Deep Reinforcement Learning HW2

109061225 許博翔

1. Model Architecture

I used six layers in this model, including three convolution layers, one max pool and two full connect layers. Since I use mac for training and need to avoid time out, so I use small model, which is about 1.4MB.

The reason I choice this architecture is based on the limitation of computation complexity and the rules, three convolution layers ensure it can process the images in relation with space. A max pool layers can reduce the complexity of the model, two full connect layers allow model to make decisions.

I had an old version, and its performance is better. However, it’s size is too large, and if I make the parameter size smaller, the performance will drop significantly. Finally, I use this model architecture.

2. Train Method

I had implemented DDQN and experience replay and epsilon-greedy exploration. We can separate the training process into several parts.

1. Initialization
    (a) The online and target networks are initialized with random weights.
    (b) The agent's memory is initialized as a deque with a maximum length of 200,000 experiences.
2. Exploration
    (a) The agent starts with an exploration rate of 1, which decreases over time using a decay factor of 0.99999975.
    (b) The minimum exploration rate is set to 0.1.
    (c) During exploration, the agent selects actions randomly with a

probability equal to the exploration rate.

3. Exploitation
    (a) When not exploring, the agent selects actions greedily based on the Q-values predicted by the online network.
4. Experience Replay
    (a) The agent stores each experience tuple (state, next_state, action, reward, done) in the memory.
    (b) During training, the agent samples a batch of experiences from the memory.
5. Training
    (a) The agent trains the online network using the sampled experiences.
    (b) The TD estimate is calculated using the Q-values predicted by the online network for the current state and action.
    (c) The TD target is calculated using the Q-values predicted by the target network for the next state and the maximum Q-value action.
    (d) The loss is computed as the smooth L1 loss between the TD estimate and TD target.
    (e) The online network's parameters are updated using the Adam optimizer to minimize the loss.
6. Target Network Synchronization:
    (a) Every “sync_every” steps, the target network's parameters are synchronized with the online network's parameters.
7. Model Saving:
    (a) The agent's model and exploration rate are saved every “save_every” steps.

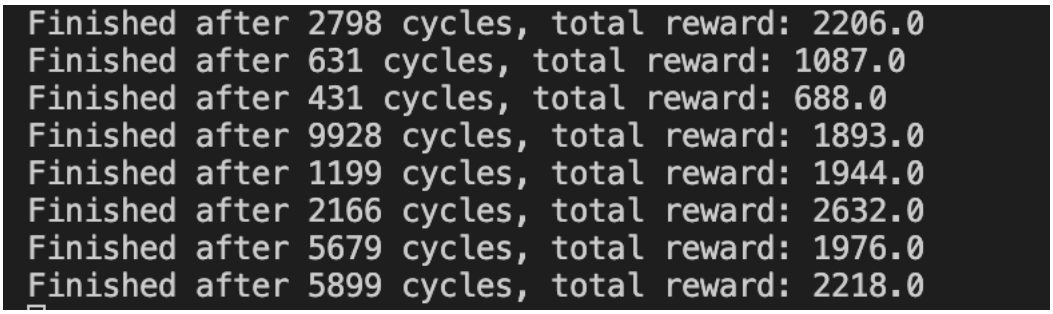
To sum up, the training process is repeated for a specified number of episodes. During each episode, the agent interacts with the environment, collects experiences, and learns from them using the DQN algorithm. The agent's performance is evaluated periodically by testing it in the environment without exploration.

The code also includes several wrappers for the environment, such as frame

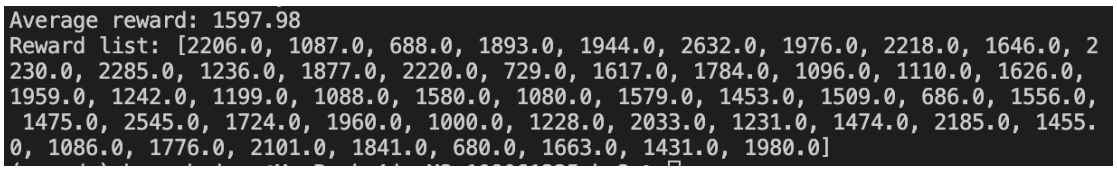
skipping, grayscale conversion, and observation resizing, to preprocess the game state before feeding it to the DQN.

3. Result

- Evaluation processing



- Result



4. Conclusion

Although I hadn’t gotten top grade in leaderboard, but I had learned some skill, including a lot of debugs, and model evaluation. At the same time, I found it very rewarding to watch my trained models go from not moving at all, to moving forward, to jumping over the mushrooms or tubes. I hope I can find a way to learn more and contribute to this field in the future.