GPT2PPO: Auto Regressive Proximal Policy Optimization

1st Andrei Cozma
Department of Electrical Engineering & Computer Science
University of Tennessee
Knoxville, United States
acozma@vols.utk.edu

2nd Hunter Price
Department of Electrical Engineering & Computer Science
University of Tennessee
Knoxville, United States
hprice7@vols.utk.edu

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—Reinforcement Learning, PPO, GPT2**

I. Introduction

In this project we explore the use of transformers in the context of Reinforcement Learning. The majority of theoretical works assume that problems follow a Markovian process, which is not always the case. Some problems need the contxt of previous states and actions to make an informed decision on the next decision. As a result, we propose an addition to the basic Proximal Policy Optimization (PPO) algorithm by using the Generative Pre-trained Transformer 2 (GPT2) model as the encoder for the critic network. This will allow the critic network to take into account the context of previous states and actions as well as apply attention to past states and actions that may be important. We will test this model on the LunarLander-v2 and Acrobot-v1 OpenAi Gym environments with discrete action spaces and compare it to the original PPO algorithm. Additionally we will test the model on BipedalWalker-v3 with continuous action spaces and compare it to the original PPO algorithm.

II. Previous Work

III. Background

All of the environments used in this project are from the OpenAI Gym library [1]. The environments are described below.

A. Lunar Lander

The Lunar Lander environment is a rocket trajectory optimization problem[1] shown in Figure 1. The goal is to actuate the lander to the landing pad at coordinates (0,0) without crashing. OpenAI Gym offers two versions of the environment: discrete or continuous. In this work we only use the discrete version. The state space is a 8-dimensional vector containing the x and y positional coordinates of the agent, its x and y linear velocities, its angle, its angular velocity, and two booleans that represent whether each leg is in contact with the ground or not. The action space is a single discrete scalar with values ranging from 0 to 3. The values corresponspond to the following actions: do nothing, fire left orientation engine, fire main engine, fire right orientation engine. The reward structure contains both positive and negative rewards. If the lander moves away from the landing pad, it gains a negative reward. If the lander crashes, it receives an -100 reward. If it comes to rest, it receives an +100 reward. Each leg with ground contact is +10 points. Firing the main engine is -0.3 points each frame. Firing the side engine is -0.03 points each frame. Firing the side engine is -0.03 points each frame. The landers initial state is at the top center of the environment with a random intial force applied to its center. The episode ends if the lander crashes, goes outside of the viewport, or comes to a resting position.

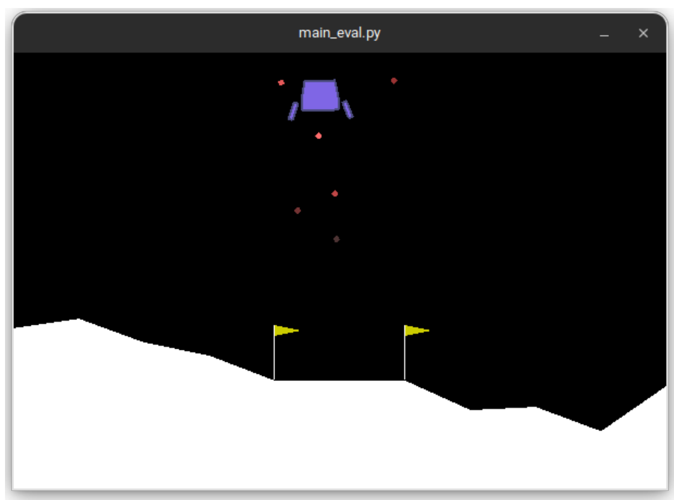

Fig. 1. The Lunar Lander environment.

B. Acrobot

Open AI Gym's implementation of the Acrobot environment[2] is based off the work of Sutton and Barto [2] shown

[1]OpenAI Gym Lunar Lander: https://www.gymlibrary.dev/environments/box2d/lunar_lander

[2]OpenAI Gym Acrobot: https://www.gymlibrary.dev/environments/classic_control/acrobot

in Figure 2. The environment is a 2-link pendulum with only the second joint actuated with a discrete action space. The goal is to swing the end of the pendulum up to a given height. The state space is a 6-dimensional vector containing the sin and cos of the two joint angles and the joint angular velocities. The action space is a single discrete scalar with values ranging from 0 to 5. The values corresponspond to the following actions: apply +1, 0, or -1 torque to the actuated joint. The reward structure contains only negative rewards. At each timestep the agent receives a reward of -1 for each step that does not reach the goal. If the goal is reached the agent receives a reward of 0. The episode ends if the agent reaches the goal height or if the episode exceeds the maximum number of timesteps.

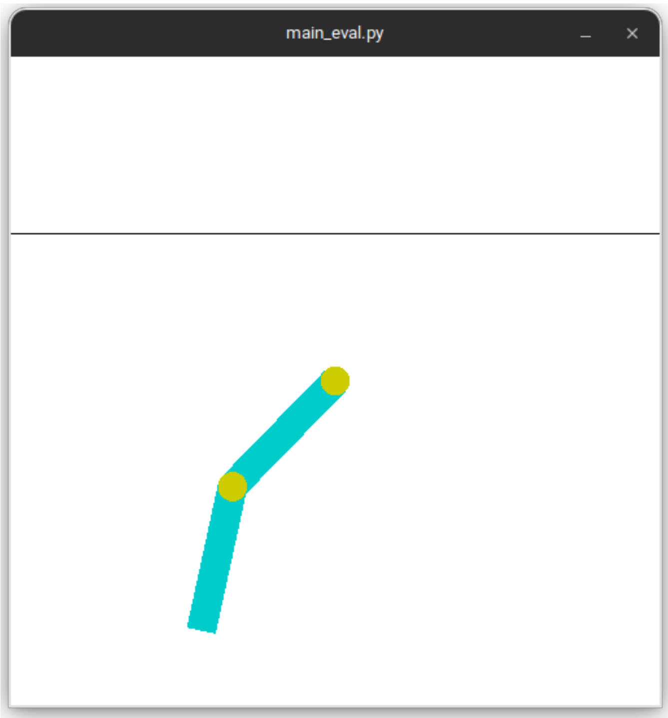

Fig. 2. The Acrobot environment.

C. Bipedal Walker

The Bipedal Walker environment is a 2D simulation of a bipedal walker with 4-joints[3] shown in Figure 3.

[3]OpenAI Gym Acrobot: https://www.gymlibrary.dev/environments/box2d/bipedal_walker

Fig. 3. The Bipedal Walker environment.

IV. Methodology

A. Reinforcement Learning Methods

B. Code Design

C. main_train.py

D. main_eval.py

V. Results

VI. Conclusion

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.