Assignment 3: Actor-Critic

Status	Reinforcement Learning

Discuss the A2C algorithm you implemented.

The Actor-Critic learning technique combines elements of both policy gradient methods and the value-based methods. It uses two neural networks an actor network and a critic network, to learn a policy and a value function simultaneously. The actor network learns a policy that maps states to actions, while the critic network learns to estimate the value function, which represents the expected return from a given state under the current policy.

Part I Cartpole: Simple Environment

Implemented A2C with a four-dimensional state space and two possible actions. The actor (ANet) uses two fully connected layers with ReLU and softmax for action probabilities. The critic (CNet) has a similar structure but estimates state values.

ANet (Actor Network)

- Input Layer: Takes an input with 4 features.
- Hidden Layer: Expands the input to 128 neurons.
- Output Layer: Reduces the 128 neurons to 2 outputs, representing the probabilities of two possible actions.

CNet (Critic Network)

- Input Layer: Also starts with 4 features from the environment.
- Hidden Layer: Processes these through 128 neurons.
- Output Layer: Maps the 128 neurons to a single output, providing a scalar value estimation of the state's value.

Each network uses a ReLU activation function in the hidden layers to introduce non-linearity. The ANet uses a Softmax function in the output layer to convert logits into action probabilities, while the CNet directly outputs a scalar value without a final activation function. These structures and functions facilitate the actor's decision-making process and the critic's evaluation of those decisions.

Part II Bipedal Walker: Complex Environment 1

The AC network model combines an actor and a critic in one architecture, efficiently processing 24-dimensional input states:

- Shared Layers: Begin with a 128-neuron layer with ReLU activation.
- Actor Branch: Processes through a 64-neuron layer to output 4 action values.
- Critic Branch: Uses a similar 64-neuron layer to output a single state value estimate.

This structure enables simultaneous policy generation and value estimation, essential for making informed decisions in dynamic environments.

Part II Grid World: Environment 2

For a grid world, with minimal states we used small network: In AC model, here are the key numbers for each component:

Actor (Policy Network)

- Input: State vector of dimension inp_dim=16 states.
- Hidden Layer: 64 neurons with hid_dim.
- Output: Probabilities for out_dim = 4 actions, using Softmax.

Critic (Value Network)

- Input: Same as Actor, inp_dim.
- Hidden Layer: Also 64 neurons (matches Actor's hid_dim).
- Output: Single value estimation.

Both the actor and critic use a ReLU activation in their hidden layers. The model processes inputs to generate both action probabilities and state value estimates in one go.

What is the main difference between the actor-critic and value based approximation algorithms?

- Value-based methods focus on learning the value function, which represents the expected return (or cumulative reward) from a given state under a fixed policy. Actor-critic methods combine both value function approximation and policy optimization. They maintain two separate networks: an actor network, which learns the policy, and a critic network, which learns the value function.
- These methods directly approximate the value function and derive the policy implicitly from it. Common examples include Q-Learning, Deep Q-Networks (DQN), and variants.
- Value-based methods are often used in environments with discrete action spaces and are well-suited for tasks where the optimal action can be determined solely based on the value estimates.

- The actor network selects actions based on the current policy, while the critic network evaluates the chosen actions by estimating their value.
- By using a critic to provide feedback to the actor, these methods can learn a better policy more efficiently than pure policy gradient methods.
- Actor-critic methods are particularly useful in environments with continuous action spaces or in scenarios where the optimal action is not directly derivable from the value estimates.

Briefly describe THREE environments that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your Assignment 2 report.

Part I Simple Environment: CartPole-v1

- In CartPole-v1, the agent controls a cart that can move left or right, with the goal of balancing a pole placed on top of it.
- The observation space consists of a 4-dimensional vector representing the position and velocity of the cart, and the angle and angular velocity of the pole.
- The action space is discrete, with two possible actions: move left or move right.
- The agent receives a reward of +1 for each timestep it manages to keep the pole balanced, and the episode terminates when the pole either falls beyond a certain angle threshold or the cart moves too far from the center.

Part II Complex Environment 1: BipedalWalker-v3

- In BipedalWalker-v3, the agent controls a bipedal robot with the goal of moving forward without falling.
- The observation space is a 24-dimensional vector representing various aspects of the robot's state, such as position, angle, velocity, and angular velocity of its body parts.
- The action space is continuous, with four possible actions corresponding to the torque applied to each leg joint.

- The agent receives rewards based on its movement, with higher rewards for moving forward and penalties for falling or making sharp movements.
- NOTE: Ideally, 200 to 300 points in 1600 episodes have to be achieved for solving bipedal, however there are several changes to recent versions which are not consistent.

Part II Grid Environment 2:

```
# start position of our skater
start_pos = [0, 0]

# goal position of our skater
goal_pos = [3, 3]

# Gems positions
gem1 = [0, 2]
gem2 = [3, 2]

# Holes positions
hole1 = [1, 3]
hole2 = [2, 0]
```

- Goal Flag Reached (Positive Reward):
  If the goal position is reached, the agent receives a reward of 17. This encourages the agent to reach the goal.
- Skating on Hole (Negative Rewards):
  If the agent skates on the first hole, it receives a penalty of -5. If the agent slides on the second hole, the agent receives a penalty of -6. These negative rewards discourage the agent from skating over holes as it will drown.
- Gem Collections (Positive Rewards):
  If the agent reaches the location of the first gem, the agent receives a reward of 5. If the agent reaches the location of the second gem, the agent receives a reward of 6. These positive rewards encourage the agent to collect gems and win a lottery along the way.

- Distance to Goal (Dynamic Rewards):
  If the current distance to the goal is less than the previous distance to the goal, the agent receives a reward of 1. This provides a positive reinforcement for moving closer to the goal.

  If the current distance to the goal is greater than the previous distance to the goal, the agent receives a penalty of -1. This penalizes the agent for moving away from the goal.

  If there is no change in distance, the agent receives a slight negative reward of -0.1. This also handles the case when he is trying to go out of the grid.

  - **Set of Actions** defined in our case:
    - We have 4 main actions which include: Left, Right, Up and Down.
    - We also ensure to keep the agent within the grid by clipping over the boundaries and restoring his location from where he tried to move out of the grid.
    - If the agent is in the same position as the previous step, choose a different action.
    - The following code handles the direction movement and clipping:
  - **Set of States:**
    - We have 6 different states that a skater agent can end up in:

      1) Agent Initial State: This is where the skater starts his race to reach the goal position.

      - Initially he is at [0, 0] on the bottom left corner.

      2) Agent going out of the Grid State: If the skater tries to go out of the grid he ends up in the state.

      - Consider from the initial position, if the agent tries to go from [0,0] to [-1, 0] or [0, -1] then the agent ends up in this going out of the grid state.

3) In Hole State: If the agent ends up in a hole then he is in this state.

- Agent slips in the hole and drowns if he lands on the locations of the rocks are [2, 0] and [2, 3]

4) On Gems States: If the skater reaches the Gems he has won a lottery himself and then reach the goal with some gems.

- The locations where the gems are present is: [0, 2] and [3, 2]

5) Intermediate State: Any other random location that the agent skates the lake grid is the intermediate state.

- Let he be present at [2, 1], then it is an intermediate state.

6) Goal State: The final goal state indicates the termination of the agents race as he has reached the goal.

- The goal state is [3,3] in the frozen grid:

- **Main Objective:**
  - The primary objective is for the agent to learn a policy that maximizes the cumulative reward over time. The agent should navigate the grid, avoid obstacles (holes), reach the goal efficiently, and collect gems from two locations for winning a lottery. The dynamic rewards based on distance encourage the agent to find an optimal path to the goal.
  - In our environment, the maximum reward that can be achieved in a single episode is determined by the structure of the reward system.

    Reaching the Goal:

    - The agent receives a reward of 10 for reaching the goal position (`[3, 3]`).

    Collecting Gems:

    - The agent receives rewards for collecting gems. Based on the reward structure, there are two gems with rewards 5 and 6, respectively.

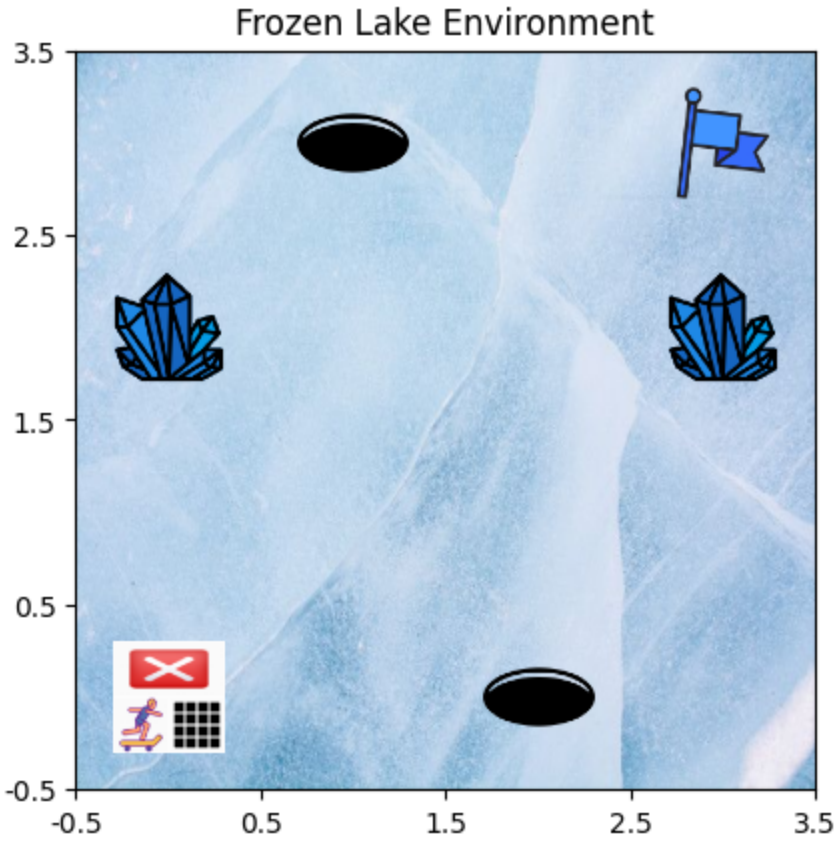
    Avoiding Holes:

- The agent should avoid holes, as touching them results in negative rewards (-5 and -6).

Moving Towards the Goal:

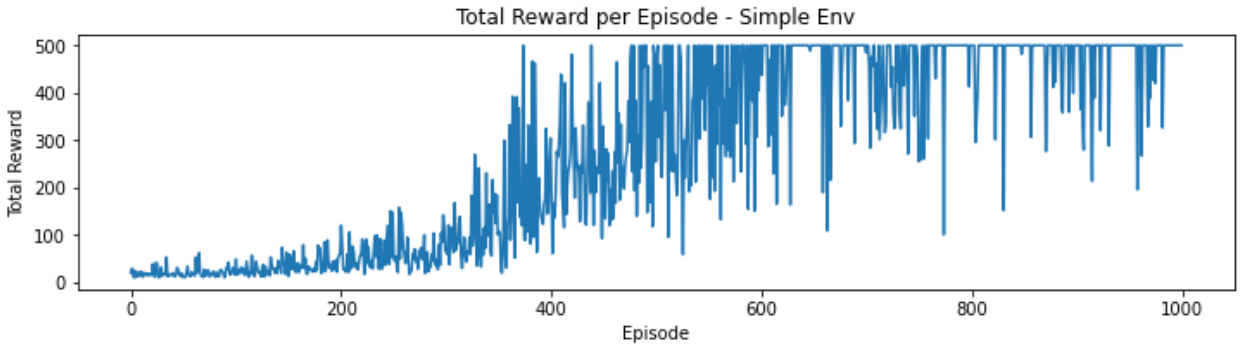
- Each step towards the goal could potentially earn a reward of 1. However, the exact reward depends on whether each step objectively decreases the distance to the goal.

Based on this, the maximum reward in a single episode would be achieved by collecting both gems and reaching the goal, while also making every move directly towards the goal.



Show and discuss your results after training your Actor-Critic agent on each environment. Plots should include the reward per episode for THREE environments. Compare how the same algorithm behaves on different environments while training.

Part I Simple Environment: Cartpole-v1

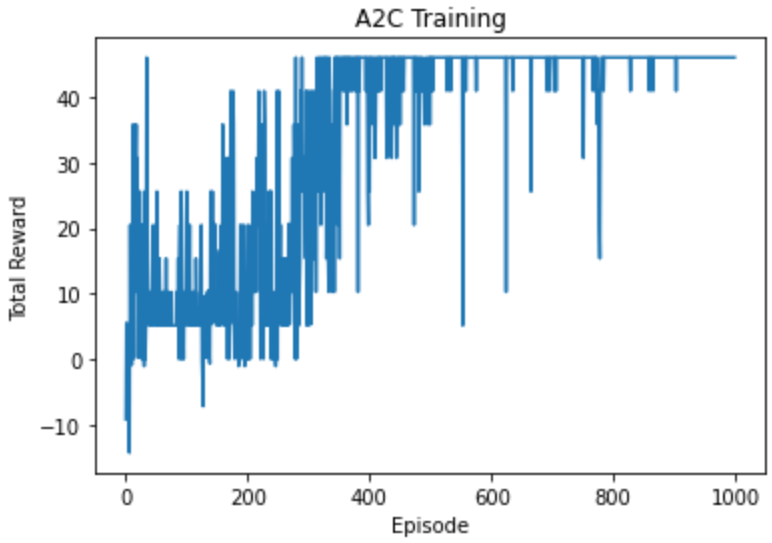


- Trained an Actor-Critic agent in 'CartPole-v1' for 1000 episodes.
- Reward discount factor set to 0.99.
- Rewards calculated, using next state values for current return estimates.
- Plot showed rewards per episode, indicating learning progression with increasing rewards and learning the OPTIMAL policy for Cartpole.
- The agent learned to balance the pole better as training progressed, shown by rising rewards.

Part II Complex Environment 1: BipedalWalker-v3

- Across 1600 episodes in 'BipedalWalker-v3', we need to get 200 to 300 around, however upon several tries, I found there are issues with the latest version of bipedal. I tried with 25000 episodes and changing network and hyperparameter tuning. However my moving average has shown a rewards structure with increasing trend. showcasing learning however not optimal rewards. sometimes in between it dips as well.
- Despite the fluctuations of rewards, towards the end of training, the agent starts showing signs of settling into a groove, managing to hold onto a relatively steady reward range, showing a better policy.
- In stark contrast to the 'CartPole-v1' environment where our agent consistently hit the reward ceiling, 'BipedalWalker-v3' threw a tougher puzzle at our agent, leading to a more unpredictable and fluctuated learning experience.

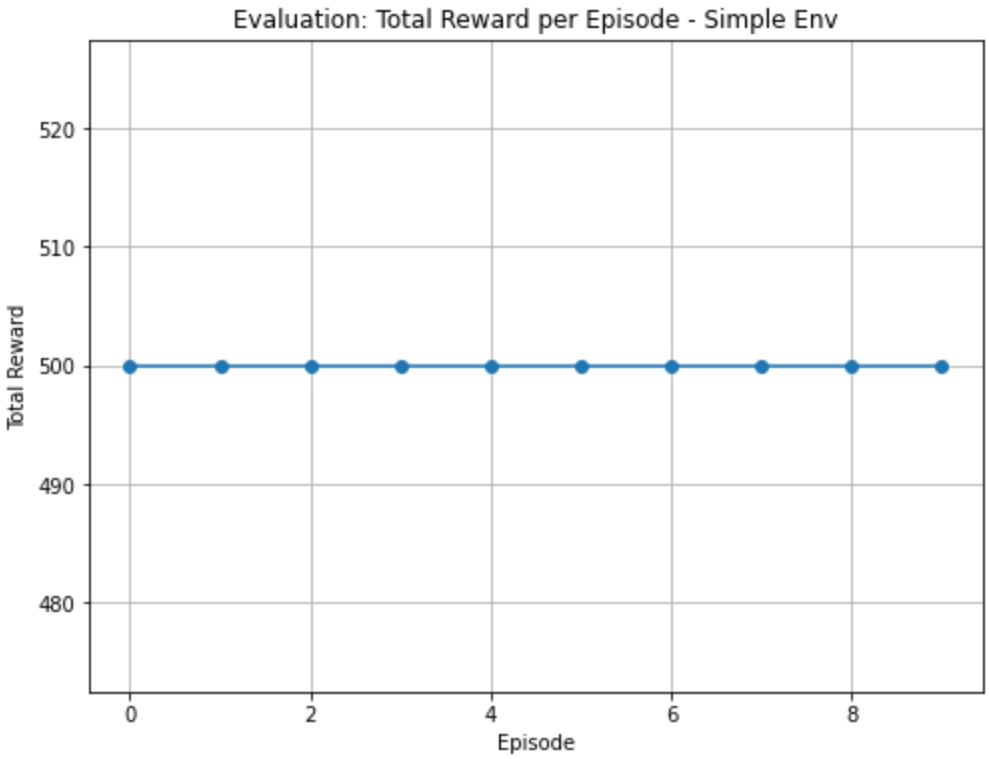
Part II Grid World Environment 2:



- **Optimal Policy:**
  - MAX Timesetps allowed = 10
  - Consider the below image:
    Episode Actions: up(1), up(1+5), right(1), right(1), right(6+1), down(-1), up(6+1), down(-1), up(6+1), up(17+1) = TOTALLING = 46
  - Ideally this would be the best rewards. The agent has learnt the policy really well, shouwing the upwards trend. Trainged over 1000 episodes.

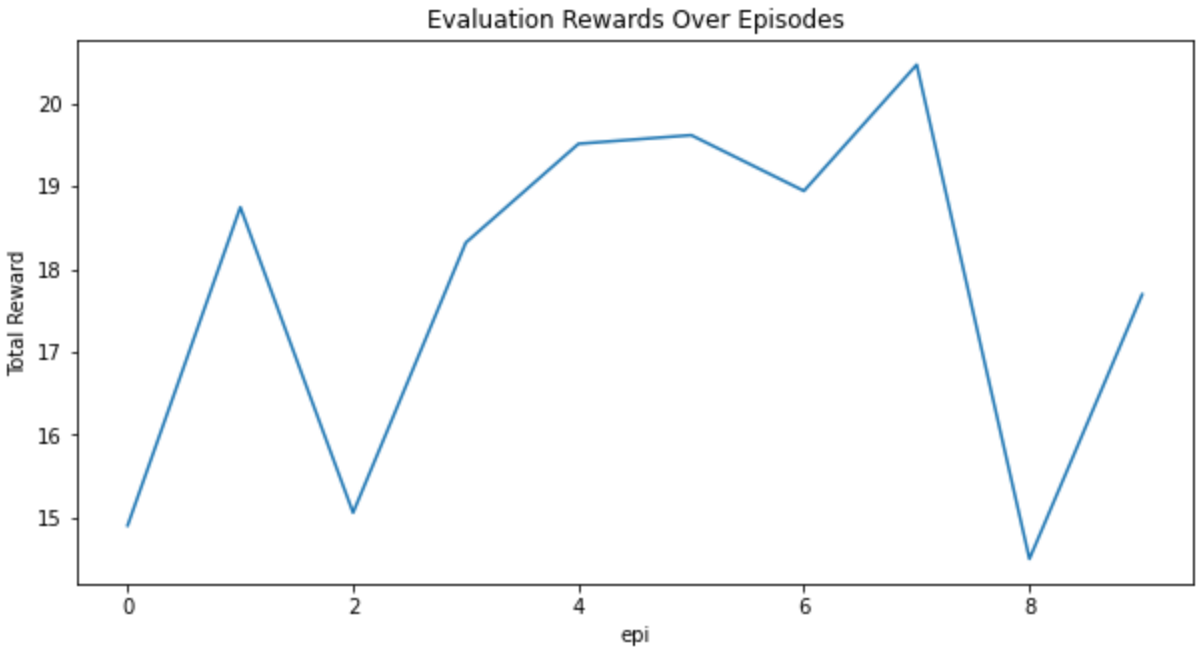
Provide the evaluation results for each environments that you used. Run your environments for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

Part I Simple Environment: Cartpole-v1



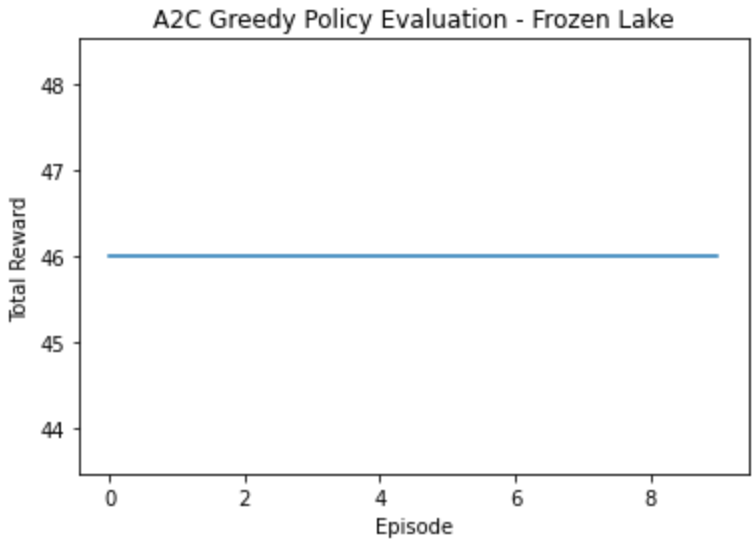
- Evaluated the Actor-Critic agent in 'CartPole-v1' for 10 episodes.
- Agent chose actions greedily based on the learned policy.
- Achieved the maximum reward of 500.0 in every episode.
- Performance was stable and consistent, indicating a well-learned policy.
- Learnt well the environment, showing reliable and robust behavior.

Part II Complex Environment 1: BipedalWalker-v3



- Over 10 episodes, the agent in 'BipedalWalker-v3' used its learned policy to select actions, strictly choosing the highest-value (greedy) option at each step.
- The evaluation scores fluctuated slightly, suggesting the agent's decisions weren't always perfect, reflecting the inherent complexity and unpredictability of the environment.

Part II Grid World Environment 2:



Agent learnt the greedy policy, evaluated for 10 episodes it has consistently received the optimal rewards, for 10 Timesteps.

If you are working in a team of two people, we expect equal contribution for the assignment. Provide contribution summary by each team member.

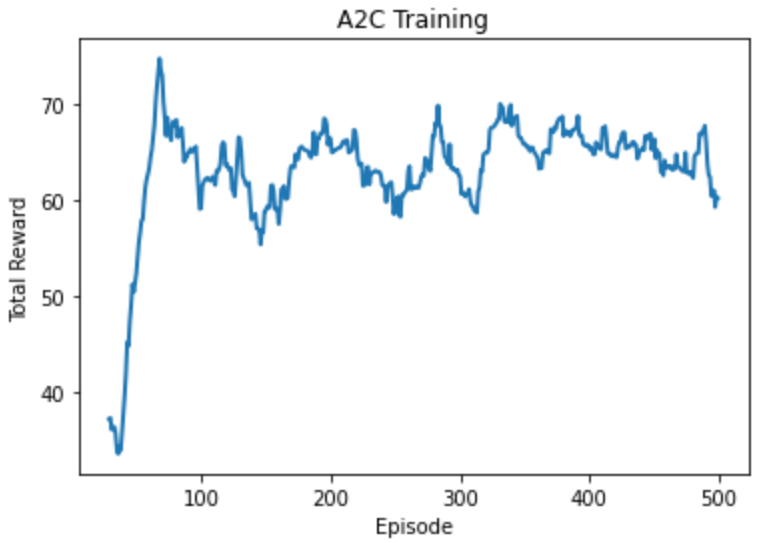
Only one member

BONUS

Solve Image-based Environment [5 points]
Use one of the environments with image representation of the state that requires a utilization of CNN (Convolution Neural Network) for the state preprocessing (e.g. Breakout).

The CarRacing environment falls into this category, where the state is represented by pixels from the game's image frame. To process this type of image data effectively, a Convolutional Neural Network (CNN) is used to extract features from the raw images, which helps in understanding the visual elements such as the track layout, obstacles, and the car's position. These features then become the input state for the agent's decision-making process. Using a CNN to preprocess the image data allows the agent to learn policies based on the visual cues, which is essential in environments like CarRacing where the state space is high-dimensional and rich in visual information.

- In CarRacing-v2, the agent controls a race car on a track with the goal of completing the race as quickly as possible.
- The observation space is a 96×96×3 image representing the RGB pixels of the game screen.
- The action space is continuous, allowing the agent to control the car's steering, acceleration, and braking.
- The agent receives rewards based on its position and speed on the track, with higher rewards for staying on the track, maintaining high speed, and completing laps efficiently.
- Car Racing Training:



REFERENCES

- https://gymnasium.farama.org/environments/box2d/car_racing/
- https://hiddenbeginner.github.io/study-notes/contents/tutorials/2023-04-20_CartRacing-v2_DQN.html

https://github.com/Kyziridis/BipedalWalker-v2

https://github.com/nikhilbarhate99/TD3-PyTorch-BipedalWalker-v2/blob/master/train.py