CS5180 Project Report

Bipedal Walker

Meishan Li, Zhida Zhang

Contents

Introduction

We worked on a bipedal robot walker problem using the BipedalWalker-v3 environment from the OpenAI organization. The visual environment includes a robot walker with a hull and two legs on horizontal terrain. We decided to work on this project because we were inspired by the bipedal humanoid robot named Atlas developed by Boston Dynamics, shown in figure 1.

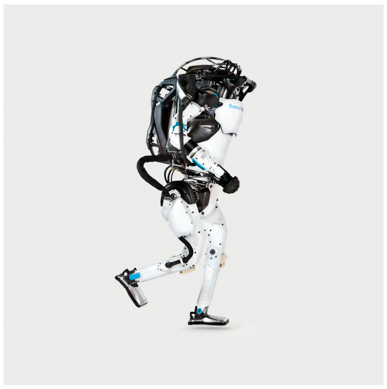


Figure 1a Humanoid bipedal robot walker named Atlas by Boston Dynamics

Atlas is a piece of art in the world of engineering, combining the most advanced intelligence of mechanical, electrical, computer engineering and computer science. What inspires us the most is that it can stand up, walk, run and jump over obstacles and even complete extremely difficult parkour tasks like backflips. It shares a lot of similarities to the bipedal walker we try to work on. We challenge ourselves to work on this project to better understand how reinforcement learning algorithms interact with the BipedalWalker-v3 environment.

Problem Statement

The main problem of this project is that the robot needs to be trained to select correct actions to stand up and eventually start to walk in an environment with a slightly uneven horizontal terrain. This is a *Markov-decision-process* (MDP) because the robot needs to choose a suitable action at each step under the current policy. Moreover, the agent receives a positive reward with respect to the distance walked on the terrain. A total of 300+ reward will be received if it walks all the way till the end[1]. However, the agent will get a reward of -100 if it tumbles. There is also some negative reward proportional to the torque applied on the joint. As a result, the agent needs to apply optimal action to learn to walk smoothly and efficiently to receive more points.

According to *OpenAI Gym BipedalWalker-v3*, state consists of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar rangefinder measurements, shown in figure 2[1]. We will study and understand how these parameters change would affect the decision making of the agent.

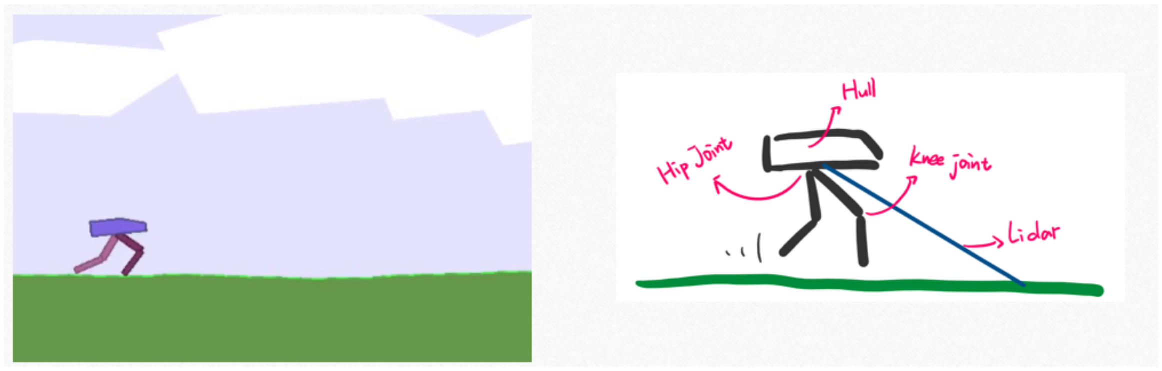


Figure 2. BipedalWalker-v3 environment and detailed annotation

Platform and Libraries

We used *OpenAI gym* which is a toolkit that provides a wide variety of simulated environments to help us compare and develop new reinforcement learning algorithms.

The main library that we use is *Pytorch*, an open source library used for studying reinforcement learning.

Algorithm

During the research process, we learned that the bipedal walker is a challenging problem with 24 observations and 4 actions. Moreover, the problem is a continuous control problem which requires algorithms like *Soft-Actor-Critic*(SAC), *Proximal Policy Optimization*(PPO), *Deep Deterministic Policy Gradient*(DDPG)and *Twin Delayed DDPG*(TD3) to solve. All of the algorithms mentioned above need an Actor-Critic Neural Network(AC network) to work. However, we did not plan enough time to research, study and fully understand the AC network and the advanced algorithms above. As a result, we based our project on the work of Rafael1s and others from Github on Deep-Reinforcement-Learning-Algorithms.

The work done by Rafael1s on the bipedal walker is complete[2]. His code trained and solved the BipedalWalker-v3 environment problem using the Twin Delayed DDPG algorithm. The best result of his work is shown below in figure 4. The training is done in 1086 episodes with a score of 304. Compared to other training results done in a larger number of sets, his work shows that the smaller the number of sets used to solve the problem, the better the training proved to be.

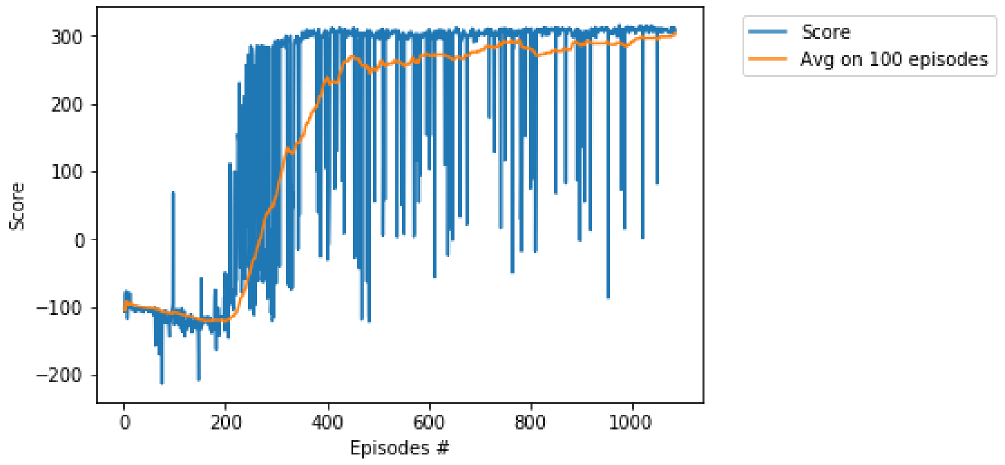


Figure 4. Best BipedalWalker-v3 training result by Rafael1s

Our main choice for the reinforcement learning algorithm is the deep deterministic policy DDPG/TD3 algorithm, pytorch as our framework. DDPG is a reinforcement learning algorithm based on the AC(Actor-Critic) framework[3]. For the AC structure neural network, the Actor network takes the observation and output action. The Critic network takes the observation and corresponding action, then outputs the Q-value which measures how good the action is. It maximizes the total reward by predicting a deterministic policy by updating the policy in a single step through the off-policy method, shown in figure 3.

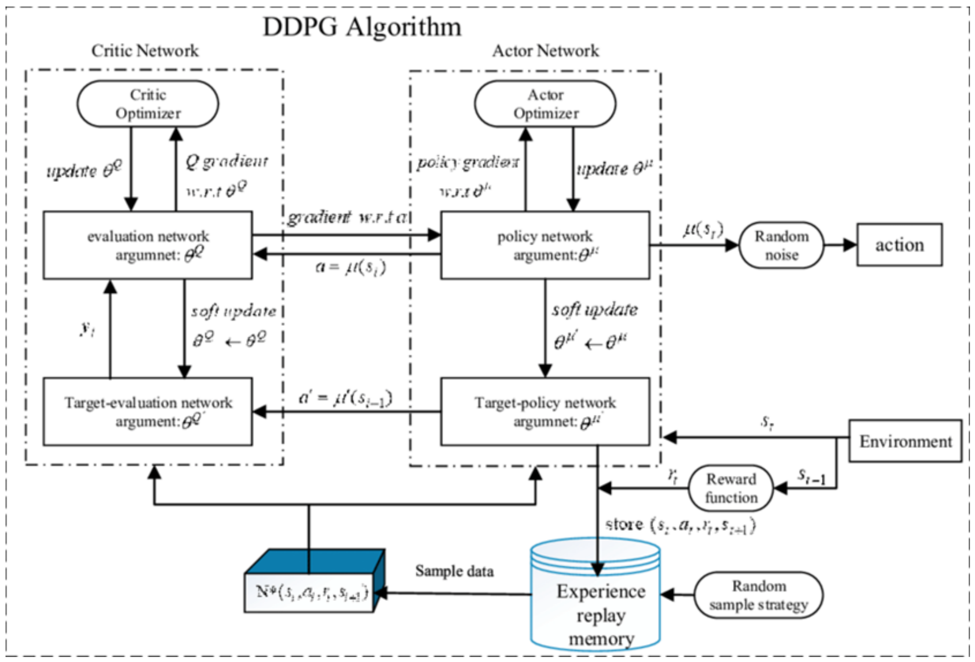

DDPG Algorithm
Critic Network
Actor Network
Critic Optimizer
Actor Optimizer
update θ^Q
Q gradient w.r.t θ^Q
policy gradient w.r.t θ^μ
update θ^μ
evaluation network argumnet: θ^Q
gradient w.r.t a
$a = \mu(s_i)$
policy network argument: θ^μ
$\mu(s_i)$
Random noise
action
soft update $\theta^{Q'} \leftarrow \theta^Q$
y_i
soft update $\theta^{\mu'} \leftarrow \theta^\mu$
Target-evaluation network argument: $\theta^{Q'}$
$a' = \mu'(s_{i-1})$
Target-policy network argumnet: $\theta^{\mu'}$
s_t
Environment
r_t
Reward function
s_{t-1}
store (s_t, a_t, r_t, s_{t+1})
Sample data
$N*(s_i, a_i, r_i, s_{i+1})$
Experience replay memory
Random sample strategy

Figure 3. DDPG Algorithm

Twin Delayed DDPG is using two sets of Critic network instead of one in DDPG. In TD3, two sets of Critic networks are used to represent different Q values and to suppress persistent overestimation by selecting the smallest one as the Target Q Value[4]. To be more specific, it uses the smaller of the two Q-values to form the targets in the Bellman error loss functions. Additionally, “delayed” means TD3 updates the policy and the target networks less frequently than the Q-function achieving better results[5]. Moreover, TD3 adds noise to the target action to make it more difficult for the policy to exploit Q-function errors by smoothing out Q along changes in action[4].

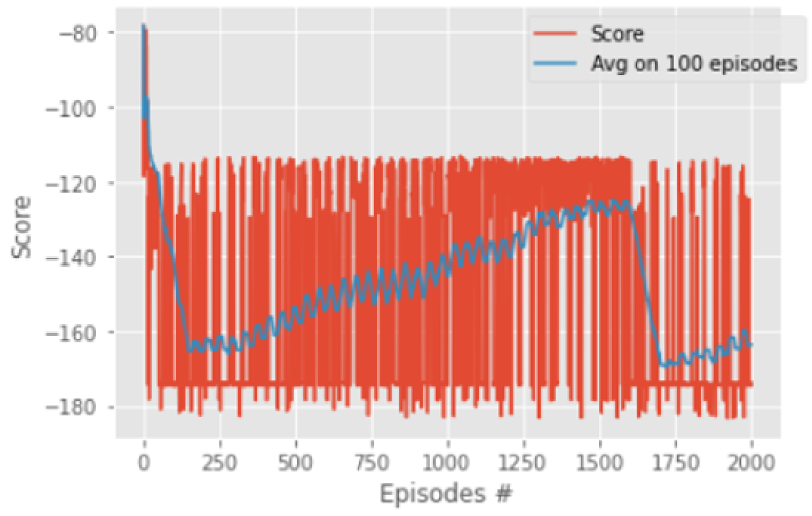
We compared algorithms such as A2C, PPO, etc. at the beginning of the project and finally chose the PPDG algorithm, which is similar to the DQN algorithm we had learned, because DDPG can handle continuous control problems that DQN cannot (maxQ(s', a') function can only handle discrete questions). And in the process of research we found that the TD3 algorithm is more efficient .

Empirical Results and Analysis

In our experiments, we focus on the comparison between the TD3 and DDPG algorithms. We wanted to verify the extent to which the improved TD3 algorithm affects the agent training performance. We performed modifications and tests, including the AC network and the delayed update part of the algorithm. We believe that all the additions made by TD3 will make the agent run better, while the enhancements from the different components may vary, we can still learn from them how to do it. Most of the experimental results are in line with our expectations, but some results still surprise us.

We first tried to train the agent using the DDPG algorithm and found that its learning was very poor, not even being able to get a positive score once, shown in plot 1 (BW_DDPG.ipynb).

```
length of scores:  2000 , len of average scores:  2000
```

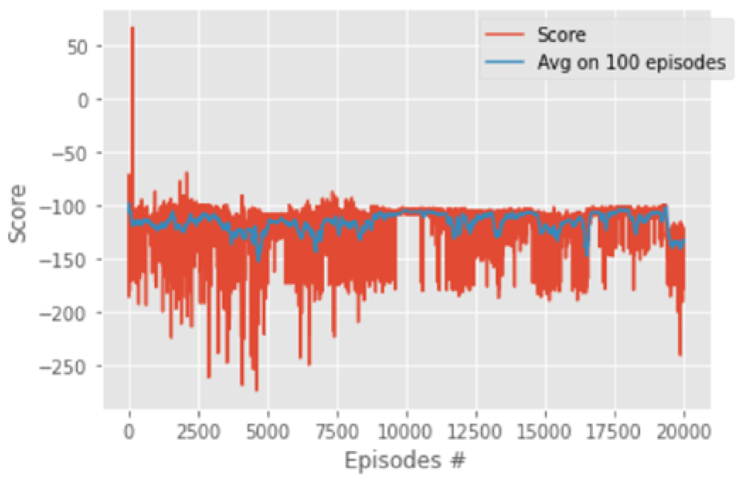


Plot 1. DDPG algorithm

Since TD3 uses an additional Critic network compared to DDPG and adds delayed updates, we try to understand the impact of these two additions on agent performance.

We make TD3 reduce a Critic network, which differs from DDPG only by delayed updates [BW_TD3_1C.ipynb] We find that the learning rate is very slow and almost indistinguishable from DDPG, and the running rate becomes slower, shown in plot 2.

```
length of scores:  20000 , len of avg_scores:  20000
```

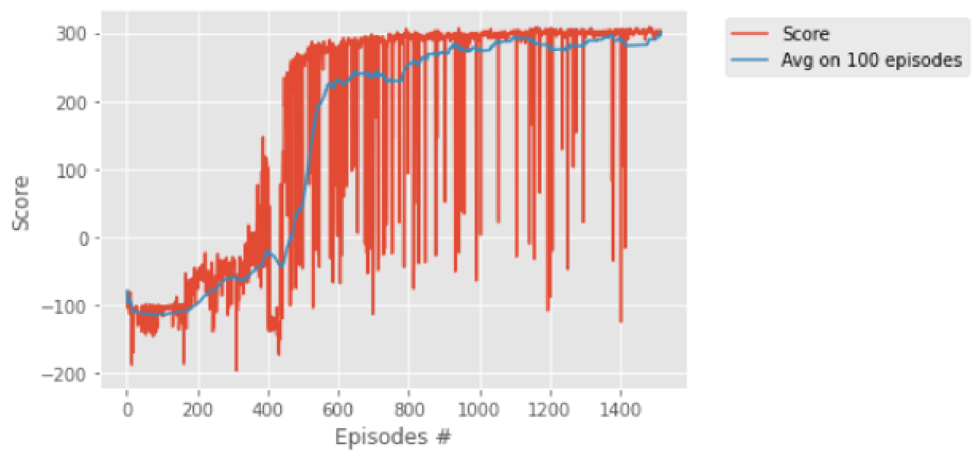


Plot 2. TD3 algorithm with only one Critic network

It looked like adding the critic network would lead to a higher learning rate, so we tried using 3 critic networks[BW_TD3_1C.ipynb] to see the effect on the results of the experiment. We found two very different results, the first time we ran it up to about 1500 episodes and learned about 270 points, very close to 300, but suddenly the average score dropped and less and less points were learned. The second run showed the same results as the DDPG algorithm, no learning at all.

We also tried to change the layers in the Actor-Critic network [BW_TD3_new_layers.ipynb], and the result was that the learning rate became slower, but we were still able to learn enough scores, shown in plot 3. This is also reasonable.

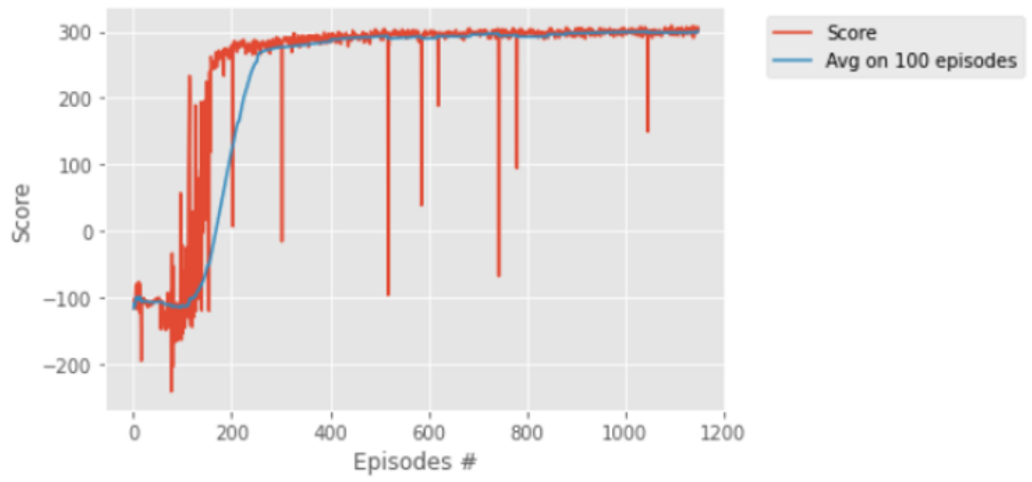
```
length of scores:  1514 , len of average scores:  1514
```



Plot 3. TD3 with new AC layers

We also compared whether the use of TD3 in the state without delayed update could make the agent learning rate change [BW_TD3_noDelay.ipynb] and found that the learning rate of "Twin DDPG" was even better than that of "Twin Delayed DDPG", shown in plot 4.

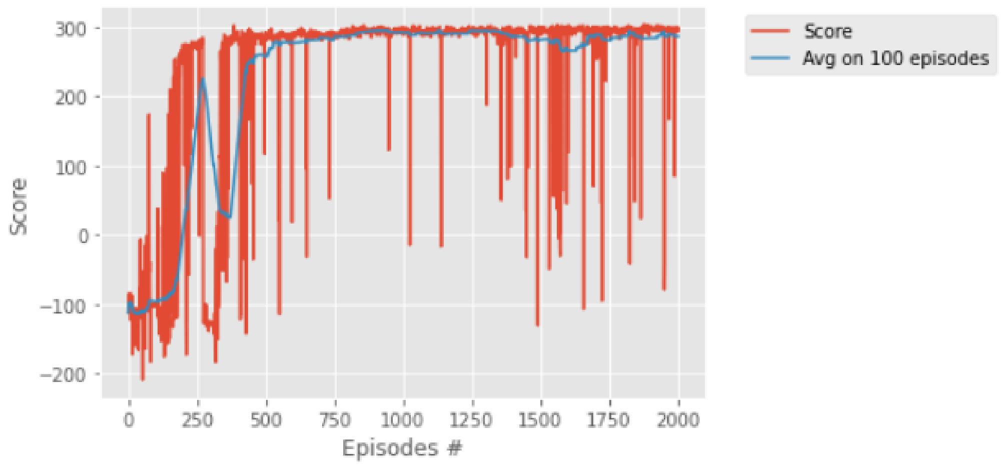
```
length of scores:  1148 , len of average scores:  1148
```



Plot 4. TD3 without delayed update

In order to avoid chance, we ran this modified algorithm again, and we can see that the results are not very stable this time, but the agent still starts learning around 100 episodes and reaches nearly 300 points sooner, so we can think that delayed update does improve the stability of the algorithm, but also makes the learning rate decrease, shown in plot 5.

```
length of scores:  2000 , len of average scores:  2000
```



Plot 5. TD3 without delayed update

Discussion and Future Direction

During the process of working on the semester-long project, we have encountered a few difficulties. First of all, the algorithm we planned to use was the Proximal Policy Gradient(PPO) algorithm. We took a look at the work found online and spent too much time trying to understand how their code works. Unfortunately, we gave up on the PPO algorithm due to the advanced coding technique they used on vectorizing the environment and how they built the neural network.

Based on our project proposal and the milestone done in the beginning, we realized that a lot of tasks did not go according to the plan. We planned to research, study, and understand to implement algorithms like Natural Evolution Strategies Algorithm, Genetic Algorithm, Augmented Random Search Algorithm, and Advantage Actor Critic Algorithm. However, we did not prepare ourselves for how difficult these algorithms are and some of them are not suited for the bipedal walker problem. As a result, we gave up on them and implemented other algorithms that are better fitted for our project.

We would definitely spend more time on studying the Actor-Critic Neural Network and understanding how different modifications done to it affects different algorithms if we had more time.

About the project, we would like to give the future students some advice. First of all, we think that reinforcement learning algorithms are a lot harder than they look. If we want to start over, we would start with simple algorithms that we learned in the class. Second, always try to stick to the week-by-week plan, because if the plan is left behind, it is nearly impossible to catch up. Lastly, do not put way too much pressure on yourself and your teammates.

Reference

[1] Huang, C. (2020). BipedalWalker v2
https://github.com/openai/gym/wiki/BipedalWalker-v2
[2] Rafael1s(2021). BipedalWalker with Twin Delayed DDPG (TD3)
https://github.com/Rafael1s/Deep-Reinforcement-Learning-Algorithms/tree/master/BipedalWalker-TwinDelayed-DDPG%20(TD3)
[3] Fujimoto, S., Hoof, H.V., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods
https://arxiv.org/abs/1802.09477
[4] Byrne, D. (2019). TD3: Learning To Run With AI
https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93
[5] Twin Delayed DDPG by OpenAI
https://spinningup.openai.com/en/latest/algorithms/td3.html