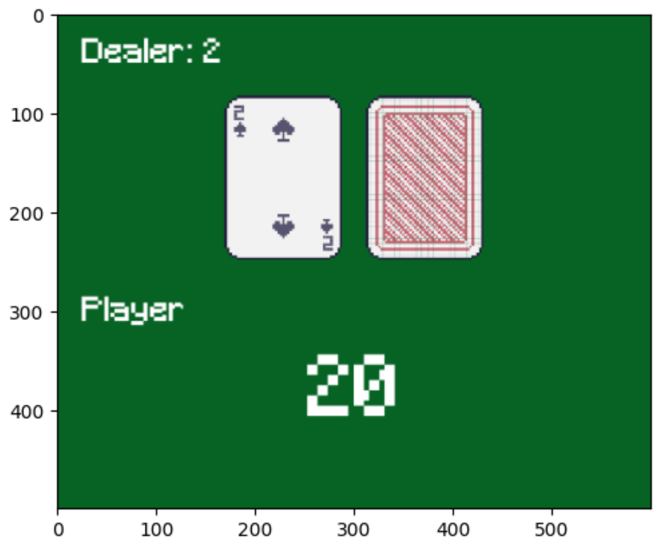
<u>COMP532 Assignment 2</u>

Law Kok Wing (201629894)

Problem 1:

<u>Introduction</u>



The gymnasium game I choose is Blackjack from the Toy Text environment. The basic idea of the game is to draw cards as close to 21 points without exceeding. Player wins if his sum of card's point is closer to 21 than the dealer, vice versa the player loses. It would be a draw if the player and the dealer get the same sum of card's point. The game would take 2 actions as input, either 0 (stick) or 1 (hit), and return 3 tuples including the player current sum, value of dealer showing card and usable ace. The reward would return +1 if win, -1 if lose and 0 if draw.

<u>Program Design</u>

The basic idea of the program is that the agent learns the best action by interacting with the environment over a number of iterations.

The program starts with the observation of the initial state, which is given by the environment. The agent would choose an action by ε-greedy policy, which is to choose either a random action at the rate of ε, or the action with the maximum Q-value at a rate of (1-ε). Then the environment would return a new state according to the agent's action. The agent would update his Q-value and repeat his action choosing process until the game is terminated and return with a reward.

The learning process would be calculating the loss function of the current Q-value and the target Q-value of a state, then backpropagating the loss value to the Deep Neural Network model to update the weight and bias. Over a number of iterations, the loss value would be minimized and the Deep Neural Network would give the best policy of maximum reward.

Model Setup

```
class DQN(nn.Module):

  def __init__(self, n_observations, n_actions):
    super(DQN, self).__init__()
    self.layer1 = nn.Linear(n_observations, 128)
    self.layer2 = nn.Linear(128, 128)
    self.layer3 = nn.Linear(128, n_actions)

  def forward(self, x):
    x = self.layer1(x)
    x = F.relu(x)

    x = self.layer2(x)
    x = F.relu(x)

    x  = self.layer3(x)

    return x
```

The Deep Neural Network model I used has 2 hidden layers, with 128 nodes each, followed by a ReLU activation function on each hidden layer. The ReLU function would return 0 if the input is negative. As the Blackjack environment returns a state of 3 tuples and the agent's action only has 2 options, the input layer of the model would have 3 nodes while the output layer has 2 nodes.

```
gamma = 0.99
epsilon = 1
eps_end = 0.01
eps_dec = 5e-4
learning_rate = 0.001
batch_size = 64
mem_size = 100000
```

As there is only a very limited state in one game (mostly around 2 to 3 states) before it terminates, I have chosen the discount factor of 0.99. It means the future reward would have almost as much influence as the immediate reward for the agent's action.

I choose a decayed epsilon starting from 1 and ending at 0.01 with a decay of $5e^{-4}$ over iteration, in order to balance between exploitation and exploration. The agent would start with exploring the reward of random action more often at the beginning, and as the agent learns more from the environment and improves his policy, the rate of random action would decrease and be replaced by greedy action. The minimum epsilon would be reached after 1500 iterations.

The learning rate I choose is 0.001, which would take a relatively small step on optimizing the loss value. As the iteration is large enough (i.e. 5000 or 20000), the loss value could reach its optimum despite the small steps.

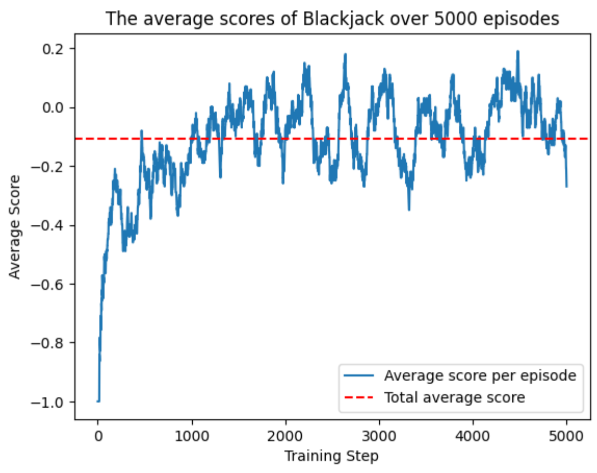
The optimizer I choose is Adam and the loss function is MSE.

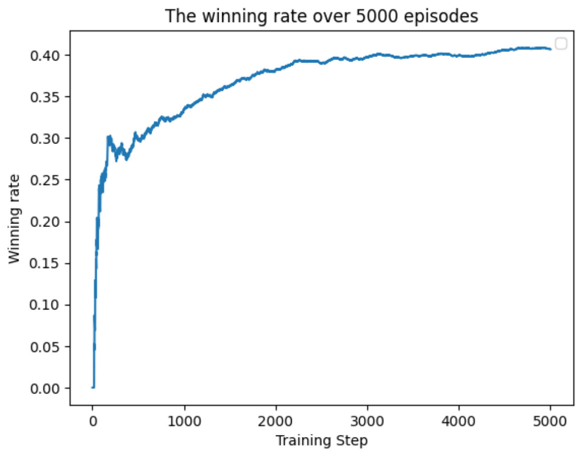
The training process would run 5000 times when using CPU while it would run 20,000 times when using GPU due to the runtime difference between CPU and GPU.

Result

DQN training of 5,000 iterations using CPU:

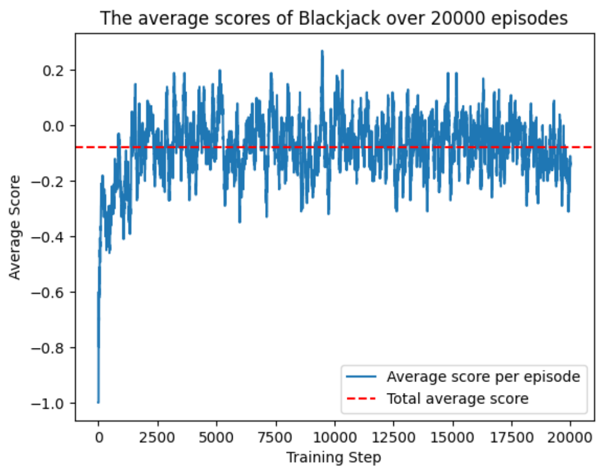
```
The total average score is -0.1062
The winning rate is 40.66%
The drawing rate is 8.06%
The losing rate is 51.28%
```

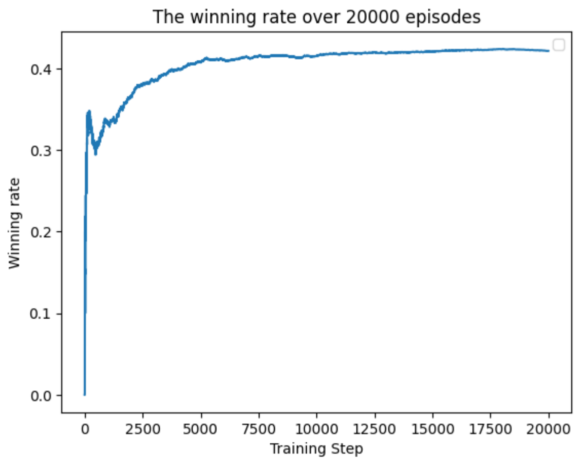




DQN training of 20,000 iterations using GPU:

```
The total average score is -0.0768
The winning rate is 42.16%
The drawing rate is 8.0%
The losing rate is 49.84%
```





Result Explanation

From the above results of 5,000 and 20,000 iterations, the average score and winning rate are both improving through the iteration. The average score starts to fluctuate around the total average score and the curve of winning rate is flatten after a number of iterations. It shows the model is exploring and learning the rewards drastically at the first few thousand episodes, then the model found the maximum reward policy and stayed with it with little learning and exploration afterward.

The difference between two numbers of iteration is that the 20,000 iterations have a slightly higher total average score and the winning rate than the 5,000 iterations. One of the reasons is that as the number of iterations is larger, the effect of the bad rewards during the exploration process at the beginning would be relatively smaller in percentage. Another reason is that the learning process of the deep neural network model has not fully completed within 5,000 iterations. The winning rate of 5,000 iterations has only 40.66% while the 20,000 iterations has 42.16%, which means the model is not optimized yet in the 5,000 iteration and still has room for improvement. This is due to the small learning rate (i.e. 0.001) chosen in the model with little improvement per step. The small learning rate also has its advantage of having a steady optimizing process and with less chance of bouncing back or over the minimum loss value.

The negative average score and the losing rate is higher than the winning rate means that the player is disadvantaged by using the best policy under this model. The average score and the winning rate might be improved by using a different model with more hidden layers or more nodes in the hidden layers. It is also possible that the agent is not exploring enough from the environment. A smaller epsilon decay value could be chosen to extend the exploration process. Choosing other loss function or optimizer might also be helpful.

Problem 2:

Exploration means choosing an action at random in order to acquire more knowledge of the environment and look for the action that can bring the agent maximum reward. In deep reinforcement learning, such exploration should be more frequent at the beginning so that the reward of different actions can be widely explored and find the one with the best reward.

Exploitation is to choose the action only with the maximum value. It is a greedy method that only focuses on maximizing the estimated value and ignores the actual value. It exploits the agent from knowing the rewards of other actions. In the long run, it reduces the total reward by not optimizing the actual maximum reward from action.

The trade-off between exploration and exploitation is important regarding the total reward in deep reinforcement learning. The exploration could try different actions and learn the action's reward at each state in the deep neural network. It requires a large amount of exploration to find the best policy if the number of states from the environment is huge. Insufficient exploration would cause the model to mistakenly estimate a lower or higher reward of an action than its actual, then affecting the agent's decision on exploitation choosing the maximum regard policy. Moreover, the deep neural network needs to take a large amount of trial and error to train for the best policy. If the number of iterations over actions is not enough, the neural network may result in poor improvement on the current policy.

Since the exploration takes a large number of iterations to train, the cost of exploration could be very high. During the exploration, it is possible that the random action would lead to a low reward, hence affecting the total reward significantly. Because of that, the exploitation could stop the agent from further choosing the bad action and turn to the action with the highest known reward. It would increase the total rewards by repeating the same action with the maximum estimated value.

To solve the exploration-exploitation dilemma, one of the strategies is to choose a decayed epsilon value to balance the number of exploration and exploitation. The chance of exploration would be high at the beginning, meaning that more diverse actions would be performed when the agent has little information about the environment. Then the possibility of exploration would decrease over time when more and more information is collected. An ideal decay rate would give enough iterations to explore most action, return with the best policy and stick to the policy with a little chance for exploring though.