Optimization and Reinforcement Learning

Final Project Report

N96134166 王星諭

**Problem overview**

This project’s goal is to develop a Deep Q-Network (DQN) agent that can play the space ship shooting game up to 10000 score. The agent will optimize its playing policy by using reinforcement learning techniques.

**Environment setup**

This project was developed using Python 3.9.21. Several essential packages were installed to support the game environment and deep learning training process, including:

* pygame: for implementing the space ship shooting game.
* sys: for system-level operations.
* torch (PyTorch): for constructing and training the DQN neural network.
* numpy: for numerical computations (version 1.23.5 is specifically used to prevent format compatibility issues).
* PIL.Image: for image processing.
* imageio: for saving gameplay recordings as video files (version 2.31.6 is used to avoid write errors).
* collections.deque: for efficiently managing the replay buffer in reinforcement learning.

In addition, I trained the model using an NVIDIA Quadro P2200 GPU.

**State/reward design**

In this project, the agent observes the environment through a stack of four continuous game frames, representing the current game state. It allows agent to get targets’ movement and direction, which can help agent for decision-making.

1. Score Reward: Since the goal is to reach a score of 10000, a reward is given whenever the score changes. To ensure this reward is emphasized over others, it is scaled by a factor of 2.
2. Health reward: Health reflects the player's ability to survive longer in the game. Therefore, a penalty is applied when health decreases. To account for the different scale between score and health, the health reward is scaled by a factor of 10, giving it moderate influence. This makes it more impactful during early stages of the game.
3. Death Penalty: Upon losing the game, the agent receives a fixed penalty of -200. This fixed deduction has a significant effect in early stages where scores are relatively low, encouraging the agent to avoid dying prematurely.
4. Milestone Bonus: An extra reward is provided when the agent reaches predefined score thresholds, reinforcing progress and helping break sparse reward problems.
5. Shooting Reward: After observing that the agent tends not to shoot frequently during training, a shooting reward is added to encourage active engagement. When a shot successfully hits a rock, the agent receives a fixed positive reward. This helps promote more aggressive and proactive behavior.
6. Survival Reward: A small positive reward is given for each frame the agent survives. This reward encourages the agent to stay alive longer, even in moments where it might not be scoring or engaging directly.
7. Final Bonus: A large final reward is planned if the agent reaches the target score of 10000. However, in the current training runs, this reward was never triggered, so its actual effect on training remains uncertain.

**Model architecture**

Throughout the development of this project, various model architectures were tested to improve training performance. These included adjustments such as changing the number of convolutional layers, and modifying kernel sizes. However, after multiple iterations, the most effective model remained the original DQN architecture provided by TA in the template.

This model is based on a Convolutional Neural Network (CNN). The input to the model is a stack of four RGB color images, allowing it to perceive temporal changes. The architecture processes the input through three convolutional layers, each followed by a ReLU activation function, which introduces non-linearity and helps the model learn complex patterns. The output of the final convolutional layer is then flattened into a 1D feature vector and passed through two fully connected layers to produce the final Q-values corresponding to each action.

This structure effectively extracts spatial and temporal features from the game screen and transforms them into actionable representations, enabling the agent to make informed decisions.

**Training process**

The training process follows the DQN reinforcement learning framework. At each time step, the agent observes the current state, selects an action based on an epsilon-greedy strategy, and interacts with the game environment. The agent starts with epsilon = 1.0 and gradually decays it to epsilon = 0.01. Enable a balance between exploration and exploitation.

The data we got from the interactions is stored in a replay buffer, which is implemented using a deque structure to store and randomly sample experience tuples. The agent periodically samples a batch of experiences from the replay buffer to update the Q-network using gradient descent. Each training step minimizes the MSE loss between the predicted Q-values and the target Q-values, which are computed using the Bellman equation. A separate target network would update periodically to improve convergence and reduce instability in target Q-value estimation. The training continues until the agent get 10000 score or reaches a predefined number of episodes.

**Result presentation (Best Record)**

After experimenting with various modifications to the DQN model architecture, replay buffer structure, and reward mechanism, the best training performance was achieved using the original DQN architecture and replay buffer structure provided by the TA. The only change I have made was the reward mechanism.

The best training result was obtained after 2 days and 10 hours of continuous training. The model reached its highest score of 7244 points at episode 3484. This demonstrates that the combination of the original stable model structure with my reward design can effectively guide the agent toward high-level gameplay, even if the ultimate goal of 10,000 points was not yet achieved.

一張含有 文字, 繪圖, 螢幕擷取畫面, 圖表 的圖片

AI 產生的內容可能不正確。

To further improve performance, I believe the reward mechanism can be refined. In the result video, the agent rarely moves to actively collect falling power items. Introducing or strengthening a reward for picking up these items during training may help the agent achieve a higher score.

Another possible enhancement is to increase the number of training episodes. However, this change would significantly lengthen the total training time. Due to time constraints, I chose not to further explore the impact of longer training or more sophisticated reward schemes on overall performance.