NUERUAL NETWORK THEORY AND APPLICATION ASSIGNMENT 4

Tianyue Cao
119034910071

1 Problem

1.1 Problem Introduction

The goal of this assignment is to apply reinforcement learning to solve a simple game, called *Easy 21*. This exercise is similar to the Blackjack example discussed in the class. However, please note that the rules of the card game are different and non-standard.

1.2 Environment Introduction

The rule of game of Easy 21 is defined as follows:

- Each draw from the deck results in a value between 1 and 10 (uniformly distributed) with a color of red (probability 1/3) or black (probability 2/3).
- At the start of the game both the player and the dealer draw one black card
- Each turn the player may either stick or hit. If the player hits then he/she draws another card from the deck. If the player sticks he/she receives no further cards
- The values of the player's cards are added (black cards) or subtracted (red cards). If the player's sum exceeds 21, or becomes less than 1, then he/she goes "bust" and loses the game (reward -1).
- If the player sticks then the dealer starts taking turns. The dealer always sticks on any sum of 16 or greater, and hits otherwise. If the dealer goes "bust", then the player wins (reward+1); Otherwise the outcome and reward is computed as follows: the player wins (reward+1) if player's sum is larger than the dealer's sum; the player loses (reward -1) if the player's sum is smaller than the dealer's sum; the reward is 0 if the two sums are the same.
- Assumption: The game is played with an infinite deck of cards (i.e. cards are sampled with replacement)

1.3 Problems

1.3.1 Implementation of Easy21 environment

You should write an environment that implement the Easy21 game. Specifically, the environment should include a function step, which takes a state (dealer's first card, player's current sum) and an action (stick or hit) as inputs, and returns a next state (which may be terminal) and a reward.

$$\textit{next state, reward} = \textit{step(state, action)}$$

You should treat the dealer's moves as part of the environment. In other words, i.e. calling step with a stick action will play out the dealer's cards and return the final reward and terminal state. There is no discounting ($\gamma = 1$). We will be using this environment for reinforcement learning.

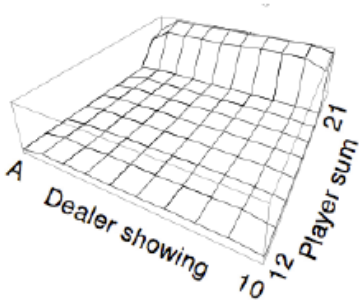

Figure 1: The optimal state-value function of Sutton and Barto's Blackjack example.

1.3.2 Q-learning in Easy21

Apply Q-learning to solve Easy21. Try different learning rates α exploration parameter ϵ and episode numbers. Plot the learning curve of the return against episode number. Can you find the optimal policy in this game? Plot the optimal state-value function using similar axes to Figure 1 taken from Sutton and Barto's Blackjack example.

$$V^*(s) = \max_a Q*(s,a)$$

2 Basic Information

2.1 Hardware and Software Environment

The hardware and software environments are shown as following:

- **Operation system:** Ubuntu 18.04 LTS
- **Processor:** Intel Core i5-4210H CPU 2.90GHz × 4
- **GPU:** None
- **Requirements:** python 3.6, numpy, matplotlib, pandas

2.2 Game

Easy 21 is a variant of Blackjack, which is not a standard card game. The rule of the game is shown clearly in Section 1.2.

3 Solution for Problem 1

In this section, I reimplement an environment of the Easy 21 game. The environment includes the implementation of state, action, step function and initialize function. The environment is used for Q-learning algorithm in the next section.

3.1 Basic representation and initialization

The state includes 2 parts: dealer's first card (d) and player's current sum (p). I represent state as an integer $d * 100 + p$, which is an integer in $\{101, 102, ..., 121, 201, 202, ..., 221, ..., 1001, 1002, \}$. Action is represented by a bool variable, 0 for stick and 1 for hit.

For the initialization of state, a black card is drawn to both the dealer and the player. So the dealer's first card or the player's current sum is a random integer between 1 and 10.

3.2 `step(state, action)` function

The `step(state, action)` function aims to do action under the current state and get the next state and the reward. The pesudo code is shown in Algorithm 1. The algorithm response according to the action and get the next state and reward.

Algorithm 1 Step Function to Get Next State and Reward

Input: s: state; a: action;
Output: next state $next\_s$, reward r

$d = s//100$ and $p = s\%100$;
if a **then**
    draw a card to player and update p
    if $p < 1$ or $p > 21$ **then return** "terminal", -1.0
    elsereturn $d * 100 + p$, 0.0
    end if
end if
if not a **then**
    repeat
        draw a card to dealer and update d
        if $d < 1$ or $d > 21$ **then return** "terminal", 1.0
        end if
    until $d \geq 17$
    if $d > p$ **then**
        return "terminal", -1.0
    end if
    if $d > p$ **then**
        return "terminal", -1.0
    end if
    if $d = p$ **then**
        return "terminal", 0.0
    end if
end if

4 Solution for Problem 2

In this section, a Q-learning algorithm is used to explore the strategy of playing the game.

4.1 Q-learning

Q-learning algorithm is a typical reinforcement learning algorithm. The pesudo code is shown in Algorithm 2. ϵ-greedy algorithm can motivate the agent to exploit more strategies randomly, and the decreasing factor α is used to decrease the long-term reward, the factor γ is used to limit the influence of the optimal result of the next state.

Algorithm 2 Q-learning Algorithm

initialize $Q(s, a)$ arbitrarily
repeat(for each episode)
    initialize s
    repeat(for each step of episode)
        choose a from s using policy derived from Q (e.g., ϵ-greedy)
        take action a, observer r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    until s is terminal
until all episodes finished

4.2 Experiments and Results

To get the optimal strategy for the Easy 21 game, I have done experiments under different hyper-parameters ($\epsilon s, \alpha s$). In this game, γ is set to 1 according to the problem description.

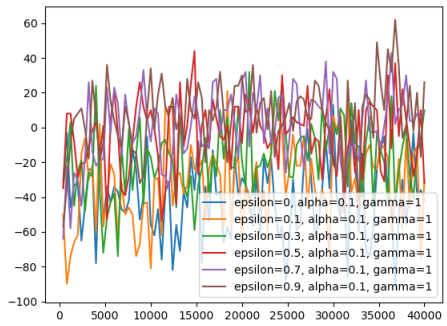

fig a. Cumulative rewards with 400 average episodes, different epsilons, and alpha=0.1.

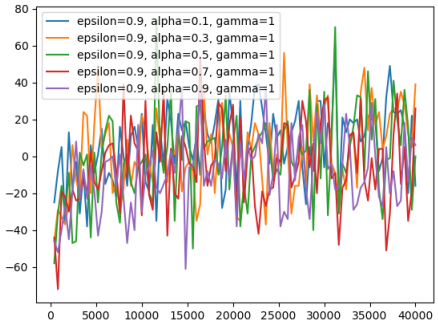

fig b. Cumulative rewards with 400 average episodes, different alphas, and epsilon=0.9.

Figure 2: Cumulative rewards with different hyper-parameters.

4.3 Hyper-parameters

The results of learning curves under different ϵs and αs are shown in Figure 2. X axis is the episode of the experiment, y axis is the cumulative reward. Since the episode number is pretty large, the rewards are averaged calculated in 400 episodes. According to the results, the cumulative reward is up and down around 0 and it is hard to tell the difference of performance between different hyper-parameters. One of the reason is that the game has much randomness, so the result of the game depends heavily on luck. So the reward goes up and down around zero.

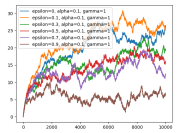

fig a.1. Sum of optimal V values with different epsilons and alpha=0.1 in episodes 10000.

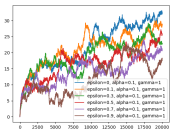

fig a.2. Sum of optimal V values with different epsilons and alpha=0.1 in episodes 20000.

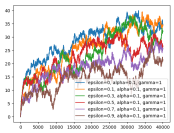

fig a.3. Sum of optimal V values with different epsilons and alpha=0.1 in episodes 40000.

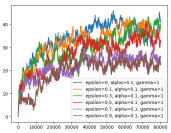

fig a.4. Sum of optimal V values with different epsilons and alpha=0.1 in episodes 80000.

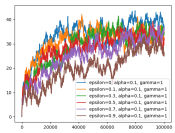

fig a.5. Sum of optimal V values with different epsilons and alpha=0.1 in episodes 100000.

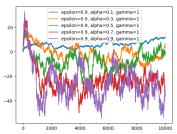

fig b.1. Sum of optimal V values with different alphas and epsilon=0.9 in episodes 10000.

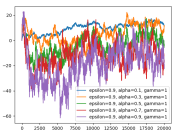

fig b.2. Sum of optimal V values with different alphas and epsilon=0.9 in episodes 20000.

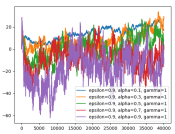

fig b.3. Sum of optimal V values with different alphas and epsilon=0.9 in episodes 40000.

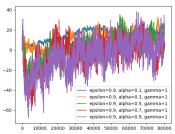

fig b.4. Sum of optimal V values with different alphas and epsilon=0.9 in episodes 80000.

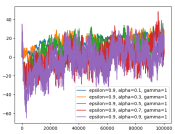

fig b.5. Sum of optimal V values with different alphas and epsilon=0.9 in episodes 100000.

Figure 3: Sum of optimal V values with different hyper-parameters.

ϵ	0	0.1	0.3	0.5	0.7	0.9
winning rate	0.44594	0.45953	0.48209	0.49478	0.50741	0.51446

Table 1: Winning rate with different ϵs and $\alpha = 0.1$.

To choose the optimal hyper-parameters, I use the sum of optimal choices' V values for all states as another evaluation indicator. The results are shown in Figure 3. Fig a.1-a.4 are results for $\epsilon = \{0, 0.1, 0.3, 0.5, 0.7, 0.9\}$ and $\alpha = 0.1$ in different episodes, while fig b.1-b.4 are results for $\alpha = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and $\epsilon = 0.1$ in different episodes. Different episode numbers shows the overview and details of the curves. Thought the sum of V values can show the performance of the algorithm, it is not the deterministic variable. So the winning rate for different hyper-parameters are shown in Figure 4. I have done 10 experiments for each hyper-patameter composition and 100,000 games for each experiment. The accuracies numbers are shown in Table 4.3 and Table 4.3.

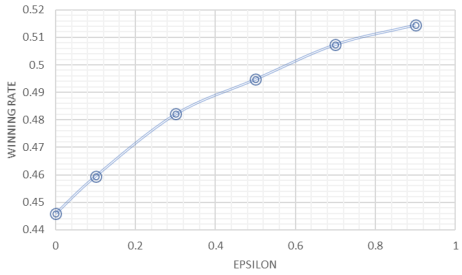

fig a. Winning rate with different epsilons and alpha=0.1.

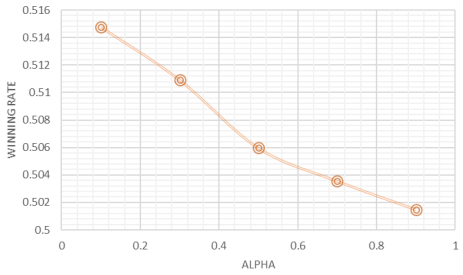

fig b. Winning rate with different alphas and epsilon=0.9.

Figure 4: Winning rate with different hyper-parameters.

α	0.1	0.3	0.5	0.7	0.9
winning rate	0.51478	0.51092	0.50596	0.50356	0.50149

Table 2: Winning rate with different αs and $epsilon = 0.9$.

ϵ is the parameter for ϵ-greedy strategy. The algorithm randomly choose action with probability $1 - \epsilon$, and greedily choose action with probability ϵ. Larger ϵ means less random exploition. According to fig a.1-a.5 in Figure 3, smaller ϵ can lead to larger sum of V values for optimal choices. But according to Table 4.3 and Figure 4, larger ϵ can achieve higher winning rate. As for winning rate is a more determinstic evaluation variable for the algorithm, $\epsilon = 0.9$ is the optimal choice.

α can be regard as the learning rate for the Q-learning algorithm. According to fig b.1-b.5 in Figure 3, larger α leads to more unstable training process. Thus the curve is more volatile. Also, the sum of V values are higher with smaller α. Table 4.3 and Figure 4 also shows that smaller α can achieve higher winning rate. So the optimal alpha is 0.1. The result makes sense that if α is too large, the algorithm is too aggresive and hard to converge.

4.4 Optimal Policy

The optimal policy is shown in Table 4.4. The value of the corresponding policy for all states are shown in Table 4.4. The color for the cells are relative to the value. Colors from high to low are from red to white to blue. The optimal state-value function in different views are shown in Figure 5.

Under the condition that the player's current sum is small and the dealer's first card is large, the reward is pretty high. And when the player's current sum is near to 21 and the dealer's first card is near to 10, the reward is also high. When the player's current sum is large and the dealer's first card is small, the value of the reward is small.

The winning rate for different strategies are shown in Table 4.4. If the player randomly choose hit or stick, the winning rate can achieve 0.45669. Q-learning policy can achieve higher winning rate. Also, hit strategy is hard to win and can only achieve 0.10576 winning rate, while stick strategy can achieve 0.51773 winning rate. The winning rate for stick

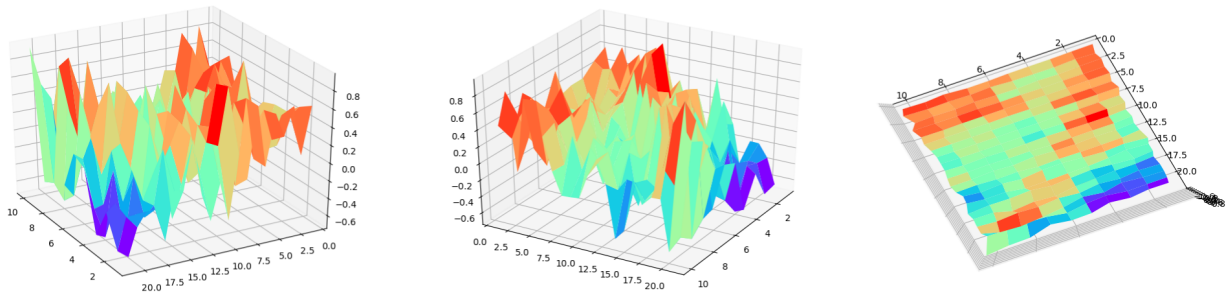

Figure 5: The optimal state-value function in different angle of views.

strategy	random	hit	stick	Q-learning
winning rate	0.45669	0.10576	0.51773	0.51478

Table 3: Winning rate with different strategies.

p\d	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	1	1	1	1	1	1	0	1	1
2	0	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1	1
4	1	0	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	0	1	1	1	0	0	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	0	1	0	0	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	0	1	1	1
13	1	0	1	1	1	1	1	1	1	1
14	1	1	1	0	1	1	1	1	0	1
15	1	1	1	1	1	1	1	1	1	1
16	1	1	1	0	1	1	1	1	1	1
17	1	1	1	1	1	1	1	1	1	1
18	1	1	1	0	1	1	1	1	1	0
19	1	0	1	1	1	1	1	1	1	1
20	1	0	1	1	1	1	1	0	0	0
21	1	0	1	1	1	1	1	1	1	1

Table 4: Policies for all states.

p\d	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.596	0.611	0.212	0.486	0.291	0.524	0.116	-0.448	0.474	0.477
2	-0.048	0.053	0.320	0.417	0.388	-0.069	0.449	0.494	0.578	0.640
3	0.000	0.096	0.248	0.602	0.486	0.327	0.650	0.484	0.682	0.701
4	0.586	-0.141	0.291	0.487	0.258	0.084	0.404	0.430	0.214	0.544
5	0.813	0.369	0.014	0.387	0.420	0.518	0.110	0.344	0.538	0.159
6	0.120	0.580	0.271	0.173	0.565	-0.031	0.563	0.053	0.003	0.921
7	0.775	0.749	0.388	0.286	0.380	0.631	0.276	0.340	0.032	0.152
8	-0.449	0.066	-0.161	-0.212	-0.236	-0.362	0.233	-0.029	0.371	0.372
9	0.527	0.846	0.743	0.746	-0.248	0.284	0.144	-0.043	0.146	-0.124
10	-0.197	-0.126	-0.411	-0.163	-0.360	-0.301	-0.012	0.146	0.012	0.081
11	0.218	0.329	0.758	0.980	0.771	0.342	0.561	-0.071	-0.054	-0.034
12	0.033	-0.301	0.631	-0.099	-0.241	-0.024	-0.463	0.476	0.041	0.158
13	0.510	-0.313	0.642	0.618	0.538	0.774	-0.431	-0.350	0.109	-0.048
14	-0.392	0.002	-0.069	-0.448	-0.124	0.023	-0.250	0.220	-0.415	0.155
15	-0.005	-0.238	-0.104	0.598	0.680	0.804	0.849	-0.238	-0.251	0.083
16	0.241	-0.305	0.200	-0.641	-0.208	0.014	-0.001	0.166	-0.077	0.047
17	-0.108	-0.219	-0.401	-0.014	0.570	0.303	0.882	0.850	0.238	-0.307
18	-0.426	-0.449	-0.277	-0.727	0.068	0.072	-0.150	-0.426	-0.199	-0.590
19	-0.323	-0.714	-0.307	-0.255	0.090	0.209	0.617	0.870	0.959	-0.208
20	-0.391	-0.734	-0.173	-0.304	-0.494	-0.371	0.351	-0.733	-0.618	-0.628
21	-0.518	-0.671	-0.150	-0.274	-0.110	0.172	0.322	0.820	0.797	0.974

Table 5: Value for all states. p represents player's current sum and d represents dealer's first card. The color level for values is red-white-blue.

strategy and Q-learning policy is similar. It makes sense that if the player always choose hit, it is easy to lose because of $p < 1$ or $p > 21$, and if the player always choose stick, he stays the initial value and waits for the dealer to lose. Q-learning methods mostly choose hit, but choose stick on some particular case, and it can achieve competitive winning rate to stick strategy. This shows the feasibility of Q-learning method. Though a good policy can be generated, the winning rate can only achieve around 0.5. So the player cannot win a lot. In probability, the dealer is hard to lose.

5 Conclusion

In this assignment, I have implemented a Q-learning algorithm to achieve a policy for game Easy 21. An environment is implemented in problem (1) and the Q-learning algorithm is implemented in problem (2). Different parameters are experimented and get an optimal policy. However, the policy can only achieve about 0.5 winning rate. The game has pretty much randomness and player cannot win a lot in the Easy 21 game.