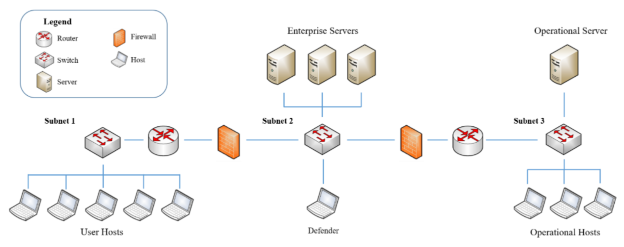
Competitive Reinforcement Learning for Autonomous Cyber Operation Attack Agents

By Jimmy Franknedy, Sammy Tesfai, Gavin Cooke, Neel Apte
February 2024

Abstract: This article serves as the design document for creating a red competitive reinforcement learning agent via fictitious play, utilizing an actor-critical framework with Uniform-based Sampling to attack a network topology (presented in Figure 1) guarded by a defending autonomous cyber operation agent.

Figure 1: *Network Topology*



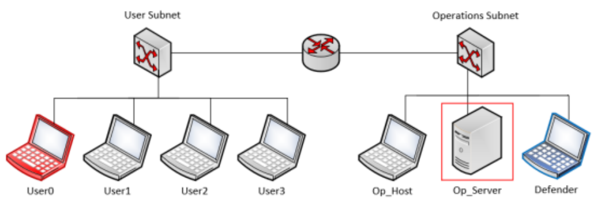
1 Introduction

As adversaries grow increasingly sophisticated, traditional defense mechanisms are often left vulnerable to novel attack vectors and techniques. In response to this relentless arms race, the concept of deploying Reinforcement Learning (RL) agents for offensive purposes in cybersecurity has emerged as a provocative and contentious frontier. In this document we outline our design for a competitive RL agent that aims to attack a computer network defended by a blue autonomous cyber operations agent.

2 Motivation

Inspired by [1], the author's use of competitive RL to develop robust agents fits our project objective very well. Given the blue and red agents' strong performance in attacking and defending a simple network (presented in Figure 2) by learning from each other dynamically, we look to deploy the same technique to develop a robust red attacking agent and analyze how well it performs in a more complex network topology.

Figure 2: *Simple Network Topology*

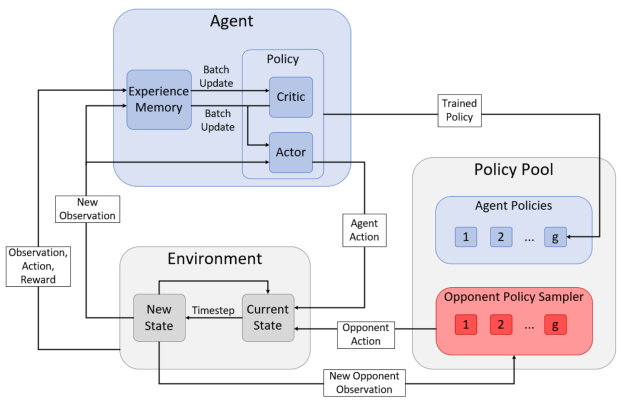


Alternative options included using a Deep Q Network (DQN) or a Double Deep Q Network (DDQN) with Prioritized Experience Replay to more efficiently approximate a policy for the red attacking agent. Despite both algorithms offering versatility (allowing the algorithms to be used in a wide variety of environments), training stability (providing reliable agent learning), and scalability (enabling the network to take in complex observations), both algorithms suffer from limited explorations (struggling with exploring the state-action space efficiently, especially in complex environments) and computational complexity (longing computation times as a result of significant state and action spaces). As a result, our choice to use an actor-critic framework in a fictitious play setting was due to its high exploration (naturally incorporating exploration strategies into the policy learning process), efficient sampling (requiring fewer samples to converge compared to value-based methods like DQN), and popularity. Furthermore, [1] includes a GitHub repository containing code that we can use to base our agent on and aid in debugging if we run into any obstacles in developing, training, or testing our agent.

3 Design

Our design follows the same implementation from [1]. Using an actor-critic framework with uniform sampling (presented in Figure 3) in a fictitious play algorithm (presented in Figure 4).

Figure 3: *Actor Critic Framework*



3.1 *Original Plan*

Our original plan consists of training a red agent from randomly initialized policies and conditioning it over time to become more robust against the blue agent over 100 generations (i.e., the complete process to train a new agent policy).

Figure 4: *Fictitious Play Algorithm*

Algorithm 3 Fictitious Play for ACO Environments.

```
$\Gamma \leftarrow$ initialize training environment
$\pi_0 \leftarrow$ set random initial policies for generation 0 $(\pi_0^{blue}, \pi_0^{red})$
$g = 0$
while within computational budget do
    $g \leftarrow g + 1$
    for each player $i$ in $[blue, red]$ do
        $\pi_g^i \leftarrow$ set random initial policy for player $i$ generation $g$
        while $\pi_g^i$ is improving do
            $M^i \leftarrow$ clear memory buffer to store new batch of samples
            while memory buffer $M^i$ is not full do
                $\pi^{-i} \leftarrow$ select opponent policy from the pool $\Pi^{-i}$
                $\Gamma \leftarrow$ reset the training environment for a new game
                $M^i \leftarrow$ store samples $(u_t^i, a_t^i, r_{t+1}^i, u_{t+1}^i)$ for every timestep $t$ in $\Gamma$
            end while
            $\pi_g^i \leftarrow$ update policy using PPO for batch of samples $M^i$
        end while
        $\Pi^i \leftarrow$ add new policy $\pi_g^i$ to pool
    end for
end while
Return $(\pi_g^{blue}, \pi_g^{red})$
```

3.2 *Backup Plan*

Our backup plan consists of creating a dedicated red agent, given the policies of open-source blue agents used for the final evaluation step of this project's submission. If the created red agent performs poorly against the blue agent (i.e., scoring below a certain threshold of points after 10 test games), we will re-train the red agent from scratch; however, instead of a randomly initialized blue policy, we will include the open-source policies of blue agents (specifically Mindrake's and Cardiff Uni's blue agents) that were used to compete in Cage Challenge #2. This way, the red agent can learn from the blue agent early on and adopt a strategy to exploit any weaknesses in its policy.

4 Implementation

To implement our agent, we will be using the Python programming language and CybORG to interface and train the agent. Code and documentation will be uploaded to a GitHub repository here.

5 Team Objectives

The role of each team member has not been strictly set; however, we plan on working on each objective collaboratively, splitting the work into pairs and ensuring that we meet our personally set deadlines to confirm that we can complete the project before the deadline and guarantee that the agent can negatively impact the blue agent's score, using open-source submission scores of Mindrake's and Cardiff Uni's blue agents as a baseline.

5.1 *Objectives*

The following lists the objectives that we have set as a path to completing the project.

- Establish the CybORG environment
- Conduct the experiment from [1] to comprehend the functionality of both the algorithm and the framework.
- Adapt the framework and algorithm to incorporate the project's network topology and action space
- Train and evaluate the agent. Employ the back up plan given poor agent performance.
- Submit code to Canvas and create presentation.

6 Bibliography

References

[1] G. McDonald, "Competitive reinforcement learning for autonomous cyber operations," May 2023.