# COSC-5P83 FINAL Report (SLIME-VALLEY-V0)

Amirmahdi Khosrvi Tabrizi
Student Number: 7278419
Email: ak21cx@brocku.ca

December 2021

## 1 Introduction

The purpose of this project is to understand how different reinforcement learning algorithms work and to compare their performance in a specific environment. Slime-Volley-V0 is the environment used in this project which is able to test algorithms in both single-agent and multi-agent mode. In addition, this is not an official OpenAI Gym environment, so some common packages, such as SLM lab, may not work.

In this environment, each agent has five lives, and their goal is to get the ball onto their opponent's ground. The game ends either when one of the agents loses all its five lives or after 3000 time-steps have passed.

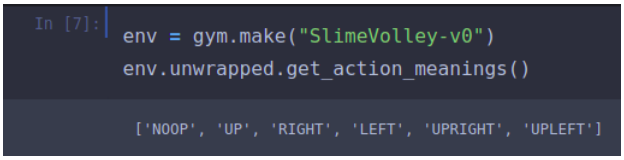
### 1.1 State Space

This environment introduce two type of state spaces, namely state-space and pixel observation. In this project, state-space observation has been used. It is a 12-dim vector representing the location of the agent, opponent and the ball. Not to mention, the origin point (0, 0) is considered for the location of the fence.

| Environment Id | Observation Space |
|---|---|
| SlimeVolley-v0 | Box(12) |
| SlimeVolleyPixel-v0 | Box(84, 168, 3) |
| SlimeVolleyNoFrameskip-v0 | Box(84, 168, 3) |

### 1.2 Action Space

There are 6 possible discrete actions for the agents in Slime-Volley environment represented by MultiBinary(3). Following figure shows the actions.




### 1.3 Reward

The agent gets the reward of +1 when the opponent loses a life, and it gets a reward of -1 in case the agent itself loses a life.

## 2 Method

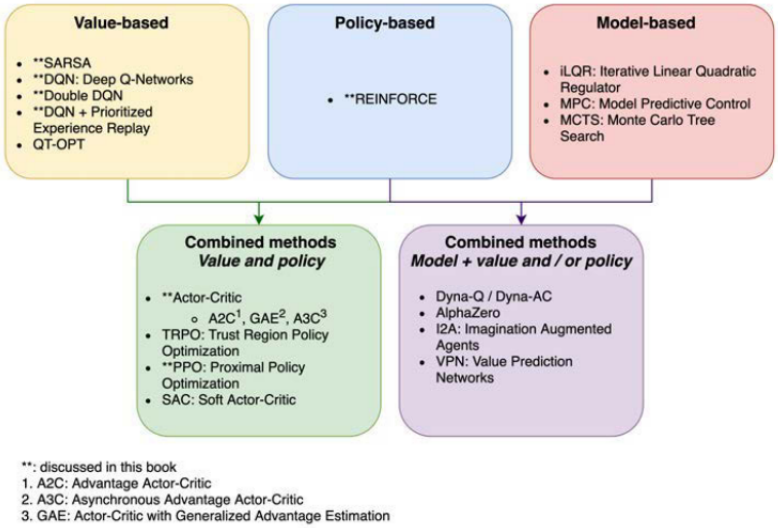
The main purpose of reinforcement learning is to train an agent being able to interact with the specific environment. To achieve this goal, the agent needs to learn number of functions, but what are these learnable functions?

There are three primary functions to learn in reinforcement learning:

1. A Policy, Π, which maps state to action.
2. A Value function, $V^{\pi}(s)$ or $Q^{\pi}(s, a)$, to estimate the expected return.
3. The environment model, $P(s^{'}|s, a)$

According to the learnable function that is chosen to learn, we may have different types of reinforcement learning methods, namely Policy-based, Value-based and Model-based. Not to mention, there are also algorithms that use the combination of these learnable functions.

At this point, the natural question to ask may be, how should we these functions? We use deep neural network as the function approximation method. The deep reinforcement learning methods used in this project are A2C, PPO1 and PPO2 which we are going to describe them in more detail.




## 2.1 A2C

A2C is one of the Actor-Critic algorithms. Just like all Actor-Critic algorithms which are considered as combined methods, it has two main components to learn—an actor, which learns a parameterized policy, and a critic which learns a value function to evaluate state-action pairs. The critic provides a reinforcing signal to the actor.

The main motivation behind these algorithms is that a learned reinforcing signal can be more informative for a policy than the rewards available from an environment. This motivation has been satisfied in A2C algorithm by defining advantage function. The advantage function determines the extent to which an action is better or worse than the policy's average action in a particular state.

### 2.1.1 Actor

As it is mentioned before, Actor learns a parameterized policy $\pi_\theta$, using the policy gradient. This is just like policy gradient in Reinforce algorithm using of advantage function as its reinforce signal instead of Monte Carlo estimate of the reward.

$$\nabla_\theta J(\pi_\theta) = E_t[A_t^\pi \nabla_\theta \log \pi_\theta(a_t|S_t)]$$

### 2.1.2 Critic and Advantage function

Critic is responsible for learning the value function to be used in order to produce the Advantage function and target value. The target value has been used in this

algorithm to compute the loss function that helps us update our critic network parameters.

Considering that learning the actual advantage function, $Q^\pi(s_t, a_t) - V^\pi(s_t)$, is complex and we need to estimate it. Following formulas show two different ways to estimate the Advantage function. Both of these formulas are using the value function.

$$\begin{aligned} A^\pi_{\text{NSTEP}}(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &\approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} \\ &\quad + \gamma^{n+1} \hat{V}^\pi(s_{t+n+1}) - \hat{V}^\pi(s_t) \end{aligned}$$

$$\begin{aligned} A^\pi_{\text{GAE}}(s_t, a_t) &= \sum_{\ell=0}^{\infty} (\gamma\lambda)^\ell \delta_{t+l}, \\ &\textit{where } \delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned}$$

Correspondingly, we need to estimate the target value:

$$V^\pi_{\text{tar}}(s_t) = r_t + \gamma r_{t+1} + \cdots + \gamma^n r_{t+n} + \gamma^{n+1} \hat{V}^\pi(s_{t+n+1})$$

$$V^\pi_{\text{tar}}(s_t) = A^\pi_{\text{GAE}}(s_t, a_t) + \hat{V}^\pi(s_t)$$

There is also another way to estimate target value using Monte Carlo estimate. In this way there is no need to use value function anymore:

$$V^\pi_{\text{tar}}(s_t) = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

## 2.2 PPO

The "Proximal Policy Optimization Algorithms" (PPO) is one of most popular policy gradient algorithms which solve the trust-region constrained policy optimization problem by introducing surrogate objective. Taking advantage of the idea of surrogate objective helps PPO to solve the most common problem of on-policy algorithms which is performance collapse.

### 2.2.1 Surrogate Objective

The main idea of using surrogate objective is to introduce the "relative policy performance identity" which measures the difference in performance between two policies. This helps PPO to chose the right step size to update its current

policy which means that the new policy should be better than the current policy- in other words, $J(\pi^{'}) - J(\pi) \geq 0$.

Now in order to accomplish this introduced ideas, we change our objective based on them. It means that in PPO maximizing the objective $J(\pi^{'})$ is now equivalent to maximizing the difference between $J(\pi^{'})$ and $J(\pi)$ in the way that, $J(\pi^{'}) - J(\pi) \geq 0$.

$$\max_{\pi'} J(\pi') \iff \max_{\pi'} \big(J(\pi') - J(\pi)\big)$$

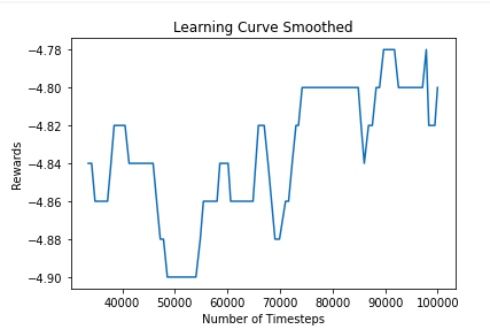
### 2.2.2 PPO1 and PPO2

PPO1 and PPO2 are two different implementation of the PPO algorithm. The main difference between these algorithms is that PPO2 is the implementation of OpenAI made for GPU. Furthermore, for multiprocessing, PPO2 takes advantage of vectorized environments compared to PPO1 which uses MPI.

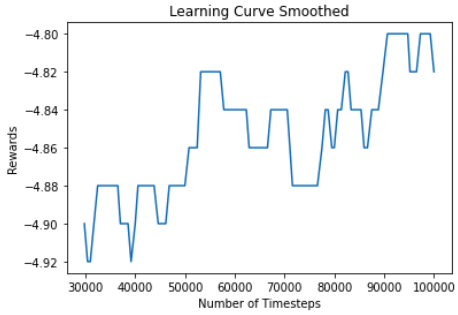
## 3 Result

All of the algorithms mentioned above, namely A2C, PPO1 and PPO2, have been tested on SlimeVolley-v0 environment. In this version of the project agents are trained by playing against the built-in baseline policy. It means that it is not implemented in a self-play environment which an agent plays against itself.

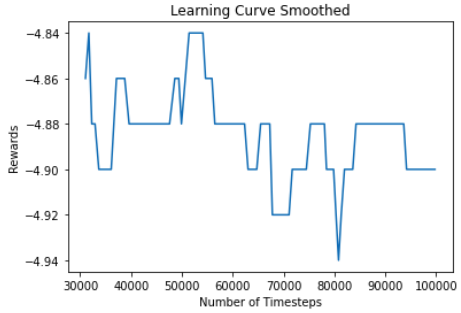
- **A2C**



- **PPO1**




- **PPO2**




## 3.1 Insight

Although both A2C and PPO1 are on-policy algorithm and fluctuated during the 100000 time-steps, it is fair to claim that PPO1 performs better and it is more reliable in long term.

Considering A2C chart, the Rewards value significantly collapses after about 50000 time-steps, while PP01 could prevent performance collapse by taking advantage of its new objective which leads to monotonic performance improvement. This new objective helps PPO1 to face two common problem among on-policy algorithms, namely sample-inefficiency and performance collapse.

PPO2 contains several modifications from the original algorithm not documented by OpenAI, but the main difference between PPO1 and PPO2 is that PPO2 is made for GPU. This could make PPO2 faster than PPO1, but PPO1 output is still more reliable. It means that PPO1 manages large updates and monotonic improvement in better way.

# 4 Conclusion

All in all, there are many deep reinforcement algorithms that we can use in our environment. Finding best algorithm to use depends on different factors- for instance, state space, action space, hyperparameters of the algorithm and the algorithm itself.

We tested three different deep reinforcement algorithm on SlimeVoley-v0, A2C and two different implementation of PPO. PPO algorithm is updated version of the A2C. This improvement makes PPO on of the most reliable reinforcement learning algorithms exists, but there are also some other algorithms that work good. For instance, rainbow is an advance reinforcement learning algorithm that performs well. It is faster than PPO and gets to the optimal state in less number of time-steps.

| Method | Timesteps (Best) | Timesteps (Median) |
|---|---|---|
| PPO | 1.274M | 2.998M |
| Data-efficient Rainbow | 0.750M | 0.751M |

Rainbow demo is reachable from this link.

# 5 Demo

Demos are reachable from the links below:

- Optimized PPO.
- PPO after 10000 time-steps.
- A2C after 10000 time-steps.
- Random Policy.

## References


- https://github.com/hardmaru/slimevolleygym
- https://gym.openai.com/docs/
- https://stable-baselines.readthedocs.io/en/master/index.html
- L. Graesser, W.L. Keng. Foundations of Deep Reinforcement Learning
- Overleaf environment of this file: https://www.overleaf.com/read/bgxbjnvhjmjx