Performance Comparison and Analysis Between Q-Learning, Advantage Actor-Critic, and Proximal Policy Optimization with Generalized Advantage Estimation in Bipedal Walker Environment

Minh T. Nguyen
Virginia Tech
mnguyen0226@vt.edu

December 1st, 2021

Abstract

In recent years, reinforcement learning (RL) algorithms have been implemented in several robotics and control systems applications. Several RL techniques are used to achieve basic autonomous controls, path-findings, vision tracker, and intelligent decision. Stabilizing bipedal walking robot is one of the challenging problems. In this paper, I will experiment and evaluate the three reinforcement learning algorithms to solve the simulated bipedal walking problem. Without any prior knowledge of its surrounding environment, the agent is able to demonstrate successful walking ability through trial and error via Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO). The results show that A2C and PPO with different bias estimation rates are capable of solving the bipedal walking problem.

Keywords: Bipedal Walker, Q-Learning, Advantage Actor-Critic, Proximal Policy Optimization, Generalized Advantage Estimation.

1 Introduction

In the beginning of class, students were introduced to one of the longstanding challenges of robotics which is learning to control agent directly from high-dimensional sensory inputs from tasks such as autonomous controls, path-finding, vision tracker, and intelligent decision in self-driving cars or autonomous robotics. RL is occasionally a go-to tools in the academic research and industry despite its reported flaws in commercial appliances [12][14]. Amongst supervised learning, unsupervised learning, and reinforcement learning, why is RL more relevant for robotic applications? To begin, there are similarities between the two fields of model predictive control and RL such as linear quadratic regulator [7]. Besides, the formalized problem of RL is occasionally from dynamical systems theory, for instance, the optimal control of incompletely-known Markov Decision Processes [18]. In addition, Jen Kober, in the paper "Reinforcement Learning in Robotics: A Survey", has raised two main reasons explained the tight connection between RL and robotics [11]. Firstly, RL offers robotic engineers a framework, tools, and algorithms for designing sophisticated and hard-to-engineer behaviour [11]. Secondly, RL enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment [11]. In details, instead of explicitly detailing the solution to a problem, RL engineers provide feedbacks in terms of scalar objective functions (which return rewards, for instance) that measures the one-step performance of the robot. The three algorithms Q-Learning, A2C, and PPO have their own strengths and weaknesses, thus it is essential to experiment, evaluate, and benchmark their performance in my interested field - robotics.

Reinforcement Learning Final Project (ECE 5984 Fall 2021), Blacksburg, Virginia, USA.

2 Problem Statement and Proposed Solution

One of the machine learning community's goals is to having a robust autonomous humanoid that is capable of passing the Turing test and assisting human in daily redundant tasks. Before achieving that goal, having a complete bipedal walking robot is the priority that robotic researchers must obtain. With the enhancement in parallel computational power and robust RL techniques in recent years, a handful number of successful bipedal humanoid robots such as Boston Dynamics's Atlas and UBTECH's Walker have been invented [3][4]. However, these machines are run mainly by control systems and hard-coded rules, not by RL algorithms. Producing a complete bipedal walking robot is a difficult task, and the problem relies on maintaining balance during locomotion due to the robot's center of gravity. A bipedal stance has high center of gravity compared to overall footprint. While this provides advantages such as increased height (and therefore field of view) compared to quadrupedal stand of the same body mass, it is also inherently less stable. Since RL algorithms allow robot to learn via trial-and-error process, they are viable solution to bipedal robots.

I will experiment, evaluate, and compare the performance of Q-Learning and the two policy gradient algorithms, A2C and PPO with Generalized Advantage Estimation (GAE), via Gym Bipedal-Walker-v2 software environment published by OpenAI [5][15]. The Bipedal-Walker-v2 environment somewhat capture the bipedal robots difficulties mentioned in the paragraph above, while allows me to experiment the algorithms without risking to damage a thousand-dollar bipedal robot.

The remainder of this paper is structured as follows. In Section 3, I will explain how the three algorithms are related and provide an overview of each algorithm. In Section 4, I will describe in details a Bipedal Walker environment, algorithmic settings, and evaluation metrics during the training process. I will present my experimental results, discussion, and comparison with OpenAi's Gym leaderboard in Section 5. I will show that Q-Learning is not a good algorithm to solve the bipedal walking problem, while A2C and PPO with different bias estimation rates can produce working bipedal walking agents.

3 Technical Background

This section contains the technical backgrounds and comparison between Q-Learning, A2C, and PPO with Generalized Advantage Estimation.

3.1 Algorithms Relationship

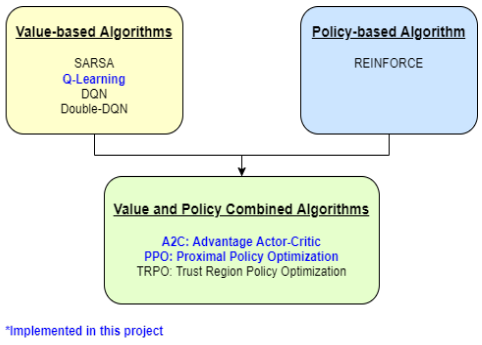


Figure 1: Relationship between Value-based algorithms and Policy-based algorithm.

Figure 1 shows the connection between the three groups of algorithms: Value-based algorithms, Policy-based algorithm, and Value-Policy combined algorithms. While Q-Learning utilizes the Q function, A2C and PPO integrate value function. Since the three chosen algorithms in this paper are either belong to one algorithmic group or a combination of two algorithmic groups, I will explain and evaluate the advantage/disadvantage of each group.

Value function and Q function: On the one hand, the value function provides information about the objective [8]. It helps the agent understand how good the states and available actions are in terms of the expected future return. The value function measures the expected return from being in states,

assuming the agent continues to act according to its current policy π [8]. The equation for the value function is:

$$V^{\pi}(s) = E_{s_0=s,\tau\sim\pi}[\Sigma^{T}{}_{t=0}\gamma^{t} r_t]$$

On the other hand, the Q function evaluates how good or bad a state-action pair is. It measures the expected return from taking action a in state s assuming that the agent continues to act according to its current policy π [8]. The equation for the Q function is:

$$Q^{\pi}(s,a) = E_{s_0=s,a_o=a,\tau\sim\pi}[\Sigma^{T}{}_{t=0}\gamma^{t} r_t]$$

Value-based algorithms: An agent learns either $V^{\pi}(s)$ or $Q^{\pi}(s,a)$ in value-based algorithms. In the standard formulation, they are mostly applicable to environments with discrete action spaces (such as the OpenAI's Gym Cliff-Walking grid-world environment). Examples of value-based algorithms are State-Action-Reward-State-Action (SARSA), Q-Learning, Deep-Q-Network (DQN), and Double DQN. Compared to policy-based algorithm, value-based algorithms are more sample efficient since they have lower variance and make better use of the data gathered from the environment [9]. However, these algorithms do not guarantee to converge to an optimum, and they are more practical in the discrete space action [8].

Policy-based algorithm: Good policies should generate actions that produces trajectories τ that maximize the agent's objective function, which is equivalent to maximize the expected rewards. Here is the objective function:

$$J(\tau) = E_{\tau\sim\pi}[\Sigma^{T}_{t=0}\gamma^{t} r_t]$$

If an agent needs to act in an environment, it makes sense to learn a policy. What constitutes a good action at a given moment depends on the state, thus a policy function π takes a state s as input to produce an action $a \sim \pi(s)$ [8]. This means an agent can make good decisions in different environmental contexts. An example of policy-based algorithm is REINFORCE. The advantages of this method are: it can be applied to both continuous and discrete environment, and it guarantee to converge to locally optimal policy which is proven by the Policy Gradient Theorem [9].

Value-Policy combined algorithms: These algorithms learn two or more of the $V^{\pi}(s)$, $Q^{\pi}(s,a)$, $J(\tau)$ functions. Examples of value-policy combined algorithms are A2C, Asynchronous Advantage Actor-Critic (A3C), PPO, Trust Region Policy Optimization (TRPO). These method utilized the advantages of both value-based and policy-based algorithms. Specifically, A2C has a learned value function that can provide a more informative feedback signal to a policy than the sequence of the rewards available from the environment. A2C also has the policy learns that use the information provided the learned value function and generate more actions [9]. Meanwhile, PPO is an improvement from A2C since it can help learning agent to perform badly and reuse off-policy data during the training process [10].

3.2 Algorithms Explained

3.2.1 Q-Learning

Q-Learning is an off-policy TD-Learning with the defined Q function [18]. the learned action-value function, Q, directly approximates the optimal action-value function q^{*}. Q-Learning is very good to find the optimal policy in grid-world (tabular setting) such as Gym's Cliff-Walking environment. It is possible to implement Q-Learning to continuous action space such as Gym's Cart Pole or Bipedal Walker environment. One approach for large state space in Q-Learning is to use function approximation for action values. Popular function approximations are linear function approximation, neural networks, or decision tree approximation. I decide to integrate Q-Learning to continuous action space by discretizing the observed state space. The OpenAI Gym's "BipedalWalker-v2" have the total of 24 observations [15]. Since only physical measurements, I will not discretize the 10 Lidar readings. Thus, the tuple of discrete state contains fourtenth values: hull angle, hull angular velocity, x-velocity, y-velocity, hip 1 joint angle, hip 1 joint speed, knee 1 joint angle, knee 1 joint speed, leg 1 ground contact, hip 2 joint angle, hip 2 joint speed, knee 2 joint angle, knee 2 joint speed, leg 2 ground contact [15]. Algorithm 1 show the steps of Q-Learning [18].

In details, Q-Learning seeks to select action which would maximize Q values. The action is chosen with and ϵ-greedy policy. With such policy, the agent will be more likely to find the global optimal

Algorithm 1 - Q-Learning

Initialize step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $totalrewards = 0$, $epsilonrate$
Initialize $Q(s, a)$, for all $s \epsilon S^+, \alpha \in A(s)$, arbitrarily except that $Q(terminal, .) = 0$
for each episode **do**
  Initialize S
  for each step of episode **do**
    Choose A from S using policy derived from Q (e.g., ϵ-greedy with $epsilonrate$)
    Take action A, discretize the observed state S', collect R
    $totalrewards \leftarrow totalrewards + R$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  end for
  Until S is terminal
end for

strategy, since it is able to balance between exploration and exploitation [6]. As the agent learns more about the environment, at the start of each new episode, the epsilon ϵ will decay by some rate that I set so that the likelihood of exploration become less and less probable as the agents learns more and more about the environment. The agent will become "greedy" in terms of exploiting the environment once it has the opportunity to explore and learn more about its surrounding [6].

3.2.2 Generalized Advantage Estimation

REINFORCE is an on-policy policy gradient algorithm [9]. The policy gradient theorem gives the gradient of the sum of discounted returns with respected to the policy parameters θ has the equation:

$$\nabla_\theta J(\pi_\theta) = E_\tau[\Sigma_{t=0}^T \nabla_\theta log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)]$$

Consider the difference between $Q^\pi(s_t, a_t)$ and $V^\pi(s_t)$. As discussed in Section 3.1, $Q^\pi(s_t, a_t)$ tells us the expected return of taking an action a_t in a state s_t, then following the policy π for the rest of the trajectory [8][9]. Meanwhile, $V^\pi(s_t)$ just tells us the expected return of the beginning in a state s_t and following the policy π from this point onward. The advantage is the difference between $Q^\pi(s_t, a_t)$ and $V^\pi(s_t)$, thus the policy gradient becomes:

$$\nabla_\theta J(\pi_\theta) = E_\tau[\Sigma_{t=0}^T \nabla_\theta log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)]$$

The two policy gradient algorithms, A2C and PPO, utilize the so-call advantage function to measure the extent to which an action is better or worse than the policy's average action in a particular state [9]. Here is the equation of the advantage estimation function:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

While the advantage function avoids penalizing an action for the policy currently being a particularly bad state, it does not give credit to an action for the policy being in a good state [8]. This is beneficial since action a can only affect the future trajectory, but not how a policy arrived in the current state. It makes sense to evaluate the action based on how it changes the value in the future.

Since $A^{\pi_\theta}(s_t, a_t)$ is dependent on $Q^\pi(s_t, a_t)$ and $V^\pi(s_t)$, it is difficult to calculate the exact advantage value, thus we have to estimate it. There are different ways to do advantage estimation. Mnih et al. mentioned that the policy gradient method could potentially improved by using generalized advantage estimation (GAE) proposed by Schulmann [16]. Here is the GAE function:

$$A_{GAE}^\pi(s_t, a_t) = \Sigma_{l=0}^\infty (\gamma\lambda)^l \delta_{t+l}$$
$$where \quad \delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

GAE is an improvement since it significantly reduce the variance of the estimator while keeping the bias as low as possible [16]. Intuitively, GAE takes a weighted average of a number of advantage estimators with different bias and variance. I will use the GAE as advantage estimator for both A2C and PPO [16].

3.2.3 Advantage Actor-Critic

The goal of RL is to find the optimal behavior strategy for the agent to obtain the optimal rewards. Policy gradient methods target at modeling and optimizing the policy directly. A2C is the actor-critic algorithm that learns the advantage function [1]. The actor learns a parameterized policy π_θ (meaning updates the policy distribution in the direction suggested by the critic) using the policy gradient [9]:

$$\nabla_\theta J(\pi_\theta) = E_\tau[\Sigma_{t=0}^{T} \nabla_\theta log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)]$$

Meanwhile, the critic is used to estimate the GAE function $A^\pi_{GAE}(s_t, a_t)$. Algorithm 2 shows the A2C with GAE algorithm [8][9][13].

Algorithm 2 - Advantage Actor-Critic with Generalized Advantage Estimation

Randomly initialize the Actor network and the Critic network parameters θ_A, θ_C
for each episode **do**
  Gather and store (s_t, a_t, r_t, s'_t) by acting in the environment using the current policy.
  for each step of episode **do**
    Calculate predicted $V^\pi(s_t)$ using the Critic network θ_C
    Calculate the GAE $A^\pi_{GAE}(s_t, a_t)$ using the Critic network θ_C
    Calculate $V^\pi_{target}(s_t)$ using the Critic network θ_C and/or the trajectory data
  end for
  Calculate value loss with MeanSquaredError (MSE):
  $Loss_{val}(\theta_C) \leftarrow \frac{1}{T}\Sigma_{t=0}^{T}(V^\pi(s_t) - V^\pi_{target}(s_t))^2$
  Calculate policy loss:
  $Loss_{pol}(\theta_A) \leftarrow \frac{1}{T}\Sigma_{t=0}^{T} - (A^\pi(s_t, a_t) log \pi_{\theta_A}(a_t|s_t))$
  Update Critic parameters with Adam optimizer
  Update Actor parameters with Adam optimizer
  Clip Actor network to avoid overfit or underfit.
end for

In detail, A2C starts by randomly setting the learning parameters θ for both network. Then, the current untrained actor network to "act" on the environment and collect the data. For each experience (s_t, a_t, r_t, s'_t), calculate the $V^\pi(s_t)$, $A^\pi_{GAE}(s_t, a_t)$, and $V^\pi_{target}(s_t)$ using the critic network. Next, we calculate the loss of both values, policy and update the learning parameters θ of both critic network and actor network with Adam optimizer. Lastly, we clip the Actor network gradient to avoid underfit or overfit.

3.2.4 Proximal Policy Optimization

A2C performs relatively well on many tasks. However, it is sensitive to hyperparameters and can be slow to train. Part of this slowness comes from the fact that A2C is an on-policy algorithm [9]. As a result, training A2C usually consists of collecting data, optimizing the policy for a single gradient step on that data, then throwing the data away [2]. Moreover, A2C is susceptible to performance collapse in which an agent suddenly starts to perform badly [8]. This causes the scenario hard to recover from since the agent will start to generate poor trajectories which are then used to further train the policy [8].

PPO solves the issues of A2C by introducing the clipped surrogate objective function introduced in "Proximal Policy Optimization" paper by Schulmann [17]. which avoids performance collapse by guaranteeing monotonic policy improvement [17]. This objective function also allows PPO to reuse off-policy data during the training process:

$$L^{CLIP}(\theta) = E_t[min(r_t(\theta)A_{GAE_t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_{GAE_t})]$$

$$where \quad r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

Figure 2 provides the visualization of the clipped surrogate objective function in the original PPO paper [17]. If the GAE is positive, then increasing the probability of taking action a_t given state

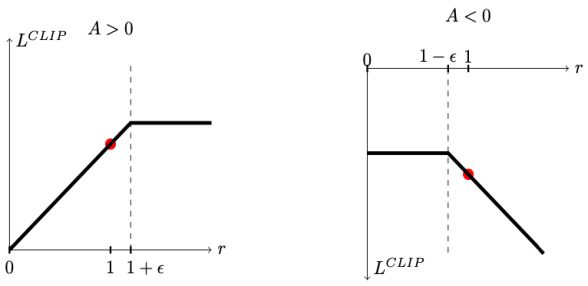


Figure 2: Clipped surrogate objective function.

s_t (and thus $r_t(\theta)$) will also increase the objective L^{CLIP} [2][10]. However, it will only increase the objective while $r_t(\theta) <= 1 + \epsilon$, after which the objective will stop increasing. Similarly, If the GAE is negative, then decreasing the probability of taking action a_t given state s_t (and thus $r_t(\theta)$) will also increase the objective L^{CLIP} [2][10]. However, it will only decrease the objective while $r_t(\theta) <= 1 + \epsilon$, after which the objective will stop increasing.

Algorithm 3 - Proximal Policy Optimization with Generalized Advantage Estimation

Initialize the clipping variable $\epsilon >= 0$
Initialize number of epochs, minibatch size, bias estimation rate
Randomly initialize the Actor network and the Critic network parameters θ_A, θ_C
Initialize the old Actor network $\theta_{A_{old}}$
for each iteration **do**
  Set $\theta_{A_{old}} = \theta_A$
  Run policy $\theta_{A_{old}}$ in the environment for T time steps and collect the trajectories
  Compute T number of GAEs $A^{\pi}_{GAE}(s_t, a_t)$ using $\theta_{A_{old}}$
  Calculate T number of value functions V^{π}_{target} using the Critic network θ_C and/or the trajectory data
  Create N number of batches with initialized batch size. These batches consist of collected trajectories, GAEs, and V^{π}_{target}
  for number of epochs **do**
    for number of minibatches **do**
      Calculate $r(\theta_A)$
      Calculate $L^{CLIP}(\theta_A)$ using the GAE from minibatch and $r(\theta_A)$
      Calculate the policy loss:
      $Loss_{pol}(\theta_A) = min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\mathrm{A}^{\pi}_{GAE}(s_t, a_t)$, clamp($\epsilon$,$\mathrm{A}^{\pi}_{GAE}(s_t, a_t)$))
      Calculate predicted $V^{\pi}(s)$ using the critic network θ_C
      Calculate the value loss using V^{π}_{target} from the minibatch:
      $Loss_{val}(\theta_A) = MSE(V^{\pi}, V^{\pi}_{target})$
      Update Actor parameters with Adam optimizer
      Update Critic parameters with Adam optimizer
    end for
  end for
end for

Algorithm 3 shows the PPO with GAE algorithm [8][10]. To begin, we first initialize th clipping ϵ, number of epochs, mini-batch size, and randomly set the learning parameters of both the actor network and the critic network. Then, we initialize an old actor network to act as $\pi_{\theta_{old}}$ for calculating the probability ratio $r_t(\theta)$. Next, we update the old actor network to the main actor network, gather trajectory data using the old actor network, and compute the GAE, and $V^{\pi}_{target}(s_t)$. We store the trajectories, GAE, and $V^{\pi}_{target}(s_t)$ in the batch to be sampled later. Next, we loop over number of epochs, sample mini-batches with pre-defined size. Here, we calculate the clipped surrogate objective, policy entropy for the actions in the mini-batch. After that, we calculate the policy loss and value loss. Lastly, we update the actor network's and critic network's learning parameters with Adam optimizer.

4 Experiment

This section will discuss the algorithm settings to fit to the Bipedal Walker environment and the evaluation metrics for comparisons. All experiments and algorithms are open-sourced for reproducibility.[1]

4.1 BipedalWalker-v2 Environment

OpenAI's Gym "BipedalWalker-v2" is a continuous control problem where an agent uses a robot's lidar sensor to sense its surroundings and walk to the right without falling. There are total of 24 numerical values that the agent collects to represent the state including hull angle, hull angular velocity, x-velocity, y-velocity, hip 1 joint angle, hip 1 joint speed, knee 1 joint angle, knee 1 joint speed, leg 1 ground contact, hip 2 joint angle, hip 2 joint speed, knee 2 joint angle, knee 2 joint speed, leg 2 ground contact, and 10 lidar readings [15]. The action of the robot is capture in four floating point numbers including hip 1 torque and velocity, knee 1 torque and velocity, hip 2 torque and velocity, knee 2 torque and velocity [15]. The environment reward the agent for moving forward to the right, up to a maximum of +300 [15]. The agent will be costed 100 if the robot falls [15]. The environment will terminate when the robot body touches the ground, reaches the goal on the right side, or records the max time step of 1600 [15].

4.2 Algorithmic Settings

For Q-Learning, I have the discount factor to be 0.99, number of episodes to be 1000, and eight different learning rate λ with values of $0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 0.9$.

For A2C, I initialize L2-regularization rate to be 0.001, the discount factor to be 0.99, number of episodes/iteration to be 1000, mini-batch size to be 2048, and six bias estimation rates λ with the values of $0.50, 0.70, 0.90, 0.95, 0.97, 0.99$. Due to the limitation in computational power, the actor and critic networks have shallow depth. Both networks contains five layers including one linear layer that takes in input feature of size 24, one linear layer that takes in input feature of size 200, one batch norm layer with batch size of 200, one batch norm layer with batch size of 128, and one linear layer that takes in input feature of 128.

For PPO, I set L2-regularization rate to be 0.001, discount factor to be 0.99, number of episodes/iteration to be 1000, mini-batch size to be 2048, clip-epsilon rate to be 0.2, optimizing epoch to be 10, and optimizing batch size to be 64. Due to the limitation in computational power, the actor and critic networks have shallow depth. Both networks contains three layers including one linear layer that takes input feature of size 24, one linear layer that takes in input feature of size 128, and one linear layer that takes in input feature of size 128.

4.3 Comparative Metrics

Three comparative metrics are chosen including: graphs of Reward per Episode, the max training time for 1000 iterations, and the number of iteration it takes for each algorithm to reach the reward of 300. Besides, I will compare the performance of my implementation with OpenAI's Gym top performers.

5 Results, Discussion, and Comparison

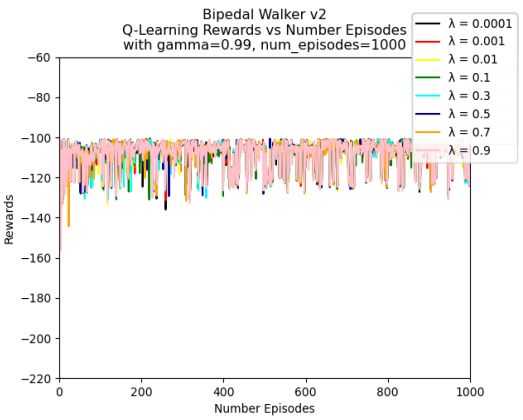
This section will discuss the results of provided by three algorithms on OpenAI's Gym "BipedalWalker-v2" environment.

5.1 Results and Discussion

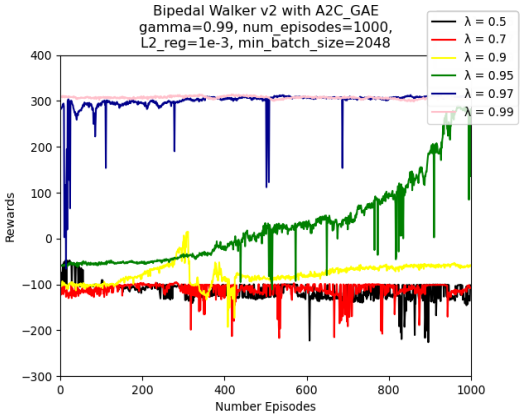
Figure 3 shows the results of Q-Learning, A2C, and PPO with different tunning parameters over 1000 iterations during the training process. To begin, figure 3a shows that Q-Learning is not an effective algorithm for training BipedalWalker-v2 regarding the experimental learning rate λ. There is oscillation around the rewards of -100 to -130 showing the agent consistently perform poorly.

[1] **GitHub:** https://github.com/mnguyen0226/rl_value_based_vs_value_policy_based

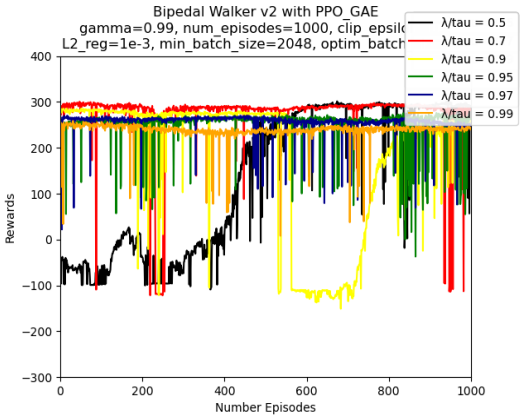
Since Q-Learning did not provide an agent that can gain a 300 reward, we conclude Q-Learning did not solve the bipedal walking problem. One reason explained for this poor performance is due to discretization of the continuous environment which cost losing information. This lead may lead to the agent did not learn anything regarding trial-and-error or ϵ-greedy policy. The recording[2] shows that the robot keeps falling at the beginning and never learn corresponding to the Figure 3a. This result shows that Q-Learning works best for discrete environment but not the continuous one.



(a) Performance of Q-Learning with different λ.



(b) Performance of A2C with different λ.



(c) Performance of PPO with different λ.

Figure 3: Performance of Q-Learning, A2C, and PPO in BipedalWalker-v2.

While Q-Learning performed poorly, A2C shows a more promising results with some bias estimation rates in figure 3b. While λ equals to 0.5, 0.7, 0.9 showed that A2C did not learn anything with the average reward is -100, λ equals to 0.95, 0.97, 0.99 gain incremental rewards over iterations. For λ equals to 0.95, the agent learns with exponential increase in collected reward. For λ equals to 0.97, the agent drops its performance in some episodes, but it gains the rewards of 300 in overall. For λ equals to 0.99, the agent immediately learn the with the rewards of more than 300; especially, it is able to keep the gained rewards over 1000 iteration consistently at 300. Since A2C agents with λ equals to 0.97 and 0.99 was able to gain rewards of +300, I conclude that they solved the bipedal walking problem. The recording shows that the robot was able to move from left to right of the environment; however, it is not able to learn the "normal walking" movement. The reason for this unrealistic walking is due to the shallow depth of actor network and critic network.

PPO shows the most promissing results amongst the three algorithms with all agents with different bias estimation rates λ able to learn as the number of iterations increases. Although PPO is considered as an "upgrade" of A2C, figure 3c shows that PPO is more unstable than A2C. Specifically, there are several momentary drop in performance in λ equals to 0.5, 0.7, 0.9, and 0.95. Meanwhiles, λ

[2]**SharedDrive:** https://drive.google.com/drive/folders/1adlMlMAl7jwFxOLdTJnXYQfEmJs8Pv8E?usp=sharing

equals to 0.97 and 0.99 are less unstable, but they both perform less effective compared to λ equals to 0.7 and 0.5. Since PPO agents with λ equals to 0.5 and 0.7 was able to gain rewards of +300, I conclude that they solved the bipedal walking problem. The recording shows that the robot was able to move from left to right of the environment. Similar to A2C, PPO agent is not able to learn the "normal walking" movement. The reason for this unrealistic walking is due to the shallow depth of actor network and critic network.

5.2 Comparison

In terms of run time, Q-Learning completes the training loop the fastest amongst the three algorithms due to the computational complexity of A2C and PPO, especially when they are trained with two neural networks. PPO takes longer to train compared to A2C since the algorithm reused the collected data, while A2C throws away data once used. Table 1 shows that Q-Learning with learning rate λ equals 10^{-4} trained the fastest among all Q-Learning λ, A2C with bias estimation rate of λ equals 0.7 trained the fastest among all A2C λ, and PPO with bias estimation rate of λ equals 0.95 trained the fastest among all PPO λ.

Table 1. The Max Training Time For 1000 Iterations

Q-Learning	secs	**A2C**	secs	**PPO**	secs
$\lambda = 10^{-4}$	248.1	$\lambda = 0.5$	5322.2	$\lambda = 0.5$	8845.2
$\lambda = 10^{-3}$	340.3	$\lambda = 0.7$	5126.2	$\lambda = 0.7$	8432.2
$\lambda = 10^{-2}$	331.7	$\lambda = 0.9$	5922.1	$\lambda = 0.9$	8537.2
$\lambda = 10^{-1}$	535.8	$\lambda = 0.95$	5931.6	$\lambda = 0.95$	8378.1
$\lambda = 0.3$	384.2	$\lambda = 0.97$	5726.1	$\lambda = 0.97$	9365.2
$\lambda = 0.5$	379.8	$\lambda = 0.99$	5643.6	$\lambda = 0.99$	10377
$\lambda = 0.7$	432.6				
$\lambda = 0.9$	529.5				

OpenAI provided a leaderboard[3] of BipedalWalker-v2 environment where each submitted agent is ranked based on the "Episodes before solve" (BipedalWalker-v2 defines "solving" as getting average reward of 300 over 100 consecutive trials). My Q-Learning algorithm, regardless of learning rate λ did not provide any agent that solve the bipedal walking problem. On the other hand, A2C algorithm with the bias estimate rate λ of 0.97 is able to provide a rewards of +300 at episode 20; while A2C algorithm with the bias estimate rate λ of 0.99 is able to provide a rewards of +300 at episode 1. Besides, my PPO implementation also solved the bipedal walking problem with the bias estimate rate λ of 0.9 gains +300 reward at episode 196, λ of 0.5 gains +300 reward at episode 675, and λ of 0.7 gains +300 reward at episode 48. Unfortunately, my results are not comparable to the ones submitted to OpenAI's Gym leaderboard for two reasons. Firstly, both my A2C and PPO agents did not execute normal walking behaviour. Secondly, the reason why the larger bias estimation rate in both A2C and PPO such as λ equals to 0.97 and 0.99 outperform other bias estimate rates is due to the training parameters are saved throughout all λ. This means that the larger the λ, the better the chance that it will solve the bipedal walking problem. With the current result, I consider the λ equals to 0.97 and 0.99 are trained for 5000 iterations and 6000 iteration respectively (These results are at the middle/bottom range of the leaderboard).

[3] **OpenAI's Gym Leaderboard:** https://github.com/openai/gym/wiki/Leaderboard

6 Conclusion

In this project, I have explained the mathematics, implemented the code, and shown relationship of Q-Learning, A2C, and PPO algorithms to facilitate the bipedal walking problem. Each algorithm has its strengths and weakness with its performance is compared in terms of graphs of Reward per Episode, the max training time for 1000 iterations, and the number of iteration it takes for each algorithm to reach the reward of 300. Q-Learning does not fit to solve BipedalWalker-v2 due to its discretization of continuous environment. A2C is outperformed by PPO, but it is more stable than PPO. As shown in the results, A2C and PPO have greater potential as RL method and would be a great research area. There are rooms to investigate and improve on this projects such as training in a larger number of iteration (50,000 for instance) or having deeper actor network and critic networks for A2C and PPO. With the help of powerful GPUs and parallel computing, the two mentioned improvement can definitely be done for this particular problem, and they may even effective enough to be applied to other problems, such as transfer learning to actual bipedal robots.

References

[1] Pieter Abbeel. *L3 Policy Gradients and Advantage Estimation (Foundations of Deep RL Series)*. 2021. URL: https://www.youtube.com/watch?v=AKbX1Zvo7r8&ab_channel=PieterAbbeel.

[2] Pieter Abbeel. *L4 TRPO and PPO (Foundations of Deep RL Series)*. 2021. URL: https://www.youtube.com/watch?v=KjWF8VIMGiY&ab_channel=PieterAbbeel.

[3] Evan Ackerman. *Bipedal Robots Are Learning To Move With Arms as Well as Legs*. 2021. URL: https://spectrum.ieee.org/bipedal-robot-learning-to-move-arms-legs.

[4] *Boston Dynamics's Atlas*. URL: https://www.bostondynamics.com/atlas.

[5] G. Brockman et al. *Openai Gym*. 2016. URL: https://arxiv.org/abs/1606.01540.

[6] Chris and Mandy. *Exploration Vs. Exploitation - Learning The Optimal Reinforcement Learning Policy*. 2018. URL: https://deeplizard.com/learn/video/mo96Nqlo1L8.

[7] DanielGörges. "Relations between Model Predictive Control and Reinforcement Learning". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 4920–4928.

[8] Laura Graesser and Wah Loon Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. 2018. URL: https://slm-lab.gitbook.io/slm-lab/publications-and-talks/instruction-for-the-book-+-intro-to-rl-section.

[9] Alexander Van de Kleut. *Actor-Critic Methods, Advantage Actor-Critic (A2C) and Generalized Advantage Estimation (GAE)*. 2020. URL: https://avandekleut.github.io/a2c.

[10] Alexander Van de Kleut. *Beyond vanilla policy gradients: Natural policy gradients, trust region policy optimization (TRPO) and Proximal Policy Optimization (PPO)*. 2021. URL: https://avandekleut.github.io/ppo.

[11] Jens Kober, J. Andrew Bagnell, and Jan Peters. *Reinforcement Learning in Robotics: A Survey*. 2013. URL: https://www.ri.cmu.edu/pub_files/2013/7/Kober_IJRR_2013.pdf.

[12] Russ Mitchell. *Two die in driverless Tesla incident. Where are the regulators?* 2021. URL: https://www.latimes.com/business/story/2021-04-19/tesla-on-autopilot-kills-two-where-are-the-regulators.

[13] V. Mnih et al. *Asynchronous methods for deep reinforcement learning*. 2016. URL: https://arxiv.org/abs/1602.01783v2.

[14] V. Mnih et al. *Playing Atari with deep reinforcement learning*. 2016. URL: https://arxiv.org/abs/1602.01783v2.

[15] *OpenAI's Gym BipedalWalker-v2*. URL: https://gym.openai.com/envs/BipedalWalker-v2/.

[16] J. Schulman et al. *High-dimensional continuous control using generalized advantage estimation*. 2018. URL: https://arxiv.org/abs/1506.02438.

[17] J. Schulman et al. *Proximal policy optimization algorithms*. 2017. URL: https://arxiv.org/abs/1707.06347.

[18] R.S Sutton and A.G Barto. *Reinforcement Learning: An Introduction*. 2018.