantees for policy convergence in self-play have been established.

2.2.3 Temporal Difference Learning with Eligibility Traces (TD(λ))

This class of methods is one of the oldest in RL theory [2]. It will serve as a basis for most of the upcoming algorithms in this section. Time Difference methods is known to be an asynchronous version of stochastic approximation algorithms.

When updating estimates, the TD(λ) algorithm makes use of eligibility traces to weight the importance of rewards based on the temporal structure. Specifically, TD(λ) uses the weighted n-step λ return, which is defined as follows:

$$G_{t:t+n} := \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}) + \sum_{i=0}^{n-1} \gamma^i R_{t+i+1} \quad , \quad t \in 0, ..., T-n \tag{2.5}$$

2.2.4 Q-Learning

Q-learning is an off-policy TD control algorithm [2]. It leverages the stochastic approximation framework and updates the estimated Q-value of each state-action pair based on the difference in value between two successive time-steps. Further, Q-learning is an off-policy algoirithm which means that it uses a separate behavior policy to explore the environment. Due to the nature of the Q-function, Q-learning is restricted to discrete state-action spaces. The update in Q-learning is defined as follows:

$$Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha_{n^(t)}(R_t + \lambda \max_{b \in A} Q^{(t)}(s_{t+1}, b) - Q^{(t)}(s_t, a_t)) \tag{2.6}$$

ϵ-soft policies

The Q-function allows for the implementation of ϵ-soft policies. Specifically, the ϵ-soft policy using the Q-function as defined as follows:

$$\pi(s_t) = \begin{cases} arg\max_{a \in A} Q^{(t)}(s_t, a) & \text{with prob. } 1-\epsilon \\ Unif(A) & \text{with prob. } \epsilon \end{cases} \tag{2.7}$$

2.2.5 Policy Gradient Methods

Policy Gradient methods are based on the *Policy Gradient Theorem* (see Appendix B). The different methods introduced here - such as REINFORCE, REINFORCE with baseline, actor-critic methods and trust tegion methods (e.g PPO) —are all based on the policy gradient theorem but differs in the derivation of the gradient of the policy with respect to the objective of maximizing the expected cumulative reward.

Williams' REINFORCE Algorithm

REINFORCE algorithm was first introduced by Williams (1992). Its name originates from "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility" [33]. The central equation of REINFORCE is as follows:

$$\Delta w_{ij} = \alpha_{ij}(R - b_{ij})e_{ij} \tag{2.8}$$
$$where\ e_{ij} = \frac{\partial ln g_i}{\partial w_{ij}}$$

REINFORCE is an on-policy algorithm based on the policy gradient. In this algorithm, a policy is parameterized with a set of parameters that are updated using stochastic approximation (e.g. SGD) and the gradient defined by the policy gradient theorem is estimated by sampling from the environment. One problem with this method is that the gradient estimate tends to have a high variance. Because of this problem, research efforts have been made to reduce the variance of the gradient estimates. One common technique to reduce the variance of the gradient estimates is to subtract a baseline from the sample returns.

Actor-Critic Algorithms

Actor-critic algorithms are a combination of actor-only methods and critic-only methods [34, 35]. In the context of actor-critic methods, the role of the critic is to approximate the value function of the policy implemented by the actor. The critic can either estimate the standard value function V^π or the so-called advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, as done in the Advantage Actor-Critic (A2C) algorithm.

Both the critic and the actor is assumed to be implemented by

parameterized functions whose parameters can be modified using gradient-based updates.

Figure 2.2 illustrates a general actor-critic algorithm and its relation to the environment. In particular, the agent is represented by the policy and value function. A time-difference error is used to update both the policy and the value function. The actor decides which action to take, whereas the critic gives a feedback about the "goodness" of the action.

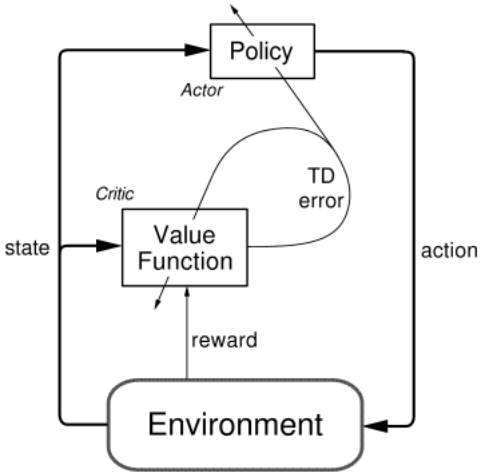


Figure 2.2 – Schematization of Actor-Critic methods with respect to the Environment. Image extracted from Towards Data Science website*.

When estimating the policy gradient, actor-critic methods can either use the standard sample returns or the *n-step* returns weighted by *eligibility traces*. The policy gradient with n-step returns can be formulated as:

$$\nabla J(\theta_t) = \mathbb{E}[(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla log\pi(A_t|S_t, \theta)] \qquad (2.9)$$

In the above formulation, the critic network is parameterized by $\mathbf{w}$ and the actor network is parameterized by θ. For example, $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$ comes from the critic, whereas $\nabla log\pi(A_t|S_t, \theta)$ is maintained by the actor.

* https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Proximal Policy Optimization (PPO)

The Proximal Policy Optimization (PPO) algorithm is a policy gradient algorithm that is a successor to the TRPO algorithm [36]. While PPO and TRPO are policy gradient algorithms just like e.g. REINFORCE, they use the policy gradient differently. Standard Monte-Carlo policy gradient methods (e.g. REINFORCE) use the estimated policy gradient that is defined in Eq. 2.8 and Eq. 2.9 directly in the stochastic gradient ascent update of the policy (e.g. either vanilla SGD or Adam). PPO and TRPO on the other hand are *trust-region methods*. Thus, once a gradient estimate $\hat{\nabla} J(\theta)$ has been obtained, PPO and TRPO solves a constrained optimization problem to update θ_t. This additional complexity to the policy gradient update can in theory provide faster convergence as the gradients are used more efficiently. The idea of trust region methods is to approximate the objective f indirectly by optimizing some simpler objective, $\tilde{f}$, that is an approximation of f.

To enforce the trust region in TRPO and PPO we use a constraint on the KL-difference between successive updates to the policy parameters θ_{old}, θ, e.g. $D_{KL}(\theta_{old}, \theta) < \delta$. However since $\pi_\theta(a|s)$ is a distribution over the entire state space $\mathcal{S}$ it becomes impractical to solve the constrained optimization problem due to the large number of constraints. Instead, TRPO and PPO use a heuristic approximation to the constrained optimization problem by adding a penalty term $c \cdot D_{KL}(\theta_{old}, \theta)$ to the objective function (a form of regularization), where c is a coefficient term.

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\left(\sum_{u=t}^{T} r_t \right) - c \cdot D_{KL}(\theta_{old}, \theta) \right] \tag{2.10}$$

By having this constraint, it becomes possible to make multiple updates using a single gradient estimate, as the constraints (in theory) ensures that the policy does not diverge too much. In vanilla policy gradient methods, doing multiple updates using the same gradient is not recommended as the trust region is not enforced it is likely that using the same gradient for multiple optimization steps will lead to divergence and too large parameter updates.

PPO implements the constraint to the policy update by using a clipped surrogate objective. This objective corresponds to the expectation of the minimum between the conservative policy Iteration component from TRPO objective and an objective that clips the reward with respect to a hyperparameter ϵ [37], i.e., by doing $R_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$. The role of this new objective function is to penalize changes to the policy that do not retrieve a better reward. This objective is a lower bound on the objective from TRPO.

Its formulation is given as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(R_t(\theta)\hat{A}_t, \mathbb{1}_{[1-\epsilon,1+\epsilon]}(R_t(\theta))\hat{A}_t)] \tag{2.11}$$

$$\leq \hat{\mathbb{E}}_t[R_t(\theta)\hat{A}_t] := \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{2.12}$$

The min operation inside the expectation ensures that $L^{CLIP}(\theta)$ is, at most, the TRPO objective (referred before as Conservative Policy Iteration component). To account for the fact that the advantage function could assume both positive and negative values, the clipping is defined to be $\mathbb{1}_{[1-\epsilon,1+\epsilon]}(R_t(\theta))$. In addition, PPO's objective can often be combined with an entropy bonus to incentivize extra exploration.

2.3 Bayesian Learning

This seciton contains an overview of Bayesian learning with a focus on methods that are related to the Bayesian Reinforcement Learning approaches described in the next section.

2.3.1 Bayesian and Bayes-Hermite Quadratures

Integral computation is key when handling marginal distributions and expectations, two concepts with rich applications in Bayesian learning. One approach for doing such computations is to use Monte-Carlo methods. However the Monte-Carlo methods have some drawbacks, such as higher variance and high sample quantity requirement. O'Hagan (1991) addresses a generic integral computation by leveraging the use of a Bayesian formulation. His technique is called Bayes-Hermite Quadrature, which is the correspondent problem of computing an integral of the form $\int f(x)g(x)dx$ by establishing a prior to one of the functions f or g. In this work, as will be seen later, these techniques are used to incorporate prior knowledge to RL algorithms.

Consider the following integral as our computation problem: $k = \int h(x)dx$. In order to formulate it statistically, one would like to consider a model that links k to the various $h(x)$. Treating each of the $h(\cdot)$ as a random variable, it is possible to assign a prior knowledge to it which in turn would yield a *posteriori* computation of k.

One could rewrite the integral as $k = \int g(x)d\mu(x)$, where $\mu(x)$ is the measure associated with $g(x)$. If one consider the probabilistic property

that the density distribution should sum up to 1, then we can consider that $f(x) = \frac{g(x)}{k}$ is a density, since $1 = \int f(x)d\mu(x)$. The sequence of variables $g(x)$ is generally modeled as a linear regression with a Gaussian process of white noises. The *posteriori* is then obtained by combining, via Bayes rule, the previous described priors with the "designed points".

Bayes-Hermite algorithm, on the other hand, aims at joining Gauss-Hermite quadrature with the previous described Bayesian quadrature. The estimation of Bayesian quadrature is itself dependent on the estimation of other integrals and therefore there is the risk of degeneration into infinite regress in case some of these base integrals are not analytically feasible [38]. To address this issue, the author explores the analytical viability of the formulation to the Bayesian quadrature to leverage the use of normal distribution specifications so as to derive closed expressions to all of the updates.

2.3.2 Kernel Methods and Gaussian Processes

Kernel methods aims to detect stable patterns from a finite data sample [39]. It relies on a correspondence between the data space and the feature space by leveraging inner products as a part of a mapping called *kernel function*. When the solution is given explicitly by solving the problem equations and finding the weights that satisfy a given relation, then this approach is called the *primal solution*. On the other hand, when one is interested in obtain a solution by leveraging only the information that comes from the data, that is "letting the data explain by itself", and, therefore finding an indirect solution, that is called as a *dual solution*. The point is that by transferring the focus of the techniques to find for a solution, it is possible to achieve a different complexity for the same task, that is $\mathcal{O}(n^3)$ for the primal and $\mathcal{O}(l^3)$ for the dual, where n is the number of data points and l is the dimension of them. Hence, if the number of data points is greater than the number of dimensions, then it is more efficient to find the dual solution. If that is not the case it is more efficient to find the primal solution.

An example to illustrate those concepts could be the linear regression case given in Shawe-Taylor et al. (2004). Consider that one is interested in solving $y = Xw$, which in linear regression typically has the solution $w = (X^TX)^{-1}X^Ty$. Assuming that the inverse $(X^TX)^{-1}$ exists, a dual representation could be expressed in the following way*:

* Note that each column of X represents a data-point and that X is a nxl matrix.

$$w = (X^T X)^{-1} X^T y = X^T X (X^T X)^{-2} X^T y = X^T \alpha \qquad (2.13)$$

In the above formulation w stands for the primal solution and α stands for the dual solution. The prediction function is then generally given as $g(x) = \langle w, x \rangle = \langle \sum_{i=1}^{l} \alpha_i x_i, x \rangle = \sum_{i=1}^{l} \alpha_i \langle x_i, x \rangle$. The matrix $G = XX^T$ is called the *Gram matrix*.

Non-linear solutions can be considered as well. In that case, an embedding map ϕ is used to convert nonlinear relations into linear. The Gram matrix is then generally defined as $G_{ij} = \langle \phi x_i, \phi x_j \rangle$. The *kernel function* is then defined to be the function that leverages the nonlinear mapping ϕ to an inner product feature space, such that $k(x, z) = \langle \phi(x), \phi(z) \rangle$.

Kernels can be designed to deal with very different nonlinear behaviours by utilising the feature space under the formulation of the inner product. An important aspect is that the use of the kernel method is particularly interesting whenever we have $n > l$. For the construction of the feature space it is desirable for it be separable and complete, and thus it is defined upon Hilbert spaces.

There are some properties to construct new kernels, where the reader can find more in the Appendix. As we will see later, these techniques will be important as a specification of a kernel matrix and can help in assigning a prior under a Bayesian quadrature formulation.

Gaussian Processes

Gaussian Processes are intimately related to kernel methods as the kernel matrix is used to specify a covariance structure of a modeling choice for data, in this case a Gaussian distribution. This is called a process in the sense that each of the data-points in time are defined to be a realization of a Gaussian distribution that correlates each time-step by a specification of the mean and the covariance is incorporated in its Kernel matrix, which translates the similarity of target points in its chosen structure (polynomial, linear, RBF, ...).

2.3.3 Sparsification procedures

The use of kernel methods can be quite demanding depending on its dimensions. When it comes to practical applications, it could be the case

that it is sometimes required to make the inversion of this matrix to compute Bayesian updates, for instance.

For this purpose, Ghavamzadeh and Engel* bring in their extensive work within Bayesian policy gradients, ways to sparsify the kernel computation, so as to achieve a better computational performance.

Engel's sparsification technique differs from the one brought by SVM algorithms. Instead of allowing the use of error-tolerant cost functions, which are often not maximally sparse, the author seeks to make use of the Representer Theorem and Mercer's Theorem to allow the use of a kernel trick that will aid the construction of a rate that is submitted to an if-clause and will allow for the choice of sparsification for a given sample. His idea is related to the feature space basis expansion by making use of the concept of linear independence from linear algebra. The reader can find descriptions of both algorithms in the Appendix.

The sparsification is achieved by considering a threshold ν to be determinant to "set linearly independent feature vectors that approximately span" [40] the feature space. Therefore, one is interested in finding coefficients $a = (a_1, ..., a_{m_{t-1}})'$ such that, for a given dictionary of samples $D_{t-1} = \{\tilde{x_j}\}_{j=1}^{m_{t-1}}$ and a new sample x_t:

$$\| \sum_{j=1}^{m_{t-1}} a_j \phi(\tilde{x}_j) - \phi(x_t) \|^2 \leq \nu \tag{2.14}$$

2.4 Bayesian Reinforcement Learning

State-of-the-art in Bayesian RL include both model-free and model based formulations. According to [41], under the model-free formulation, key algorithms are Bayesian Q-learning, Gaussian Process Temporal Difference (which are value-function based methods), Bayesian policy gradient, and Bayesian actor-critic. Under the model-based formulation we have the POMDP and Dynamic programming formulation of Bayesian RL. The Bayesian approach has also been resourceful to handle partial observable and multi-agent scenarios.

* In particular, Engel's PhD thesis has an extensive analysis for this matter [40].

2.4.1 Bayesian Q-learning

Bayesian Q-learning (BQL) [42] maintains an explicit distribution over the Q-values. This contrasts with Watkins' Q-learning (Watkins 1989) that does not maintain explicit distributions over the Q-values but rather use point-estimates. In BQL, the distribution over Q-values is initialized with a known prior and combined, using Bayes' theorem, with the likelihood function that comes from the specification of the modeling choice for Q-values data.

Below, we begin by explaining how prior and likelihood information could be combined to generate a posterior using Bayes' theorem. Then, we briefly address the assumptions of BQL derivation to motivate modeling choices.

Bayes' rule (equation 2.15) is useful for defining a posterior distribution as a function of the likelihood and the prior distributions, as the following equations 2.16 and 2.17 illustrate:

$$\mathbf{Posterior}(\mu, \sigma|D) = \frac{\mathbf{P}(\mu, \sigma, D)}{\mathbf{P}(D)} \qquad (Bayes'\ rule) \tag{2.15}$$

$$= \frac{\mathbf{Likelihood}(D|\mu, \sigma)\mathbf{Prior}(\mu, \sigma)}{\mathbf{P}(D)} \tag{2.16}$$

$$\propto \mathbf{Likelihood}(D|\mu, \sigma)\mathbf{Prior}(\mu, \sigma) \tag{2.17}$$

This technique aims to learn the Q-function by reducing the uncertainty about the expectation of a distribution over the sum of discounted rewards [41]. In the following we will assume the following three assumptions.

Assumption 1: The rewards, as a function of the states and actions, are normally distributed. This means that it suffices to specify the mean and precision* $\mu_{s,a}$ and $\tau_{s,a}$.

Assumption 2: The priors over $\mu_{s,a}$ and $\tau_{s,a}$ are independent for different states and actions.

Assumption 3: The prior $p(\mu_{s,a}, \tau_{s,a})$ is a normal-gamma distribution with hyperparameters $\langle \mu_0, \lambda, \alpha, \beta \rangle$, i.e., $p(\mu, \tau) \propto \tau^{\frac{1}{2}} e^{-\frac{1}{2}\lambda\tau(\mu-\mu_0)^2} \tau^{\alpha-1} e^{\beta\tau}$.

The dynamic of the Bayesian Q-learning algorithm is given as follows. At each iteration it is given a prior to the agent regarding the parameters of

* The inverse of the variance.

the Q-distribution. Then, an action is sampled considering the information that comes from this prior. Finally, an update to the prior distribution of the parameters is made and the new posterior is the prior of the next iteration.

In summary, the BQL algorithm uses $\mathbf{Likelihood}(D|\mu,\sigma)$ as the target for action samples, which is defined with respect to the current values of μ and σ*. The $\mathbf{Posterior}(\mu,\sigma|D)$ is computed to update the parameters of the Q-distribution and $\mathbf{Prior}_{t+1}(\mu,\sigma) = \mathbf{Posterior}_t(\mu,\sigma|D)$. For example, considering the assumptions in the beginning of this subsection, we have:

- $\mathbf{Prior}(\mu,\sigma) \sim Normal - Gamma(\mu_p,\sigma_p,\alpha_p,\beta_p)$;
- $\mathbf{Likelihood}(D|\mu,\sigma) = Normal(\cdot|\mu,\sigma)$;
- $\mathbf{Posterior}(\mu,\sigma|D) \sim Normal - Gamma(\mu_P,\sigma_P,\alpha_P,\beta_P)$ such that the tuple $(\mu_P,\sigma_P,\alpha_P,\beta_P)$ depends on D.

Practical issues

Since the integral of the posterior above does not have a closed form we approximate the posterior using some variational Bayes technique, e.g. by minimizing the KL-divergence:

$$G(D(s,a)|d) \triangleq \mathbf{P}[D(s,a)|d] \tag{2.18}$$

$$G(D(s,a)|d) \approx Q(D(s,a)) \tag{2.19}$$

$$D_{KL}(Q||G) \triangleq \sum_{s,a} Q(D(s,a)) log \frac{Q(D(s,a))}{G(D(s,a),d)} \tag{2.20}$$

2.4.2 Bayesian Policy Gradient

By leveraging the techniques of Bayes Quadrature, a posterior reward can be combined with the score function to compute a posterior of the gradient of the expected reward. The prior over the reward leverages the dual space of the data and it is given a Gaussian distribution with a covariance term that is proportional to a kernel matrix depending on the size of the data.

A further step on this approach would be to establish a prior on one of the components of the gradient of the loss function upon which the REINFORCE algorithm is defined in (as described in the beginning of this section). The key

* i.e., from the previous computed posterior

formulations are as follows for the prior of the rewards, the posterior of the rewards and the posterior for the gradient of the loss function.*

Reward Prior

$$f(\cdot) \sim N(f_0(\cdot), k(\cdot,\cdot)) \tag{2.21}$$

Reward Posterior

$$\mathbf{E}(f(x)|D_M) = f_0(x) + k_M(x)^T C_M(y_M - f_0) \tag{2.22}$$

Loss-Gradient Posterior

$$\mathbf{E}(\rho|D_M) = \rho_0 + z_M(x)^T C_M(y_M - f_0) \tag{2.23}$$

$$with\ \rho_0 = \int f_0(x)p(x)dx \tag{2.24}$$

An additional element to introduce is the the kernel function $k(\cdot,\cdot)$. Given the modeling choice of putting a prior over the rewards, the Fisher kernel can be used to make the following formulation: $k(\varepsilon_i, \varepsilon_j) = u(\varepsilon_i)^T G^{-1} u(\varepsilon_j)$, where G is the Fisher Information matrix and $u(\varepsilon_i)$ is the score function of for path i [43]. In practice, the Fisher kernel can also be given as $k(\varepsilon_i, \varepsilon_j) = u(\varepsilon_i)^T u(\varepsilon_j)$ [39].

Alternatively, it is possible to use other kernel functions such as the Gaussian kernel or the Cauchy kernel that make use of the score function. Cauchy kernel, on the other hand, has a heavier tail and, therefore, gives more importance to smaller values if the gradients have very small components.

Finally, the sparsification procedure described in [40] allows for a faster Bayesian algorithm implementation with the cost of a low loss in performance. M paths are sampled, where an Approximate Linear Dependence (ALD) condition check for linearly independent feature vectors. Hence, for a new path be accepted and incorporated into the training procedure we check the threshold $\delta_t > \nu$, where $\delta_t = k_{tt} - \tilde{k}_{t-1}(x_t)^T \tilde{K}_{t-1}^{-1} \tilde{k}_{t-1}(x_t)$. As the kernel can be interpreted as a covariance matrix, the ALD condition states that the variance of the current path minus the normalized covariance of the current path with the previous ones must be greater than a given threshold ν. This means if the new path x_t can provide with new relevant information for the algorithm or not.

* Formulations (1), (2), (3) and (4) are extracted from [43]

2.4.3 Gaussian Process Temporal Difference

Temporal Difference methods, use updates with respect to the successive gains of information. In other words, they minimize the TD error. On the other hand, Gaussian Processes use a Normally-distributed prior over consecutive realizations of a random variable, often by leveraging kernel methods and data's dual space. Consequently, Gaussian Process Temporal Difference methods (GPTD) [44, 45, 40, 41, 46] unify these two methods to define a Gaussian independent prior for the random variable for each time-step.

Engel et al. (2005) propose an approach which implements GPTD by modeling the value via discounted return. The objective we are interested to maximize is the total discounted rewards $D(x)$, given as:

$$D(x) = \sum_{i=0}^{\infty} \gamma^i R(x_i) | x_0 = x \quad with \quad x_{i+1} \sim p^{\mu}(\cdot|x_i) \tag{2.25}$$

That is, the total discounted reward is given the *policy-dependent state transition probability distribution* p^{μ}. In this sense, the authors refer to the MDP problem as having an *intrinsic source of randomness*. This is because the randomness of $D(x)$ is both dependent on the following states from x and the randomness in the rewards.

By using the definition that $V(x) = \mathbb{E}_{\mu}[D(x)]$ and using the stationarity assumption of the MDP it is possible to write $D(x_i) = R(x_i) + \gamma D(x_{i+1})$. From this, one can define three different stochastic processes. One for the rewards $\{R_t\}_t$, one for the value functions $\{V_t\}_t$* and another for a noise process $\{N_t\}_t$, such that $R_{t-1} = H_t V_t + N_t$, where:

$$H_t = \begin{bmatrix} 1 & -\gamma & 0 & ... & 0 \\ 0 & 1 & -\gamma & ... & 0 \\ \vdots & & & & \\ 0 & 0 & ... & 1 & -\gamma \end{bmatrix} \tag{2.26}$$

In equation 2.26, H_t represents a matrix that "produces" the temporal difference. Our prior choices are then with respect to the value function, since the rewards can be re-written as $R(x) = V(x) - \gamma V(x') + N(x, x')$. Finally, to define the prior and the posterior for the value function, consider K_t as the kernel matrix and $\Sigma_t = \sigma^2 H_t H_t^T$. We can then specify the prior, the likelihood and the posterior for the value function as follows:

* It is also possible to model the Q-function instead, as illustrated in [41].

- **A Priori:** $V_t \sim N(0, K_t)$;
- **Likelihood:** $V(x) = \mathbb{E}_\mu[D(x)]$;
- **A Posteriori:** $V_t^{post} \sim N(\hat{v}_t(x), p_t(x))$.

Where: $\hat{v}_t(x) = k_t(x)^T \alpha_t$, $p_t(x) = k(x,x) - k_t(x)^T C_t k_t(x)$, $\alpha_t = H_t^T (H_t K_t H_t^T + \Sigma_t)^{-1} r_{t-1}$ and $C_t = H_t^T (H_t K_t H_t^T + \Sigma_t)^{-1} H_t$. The online sparsification is applied similarly to the application described for BPG algorithm. Finally, next we address some assumptions brought by the authors when deriving the method.

Figure 2.3 exemplifies the recursive relations of GPTD learning. The noise variables ΔV are defined as $\Delta V(s) := D(s) - V(s)$. The observed variables are only the rewards. Given that the formulation of the value function is dependent on previous rewards by the Markovian property of the latent values, the prior to be defined over the rewards is, therefore, well-defined.

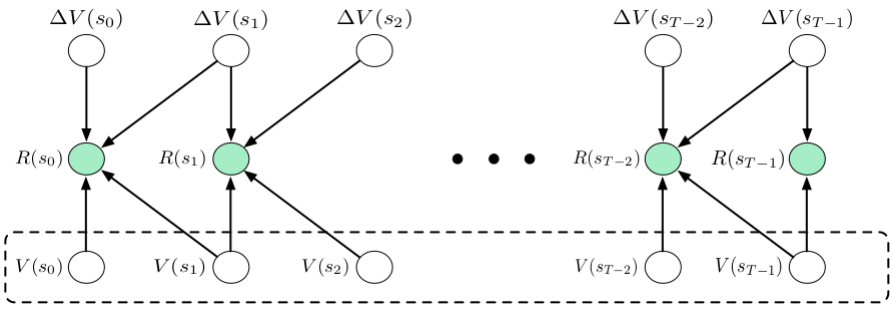


Figure 2.3 – A graphical model that illustrates conditional independencies GPTD learning. V represents the latent value variables, R represents the observable rewards and ΔV represents the noise variables. All the $V(s_t)$ variables should point to each other by arrows. This is represented by the dashed line box to avoid cluttering the diagram. Source of the figure: [46].

In order for GPTD be implemented by taking advantage analytical constructions of Bayesian learning it may requires several assumptions to be satisfied. Much of other work of Ghavamzadeh and Engel is to generalize GPTD to frameworks with more relaxed assumptions so as to drive to practical applications. These assumptions are the following:

Assumption 1: Residuals of the value-function must be possible to be modeled as a Gaussian Process.

Assumption 2: Each of the residuals of the value-function must be generated independently from each other.

2.4.4 Bayesian Actor-Critic

The Bayesian Actor-Critic (BAC) method [47, 46, 48, 40, 41] is an actor-critic method with a GPTD prior over the critic. As a policy gradient method, the objective is to maximize the total discounted reward by applying gradient ascent methods. Given that BAC models the critic in a Bayesian way, a prior is put over the Q-function. As the goal is to optimize the objective $\nabla\eta(\theta) = \int dsda\nu(s;\theta)\nabla\mu(a|s;\theta)Q(s,a;\theta)$, one can consider the following equations as key steps to make a Bayesian optimization of the objective:

Q Prior

$$Q(z) \sim N(\cdot, k(\cdot,\cdot)) \tag{2.27}$$

Q Posterior

$$\hat{Q}_t(z) = \mathbb{E}[Q(z)|D_t] = k_t(z)^T\alpha_t \tag{2.28}$$

Loss-Gradient Posterior

$$\mathbb{E}[\nabla\eta(\theta)|D_t] = \int dz g(z;\theta)k_t(\theta)^T\alpha_t \tag{2.29}$$

Where $\alpha_t = H_t^T(H_tK_tH_t^T + \Sigma_t)^{-1}r_{t-1}$ with r_{t-1} representing an observed reward, $z = (s,a)$ and $g(z;\theta) = \nu(s;\theta)\nabla\mu(a|s;\theta)$. The kernel K_t consists of two components: $k(z,z^{'}) = k_s(s,s^{'}) + k_F(z,z^{'})$. The first component $k_s(s,s^{'})$ is a kernel constituted from the states and the second component is the Fisher kernel. In this work we have decided to model both of them together with an RBF kernel (also inspired by the work of [49]). Note that the recursive relations of the Q-function posterior leads to a prior defined over the rewards. For that reason, in the following experiments' chapter we will define the prior by setting a "customized reward"*.

Complexity Tej et al. (2020) report that Bayesian Actor-Critic algorithms have complexity $\mathcal{O}(m^2n + m^3)$ for time and $\mathcal{O}(mn + m^2)$ for storage complexity, where m stands for the dictionary size and n for the number of parameters.

* This customized reward is a predefined heuristic upon which the agent might bias its decisions.

2.5 Theoretical Pairwise Comparison of Bayesian Reinforcement Learning Methods

Below we provide a summary of the differences between the Bayesian reinforcement learning algorithms under study.

BQL and BPG Even though policy gradient and tabular methods are notably different in nature, [50] shows a correspondence between policy gradient methods and soft Q-learning.

BQL and BAC Both the BQL algorithm and the BAC algorithm put a prior over the Q-function and model it as a normal distribution. The BQL algorithm models the prior over the Q-function for each state-action pair whereas the BAC algorithm models the prior over the Q-function as a multivariate normal distribution with covariance matrix given by the kernel function computed over the score functions of the dictionary of samples. The difference between the two algorithms is that BPG uses a parameterized function and relies on gradient-based learning techniques for optimization. The benefits of using policy gradient methods rely mainly on being able to handle larger state-action spaces and more complex scenarios.

BPG and BAC Both BPG and BAC are policy gradient methods that incorporate the Bayesian framework by using Bayes Quadrature and Gaussian Processes. However, they differ in the sense that BPG considers the trajectories as a basic observable unit, whereas BAC considers the system as being Markovian , i.e., the basic observable unit is the one-step system transition [46].

Chapter 3

Related Work

In this section, we describe the related work. This thesis extends prior work on reinforcement learning applied to network security, Bayesian reinforcement learning, and game theoretic modeling of security.

Reinforcement Learning For Intrusion Prevention For an overview of RL methods applied to network security problems, see the survey by Nguyen and Reddi (2020). Below we discuss the pior work on using reinforcement learning for intrusion prevention that most relate to this thesis.

Elderman et al. (2017) use a reinforcement learning approach to learn attack and defense policies in a cyber security simulation consisting of two agents —one defender and one attacker. This problem formulation is close to what is proposed in this work. To find policies, they evaluate several algorithms, namely: Monte-Carlo RL, combined with ϵ-greedy strategy, Softmax and upper confidence bounds (UCB); and Q-learning and MLPs applied together with stochastic approximation algorithms. To learn the policies, the agents play against each other and update their policies using one of the mentioned algorithms (but not necessarily the same). The results demonstrate that Monte-Carlo Softmax RL performs best as the defender and Monte-Carlo Softmax RL and UCB-based RL methods perform best as the attacker. Further, the authors explicitly mention that algorithms based on neural networks peform worse than tabular algorithms when training the defender.

Ridley (2018) introduces the concept of cyber resilience as being the capability of being able to maintain a certain level of performance despite the presence of adversaries. The author then poses a series of challenges for achieving such system performance, such as being capable of detecting

movement and presence of attacker, automating the reaction speed to high volume of alerts and mitigate imbalances in workload. Similar to the approach taken in this thesis, [9] also uses graph-based model and a RL-method to compute policies.

Our work differs from the above mentioned works by studying Bayesian reinforcement learning algorithms and by applying state-of-the-art reinforcement learning algorithms, e.g. PPO and BAC, rather than traditional ones like Q-learning.

Hammar and Stadler (2020) define a Markov game model of intrusion prevention and develop a self-play reinforcement learning method that automatically finds effective attack and defense strategies starting from zero prior knowledge [8]. The algorithm that they propose for the problem is PPO-AR (autoregressive action-sampling version of PPO). In this case, the policy $\pi(a, n|o)$ is implemented using two sub-policies by using two different neural networks, one for $\pi(a|n, o)$ and another for $\pi(n|o)$, the first for choosing the action and the second for choosing the node. Further, an extension to their approach is provided in [29], where intrusion prevention is modeled as an optimal stopping problem. The work in this thesis can be seen as a direct extension to the work by [8] as we use the same game model for our study. The work in this thesis differs from [8] in the following ways. First, we take a Bayesian approach, where we incorporate prior knowledge into the agents policies rather than starting from zero. Second, we focus primarily on learning against static opponents rather than self-play learning. Finally, we leverage the use of Bayes quadrature to allow for the incorporation of the prior knowledge with several extra samples taken from a same policy network.

Bayesian Reinforcement Learning: Deep Bayesian Quadrature Policy Optimization Tej et al. (2020) comment on Ghavamzadeh and Engel work on Bayesian policy gradients. In particular, the scalability problem is studied. An argument is put forward in favor of a lack of stochasticity treatment for gradient estimation. Therefore, their contribution is towards a distinctive kernel choice and implementation to account for a universal and faster Bayesian framework that is reproducible when combining the method with deep neural networks. Their technique is claimed to achieve significant improvement in complexity and return for vanilla and natural policy gradients and trust region policy optimization (TRPO) on 7 diverse MuJoCo domains [49].

Game-theoretic approaches A different game-model than the model used in this thesis is FlipIt. FlipIt is a two-player game played in continuous time where any player can take actions independently of rounds. The goal is defined as to take control of a resource. To symbolize this, a button is pushed to represent the transfer of the ownership. Subsequently, any player can remain several time-steps without knowing if it is keeping the ownership to itself. Pushing the button has a cost, which can define a utility function.

In the work by [51], the optimal strategies in the FlipIt game is studied under different scenarios. Examples of strategy-types considered are: non-adaptative, renewal, periodic, exponential, adaptative and no-play strategies. Feedback-types considered are: non-adaptative, last move and full history. Other information taken into consideration are: the initial information to be incorporated as the rate of play or knowledge of strategy, gains and benefits and views and history.

Chapter 4

Modeling the Intrusion Prevention Game

In this chapter, we describe our method. First, we describe the Markov game model that we use to model the intrusion prevention use case. Second, we describe how we run simulations of the game and how we evaluate our Bayesian reinforcement learning approach.

4.1 The Intrusion Prevention Game

We use the Markov game model defined in [8] to model intrusion prevention as a game. The game is one of two players—an attacker and a defender —that is played on a computer infrastructure. The right side of Fig. 4.1 shows the infrastructure underlying our use case. It is depicted as a graph that includes four network components.

The component N_{start} represents the attacker's computer and the remaining components represent the defender's infrastructure, where N_{data} is the component that the attacker wants to compromise. To achieve this, the attacker must explore the infrastructure through reconnaissance and compromise components on the path to N_{data}. At the same time, the defender monitors the network and increases its defenses to prevent the attacker from reaching N_{data}. In this adversarial process, both the attacker and the defender have a partial view of the network.

At the beginning of the game, the attacker knows neither the topology of the infrastructure nor its vulnerabilities. In contrast, the defender has complete knowledge of the network's topology and the vulnerabilities of its components, but cannot observe the status of attacks.

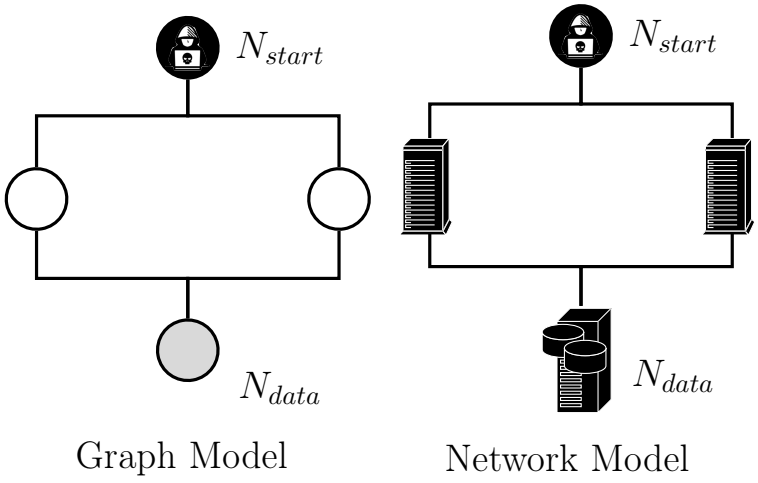


Figure 4.1 – Modeling intrusion prevention as a Markov Game. On the left the graph representation of the IT infrastructure on the right. Image extracted with permission from [8].

The described adversarial process evolves as a round-based game. In each round, the attacker and the defender perform actions on components in the network, continuing until either the attacker wins the game by compromising N_{data}, or the defender wins the game by detecting the attacker.

The nodes, denoted by $\mathcal{N}$, of the graph represent the components of the infrastructure. Similarly, the edges of the graph, denoted by $\mathcal{E}$, represent connectivity between components. The graph has two special nodes, N_{start} and N_{data}, representing the attacker's starting position and target, respectively. Moreover, each node $N_k \in \mathcal{N}$ has an associated node state, $S_k = \langle S_k^A, S_k^D \rangle$. The node's defense status S_k^D is only visible to the defender and consists of $m+1$ attributes, $S_{k,1}^D \ldots, S_{k,m+1}^D$. Likewise, the node's attack status S_k^A is only visible to the attacker and consists of m attributes, $S_{k,1}^A, \ldots S_{k,m}^A$. The attribute values are natural numbers in the range $0, \ldots, w$, where $w > 0$.

The values of the attack attributes $S_{k,1}^A, \ldots, S_{k,m}^A$ represent the strength of m different types of attacks against node N_k, e.g. denial of service attacks, cross-site scripting attacks, etc. Similarly, the values of the defense attributes $S_{k,1}^D, \ldots, S_{k,m}^D$ represent the strength of the nodes' defenses against the m attack types and encodes the node's security mechanisms, such as firewalls and encryption functions. Additionally, each node N_k has a special defense attribute $S_{k,m+1}^D$, whose value represents the node's capability to detect an attack.

The attacker can perform two types of actions on a visible node N_k: a

reconnaissance action, which renders the defense status S_k^D visible to the attacker, or an attack of type $j \in 1 \dots m$, which increases the attack value $S_{k,j}^A$ by 1.

The detection of an unsuccessful attack is determined by a Bernoulli trial $Ber(\frac{S_{k,m+1}^D}{w+1})$. If the output of the distribution is 0, then this means that the attack haven't been detected and the corresponding attack attribute is incremented by 1. If the attack value exceeds the node's defense value, i.e. $S_{k,j}^A > S_{k,j}^D$, then the attacker has compromised the node and the node's neighbors become visible to the attacker. Conversely, if the attacker performs an attack that does not compromise a node, then the attack is detected by the defender with probability $p = \frac{S_{k,m+1}^D}{w+1}$, defined by the node's detection capability $S_{k,m+1}^D$.

Just like the attacker, the defender can perform two types of actions on a node N_k: either a monitoring action, that improves the detection capability of the node and increments the detection value $S_{k,m+1}^D$, or a defensive action, that improves the defense against attacks of type $j \in 1 \dots m$ and increments the defense value $S_{k,j}^D$.

The actions are performed on a round-by-round basis in the game. In each round, the attacker and the defender perform one action each, which brings the system into a new state. The game ends either when the attacker compromises the target node, N_{data} (attacker wins), or when the attacker is detected (defender wins). The winner of a game is rewarded with a utility of $+1$ whereas the opponent receives a utility of -1.

The game evolves as a stochastic game with the Markov property, $\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \dots, s_t]$, which follows trivially from the game dynamics defined above. The size of the state space $\mathcal{S}$ of the Markov game is $(w+1)^{|\mathcal{N}| \cdot m \cdot (m+1)}$. Finally, the size of the action spaces $\mathcal{A}_1$ and $\mathcal{A}_2$ for both the attacker and the defender is $|\mathcal{N}| \cdot (m+1)$.

4.2 Environment Instantiation of the Game Model

In this section, we describe the instantiations of the Markov game model defined in the previous section. We use the instantiations of the game to evaluate our approach. We denote one instantiation of the model by an *environment*.

For the set of experiments of this work, we have considered two versions of environments, namely environment `Env1` and environment `Env2`. The

differences between the two versions are defined in Table 4.1. The main differences between the two environments are: i) the randomization of the starting node of the attacker (with no randomization, the attacker always starts at N_{start}; ii) the initial detection capability; iii) the distribution of the defense attributes per node; iv) the number of vulnerabilities; v) vulnerability value; vi) the number of attributes per node; vii) the reward function used (which is not illustrated in the figure) and viii) the account for *reconnaissance* activities.

We consider two different reward functions: *sparse rewards* and *dense rewards*. Both reward functions have the same structure but differ in the density of the rewards. The sparse reward function only give the agents a reward at the end of every episode which depend only on the outcome of the game whereas the dense reward function give rewards that depend on the final state of the game (the defense and attack attributes) and not only about the outcome of the game. For example, using the dense reward function, the defender receives a lower reward when winning a game if the attacker has compromised some nodes compared to the case when the defender wins the game and the attacker did not compromise any node.

Table 4.1 – Differences between environments `Env1` and `Env2`.

		`Env1`	`Env2`
i)	Random start	No	Yes
ii)	Detection capability	2	1
iii)	Defense attribute distr.	$Unif\{2\}$	$Unif\{7, 8, 9\}$
vi)	# Vulnerabilities	1 per node	1 per layer
v)	Vulnerability value	0	1
vi)	# Attributes	10	4
vii)	Reward	Sparse	Dense
viii)	Reconnaissance	No	Yes

An example instantiation of `Env2` is shown in Fig. 4.2. Another example can be obtained by changing the initial attack and defense attributes, for example: $S_1^D = [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]$, $S_2^D = [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]$ and $S_3^D = [2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2]$.

4.2.1 Reconnaissance Activities

In `Env2`, the attacker may play with one of its four attributes or the reconnaissance activity, which consists of recognizing the defender attributes

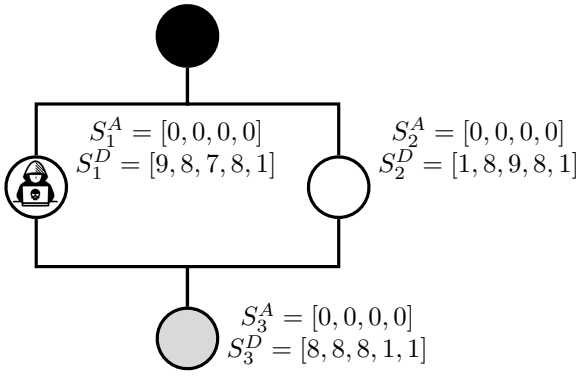


Figure 4.2 – Graph Model sample for `Env2`. The attacker may start in any of the nodes with exception to the server (grey node). The figure presents an example of typical defense values randomly sampled from a configuration distribution. Figure template used with permission from [8].

from a chosen node. This process is illustrated in Figure 4.3.

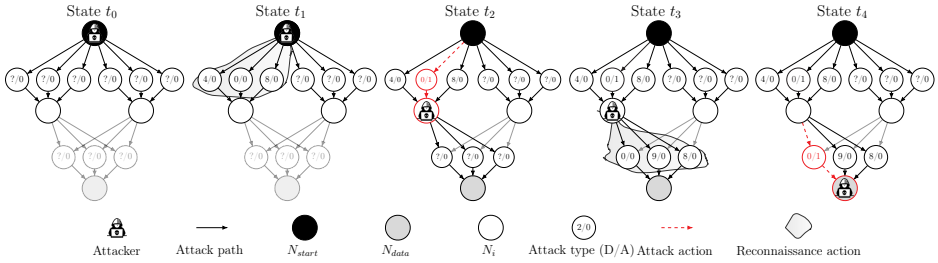


Figure 4.3 – An illustration of an attack strategy, evolving from left to right. The attacker first scans a neighboring node for vulnerabilities (low defense attributes) (state t_1). Then exploits the found vulnerability (state t_2), compromises the node, and scans the target node N_{data} (state t_3). Finally, the attacker completes the intrusion by attacking N_{data} (state t_4). Image extracted with permission from [8].

4.3 Static Opponents' Policies

For evaluating policies, we define two baseline policies, one baseline defender and one baseline attacker, as follows.

Definition 1 (Minimal Defender): A static opponent against the attacker which is a heuristic policy that updates the attribute with the minimal defense value across all nodes.

Definition 2 (Maximal Attacker or Greedy Attacker): A static opponent against the defender. It is a heuristic policy that updates the attribute with the maximal attack value across all nodes that are visible to the attacker. If several choices exist, then it makes a random choice.

4.4 Risk Averse Agents

One general principle that can used to incorporate prior knowledge in reinforcement learning policies for intrusion prevention is the principle of *risk aversion*. Risk Aversion is the tendency to prefer outcomes with low uncertainty to those outcomes with high uncertainty, even if the average outcome of the latter is equal to or higher in utility than the more certain outcome. In the context of intrusion prevention, risk aversion of the defender may take the form of a preference for strategies that minimizes the maximum likelihood of a successful intrusion over a strategy that minimizes the minimum likelihood of a successful intrusion against a particular attack strategy, for example. In this section, we define risk aversion for our game model of intrusion prevention and explain how this can be used to define a risk averse prior to learn intrusion prevention strategies in a Bayesian way.

In this context, risk aversion is a preference to increment the minimum attribute (Definition 3). Note that a risk averse defense strategy is not necessarily optimal as the optimal defense strategy depends on the attacker's strategy. However, assuming that the defender has not learned the strategy of the attacker, to be risk averse can be considered as the rational behavior of a defender, hence it can serve as a better starting point of a defender policy than starting from a random policy.

Definition 3 (Risk Averse Defender): *Consider each node to have m attributes and $|N|$ to be the number of nodes. We say that an intrusion prevention strategy $a = \pi(o)$ of a defender in the intrusion prevention game of Hammar and Stadler (2020) is risk averse iff*

$$\pi(o) = \underset{i,k}{\operatorname{argmin}} \, S^D_{i,k} \ , \ for \ i \in \{1, ..., |\mathcal{N}|\} \ and \ k \in \{1, ..., m+1\} \quad (4.1)$$

where $S^D_{i,k}$ refers to the k-th defensive attribute of node $N_i \in \mathcal{N}$. That is, an intrusion prevention strategy is risk averse if, given an observation of the state of the infrastructure, the strategy selects the action that increments the currently lowest defense attribute among all nodes and attributes in the

infrastructure.

4.5 Theoretical Win Probability

To be able to quantify the quality of policies, we are interested in specifying theoretical upper bounds on the hack probability for the attacker and the defender. These upper bounds will be compared to the policies found with reinforcement learning techniques, which are presented in subsequent chapters.

To define a theoretical upper bound on the hack probability, consider the graph model in Fig. 4.1. In this model, we are interested in specifying a joint distribution for the time it takes for the attacker to compromise of each node under an optimal policy, which then will be used to compute an upper bound on the hack probability. Note that, in the model, the attacker cannot move backwards in the node chain*.

4.5.1 Upper and Lower Bounds on the Hack Probability

The *Hack Probability* is also referred as the *Attacker Win Probability* in our plots and in the subsequent results' chapters.

For the Attacker Agent

Our goal is to compute the probability that the attacker will breach every node along the way up to the server. $\mathbb{P}(S^A_{k,j} > S^D_{k,j})$ for a given node k. Assume that the time-step when the attacker is detected at node k while performing an attack in attribute type j is geometrically distributed with probability $\frac{S^D_{k,m+1}}{10}$. Assume also that the policy for the attacker is known and the probability for each action is equal to 1.0 for the considered nodes and attributes. This is the reason why it will be omitted in our formulations†. In addition, consider the

* From [52] this could be considered as a Bayes' network, since we are assuming that the attacker must only move forward in order to achieve the goal. It is also possible to consider the agent moving backwards. In this case, one would have a Markov network instead.

† For the purpose of deriving an upper bound, the actions are deterministic, because we do the calculation for the scenarios that are interesting for us to consider. So, only these are taken into account.

detection capabilities to be fixed for both environments. Therefore:

$$\mathbb{P}(Hack\ N_k) = \mathbb{P}(S^A_{k,j} > S^D_{k,j}) = (1 - \frac{S^D_{k,m+1}}{10})^{S^D_{k,j}} \tag{4.2}$$

Under the aforementioned considerations one could say that, for `Env1`, $\mathbb{P}(Hack\ N_k) = (1 - \frac{2}{10})^0 = 1.0$* for node $k = 1$ or $k = 2$†. The nodes N_k are represented as white circles in the Graph Model of Figure 4.1.

It is reasonable to expect the defender to increase the defense attribute with value equal to zero in the node N_{data}, once we are considering the `DefendMinimal` opponent - a static opponent. Therefore, the hack probability for node N_{data} would be given by $\mathbb{P}(Hack\ \ N_{data}) = (1 - \frac{2}{10})^1 = 0.8$. The joint hack probability, which corresponds to the attacker win probability, will be given as $\mathbb{P}(Attacker\ Win) = \mathbb{P}(Hack\ N_{data})\mathbb{P}(Hack\ N_k) = 0.8 \times 1.0 = 0.8$, for both $k = 1$ or $k = 2$. Moreover, as both intermediate nodes present the same level of vulnerability, we consider as equally probable for the attacker choose between them.

Now, consider the case for the computation of the hack probability for `Env2` and take into account the assumptions from Table 4.1. Assume as an example the environment sample from Figure 4.2‡. If the attacker starts from the node N_k, then one should have $\mathbb{P}(Hack\ N_{Data}) = (1 - \frac{1}{10})^2 = 0.81$. As the defender would have time to place an extra defense on the vulnerable attribute, while the attacker performs the reconnaissance.

However, if one starts from N_{start}, then with probability $\frac{1}{2}$ we can perform a reconnaissance activity in the node with the vulnerability. This accounts for the fraction of the hack probability corresponding to 0.5×0.9^1. Meanwhile the reconnaissance activity is conducted, the `DefendMinimal` opponent would ideally be placing an extra defense on the vulnerability of node N_{data}. This, in turn, would account with the fraction of 0.9^3 to be multiplied to the previous component. Thus, so far we have the following hack probability $\mathbb{P}(Hack\ N_{Data}) = 0.9^3 \times (0.5 \times 0.9^1 + p_2)$, where the computation of p_2 is

* Here we omit the condition of $\mathbb{P}(Hack\ \ N_k)$ on the starting node, since this breaching probability is independent of the previous nodes attributes.

† In our case both nodes $k = 1$ and $k = 2$ has the same hack probability since both admits at least one attribute with respective defense value zero.

‡ As the vulnerabilities are considered to have a fixed value, this could be considered as a representative sample.

described as follows.

For the computation of p_2 we will present two main forms to achieve this result. First, the agent could merely perform a reconnaissance activity on the remaining node, to find out the vulnerability. In this case, the defender would have extra steps to improve in one more unity the vulnerable defense of the vulnerable node. This would yield the following probability: $\mathbb{P}(Hack\ N_{Data}) = 0.5 \times 0.9^3 \times 0.9^1 + 0.5 \times 0.9^3 \times 0.9^2 \approx 0.62$.

Secondly, if the agent insists in attempting to breach the node it is currently in, then we must consider the expectation for the minimal attribute value*. As `Env2` is defined to have defense attributes uniformly distributed in the set $\{7, 8, 9\}$, as a simplification one could use the minimum possible defense attribute value, i.e., 7. Hence, $\mathbb{P}(Hack\ N_{Data}) = 0.9^3 \times (0.5 \times 0.9^1 + 0.5 \times 0.9^7) \approx 0.50$.

Finally, we should take into consideration a uniform distribution for the starting node. Therefore for the two cases considered above one could compute the hack probability for the attacker as: $\mathbb{P}(Hack\ N_{Data}) = \frac{1*0.62}{3} + \frac{2*0.81}{3} \approx 0.75$ and $\mathbb{P}(Hack\ N_{Data}) = \frac{1*0.50}{3} + \frac{2*0.81}{3} \approx 0.70$, respectively. Those are upper-bounds for the attacker win probability against `DefendMinimal` opponent regarding `Env2` and under the considered assumptions.

For the Defender Agent

In this case we are interested in finding a lower bound for the attacker win probability. Now we are computing the hack probability with respect to the defender agent. As we wish the defender agent to win over the attacker, we want it to be a better player then `DefendMinimal`. Thus, this would set lower bound for the attacker win ratio.

The assumptions used here are similar to those of the previous subsection. We begin with illustrating two possible scenarios for both environments, the best one and a bad one. Those are: one where the defender attribute is to be the vulnerability and another where it is minimal but not the vulnerability.

The intuition for deriving a lower bound for the attacker win probability for the defender case is pretty simple and works for both environments `Env1` and `Env2`. As we consider an opponent that is static and attacks accordingly to a maximal attribute type, we can basically work with two simplified scenarios.

* This formulation could be trickier to formulate once it would require that we define the distribution for the minimum in a given node. Thus it might suffice to substitute by the minimal possible for defense value distribution (disregarding vulnerability).

The first scenario considers that the attacker chooses the attribute type with the vulnerability. In this case the defender can choose to defend the vulnerability of the sensible node and buy time to increase the vulnerability of other defense attributes or the detection capability.

In a second scenario, one could consider that the attacker does not start attacking the vulnerable attribute type and, instead, it starts randomly distributed through the other types. As the other types has a higher initial defense value, the defender could instead focus in increasing the detection capability. In addition, this could be combined with the increase of some attribute types.

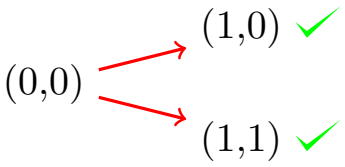
Even though the second scenario is more harmful to `Env1`, this problem could aggravate for `Env2` if one consider the randomization of the attacker starting node. A lower bound considered to be zero (which take first scenario into account) is a reasonable safer bound for both environments. In addition to our set of experiments, we inspect the learned policies so as to derive good conclusions from the learned environments.

In order to illustrate some possibilities of attribute types' realizations Figure 4.4 and 4.5 present some cases for both environments. Figure 4.4a illustrates the case for the `MaximalAttacker` agent starting in the vulnerable attribute type. The red arrows represent a possible realization of a sequence of states which would lead to the winning of the attacker (marked with a green check). Figure 4.4b illustrates the case for a non-vulnerable attribute type. One can notice that a realization of a winning scenario is sparser.

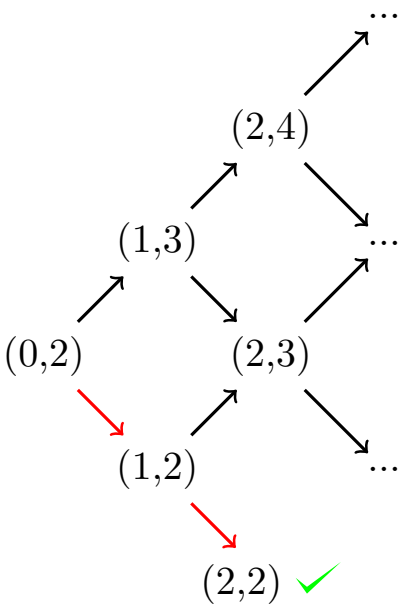
Figure 4.5a illustrates the case for the `MaximalAttacker` agent starting in the vulnerable attribute type for environment `Env2`. The sequence of realization of a attacker win event shows that, at each time-step, the defender has a possibility to remain one step-ahead of the attacker. On the other hand, in Figure 4.5b one could notice the same effect of sparsification for the event of "attacker winning". In particular, since attribute types are higher for this case, the effect of this sparsification is stronger. The blue arrows illustrate a possible sequence of states that could potentially lead to an "attacker win" scenario.

These considerations of possible scenario draws and deriving a lower bound for attacker win probability are particularly interesting for us to: i) derive a reasonable theoretical baseline, so as to study the performance of our algorithms; ii) facilitate the inspection of learned policies and help to draw conclusions; iii) motivate and help the formulation of possible different risk

averse agents.



(a) The case with vulnerability.



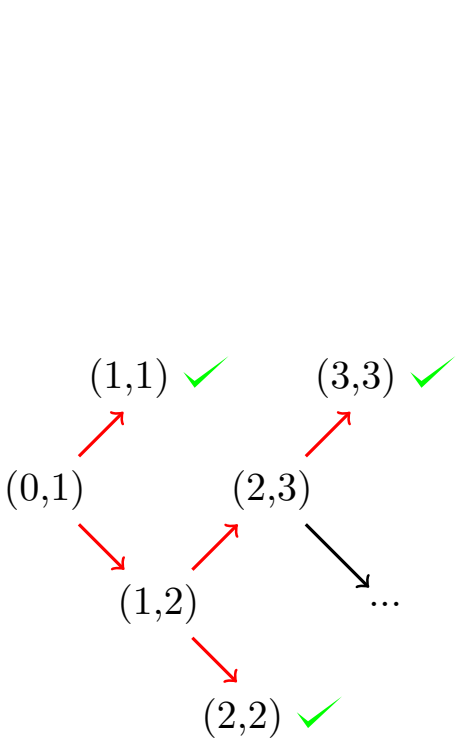
(b) The case without vulnerability.

Figure 4.4 – Some examples of how an attribute type could be filled with attack and defense attributes for `Env1`. The red arrows indicate the sequence of states that leads to the attacker to win, indicated as a green check.

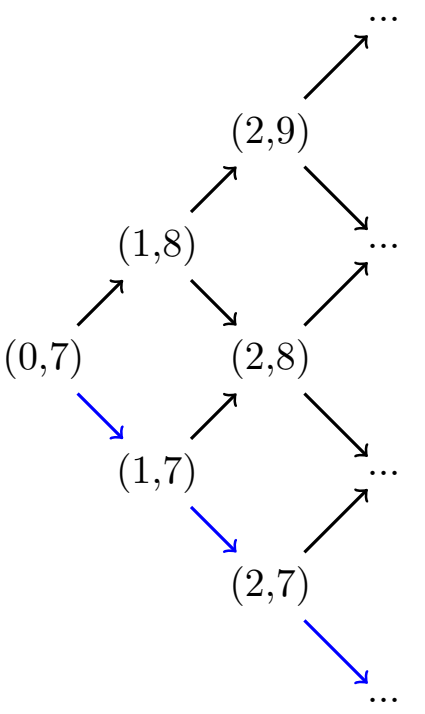
4.5.2 Evaluation Methods

To evaluate the studied algorithms with respect to the research questions, we train attacker and defender policies in simulation and measure the average win ratio (hack probability) of the attacker over time.

Additionally, for the cases of the models trained on `Env2`, we evaluate the algorithms by running evaluation episodes using the latest version of the model, which is presented in boxplots. These boxplots accounts for two different evaluation cases: one regarding sampling from the policy distribution and another by doing an *argmax* operation. They are evaluated for 1000



(a) The case with vulnerability.



(b) For the minimal attribute type case, excluding vulnerability.

Figure 4.5 – Some examples of how an attribute type could be filled with attack and defense attributes for `Env2`. The red arrows indicate the sequence of states that leads to the attacker to win, indicated as a green check. The blue arrows indicate a possible sequence of states that could eventually lead to an attacker win.

games for three different training seeds. These two approaches aim to indicate whether the agent has learned a deterministic or a stochastic policy.

4.6 Experiments

In the following chapters, we investigate the performance of the studied Bayesian reinforcement learning algorithms for the two environments: `Env1` and `Env2`.

In Chapter 5, we apply Bayesian Q-Learning to the simpler environment version `Env1`. In Chapter 6, Bayesian REINFORCE is applied to the more challenging environment `Env2` which includes reconnaissance activities and random initializations, making the BQL algorithm impractical. Finally,

Chapter 7 presents the evaluation of our implementation of Bayesian Actor-Critic in environment `Env2`.

Chapter 5

Bayesian Q-Learning for Network Intrusion Prevention

This chapter investigates the Bayesian Q-learning algorithm for training attacker and defender policies against a static opponent. Since Bayesian Q-learning is a tabular algorithm, we focus here on a the simplest version of our game model: `Env1`, as described in the previous chapter. We begin by explaining our method, including the priors, the baselines and the hyperparameters. Then, we present the results of our experiments.

5.1 Experiments' Configuration

Defining a prior for the Bayesian Q-learning algorithm corresponds to define, *a priori*, values for the parameters of the Normal-Gamma distribution. Specifically, for each prior, we define a Gaussian mean and covariance matrix for the values of the Q-table and the α and β hyperparameters to define the shape and the rate of the gamma distribution that models the variance of the Q-table distribution.

As explained in Section 2.4.1, the Bayesian Q-Learning algorithm models the reward as a normal distribution. To incorporate prior knowledge in the BQL algorithm, we explore the relation between the rewards and the Q-distribution. Specifically, we use a Q-distribution and a reward for each state-action pair. In order to prioritize the state-actions that would trigger a higher expected reward, we want to select hyperparameters for the Normal-Gamma distribution such that it will produce posterior parameters for the Q-distribution (for each state-action pair) that will lead to a particular mean value over the rewards distribution. In the following subsections we define three

priors, two for the attacker and one for the defender.

5.1.1 Model Configurations for the Attacker Agent Training

BQL Prior 1 (attacker case): This prior achieves the best possible theoretical reward. The hyperparameter μ is chosen in such a way to enforce a prior associated to the best possible reward on the attribute's configuration over the possible nodes. It sets the Normal-Gamma parameters as $\mu = -10$, $\sigma^2 = 0.25$, $\alpha_0 = 3.0$ and $\beta_0 = 4.0$. The variance and Gamma parameters are chosen so that the model can account for minimal exploration.

BQL Prior 2: This prior enforce a zero-knowledge of the advantageous actions to take. Moreover, given the higher uncertainty we enforce a more inflated variance. The parameters of the Gamma distribution for the variance are chosen in a way such that the mean of the Gamma distribution is zero and its variance is 100.0[*]. Thus, one should consider for this prior: $\mu = 0$, $\sigma^2 = 1.0$, $\alpha_0 = 0.1$ and $\beta_0 = 0.1$[†].

5.1.2 Model Configurations for the Defender Agent Training

BQL Prior 1 (defender case): This prior enforce a zero-knowledge of the advantageous actions to take. It sets the Normal-Gamma parameters as $\mu = 0.0$, $\sigma^2 = 5.0$, $\alpha_0 = 0.2$ and $\beta_0 = 0.2$.

5.1.3 Baselines and Hyperparameters

As Bayesian Q-Learning is an extension from Tabular Q method, it is natural to consider this as a baseline. Moreover, the theoretical bounds introduced in Chapter 4 were also considered as baselines.

TabularQ: This baseline is the non-Bayesian version of Q-learning, introduced in Section 2.2.4.

Table 5.1 presents critical parameters for the implementation of BQL, as the discount factor, learning rate and exploration range used.

[*] Consider that $\mathbb{E}(Gamma(\alpha, \beta)) = \frac{\alpha}{\beta}$ and $\mathbb{V}ar(Gamma(\alpha, \beta)) = \frac{\alpha}{\beta^2}$.

[†] Note that the Gamma distribution domain is defined to be strictly positive, so we can only consider a mean value approximately zero to have a finite β.

Table 5.1 – Hyperparameters for TabularQ and BQL.

Discount Factor	Learning Rate	ϵ
0.999	0.0005	[0.01, 1.0]

5.2 Experimental Results

In this section, we outline the evaluation results of the BQL algorithm on `Env1`.

5.2.1 Analysis of Attacker Win Probability for Attacker and Defender Training

Figure 5.1 illustrates the training of the attacker agent against static baseline under model configurations *BQL Prior 1* and 2. Prior 1 places an optimal mean value for the reward with a moderate variance to account for precision and a small level of exploration. It can be observed that this prior converges faster to values close to the theoretical bound whereas prior 2 performs worse than the Tabular-Q baseline.

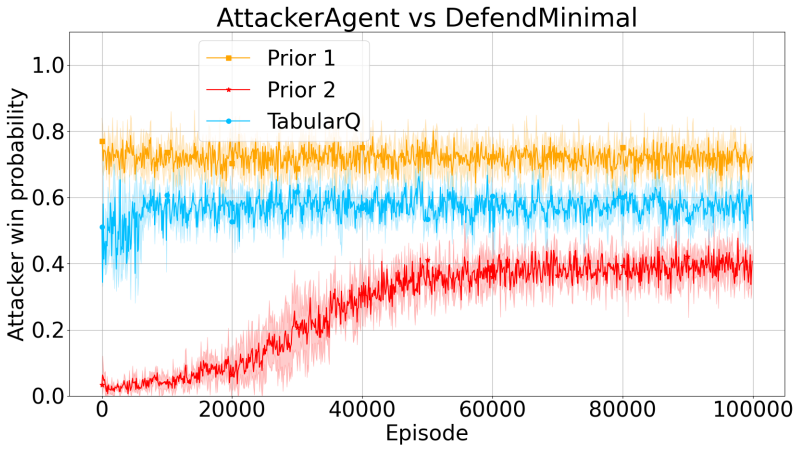


Figure 5.1 – Attacker win ratio against the number of training iterations; the graph show the results from training the attacker for *Env1*; the results are averages over three training runs with different random seeds; the shaded regions show the standard deviation.

Figure 5.2 illustrates the training of the defender agent against static baseline. As one can observe, defender training under *BQL Prior 1* has not succeeded to converge faster than Tabular Q baseline. In fact, a grid search over the defender distribution parameters was done. This has demonstrated that the defender learning usually achieves attacker win ratio performances above 0.2 and often converging to a similar level as the one illustrated in the figure.

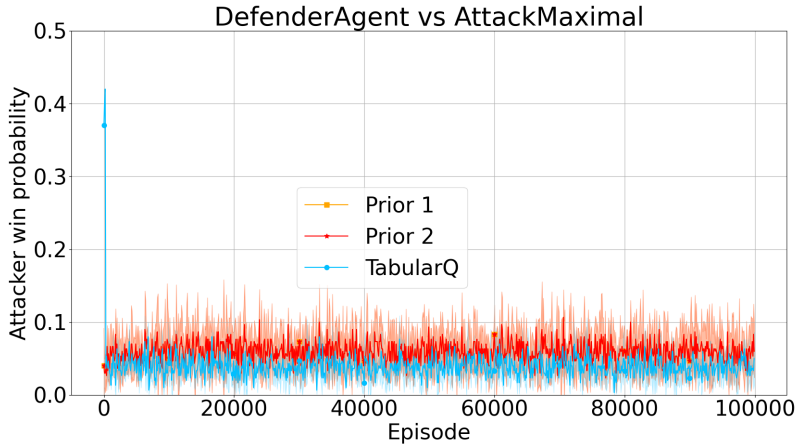


Figure 5.2 – Attacker win ratio against the number of training iterations; the graph show the results from training the defender agent against `AttackMaximal` opponent for *Env1*; the results are averages over three training runs with three different random seeds; the shaded regions show the standard deviation.

5.3 Discussion of the Results

As one can observe in Fig. 5.1, the Bayesian approach converges with significantly less amount of data, depending on the specification of the prior, compared to the non-Bayesian counterpart. Prior 1, which corresponds to the optimal initial value of the distribution parameters performs best and is close to the optimal hack probability of 0.8. Prior 2, on the other hand, is less efficient than the non-Bayesian approach. This indicates that the choice of prior determines the peformance of BQL.

Chapter 6

Bayesian REINFORCE for Network Intrusion Prevention

We begin by explaining the interpretation for the kernel and how a typical initialization look like. Then, we present several different configurations for our model. Lastly, we present the baselines, hyperparameters' definitions, experimental results and the discussion of the results.

6.1 Experiments' Configuration

In our experiments, we use model type 2 from [46] and consider different priors with respect to the reward function. This leads to the definition of a posterior over the prior by a Gaussian process that observes the true data that comes from it and a prior that is given by a kernel function.

Figure 6.1 illustrates a random sample of a kernel from a non-trained model. This kernel is represented as a heatmap with its correspondent scale indicated on the right side. The figure presents a matrix with 50 rows and 50 columns, which corresponds to the kernel function of paired combinations of the gradients of the policy networks with respect to the sampled games. In practice, the kernel matrix works as a covariance matrix for the underlying model.

6.1.1 Model Configurations for the Attacker Agent

In this section, four different priors are formulated and six different configurations for the attacker are provided to support the analysis. The priors

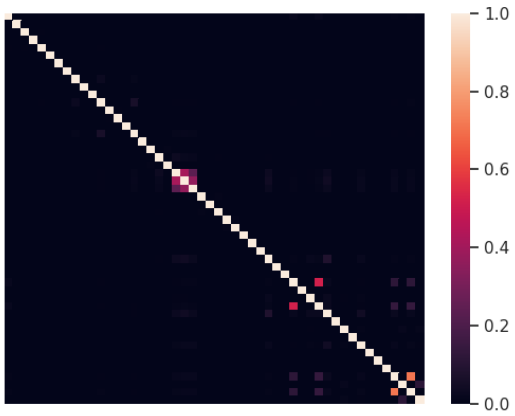


Figure 6.1 – Random sample of a normalized Kernel initialization. The matrix represents the kernel of all pairs of gradients fro the sampled games' policies.

are defined over the rewards in the environment and exploit different types of heuristics and domain knowledge about the environment:

Vanilla BPG: This model is the default configuration that follows directly from Ghavamzadeh's algorithm. Its prior is defined according to the derivation of the Bayesian Policy Gradient algorithm and uses a zero mean for the reward distribution. This model configuration accounts for kernel and noise variance of 10.0. The discount factor considered for this case is equal to 1.0. For the attacker case, we also consider a version of this model with discount factor equal to 0.999.

Prior Type 1: This prior checks for past reconnaissance activities. If they are available, then this prior associates a higher reward to the smallest defense attribute with the following rule:

$$R(a) = 1 - \frac{min(\{S^D_{k,j}(a)\}_j)}{10} \tag{6.1}$$

In this formula, $\{S^D_{k,j}(a)\}_j$ is the set of the defender attributes (for each attribute j) for node k, reconstituted from past reconnaissance activities. It depends on the action a to compute the target node. This formulation is then divided by the distance from the current node to the final one (i.e., the server).

In case the past reconnaissance activities are not available, then the prior associates a higher reward to the maximum attack attribute (consider $\{S^A_{k,j}(a)\}_j$ the attacker attributes function defined from the action a), i.e.,

$$R(a) = \frac{max(\{S^A_{k,j}(a)\}_j)}{10} \tag{6.2}$$

Noise variance used is equal to 1.0.

Prior Type 2: This prior seeks to penalize the fact that the attacker has not yet performed a reconnaissance activity. Thus, in this case $R(a) = -0.9999$.

$$R(a) = 1 - \frac{min(\{S^D_{k,j}(a)\}_j)}{10} \tag{6.3}$$

Noise variance used is equal to 1.0.

Prior Type 3: This prior uses the information of the current state and disregard the current action. By doing this, a reward is given to the best action that might not be the current one. This approach may introduce extra noise to the attributed reward and it might perform worse. Moreover, it uses the same formulation as Prior Type 2. Noise variance used is equal to 1.0.

Prior Type 4: This prior does not consider reconnaissance activities and it was first designed to handle `Env1`. Its formulation mimics equation 6.2. Noise variance used is equal to 1.0.

Note that all priors and models specified under this subsection are also multiplied by the inverse of the distance of the attacker from the server node. This intends to ensure the attacker finds an strategy according to the *shortest path*.

6.1.2 Model Configurations for the Defender Agent

In this section, we define the model configurations for the customized prior of the defender.

Vanilla BPG This model configuration is the default model from the algorithm. It uses kernel and noise variance equal 10.0.

Risk Averse This model configuration does not use the kernel, it accounts for noise variance of 1.0 and uses the risk averse framework described in 4.4.

6.1.3 Baselines and Hyperparameters

PPO This baseline is the state-of-the-art for the given problem in this work under the same environment scenario [8].

REINFORCE This baseline is also used in the seminal work of [8]. Bayesian Policy Gradient is an extension upon this type of algorithm and, therefore, it is a natural baseline for comparison.

Theoretical Bound This baseline follows the reasoning in Section 4.5. Regarding the attacker training against `DefendMinimal` opponent, we consider 0.7 as an upper bound. Regarding the defender training against `AttackMaximal` opponent, we consider 0.0 as a lower bound.

The baselines will not only offer a performance reference for our algorithms, but also serve as a training behaviour comparison as it is possible to inspect the policies learned along the training episodes.

Our approach for the parameterized policy considers a feed-forward network with two layers with hidden dimension specified as in Table 6.1. The following table summarizes the relevant hyperparameters for training. Moreover, we have investigated the performance with two different values for the discount factor: 0.999 and 1.0.

Table 6.1 – Hyperparameters for RL algorithms.

Parameter	BPGs	REINFORCE	PPO
Batch Size	1	32	2000
Hidden Dim	128	128	128
Hidden Layers	2	2	2
Discount Factor	0.999 or 1.0	1.0	0.999
Learning Rate	0.0001	0.0001	0.0001
Learning Rate Decay	No	No	No
Optimizer	Adam	Adam	Adam
Dictionary Size	50	-	-

6.2 Experimental Results

6.2.1 Analysis of Attacker Win Probability for Attacker Training

The attacker's performance is measured in terms of the attacker win probability. Figure 6.2 illustrates the performance of *Vanilla BPG* against the baselines. The BPG algorithm was able to achieve higher values for the attacker performance.

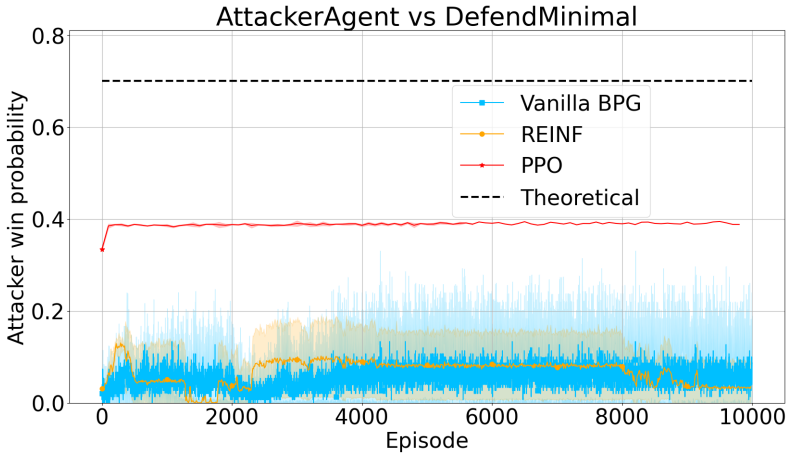


Figure 6.2 – Attacker win ratio against the number of training iterations; the graph show the results from training the attacker for `Env2`; the results are averages over three training runs with different random seeds; the shaded regions show the standard deviation. These trials make use of noise variance of 10.0 with kernel learning with RBF kernel variance of 10.0. These results consider a discount factor of $\gamma = 1.0$.

Figure 6.3 illustrates the results when we consider the training strictly in relation to the information that comes from the *Prior Types* defined in Section 6.1.1 for the case of discount factor equal to 1.0. At the bottom, the boxplot illustrates statistical summaries for the evaluation of those priors with respect to the latest version of the trained model. This evaluation accounts for 1000 runs for each of the three different seeds used during training. For each prior type there are two boxplots, one for sampling action during evaluation and another for the action with maximal probability. The statistical summaries that the boxplots provide are: the quantiles (marked by the different segmentation parts), the mean (small green triangle), the median (thicker black horizontal

line inside the box).

One can observe that prior types 1 and 4 performed worse during training. However, prior type 1 performed much worse during evaluation as is demonstrated in the boxplot. It can also be seen that prior type 3 has the best performances for both training and evaluation. Prior type 2 demonstrates much smaller variance during evaluation.

The median line in the boxplots indicates the level that splits the distribution in 50%. For prior types 3 and 4, for instance, half of the attacker win ratios were registered above the median value line. Lastly, the boxplots exhibit similar behaviour for both evaluation methods.

When using a discount factor equal to 0.999, Figure 6.4 illustrates the same prior types' case for the attacker agent as the previous graph. For this case, we are interested in distinguishing the performance between prior types 3 and 4 only. For the top figure, the mean values were plotted by using a running mean of window size 10 in order to aid the distinction between priors' performances.

The first thing one can observe is the reduced variance for prior types 3 and 4 under the two evaluation methods. In addition, despite the fact that prior type 4 has a slightly better performance during training, it performs worse than prior 3 during evaluation. Lastly, the Vanilla BPG demonstrates the same behaviour for both evaluation methods.

6.2.2 Analysis of Attacker Win Probability for Defender Training

Figure 6.5 illustrates the training for the Vanilla BPG model against the baseline under configuration for discount factor equal to 1.0. One can observe that BPG converges as to the same performance as the REINFORCE baseline and also achieve a better performance after a fewer number of training steps.

Figure 6.6 illustrates Vanilla BPG and Risk Averse cases for the defender training against the static opponent for the scenario where the discount factor is equal to 0.999. The top figure shows the attacker win probability for 10000 training episodes with three different random seeds. The bottom figure shows the evaluation under the two described methods (*sampling* and *argmax*) for 1000 games with each of the random seeds used for training.

It is shown that, with a smaller discount factor, the agent does not achieve as good performance in training as the previous case for $\gamma = 1.0$.. In fact,

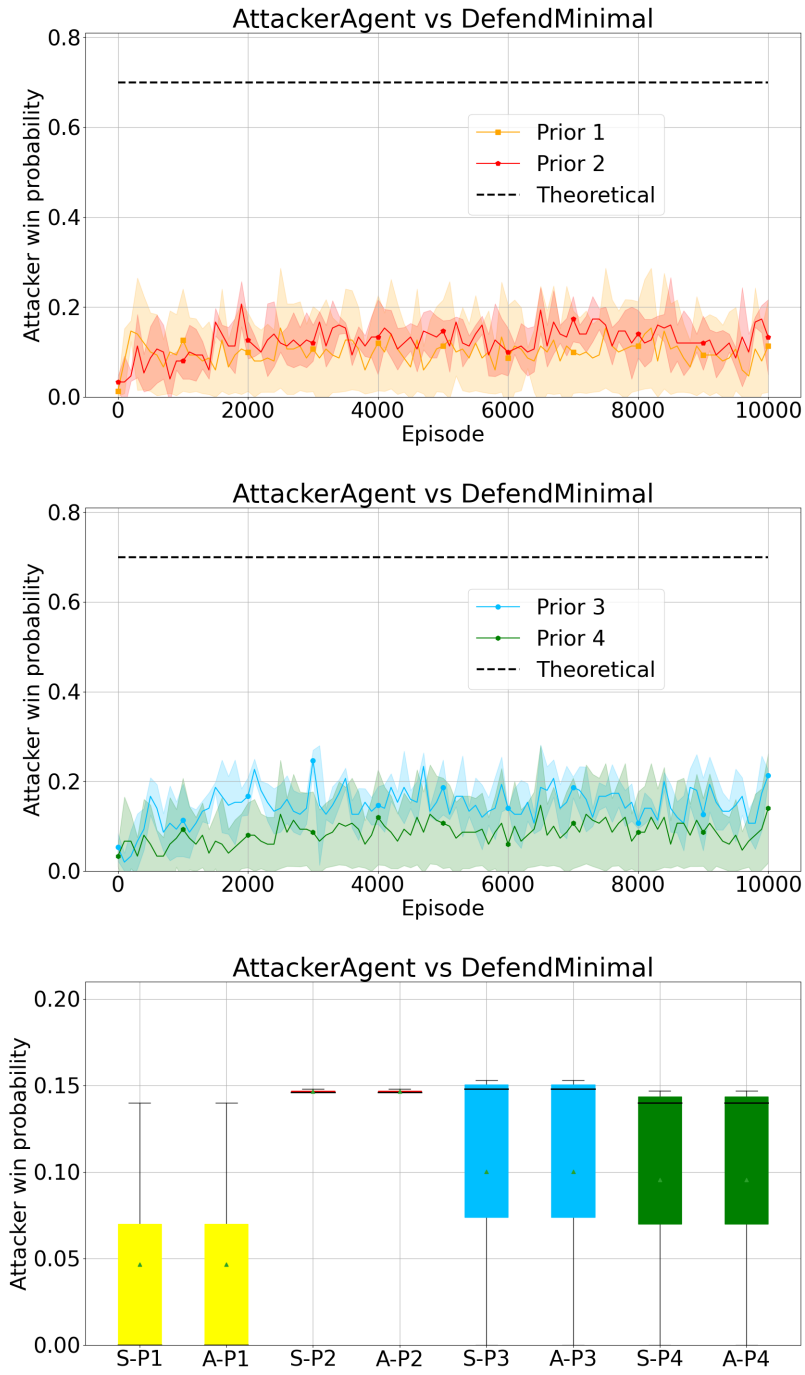


Figure 6.3 – Attacker win ratio for training and evaluation; top and middle figures illustrates the training of attacker agent against static opponent; bottom figure illustrates the evaluation distribution the for the latest trained model version. S-PX: sampled actions for prior X. A-PX: argmax of actions for prior X.

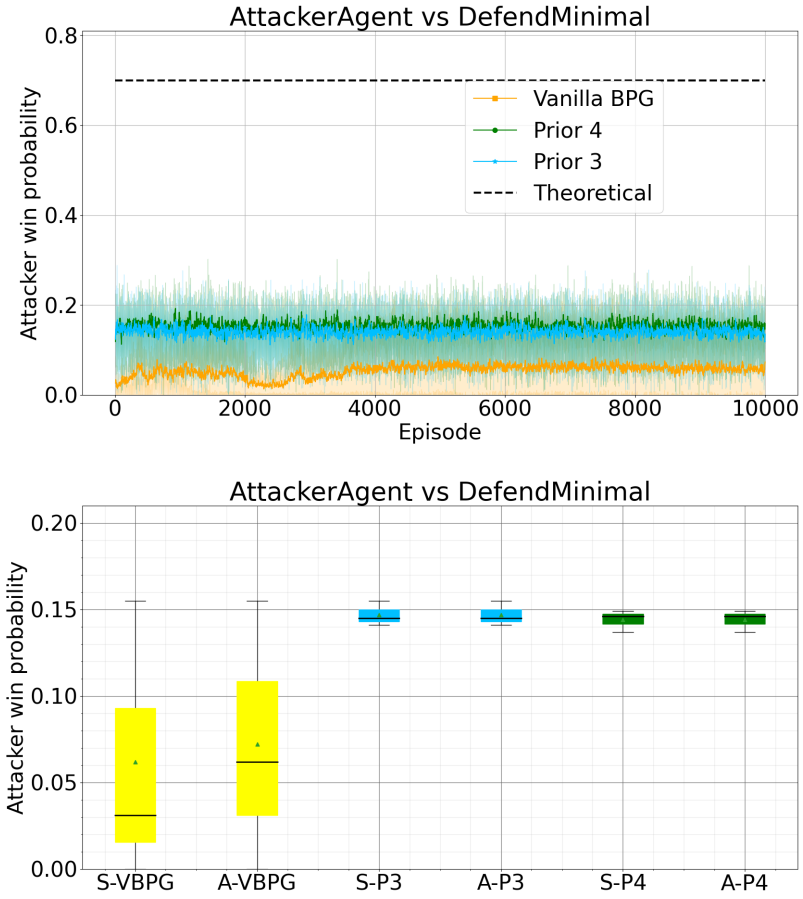


Figure 6.4 – Attacker win ratio of attacker agent against `DefendMinimal` agent. The top figure shows the training attacker win ratio for three different seeds with mean values under a rolling mean with window size 10. The bottom figure shows the boxplot distribution for the last trained model. Discount factor $\gamma = 0.999$. S-PX: sampled actions for prior X. A-PX: argmax of actions for prior X. VBPG: Vanilla BPG.

the REINFORCE baseline performs better for this value of the discount factor. However, it can be observed from the corresponding boxplots that the two evaluation cases present distinct behaviour for both models, but with opposite trends. While the risk averse model has achieved a higher variance for the *argmax* evaluation case, the vanilla model has higher variance for the *sampling* case.

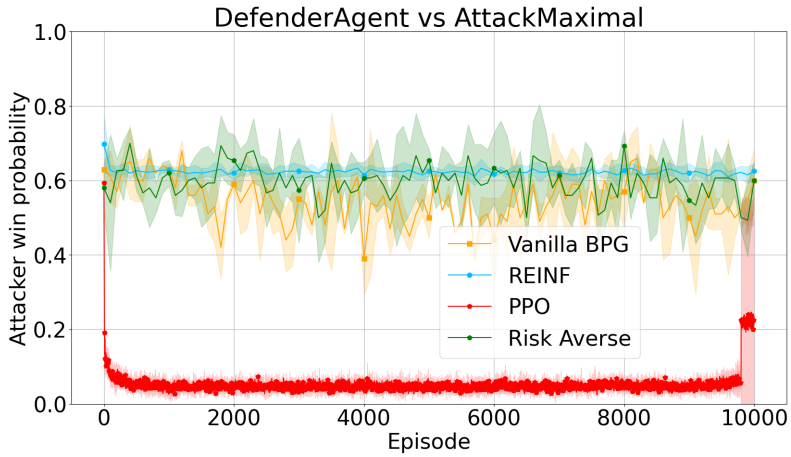


Figure 6.5 – Attacker win ratio against the number of training iterations; the graph show the results from training the defender for environment v19; the results are averages over three training runs with different random seeds; the shaded regions show the standard deviation. These trials make use of noise variance of 10.0 with kernel learning with RBF kernel variance of 10.0. These results consider a discount factor of $\gamma = 1.0$.

6.2.3 Kernel Representations

In this section, we analyze the kernel representation from the training process. In addition, we investigate what kind of kernel would be produced considering a model that have only trained by means of heuristic specification, with no account for the kernel learning.

Figure 6.7 shows the case for **Prior Type 3**, which did not use the kernel formulation during training and instead only used the information that comes from the prior. By investigating how the kernel responds in this case we obtain insights about how the kernel works. The top left figure shows how the kernel looks like if it were to consider the full kernel matrix, i.e., to disregard the sparsification procedure and consider all the sampled states. In particular, the top left case of Figure 6.7 is the same case for the bottom left, which accounts for the sparsification.

By tuning an accuracy parameter ν, one can control the dictionary of sampled game states by avoiding repeated information that are linearly dependent up to this accuracy parameter. In fact, one can observe that the represented matrix heatmap on top left of Figure 2.3.3 repeat very similar patterns across its columns and rows.

As the variance of the Gaussian kernel function is tuned, for the cases of

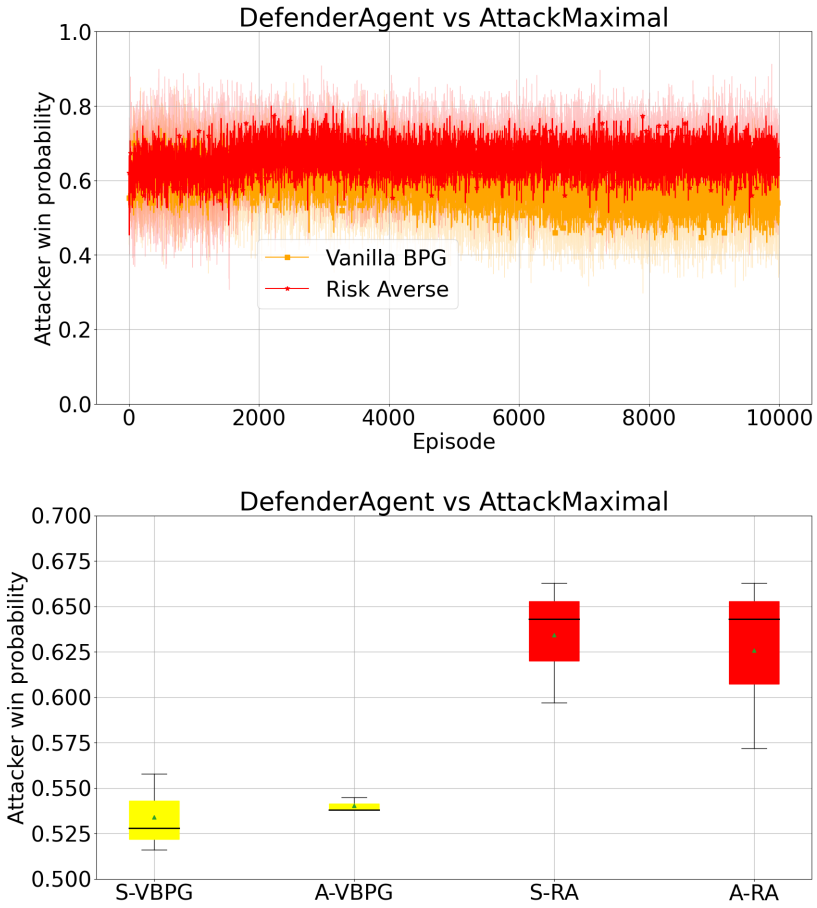


Figure 6.6 – Attacker win ratio of defender agent against `AttackMaximal` agent. The top figure shows training results for Risk Averse and Vanilla Defender. The bottom figure shows the boxplot for the evaluation of the latest model. S: Sampled. A: argmax. VBPG: Vanilla BPG. RA: Risk Averse. $\gamma = 0.999$.

1.0 (top right), 0.01 (bottom left) and 0.0001 (bottom right), one can observe the need for including more samples into the dictionary. The first case makes use of 5 samples, the second 12 and the third 14. Moreover, below a threshold, the number of used samples remains fixed at the level of 14. This differs from the results when training the defender, as we discuss below.

Another difference is regarding the range of the values. Despite the fact that there is a need for a fixed number of 14 samples, if one decreases the variance, the range of values tend to collapse between a diagonal value of 1.0 and a covariance of zeros. A final remark about this picture is that this was

the case for the best prior, the analysis for the worst prior revealed that the distinction of the values for different levels of kernel variances was poor.

The case for the defender against `AttackMaximal` opponent can be observed in Figure 6.8. In the same order as before, these plots are related to the variance levels of 10.000, 100, 5.000 and 100.000, respectively. The top left illustrates the kernel when considering all samples from the dictionary. As we lower the variance, one can observe that less samples are considered. Another distinctive difference for the defender case is the order of the variance level, which suggests that the training for the defender requires a different kernel choice than for the attacker. We can notice also an opposite trend regarding the covariance values as the attacker. For the attacker, reducing the kernel variance led to kernel values that collapses in either 1.0 or 0.0. For the defender, lowering the kernel variance makes all the kernel values tend to collapse to 1.0.

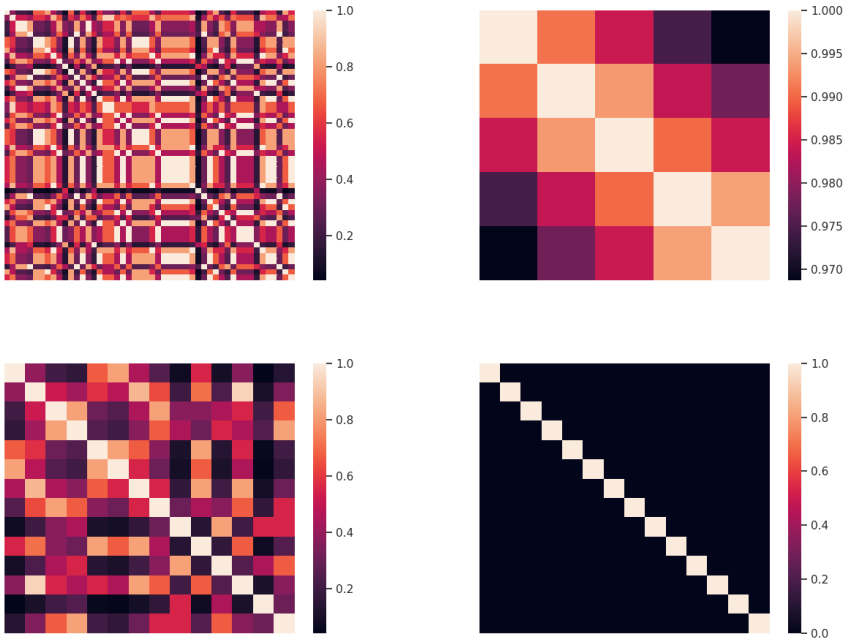


Figure 6.7 – Kernel matrices produced by a trained attacker agent against `DefendMinimal` agent. Top left: variance of 0.01, top right: variance of 1.0. Bottom left: 0.01, bottom right: 0.0001.

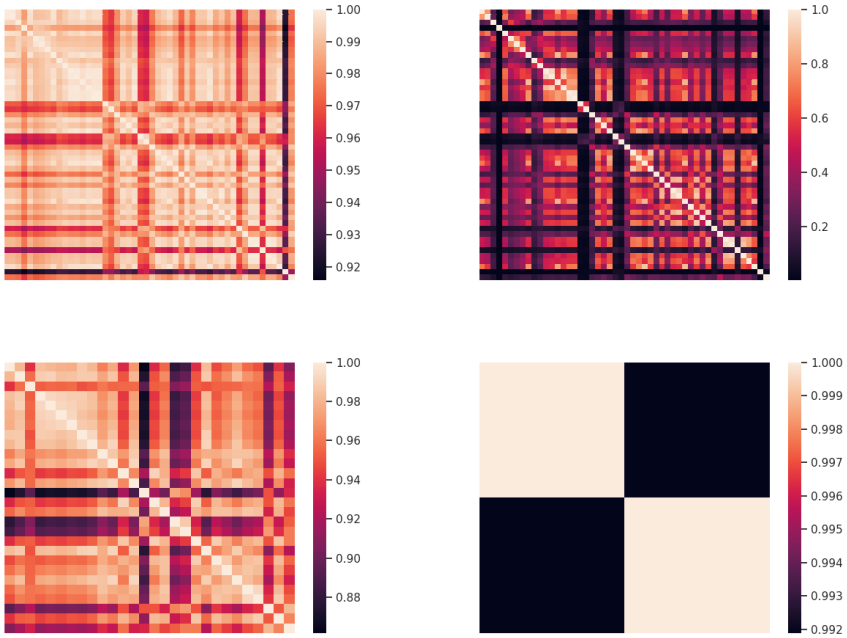


Figure 6.8 – Kernel matrices produced by a trained defender agent against `AttackMaximal` agent. Top left: variance of 10000, top right: variance of 100. Bottom left: 5000, bottom right: 100000.

6.2.4 Working Time Profiling

Figure 6.9 presents a profiling of the working time for the BPG algorithm for the attacker case. The defender has demonstrated a similar behaviour during our investigations. The bar on the left represents the composition of the bigger part of the bar on the right. The estimations are averaged over five training episodes and the vertical line on the top of each column represents a standard deviation. As one can observe the working time scales up linearly with the number of samples as demonstrates the algorithm complexity formula of Section 2.3.

6.3 Discussion of the Results

Attacker As the trained policies of the attacker perform similar in both evaluation scenarios, we can conclude that the converged policies tend to be deterministic and that the policies are stochastic only in a few cases.

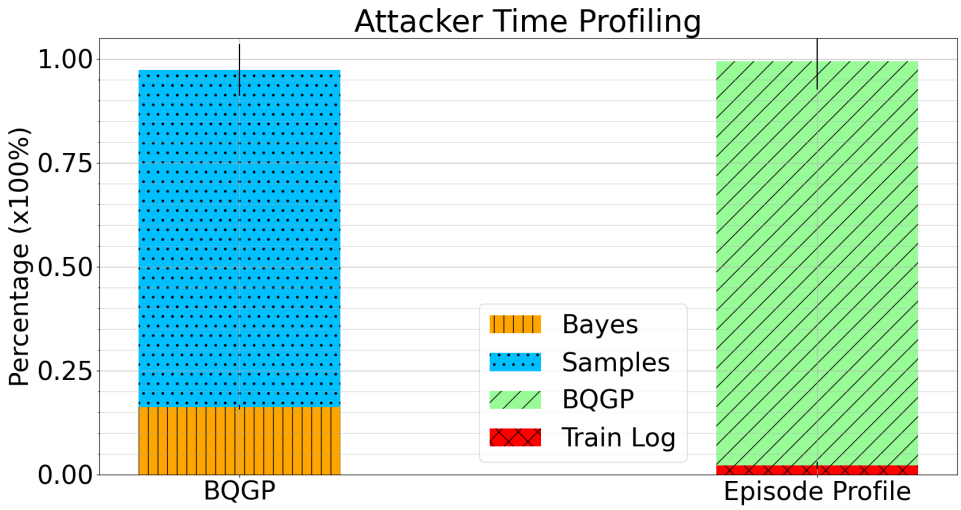


Figure 6.9 – Profiling of time required for different tasks our Bayesian REINFORCE algorithm; the bars corresponds to a percentage in relation to the time required for a full episode averaged over 5 initial runs and considering M=50. The values for BQGP column corresponds to the cumulative time for all M samples. The vertical lines on the top of each bar corresponds to the standard deviation divided by the mean value of a full episode. BQGP: Bayesian Quadrature Gaussian Process. 100% corresponds to a mean of 2.54 seconds.

Defender Regarding defender training against a static opponent, the results for the case $\gamma = 0.999$ suggests that either the agents have converged to a stochastic policy or the training process might require extra episodes in order to converge. In particular for Vanilla BPG, the *argmax* evaluation had a smaller variance, which indicates that the defender has learned a stochastic policy. On the other hand, the increased *argmax* evaluation for the risk averse defender suggests that the agent might require extra training to converge to a deterministic policy.

Chapter 7

Bayesian Actor-Critic for Network Intrusion Prevention

This chapter evaluates the Bayesian Actor-Critic algorithm* that is described in Section 2.4.2. First, we specify the priors for our set of experiments. Then, we define the used baselines and hyperparameters. Finally, we present the results and a discussion.

7.1 Experiments' Configuration

The BAC algorithm uses a prior over the Q-function which is recursively defined by the rewards.

7.1.1 Model Configurations for the Attacker Agent

Vanilla BAC: This model uses GPTD learning with kernel and noise variance equal 10.0. We use two versions of this model: one with $\gamma = 1.0$ and another with $\gamma = 0.999$.

Prior Type 1: This prior penalizes the attacker if it has not yet performed a reconnaissance activity. In this case $R(a) = -0.9999$.

* Our implementation of the BAC algorithm have is inspired by the MATLAB code developed by SequeL team. This code is available through the platform https://team.inria.fr/sequel/software/bac/.

$$R(a) = \frac{1}{2}(1 - \frac{min(\{S^D_{k,j}(a)\}_j)}{10}) \tag{7.1}$$

Prior Type 2: This prior uses the information of the current state and disregards the current action. A reward is given conditioned on the current state. It uses the same formulation as Prior Type 2 from Chapter 6.

Note that all priors and models specified under this subsection are also multiplied by the inverse of the distance from the attacker to the target node. The purpose of this is to encode a preference of the attacker for attack strategies with shorter paths to the target node.

7.1.2 Model Configurations for the Defender Agent

We use the same model configurations as in the previous chapter for training the defender agent:

Vanilla BAC This model configuration is the default model from the algorithm. It uses kernel and noise variance equal to 10.0. For this model we use the values 1.0 and 0.999 for the discount factor (*Vanilla BAC* types 1 and 2, respectively).

Risk Averse This model configuration does not use the kernel representation (and therefore GPTD is not needed to learn a reward process), it uses a noise variance with value 1.0 and uses the risk averse framework described in Section 4.4.

7.1.3 Baselines and Hyperparameters

For this set of experiments we have used the same baselines as in Chapter 6. The hyperparameters are listed in Table 7.1.

Table 7.1 – Hyperparameters for RL algorithms.

Parameter	BACs	REINFORCE	PPO
Batch Size	4	32	2000
Hidden Dim	128	128	128
Hidden Layers	2	2	2
Discount Factor	0.999	1.0	0.999
Learning Rate	0.0001	0.0001	0.0001
Learning Rate Decay	No	No	No
Optimizer	Adam	Adam	Adam
Dictionary Size	50	-	-

7.2 Experimental Results

7.2.1 Analysis of Attacker Win Probability for Attacker Training

Figure 7.1 shows the results for the case when the discount factor is equal to 0.999. Despite the fact that it has presented a reduced variance during training, the performance level is approximately the same as in the previous case.

7.2.2 Analysis of Attacker Win Probability for Defender Training

Figure 7.3 illustrates the training of the defender agent against a static opponent under model configurations *Vanilla BAC* types 1 and 2. After some training episodes it presents an inflated variance behaviour, similar to the phenomenon observed in the work [8]. Moreover, the change of noise variance seems to not have any significant effect.

Figure 7.4 illustrates the same case as before but with a reduced discount factor. It can be observed that the *Vanilla BAC* model converges faster and to better values than the previous case. The inflated variance is still notable, though. The risk averse defender had a worse performance and behave similarly to the REINFORCE baseline, but with a higher variance. The

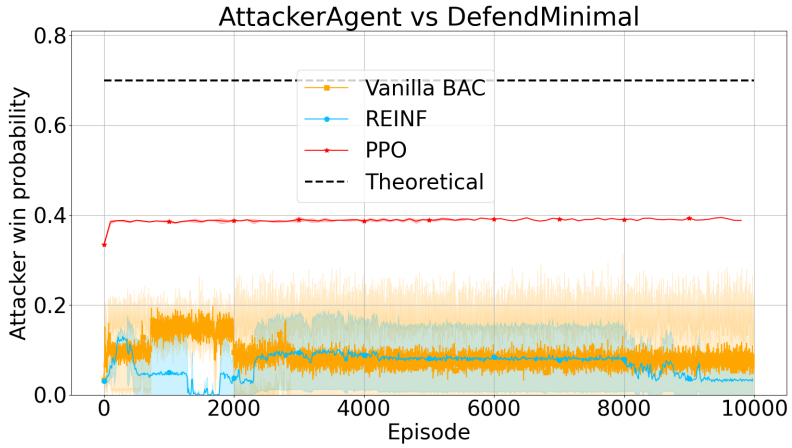


Figure 7.1 – Attacker win ratio of attacker agent against `DefendMinimal` agent. The top figure shows actual values in a frame of one episode for a shorter time-horizon. The bottom figure shows the boxplot distribution for the last 5000 episodes from the middle figure. The green triangle represents the mean values. These results consider a discount factor of $\gamma = 0.999$.

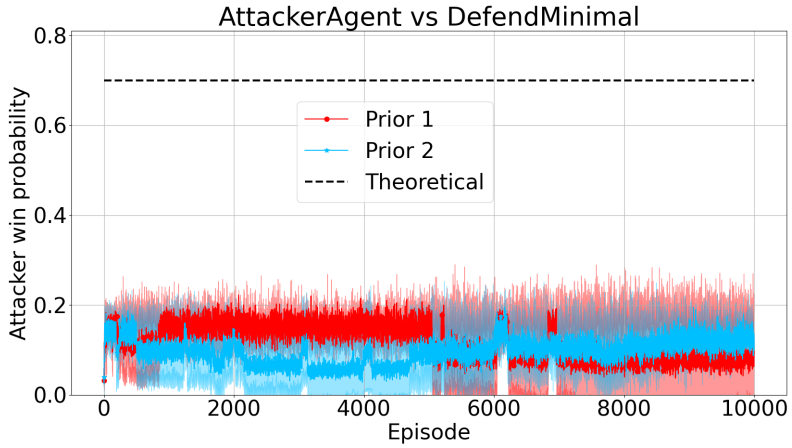


Figure 7.2 – Attacker win ratio of attacker agent against `DefendMinimal` agent. This plot exhibits the performance considering different priors.

boxplots from this figure indicate that the *sampled* and *argmax* evaluation had the same effect.

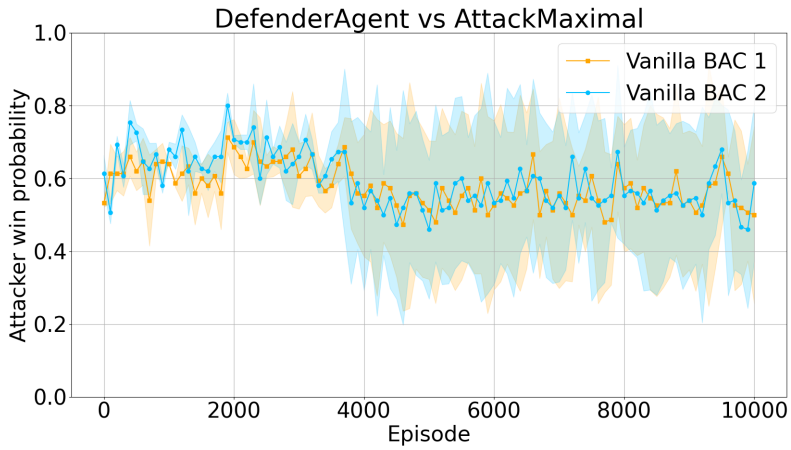


Figure 7.3 – Attacker win ratio against the number of training iterations; the graph show the results from training the defender for environment v19; the results are averages over three training runs with different random seeds; the shaded regions show the standard deviation. These trials make use of RBF kernel variance of 10.0. Vanilla BAC 1 has noise var. 10.0 and Vanilla BAC 2 has noise var. 1.0. These results consider a discount factor of $\gamma = 1.0$.

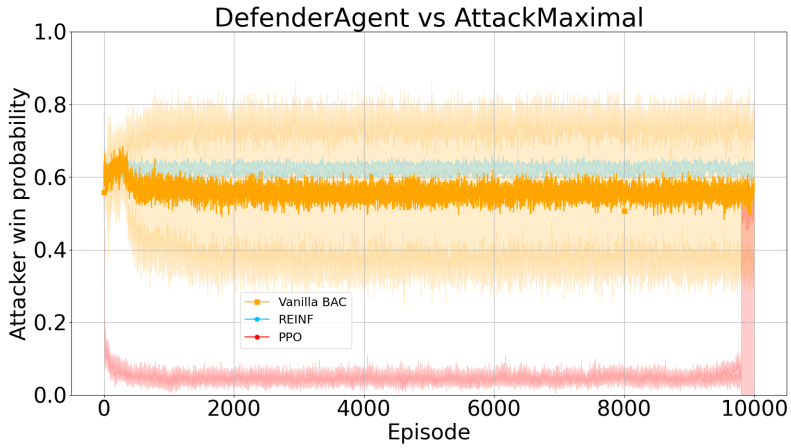


Figure 7.4 – Attacker win ratio of defender agent against `AttackMaximal` agent. The top figure shows actual values in a frame of one episode for a shorter time-horizon. The bottom figure shows the boxplot distribution for the last 5000 episodes from the middle figure. The green triangle represents the mean values. These results consider a discount factor of $\gamma = 0.999$.

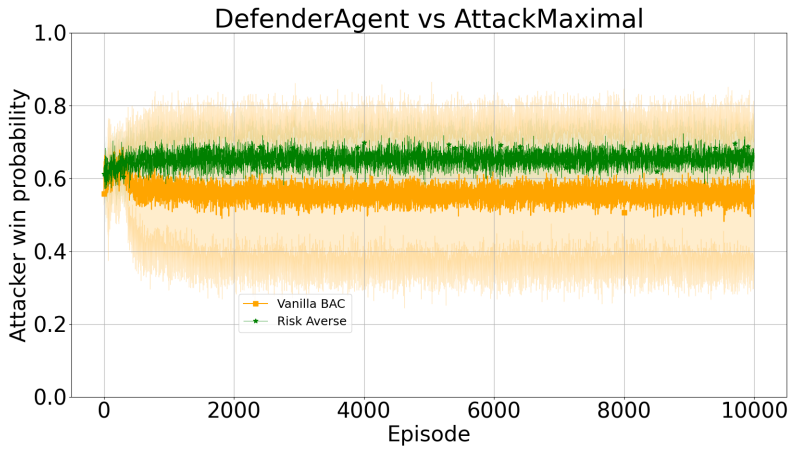


Figure 7.5 – Attacker win ratio of defender agent against `AttackMaximal` agent. The figure shows the performance for the Risk Averse agent and Vanilla BAC.

7.2.3 Kernel Representations

Figure 7.6 illustrates the kernel matrices that are produced when training the attacker agent with Vanilla BAC 1 model against a static opponent. In the figure, the kernel variance is set to $10.$, 0.1 and 0.005, respectively. It can be observed that the minimal value decreases as the variance is decreased. Moreover, the maximum dictionary size is 47. Here, we can also observe the effect of a threshold for the attacker training. This means that if the variance level is reduced to a value below 0.005, then no change in the size of the samples dictionary is detected. The minimum dictionary size observed is equal to one and it happens only under settings with high variances. Regarding the defender training, the dictionary size is equal to one for all model cases and under a wide range of variance values (from 0.001 to 10^{10}).

7.2.4 Working Time Profiling

Figure 7.7 presents the working time profile for the BAC algorithm for the attacker case. The defender has a similar profile. The bar on the left represents the composition of the bigger part of the bar on the right. The estimations are averaged over five training episodes and the vertical line on the top of each column represents a standard deviation.

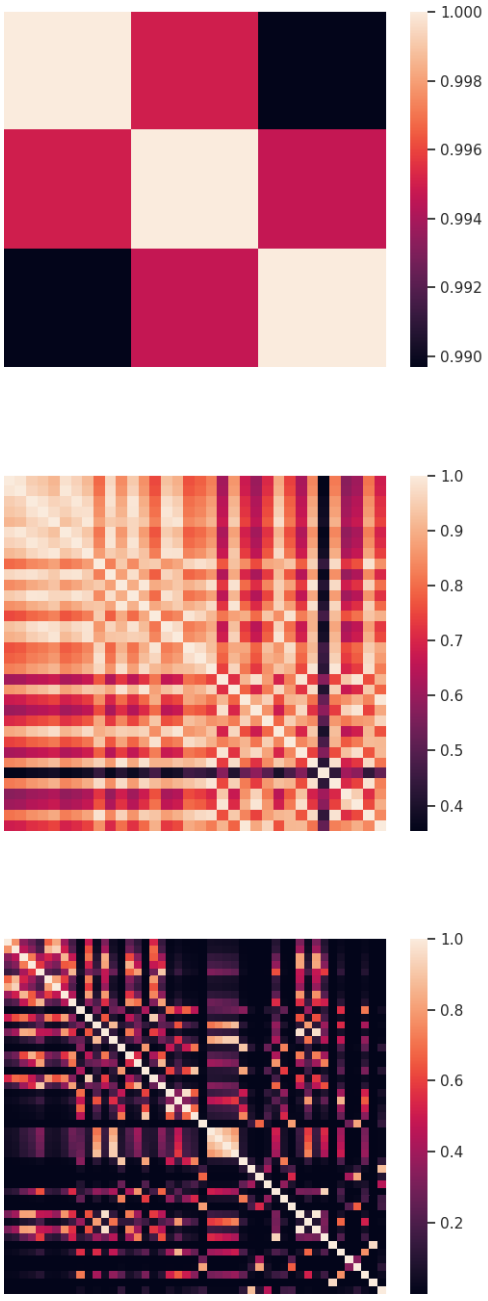


Figure 7.6 – Kernel representation from the attacker training under model type Vanilla BAC 1. From top to bottom, kernel variances are: 10., 0.1 and 0.005. Dictionary sizes are: 3, 34 and 47, respectively.

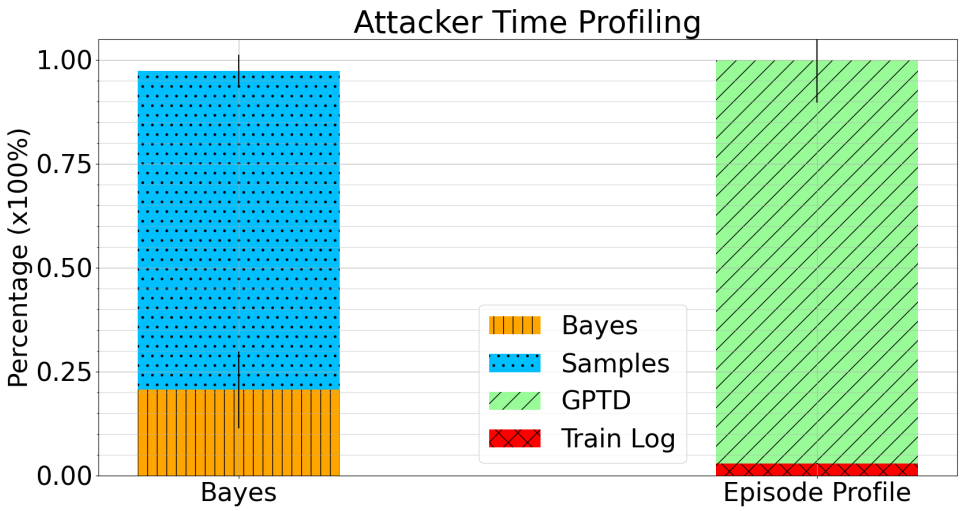


Figure 7.7 – Profiling of time required for different tasks of Bayesian Actor-Critic algorithm; the bars corresponds to a percentage in relation to the time required for a full episode averaged over 5 initial runs and considering M=50. The values for GPTD column corresponds to the cumulative time for all M samples. The vertical lines on the top of each bar corresponds to the standard deviation divided by the mean value of a full episode. GPTD: Gaussian Process Temporal Difference.

7.3 Discussion of the Results

In our results, we observe an inflated variance effect when training the defender. A similar effect was observed in [8] when training the defender agent against the same static opponent.

Chapter 8

Discussion of the Results

This chapter summarizes the findings of the thesis and relates them to the research questions.

8.1 The Experimental Results in Relation to the Research Questions

In this section we summarize the experimental results and relate them to the research questions that are stated in the beginning of this thesis.

8.1.1 Efficiency of Exploration

One of our research questions is whether the Bayesian reinforcement learning method can allow more efficient exploration when learning intrusion prevention policies compared with the non-Bayesian counterpart. To quantify the efficiency of exploration, we analyze the number of episodes required of each method to converge, see Fig. 8.1 and Fig. 8.2.

In Fig. 8.1 it can be seen that the number of episodes required to converge when training the attacker with the PPO and BAC methods is higher than the number of episodes required to converge with BPG and REINFORCE. Figure 8.2 shows the number of episodes required to converge when training the defender. In contrast to the number of episodes required to train the attacker, the number of episodes required to train the defender is approximately the same for all evaluated methods.

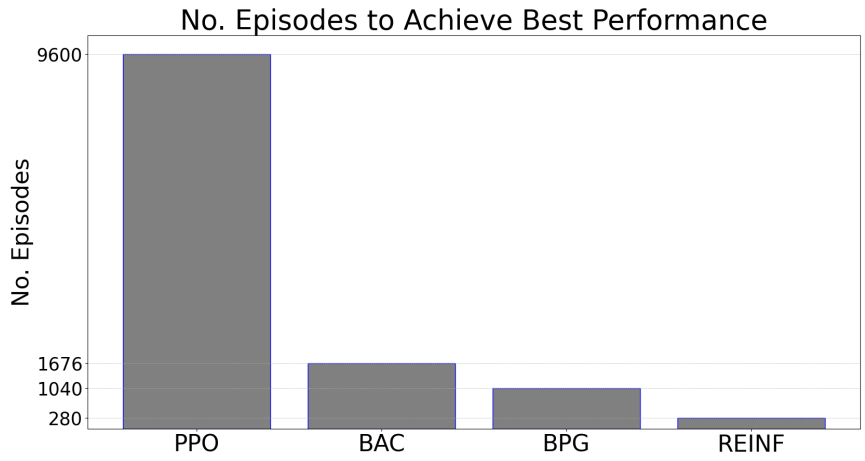


Figure 8.1 – (For Attacker training). Number of episodes to achieve the best mean performance for attacker training under policy gradient methods.

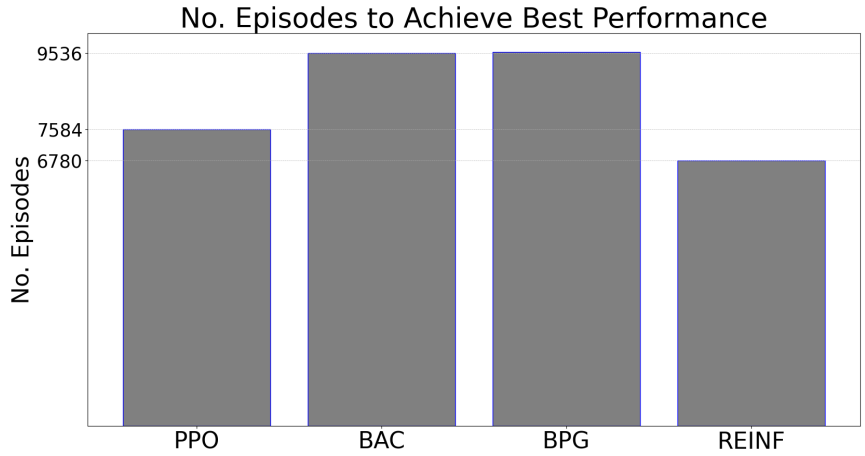


Figure 8.2 – (For defender training). Number of episodes to achieve the best mean performance for defender training under policy gradient methods.

8.1.2 Sample Efficiency

To analyze the sample efficiency of the evaluated method, we look at the number of episodes to reach a certain performance level. In Fig. 8.3 the number of episodes to reach the best performance is shown and in Fig. 8.4 the number of episodes to reach a specified performance level is shown. The plots were produced by using a graph projection where the worst best performance

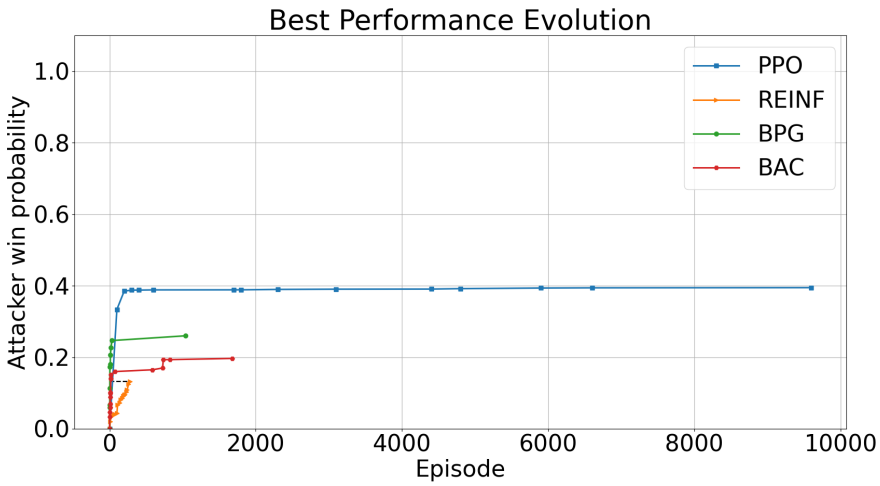


Figure 8.3 – Best mean performance evolution for attacker training. The dots illustrate whenever the training has achieved a higher performance value. The lines have different lengths according to the method's requirement for achieving higher performances in more or less episodes. The performance is measured in terms of attacker win probability. The results represent average of three different seeds.

of all methods (baselines included) were projected on the y-axis.

A sample-efficient method should achieve a higher performance with a fewer number of episodes. In the figures it can be seen that REINFORCE has the worst best performance, namely 0.13281 mean Attacker Win Probability in the 280^{th} episode. It is also shown that BPG achieves the best data efficiency (third episode), followed by BAC (sixteenth episode) and PPO (fortieth episode). Further, Fig. 8.6 shows the zoomed-in version of the Best Performance Evolution graph of the defender training, where it is shown that REINFORCE also exhibits the worst performance for this case, achieving 0.60375 as its highest mean Attacker Win Probability. We also observe from the y-axis projection, that BPG and BAC have achieved similar data efficiency performance (on the first episode), slightly better than PPO. In summary, we can say that, empirically, BPG and BAC are more data-efficient than PPO for training both the attacker and the defender.

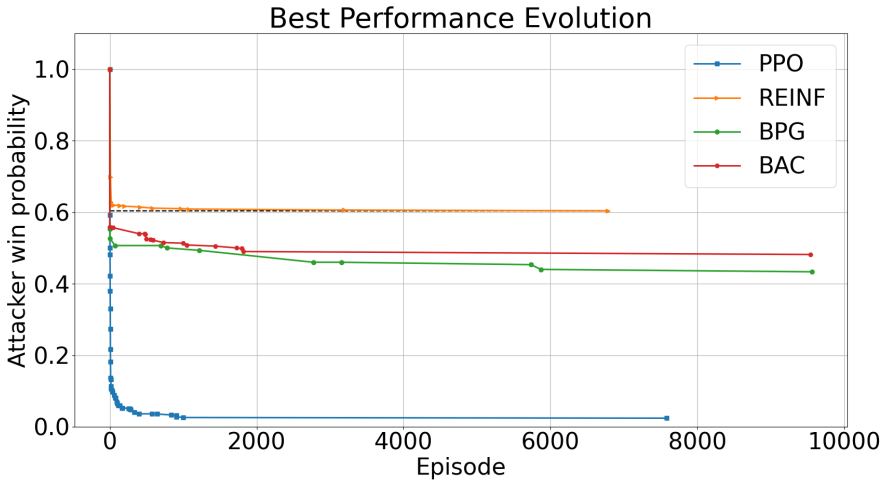


Figure 8.4 – Best mean performance evolution for defender training. The dots illustrate whenever the training has achieved a higher performance value. The lines have different lengths according to the method's requirement for achieving higher performances in more or less episodes. The performance is measured in terms of attacker win probability. The results represent average of three different seeds.

8.2 Trade-off Evaluation Summarizing

In this section, we empirically analyze the trade-offs that arise from the algorithm implementation. In particular, we pay attention to two kinds of trade-offs: one regarding computational complexity, i.e., the time required to complete an episode and how it effects the performance of the algorithm; and, the trade-off regarding kernel covariance values and how it related to the kernel variance hyperparameter.

8.2.1 Computational Complexity

Figure 8.7 summarizes the trade-off between the computational time required for one episode and the best mean level of performance achieved. The dashed blue line illustrates a possible region of trade-off for non-Bayesian methods. Our Bayesian Policy Gradient approaches are presented by the yellow dots. The more distant these dots are perpendicular to the blue line, the better the trade-off is. One can observe that BAC clearly presents a better trade-off in comparison with BPG.

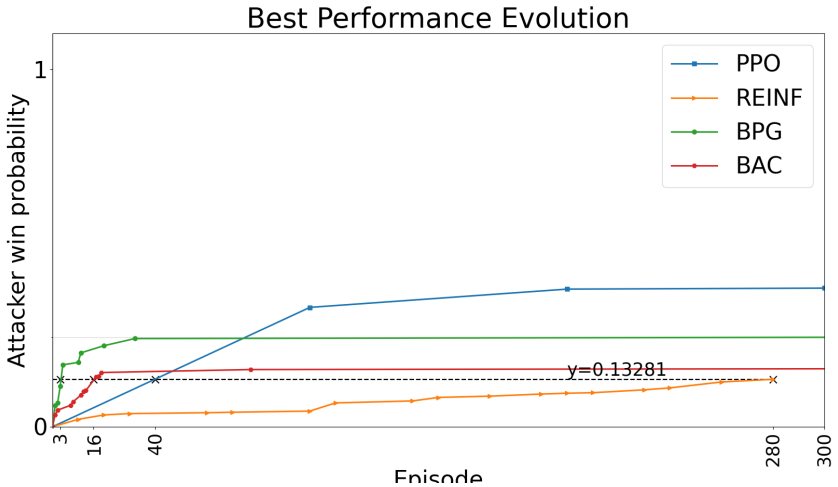


Figure 8.5 – Best mean performance evolution for attacker training. The y value corresponds to the worst best performance related to REINFORCE method. The performance is measured in terms of the attacker win probability on the y-axis. This graph is zoomed-in to emphasize the slopes above the y threshold for the given episodes. The results represent average of three different seeds.

8.2.2 Kernels' Trade-offs

Table 8.1 shows a summary of our findings for the kernel matrices for BPG and BAC cases. It summarizes the magnitude relation between different kernel parameters. For the attacker we see that as the kernel variance increases, the off-diagonal values also increase and the dictionary size decreases. However, the opposite is observed for the defender.

Finally, an interesting behaviour that was observed is as follows. The attacker presents a threshold for the dictionary size which was not observed for the defender case. At some level, if one keep decreasing the kernel variance, the off-diagonal values keep decreasing up to zero. However, the dictionary size increases up to some level below the maximum dictionary size. Then, it remains unaltered no matter further decrease of kernel variance.

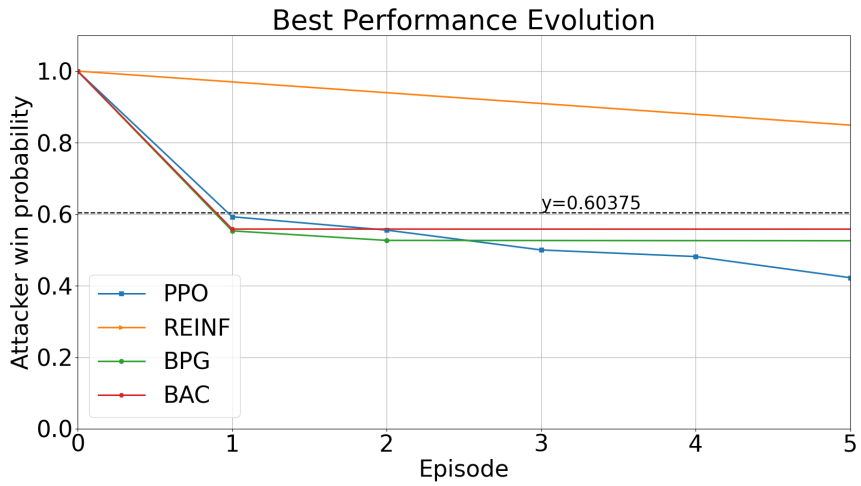


Figure 8.6 – Best mean performance evolution for defender training. The y value corresponds to the worst best performance related to REINFORCE method. The performance is measured in terms of the attacker win probability on the y-axis. This graph is zoomed-in to emphasize the slopes below the y threshold for the given episodes. The results represent average of three different seeds.

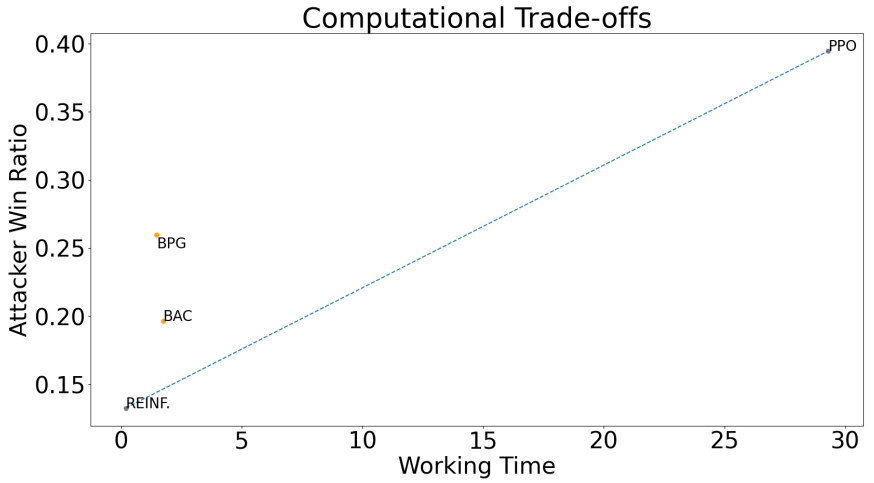


Figure 8.7 – Computational Trade-offs for Attacker training; the graph exhibits the Attacker Win Ratio (y-axis) by the Working Time required (x-axis). The horizontal dashed blue line represents an approximate computational trade-off dimension of non-Bayesian methods. The yellow dots represents the Bayesian methods.

Table 8.1 – Summary of magnitude relation for Kernel matrices in BPG and BAC algorithms.

	Kernel Var	Off-diagonal Values	Dictionary Size
Attacker	$\uparrow$	$\uparrow$	$\downarrow$
Defender	$\uparrow$	$\downarrow$	$\uparrow$

Chapter 9

Conclusion & Future Work

In this chapter we present our conclusions and interpretations of the results.

9.1 Discussion

In this thesis, we have implemented and evaluated three different Bayesian reinforcement learning algorithms for the use case of network intrusion prevention. This section contains our interpretations of the results.

9.1.1 Summary of Findings for Each Method

BQL The research question was answered for one of the chosen priors for the attacker training. We find that the choice of the modeling distribution for the rewards can dramatically impact the performance of BQL algorithm. The defender training has performed poorly.

BPG The model has satisfied the research questions posed. On the other hand, training the defender agent revealed to be much harder than training the attacker agent. Additionally, the inspection of the kernel matrices suggests the method is sensitive to the choice of hyperparameters.

BAC Bayesian Actor-Critic has a similar performance as BPG, with exception to the defender training under the fully Bayesian approach (with kernel learning). This phenomenon resembles PPO training behaviour for the defender in [8]. Moreover, our results also empirically demonstrate that BAC has an increased complexity over BPG for the Bayesian framework.

9.1.2 Conclusion

In conclusion, this work has successfully demonstrated that, regarding the research questions from Section 1.2, it is possible to improve the efficiency of exploration by using a Bayesian approach in comparison to the non-Bayesian counterpart for most of the cases. In particular, depending on the parameter configuration, it is possible to achieve a better strategy. Furthermore, it is possible to learn strategies for intrusion prevention with less data by using a Bayesian approach.

To summarize, the present work contributes to understand the following facts regarding Bayesian learning for intrusion prevention games:

- The discount factor has proved to be effective to reduce the variance during the training of the agents. However, it hurts the performance when training the defender for some cases (specially for the risk averse defender).
- The choice of the prior has a large impact on the performance of the algorithms.

9.2 Future Work

There are some ways to extend the applications of this thesis. One can consider the following directions:

- Fine tune hyperparameters and investigate the combined effect of prior collocation and kernel learning;
- To consider dynamic opponents;
- Investigate how Bayesian RL methods would behave under Self-Play learning as in [8];
- Investigate risk-averse formulations further and their expected behaviour, such as risk-aware (or risk-sensitive) agents as in [53]. This could connect with the case of deriving upper bounds to hack probability depending on initial assumptions about the environment. Each assumption has a likelihood for happening and, therefore, have a distribution which could be used for risk-sensitive agents;
- Investigate larger topologies and how it could impact the estimation of the kernel matrix. Moreover, other formulations for kernels can be considered.

9.3 Experiences

Firstly, the use of a Adam optimizer proved to be more efficient than the SGD* optimizer. The reason for this may rely in the sense that the Adam optimizer has stronger assumptions regarding the convexity of the optimization procedure.

Secondly, the online kernel sparsification procedure did not work well when using the Fisher kernel for our use case. This is empirically related to the computed values for the gradients. Furthermore, we have also experimented with the Cauchy kernel, which has a heavier tail in its distribution. This was found to be useful as well, as the gradients often presented a high quantity of very small values.

Thirdly, the noise covariance matrix was found to enforce exploration of BPG and BAC. This is considered to be reasonable given that a better prior should require less exploration. Nonetheless, learning performance was harmed by very small noise covariance, which in turn required a minimum level of exploration.

Lastly, an additional entropy term was tried out for the case of the Bayesian Policy Gradient agent. This was to ensure more exploration for the agent, as it attempts to minimize a Kullback-Leibler divergence. However, such method was found to be harmful to learning.

9.4 Ethical Consideration

The United Nations Sustainable Development Goals are 17 in total and seek to set targets that could be considered by all nations - developed and developing - to drive to structured strategies to mitigate poverty, improve health and education, reduce inequality and spur economic growth †.

The Organization for Security and Co-operation in Europe (OSCE) has developed a partnership with UN Economic Commission for Europe in order to drive efforts towards secure and sustainable transport. Some key topics were digital connectivity and energy safety. Therefore, this work offers a possibility for assessing this risk, in particular with respect to the promotion of a secure data exchange in favour of terrorism prevention.

---

* Stochastic Gradient Descent. † The reader can consider the following portal in this matter: https://sdgs.un.org/goals

References

[1] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. ISBN 0133773922, 9780133773927

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. London: The MIT Press, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

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

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *CoRR*, vol. abs/1504.00702, 2015. [Online]. Available: http://arxiv.org/abs/1504.00702

[6] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[7] T. T. Nguyen and V. J. Reddi, "Deep Reinforcement Learning for Cyber Security," arXiv:1906.05799v3 [cs.CR], Jul. 2020.

[8] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and Self-Play," in *International Conference on Network and Service Management (CNSM)*, Izmir, Turkey, Nov. 2020.

[9] A. Ridley, "Machine learning for autonomous cyber defense," the Next Wave, Vol 22, No.1 2018., 2018.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[11] M. A. Maloof, *Machine Learning and Data Mining for Computer Security: Methods and Applications (Advanced Information and Knowledge Processing)*. Berlin, Heidelberg: Springer-Verlag, 2005. ISBN 184628029X

[12] D. Gollmann, *Computer Security*. Wiley, 2011.

[13] A. Fuchsberger, "Intrusion detection systems and intrusion prevention systems," *Inf. Secur. Tech. Rep.*, vol. 10, no. 3, p. 134–139, Jan. 2005. doi: 10.1016/j.istr.2005.08.001. [Online]. Available: https://doi.org/10.1016/j.istr.2005.08.001

[14] K. A. Scarfone and P. M. Mell, "Sp 800-94. guide to intrusion detection and prevention systems (idps)," Gaithersburg, MD, USA, Tech. Rep., 2007.

[15] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35 365–35 381, 2018. doi: 10.1109/ACCESS.2018.2836950

[16] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," in *2018 10th International Conference on Cyber Conflict (CyCon)*, May 2018. doi: 10.23919/CYCON.2018.8405026 pp. 371–390.

[17] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*, 1st ed. Boston, MA, USA: Auerbach Publications, 2011. ISBN 1439839425, 9781439839423

[18] R. A. Bridges, C. L. Jones, M. D. Iannacone, and J. R. Goodall, "Automatic labeling for entity extraction in cyber security," *CoRR*, vol. abs/1308.4941, 2013. [Online]. Available: http://arxiv.org/abs/1308.4941

[19] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, "Using machine learning techniques to identify botnet traffic," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, Nov 2006. doi: 10.1109/LCN.2006.322210 pp. 967–974.

[20] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL detection using machine learning: A survey," *CoRR*, vol. abs/1701.07179, 2017. [Online]. Available: http://arxiv.org/abs/1701.07179

[21] S. Siddiqui, M. S. Khan, K. Ferens, and W. Kinsner, "Detecting advanced persistent threats using fractal dimension based machine learning classification," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, ser. IWSPA '16. New York, NY, USA: ACM, 2016. doi: 10.1145/2875475.2875484. ISBN 978-1-4503-4077-9 pp. 64–69. [Online]. Available: http://doi.acm.org/10.1145/2875475.2875484

[22] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1153–1176, Secondquarter 2016. doi: 10.1109/COMST.2015.2494502

[23] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238. [Online]. Available: http://dl.acm.org/citation.cfm?id=1039834.1039864

[24] I. Saeed, A. Selamat, and A. Abuagoub, "A survey on malware and malware detection systems," *International Journal of Computer Applications*, vol. 67, pp. 25–31, 04 2013. doi: 10.5120/11480-7108

[25] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers: A case study on pdf malware classifiers," in *NDSS*, 2016.

[26] M. D. Fraze, "Cyber grand challenge (cgc)," 2019, dARPA, https://www.darpa.mil/program/cyber-grand-challenge. [Online]. Available: https://www.darpa.mil/program/cyber-grand-challenge

[27] T. Avgerinos, D. Brumley, J. Davis, R. Goulden, T. Nighswander, A. Rebert, and N. Williamson, "The mayhem cyber reasoning system," *IEEE Security Privacy*, vol. 16, no. 2, pp. 52–60, March 2018. doi: 10.1109/MSP.2018.1870873

[28] R. Elderman, L. J. J. Pater, A. S. Thie, M. M. Drugan, and M. A. Wiering, "Adversarial reinforcement learning in a cyber security simulation," Feb. 2017. doi: 10.5220/0006197105590566. [Online]. Available: https://www.researchgate.net/publication/314153799

[29] K. Hammar and R. Stadler, "Learning intrusion prevention policies through optimal stopping,," 2021. [Online]. Available: https://arxiv.org/pdf/2106.07160.pdf

[30] M. L. Littman, "Markov Games as a framework for multi-agent reinforcement learning."

[31] D. S. et al., "Mastering the game of Go without Human knowledge," Nature, Oct. 2017.

[32] J. Heinrich and D. Silver, "Deep Reinforcement Learning from Self-Play in Imperfect-Information Games," arXiv:1603.01121v2 [cs.LG], Jun. 2016.

[33] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning, 8, 229-256, Kluwer Academic Publishers, Boston*, 1992.

[34] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," Apr. 2001. [Online]. Available: https://www.researchgate.net/publication/2354219

[35] V. Konda, "Actor-critic algorithms," PhD dissertation, Department of Electrical Engineering and Computer Science, MIT, Jun. 2002.

[36] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," arXiv:1502.05477v5 [cs.LG], Apr. 2017.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv:1707.06347v2 [cs.LG], Aug. 2017.

[38] A. O'Hagan, "Bayes Hermite Quadrature," Journal of Statistical Planning and Inference 29 245-260, 1991.

[39] J. Shawe-Taylor and M. Cristianini, *Kernel Methods for Pattern Analysis*. London: Cambridge, 2004.

[40] Y. Engel, *Algorithms and Representations for Reinforcement Learning*. Hebrew University, Apr. 2005.

[41] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. The Netherlands: Springer, 2012.

[42] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," 1998.

[43] M. Ghavamzadeh and Y. Engel, "Bayesian Policy Gradient Algorithms," 2006.

[44] Y. Engel, S. Mannor, and R. Meir, "Bayes meets bellman: The gaussian process approach to temporal difference learning," *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003*, 2003.

[45] ——, "Reinforcement learning with gaussian processes," *Proceedings of the 22 nd International Conference on Machine Learning, Bonn, Germany, 2005.*, 2005.

[46] M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar, "Bayesian Reinforcement Learning: A Survey," arXiv:1609.04436v1[cs.AI], 2016.

[47] M. Ghavamzadeh, Y. Engel, and M. Valko, "Bayesian Policy Gradient and Actor-Critic Algorithms," Journal of Machine Learning Research 17 1-53, Sep. 2016.

[48] S. Homer, "Maximum Entropy Bayesian Actor Critic," 2019.

[49] A. R. Tej, K. Azizzadenesheli, M. Ghavamzadeh, A. Anandkumar, and Y. Yue, "Deep Bayesian Quadrature Policy Optimization," arXiv:2006.15637v3 [cs.LG], Dec. 2020.

[50] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft q-learning," Oct. 2018. [Online]. Available: https://www.semanticscholar.org/paper/Equivalence-Between-Policy-Gradients-and-Soft-Schulman-Abbeel/d0352057e2b99f65f8b5244a0b912026c86d7b21

[51] M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "FlipIt: The Game of "Stealthy Takeover"," arXiv:1904.12901v1 [cs.LG], Feb. 2012.

[52] D. Koller and N. Friedman, “Probabilistic graphical models: Principles and techniques,” *The MIT Press*, 2009.

[53] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, “Risk-constrained reinforcement learning with percentile risk criteria,” *Journal of Machine Learning Research 18 (2018) 1-51*, Apr. 2018.

[54] N. A. Vien, H. Yu, and T. Chung, “Hessian matrix distribution for bayesian policy gradient reinforcement learning,” *Information Sciences 181 (2011) 1671–1685, Elsevier*, 2011. doi: 10.1016/j.ins.2011.01.001

Appendix A

Key Algorithms

Following is presented the algorithms for the Bayesian REINFORCE, Gaussian Process Temporal Difference (GPTD) and Bayesian Actor-Critic.

The algorithm 1 is a combination of the algorithms presented in [40, 43, 47, 54]. We present some local modifications so as to incorporate our kernel choice. For the exposition of this algorithm, we consider the key references to be [44, 45]. The Bayesian Actor-Critic algorithm, as introduced in the background section, makes use of the GPTD algorithm mentioned before.

Algorithm 1 Online Bayesian REINFORCE

Initialize $\mathbf{D}_1 = \{x_1\}$, $m = 1$, $\tilde{K}_1^{-1} = [1/k_{11}]$
for i=2 **to** M **do**
  Sample path ϵ_i using current policy
  Compute $\tilde{k}_{i-1}(\epsilon_i)$
  $a_i = \tilde{K}_{i-1}^{-1}\tilde{k}_{i-1}(\epsilon_i)$
  if $k_{ii} - \tilde{k}_{i-1}(\epsilon_i)^T a_i \leq \nu$ **then**
    $D_i = D_{i-1} \cup \{\epsilon_i\}$
    $\mathbf{u}(\epsilon_i) = \sum_{t=0}^{T_i-1} \nabla log\mu(a_t|s_t, \theta)$
    $R(\epsilon_i) = \sum_{t=0}^{T_i-1} r(s_t\ a_t)$
    $\tilde{K}_i^{-1} = \frac{1}{\sigma^2}\begin{bmatrix}\sigma^2\tilde{K}_i^{-1} + a_i a_i^T & -a_i \\ -a_i^T & 1\end{bmatrix}$
    $A_i = \begin{bmatrix}A_{i-1} & 0 \\ 0^T & 1\end{bmatrix}$
    $s_i = \sigma^2 + k(\epsilon_i, \epsilon_i) - \tilde{k}_{i-1}(\epsilon_i)^T A_{i-1}^T \tilde{Q}_{i-1} A_{i-1}\tilde{k}_{i-1}(\epsilon_i)$
    $g_i = \tilde{Q}_{i-1}A_{i-1}\tilde{k}_{i-1}(\epsilon_i)$
    $f(\epsilon_i) = R(\epsilon_i)$
    $Z(:, i) = \mathbf{u}(\epsilon_i)$
  else
    $A_i = \begin{bmatrix}A_{i-1} \\ a_i^T\end{bmatrix}$
    $s_i = \sigma^2 + a_i^T\Delta\tilde{k}_{i-1} - \Delta\tilde{k}_{i-1}^T A_{i-1}^T\tilde{Q}_{i-1}A_{i-1}\Delta\tilde{k}_{i-1}$
    $g_i = \tilde{Q}_{i-1}A_{i-1}\Delta\tilde{k}_{i-1}$
  end if
  $\tilde{Q}_i = \frac{1}{s_i}\begin{bmatrix}s_i\tilde{Q}_{i-1} + g_i g_i^T & -g_i \\ -g_i^T & 1\end{bmatrix}$
end for
Output: $\mathbf{E}[\nabla\eta(\theta)|D_M] = Z_M^T\tilde{Q}_M f_M$

Appendix B

Key Theorems

Theorem 1 ***(Policy Gradient):*** *The gradient of the expected reward is the expectation of the product between the reward and the gradient of the log of the policy. In mathematical notation, for a cumulative reward at time (or episode) T, $r(T)$, and a parameterized policy also at time-step T, $\pi_\theta(T)$, one could formulate it in general as:*

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r(T)] = \mathbb{E}_\pi[r(T)\nabla_\theta log \pi_\theta(T)] \tag{B.1}$$

The demonstration of this theorem can be found in Sutton and Barto (2018). The authors claim that policy gradient methods has stronger convergence guarantees than action-value methods. This is argued to be a consequence of policy parameterization as action probabilities have a more smooth distribution in relation to the this parameterization, while ϵ-greedy accounts for less subtle changes for an arbitrary small change in estimated action values.

For DIVA


```
{
"Author1": {
            "Last name": "Nesti Lopes",
            "First name": "Antonio Frederico",
            "Local User Id": "u100001",
            "E-mail": "afnl@kth.se",
            "ORCiD": "0000-0002-3493-1933",
            "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                             }
          },
"Degree": {"Educational program": "Master's Programme, Machine Learning, 120 credits"},
"Title": {
            "Main title": "Bayesian Reinforcement Learning Methods for Network Intrusion Prevention",
            "Language": "eng" },
"Alternative title": {
            "Main title": "",
            "Language": "swe"
},
"Supervisor1": {
                "Last name": "Hammar",
                "First name": "Kim",
                "Local User Id": "u100003",
                "E-mail": "kimham@kth.se",
                "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                                 "L2": "Computer Science" }
              },
"Examiner1": {
                "Last name": "Stadler",
                "First name": "Dr. Rolf",
                "Local User Id": "u100004",
                "E-mail": "stadler@kth.se",
                "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                                 "L2": "Computer Science" }
              },
"Other information": {
"Year": "2021", "Number of pages": "xvi,??"}
}
```