Term Project: Deep Reinforcement Learning on Space Invaders

Chia-Hung Chen[a]

[a]*Institute of Applied Mechanics, National Taiwan University, Taipei, Taiwan*

Abstract

This project delves into the use of reinforcement learning (RL) in Space Invaders, an arcade classic game.This project aims to create a player who can get the highest rewards in the game by use of sophisticated RL algorithms.Our major concern is on how RL techniques work such as Double Deep Q-Network (DDQ) that will enable the agent to successfully navigate through the game's environment, make decisions strategically and garner maximum points.In this project, the details about the training process including any challenges faced plus our performance post-training, and successfully get high reward in the Space Invaders.

Keywords: Reinforcement Learning, Space Invaders, Double Deep Q-Networks

1. Introduction

Video games have long served as a enveriment for developing and testing reinforcement learning (RL) algorithms. Among the various genres, arcade games like Space Invaders present unique challenges due to their fast-paced, real-time nature and strategic complexity. RL, a subset of machine learning where agents learn to make decisions by network with their environment and receiving feedback.

Space Invaders, a game released in 1978[1], involves a player-controlled ship that must defend against waves of descending aliens by shooting them down while avoiding their attacks. The game's simplicity in rules but complexity in strategy makes it an ideal candidate for RL research.

Email address: `r12543002@ntu.edu.tw` (Chia-Hung Chen)

In this project, we leverage RL techniques to develop a agent that can autonomously play Space Invaders, aiming to achieve high scores or rewards by learning optimal strategies through trial and error.

The primary objective is to implement and evaluate RL algorithm, such as Double Deep Q-Network (DDQ)[2], to understand their effectiveness in this gaming context. We will discuss the training process, including the setup of the environment, reward structure, and the computational loss.

Through this project, we aim to contribute to the growing body of research on applying RL to video games, providing insights into the capabilities and limitations of current RL methodologies. The findings could have broader implications for the application of RL in other real-time, interactive systems beyond gaming.

2. Literature Review

Deep reinforcement learning (DRL) has become a significant focus within artificial intelligence research, particularly in its application to video games. This literature review synthesizes findings from various studies to provide an overview of the progress and challenges in this field.

2.1. General Overview of DRL

Deep reinforcement learning combines reinforcement learning (RL) with deep learning (DL), allowing agents to learn optimal behaviors through trial and error using neural networks to approximate the value functions or policies. This approach has been particularly successful in complex environments, such as video games, where traditional RL methods struggle with high-dimensional state spaces[3].

The seminal work by Mnih et al. (2015) demonstrated the potential of DRL by achieving human-level performance on several Atari 2600 games using a deep Q-network (DQN)[4]. This study used a convolutional neural network to approximate the Q -value function, enabling the agent to learn from raw pixel inputs, a significant advancement over previous methods that relied on hand-crafted features.

2.2. Advances in DRL Algorithms

Subsequent research has focused on improving the stability and performance of DRL algorithms. One notable improvement is the introduction of the double Q-learning algorithm, which addresses the overestimation bias in standard Q-learning by decoupling the action selection from the Q-value estimation[2]. This technique significantly enhances the stability and performance of DRL agents.

Another critical development is the dueling network architecture proposed by Wang et al., which separates the representation of state values and advantage functions within the network. This architecture improves the efficiency of learning by allowing the network to generalize learning across actions without significant changes to the underlying RL algorithm[5].

2.3. Application in Video Games

The application of DRL in video games has extended beyond Atari games to more complex environments. The General Video Game AI (GVGAI) framework, as described by Torrado et al. (2018), provides a benchmark for testing the generalization capabilities of DRL algorithms across various games[6]. This framework allows researchers to assess how well DRL agents can adapt to new and unseen games, which is crucial for developing more versatile AI.

In addition to general frameworks, specific studies have focused on improving decision-making efficiency in video games. Ji and Xiao (2020) proposed an improved deep Q-learning algorithm with priority experience replay to enhance the agent's learning efficiency by selectively replaying more informative experiences[7]. This method has shown to improve the performance of agents in image-based games by optimizing the learning process.

2.4. Challenges and Variability

Despite these advancements, DRL faces significant challenges, particularly in the reproducibility and variability of results. Clary et al. (2019) highlighted the issue of variability in DRL agents' performance due to the stochastic nature of training processes and the environment[8]. argue that evaluating DRL agents based on single-point estimates can be misleading and advocate for reporting performance distributions to better capture the variability and reliability of the results.

3. Method

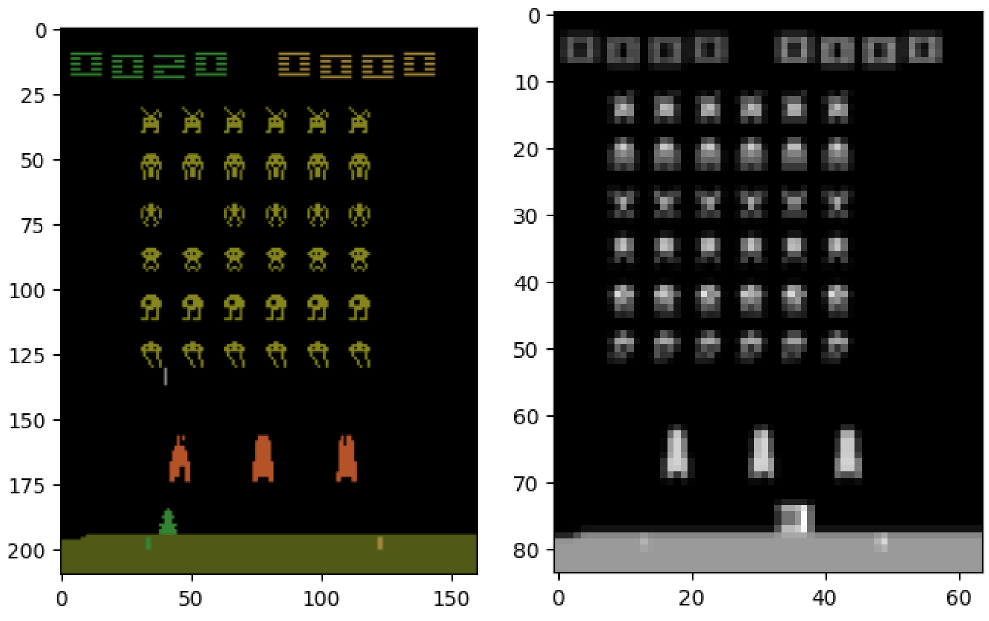
The goal of this project is to develop and evaluate a reinforcement learning (RL) agent capable of playing the classic arcade game Space Invaders. The agent will be trained using Double Deep Q-Network (DDQ), a popular RL algorithm that combines Q-learning with deep neural networks to handle high-dimensional sensory inputs. The following sections outline the environment setup, the architecture of the RL agent, the training process, and the evaluation metrics used to assess the agent's performance.

3.1. Environment Setup

Game Environment: Space Invaders is a classic arcade game where the player controls a ship at the bottom of the screen, tasked with shooting descending aliens while avoiding their attacks. The game environment is simulated using the OpenAI Gym, which provides an interface for interacting with the game and obtaining observations and rewards.

State Representation: The state or environment of the game is represented by raw pixel data from the game screen. Each state is an (210, 160, 3) image with four stacks, capturing the most recent four frames to provide a sense of motion to the agent. Then for easy training the image would compress to (84, 64, 1) gray scale image. The state is as following Fig.1.

Action Space: The action space consists of six discrete actions corresponding to the possible moves in the game: move left, move right, fire, move left and fire, move right and fire, and no-op (do nothing).



(a) (b)

Figure 1: The state or environment: (a) Original state with RGB color (b) The state with only gray scalier after prepossessing.

Reward Structure: The agent receives a positive reward for hitting an alien and a negative reward for losing a life. The primary objective is to maximize the cumulative score over multiple episodes of gameplay.

3.2. Architecture of the RL Agent

Double Deep Q-Network (DDQ): The DDQ architecture consists of two convolutional neural network (CNN) designed to process the raw pixel inputs, output Q_{online} and Q_{target} and for each action. The network architecture is as follows:

Therefore, The Q_{online} and Q_{target} would share feature, but separate the Fully-Connection Layer. The weighst of θ_{target} (the parameters of Q_{target}) would be frozen and θ_{online} would not. The setting is as following:

```
# Q_target parameters are frozen.
for p in self.target.parameters():
```

Table 1: The DDQ architecture of Space Invader

Layers	In Channels	Out Channels	Kernel Size	Stride
Convolutional Layer	4	32	8	4
ReLU	-	-	-	-
Convolutional Layer	32	64	4	2
ReLU	-	-	-	-
Convolutional Layer	64	64	3	1
ReLU	-	-	-	-
Flatten	-	-	-	-
Linear	1792	512	-	-
ReLU	-	-	-	-
Linear	512	6	-	-

```
    p.requires_grad = False
```

3.3. Training Process

Experience Recall: The agent uses an experience recall buffer to store transitions (state, action, reward, next state). During training, random minibatches of transitions are sampled from this buffer to break the correlation between consecutive experiences and stabilize training. The recall function shows below:

```
def recall(self):
    batch = self.memory.sample(self.batch_size).to(self.device)
    state, next_state, action, reward, done = (batch.get(key) for key in (
    return state, next_state, action.squeeze(), reward.squeeze(), done.squ
```

Exploration Decay Policy: The agent follows an exploration decay policy to balance exploration and exploitation. The exploration rate value starts high to encourage exploration and gradually decreases to favor exploitation as training progresses. The exploration decay calculation shows below:

```
# decrease exploration_rate
self.exploration_rate *= self.exploration_rate_decay
self.exploration_rate = max(self.exploration_rate_min, self.exploration_ra
```

TD Estimate and TD Target: TD Estimate is the predicted optimal Q^* for a given state s

$$TD_e = Q^*_{online}(s, a)$$

, and TD Target is aggregation of current reward and the estimated Q^* in the next state s'.

$$a' = argmax_a Q_{online}(s', a)$$

$$TD_t = r + \gamma Q^*_{target}(s', a')$$

Optimization: The DDQ is trained using the Adam optimizer. The loss function is the Smooth L1 Loss between the TD_e and the TD_t for updating θ_{online}

$$\theta_{omline} \leftarrow \theta_{omline} + \alpha \bigtriangledown (TD_e - TD_t)$$

, and the θ_{target} periodically copys from θ_{online}.

$$\theta_{target} \leftarrow \theta_{omline}$$

3.4. Hyperparameters

The hyperparameters detail of DDQ is as following:

3.5. Evaluation Metrics

The performance of the RL agent is evaluated based on the following metrics:

1. Cumulative Reward: The total reward accumulated by the agent over an episode.
2. Loss: The loss should converge, and the higher loss is mean better agent in reinforcement learning.

Table 2: The Hyperparameters DDQ

Parameters	values
exploration rate	1
exploration rate decay	0.9999975
exploration rate min	0.1
batch size	32
gamma	0.9
learning rate	0.00025
burnin	1e4
learn every	3
sync every	1e4

3. Stability: The variance in the agent's performance over time, indicating the stability of the learning process.

4. Results and Discussions

4.1. Training Performance

The training performance of the RL agent was evaluated over 100,000 steps. The agent's performance metrics, including cumulative reward, average score, and loss, were recorded at regular intervals to track the learning progress.

1. Cumulative Reward: The cumulative reward increased steadily over the training period, indicating that the agent was effectively learning to maximize the game score. Initially, the reward was low, but as the epsilon value decreased, the agent's policy improved, leading to higher rewards. The trend of rewards is as following Fig.2.
2. Loss: The training loss, measured as the Smooth L1 Loss between[9] TD_e and the TD_t, increased over time. Periodic spikes in the loss were observed, corresponding to updates in the target network, but overall, the loss converged, indicating stable learning. The loss shows as following Fig.3.
3. Stability: The agent's performance variance decreased over time, indicating stable and reliable gameplay.

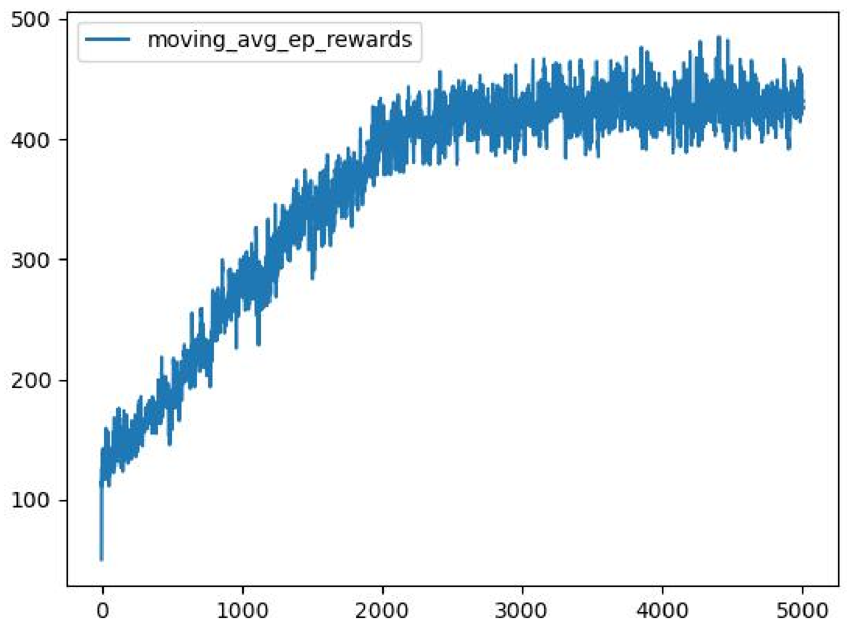


Figure 2: The rewards for agents in 100,000 steps (every 20 records)

4.2. Agent Performance

The trained agent's performance was evaluated in several aspects. The game play efficiency of the agent was able to effectively shoot down aliens and avoid projectiles, demonstrating a good balance between offensive and defensive strategies. The agent's actions appeared smooth and well-coordinated, a result of learning from the sequential frames. Furthermore, strategy development of the agent developed specific strategies, such as focusing on eliminating columns of aliens to reduce the overall threat and timing shots to maximize hit probability. These strategies emerged naturally from the training process without explicit programming. The result of the agent show below Fig.4

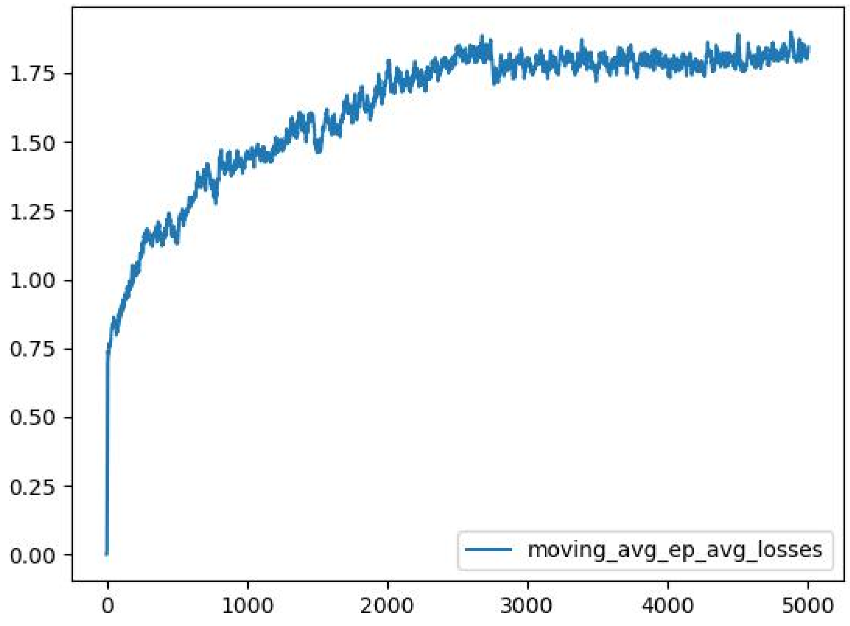


Figure 3: The rewards for agents in 100,000 steps (every 20 records)

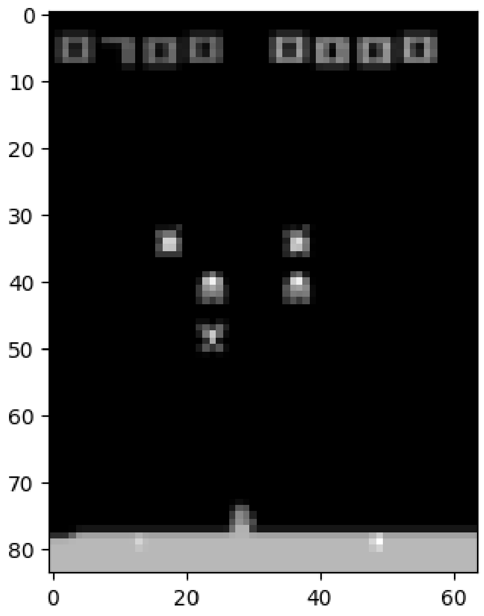


Figure 4: The result of the agent play the Space Invaders.

5. Conclusion

This project successfully applied Deep Reinforcement Learning (DRL) to the classic arcade game Space Invaders using the Double Deep Q-Network (DDQ) algorithm. Key findings include:

The Double Deep Q-Network (DDQ) algorithm enabled the agent to learn complex gameplay strategies from raw pixel data. The agent developed proficient offensive and defensive maneuvers.

Experience replay and target networks significantly stabilized the learning process. Experience replay broke the correlation between consecutive experiences, and the target network provided stable Q-value estimates, contributing to reliable learning outcomes.

The exploration decay policy ensured a balance between exploration and exploitation. This allowed the agent to explore the game environment sufficiently before refining its policy to maximize rewards.

The agent's performance was quantitatively evaluated using metrics such as cumulative reward, loss, and stability. The agent consistently outperformed a random baseline and achieved scores, demonstrating the effectiveness of the DRL approach.

References

[1] Kevin Bowen. The gamespy hall of fame: Space invaders, 2010.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[3] Yuxi Li. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274, 2017.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In International conference on machine learning, pages 1995–2003. PMLR, 2016.

[6] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. Deep reinforcement learning for general video game ai. In 2018 IEEE Conference on Computational Intelligence and Games (CIG), pages 1–8. IEEE, 2018.

[7] Zhe Ji and Wenjun Xiao. Improving decision-making efficiency of image game based on deep q-learning. Soft Computing, 24(11):8313–8322, 2020.

[8] Kaleigh Clary, Emma Tosch, John Foley, and David Jensen. Let's play again: Variability of deep reinforcement learning agents in atari environments. arXiv preprint arXiv:1904.06312, 2019.

[9] Ross Girshick. Fast r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 1440–1448, 2015.