Learning Locomotion Behaviour on Legged Robots Using Proximal Policy Optimization

Samuel Oladejo
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg

Prof. Benjamin Rosman (Supervisor)
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg

***Abstract*—The ability for legged robots to navigate complex environments and perform tasks similar to humans and animals is highly desirable, but programming these robots to walk effectively is a challenging task due to the high-dimensional, continuous nature of the task. To tackle this challenge, researchers have turned to Proximal Policy Optimization (PPO), a Reinforcement Learning algorithm that has shown exceptional performance in a variety of continuous optimization tasks. Despite its success, selecting the appropriate hyper-parameters and avoiding getting stuck in local minima remains a challenge when using PPO for learning locomotion skills in legged robots. In this paper, we investigate the impact of code-level optimization on the PPO algorithm and analyze the behavior of the agent using both conventional metrics and a novel approach based on the agent's decision-making process. We focus on the adjusting the PPO value and policy network's hyper-parameters, which is essential for achieving positive learning outcomes, and investigate the impact these changes have on the dynamics of agent training in the locomotion domain. Our findings suggest that code-level optimization can have a significant impact on the performance and stability of the PPO algorithm for legged robot navigation tasks.**

Index - neural networks, deep reinforcement learning, proximal policy optimization, OpenAI gym, MujoCo, legged robots

1. Introduction

Legged robots have the ability to navigate a variety of complex and unstructured environments, making them useful for a wide range of functions such as search and rescue, inspection, and transportation. However, programming these robots to walk effectively can be challenging due to the numerous variables involved in locomotion, such as terrain, balance, and gait which are high-dimensional tasks and are continuous in nature. As a result, there has been a significant amount of research focused on developing algorithms that can aid this ability in robots through trial and error. In this research paper, we focus on the locomotive component of these tasks is based on training robots to coordinate their gaits to move using PPO, a reinforcement learning algorithm.

Reinforcement Learning (RL) algorithms have become the default approach for solving these types of complex tasks due to their performance and successes in domains like Chess [Silver et al. (2017a)], Go [Silver et al. (2016)], Atari [Mnih et al. (2013)] to mention a few. Furthermore, RL algorithms have been applied to various tasks, including control, optimization, and decision-making [Sutton et al. (2018)]. This was made possible through the integration of deep neural networks and other non-linear function approximators.

RL is a type of machine learning that involves an agent taking actions in an environment to maximize a reward signal. In the context of legged robots, RL algorithms can be used to learn locomotion skills by receiving a reward for successfully moving in a desired direction or maintaining balance. One RL algorithm that is suitable for the task of learning locomotion behaviors in robots is proximal policy optimization (PPO) [Schulman et al. (2017)]. PPO is an actor-critic algorithm that incorporates the advantages of value-based and policy-based methods. It updates the policy based on the actions obtained by the current policy to decide the best course of action based on that estimate. It uses a value function to estimate the expected future reward of a given state. This enables the robot to alter its behavior in response to feedback from its environment. In addition, PPO has several advantages for learning locomotion skills in legged robots. It is relatively simple to implement and can be used with a variety of network architectures, it is relatively sample-efficient, meaning that it requires fewer interactions with the environment to learn a task and it has been shown to be more stable and less sensitive to hyperparameter tuning compared to other RL algorithms [Schulman et al. (2017)].

A number of research works have demonstrated the effectiveness of PPO, like in a paper by Heess et al. (2017), Distributed PPO was used to teach a quadruped robot to walk and trot on several complex terrains. The authors found that PPO was able to learn a variety of gait patterns and was able to adapt to changes in the robot's body shape and weight distribution. Similarly, in a paper by Choi and Kim (2019), PPO was used to teach a biped robot to walk.

The researchers discovered that PPO was able to adjust to modifications in the terrain and the robot's body shape as well as learn a stable walking pattern. Nevertheless, despite these successes, tuning of the hyper-parameters, which consist of the entropy coefficient, the discount factor, and the learning rate [Kirk et al. (2021)] is one of the main obstacles to using PPO for legged robot control. This has been primarily resolved by manually fine-tuning by experts certain agent or environment characteristics, which is time-consuming and thwarts the main exciting objective of RL. [Tan et al. (2018)]. Due to these issues, RL is less reliable than supervised learning because it lacks diverse exploration, and has brittle convergence properties [Henderson et al. (2017); Zhang et al. (2018)], and is wildly inaccurate across iterations [Henderson et al. (2017)].

With an emphasis on the hyper-parameters that enable the best locomotion learning behavior based on stability, and speed, additionally, we investigate the impact of code-level optimization on PPO in this paper. Furthermore, we examine the agent's behavior using both the typical metric of cumulative reward and more exact algorithmic attributes. Our experiments involved training of the simulated OpenAI Gym [Brockman et al. (2016)] quadruped robot called Ant [Schulman et al. (2015a)] and the Pybullet Minitaur [Coumans and Bai (2016)] severally to assess the effectiveness of the optimisation with various hyperparameter settings. Although the implemented optimization technique and adjusted hyperparameters could not match the performance of the initial baseline from OpenAI baselines predicated on the average cumulative rewards, they were able to yield favorable results that aided in demonstrating how code-level optimisations can significantly impact the PPO algorithm as alluded by Engstrom et al. (2020). We observed that the performance of PPO is most significantly impacted by the neural network architecture policy and value function approximators, learning rate and entropy coefficient. This shows how various optimization techniques can significantly change how algorithms work and analyze the principles underlying agents' operations. The experiments also show that, in addition to environmental brittleness, PPO is extremely sensitive to the choice of hyper-parameters and neural network architecture, supporting earlier findings that showed how fragile deep policy gradient methods are.

In subsequent sections, we give a brief overview of the background information relating to which concerns of this research, such as Reinforcement Learning, Policy Gradient Methods, and most importantly, Proximal Policy Optimization in Section 2. We further explore works relating to this paper in Section 3, give an analysis of the methods used for the experiments, and how we set up the experimental environments in Section 4 and Section 5 respectively. Lastly, we discuss the results from the experiments Section 6 and conclude of this paper in Section 7

The code for all the results in this paper is available at (https://github.com/saolxs/legged-robots)

2. Background

This section gives some important background information and emphasizes the advantages of teaching robots how to move using reinforcement learning, a class of machine learning algorithms. Proximal Policy Optimization, a policy gradient method, which is the state-of-the-art RL algorithm for continuous control problems is also reviewed, along with the justification for its use in this paper.

2.1. Locomotion Behavior Learning on Legged Robots

Learning locomotion behaviours on legged robots is an important area of research due to the numerous potential applications of legged robots in a variety of environments. Legged robots have the ability to navigate complex and unstructured environments, making them useful for tasks such as search and rescue, inspection, and transportation. In addition, legged robots are able to adapt to changes in their surroundings and can operate in situations where wheels or tracks may not be suitable, such as uneven or rocky terrain.

There are several approaches to teaching legged robot locomotion skills, including supervised learning, evolutionary algorithms, and reinforcement learning. Reinforcement learning (RL) algorithms, in particular, have become a popular choice for solving these types of complex tasks due to their performance and successes in a variety of domains. RL algorithms have been used for tasks like control, optimization, and decision-making. They entail an agent acting in the environment to maximize a reward signal. Deep neural networks and other non-linear function approximators have improved RL methods' capacity to handle challenging problems [Sutton et al. (2018)].

The performance of the policy gradient reinforcement learning algorithm, PPO [Schulman et al. (2017)], on quadrupedal robots, or robots with four legs, is of particular interest to us in this paper.These robots have a wide range of gaits available to them, including walking, running, and trotting, which makes them suitable for a variety of environments and tasks. Furthermore, quadrupedal robots are able to adapt to changes in their surroundings, such as changes in terrain or body shape, by modifying their gait Carpentier and Wieber (2021). Typically, the stability, versatility, and adaptability of quadrupedal robots make them an ideal choice for learning locomotion behaviour [Tan et al. (2018)].

2.2. Reinforcement Learning

A branch of machine learning called reinforcement learning (RL) seeks to maximize the expected cumulative reward from an agent's interactions with its environment in order to

optimize the behavior policy of the agent. It is a powerful tool that has shown to perform well in complex tasks, such as Atari games Mnih et al. (2013), chess Silver et al. (2017a) and most notably AlphaGo Silver et al. (2017b). In other to accomplish a learning task (discrete and continuous), RL poses the task as an optimization problem, Markov Decision Process (MDPs), which it needs to solve i.e an agent in an environment need to select the best actions based on its current state, as shown in Figure **??**.

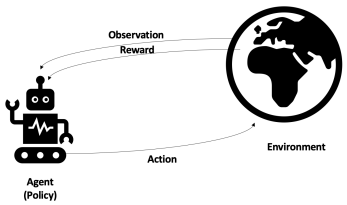

Figure 1. A depiction of how an agent learns in an environment using Reinforcement Learning

2.3. Markov Decision Process (MDPs)

Markov Decision Process (MDPs) is a mathematical framework for modelling a reinforcement learning problem Bellman (1957) in an environment. It consists of:

- $\mathcal{S}$ a set of states.
- $\mathcal{A}$ a set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the environment transition function, where $\mathcal{T}(s' \mid s, a)$ specifies the probability of ending up in a certain state s' after starting in another state s and performing a specific action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, where $R(s_l, a_t) = R_{t+1}$ specifies the reward obtained from performing action a_t in a state s_t.
- $\gamma \in [0, 1]$ is the environment discount factor, specifying how long term and short term rewards should be weighted.

Over the years, due to the performance and feasibility on RL on various tasks, a number of algorithms, useful in different use cases have been developed. A notable one which we will be exploring in this research is the policy gradient methods.

2.4. Policy Gradient Methods

On-policy reinforcement learning techniques known as "policy gradient methods" [Sutton et al. (1999)] are geared toward modeling and directly optimizing the policy. It is dependent on parameterized policies that underwent gradient ascent optimization in terms of the anticipated return (long-term cumulative reward). These strategies' main goal is to make it more likely that decisions will yield higher returns while making it less likely that decisions will yield lower returns until the agent discovers the best course of action.Numerous issues that have plagued traditional reinforcement learning techniques, such as the lack of confidence in a value function, the intractability issue caused by unclear state information, and the complexity caused by continuous states and actions, are not present in these approaches. The objective function which is used for optimisation is represented as:

$$L^{PG}(\theta) = \underset{T \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) A^{\pi_\theta} (s_t, a_t) \right] \tag{1}$$

where, $\pi_\theta (a_t \mid s_t)$ refers to a policy that is controlled by a neural network with parameters that accepts observed environmental states as input and generates action suggestions as outputs, $A^{\pi_\theta} (s_t, a_t)$ is the advantage function for the current policy, which is an estimation of how well the action is as compared to the average action for a specific state primarily based on the infinite-horizon discounted return. $J(\pi_\theta)$ denotes the expected finite horizon undiscounted return of the policy.

With that being said, the advantage estimates can however be noisy as they are quite sensitive to the step size that is selected; if it is too little, progress will be extremely slow, and if it is too large, the response will be excessively noisy, making it very challenging. The solution to this problem as been the basis of a number of papers but in this project, we will be focusing on the implementation of the Proximal Policy Optimization (PPO) Schulman et al. (2017) algorithm.

2.5. Trust Region Policy Gradient (TRPO)

By maximizing the size of the policy update, TRPO Lee et al. (2020) aims to increase training stability. It updates its policy using a trust region constraint, maintains a small enough distance between old and new policies which are measured using a Keiber Leiber (KL) Divergence to ensure they do not diverge when an hard constraint is met.

$$\mathbb{E}_{s \sim \rho^{x_{\text{odd}}}} \left[D_{\text{KL}} \left(\pi_{\theta_{\text{akl}}} (\cdot \mid s) \| \pi_\theta (\cdot \mid s) \right] \leq \delta \right. \tag{2}$$

With the objective Function,

$$J^{\text{TRPO}}(\theta) = \mathbb{E} \left[r(\theta) \hat{A}_{\theta_{\text{cld}}} (s, a) \right] \tag{3}$$

It is suitable for continuous control tasks, but it struggles with algorithms that use common parameters for a policy and a value function, is difficult to implement, is very sensitive to hyperparameters, and can be computationally expensive. For these reasons, a simpler algorithm known as Proximal Policy Optimization was developed by Schulman et al. (2017).

2.6. Proximal Policy Gradients (PPO)

PPO is primarily concerned with direct parameterized policy optimization in terms of expected return until the optimal policy is discovered. It is a simple first-order actor-critic policy gradient method that attempts to improve a policy using previously collected data. It is derived from another policy gradient method TRPO Schulman et al. (2017) but instead of using KL divergence, it clips the surrogate objective and still achieves comparable performance to TRPO.

PPO is designed to perform multiple mini-gradient steps using the same data from the environment to make progress in each iteration, and it is able to do so by ensuring that the objective function takes the smallest difference between the clipped and original value, ensuring that its policy is always stable. This lowers computing costs and provides users with an unlimited range of actions and/or states from which to estimate values. When the new policy is far away from the old policy, or when the probability ratio between the two policies is outside the acceptable range, $(1 - epsilon)$ and $(1 + epsilon)$, the advantage function is clipped to the value it will have between those two ranges. This is done to discourage any large policy changes when the ratio of both policies falls outside the expected zone.The probability ratio between old and new policies is defined as follows:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{old}(a_t \mid s_t)} \tag{4}$$

and the objective function for PPO is defined as:

$$L^{CLIP}(\theta) = \hat{\mathrm{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{5}$$

It can be simplified as:

$$L^{CLIP}(\theta) = \hat{\mathrm{E}}_t[\min\left(r_t(\theta)A^{\pi\theta_k}(s,a), \quad g\left(\epsilon, A^{\pi\partial_{\theta_k}}(s,a)\right)\right] \tag{6}$$

where,

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases} \tag{7}$$

given θ as the parameter, ϵ, as an hyper-parameter that is often times $\mathbf{0.2}$ used to clip the probability ratios, and $A^{\pi\theta_k}(s,a)$ as the advantage function, which can be further expressed as the difference between the discounted sum of rewards and baseline estimate. The baseline estimate is a prediction of what the discounted sum of rewards (return) will be from the current state and is frequently updated during training using the experience the agent learns from the environment. The discounted sum of rewards is simply the weighted sum of rewards the agent received during each time-step of the current episodes. This is used to determine how the action will perform compared to expectations for that state, computed over batches of trajectories.

Furthermore, PPO can be applied to a network architecture with shared or separate parameters for both the actor policy and the critic value functions, as well as the clipped reward, where the objective function is supplemented with an error term on value estimation and an entropy term to encourage adequate exploration.

$$L_t^{PPO}(\theta) = \hat{E}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\right] \tag{8}$$

where both $\mathbf{c_1}$ and $\mathbf{c_2}$ are two hyper-parameter constants, S denotes an entropy bonus which ensures the agent explores the environment during training, and L_t^{VF} is the squared error loss $(V_\theta(s) - V_{\text{target}})^2$ which updates the baseline network to determine how good it is to be in a particular state. The detailed pseudocode for the Clipped Surrogate Objective PPO, adapted from ... and used in this research is outlined in alg:ppo

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\mathbf{r_t} A^{\pi\theta_k}(s_t, a_t), g\left(\epsilon, A^{\pi\theta_k}(s_t, a_t)\right)\right) \tag{9}$$

7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2 \tag{10}$$

typically via stochastic gradient ascent with Adam. =0

3. Related Work

Proximal policy optimization has been used extensively in recent years to study how to teach legged robots how to walk (PPO). Li et al. (2018) study is one noteworthy example of using PPO to train a quadruped robot to run and walk on different types of terrain. The researchers discovered that even on difficult terrain, like stairs and slopes, PPO was able to achieve stable and effective locomotion skills in the robot. The performance of PPO for teaching locomotion skills to a humanoid robot was improved in a different study by Choi and Kim (2019). By breaking down complex locomotion skills into smaller, more manageable tasks, the authors' proposed hierarchical PPO framework enables the robot to learn complex locomotion skills. The performance and learning rate of the robot were shown to be significantly enhanced by this hierarchical approach.

However, the requirement for meticulous hyperparameter tuning poses a potential barrier to using PPO for teaching locomotion skills to legged robots Gupta et al. (2018). The values of the various hyperparameters, including the learning rate and the entropy coefficient, can have a significant impact on PPO's performance. To get the best results in their study, Li et al. (2018) extensively tuned the hyperparameters. The application of automated hyperparameter tuning algorithms is one potential remedy for this restriction. Another limitation is the computational cost of PPO, which can be high for large and complex environments Schulman et al. (2015b).

To address these limitations, several solutions have been proposed. As an illustration, Yang et al. (2019) suggest a technique for lowering the computational expense of PPO by combining model-based and model-free methods. To increase the convergence and stability of PPO, Schulman et al. (2015a) suggested a different strategy that makes use of the trust region optimization method.

Furthermore, We draw inspiration from Engstrom et al. (2020), to focus on the code-level optimization by breaking the core algorithm into smaller pieces to understand the effects of the various components that make it work by making little modifications based on the observation made by Henderson et al. (2017) who brings to light the brittleness, experimental practice, and reproducibility issues in deep RL algorithms. Huang et al. (2022) further outlines in the code implementation details of the PPO algorithm using the OpenAI baselines as a benchmark to expatiates on the necessity of reproducibility.
Overall, proximal policy optimization has demonstrated promising results for learning locomotion skills on legged robots, but careful attention to hyperparameter tuning is required to achieve the best performance.

4. Methodology

Many algorithms benefit greatly from having their hyperparameters fine-tuned. However, in related literature, the optimal hyperparameter configuration is often not consistent, and the range of values considered is often not reported. Furthermore, poor hyperparameter selection can make it difficult to compare algorithms fairly. In this paper, we investigate the effects of hyperparameter and neural network selection on algorithm performance when they are not properly tuned. This was inspired by the claims made by Engstrom et al. (2020), discussed in Section 3. We based our code implementation of the PPO algorithm on the OpenAI Baselines2. For each evaluation, we conduct five experiment trials with different preset random seeds to ensure fairness. When possible, we use the default configuration and only change the relevant hyperparameters. In addition to the hyperparameter and neural network optimization, we also examine the ability of legged robots to learn agile locomotion skills (trotting and pacing) and their stability during locomotion using different gaits.

To optimize the agent's control policy, we implemented separate two-layer multilayer perceptron function approximators Andrychowicz et al. (2020) with 64 neurons for the policy (actor) and value (critic) functions, using the Hyperbolic Tangent as the activation function. The hidden layer sizes and activations are denoted as (N, M, activation). The network processes its input using the moving average normalized observation and clips it between [-10, 10] for generalization and optimal performance during the training of the neural networks. To avoid sampling invalid actions, the actions are represented as a normal distribution and clipped within a valid range, while unclipped actions are stored as part of the episodic data Haarnoja et al. (2018). Unlike the implementation by Schulman et al. (2017),

$$L^V = \left(V_{\theta_t} - V_{\text{targ}}\right)^2 \tag{11}$$

the value network was clipped around the previous value estimates with a fixed learning rate

$$L^V = \max\left[\left(V_{\theta_t} - V_{\text{targ}}\right)^2, \left(\text{clip}\left(V_{\theta_t}, V_{\theta_{t-1}} - \varepsilon, V_{\theta_{t-1}} + \varepsilon\right) - V_{\text{targ}}\right)^2\right] \tag{12}$$

similar to other standard PPO implementations. We track an approximate average KL divergence between the old and new policy for each update step to the network weights and force an preemptive stop to the policy weights if the KL divergence exceeds a set limit.

4.1. Evaluation Metrics

To evaluate the method proposed in this research and compare it against the baseline, we will consider the use of the following metrics:

- The agents' ability to move its locomotion gaits to navigate the environment and how stable it becomes overtime.
- Average return for each episode, to judge the learning rate and assess the best performing method where an higher reward means the better performance.
- The agents algorithmic properties such as variance, entropy loss, policy loss, and value loss.

4.2. Reward Structure

The reward for all tasks are simple and consistent across all episodes, iterations and environment. The reward consists of a main component proportional to the velocity along the x-axis, which encourages the agent to make faster forward running speed together with a healthy reward of value 5, and a healthy z range of (0.5,2) . We use reward scalingDuan et al. (2016); Gu et al. (2016), where rewards are clipped to a range of [-10, 10] similar to Carpentier and Wieber (2021) improve results, and rewards are divided by the standard deviation of a rolling discounted sum of rewards. The detailed code level optimization is seen below:

Algorithm 2 Reward Scaling Optimization.

procedure INITIALIZE-SCALING()
$R_0 \leftarrow 0$
$R_S = RUNNINGSTATISTICS()$
procedure SCALE-OBSERVATION(r_t)
$R_t \leftarrow \gamma \mathbf{R_t 1} + \mathbf{r_t}$.
$ADD(R_S, R_t)$
return r_t/STANDARD-DEVIATION(R_S)

Where, RUNNINGSTATISTICS() initializes a new running statistic class that tracks mean, and standard deviation. SCALE-OBSERVATION(r_t) inputs a reward r_t, updates the reward and returns a scaled reward.

5. Experiments Setup

The *environment* that is used frequently has a significant impact on how well a new algorithm proposal performs

in comparison to benchmarks. Continuous control tasks frequently involve environments with stochastic randomness, condensed trajectories, or other dynamic characteristics. We show that these variations can affect the performance of the algorithms across environments, and that the best algorithm across all environments is not always obvious. We used the OpenAI gym Brockman et al. (2016) with the MujoCo engine Todorov et al. (2012) and the Pybullet physics engine Coumans and Bai (2016), which are the reinforcement learning industry standard simulation environments, for the experiments carried out in this research. In particular, the Ant-v3 Schulman et al. (2015a) environments from the OpenAI gym MujoCo library and the Minitaur-v0 from the Pybullet Physics engine, were used without alteraions to the basic structure of the environments. Further details regarding the environments are outlined below:

Ant-v3 : This is an environment with a continuous action space which consists of a 3-dimensional quadruped robot with 1 torso and four legs (each leg having two links). It is a quadruped robot which has 27 observation spaces with positional values of the Ants various body parts and 8 continuous action spaces ranging between (-1, 1). The goal in the environment is to coordinate the movements of the robots' leg to move forward by applying torques on the hinges. The agents' observation space can also be grouped into two sets:

- A group of prio-ceptive features with two additional sensors attached to the feet and legs and a torso equipped with joint angles, angular velocities, a velocimeter, an accelerometer, and a gyroscope to measure velocity and acceleration.
- A set of exteroceptive features containing relevant task information including the position with respect to the distance from the desired goal.

MinitaurBulletEnv-v0 : This environment is a 3-dimensional platform with a continuous action space consisting of a quadruped robot with 4 legs and 8 actuators for movement. It has 28 observation spaces and 8 continuous action spaces ranging from (-1, 1).

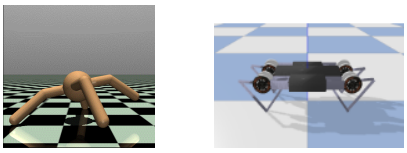

Figure 2. Pictorial representation of the environments used for experiments: (a) Ant-v3 . (b) MinitaurBulletEnv-v0

5.1. Hyper-parameters

The baseline hyperparameters used in our implementation, as described by Schulman et al. (2017), are shown in table 1 and were tested on 2 different environments (see Section 5). We altered each hyperparameter individually, while keeping the others at their default values, to evaluate the performance of our method. We will provide more information about the hyperparameter optimization in the Results, (Section 6)

TABLE 1. HYPER-PARAMETERS USED IN PPO LEARNING

Hyperparameter	Value
Learning rate (α)	3e-4
Discount factor (γ)	0.99
GAE Lambda (λ)	0.95
Normalize Advantages	True
Anneal Learning Rate	True
Policy Clip	0.2
Critic Loss Weight	0.1
Entropy Loss Weight	0.0
Batch Size	2048
Mini-batch size	64
Update per Epoch	10 (steps)
Optimizer	Adam
Seed	42

5.2. Network Architecture

In order to study the impact of network architecture on algorithm performance, we implemented 2-layer and 3-layer multilayer perceptron (MLP) network architectures with various activation functions (see Section 5.2) for both the value and policy networks, using tanh and ReLU activations. We initially implemented a two-layer network with 64 and 64 neurons (the baseline) and a default learning rate of $\mathbf{3^e - 4}$ on all environments to test our implementation of the baseline algorithm. Then, we adjusted the learning rate to decay from $\mathbf{1^e - 4}$ and $\mathbf{2^e - 4}$, respectively, and used a tanh activation function on the Ant-v3 and Minitaur-v0 environments over 2 million and 1 million steps, respectively. We also increased the number of neurons in the policy and value networks to 128 and 128, respectively, over the same time steps. This was done to determine which network architecture and learning rate would result in the highest episodic returns, in order to ensure that the agent learns at an appropriate pace. Due to limitations in computational resources, we added a third layer to the network only to investigate its effect on the agent's learning stability during training. We found that the simpler networks on both environments had higher returns, possibly due to the small number of environmental observations, but the more complex network architecture performed better overall (Section **??**).

TABLE 2. 2-LAYER NETWORK SIZE

Stream	Layer	Neurons/Size
Critic	Input layer	State HxW
	Hidden layer 1	64, 128
	Hidden layer 2	64, 128
	Output Layer	**1**
	Activation Function	Tanh, ReLU
Actor	Input layer	State HxW
	Hidden layer 1	64, 128
	Hidden layer 2	64, 128
	Output layer	$\mid A \mid$
	Activation Function	Tanh, ReLU

TABLE 3. 3-LAYER NETWORK SIZE

Stream	Layer	Neurons/Size
Critic	Input layer	State HxW
	Hidden layer 1	128
	Hidden layer 2	64
	Hidden layer 3	32
	Output Layer	**1**
	Activation Function	Tanh, ReLU
Actor	Input layer	State HxW
	Hidden layer 1	128
	Hidden layer 2	64
	Hidden layer 3	32
	Output layer	$\mid A \mid$
	Activation Function	Tanh, ReLU

6. Results

The results in Figure 3 and Figure 6.1 are representative of the best 3 runs and 6 runs based on the average returns for the Ant-v3 and Minituar-v0 environments, respectively. Due to the unpredictable nature of RL algorithms, tracking only the maximum returns is commonly insufficient for fair comparison. Even reporting average returns can be misleading because it is unknown how different seeds and trials will perform. Therefore, we measured more detailed algorithmic properties such as variance, entropy loss, policy loss, and value loss and compared the agents with the highest average returns using these metrics in Figure 6.1 and Figure 6.1. The agents were trained using the hyperparameters described in Section 5.1 and the network architectures in Section 5.2. The code for all the results shown in this work, along with the videos of the agents' performance for each run, is available at https://github.com/saolxs/legged-robots.

6.1. Discussion

The results in Figure 6.1 show that the Ant robot with a 2-layer neural network of 64 neurons each, using a ReLU activation function and a learning rate of $3e^{-4}$, was able to effectively learn locomotion skills and stabilize itself when it missed steps. It was also able to progress from a basic walk to trotting without falling, which meets the goal of teaching the quadruped robot to navigate an environment by learning locomotion skills. In contrast, the baseline model performed poorly, with the agent struggling to determine which locomotion gait to use for navigation. This demonstrates that the performance of algorithms and agents can be affected by the neural network architecture, as noted in Henderson et al. (2017). Therefore, it is important to evaluate various configurations to find ones that produce results that are equal to or better than the baseline algorithm's initial performance. Additionally, based on the policy loss, the agent with 128 neurons in each of its 2 layers and a Tanh activation function did not change its actions as much as the other two agents, leading to it being unable to decide on the optimal locomotion gaits and falling after moving a certain distance. The baseline model with 64 neurons in each of its 2 layers and a Tanh activation function was also unable to effectively predict the value of each state it was in, which may have contributed to its inability to move.

Environment	Neurons	Activation Function	Returns per Episode	approx_kl	Entropy Loss	Variance
Ant-v3 - 1669089503	[64, 64]	Tanh	74.4226	0.005882927216589451	-18.23774	0.6195406
Ant-v3 - 1669077181	[128, 128]	Tanh	66.8233	0.001088822260499003	-17.38208	0.5219511
Ant-v3 - 1669076559	[64, 64]	ReLU	1796.0352	0.000857936218380928	-14.87361	0.1839557

Figure 3. A table of the results from the OpenAI Gym Ant-v0 Environment

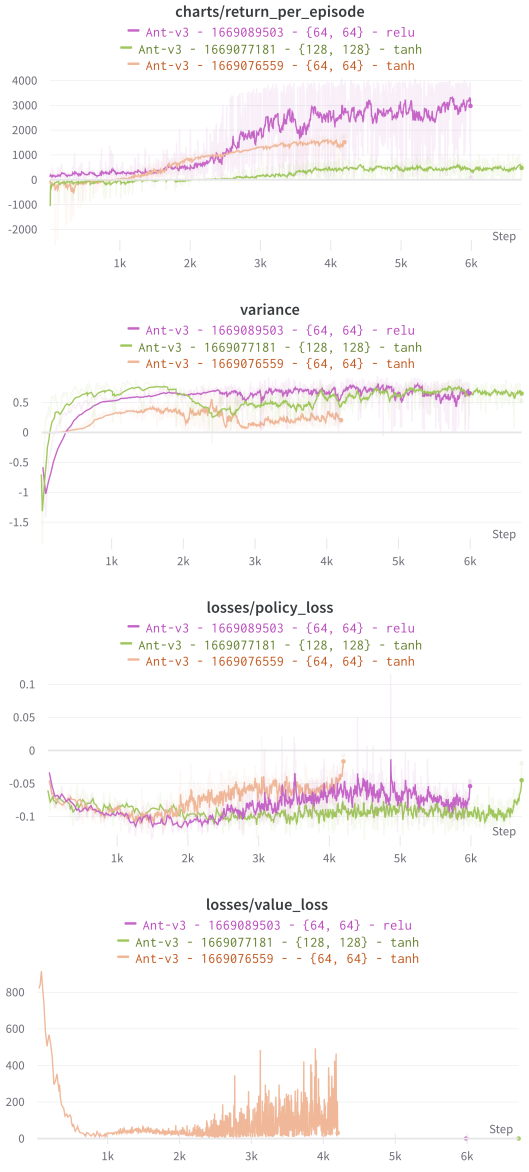

Figure 4. Results on the OpenAI Gym Ant-v0 Environment

The results in Figure 6.1 show that the Minitaur robot with a 2-layer [64, 64] network architecture using a Tanh activation function and a learning rate of $\mathbf{2e^{-4}}$ had the highest

performance. In contrast, the baseline model with a learning rate of **3e⁻4** had the lowest performance. Although the 3-layer architecture also performed well overall, on average it was not as effective as the 2-layer network with a learning rate of **3e⁻4**. However, it should be noted that the 3-layer minitaur agent with a learning rate of **3e⁻4** and a ReLU activation function did not change its actions, whereas the agent that achieved the highest returns was able to do so, leading to effective movement. When evaluating the agent's ability to generalize to different environments, it was observed that PPO struggled in more complex domains. For example, the average returns in the Ant environment were significantly lower compared to the Minitaur environment, even when using the same network architecture. In conclusion, the results presented in Figures 5 and 6 demonstrate the performance of various agents in the Pybullet Minitaur-v0 environment.

Environment	Learning rate	Layers	Neurons	Activation Function	Returns per Episode	approx_kl	Entropy Loss	Variance
Minitaur-v0	0.0003	2	[64, 64]	Tanh	7.12	0.00009469	2.2940254	0.3469809
Minitaur-v0	0.0002	2	[64, 64]	Tanh	25.88	0.00001829	7.1967411	0.6925231
Minitaur-v0	0.0001	2	[64, 64]	Tanh	16.01	0.00000109	9.8472080	0.2716389
Minitaur-v0	0.0003	3	[128, 128]	ReLU	19.94	0.00300928	4.6315741	0.1558069
Minitaur-v0	0.0002	3	[128, 128]	ReLU	12.08	0.00006760	4.0946068	0.9301133
Minitaur-v0	0.0001	3	[128, 128]	ReLU	17.24	0.00001006	9.3398218	0.7838997

Figure 5. A table of the results from Pybullet Minitaur-v0 Environment

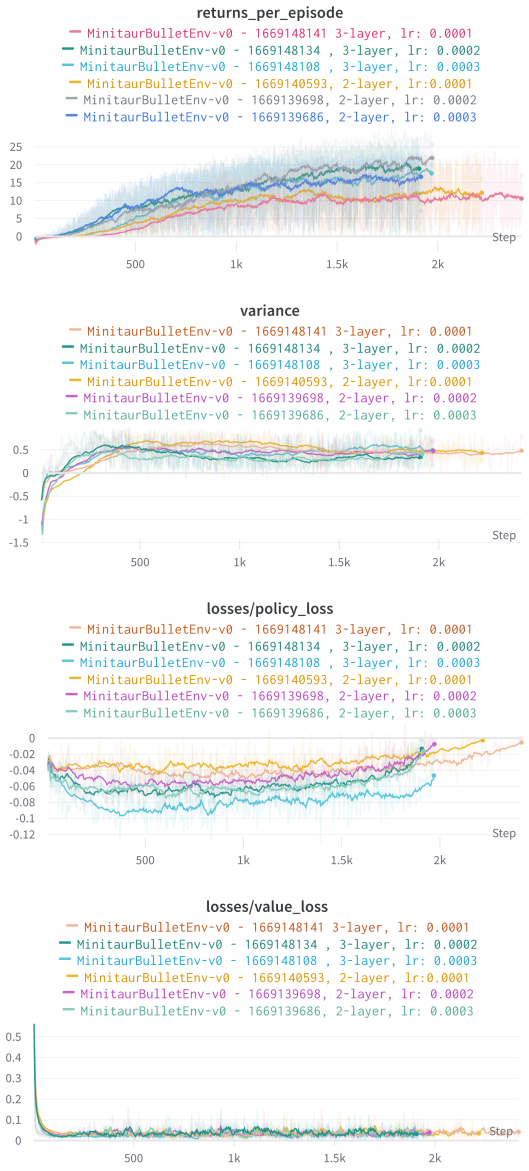

Figure 6. Results on the Pybullet Minitaur-v0 Environment

7. Conclusion

In this study, we examined the use of various optimization techniques and their effect on the performance of an agent using the PPO algorithm for legged robot locomotion learning. While none of the implemented parameters were able to match the speed of the initial baseline, they still produced valuable results that helped us understand the impact of innate strategies on the performance of the PPO algorithm. These findings can be useful for understanding the behavior of agents from scratch and can significantly alter the function of the algorithm in ways that are not intended by the policy gradient framework. One limitation we identified was that the implemented methods struggled to generalize to more

complex domains that were very different from the training tasks.

Overall, our findings suggest that these optimizations both confirm previous research indicating the fragility of deep policy gradient methods and demonstrate that, in addition to environmental fragility, the neural network architecture and selection of hyperparameters have a significant impact on PPO. In future work, we plan to investigate the use of PPO for navigating to random positions in an environment with obstacles while leveraging the learned locomotion skills to adapt to the complex environment.

8. Acknowledgement

I would like to appreciate the Oladejo family, Prof. Benjamin Rosman and Phemelo Mahamuza for their support on this journey.

References

M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? A large-scale empirical study," *CoRR*, vol. abs/2006.05990, 2020. [Online]. Available: https://arxiv.org/abs/2006.05990

R. Bellman, "A markovian decision process," *Indiana University Mathematics Journal*, vol. 6, no. 4, p. 679–684, 1957.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. [Online]. Available: https://arxiv.org/abs/1606.01540

J. Carpentier and P.-B. Wieber, "Recent progress in legged robots locomotion control," *Current Robotics Reports*, vol. 2, 09 2021.

S. Choi and J. Kim, "Trajectory-based probabilistic policy gradient for learning locomotion behaviors," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 1–7.

E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," *CoRR*, vol. abs/1604.06778, 2016. [Online]. Available: http://arxiv.org/abs/1604.06778

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on PPO and TRPO," *CoRR*, vol. abs/2005.12729, 2020. [Online]. Available: https://arxiv.org/abs/2005.12729

S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," *CoRR*, vol. abs/1611.02247, 2016. [Online]. Available: http://arxiv.org/abs/1611.02247

A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," *CoRR*, vol. abs/1802.07245, 2018. [Online]. Available: http://arxiv.org/abs/1802.07245

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017. [Online]. Available: http://arxiv.org/abs/1707.02286

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *CoRR*, vol. abs/1709.06560, 2017. [Online]. Available: http://arxiv.org/abs/1709.06560

S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," in *ICLR Blog Track*, 2022, https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of generalisation in deep reinforcement learning," *CoRR*, vol. abs/2111.09794, 2021. [Online]. Available: https://arxiv.org/abs/2111.09794

J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

Z. Li, X. Jiang, L. Shang, and H. Li, "Paraphrase generation with deep reinforcement learning," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3865–3878. [Online]. Available: https://aclanthology.org/D18-1421

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015. [Online]. Available: https://arxiv.org/abs/1506.02438

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and et al., "Mastering the game of go with deep neural networks and tree search,"

Nature, vol. 529, no. 7587, p. 484–489, 2016.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

R. Sutton, F. Bach, and A. Barto, *Chapter 13*. MIT Press Ltd, 2018, p. 321–339.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999.

J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," 2018. [Online]. Available: https://arxiv.org/abs/1804.10332

E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," *CoRR*, vol. abs/1907.03613, 2019. [Online]. Available: http://arxiv.org/abs/1907.03613

A. Zhang, Y. Wu, and J. Pineau, "Natural environment benchmarks for reinforcement learning," *CoRR*, vol. abs/1811.06032, 2018. [Online]. Available: http://arxiv.org/abs/1811.06032