RL Final Project

Name: 陳以瑄 Student ID: 109705001

Methodology Introduction

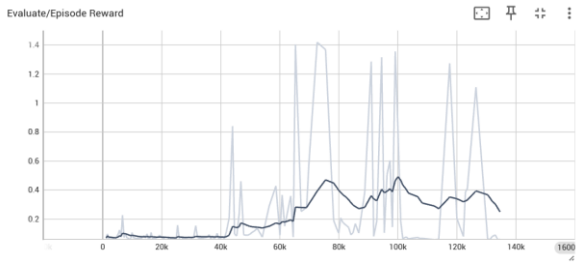
- **Introduction of the methodology used for each map**
  I use PPO for both Circle & Austria.
- **Explanation for the choice of methods for each map**
  Since using continuous actions, the agent needs to handle a wide range of actions, which makes the learning process difficult. Therefore, I've chosen PPO, which is an algorithm that works well with discrete action spaces This decision is informed by past experiences, where using DDQN in the Enduro environment took a considerable amount of time and still failed to train effectively. However, employing PPO yielded excellent results effortlessly.

Experiment Design and Implementation

- **Description of the training processes**
  a. Design of reward function
     I have set up special rewards for the following four scenarios
     1. wrong way => reward = -0.0006
     2. wall collision and at sharp turn => reward = -0.03 (the agent should start to turn right before 0.03 progress)
     3. other wall collision => reward = -0.01
     4. multiply the reward by 500 before storing them in the buffer, so that the critic can learn easier.

  b. Circle model
     Actions: Since this is a simple map, I have designed it as follows:
     - Motor: I set it to 1 to make the car run at the maximum speed.
     - Steer: Since the direction is clockwise, the car only needs to go straight or turn right. Thus, I set the steer action to have only two values: 0 and 1.

     Reward: I use the 1st and the 3rd special reward in this case.
     Other tricks: I use a frame stack (4 frames)
     Training curve:



c. Austria model

Since the Austria map is much more complicated than the circle map, the training process is much more complex.

Actions:

- Motor: Because the TA has advised that high speed may lead to collisions, I have set the motor action space to be {0.05, 0, -0.05}
- Steer: I set the action space to be {-1, 0, 1}
- Total action space: the combination of the above actions, 9 in total.

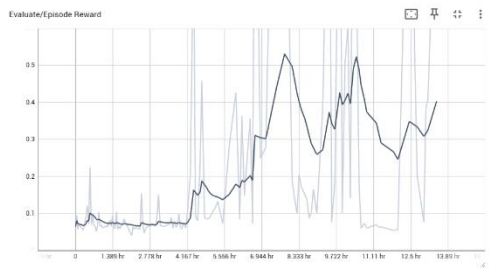
Reward: I use all four special rewards in this case.

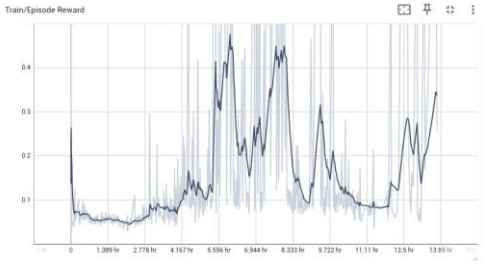
Other tricks:

- Frame stack: The TA suggested that there is just a slight difference between one frame and the next, but directly using frame skip may result in persisting with the same action for too long, accumulating too much noise at once. Therefore, I've adopted a compromise solution. I have five sets of frame stacks, each formed by stacking frames at intervals of five. For instance, {0,5,10,15}, {1,6,11,16}, {2,7,12,17}, {3,8,13,18}, {4,9,14,19}. This way, each frame stack captures information within 0.32 seconds (16 frames * 0.02 seconds). Due to multiple stacks, no frames are skipped.
- Random starting point: TA pointed out that Austria has a variety of turns, running the agent from the beginning each time could lead to confusion. Therefore, I use a random start point so that the agent can learn different turns without always starting from the beginning.
- collisionStop env: I configured the environment to 'austria_competition_collisionStop' to prevent the agent from stacking frames before and after a collision, which is not right. This way also prevents the agent from misunderstanding that colliding with a wall allows for teleportation, leading to high rewards, and a tendency to deliberately collide with walls.

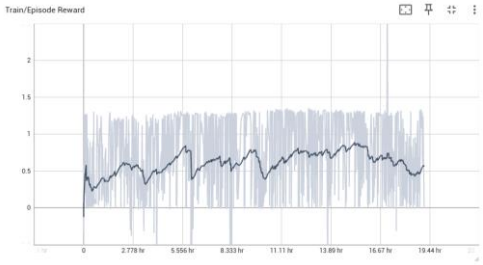
Training curve:

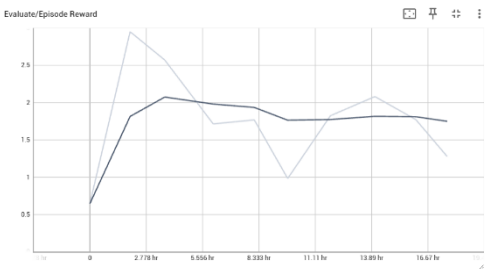
- The plot after setting the action and doing the special frame stack trick:





- The plot after every trick:





- **Neural network architectures**

  I directly use the network architecture in Lab 3.

```
self.cnn = nn.Sequential(nn.Conv2d(4, 32, kernel_size=8, stride=4),
                                nn.ReLU(True),
                                nn.Conv2d(32, 64, kernel_size=4, stride=2),
                                nn.ReLU(True),
                                nn.Conv2d(64, 64, kernel_size=3, stride=1),
                                nn.ReLU(True)
                                )
self.action_logits = nn.Sequential(nn.Linear(1024, 512),
                                nn.ReLU(True),
                                nn.Linear(512, num_classes)
                                )
self.value = nn.Sequential(nn.Linear(1024, 512),
                                nn.ReLU(True),
                                nn.Linear(512, 1)
                                )
```

- **Details of the hyper-parameters**

  This part is almost the same as the setting in Lab 3, the only difference is that, since I use a special frame stack trick, I need to define the ‘stack_amount’ and ‘stack_interval’

```
"update_sample_count": 10000,
"discount_factor_gamma": 0.99,
"discount_factor_lambda": 0.95,
"clip_epsilon": 0.2,
"max_gradient_norm": 0.5,
"update_ppo_epoch": 3,
"learning_rate": 2.5e-4,
"value_coefficient": 0.5,
"entropy_coefficient": 0.01,
"stack_amount": 4,
"stack_interval": 5,
"horizon": 128,
```

- **List of packages, tools, or resources used**

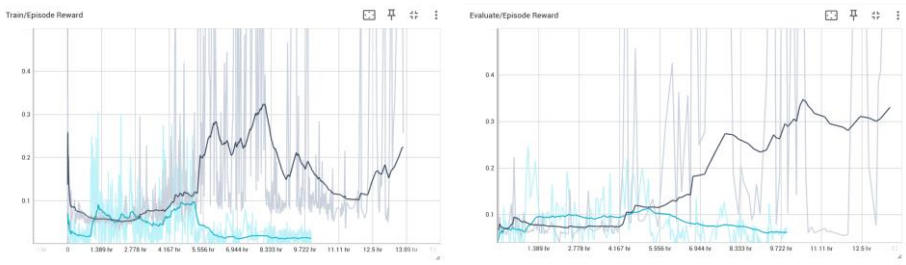
  I use the source code in Lab3, so the used packages are same as those in Lab3.

Method Comparison and Evaluation

- **Trying different methods and showing their results**

  I have trained DDQN with the same setting as the one I used in the PPO and trained for about 10 hours. However, it failed to learn, just like what it did in the Lab2.

Blue: DDQN, black: PPO.

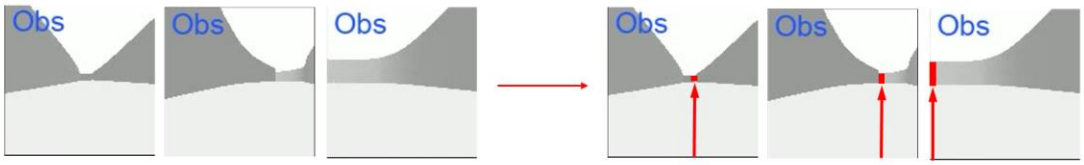


- **Comparing the effectiveness of different approaches**

  The result of the above graph shows that PPO is better than DDQN.

  Except for different algorithms, I also tried different scale of motor using 3 motor options (motor = {-x, 0, x}) with x = 0.06, 0.01, 1. With x = 0.01, it is trained well but it is quite slow (1.5 lap without collision). With x = 1, it is quite unstable, it usually collides at about 1.4 lap and it has probability of 0.05 to run one episode without collision.

  I also tried human feature extraction. While analyzing the agent's observation from the human view, I found out that a crucial aspect in deciding whether to turn left or right is to determine if the road's end is on the left, center, or right in the image. If it is on the right, it indicates that there is a right turn ahead, so we should turn to the right to navigate the bend. Therefore, for each state, I examined the end of the road by identifying the x-coordinate where the sky is closest to the ground, as illustrated in the diagram below.



  However, this human feature extraction doesn’t improve the performance in the end.

- **Analyzing the successful and unsuccessful cases**

  I would like to compare DDQN (unsuccessful case) and PPO (successful case). From the lectures, we learned that the main difference between DDQN and PPO is that the former learns a deterministic policy, while the latter learns a stochastic policy. In the example of the aliased grid world in the lecture, we observed that when observations are the same, a stochastic policy is a better choice. In the case of the first-person perspective racing scenario, I believe this is a reasonable explanation for the difference in results between DDQN and PPO.

- **Discussing the key observations and insights**

  1. Discrete v.s. Continuous in driving

Unlike the environments in previous labs, this environment uses 1st person view as observation, which is like a real person driving a car. As a person who drives cars, I consider driving under discrete control to be quite impossible, but the result shows that it indeed works. Discrete actions work because the timestep is small (0.02s), so continuous action is not necessary. Suppose you have action (0,0) on 0s and (1,1) on 0.02s, it is similar to using (0.5, 0.5) on 0.01s. By using many actions, you can also approximate the speed & change of direction when using continuous actions.

2. Human feature extraction
   The sharp turn is hard to observe using my hand-crafted feature extractor. It only observes one point in the middle when it should start to turn because it can't distinguish the flat line of a sharp turn and the normal line of a straight road. Designing and implementing feature extraction is time-consuming and might cause some cases in which the feature extractor might fail. Using a learning approach might take more time to train but it saves a lot of time when implementing the code, and the result is usually better.
3. Weaving when driving.
   I watched the video of the evaluation, and I observed that the agent sometimes goes left and right repeatedly when driving in a straight line. Because the agent moves left and right, it has a wider view when driving, making it able to observe rapid turns, so my agent then learned to drive left and right (or maybe because of the noise since I have small actions). This might cause the score to be lower but is way safer.

Challenges and Learning Points

- **Encountered challenges during the implementation process**
  1. Frame stack
     In the circle map, I directly stack continuous 4 frames, just as I did in the previous lab. However, this approach doesn't work well on the Austria map. That's why I adopted a special frame stacking technique, and this design has indeed proven to be effective.
  2. Collision penalty
     Since I added the collision penalty on my own, determining its appropriate value requires a considerable amount of trial and error. I initially anticipated that setting a high penalty would encourage the agent to avoid collisions, ultimately leading to higher scores. However, the results were not as expected; in fact, the agent's performance even worse.
- **The learnings from these challenges**

1. Frame stack
   Before the TAs shared their experiences, I did not find out that the 4 frames are too similar to each other. If I had used some tool like cv2 to get the 4 images, I would have found out this issue.
2. Collision penalty
   The collision penalty is a trade-off, if it's set too low, the agent may not prioritize the issue of collisions; if set too high, the agent might hesitate to move forward, choosing to stay in place.

Future Work

- **The proposal of ideas for potential improvements**
  1. Make the motor value larger
     Due to time and equipment constraints, I only tried motor parameters of 0.01, 1, and 0.06. Among them, 0.06, which falls in between, proved to yield the best performance. However, since it is a racing competition, I believe a slightly faster speed should result in better performance, such as 0.1 or similar values.
  2. A larger action space
     Due to the limited set of actions (only 3) in my discrete action space, I found that the agent struggles to drive smoothly, especially during turns. Perhaps introducing more actions could help address this issue.
- **The suggestion of additional research directions for the future**
  Use RNN or trajectory transformer because recent actions / observations are also important.