Solving the Bipedal Walker Problem with TD3 Reinforcement Learning Algorithm

Andrei Eugen Lapusneanu, Ana-Maria Lupoaea

January 16, 2024

Abstract

This report explores the use of the Twin Delayed DDPG (TD3) algorithm to solve the Bipedal Walker problem in reinforcement learning. TD3 proves effective in training the agent for stable locomotion and navigating the bipedal robot through challenging environments. The results demonstrate improved learning and successful completion of the task. Performance metrics and comparisons with other reinforcement learning approaches are discussed, providing a comprehensive overview of TD3's efficacy in addressing this challenging problem.

1 Introduction

This report thoroughly explores the application of the Twin Delayed DDPG (TD3) reinforcement learning algorithm to solve the Bipedal Walker problem. The study is centered around efficiently applying TD3 to optimize the locomotion of a bipedal-legged robot across diverse environments. By examining benchmark scenarios with different complexities, we assess the algorithm's performance and effectiveness in delivering competitive solutions for the Bipedal Walker problem.

The report begins with a quick introduction, providing an overview of the Bipedal Walker problem.

The Related Work section reviews existing literature, setting the context for our approach.

In the Algorithm section, we outline the application of the Twin Delayed DDPG (TD3) algorithm, detailing its customized features for addressing the Bipedal Walker problem.

Next, the experimental setup is described.

In the next section, we present and interpret the result of the experiments with respect to the quality of the result and the time needed for the program to compute it.

Finally, the last section summarizes the conclusions that have been drawn after the experiment, and the last section contains all the sources that have been used for inspirational purposes.

2 Problem Definition

2.1 Reinforcement Learning

Reinforcement learning is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives when interacting with a complex, uncertain environment. In other words, it is a type of machine learning that focuses on how an intelligent agent should take actions in a dynamic environment to maximize cumulative reward. Unlike supervised learning, reward-based learning does not require a labeled dataset. Instead, the agent learns by interacting with the environment and receiving feedback in the form of rewards or penalties. Reinforcement learning has been successfully applied to a wide range of problems, including game playing, robotics, and recommendation systems. [1]

2.2 Bipedal Walker

The Bipedal Walker problem is a classic challenge in the field of reinforcement learning. It involves training a two-legged agent to navigate a complex terrain. The agent must learn to balance and move forward without falling over, despite the irregularities and obstacles in the terrain.

In the Bipedal Walker problem, the agent operates in an environment defined by specific characteristics: [2]

1. **Action Space -** Actions are motor speed values within the [-1, 1] range for each of the 4 joints located at both hips and knees.
2. **Observation Space -** The state includes various parameters such as hull angle speed, angular velocity, horizontal and vertical speed, joint positions and angular speeds, legs' contact with the ground, and 10 lidar rangefinder measurements. Notably, there are no coordinates in the state vector.
3. **Rewards -** The agent receives a reward for moving forward, accumulating up to 300+ points toward the far end of the terrain. Falling results in a penalty of -100 points, and applying motor torque

incurs a small cost. A more optimal agent achieves a higher score.

4. **Starting State** - The walker begins the episode standing at the left end of the terrain with the hull in a horizontal position, and both legs are in the same initial position with a slight knee angle.

5. **Episode Termination** - The episode concludes if the hull makes contact with the ground or if the walker exceeds the right end of the terrain length.

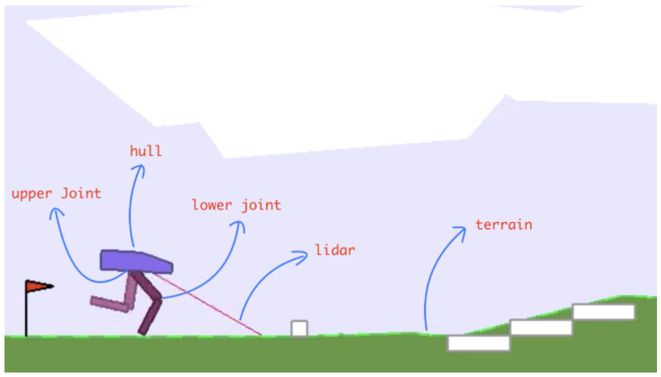

Figure 1: Bipedal Walker

3 Related Work

The Bipedal Walker problem has been a subject of interest in the field of Reinforcement Learning, with multiple solutions proposed to tackle it. Some popular solutions are:

- **Deep Q-Learning** - Deep Q-Learning uses a deep neural network to approximate the Q-function, which can handle high-dimensional state spaces. However, it often prematurely converges to suboptimal local maxima due to the coarsely discretized action space. This means that the algorithm may settle for a less optimal solution because it cannot distinguish between different actions that lead to similar outcomes. Despite this, Deep Q-Learning has been widely used in various reinforcement learning problems due to its simplicity and effectiveness. [3]

- **Monte Carlo Learning** - Monte Carlo methods are part of a broader class of algorithms in reinforcement learning that solve the problem by averaging sample returns. For the Bipedal Walker problem, Monte Carlo Learning can provide a straightforward and effective approach. However, one of the main challenges with Monte Carlo methods is that they can only apply updates after the completion of an episode. This means that Monte Carlo methods can be inefficient in environments where episodes are long. [4]

- **DDPG (Deep Deterministic Policy Gradient)** - DDPG is a robust reinforcement learning algorithm tailored for tasks with continuous action spaces, such as the Bipedal Walker problem. By combining an actor-critic approach with deep neural networks, DDPG efficiently approximates policy and value functions. Its deterministic policy learning enhances stability, though it demands careful hyperparameter tuning and management of the exploration-exploitation trade-off. Despite these challenges, DDPG has demonstrated efficacy in diverse robotic control applications, showcasing its versatility in reinforcement learning scenarios. [5]

4 Proposed Solution

The Twin Delayed DDPG (TD3) algorithm emerges as a powerful solution for addressing the Bipedal Walker problem in reinforcement learning. Built upon the foundation of DDPG, TD3 incorporates several key refinements to enhance stability, robustness, and overall performance during training.

While DDPG can demonstrate impressive performance in certain scenarios, it frequently proves to be delicate in terms of hyperparameter tuning. A common pitfall in DDPG is the tendency for the learned Q-function to excessively overestimate values, leading to potential policy instability. Twin Delayed DDPG (TD3) addresses this challenge through the implementation of three key enhancements: [6]

- Clipped Double-Q Learning: TD3 adopts the concept of ”twin” Q-functions, training two instead of one. The smaller of the two Q-values is utilized to shape targets, contributing to a more stable learning process.

- “Delayed” Policy Updates: TD3 strategically reduces the frequency of policy updates (and target network updates) compared to Q-function updates. The recommended ratio is one policy update for every two Q-function updates.

- Target Policy Smoothing: TD3 introduces randomness to the target action, making it challenging for the policy to exploit errors in the Q-function by introducing a smoothing effect on Q-values during changes in action.

By doing these three things, TD3 significantly improves how well the algorithm performs compared to the original DDPG.

The algorithm could be described as follows:

1. **Initialization:**

- Set up initial parameters for the actor and twin critic networks, as well as their target networks. For both actor and critic we used neural networks having 3 dense layers.
- Create a replay buffer to store experiences during training.
- Define hyperparameters like exploration noise, discount factor, policy update frequency, and target policy smoothing coefficient.

2. **Training Loop:**
   (a) For each episode:
      i. Start in an initial state.
      ii. For each time step within the episode:
         - Choose an action using the current policy with some exploration noise.
         - Observe the next state, the reward, and whether the episode is done.
         - Store this experience in the replay buffer.
         - Sample a batch from the replay buffer for training.
         - Update the twin critic networks based on the observed rewards.
         - Periodically update the actor policy based on the critic's feedback.

3. **Policy Evaluation:**
   - After training, use the learned actor policy for decision-making in the environment.

5 Experimental Setup

5.1 Parameters

In our quest to develop an effective solution for the Bipedal Walker problem, we recognize that finding an optimal configuration requires a comprehensive exploration of various hyperparameters, each influencing the learning process in distinct ways. In the initial stages of our project, we started with a set of default values based on existing literature and initial assumptions. Subsequently, we systematically adjusted these values based on the results obtained from our experiments, fine-tuning each parameter to enhance the performance and stability of our reinforcement learning algorithm. Central to our analysis were the following parameters:

- **Learning rate** - Ensuring an appropriate learning rate is crucial in managing the delicate balance between convergence speed and stability. If the learning rate is excessively high, the model risks overshooting optimal weights, whereas an excessively low learning rate may lead to sluggish convergence or being trapped in local minima. The most effective value we experimented with was **0.001**.

- **Maximum number of episodes** - Setting a maximum number of episodes helps to control the training duration and prevents the algorithm from running indefinitely. It is useful for limiting computational resources and ensuring that the agent has sufficient exploration time without training for an excessive number of episodes. The parameter initially had a value of 1000, but we chose to raise it to **1500** during our experiments.

- **Gamma** - Represents a discount factor that determines the importance of future rewards in the agent's decision-making. A higher gamma value makes the agent more farsighted, considering future rewards more heavily. Lower gamma values make the agent focus more on immediate rewards. A proper value for our solution proved to be **0.99**.

- **Polyak** - Determines the rate at which the target networks are updated and controls how much of the new weights are integrated into the target networks at each update, helping to stabilize the training. A smaller POLYAK value makes the target networks update more frequently, providing faster convergence but potentially leading to more instability. A larger POLYAK value leads to slower updates, providing more stability at the cost of slower convergence. **0.995** proved to be the most suitable value for this parameter.

- **Policy noise** - Introduces exploration noise to the selected actions during training. Adding noise to the actions allows the agent to explore a wider range of actions, improving its chances of discovering better policies and preventing it from getting stuck in suboptimal policies. The value **0.2** demonstrated the highest efficiency in our experiments.

- **Noise clip** - Ensures that the exploration remains controlled. Clipping the exploration noise helps in preventing extreme and unrealistic actions, ensuring that the noise doesn't dominate the agent's actions. It provides a balance between exploration and maintaining reasonable actions. A value of **0.5** led to the best results.

- **Policy delay** - Ensures that the Q-values have sufficient time to stabilize and provide more accurate, reliable feedback. Delaying the policy helps mitigate potential instabilities that may arise if it is updated too frequently, contributing to a more stable and effective learning process.

6 Results

The training progress is shown in graphs that display the rewards received in each training episode, the average reward over time, and the reward achieved during testing. The agent's improvement is clear, reaching a reward of 300 in less than 1000 episodes. This quick improvement indicates that the algorithm effectively trained the agent, leading to good performance not only in training but also in testing. The consistent high rewards during testing show that the algorithm can adapt well, handling the challenges of the environment. The graphs visually confirm the algorithm's success in learning the task and becoming proficient in a relatively short training period.

These results are the best achieved, as the agent not only successfully completes the game but also moves adeptly. This efficiency highlights the algorithm's success in quickly training the agent to master the task. A detailed exploration of these results and a thorough comparative analysis with other results we got can be found in the Ablation Study section.

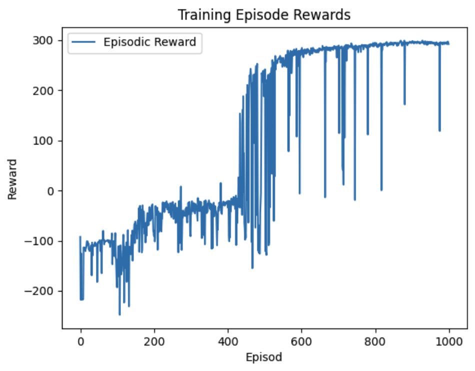

Figure 2: Training Reward On Run A

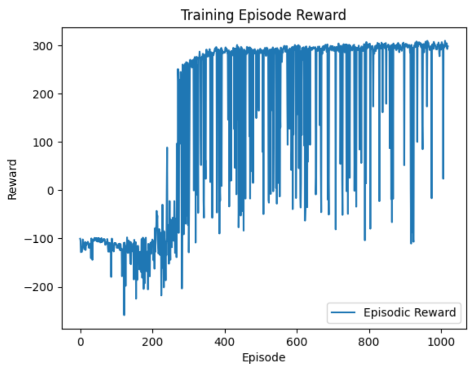

Figure 3: Training Reward On Run B

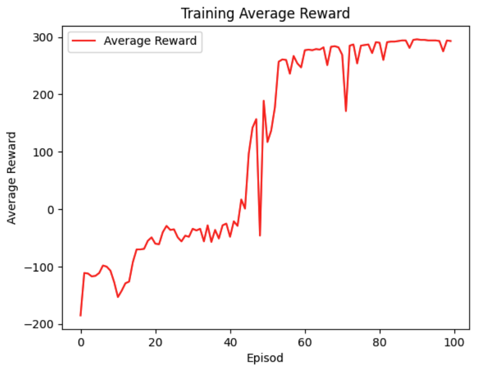

Figure 4: Average Training Reward On Run A

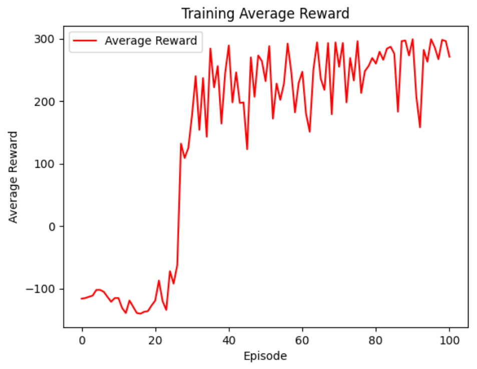

Figure 5: Average Training Reward On Run B

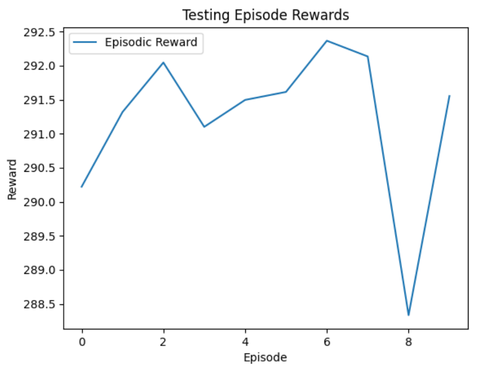

Figure 6: Testing Reward - Agent Trained in Run A

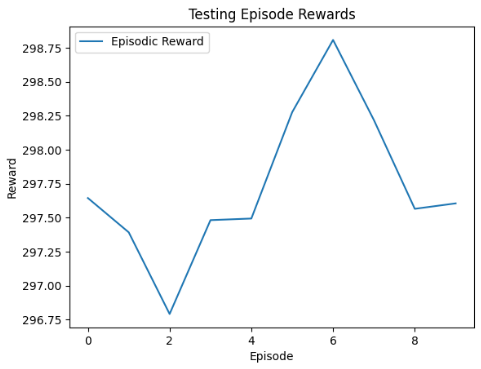

Figure 7: Testing Reward - Agent Trained in Run B

7 Ablation Study

In our exploration of different approaches, the first strategy involved generating arrays for all possible movements of each knee and joint with values between [-1, 1]. A neural network with three dense layers was employed to predict the agent's next action, specifically focusing on the movement of a single knee/joint. Despite the incorporation of Q-learning and experience replay for reinforcement learning, the outcome was not as expected. It became apparent that a different approach might yield better results, prompting our consideration of an Actor-Critic implementation.

Subsequently, we attempted a second variant, where we generated an array containing possible movements with integer values from {-1, 0, 1}. Again, a neural network with three dense layers was utilized to predict the agent's actions, encompassing the movements of each knee and joint. In addition to Q-learning and experience replay, we introduced ϵ-greedy exploration as a means to balance exploration and exploitation during training. The ϵ-greedy strategy involves choosing a random action with probability ϵ (exploration) and selecting the action with the highest estimated value with probability 1-ϵ (exploitation). Despite these enhancements, this approach also fell short of achieving the desired outcomes.

Given the limitations of the aforementioned strategies, we eventually turned to an Actor-Critic implementation, a hybrid model that combines elements of both policy-based (Actor) and value-based (Critic) methods. This choice proved to be more effective in successfully training the reinforcement learning agent, providing a more stable and robust solution to navigate the complexities of the task at hand.

Despite the increased computational cost associated with the final solution, which involved the use of multiple neural networks (Actor, 2 Critics, Targets), as opposed to a single neural network utilized in the previous two approaches, the effectiveness of the final solution outweighed the drawbacks. The initial strategies, relying on arrays and a simpler neural network, demanded approximately 5000 episodes for training, yet failed to achieve satisfactory results comparable to the proposed solution. In stark contrast, the Actor-Critic implementation presented here required only 1000 episodes for training, showcasing a significantly accelerated learning process.

The final method worked better not only in how quickly it learned but also in how well the agent performed. The earlier tries had problems with the agent moving slowly and sometimes getting stuck, especially when there were differences in the terrain levels or when it relied too much on one leg. On the other hand, the Actor-Critic model could smoothly and easily navigate the game environment. What's crucial is that the suggested solution not only got better results but also successfully finished the task, handling the complexities of the game well. In comparison, the earlier methods often led to the agent getting stuck or not doing as well.

In conclusion, even though the Actor-Critic method needed more computer power, its better performance, quicker learning, and ability to master the task more effectively make it the better choice. The mix of policy-based (Actor) and value-based (Critic) methods, along with using target networks, turned out to be a successful strategy. This highlights how using a combination of approaches is important when dealing with challenging reinforcement learning problems.

8 Conclusions and future directions

In conclusion, our systematic hyperparameter tuning process has resulted in a set of values that significantly improved the learning efficiency and stability of our reinforcement learning agent. As we look ahead to future work, our research trajectory will encompass several key directions.

Firstly, we plan to extend our experimentation to include the hardcore variant of the BipedalWalker game, exploring the adaptability and generalization capabilities of our agent in more demanding scenarios. Additionally, our focus will be on refining and further tuning the existing TD3-based solution, seeking ways to enhance its performance and address any remaining challenges.

Furthermore, we are eager to explore alternative optimization techniques, considering state-of-the-art approaches and methodologies that might offer additional improvements. The incorporation of additional hyperparameters, as well as the investigation of different algorithmic approaches, will be part of our ongoing efforts to push the boundaries of our agent's capabilities.

In parallel, the optimization of computational performance will be a key area of emphasis. We aim to identify and implement techniques that streamline the training process, ensuring our solution remains efficient and scalable as we tackle more complex tasks in the realm of reinforcement learning.

All in one, our future work aims to extend beyond the current achievements, refining our current solution and exploring innovative strategies to elevate the overall performance of our agent.

References

[1] Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barton

[2] Online Gym Documentation, Bipedal Walker, accessed on January 2024

[3] Teaching a Robot to Walk Using Reinforcement Learning, Jack Dibachi and Jacob Azoulay

[4] OpenAI Bipedal Walker, Derek Brenner and Sidney Lafontaine

[5] Bipedal Walking Robot using Deep Deterministic Policy Gradient, Arun Kumar, Navneet Paul, S N Omkar

[6] Twin Delayed DDPG, Online OpenAI Documentation, Accessed on January 2024