AI Capstone Project 2 Report

111550108 吳佳諭

1. Video clip links

Space Invader: https://youtu.be/gyPCVXwP77E

Cart Pole: https://youtu.be/iHkwjJA5J7o

Blackjack: https://youtu.be/2Sp5grFCTRY

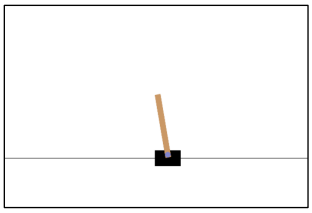
2. Introduction

In this project, I choose three environments to play with reinforcement learning. The first environment is Space Invader from Atari. The second environment is Cart Pole from Classic Control. The last environment is Blackjack from Toy Text.

Space Invader (Atari)

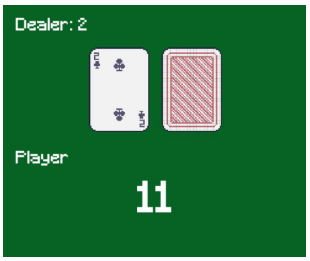
In Space Invader, we can control a laser cannon to move left and right or on fire. There are space invaders in the sky, and the objective is to destroy them by shooting at them before they reach the Earth. The game ends when three of my lives are lost after taking enemy fire, or when the enemy reach the Earth.

Cart Pole (Classic Control)



In this environment, a pole is placed upright on the cart. We can add force to the cart in the left and right direction. The goal is to balance the pole to make it stand on the cart as long as possible. The game ends if the pole angle is more than ±12°, or the cart reach the edge of the display, or the episode length is greater than 500.

Blackjack (Toy Text)

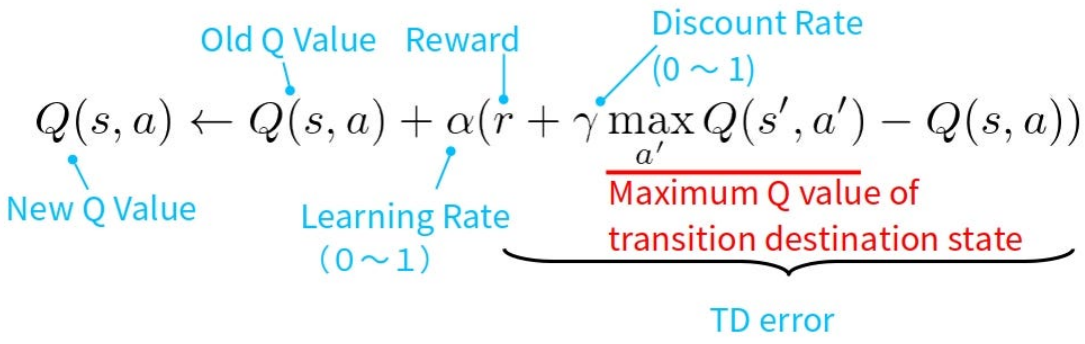


Blackjack is a popular card game played between player and dealer, with a goal of having a hand value as close to 21 as possible without exceeding it. Both the player and the dealer get two cards at the beginning. The player can request additional cards (hit) until they decide to stop (stick) or exceed 21, then the dealer draws card. Anyone having cards exceed 21 loss the game. If no one exceeds 21, then the one with a larger sum win.

3. Implementation

I used Q learning to train my agent. Q learning is a well-known reinforcement learning algorithm that aims to find the optimal action for an agent in a given environment. The Q value function estimates the future rewards for state-action pairs. The agent updates its Q values based on the reward received after performing an action and transitioning to a new state. The goal is to maximize the total reward over time by choosing actions that lead to the highest Q values. Through repeated interactions with the environment, Q learning converges to the optimal policy without requiring a model of the environment.

(1) Q learning



The Q value function and Q learning (source: https://www.tcom242242.net)

Here are some hyperparameters I used for different tasks in this project.

Task	Learning rate	Gamma	Epsilon	Episode
Space Invader	0.1	0.9	1.0 ~ 0.1	3000
Cart Pole	0.5	0.97	0.05	3000
Blackjack	0.01	0.95	0.1 ~ 1.0	100000

When I train the agent, I use **epsilon decay** to modify epsilon over time. At the beginning of the training, we would like the agent to take more random actions, as the Q table is still incomplete and lacks sufficient information. Therefore, I set start epsilon to 1.0, which means that the agent will always take random action. And epsilon will decrease every episode by $epsilon = \max(end\ epsilon, epsilon - (\frac{start\ epsilon}{episode}))$, where start epsilon = 1.0 and end epsilon = 0.1.

In Cart Pole, I **discretize the continuous state** because Q table can only record discrete state. I split the continuous state values range into 7 discrete indexes, and map the received state values into the 7 indexes. This way, we can use Q learning even if the state is continuous.

(2) DQN

I also use DQN to train the agent. It is similar to Q learning, but its Q table is maintained by the neural network.

Here are my network architecture used in this project.

Space Invader	`Model(`
	`  (conv1): Conv2d(3, 32, kernel_size=(8, 8), stride=(4, 4))`
	`  (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))`
	`  (conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))`
	`  (flatten): Flatten(start_dim=1, end_dim=-1)`
	`  (fc1): Linear(in_features=22528, out_features=512, bias=True)`
	`  (fc2): Linear(in_features=512, out_features=256, bias=True)`
	`  (fc3): Linear(in_features=256, out_features=6, bias=True)`
	`)`

Cart Pole	`Net(`
	`  (fc1): Linear(in_features=4, out_features=32, bias=True)`
	`  (fc2): Linear(in_features=32, out_features=50, bias=True)`
	`  (fc3): Linear(in_features=50, out_features=2, bias=True)`
	`)`

(3) Stack frame

In Space Invader, the state is represented by a 210*160 rgb image, which is a frame in the game. By stacking multiple consecutive frames, the agent can capture the temporal aspect of the environment and gain a sense of motion or movement. This allows the agent to make better decisions by understanding how the environment is changing over time, rather than relying on a single frame that may not provide enough context for motion or actions that unfold over multiple timesteps. Hence, I use DQN and frame stacking to train the agent in Space Invader environment.

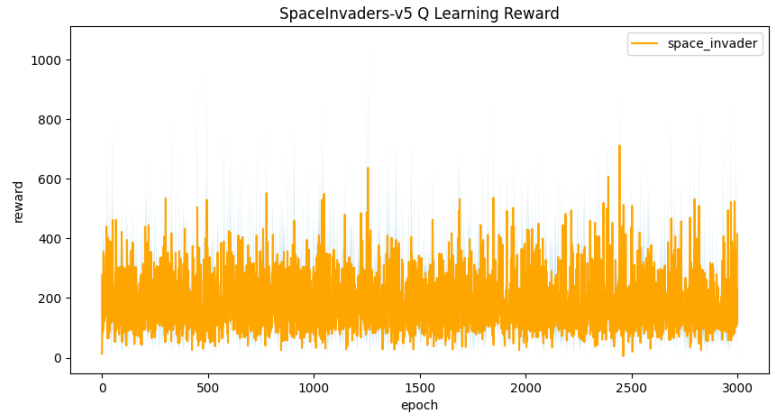
4. Result

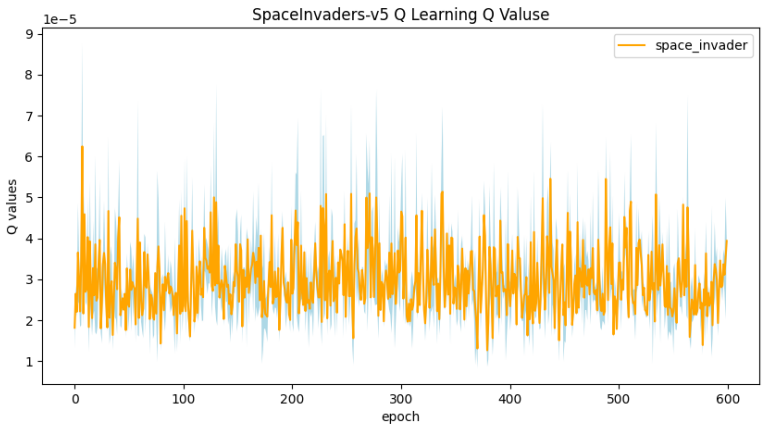
(1) Space Invader

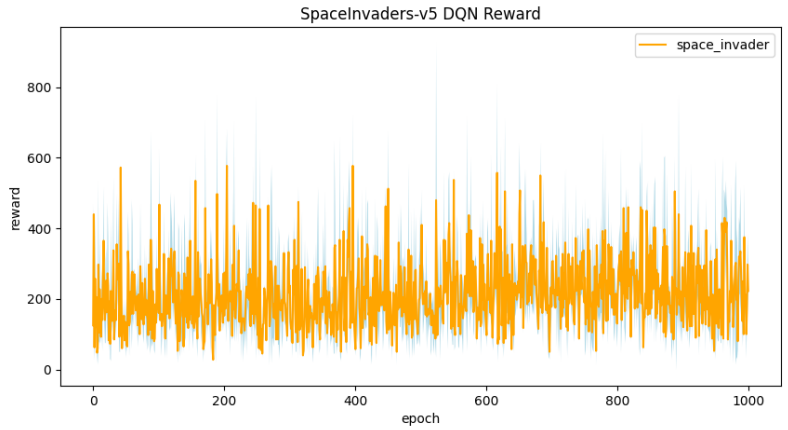
For Space Invader environment, I used Q learning, DQN, DQN + stack frame, total 3 kinds of way to train the agent. And taking random action as baseline.

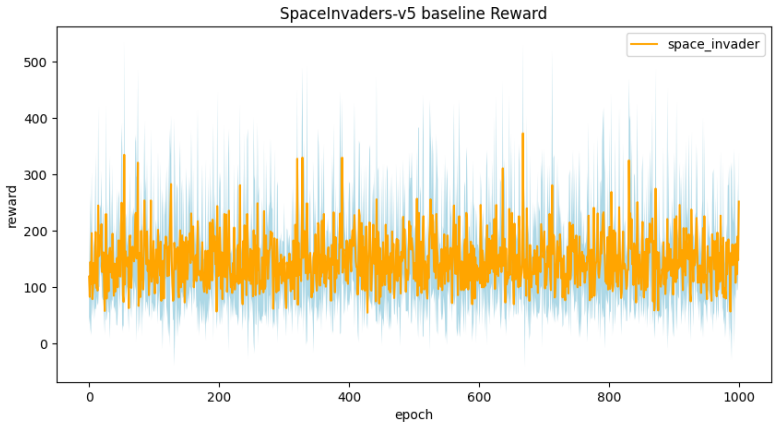
Figures in the next page show the reward and Q value over time. By taking random action, the average is only about 150, but after training, the average can reach 285.

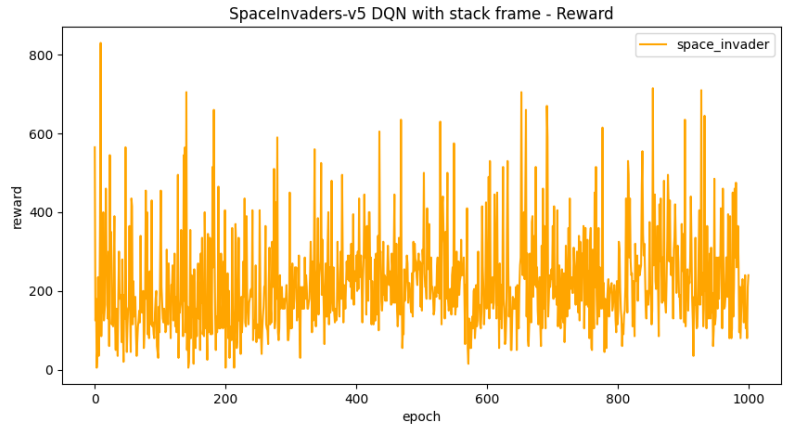
For Q learning and DQN. Q learning can only get 170 rewards, while DQN can get 285 rewards, showing that using neural network to do Q learning indeed improve the performance of the agent.

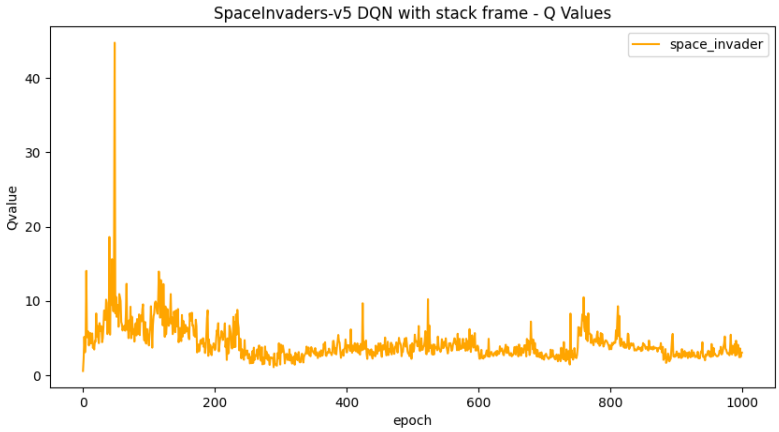












a	b
c	d
e	f

Fig. 1 Reward and Q value of different algorithm. (Space Invader)

a. Reward of Q learning
b. Q value of Q learning
c. Reward of DQN
d. Reward of baseline (random action)
e. Reward of DQN + stack frame
f. Q value of DQN + stack frame

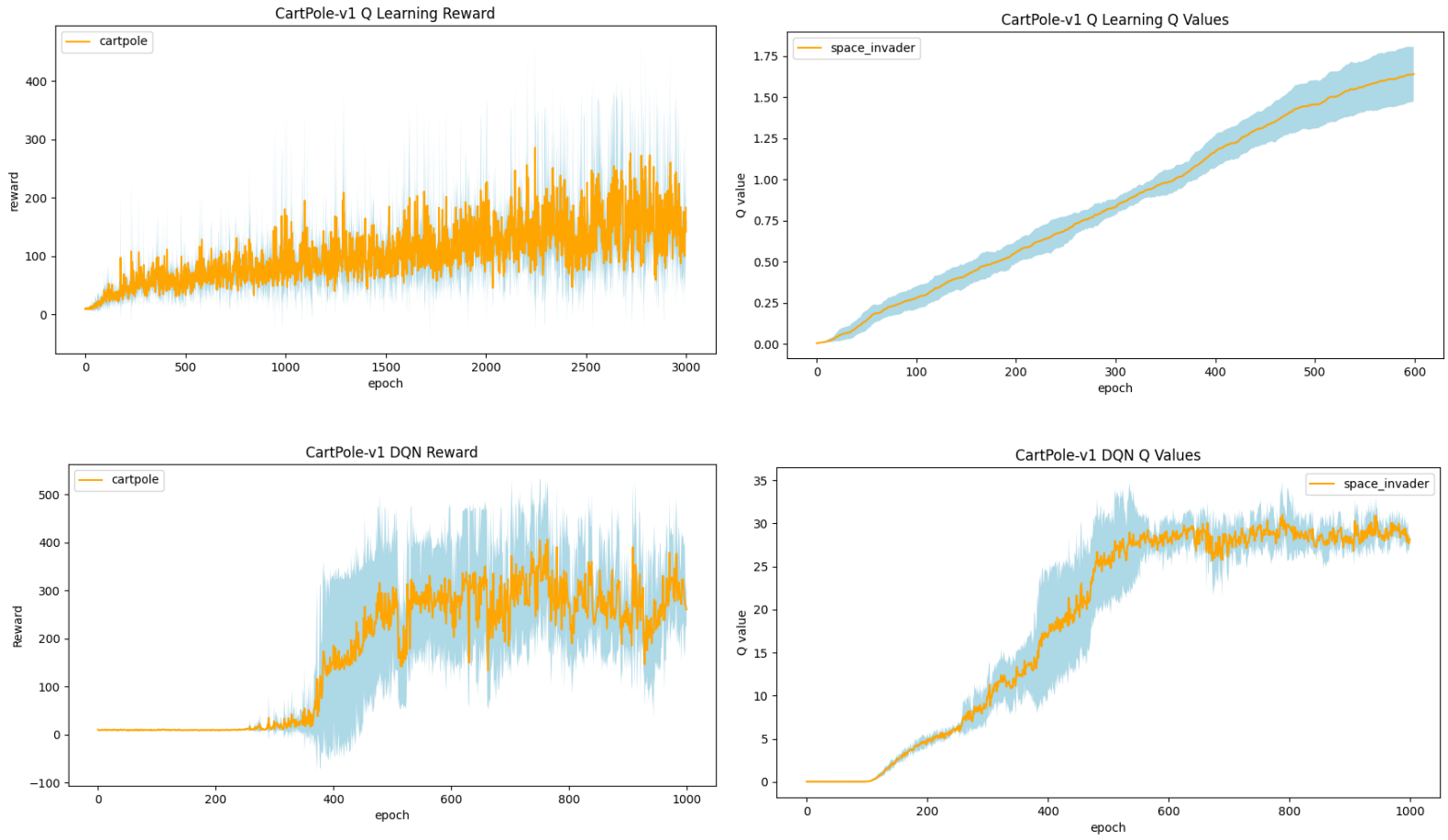
(*baseline, Q learning, and DQN without stack frame are run multiple times and plot the figure based on the average value. The blue area is the stander deviation)

Algorithm	Average rewards
Baseline (random)	150
Q learning	170
DQN	285
DQN + stack frame	285

For DQN with and without stack frame. I thought adding the stack frame technic can improve the performance and get more rewards, since it helps the agent to capture the motion. However, it turned out that even using stacking frame, the final rewards don't increase. But we can still observe some interesting things from the above rewards figures. The maximum reward from DQN without stack frame is about 600, and most of the reward is under 400. As for DQN with stack frame, many reward values are over 400, and the maximum reward is over 800. This shows that stack frame indeed helps the training and help the agent get higher reward.

(2) Cart Pole

For Cart Pole, I use Q learning and DQN to train the agent. Here are the reward and Q value figures. The Q value figures on the righthand side show that the Q table update and increase the Q value as the game (episode) progress.



a	b
c	d

Fig. 2 Reward and Q value of different algorithm. (Cart Pole)

a. Reward of Q learning
b. Q value of Q learning
c. Reward of DQN
d. Q value of DQN

Algorithm	Average rewards
Q learning	96
DQN	312

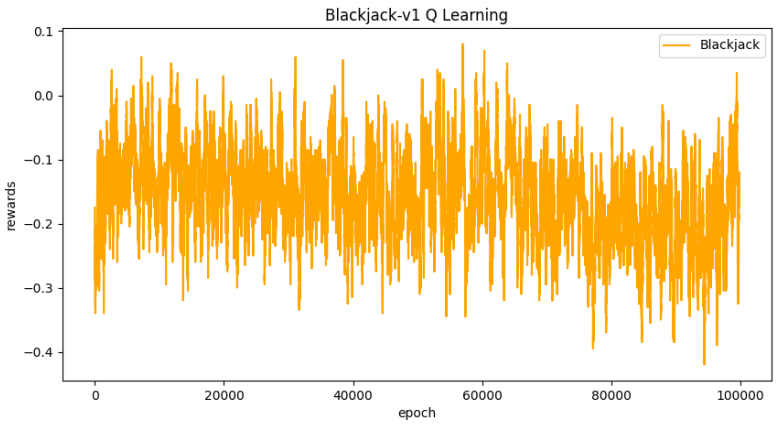
It is clear to see that using DQN can get higher reward. It is because in Cart Pole environment, the state values are continuous (cart position, cart velocity, pole angle, pole angular velocity). Q learning use a Q table to record the Q value after taking the action. Although we can discretize the state value to map the continuous state into discrete index, we will lose some information during this transformation. On the other hand, DQN use a neural network to update the Q value, which can better handle the continuous values, and therefore give a better performance. That is why DQN get a higher reward than Q learning.

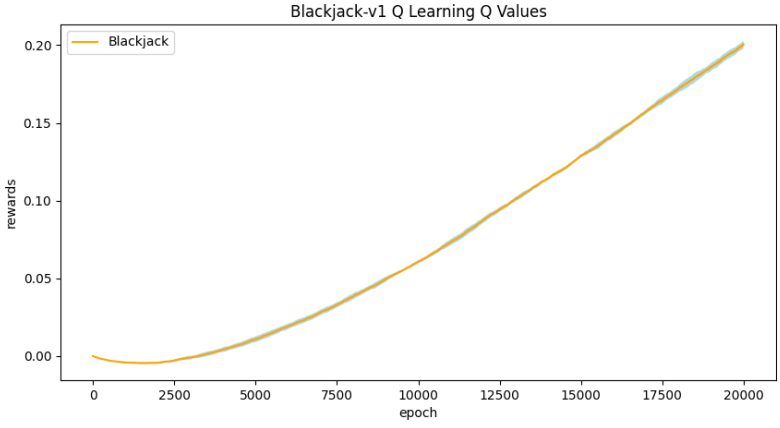
(3) Blackjack

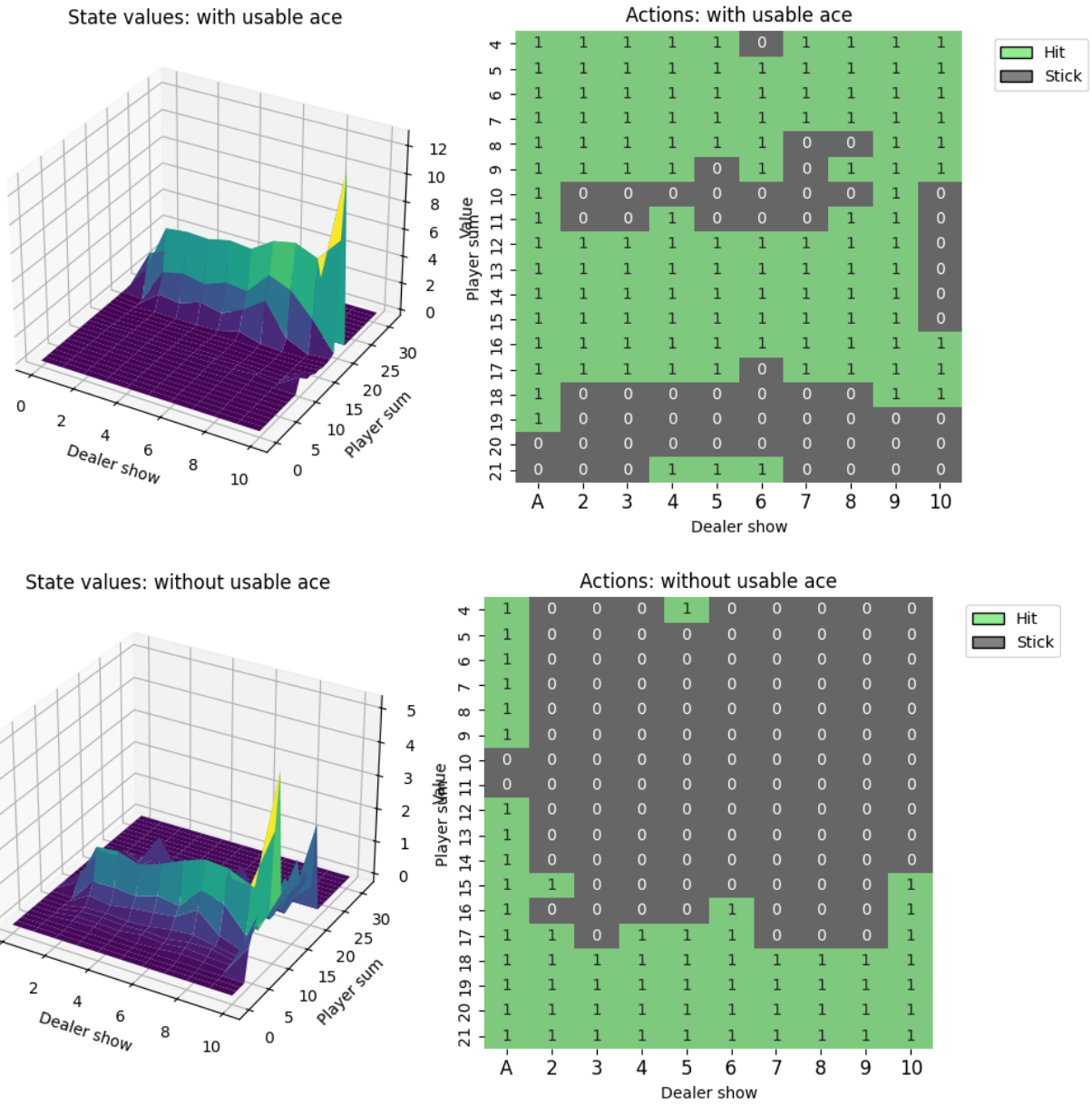
Blackjack environment only contains discrete action space and observation space, so I only do Q learning. The following first two figures show the reward and Q value over time. We can also see that the Q value increase as the game progress.

At first, I set the episode length to 3000, but the training result is bad. No matter how I adjust the learning rate or gamma, the performance just cannot improve. Later I realized that the game is too short. In other words, the agent can only update the Q table one or two times in an episode, which is not enough for the agent to learn the optimal action selection. After increase the episode length, the agent can have a better performance.

The observation space of Blackjack has three dimensions, player current sum, dealer showing card value, and usable Ace. And the action space is 0 or 1 meaning stick (player stop) or hit (player takes a card). The following four figures shows the Q values and the action taken by the agent in different player sum and dealer card when the player has or has no usable Ace. From the action figures on the right, we can see that the agent tends to take cards when there are usable Ace, and stop when there are no usable Ace.







5. Discussion

(1) Reinforcement learning and Space Invader

In this project, I learned reinforcement learning and gym environment. During the class, I thought I understand the concept of reinforcement learning, but when I start to implement it, I found that it is really hard to train a good agent. The first task I did is Space Invader, which is also the most difficult task in this project in my opinion. I tried different hyperparameters, different network structures, adding little tricks like epsilon decay, learning decay, or stack frame to improve the performance of the agent, but none of them give a significant improve.

I also tried to define the reward by myself. Sometimes when I visualize the game, the agent does “no operation” for the whole episode, or stand at a fix position to on fire and never move. As a result, I tried to specify the reward by myself, trying to give the agent some penalty if it never moves or keeps shooting even though there are no enemy on top of the agent. I want the agent to move more often to avoid the fire from the enemy, just like what we will do when we play the game. However, when I trained the agent with the self-defined rewards, the agent even performs worse.

Even if the performance is not as my expectation, I still gain the experiment of training a reinforcement agent. I read many articles and codes, trying to find some ways to improve the training. Although these ways do not help the training very much, they give me a deeper understanding of reinforcement learning. I also know more about how to interpret the training result, like plotting the reward, Q values, and the action the agent takes. And rendering the game is also fun. Looking at the agent to play the game by itself and win is really cool. (although it is also upset when seeing the agent choose a bad action and lose the game XDD)

(2) Gymnasium environment

I also learn gymnasium environment. It is really useful for use to play with reinforcement learning. There are so many tasks and games in the environments, from easy to hard. There are not only classic control tasks like Cart Pole, there are also many interesting games, like those in Atari environment. I actually spend sometimes to decide which tasks to do in this project because all of them looks fun.

(3) Box 2D

At first, I want to choose Bipedal Walker in Box2D environment as one of the tasks in this project. But when I finish the code and want to run it, I found that I cannot install the Box2D environment. I tried to install many different packages and wheels, or install it using different commands. I even tried to use different version of python and gym, but I just cannot install it. After trying to install it for over six hours and all the tries end up with error, I give up in the end and choose other tasks. I am really confused about why I cannot install it. Maybe next time when I have more time, I can try to install it again and try the Box2D environment.

6. Reference

My code: https://github.com/chia-yuu/AI-Capstone/tree/main

Space Invader: https://ale.farama.org/environments/space_invaders/

Cart Pole: https://gymnasium.farama.org/environments/classic_control/cart_pole/

Blackjack: https://gymnasium.farama.org/environments/toy_text/blackjack/

Stack frame 1: https://github.com/simoninithomas/Deep_reinforcement_learning_Course

Stack frame 2: https://github.com/andychang0207/RL_SpaceInvader

Stack frame 3: https://danieltakeshi.github.io/2016/11/25

Space Invader network architecture: https://github.com/nicknochnack