Bipedal Walker

Nguyen Quoc Anh
National University of Singapore
`e1124714@u.nus.edu`

Abstract

For reproducibility, all code for this project is here on github

1 Approach

Before discussing implementations, for evaluation, we benchmarked our model in the Bipedal Walker gym environment in 3 episodes. As previously mentioned, an episode terminates when the agent falls down, or the episode runs for 1600 steps in Normal mode and 2000 steps in Hardcore mode. Metrics used to compare models are mean and standard deviation of reward across 3 episodes. Since the reward is scaled to add up to 300 points if successful and deducted by 100 points if failure, if the run is successful, the mean would usually be roughly around 300. If the algorithm works for the environment consistently, then the standard deviation of reward should be as low as possible, and vice versa.

From numerous algorithms that were designed to solve sequential decision making problems, we have selected a few that are fitting for this problem.

1.1 Evolutionary Algorithm

1.1.1 Overview

Recently, researchers have proved that Evolutionary Algorithms (EA) can provide a viable alternative to solving problems usually solved with reinforcement learning[11]. While traditional genetic algorithms focused on evolving on the solution set, the policy required for Bipedal Walker is an array of continuous values. Using the Deep NeuroEvolution paradigm, we look to evolve the neural network mapping from state to action values. Although there are many methods to evolve [6], we chose to use the following in these steps with these exact hyper-parameters:
- S1: Initiate a population of 48 agents for a new generation.
- S2: For each agent in the population, run 3 episodes. The average reward will be used as the fitness function.
- S3: Selection: Select 25% neural networks that produce the best mean reward.
- S4: Crossover and Mutation: Choose 75% of the population randomly (weighted by its fitness), and use them to breed by masking and combining parameters to produce offsprings (same architecture, combined weights). Then the offspring is mutated with a rate of 10% of its weights, by adding some random noise.
- S5: Repeat S2 to S4 for 40 consecutive generations.

1.1.2 Implementation

We trained multiple neural networks in the aforementioned specifications, with the network containing 1 hidden layer of size 512 and tanh activation. After 40 generations for EZ mode, the best fit network output achieved a mean reward of 134.52 and a standard deviation of 8.89. After 60 generations for Hardcore mode, the best fit network achieved only -116.42 and 23.98 mean and standard deviation of reward respectively. The training at generation 60 have reached -82.97 mean reward, although in testing the network's score is much worse, suggesting that in the network has failed to learn a lot of possible states during its training.

1.2 Deep Deterministic Policy Gradients (DDPG)

1.2.1 Overview

DDPG is a model-free, off-policy, reinforcement learning algorithm [8]. Based on the Deterministic Policy Gradient [10] algorithm, it has 2 network, an Actor network that deterministically maps states to action that tries to maximize Q-value, and an Critic network that maps both states and the output actions to a Q-value that minimizes Mean Squared Bellman Error to get a better Q-value estimation. Inspired by Double Deep Q-Network [12], it also incorporate an experience replay buffer to minimize correlation between training samples, and a target actor-critic network that uses soft target update, to better stabilize the training.

1.2.2 Implementation

We first tried to train with a set of hyper-parameters used in pre-trained Stable Baselines 3 [9] model, however we found that the learning rate of 0.001 was too high and the model did not find a working policy for EZ mode. Then we lowered the learning rate to 0.0001, and successfully achieve a mean reward of 313.46 after 1,000,000 time steps and buffer size 20,000, using Multiple Layer Perceptron for each sub-network. Using the same setup, we trained for Hardcore mode, but the model failed to converge to an optimal policy within the number of time steps, with the mean reward never passing 1.1 during the training process. Best models are evaluated and saved every 25,000 time steps. For results, see Table 1

1.2.3 Comparing with EA

Although there are methods to parallelize NeuroEvolution, in training on a single CPU alone, EA takes much longer to train compared to DDPG, 11 hours vs 2.5 hours on the same r5.large AWS instance with 2 core 16GB RAM, and also failed to find a neural network robust enough to consistently get 300 points. Comparatively, our current implementation of EA lacks granularity, as it uses the mean reward after 3 episode as fitness criteria at the end, it cannot observe how specific action-state in each time step result in reward (unlike DDPG), and thus EA is receive less information from the environment. And since the sample used for training in EA are whole episodes and copies of neural networks, EA requires both long training time and large RAM to run. Thus, this implementation of EA is shown to be inferior to DDPG.

1.3 Reward Modification

As demonstrated by experiments in TD3, the default reward design of this gym environment has been able to guide the model to learn the goal of "keeping balance and walking with two legs" in even in Hardcore mode. However, if we were to define the goal for Bipedal Walker as "walking like a human", then some algorithms has been guided to find a locally optimal solution, since the reward system communicates less than adequate requirements for this purpose. For example, DDPG trained on EZ mode output the optimal policy such that the agent would use one leg behind to balance itself and the other to move itself forward [1], which is unlike how humans walk or run. Furthermore, the solution and training process is far from perfect. For the DDPG trained on EZ mode, both its found optimal policy and training process not as fast as it can be. And although we have shown that a different algorithm could increase both the running speed of the agent and decrease the training time till loss convergence, we also explored redesigning the rewards to induce these behavior, and trained DDPG on them.

By default, SB3 [9] runs the agent for 1600 steps if the agent's does not touch the ground (i.e. falls down). We observed that for a lot of the earlier iterations of the model training, the agent gets idle in one place, not moving for the rest of the episode [2]. This will not reward the model for moving forward, but also will not punish the model for moving its joints. From then on, any sample collected will consistently output a reward purely dependent on the hull's angle. Duplicates of samples provide no useful information and should not be put into the experience replay buffer, if the model utilizes it. Additionally, it waste training time by not ending the episode. Therefore we added a Reward wrapper called no_idle, that if the horizontal velocity of the agent's hull does not pass a threshold for 50 consecutive timesteps, it immediately gets -100 points, terminating the episode.

Secondly, we redesign the reward to incentivize the agent to increase speed, naming it run_faster. We use the base case in [1] and calculate the average horizontal velocity across 3 episodes as the speed threshold. Then, we wrote a reward wrapper that punish the agent for running below the speed threshold, and reward the agent for running faster. Since reward designing is a delicate task and the author of Bipedal Walker has definitely spend time tuning and scaling its reward, we choose the run_faster reward as a constant fraction of the punishment for applying a joint's torque per timestep (called $joint\_torque\_punishment$). Choosing to anchor run_faster reward value to an established reward value, we believe, is a reliable workaround to not have to tune the reward ourselves and yield reasonable results.

Combining run_faster and no_idle, DDPG was able to find an optimal policy that both balances the agent's hull while running faster, increasing the average speed from 4.83 in [1] to 6.05 (with the speed threshold being 4.2), decreasing the average timesteps per successful episode from 865.33 to 689.67. Furthermore, to run faster, the agent learned to run with both legs, like a human would[5].

Thirdly, we tried to design a reward that encourages the agent to hop. Many algorithms have optimized to an agent that use one leg to hop and the other raised forward to balance itself. Such policy has been shown to be viable in some Hardcore cases. We wrote $no\_leg\_contact$ to punish the agent by doubling its $joint\_torque\_punishment$ if leg 0 (the lighter color leg) touches the ground, and reward it 1/4 of its $joint\_torque\_punishment$ value if leg 0 does not. Unfortunately, DDPG found it easier to just runs as fast as it can to minimize the leg 0 contact[4].

Lastly, we tried to design a reward that encourages the agent to jump. As Hardcore includes a lot of obstacles, jumping is a valuable strategy to learn and apply. We wrote $jump\_higher$ to reward the agent when the hull's height is above certain height thresholds, and punish it otherwise. Interestingly, the optimal policy it found was to hop, so that it'll maintain its height while moving forward, and run in really small steps, so that the agent can keep its legs as straight as possible[3].

Reward design is tricky and can lead to unexpected emergent behaviors. Since we may have unwittingly induce hopping with $jump\_higher$, another redesign could be to directly induce hopping by rewarding the agent for keeping its joint at a certain height. Right now, to encourage the agent to keep balance, the agent is being punished for keeping its hull at any other angle than 0 degrees from the horizontal axis. Since solving the Hardcore will require the agent be more flexible in its movement, another useful redesign could be to relaxes this reward and only punish when the hull is outside a certain angle range. This will allow the agent to explore more diverse policy without being constantly punished (for example, in this demonstration [7], the agent has to angle its hull to pass the environment). Further tuning and scaling the added reward could also yield better results. In conclusion, model improvement is just one of the methods to solving a problem; reward design or even adding more observations to the agent can have as much potential to explore and find an optimal policy.

Member	Setting	Mean Reward	Std-dev Reward
EvoAlg	EZ	134.52	8.9
EvoAlg	Hardcore	-116.42	23.98
DDPG	EZ	313.46	1.04
DDPG	Hardcore	-97.21	15.57

Table 1: Metrics of evaluation for all algorithms

References

[1] Nguyen Quoc Anh. Ddpg trained on gym's bipedal walker, normal mode. https://drive.google.com/file/d/1Bx3fY300LXJQzhUtKl6bdi608eTS9wcY/view?usp=sharing, 2023.

[2] Nguyen Quoc Anh. Ddpg trained on gym's bipedal walker, hardcore mode, episode 200, idle issue. https://drive.google.com/file/d/1bqCT2pXUPhv0lFfv5kmRQhIZQ4VzPidP/view?usp=sharing, 2023.

[3] Nguyen Quoc Anh. Ddpg trained on gym's bipedal walker, normal mode, no_idle and jump_higher reward wrapper. https://drive.google.com/file/d/1lcvT5wTtHFS7DxCINL0clHMy5Hx7Xcl8/view?usp=sharing, 2023.

[4] Nguyen Quoc Anh. Ddpg trained on gym's bipedal walker, normal mode, no_idle and no_leg_contact reward wrapper. https://drive.google.com/file/d/1gXb91a8G9cIHYFTqt8pfHNG3AXzR_anv/view?usp=sharing, 2023.

[5] Nguyen Quoc Anh. Ddpg trained on gym's bipedal walker, normal mode, no_idle and run_faster reward wrapper. https://drive.google.com/file/d/1RfRpmMcGTjrzdafL-PSt7xAfHukzBWEQ/view?usp=sharing, 2023.

[6] Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *CoRR*, abs/2006.05415, 2020. URL https://arxiv.org/abs/2006.05415.

[7] Uber AI Labs. Enhanced poet: Open-ended reinforcement learning. https://www.youtube.com/watch?v=QlPnjpYghAk, 2020.

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[9] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[10] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 387–395. JMLR.org, 2014. URL http://dblp.uni-trier.de/db/conf/icml/icml2014.html#SilverLHDWR14.

[11] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017. URL http://arxiv.org/abs/1712.06567.

[12] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.