FMAN45 - Assignment 4

Jingmo Bai

May 23, 2023

1 Exercise 1

First, we want to find out how many possible states for a length-3 snake to be in the grid. There are 5*5=25 possible positions for the head of the snake. Next, we find the possible body shape for each position of the head. The results are:

4	6	7	6	4
6	10	11	10	6
7	11	12	11	7
6	10	11	10	6
4	6	7	6	4

The total number of possible head positions and body shapes for a snake in the grid is 188. Then for each possible state of the snake, there are 25-3=22 possible positions for the apple. So in total, the value of K is 188*22=4136.

2 Exercise 2

a)

$$Q^*(s,a) = \mathbb{E}[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

b)

$$Q^*(s,a) = \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^T R_T]$$

c)

It is saying that the optimal Q-value for a given state-action pair (s,a) is the expected reward if we start in state s, take an action a, and then forever act according to the optimal policy, it is the reward we can get from the current state plus the maximum possible reward we can get from the next state according to the optimal policy times a discount factor.

d)

The policy we follow in (1) is the optimal policy in the environment. The "max" part is where we make this assumption, we take the action a' that gives us the maximum possible reward.

e)

γ is the discount factor, the effect is the reward will be discounted by how far off in the future they're obtained. With a γ closer to 0, the reward in a far future doesn't count any more.

f)

If the action will not lead to eating the apple, then for the next state decided by the movement of the snake, $T(s, a, s') = 1$, for the other else states, $T(s, a, s') = 0$. If the action will lead to eating the apple. Once the snake eats the apple, there will be a new one placed at uniform random on an unoccupied pixel in the grid, so the transition probability for each state is the same for the rest 22 empty pixel, which is $T(s, a, s') = 1/22$.

3 Exercise 3

a)

On-policy means the policy that is used for updating and the policy used for acting are the same. Off-policy means the updated policy is different from the behaviour policy. On-policy methods attempt to evaluate or improve the policy that is used to make decisions. In contrast, off-policy methods evaluate or improve a policy different from that used to generate the data.

b)

Model-based methods mean the agent has access to (or learns) a model of the environment. By a model of the environment, we mean a function which predicts state transitions and rewards. Model-free methods mean the agent has no access to the model of the environment.

c)

For passive reinforcement learning, the agent's policy is fixed. In contrast, in active reinforcement learning, an agent needs to decide what to do as there's no fixed policy that it can act on. Therefore, the goal of a passive RL agent is to execute a fixed policy and evaluate it. The goal of an active RL agent is to act and learn an optimal policy.

d)

Supervised learning is the case when we train with labelled data, the purpose is to predict the output of new data as input. For example, classification and regression problems. Unsupervised learning is the case when we train with unlabelled data, the purpose is to find the hidden patterns in data. For example, clustering and association problems. Reinforcement learning is to train an agent to perceive and interpret the environment, to learn the optimal actions to achieve some goals by trial and error, reward and punishment.

e)

For a dynamic programming approach, we need to know the complete information of our environment, to describe it as a Markov Decision Process. And it will always give us a deterministic optimal policy. For reinforcement Learning approaches we don't need the complete model. It will create policy solely based on evaluating which of its actions return a higher reward by interacting with its environment. But there is no guarantee that the model will converge.

4 Exercise 4

a)

$$V^*(s) = \max_a \mathbb{E}[R(s, a, s') + \gamma V^*(s')]$$

b)

It is saying that the optimal V-value for a given state is the expected reward if we start in state s, and always act according to the optimal policy, it is the reward we can get from the current state plus the maximum possible reward we can get from the next state according to the optimal policy times a discount factor.

c)

The max operator is to pick the action that gives the highest reward so it is the optimal policy at that step in the environment.

d)

π^* is the optimal action taken to get the next optimal V^*.

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

e) Because the Q^* is the optimal Q value when the taken optimal action a is one of its arguments, given by the state-action pair. So when we want to find the action we just use the arg max. But the V value does not contain the action as an argument, we know it is an optimal value but the action is not given.

5 Exercise 5

a)

```
% FILL IN POLICY EVALUATION WITHIN THIS LOOP.
    a = policy(state_idx);
    v = values(state_idx);
    next_idx = next_state_idxs(state_idx,a);
    if next_idx == -1
        values(state_idx) = rewards.apple;
    elseif next_idx == 0
        values(state_idx) = rewards.death;
    else
        values(state_idx) = rewards.default + gamm*values(next_idx);
    end
    Delta = max(Delta, abs(v-values(state_idx)));

% FILL IN POLICY IMPROVEMENT WITHIN THIS LOOP.
    next_idxs = next_state_idxs(state_idx,:);
    for i=1:3
        next_index = next_idxs(i);
        if next_index == -1
            temp_values(i) = rewards.apple;
        elseif next_index == 0
            temp_values(i) = rewards.death;
        else
            temp_values(i) = rewards.default + gamm*values(next_index);
        end
    end
    [~,idx] = max(temp_values);
    if idx ~= policy(state_idx)
```

```
        policy_stable = false;
    end
    policy(state_idx) = idx;
```

b)

γ	policy iterations	policy evaluations
γ=0	2	4
γ=1	NULL	NULL
γ=0.95	6	38

Table 1: The number of policy iterations and evaluations for each γ

For $\gamma = 0$, the snake agent doesn't play optimally, it will circle around and could not find the apple to eat. Because the future rewards do not count any more, the agent will not find the apple to eat.

For $\gamma = 1$, the model can not converge, the delta always stays at 5.5023. Because the future rewards will count equally as the present ones, the code is stuck in an infinite loop, and the policy evaluation will never finish.

For $\gamma = 0.95$, we can get an optimal snake agent, it can find the optimal actions to eat the apple and never goes wrong. This is a reasonable discount factor.

c)

ϵ	policy iterations	policy evaluations
$\epsilon = 1e-4$	6	204
$\epsilon = 1e-3$	6	158
$\epsilon = 1e-2$	6	115
$\epsilon = 1e-1$	6	64
$\epsilon = 1e0$	6	38
$\epsilon = 1e1$	19	19
$\epsilon = 1e2$	19	19
$\epsilon = 1e3$	19	19
$\epsilon = 1e4$	19	19

Table 2: The number of policy iterations and evaluations for each ϵ

All of the final snake agents using various ϵ above seem to work reasonably. They can all find and eat the apple without hitting the wall. When the ϵ is smaller, the tolerance for each iteration is smaller, so it takes more evaluations.

6 Exercise 6

a)

If terminate

```
sample                  = reward ;
pred                    = Q_vals(state_idx,action);
td_err                  = sample - pred;
```

```
Q_vals(state_idx, action) = Q_vals(state_idx, action) + alph*td_err;
```

If not

```
sample                    = reward + gamm*max(Q_vals(next_state_idx,:));
pred                      =  Q_vals(state_idx,action);
td_err                    = sample - pred;
Q_vals(state_idx, action) = Q_vals(state_idx, action) + alph*td_err;
```

b)

ϵ	α	Reward(apple/death)	Score
0.2	0.5	100/-1	60
0.2	0.5	100/-100	5086
0.1	0.5	100/-100	7849

Table 3: Parameters configurations and scores

I begin by tuning the parameters including ϵ and α, but the result is it could never reach a high score, then I tried to change the rewards for the apple and death, and it works. For the first attempt, it doesn't work well, because the punishment of death is not enough, so it died too soon. I increased the punishment to death, and it turns out to pass the task. And then I find that a smaller value could be better for ϵ. However, it will also take a longer time to train.

c)

My final settings:
Reward: 100 for apple, -100 for death
γ: 0.9
α: 0.5
ϵ: 0.025
Score: Infinite
I believe I find a pretty good policy. If I don't terminate it, the score just keeps increasing. But I have no method to evaluate if the agent reaches the apples as fast as possible. So it's hard to say if it is an optimal one.

d)

According to the 1st exercise, we know that in this snake game, there can be 4136*3 state-action pairs in total. I think it is not enough to explore the entire environment within 5000 episodes, that's why it can be difficult to achieve optimal behaviour in this task within 5000 episodes.

7 Exercise 7

a)

If terminate

```
target  = reward ;
pred    = Q_fun(weights, state_action_feats, action);
td_err  = target - pred; % don't change this
weights = weights + alph*td_err*state_action_feats(:,action);
```

If not

```
target  = reward + gamm*max(Q_fun(weights, state_action_feats_future));
pred    = Q_fun(weights, state_action_feats, action);
td_err  = target - pred; % don't change this
weights = weights + alph*td_err*state_action_feats(:,action);
```

b)

- Attempt 1

Parameters configurations:
number of features = 2;
rewards: 'default', 0, 'apple', 10, 'death', -10
gamma = 0.99;
alph = 0.5;
eps= 0.2;
initial weights = [1;1]
State-action features functions:

```
[apple_loc_r, apple_loc_c] = find(grid == -1);
apple_loc = [apple_loc_r, apple_loc_c];
[next_head_loc, next_move_dir] = get_next_info(action, movement_dir, head_loc);
distance = norm(next_head_loc-apple_loc,1); % distance to the apple
maxlength = sqrt(2)*(N-2);
state_action_feats(1, action) = distance/maxlength; % max 1 min 0
state_action_feats(2, action) = grid(next_head_loc(1), next_head_loc(2));
% hit the wall or body/ eat the apple
```

Score: Agent seems stuck in an infinite loop.

- Attempt 2

Parameters configurations:
number of features = 2;
rewards: 'default', -1, 'apple', 10, 'death', -10
gamma = 0.99;
alpha = 0.5;
epsilon= 0.1;
initial weights = [1;1]
State-action features functions: remain the same
Score:
Final weights:
-31.1868
1.2278
Mean score after 100 episodes: 6.95

- Attempt 3

Parameters configurations:
number of features = 2;
rewards: 'default', -1, 'apple', 10, 'death', -10
gamma = 0.99;

alpha = 0.5;
epsilon= 0.2;
initial weights = [1;1]
State-action features functions: remain the same
Score:
Final weights:
-30.0568
0.4107
Mean score after 100 episodes: 11.71

On the first attempt, the agent is stuck in an infinite loop. I tried to tune the parameters, but it didn't work. And I also tried to use some other features functions, but it did not change the situation, so I change them back. I think the features functions I use are reasonable, one is to find the distance from the head to the apple, and the other one is to find if the next step will hit the wall or body, or eat the apple. There is no need to change them.

On the second attempt, I changed the reward for default to -1, which makes the model work. It seems like we have to punish the model for playing safe.

On the third attempt, I tried to tune the discount factor, learning rate and epsilon. I got a slightly better score, but it is not enough.

c)
Parameters configurations:
number of features = 2;
rewards: 'default', -1, 'apple', 10, 'death', -10
gamma = 0.9;
alpha = 0.5;
epsilon= 0.1;
initial weights = [1;1]
State-action features functions:

```
[apple_loc_r, apple_loc_c] = find(grid == -1);
apple_loc = [apple_loc_r, apple_loc_c];
[next_head_loc, next_move_dir] = get_next_info(action, movement_dir, head_loc);
distance = norm(next_head_loc-apple_loc,1); % distance to the apple
maxlength = sqrt(2)*(N-2);
state_action_feats(1, action) = distance/maxlength; % max 1 min 0
state_action_feats(2, action) = grid(next_head_loc(1), next_head_loc(2));
% hit the wall or body/ eat the apple
```

Score:
Final weights:
-15.5878
-4.4622
Mean score after 100 episodes: 42.52

	Feature 1	Feature 2
Initial weight w_0	1	1
Final weight w	-15.5878	-4.4622

Table 4: Initial weights and final weights

Compared to the previous attempts, I just changed the discount factor to 0.9. The reason why it works better than 0.99 is that the future rewards are not discounted enough when we use 0.99. The agent will get almost the same rewards even if it is still several steps away from the apple.

However, after I get this score, I tried many times with different combinations of all the tunable parameters. But the average scores remain the same at 42.52, despite the final weights being different every time.

I believe the score will be higher if we use a proper third feature function, but I tried several different ones, they could not make it better.