Deep Reinforcement Learning on BipedalWalker

Alec Reed
Computer Science Department
University of Colorado at Boulder
Boulder, US
alre5639@colorado.edu

Xuefei Sun
Computer Science Department
University of Colorado at Boulder
Boulder, US
xusu2102@colorado.edu

***Abstract*—In this paper, we will be teaching a 2D robot to walk using reinforcement learning. We will be using the OpenAI Gym BipedalWalker-v3 as our environment for this problem. We design a DQN network with discrete action outputs from scratch and achieve little to no success in training the robot to walk. We then implement a Deep deterministic policy gradient (DDPG) and successfully training the agent to complete the standard course and make some progress on the hardcore course (where there are steps, holes and walls). Implementing these two algorithms we show the importance of understanding the action space of the problem and design decisions that must be made when implementing reinforcement learning algorithms. Additionally, we will recommend steps that one may take to increase the performance of these already successful algorithms**

***Index Terms*—reinforcement learning, multi-joints robot, DeepQLearning, DQN, DDPG, Bipedal robots**

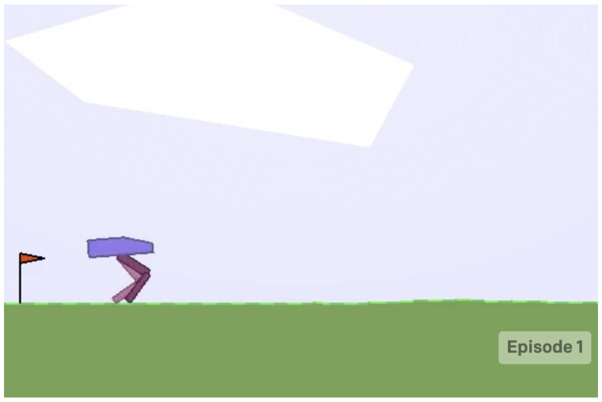
Fig. 1. Standard BipedalWalker-V3 environment.

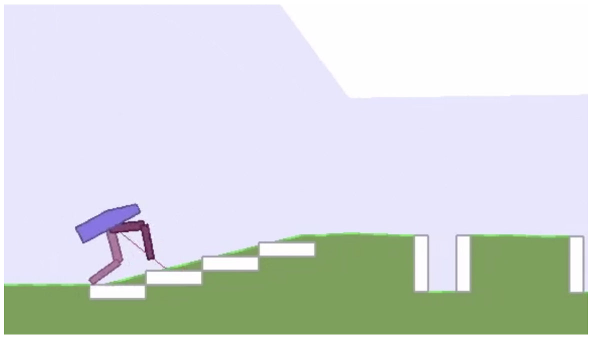
Fig. 2. Hardcore BipedalWalker-V3 environment.

I. Introduction

Bipedal robots increasing in popularity not only because we can design robots to look more ”human-like” but also can reduce energy usage when compared to wheeled vehicles and increase the range of achievable tasks. For example Stairs are a common problem for most wheeled robots, but a bipedal robot can be trained/programmed to identify obstacles such as stairs and traverse them.

However figuring out how to make robots walk is a difficult task. You could imagine a naive solution in which one designs an empirical tested transition function for any range of robot states. Not only is this approach incredibly tedious and time consuming, it is highly sensitive to error. For example, perhaps a designer didn’t take into account changing terrain when designing their transition function. Any robot designed in this method will not be able to navigate in environments which were not foreseen and explicitly designed for.

A better solution is to use reinforcement learning to train a robot to walk. The robot will simulate millions of ”attempts” to traverse the environment, and learn what state action pairs result in the highest reward.

II. Background and Related Work

A. reinforcement learning on robot

The robots have long moved out of research labs to venture into new spheres. The robots, such as Google’s worker robots, are proved to be not only capable but also effective at basic tasks and jobs at a lower cost compared to the human. To insure that every robot we use is operating the task as we wanted, if we manually control the robot, then we did not actually make the most use out of it. This is where reinforcement learning comes in as the robot autonomously discover an optimal behavior through trial-and-error interactions with its environment. There are different kind of robots, such as robot arms, autonomous cars and aircraft, thus the research conducted on robotics had greatly divided into different branches. [1] [2] [3] had specifically investigated into the robot arms and [4] [5] focused their work on the mountain cars using reinforcement learning.

B. Neural Network

Neural Networks(NNs), also known as artificial neural networks(ANNs) or simulated neural networks(SNNs), had played a key role in the machine learning and artificial intelligent field. Computer scientists decided to name the network because of the similarity between its structure and the biological neurons. Neural network is usually composed by multiple layers of neurons/nodes (an input layer, one or more hidden layers, and an output layer). The input might be some image features or language embeddings and the output is determined according to the specific task. For example, if the task a binary classification problem which is determine if the input is a dog or cat image, the output of the neural network can be a boolean value. If the output returns 1, the neural network thinks it is a dog image. The neural network is typically trained using stochastic gradient descent optimization algorithm and weights are updated using the backpropagation of error algorithm. Variations on the classic neural network design allow us to perform various tasks and scientists had developed specific types of artificial neural networks including feed-forward neural networks, recurrent neural networks and convolutional neural networks.

There are a lot of advantages of the neural network including the its parallel processing abilities enable the network to perform more than one job at a time. The neural network has the ability to learn hidden relationships in the data without commanding any fixed relationship, thus it is capable of modeling highly volatile data and non-constant variance. On the other hand, the neural network is facing the problem that it is highly dependent on the hardware. The neural network also requires a numerical input, which mean whatever you want to feed into the neural network, you have to transform the input into a numerical format first.

III. Problem Formulation

A. Environment

We are using the gym openAI BipedalWalker-V3 environment to train our agent to walk. This envirorment can be formulated to the following state spaces:

- Action space: actions are motor position values from [-1,1] for each of the 4 points at both hips and knees
- Observation space: state consists of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with the ground, and 10 lidar rangefinder measurements.
- Rewards: Reward is given for moving forward, totaling 300 points by the end of the course. Falling is -100 points. Applying any motor torque costs a small amount of points
- Termination: The episode will terminate if the hull contacts the ground or if the robot passes the end of the terrain length.

There are two different environments in which the Bipedal walker can operate, a standard environment with little slope (Fig. 1) and a hardcore environment (Fig. 2) that includes stairs, holes and walls the agent must traverse.

B. Deep Q Learning (QDN)

DQN is an reinforcement learning algorithm built on both neurak network and the Q learning which is a simply but fundamental algorithm that uses the Q value to predict the next action to take. The input for the network is the current state, while the output is the corresponding Q-value for each of the action.

We first execute some number of actions at random and generate a set of information including the current state, the rewards, the next state... Each set of information is seen as one training example and saved into the buffer for neural network training. But so far, there is no neural network training happens during this phase. After collected enough training samples, Q Neural Network is trained by adjusting the weight at each iteration and performs a stochastic gradient descent on the loss function. After every iteration, we save the current Q Neural Network into target Q Neural Network to ensure the stability. The Q Network predicts the Q-values of all actions that can be taken from the current state and we choose the action using ϵ-greedy algorithm. ϵ-greedy is a great way for us to balance exploration and exploitation by choosing between exploration and exploitation randomly. The ϵ-greedy, where ϵ refers to the probability of choosing to explore randomly which is at a very small chance but can improve its current knowledge and gain better rewards in the long run. However, when it exploits, the action taken is more reliable and more likely to get most reward, even if it is not the actual optimal behavior.

Compared to other reinforcement learning algorithms, training a DQN is very time consuming since we need to finetune the weight every time after we collect new bag of training samples. DQN can not be directly applied to the continuous action space since it tries to find the action that maximize the Q value computed by the Q Neural Network, which in the continuous valued environment requires an iterative optimization process at every step.

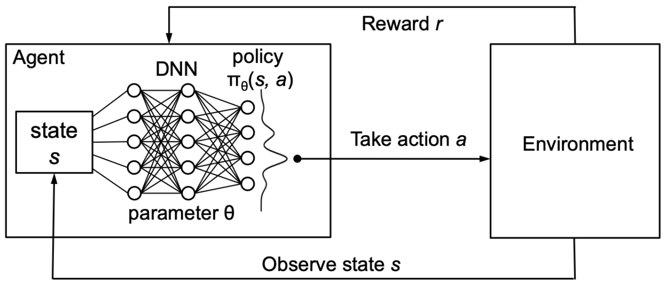

Fig. 3. Q Neural Network [6]

C. Deep Deterministic Policy Gradient (DDPG)

A better approach for this problem than DQN is DDPG. Given the nature of the action space, the agent needs to be able to select 4 actions (the position of its 4 legs joints) simultaneously. The best possible outcome for a DQN approach would be a very slow robot that could only move a single joint per state inpute. DDPG operates in continuous action spaces which allows for more than 1 action to be selected.

DDPG is shares many silimilaritys with DQN. Much like DQN, DDPG attempts to learn a Q function, DDPG also uses a

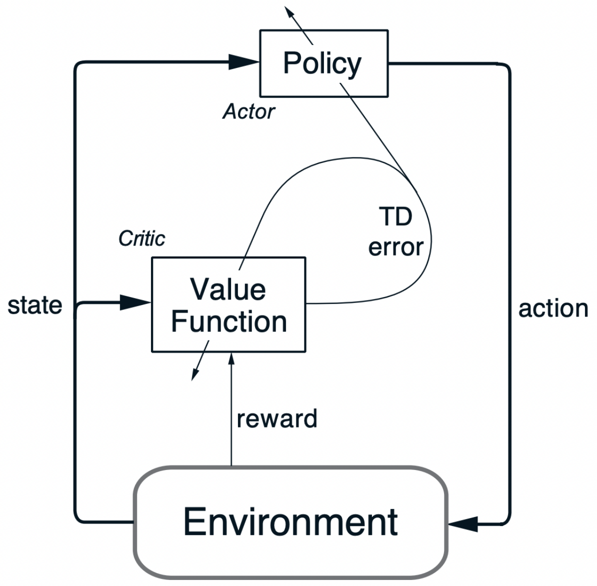

Fig. 4. Simple actor critic method [7]

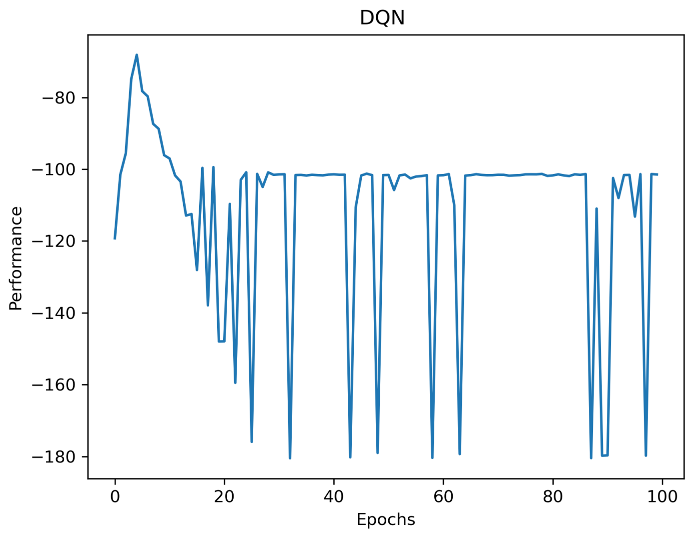

Fig. 5. Simple BipedalWalker loss plot. 100 Epochs

replay buffer like DQN and is therefore an off-policy learning algorithm. Unlike DQN however, DDPG is an actor critic algorithm. actor critic methods simultaneously train 2 neural networks. One network, the critic, estimates the Q values of the environment while the other network, the actor, updates the policy as suggested by the critic (Fig. 4). DDPG is very similar to the simple actor critic method, however rather than extracting actions form a probability distribution (like DQN), the actor network of DDPG directly calculates actions from the input states. This is the biggest advantage of DDPG in the BipedalWalker environment. Since there are 4 joints that all need to operate simultaneously for successful operation, DDPG allows us to give the actor netowrk 4 output nodes, each mapped to one joint of the 4 joint robot that provide continuous action outputs.

To use DDPG to train our walker we used Spinning Up [8], a deep learning library produced by OpenAI. Spinning Up provides an implementation of DDPG, which we were able to use in our environment to train the agent. Spinning Up allows for modification of many of the DDPG hyper parameters, allowing us to tune things like learning rate, number of epochs, and exploration noise.

IV. Results

A. DQN

DQN has a relative good start as the performance is increasing by the number of epochs we trained. But after around 20 epochs, the performance has consistently staying below -100 and more epochs would not help the performance. By debugging and reviewing the algorithm, we believe that we had correctly implemented the DQN for BipedalWalker. After then, we had done some research and found out the DQN only works well when action spaces that are discrete and low-dimensional [9]. BipedalWalker has four joints including Hips1, Hips2, Knee1 and Knee2. Each of the joint can take a value ranging from -1 to 1. In the experiment, we discretized the action space into 36 different actions modeling as (joint name, value). It is impractical for DQN to compute the optimal action on a continuous space as the optimal action might not be included in our discretized actions. We might solve this problem by transfer the action space into a finer grained discretization but the computation would be too expensive as the number of actions increases exponentially with the number of degrees of freedom.

B. DDPG

DDPG was very successful in the standard BipedalWalker environment (Fig. 6). This agent was trained for 100 epochs, with 4000 steps per epoch. the trained agent was able to complete the simple course nearly every time it ran.

The primary issue with the training policy is the noise in the reward. The ideal curve would resemble a logarithm curve, where once a good reward is reached the actor constantly executes that policy. While our maximum reward increases over the epochs, the agent frequently reverts back to worst policy. It is likely related to the noise that is added to the exportation policy. If we had some decay of the exporation noise I would expect a more reliable curve.

DDPG was less successful in the hardcore environment (Fig. 7 and Fig .8). The agent was unable to learn to walk in 100 epochs so the training time was increased to 1000 epochs. After 1000 epochs the agent was inconsitantly able to walk up stairs and over walls. The holes in the environment were the primary challenge for the agent. The agent was never able to complete the course in 10 runs of the fully trained policy.

We believe the main issue of the training is that the obstacles had no penalty associated with them. The agents often falls

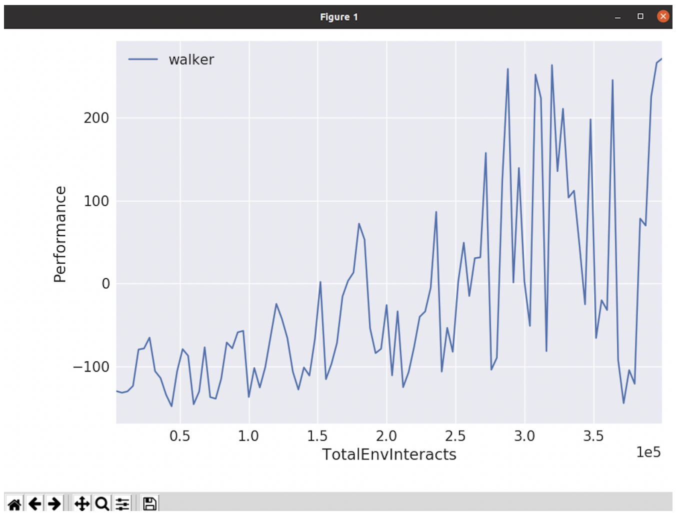

Fig. 6. Simple BipedalWalker loss plot. 100 Epochs

in holes and gets stuck, since the simulation only terminates when the hull contacts the ground or the end is reached, and therefore the network has no way to learn how to navigate the obstacles. One solution to this could be to end the simulation after some time of no movement, and apply a penalty. This may allow the agent to associate getting stuck on the obstacles with a negative reward.

Another potential solution for this issue would be to train on obstacles independently. for example add a single high wall in the simple environment. Once the agent is successfully navigating that, add a hole, and continue adding obstacles after the agent learns. since the exploration policy decays over time you would likely need to re-initialize epsilon when an obstacle is added to encourage exploration around the obstacles.

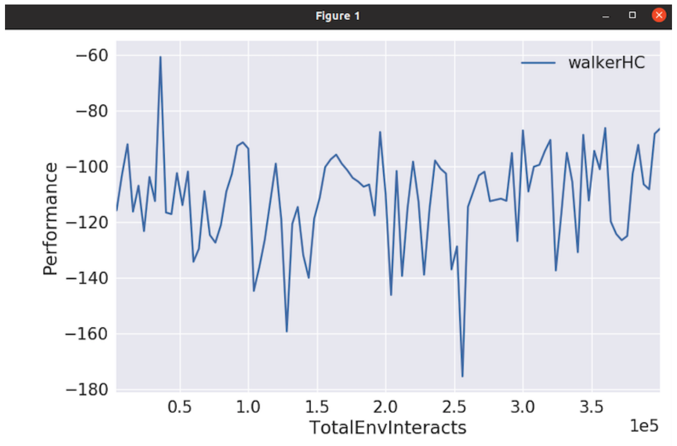

Fig. 7. Hardcore BipedalWalker loss plot. 100 Epochs

V. Conclusion

While an algorithm like DQN can be very effective in simple action spaces that can be logically discretized, the algorithm cannot be used more complex environments that

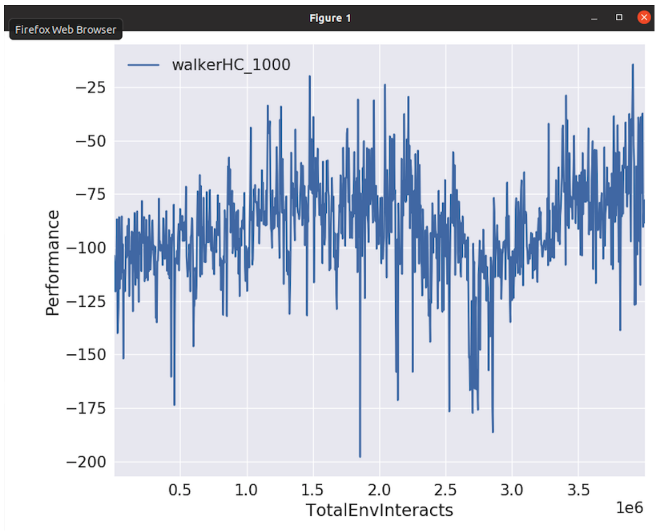

Fig. 8. Hardcore BipedalWalker loss plot. 1000 Epochs

require continuous action spaces. Although an environment like the BipedalWalker could potentially operate in a discreet action space, there is no way for the agent to properly explore to learn a Q function that would provide sufficient information to traverse the enviornment. DDPG is a simple and effective solution to this environment as it allows for multiple control surfaces (as outputs of the agents neural network) with continuous action spaces.

While the agent was consistently able to traverse the simple environment, it failed in the hardcore environment due to the obstacles provided unclear data for the agent to learn on. Given there is lidar data as part of the state of the robot, the robot should in theory be able to to traverse these obstacles, however negative re-enforcement is for the robot to learn not to get stuck on the obstacles.

A. *Future Work*

As a classic reinforcement learning problem, there are many algorithms that could be tested on this environment in future work. One could improve our algorithm by training the agent on individual obstacles rather than all obstacles at once. Additionally providing a clear indication to the robot that it has failed to navigate an obstacle would also increase the performance of the robot. Other algorithms for training the agent could also be explored.

One promising area is that of evolutionary learning. Deep evolutionary Reinforcement Learning (DEL) [10] for example attempts to develop a reinforcement learning algorithm that mimics how natural selection leads to the evolution of a species. The algorithm fosters many potential learners, and then selects learners that learn faster and evolve better in more complex environments and allows for learned traits to be passed down to descendent to use in their early learning life. it would be very interesting to see if this evolutionary method would be able to generate a agent capabile of learning to navigate the complex enviornment.

VI. Contributions and Release

Alec set up the environment, implemented DDPG, analyzed the result and wrote the report. Xuefei implemented DQN, analyzed the reason why DQN failed and wrote the report.

The authors grant permission for this report to be posted publicly.

References

[1] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention," 2021. [Online]. Available: https://arxiv.org/abs/2104.11203

[2] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni, "Robotic arm control and task training through deep reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2005.02632

[3] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1703.02949

[4] C. M. Bowyer, "Improving generalization in mountain car through the partitioned parameterized policy approach via quasi-stochastic gradient descent," 2021. [Online]. Available: https://arxiv.org/abs/2105.13986

[5] W. Knox, A. Setapen, and P. Stone, "Reinforcement learning with human feedback in mountain car." 01 2011.

[6] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[7] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[8] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[10] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei, "Embodied intelligence via learning and evolution," *Nature communications*, vol. 12, no. 1, pp. 1–12, 2021.