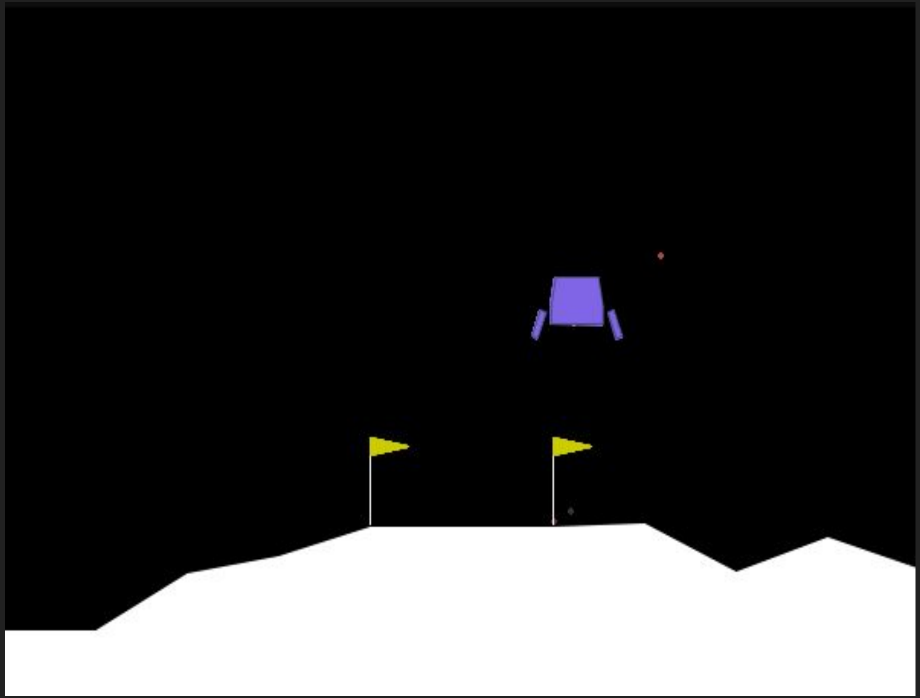
Reinforcement Machine Learning on OpenAI’s Lunar Lander

By Amrit Banga, Cole Clark, and Grant Erickson

Problem

- How effectively could reinforcement machine learning models perform on OpenAI’s Lunar Lander environment?

- How would this performance compare to pre existing ML algorithms and human testing?



Solution

- Develop Three ML algorithms:
  - Q-Learning
  - Deep Q-Network (DQN)
  - Advantage Actor-Critic (A2C)
- Train the models over the Lunar Lander environment
- Analyze each of the algorithms performance
- Compare that performance to pre existing algorithms and human performance
  - Proximal Policy Optimization (PPO)

Lunar Lander Environment

- Action space
  - Do nothing
  - Fire left engine
  - Fire main engine
  - Fire right engine
- Observation space
  - X and y coordinates
  - X and y velocities
  - Angle
  - Angular velocity
  - Booleans to indicate whether the legs are touching the ground

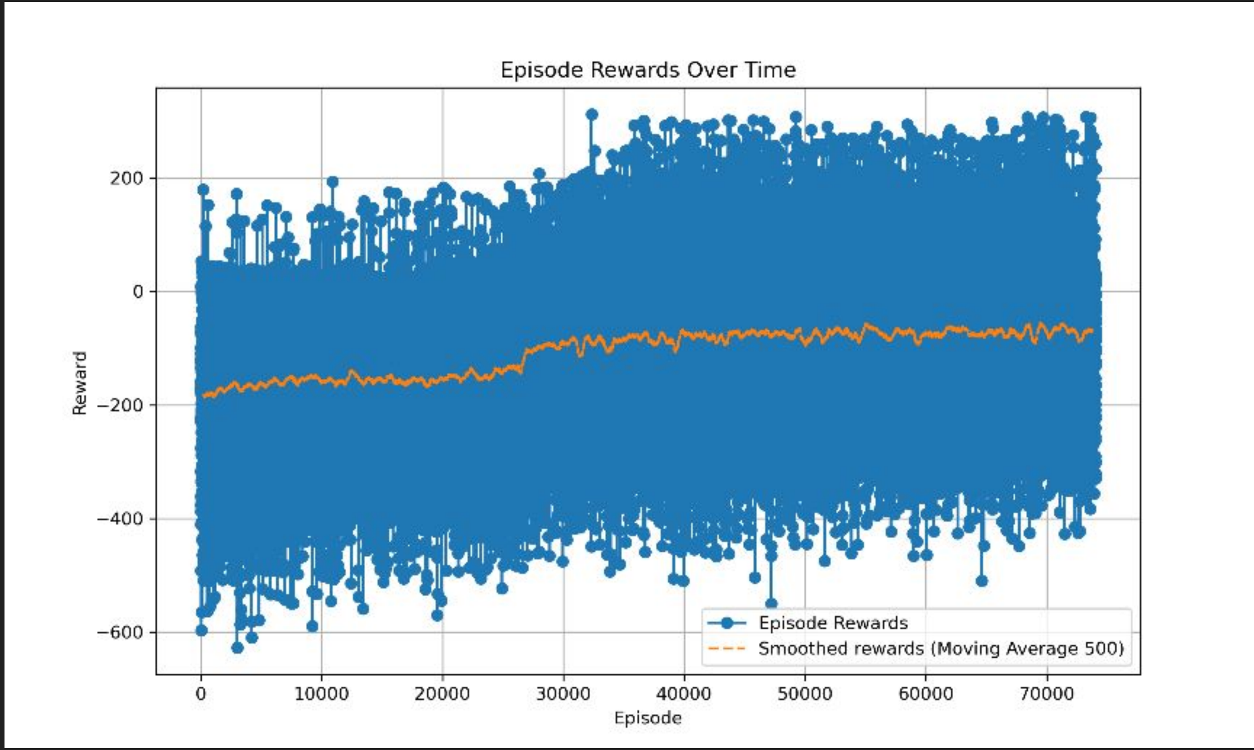
Lunar Lander Environment

- Rewards
  - Moving closer to the landing pad
  - Moving slowly
  - 10 points for each leg touching the ground
  - 100 points for landing safely
- Penalties
  - Moving away from the landing pad
  - Moving too fast
  - Not staying horizontal
  - -0.03 for firing side engine
  - -0.3 for main engine firing
  - -100 for crashing
- 200+ points is considered a successful landing

Q Learning Description

- Keeps weights for each state action pair
  - A state is the observation space (8 float values)
- Updates weights based on the result of taking an action
- Problems
  - Too many states since modifying one number by a small amount creates a new state
  - The agent may only try one action from a state without exploring
- Solutions
  - Round each value in the observation space to nearest whole number
  - Choose some random actions while training (decrease odds of random action over time)

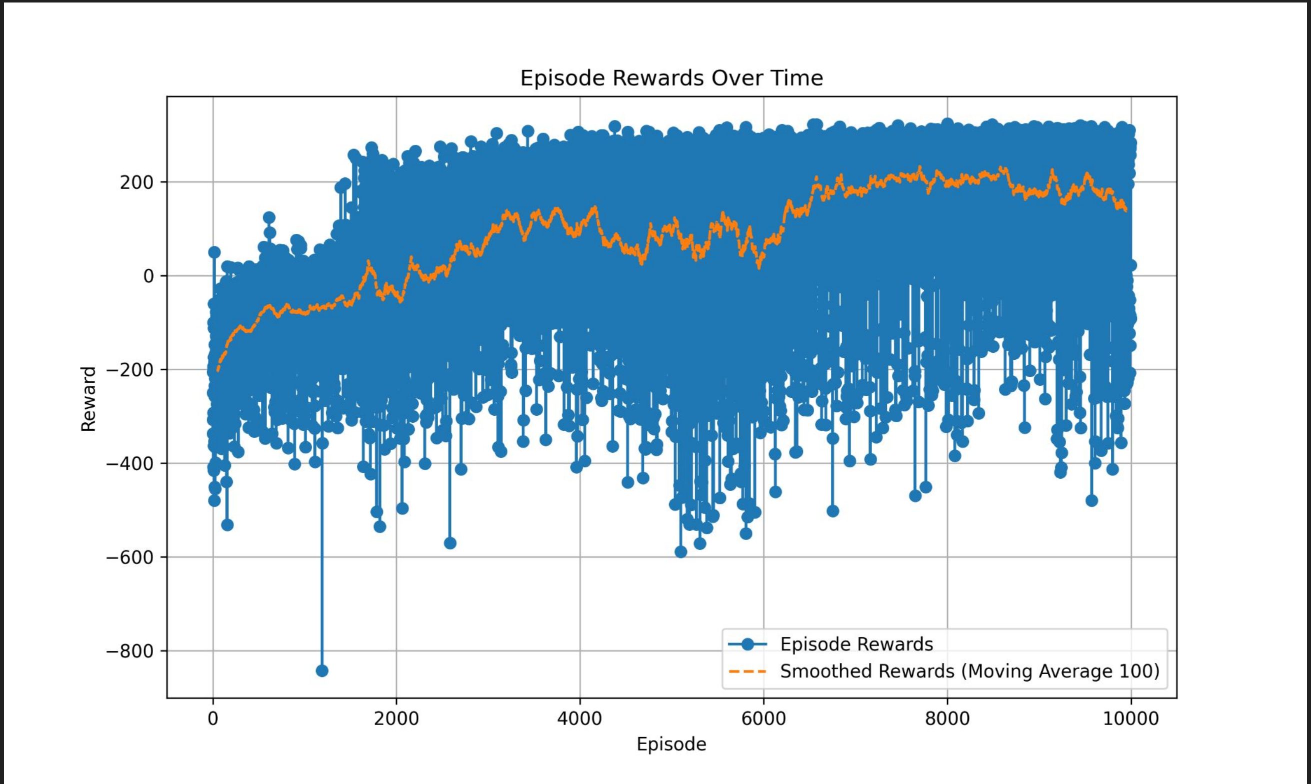
Q Learning Results



DQN Description

- How does it work?
  - Interaction is made with the environment
  - Agent stores the its experiences in a replay buffer
  - Use a two neural network system
    - Main network
    - Target network
- Problems
  - Training is computationally expensive and therefore slow
- Solutions
  - Prioritized Experience Replay
  - Added Reward Scaling

DQN Results



Episode Rewards Over Time
Reward
Episode
200
0
−200
−400
−600
−800
0
2000
4000
6000
8000
10000
Episode Rewards
Smoothed Rewards (Moving Average 100)

A2C Description

How It Works:

- Actor chooses actions; Critic evaluates how good those actions are.
- Uses Advantage Estimation to decide if an action is better than average.

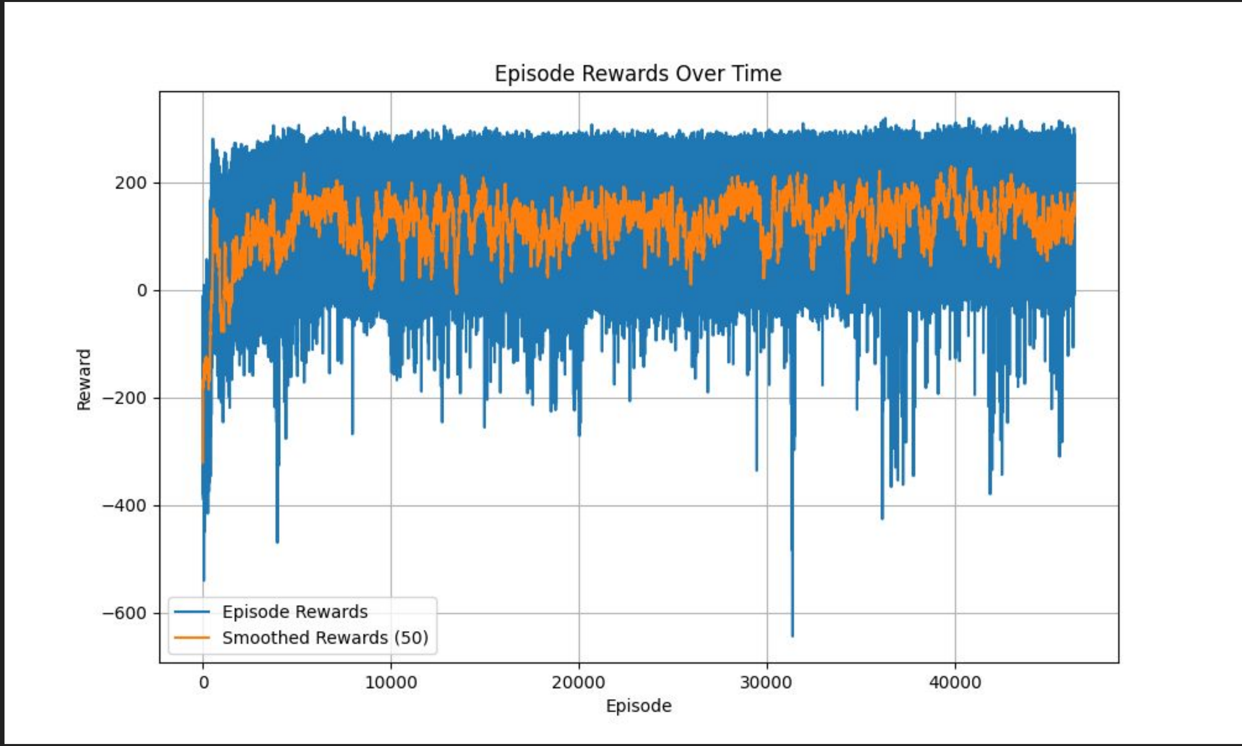
Problems:

- Actions can have high variance, making training unstable.
- Hard to balance exploring new actions vs. using what's already known.
- Sensitive to hyperparameters like learning rate.

Solutions:

- Use Advantage Normalization to stabilize training.
- Add Entropy to encourage exploration.
- Clip Gradients to prevent extreme updates.

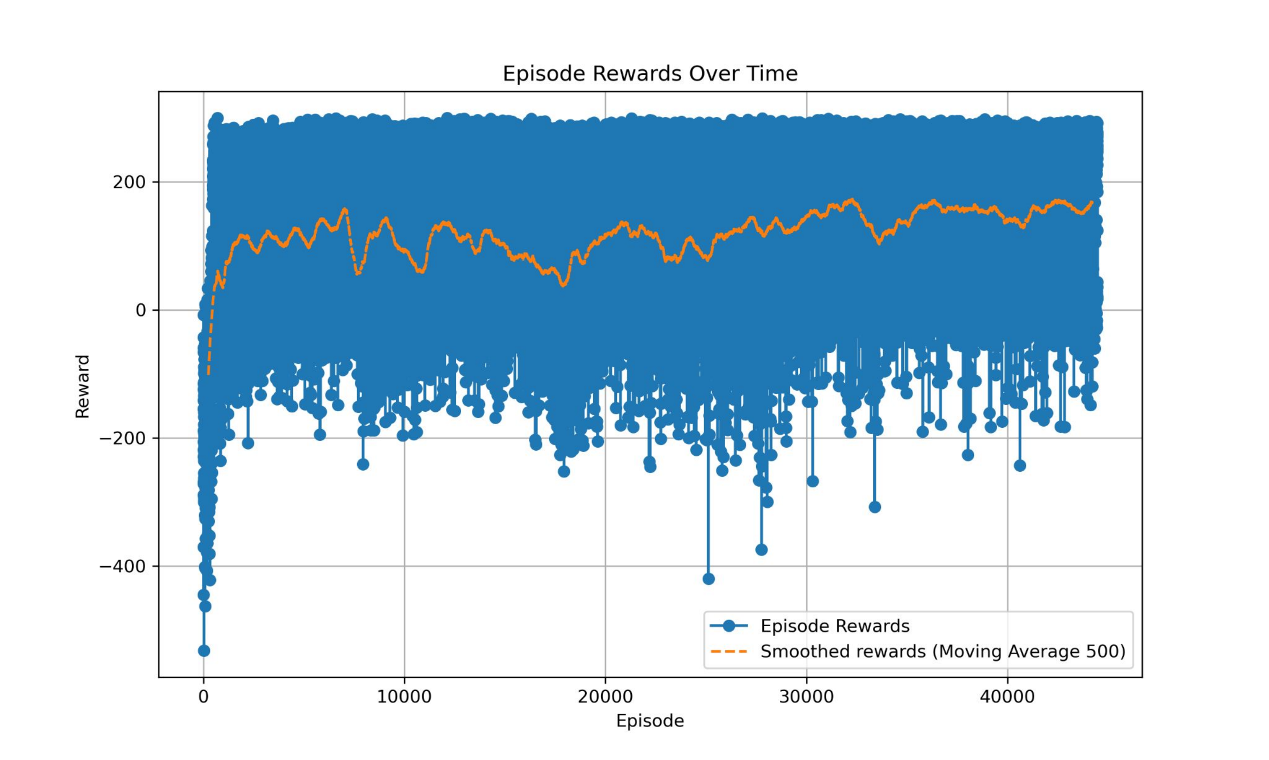
A2C Results


Episode Rewards Over Time
Reward
Episode
200
0
−200
−400
−600
0
10000
20000
30000
40000
Episode Rewards
Smoothed Rewards (50)

PPO Description

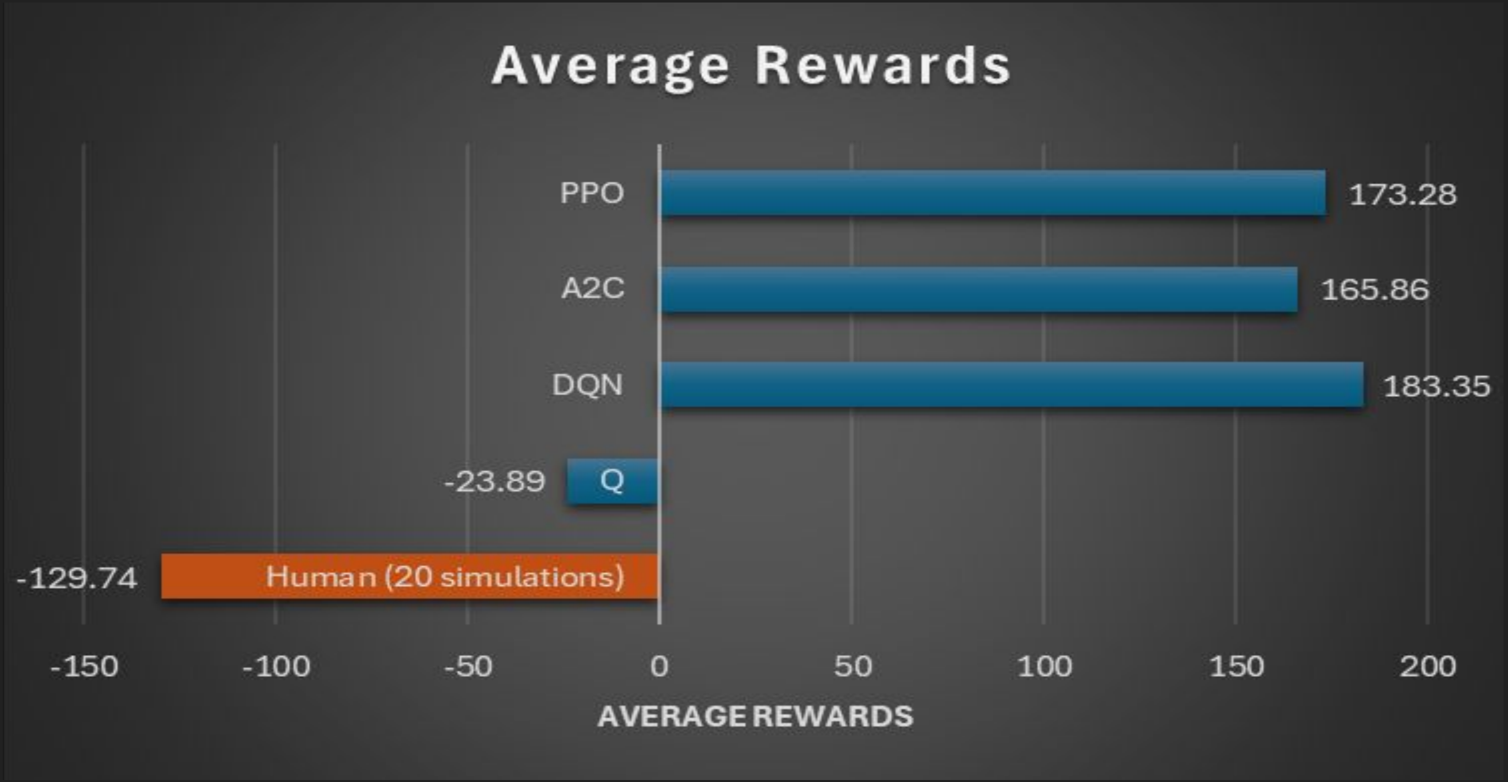
- Our PPO is modified from Costa Huang's tutorial to work on the lunar lander environment
    - https://github.com/vwxyzjn/ppo-implementation-details/blob/main/ppo.py
- Very similar to A2C but improves on it and other older methods
- Differences
    - PPO always includes clipping when calculating loss while some A2C algorithms might not
    - A2C is often simpler than PPO
    - More stable and efficient

PPO Results



Episode Rewards Over Time
200
0
−200
−400
Reward
0
10000
20000
30000
40000
Episode
Episode Rewards
Smoothed rewards (Moving Average 500)

Algorithm Comparison



Link to Code

https://github.com/cole853/CPTS437_project