A Review of the Applications of Proximal Policy Optimization (PPO) and Actor-Critic methods in Reinforcement Learning

Sambit Prabhu(20117108)
Sanskriti Sharan (20115123)

Abstract

This report explores the applications of two reinforcement learning techniques, Proximal Policy Optimization (PPO) and Actor-Critic Method, in the domains of humanoid walking and reinforcement learning-based football agent. The report provides an overview of these techniques, their background, and explains how they are utilized for training agents in these specific domains. It also discusses the advantages and limitations of these approaches and concludes with insights into their effectiveness. Reinforcement learning has wide-ranging applications in control theory and robotics, and legged robots have exhibited remarkable resilience in walking across diverse environments on various robot platforms through the utilization of deep reinforcement learning (RL) based controllers. We train the PPO algorithm in the MuJoCo simulation environment for humanoid robots. On the other hand, RL gained traction in the early 2000s with its successes in developing AI bots for video games, so we also implemented a bare-bones version of the Proximal Policy Optimization algorithm for the purpose of training an AI bot to play the game of football in Google Football.

Introduction

Reinforcement Learning has applications ranging from robotics and control to AI bots for games. Proximal Policy Optimization is an improved policy gradient method developed by OpenAI in 2017.
In recent years, learning-based approaches, particularly model-free deep reinforcement learning (RL) for control, have opened up new possibilities in controlling legged robots. These RL policies can be trained to accomplish tasks like balance, locomotion, and complex manipulation skills. While there have been remarkable demonstrations of bipedal walking in both simulation and real robots, the widespread application of such controllers in practical settings is still on the horizon.

An essential aspect of practical robots is their ability to adhere to user-specified commands, particularly regarding desired walking modes. In practical deployment, it is valuable to have a controller that can execute walking on curved paths, flat terrains, and stairs, while supporting forward and backward walking and the ability to stand still—all in response to user commands. Additionally, the bipedal robot should seamlessly transition between these modes without the need to switch to a different controller. Traditionally, model-based control frameworks achieve this through a footstep plan that includes target foot positions and orientations, coupled with a finite-state machine (FSM). Footstep plans effectively reduce uncertainty in controller behavior by providing advanced knowledge of when and where to place the feet. This approach enables omnidirectional walking on flat surfaces and 3D terrains, focusing on achieving user-specified walking modes.

Background

Reinforcement Learning

Reinforcement Learning (RL) involves the task of learning to determine an action *a* based on an input state s in order to maximize a reward *r*. The world is typically represented as a discrete-time Markov Decision Process (MDP) comprising a continuous state space denoted as *S ∈ R^n*, an action space denoted as *A ∈ R^m*, a state-transition function denoted as *p(s, a, s')*, and a reward function denoted as *r(s, t)*.

The state-transition function, *p: S × A × S → [0, 1]*, defines the dynamics of the world by providing the probability density of transitioning to the next state, *s'*, when taking action *a* in the current state *s*.
It is important to note that the state-transition function *p* is assumed to be unknown beforehand. Additionally, the reward function, *r: S × A → R*, generates a time-dependent scalar signal at each state transition.

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a popular reinforcement learning algorithm that aims to optimize policies in a stable and efficient manner. It is particularly well-suited for problems with continuous action spaces.

PPO builds upon the policy gradient approach and addresses some of its limitations, such as sensitivity to step sizes and instability during optimization. It strikes a balance between sample efficiency and policy update stability.

The key idea behind PPO is to perform multiple epochs of policy updates using data collected from interacting with the environment. During each update, a surrogate objective function is optimized to ensure that the updated policy does not deviate too far from the previous policy. This constraint is enforced by using a clipped surrogate objective, which limits the update to a specified range around the old policy.
The key contribution of PPO is ensuring that a new update of the policy does not change it too much from the previous policy. This leads to less variance in training at the cost of some bias, but ensures smoother training and also makes sure the agent does not go down an unrecoverable path of taking senseless actions.
By carefully balancing exploration and exploitation, PPO achieves reliable and consistent policy improvement. It has been successfully applied to a wide range of reinforcement learning tasks, including both simulated and real-world environments.

Actor-Critic Algorithm

The Actor-Critic algorithm is a popular reinforcement learning technique that combines elements of both value-based and policy-based methods. It aims to learn both a policy (actor) and a value function (critic) simultaneously to improve decision-making in an environment.

In the Actor-Critic algorithm, the actor component represents a policy that selects actions based on the observed states. Its objective is to maximize the expected cumulative reward by directly estimating the policy gradient. The critic component, on the other hand, approximates the value function, which estimates the expected cumulative reward from a given state. The critic provides feedback to the actor by evaluating the quality of the chosen actions and guiding the policy improvement process.

The training process involves iteratively updating the actor and critic using the observed state-action-reward transitions. The critic learns by minimizing the temporal difference error between the estimated value and the actual discounted cumulative reward. The actor learns by maximizing the expected return using gradient ascent on the policy objective, which is often based on the advantage function derived from the critic's value estimates.

Training and Simulation

1. Humanoid bot learning to walk

Training of PPO with Actor-Critic neural networks was done in the Mujoco humanoid environment. The training was done for 20000 iterations, roughly corresponding to 100 million time steps of 0.025 seconds each.

The actor is a neural network which evaluates softmax outputs for all possible actions and selects an action according to a Gaussian distribution.

The critic is a value function approximator network that estimates the state value V of a state.

Input normalization is done before feeding the inputs to the NNs.

The hierarchical control framework adopted consists of a higher-level RL policy that makes joint position predictions at a slow update rate of 40 Hz, and a lower-level PD controller working at 1000 Hz responsible for converting the desired joint positions to desired joint torques.

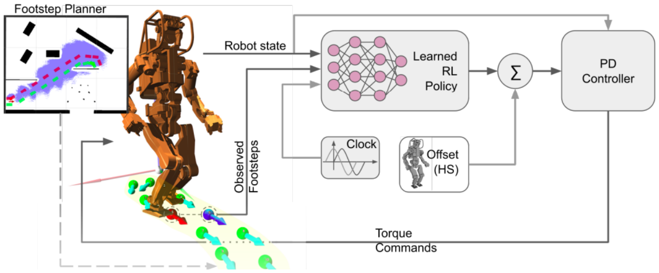
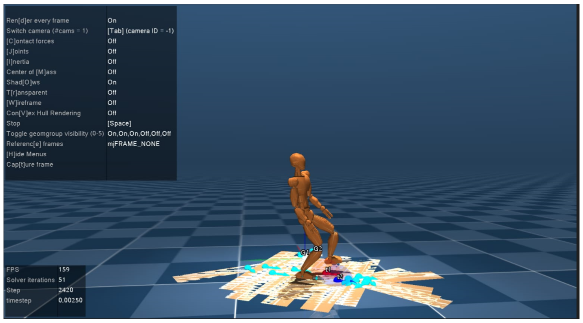


Fig 1: Control schematic

The external state is described by the 3D position and 1D heading of the two upcoming steps T1 = [x1, y1, z1, θ1] and T2 = [x2, y2, z2, θ2], defined in the frame of the robot's root as rT1 and rT2, respectively. The heading θ acts as a reference for the desired root orientation of the robot.

Since a footstep planner is providing the sequence of footsteps along with their state T, we need not worry too much about various walking conditions or uneven terrain. Simply learning to follow the desired footsteps generated by the planner guarantees that the bot does not fall or stumble.

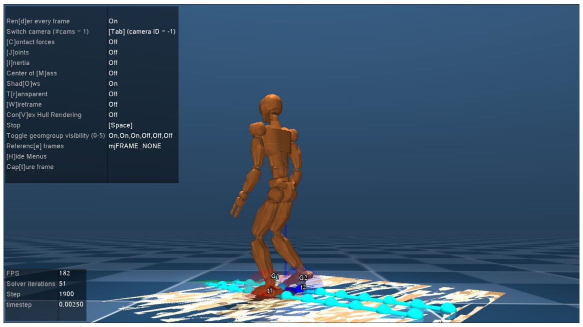
Shown below are various cases we have tested:



```
Ren[d]er every frame                 On
Switch camera (#cams = 1)            [Tab] (camera ID = -1)
[C]ontact forces                     Off
[J]oints                             Off
[I]nertia                            Off
Center of [M]ass                     Off
Shad[O]ws                            On
T[r]ansparent                        Off
[W]ireframe                          Off
Con[V]ex Hull Rendering              Off
Stop                                 [Space]
Toggle geomgroup visibility (0-5)    On,On,On,Off,Off,Off
Referenc[e] frames                   mjFRAME_NONE
[H]ide Menus
Cap[t]ure frame

FPS                  159
Solver iterations    51
Step                 2420
timestep             0.00250
```

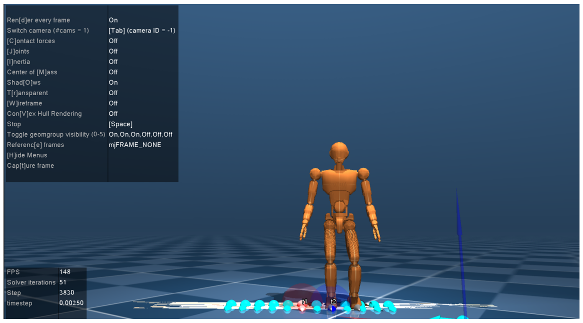
Forward walking



```
Ren[d]er every frame                 On
Switch camera (#cams = 1)            [Tab] (camera ID = -1)
[C]ontact forces                     Off
[J]oints                             Off
[I]nertia                            Off
Center of [M]ass                     Off
Shad[O]ws                            On
T[r]ansparent                        Off
[W]ireframe                          Off
Con[V]ex Hull Rendering              Off
Stop                                 [Space]
Toggle geomgroup visibility (0-5)    On,On,On,Off,Off,Off
Referenc[e] frames                   mjFRAME_NONE
[H]ide Menus
Cap[t]ure frame

FPS                  182
Solver iterations    51
Step                 1900
timestep             0.00250
```

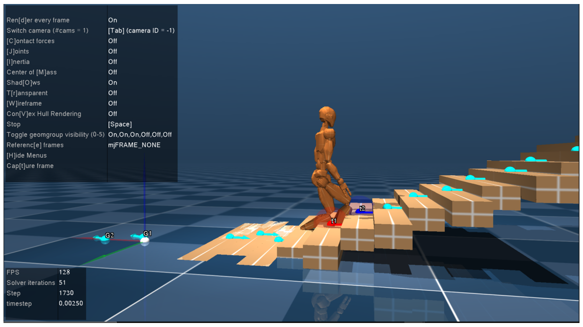
Fig 2: Reverse walking



```
Ren[d]er every frame                 On
Switch camera (#cams = 1)            [Tab] (camera ID = -1)
[C]ontact forces                     Off
[J]oints                             Off
[I]nertia                            Off
Center of [M]ass                     Off
Shad[O]ws                            On
T[r]ansparent                        Off
[W]ireframe                          Off
Con[V]ex Hull Rendering              Off
Stop                                 [Space]
Toggle geomgroup visibility (0-5)    On,On,On,Off,Off,Off
Referenc[e] frames                   mjFRAME_NONE
[H]ide Menus
Cap[t]ure frame

FPS                  148
Solver iterations    51
Step                 3830
timestep             0.00250
```

Fig 3: Sideways walking



```
Ren[d]er every frame                 On
Switch camera (#cams = 1)            [Tab] (camera ID = -1)
[C]ontact forces                     Off
[J]oints                             Off
[I]nertia                            Off
Center of [M]ass                     Off
Shad[O]ws                            On
T[r]ansparent                        Off
[W]ireframe                          Off
Con[V]ex Hull Rendering              Off
Stop                                 [Space]
Toggle geomgroup visibility (0-5)    On,On,On,Off,Off,Off
Referenc[e] frames                   mjFRAME_NONE
[H]ide Menus
Cap[t]ure frame

FPS                  128
Solver iterations    51
Step                 1730
timestep             0.00250
```

Fig 4: Walking on steps

2. AI Bot for football

The actor model performs the task of learning what action to take under a particular observed state of the environment. In our case, it takes the RGB image of the game as input and gives a particular action like shoot or pass as output. We send the action predicted by the Actor to the football environment and observe what happens in the game. If something positive happens as a result of our action, like scoring a goal, then the environment sends back a positive response in the form of a reward. If an own goal occurs due to our action, then we get a negative reward.

The job of the Critic model is to learn to evaluate if the action taken by the Actor led our environment to be in a better state or not and give its feedback to the Actor, hence its name. It outputs a real number indicating a rating (Q-value) of the action taken in the previous state. By comparing this rating obtained from the Critic, the Actor can compare its current policy with a new policy and decide how it wants to improve itself to take better actions.

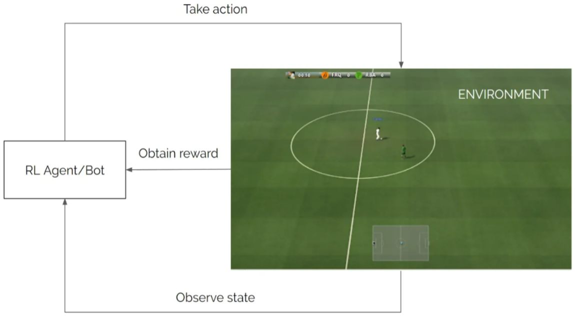


Fig 5: Football bot Schematic

Fig 6: Agent scoring a goal

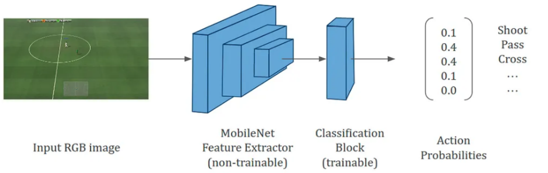

0.1
0.4
0.4
0.1
0.0
Shoot
Pass
Cross
...
...
Input RGB image
MobileNet Feature Extractor (non-trainable)
Classification Block (trainable)
Action Probabilities

Fig 7: The actor model

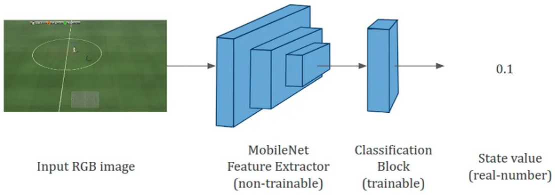

0.1
Input RGB image
MobileNet Feature Extractor (non-trainable)
Classification Block (trainable)
State value (real-number)

Fig 8: The critic model

Conclusion

The applications of Proximal Policy Optimization and Actor-Critic methods in humanoid walking and teaching an AI-agent to play football/soccer using reinforcement learning and demonstrate the effectiveness of these techniques in training the agents in complex environments. By combining policy optimization and value estimation, these methods address stability issues and provide a powerful framework for agent training. Future research should focus on addressing these challenges and further improving the performance of RL agents in these domains.

References:

[1] Rohan P. Singh , Mehdi Benallegue , Mitsuharu Morisawa1 , Rafael Cisneros , Fumio Kanehiro | Learning Bipedal Walking On Planned Footsteps For Humanoid Robots

[2] Rohan P. Singh, Zhaoming Xie, Pierre Gergondet, Fumio Kanehiro | Learning Bipedal Walking for Humanoids with Current Feedback

[3]https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6