National Taiwan University
Department of Electrical Engineering
Name: Tzu-Tsen, Hsieh
ID: R11921072

113-1 (Fall 2024)
Reinforcement Learning

Assignment 3 Report

Q1. Try using at least two different algorithms to train your 2048 model. Present your results and discuss possible reasons why some algorithms perform better than others in this task. (12 pts.)

For 2048 models, I utilized A2C and DQN to train the model.

1 A2C

A2C is based on online learning, where the policy is updated in real-time at each time step based on the current strategy. A2C updates the policy using the policy gradient method, while the value estimate of the Critic helps reduce the variance of the updates, making the learning process more stable. However, if exploration is insufficient during training, the algorithm may prematurely converge to a narrow action space, causing A2C to fall into a local optimum. Additionally, the policy gradient update may smooth out the updates, making it difficult to break through the current local maximum. Experimental results show that A2C converges faster than DQN, but this early convergence leads to a long period where the results remain nearly the same, and reaching higher performance, such as 1024, becomes very difficult. Under the same reward design, even after running hundreds of epochs, the score converges early and is hard to improve upon during training.

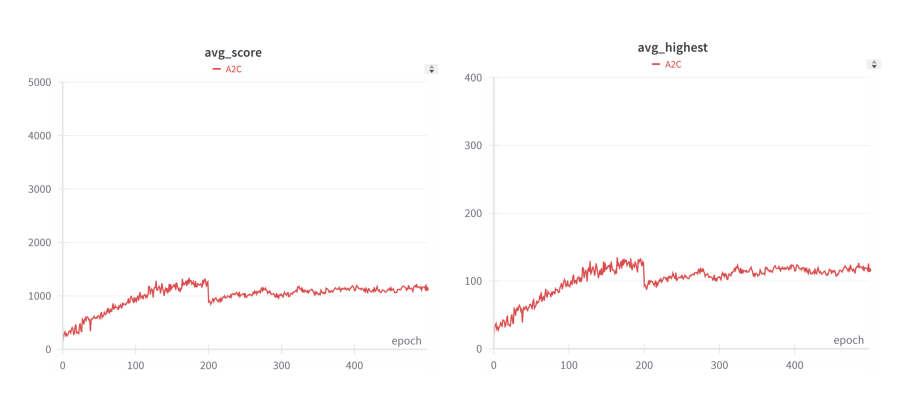

Figure 1: A2C Result

2 DQN

DQN is based on offline learning, meaning it learns the policy through historical experience. The experience replay mechanism helps break the correlation between data points and allows past experiences to be used repeatedly for training. For policy selection, a custom-designed CNN policy is used. The architecture consists of two convolutional layers that extract spatial features from the image, which are then flattened for input into a linear layer. The linear layer transforms the flattened features into a vector of the specified dimension, which serves as the input for the subsequent DQN decision process. This design of the CustomCNN feature extractor is efficient at processing image data, extracting multi-level local features through convolutional layers, and converting them into a lower-dimensional feature vector that suits the decision-making needs of reinforcement learning. Using CNNs allows for effective capture of spatial structural information in images, and through

flattening and combining with the linear layer, these features are further compressed into a fixed-dimensional vector, facilitating the subsequent Q-value estimation. As a result, DQN performs better than A2C, and it can achieve higher scores in fewer than 100 epochs compared to A2C.

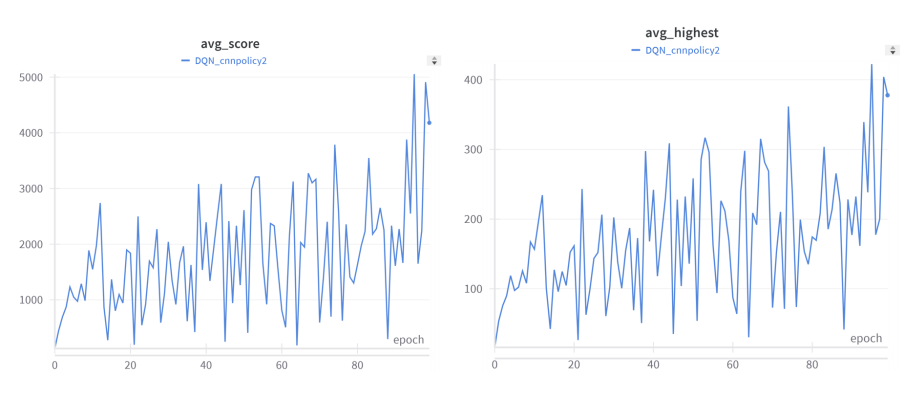

Figure 2: DQN Result

3 Conclusion

Therefore, I chose DQN as my best model for 2048 game.

Q2. Show your best tile distribution in 2048. (4 pts.)

My best result is to utilize DQN to train the agent.

Tile Distribution: $\{8 : 2, 64 : 3, 128 : 7, 256 : 26, 512 : 57, 1024 : 5\}$

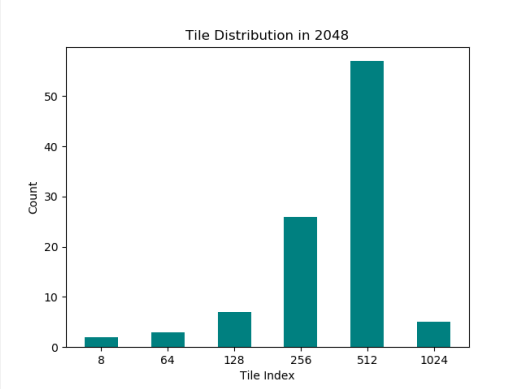

Figure 3: Tile Distribution in 2048

Hyperparameter	Value
Algorithm	DQN
Policy Network	CnnPolicy
Epoch Number	100
Timesteps per Epoch	100000
Evaluation Episode Number	20
Learning Rate	5×10^{-4}
Buffer Size	100000
Batch Size	32

Table 1: DQN Hyperparameter

Q3. Describe what you have done to train your best model. (10 pts.)

1 Set Penalty for Illegal Move

To encourage the agent to make valid moves, a penalty should be imposed for any illegal moves it attempts. The purpose of this penalty is to discourage actions that do not contribute to the intended progress or strategy of the game. However, it is important to calibrate this penalty carefully. If the

penalty is set too high, it may cause the overall loss function to increase excessively, which could impede the agent's learning process by disproportionately focusing on avoiding penalties rather than pursuing a successful strategy. Therefore, the penalty should be significant enough to discourage illegal moves without overwhelming the agent's learning objectives. In the experiment, I set -5 for penalty.

2 Provide chances to explore

To encourage the agent to explore other actions, a threshold is set to tolerate illegal moves. If the agent encounters an illegal move, the episode will not end immediately. Within this tolerance, the agent has the opportunity to try a different action and learn from the experience. This approach helps prevent the agent from repeatedly getting stuck in the same situation.

3 Set Weight Matrix

A weight matrix can be introduced to guide the agent towards prioritizing certain areas on the board. Specifically, the matrix can encourage the agent to sum numbers in the corners, where higher-value tiles tend to be more advantageous in many grid-based games. This approach can help the agent develop a strategic preference for corner-based accumulations, which may ultimately lead to more efficient tile merging and higher scores. The weight matrix should be designed to subtly reinforce corner placements without overly restricting the agent's decision-making freedom, allowing it to explore other effective strategies as well.

```
weight = np.array([
                [0, 0, 0, 1],
                [0, 0, 1, 2],
                [0, 1, 2, 3],
                [1, 2, 3, 4]  # Higher values towards the end
            ])
```

4 Set Reward for Higher Tile

To motivate the agent to achieve higher tile values, incremental rewards should be assigned as the agent reaches specific tile values. These rewards serve as positive reinforcement, encouraging the agent to develop strategies that maximize the board's tile values. For instance, when the agent achieves a 256 tile, an additional reward of +10 can be given. Similarly, reaching a 512 tile could yield a reward of +30, and achieving a 1024 tile could yield a reward of +50. This tiered reward system aligns with the agent's learning objectives, pushing it towards achieving high-value tiles while progressively learning optimal strategies.

```
if tile in self.Matrix:
    if tile == 256:
        reward += 10
    elif tile == 512:
        reward += 30
    elif tile == 1024:
        reward += 50
```

Q4. Choose an environment from the Gymnasium library and train an agent. Show your results or share anything you like. (10 pts.)

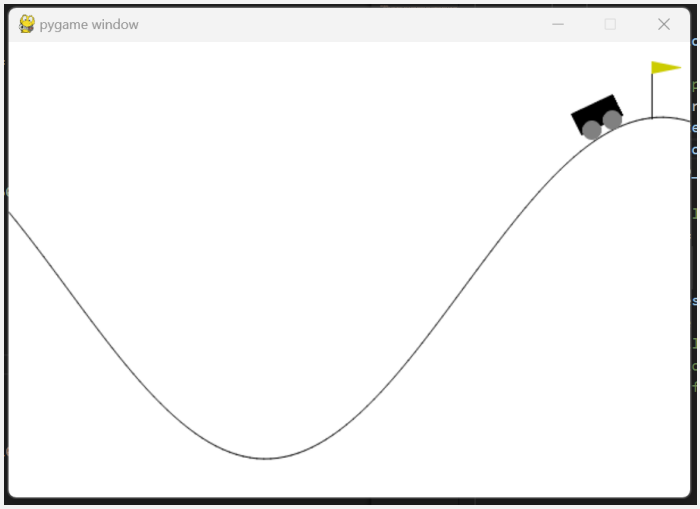

Figure 4: Mountain Car

I choose Mountain Car from Gymnasium. In the environment design, a reward mechanism is specially devised for the 'step' function to encourage the car to move steadily and efficiently toward the goal position, with additional rewards for specific behaviors. The car operates within a position range of -1.2 to 0.6, with a maximum speed limit of 0.07 to maintain stability. The base reward is calculated based on the car's position and velocity, using a cosine function to vary the score with position and adjusting it according to velocity, forming a fundamental score. As the car approaches the target position (0.5), it receives increasing rewards with closer proximity to the goal, reinforcing goal-oriented movement. Additionally, when the car accelerates rightward from the initial position and reaches a certain velocity, it receives further rewards to encourage positive velocity behavior. Finally, reaching the goal position marks task completion and terminates the episode. This reward function not only drives the car toward the goal but also emphasizes stability in controlling speed and position.

For the model selection, I have tried DQN, PPO, A2C for 100 epochs.

1 DQN

DQN stabilizes the training process by introducing Experience Replay and a Target Network. Experience Replay stores past state-action-reward-next state tuples, allowing the model to train without relying on the immediate sequence of states, which reduces correlations between samples. The Target Network prevents the target values from changing too frequently, further enhancing stability. In this environment, the reward mechanism requires the model to account for long-term rewards, such as gradually increasing the reward as the distance to the target position decreases. Although DQN is designed for discrete data, and experiences some oscillations when training on continuous data, it can still consistently converge to better results

Hyperparameter	Value
Algorithm	DQN
Policy Network	MlpPolicy
Epoch Number	100
Timesteps per Epoch	10000
Evaluation Episode Number	20
Learning Rate	5×10^{-3}

Table 2: DQN Hyperparameter

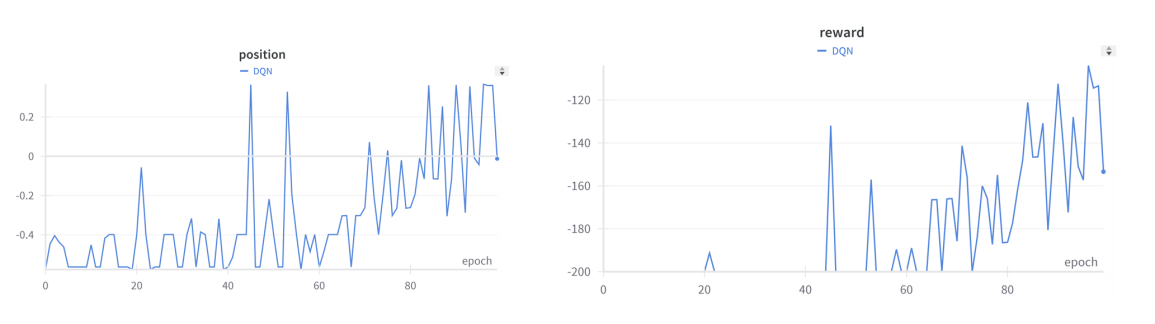

Figure 5: Mountain Car result (DQN)

2 PPO

Unlike DQN, PPO is better suited for handling continuous action spaces. In this environment, if the car's control is continuous, PPO can more effectively learn how to adjust these controls to achieve the goal, making it more appropriate for continuous decision-making problems than DQN. PPO also excels at optimizing long-term goals because it can learn complex strategies and achieve the final goal after multiple steps. This is particularly helpful for the reward design in this task, as the car needs to gradually approach the target position over several time steps. PPO limits the range of each update by clipping the objective function, preventing excessively large strategy updates and ensuring the stability of the policy, especially when updates are frequent. Compared to DQN, PPO demonstrates better stability during training. Although PPO converges more slowly than DQN, it is much more stable compared to DQN, which often exhibits significant fluctuations.

Hyperparameter	Value
Algorithm	PPO
Policy Network	MlpPolicy
Epoch Number	100
Timesteps per Epoch	10000
Evaluation Episode Number	20
Learning Rate	5×10^{-3}

Table 3: PPO Hyperparameter

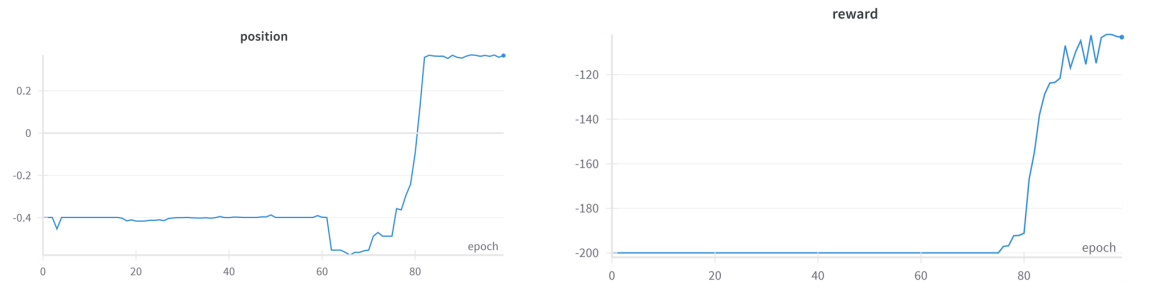

Figure 6: Mountain Car result (PPO)

3 A2C

A2C achieved the least favorable results. Under the same reward design, A2C clearly suffered from insufficient exploration, which led to an inability to break out of the initial state. Even after reducing the learning rate from 1e-3 to 1e-5, the model struggled to converge. This suggests that the default reward mechanism might not be sufficient to encourage adequate exploration, which is crucial for the agent to discover and exploit optimal strategies. To improve the performance of A2C, it may be necessary to design additional reward mechanisms or modify the exploration-exploitation balance to better support the agent in exploring new actions and overcoming the limitations imposed by the initial state.

Hyperparameter	**Value**
Algorithm	A2C
Policy Network	MlpPolicy
Epoch Number	100
Timesteps per Epoch	10000
Evaluation Episode Number	20
Learning Rate	5×10^{-5}

Table 4: A2C Hyperparameter

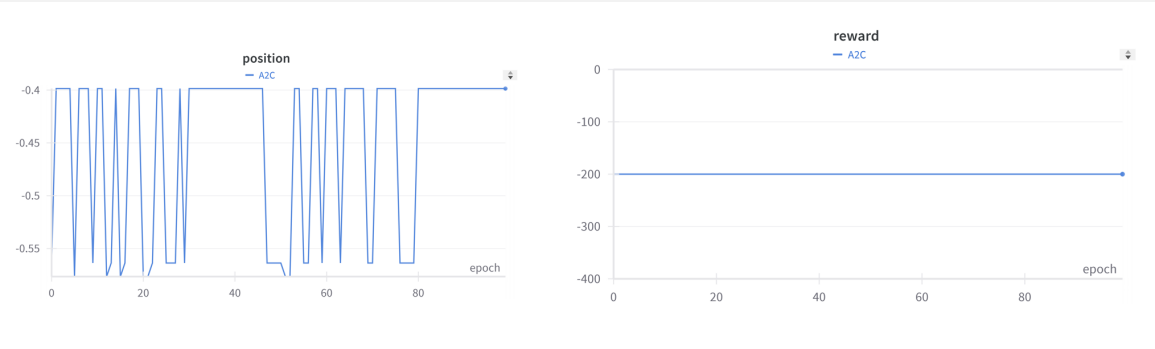

Figure 7: Mountain Car result (A2C)