

Figure 1: Bipedal Walker Environment and Agent

1 Problem Definition

The Bipedal Walker environment is a well-known benchmark in reinforcement learning (RL) for testing agents in continuous control tasks. This environment simulates a bipedal agent that must learn to walk across terrain by controlling four joints. The challenge is considered solved if the agent achieves an average score of over 300 points across 100 runs. This means it must learn how to balance and coordinate its movements.

States

The state space is represented by a 24-dimensional vector, containing the agent’s current configuration and environment, including:

• Hull Angle and Angular Velocity • Horizontal and Vertical Speeds

• Position and Angular Velocity of Joints • Legs Contact with Ground

• Lidar Rangefinder Measurements

Each dimension has defined upper and lower bounds that constrain the sensor readings, as highlighted in the appendix.

Actions

The action space is represented by a 4-dimensional vector, with each dimension corresponding to a control signal for one of the four joints (two hips and two knees). The action values range between [-1,1] where:

• -1 represents the maximum speed in one direction

• 1 represents the maximum speed in the opposite direction • 0 represents no movement.

As part of its training, the agent must learn to apply the correct torque to these joints to achieve stable and efficient walking.

Transition Dynamics

The transition dynamics are governed by the Box2D physics engine, which simulates interactions between the agent and its environment. Given a state and action, the environment deterministically calculates the next state, considering factors like gravity, friction, and joint forces. Transition proba-bilities are not explicitly provided, as the environment relies on a deterministic physics simulation.

Reward Function

The reward function is designed to encourage the agent to move forward efficiently while penalising energy expenditure and falls:

• The primary reward is based on the distance moved forward along the terrain. The agent can accumulate over 300 points for successfully reaching the far end of the map

2

• A small penalty is subtracted from the reward for applying motor torque. This encourages the agent to find an energy-efficient strategy which minimises unnecessary movements or excessive joint actuation

• If the agent’s hull touches the ground, the episode ends immediately, and the agent receives a penalty of -100. This discourages the agent from falling and promotes learning a stable walking pattern

Initial State

At the beginning of each episode, the agent starts in a pre-set configuration:

• The agent’s hull is horizontal

• Both legs are in the same initial position

Episode Termination

An episode can end under the following conditions:

• If the agent falls and its hull touches the ground,

• If the agent successfully reaches the right end of the terrain,

• If the agent exceeds the maximum number of allowed time steps (1,600 steps for the normal version and 2,000 steps for the hardcore version) without either falling or reaching the end of the terrain.

2 Background

Continuous action spaces introduce unique challenges due to the infinite size of the action set (making exhaustive search impossible). This necessitates a different suite of algorithms that can effectively explore and learn about these environments.

Policy Gradient methods are well-suited to continuous action spaces. Unlike value-based meth-ods (which derive policies indirectly from value functions), policy gradient methods directly op-timise the policy, to maximise the expected cumulative reward. The objective being to maximise

J(θ) = Eτ∼π [R(τ)] where τ denotes a trajectory and πθ represents some policy (typically a normal

θ

distribution over actions given states).

The gradient of this objective can be derived using the REINFORCE algorithm [5], where the policy parameters are updated using the gradient ∇θJ(θ) = Eτ∼πθ [∇θlogπθ(at|st)R(τ)]

The strengths of policy gradient methods include their expressiveness, as they can naturally handle high-dimensional and continuous action spaces, and their ability to model stochastic policies, which aids in exploration. However, they suffer from high variance in gradient estimates, leading to unstable updates and slow convergence, and are often sample-inefficient.

Policy gradient methods have shown success in various continuous control tasks. The Deep Determin-istic Policy Gradient (DDPG) algorithm [1] extends the deterministic policy gradient [3] to work with deep neural networks, demonstrating effectiveness in complex robotic manipulation tasks. The Trust Region Policy Optimisation (TRPO) algorithm [2] introduces a trust region constraint to stabilise policy updates, significantly improving performance in continuous control environments like the MuJoCo [4] suite.

Actor-Critic methods combine the strengths of policy gradient and value-based methods. In these algorithms, the "actor" refers to the policy (which selects actions), and the "critic" refers to the value function (which evaluates the action taken by the actor). The actor’s policy is updated based on feedback from the critic, helping to reduce the variance of the gradient estimates.

TRPO is a prominent actor-critic algorithm that addresses the high variance issue in policy gradi-ents by constraining the step size of the policy update. Instead of performing a simple gradient ascent, TRPO optimises maxθEτ∼πθOLD[πθπθ(a(at|st)AπθOLD(s,a)] subject to the constraint

t t

|s )

OLD

3

Es∼ρθOLD[DKL(πθOLD(·|s)||πθ(·|s))] ≤ δ where θOLD denotes the policy prior to the update.

AπθOLD (s,a) is the advantage function and δ is a small positive constant.

TRPO’s strengths are its stability, ensured by the KL-divergence constraint preventing large policy updates, and its sample efficiency, enhanced by the critic’s role in improving policy gradient meth-ods. However, TRPO is complex as it requires computing second-order derivatives and solving a constrained optimisation problem.

Proximal Policy Optimisation (PPO) offers a simpler and more power solution as it can strike a balance between the stability and reliability of TRPO and the simplicity of implementation. While TRPO ensures stable policy updates through complex calculations, PPO achieves similar stability with a much simpler approach. It leverages the parallel environment approach from A2C and integrates the concept of a trust region from TRPO (while avoiding the complex constraints).

3 Method

Proximal Policy Optimisation

This implementation uses Proximal Policy Optimisation (PPO). A stable and efficient deep RL algorithm proposed by OpenAI in 2017 which combines the strengths of value-based and policy-based methods. It builds on the parallel environment approach from A2C, incorporating the concept of a trust region from TRPO. However, it avoids the complex constraints associated with TRPO, making it simpler and more efficient. This has the dual benefit of allowing for more efficient training and constraining policy updates to ensure stability by preventing significant deviations from the current policy. The fundamental idea being that after each policy update, the new policy should not deviate significantly from the old one. This is achieved through clipping, which restricts the extent of policy updates. Specifically, it uses a probability ratio to measure the difference between the current policy and the old policy. By clipping this ratio within a predefined range [1 + ϵ,1 − ϵ], PPO ensures that the policy updates remain conservative which mitigates the risk of large updates destabilising training. Smaller updates are empirically more likely to converge to an optimal solution, while large updates are more likely to degrade the policy, resulting in poor performance.

rt(θ) = πθπθ(a(at|st) This term is the probability ratio (or importance sampling ratio)

t t

|s )

OLD

At is the estimated advantage function at each time step. It represents how much better (or worse) a

ˆ

specific action was compared to the average action, given the current policy and state. The objective function is averaged over all the time steps in the batch of experiences. LCLIP = Et[min(rt(θ)At,clip(rt(θ),1 − ϵ,1 + ϵ)At]

ˆ ˆ

To achieve stability, PPO employs a ratio that compares the current policy with the former one. By clipping this ratio within a specific range, PPO removes the incentive for the current policy to diverge too far from the old one. This conservative approach ensures that policy updates are gradual and controlled, fostering a more stable and reliable training process.

Implementation

Actor and Critic Networks

The Actor network maps the input state to actions using a neural network with two hidden layers of 64 units each and a Tanh-activated output layer, producing actions with a learnable Gaussian distribution. The Critic network estimates the value of a given state using a similar architecture, but its output is a single scalar value representing the expected return from that state. Both networks use ReLU activations in their hidden layers. We also employed the Adam optimiser, as used in the original implementation.

4

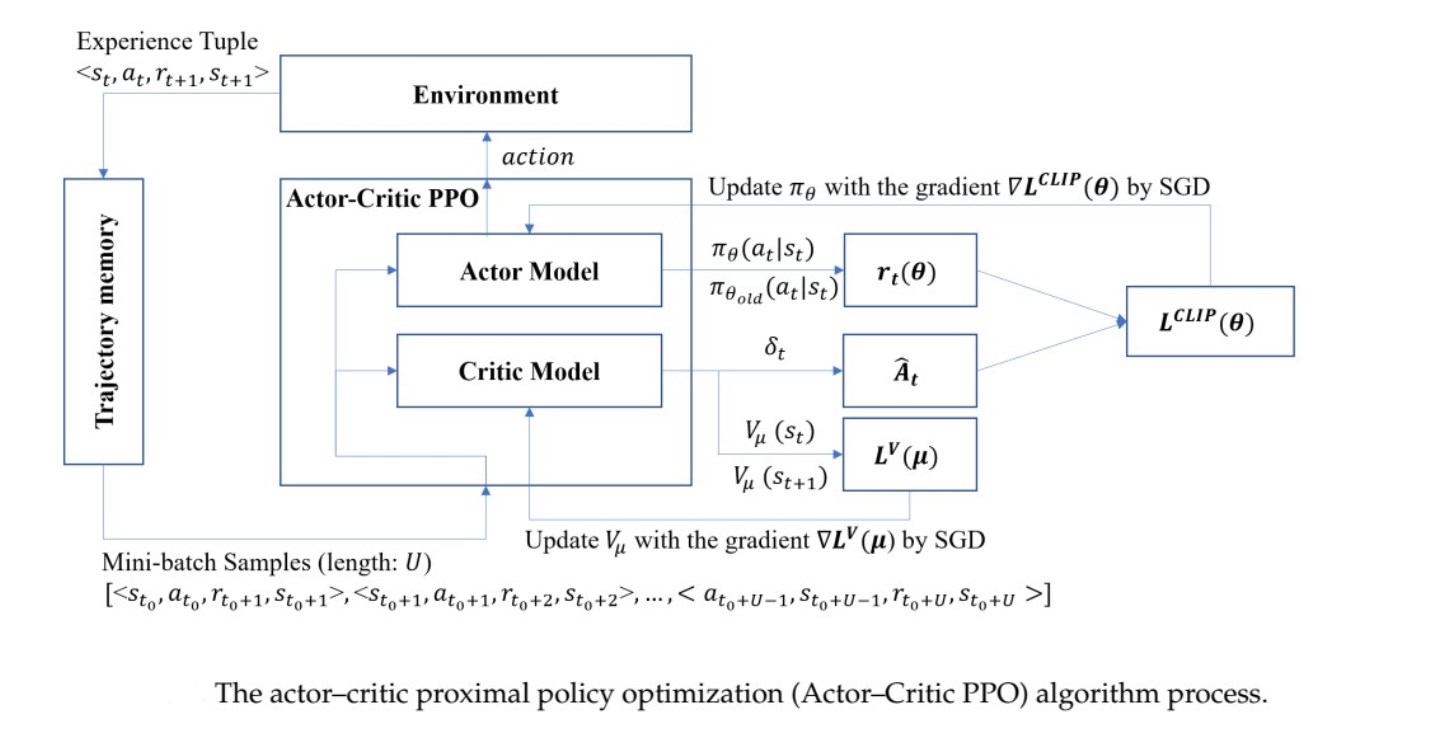


Figure 2: The actor-critic proximal policy optimisation (Actor-Critic PPO) algorithm process-

Generalised Advantage Estimation (GAE)

We used Generalised Advantage Estimation (GAE) to reduce variance in advantage estimates while maintaining some bias. GAE is useful in PPO because it strikes a good balance between bias and variance, leading to more stable and efficient training.

Minibatch Update

During training, we split the collected experience data into minibatches for updating the policy network. This approach allowed for more stable gradient updates compared to using the entire batch at once and improved the efficiency of the learning process.

Advantage Normalisation

To ensure that the scale of the advantages didn’t skew the gradient updates, we normalised the advantages to have a zero mean and a standard deviation of one. This step is important for stabilising learning, as it kept the updates consistent and prevented any one advantage from dominating the learning process.

Value Loss Clipping

For the value function, we applied clipping to prevent excessive divergence in value updates. This ensured that value estimation remained stable, which is a core part of PPO.

4 Results and Discussion

The best score we achieved was a maximum of 298.32 with the best average score over 100 runs 290.90 +/- 14.67. Although these results do not achieve an average score of over 300, which would demonstrate the problem has been solved, it comes very close.

Version

Action Random Uniform Action Random Normal

Our best PPO Implementation

Mean Score (100 eps) -100.84

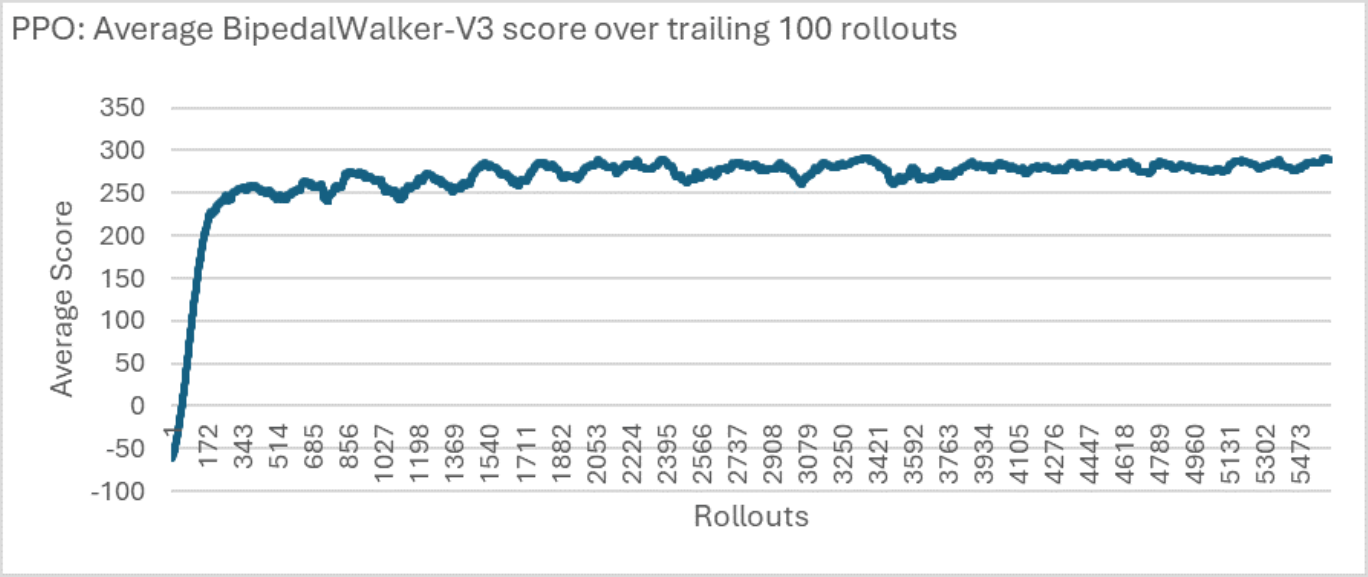
-112.17 290.90

Std Dev (100 eps) 14.13

11.70 14.67

The training pattern demonstrates the agent is quick to learn a strategy in approximately the first 500 iterations, which it then slowly improves on. However, since the agent does not achieve a score of over

5



300, this suggests that the policy is stuck in a local minima. Further tuning of the hyperparameters is required to find an optimal setup.

Figure 3: Results

5 Future Work

In future work, we aim to explore the potential of different PPO variants to enhance our model’s performance and efficiency. Our current implementation focused on PPO-Clip, known for its simplicity and ease of tuning. However, another variant, PPO with a Kullback-Leibler (KL) penalty, presents an alternative approach by incorporating a penalty based on the KL divergence between old and new policies. This method provides more control over policy changes by adding dynamic regularisation to the loss function. Although the KL penalty provides a direct measure of policy deviation, it introduces more complexity when tuning, and requires adjustment to avoid excessive regularisation.

6 Personal Experience

Implementing the actor action distribution was challenging, particularly when it came to deciding whether to learn or fix the standard deviation in policy learning. We had to weigh the flexibility that learning the standard deviation offers against the potential benefits of using a fixed standard deviation, which can be advantageous during the initial stages of learning. In the end we made the log\_std a trainable parameter in the model.

Configuring the gradient updates was also particularly challenging, especially when determining where to apply the detach() method in PyTorch to properly manage the gradient flows. Misplacing this function initially caused numerous errors, disrupting gradient flows and leading to ineffective learning or exploding gradients. To address these issues, we had to make adjustments and perform extensive debugging to ensure the stability and efficiency of the model.

6

References

[1] T.P. Lillicrap et al. Continuous control with deep reinforcement learning. 2016. DOI: [https:](https://doi.org/https://arxiv.org/abs/1509.02971) [//arxiv.org/abs/1509.02971.](https://doi.org/https://arxiv.org/abs/1509.02971)

[2] J. Schulman et al. Trust Region Policy Optimization. 2015. DOI: [https://arxiv.org/abs/](https://doi.org/https://arxiv.org/abs/1502.05477) [1502.05477.](https://doi.org/https://arxiv.org/abs/1502.05477)

[3] D. Silver et al. Deterministic policy gradient algorithms. 2014. DOI: [https://proceedings.](https://doi.org/https://proceedings.mlr.press/v32/silver14.html) [mlr.press/v32/silver14.html.](https://doi.org/https://proceedings.mlr.press/v32/silver14.html)

[4] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. 2012. DOI: [https://mujoco.org.](https://doi.org/https://mujoco.org)

[5] RJ Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 1992. DOI: [https://doi.org/10.1007/BF00992696.](https://doi.org/https://doi.org/10.1007/BF00992696)

7

Appendices

Appendix 1: State Space Bounds

Lower and upper boundaries for the state space

lower\_bound = [-3.1415927, -5.0, -5.0, -5.0, -3.1415927, -5.0, -3.1415927, -5.0, -0.0, -3.1415927, -5.0, -3.1415927, -5.0, -0.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0]

upper\_bound = [3.1415927, 5.0, 5.0, 5.0, 3.1415927, 5.0, 3.1415927, 5.0, 5.0, 3.1415927, 5.0, 3.1415927, 5.0, 5.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

8