Assignment 4 - Machine Learning

Theodora Gaiceanu

Exercise 1

The snake can have the following forms: line (with the head at the upper end or with the head at the bottom end), bent to the left or bent to the right. We also know that the head can point N/S/E/W. One can first take into account just one direction, let's say N. For this direction we can have the following states:

1. line - $3 \cdot 5 = 15$ positions
2. bent to the left - $4 \cdot 4 = 16$ positions
3. bent to the right - $4 \cdot 4 = 16$ positions

Therefore, without taking into consideration the apple, we have $4 \cdot (3 \cdot 5 + 4 \cdot 4 + 4 \cdot 4) = 188$ states. For every state, the apple can be in one of the $5 \cdot 5 - 3 = 22$ left cells. Consequently, in total we have:

$$K = 188 \cdot 22 = 4136$$

Exercise 2

a

In general, the expectation of a random variable X is:

$$\mathbb{E}[X] = \sum_{i=1}^{n} P(x_i) \cdot x_i,$$

where $P(x_i)$ is the probability of the random variables x_i.
And we have the given expression:

$$Q^*(s,a) = \sum_{s^{'}} T(s,a,s^{'})[R(s,a,s^{'}) + \gamma max_{a^{'}} Q^*(s^{'},a^{'})]$$

Consequently, we can view $T(s,a,s^{'})$ as the probability and $R(s,a,s^{'}) + \gamma max_{a^{'}} Q^*(s^{'},a^{'})$ as the random variable. This way, it holds that:

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

b

Equation (1) can be written also as:

$$Q^*(s,a) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a],$$

where $r_{t+i}, i = 0, ..$ are the components of the reward function.

c

Equation:

$$Q^*(s,a) = \sum_{s^{'}} T(s,a,s^{'})[R(s,a,s^{'}) + \gamma max_{a^{'}} Q^*(s^{'},a^{'})]$$

is saying that the Q-value of an action a in state s can be actually expressed as the expected value of the reward $R(s,a,s^{'}) + \gamma max_{a^{'}} Q^*(s^{'},a^{'})$ with probability $T(s,a,s^{'})$ under optimally conditions for the agent after action a. s is the state, $s^{'}$ is the following state, $a^{'}$ is the following action, T is the transition function, R is the reward function, γ is the discount factor, $Q^*(s,a)$ is the expected utility starting out having taken action a from state s and acting optimally after.

d

The equation (1) from the assignment description uses the optimal policy. This is because we take the maximal Q-value ($max_{a^{'}} Q^*(s^{'},a^{'})$).

e

γ is the discount factor (it is a tuning parameter of the future reward) and it helps reaching the goal as soon as possible. A small value of γ means that we have a small "horizon" - the agent can focus only on short term; a big value of γ means that we have a large "horizon". Typically, $0 < \gamma < 1$. When $\gamma = 0$, it means that we are not taking into consideration the future rewards and the snake knows what is optimal by looking one step in the future. When $\gamma = 1$, it means all the future rewards are taken into consideration, but the agent doesn't consider the shortest number of strps to eat the apple.

f

For the small snake game, the transition function $T(s,a,s^{'})$ can have only 3 values: 0, 1 and $\frac{1}{22}$ (as there are 22 possible cells for the apple and the position of the apple is picked from a uniform distribution). So if s is a state in which the head of the snake is further than one cell from the apple and the agent takes an action a that is possible given the configuration of the snake and the action a, then the snake will be in an updated position in state $s^{'}$ and the apple will be in the same place. The transition function $T(s,a,s^{'})$ is 1 in this case. But if s is a state in which the head of the snake is further than one cell from the apple and the agent takes an action a that is NOT possible given the configuration of the snake and the action a, then the transition function $T(s,a,s^{'})$ is 0. If s is a state in which the head of the snake is one cell away from the apple and a is an action that makes the snake move in the direction of the apple, then in state $s^{'}$ the snake is updated and the apple is in another cell (in one of the 22 possible cells). In this case, the transition function $T(s, a, s^{'})$ is $\frac{1}{22}$.

Exercise 3

a

With on-policy learning, a policy is used for selecting the actions for the agent and then the same policy is evaluated and improved. With off-policy learning, a policy is used for selecting the actions and a different one is used for evaluation and improvement. Therefore, off-policy doesn't depend on the actions of the agent. Also, on-policy learns the suboptimal policy, while off-policy learns the optimal policy. For example, by using off-policy, the Q-learning converges to the optimal solution even if you're acting suboptimally [1].

b

In model-based reinforcement learning, the first step is to learn the MDP quantities: transition function $T(s, a, s^{'})$ and reward function $R(s, a, s^{'})$. Then one can use MDP algorithms (such as policy iteration) in order to find the policy [1]. The advantage is that it is more sample efficient. The downside is that the policy can only be as good as the model is.

In model-free reinforcement learning, the transition function and the reward function are not learnt explicitly. The agent learns directly the policy / state-value (for instance, using Q-learning) [1]. The advantage is that the policy is not bounded by the model anymore, but the downside is that it takes longer to learn.

c

First of all, in both passive and active reinforcement learning, the transition $T(s, a, s^{'})$ and the reward $R(s, a, s^{'})$ are not known. But in passive reinforcement learning, the policy π is given and it is fixed and the goal is to learn state values $V(s)$ [1]. The agent doesn't make any choice about the actions, it just executes the policy and it learns from experience. Meanwhile, in active reinforcement learning, the agent actually chooses the actions and the goal is to learn the optimal policy π and maybe the state value $V(s)$ [1].

d

In supervised learning, one has the features and the labels for them. In unsupervised learning, one has only the features, without any labels. In reinforcement learning, one has agents that can make choices and act rationally (it takes actions that maximize the reward). No labels are needed in reinforcement learning. Here, there is a trade off between exploration (of the unknown environment) and exploitation (of the current knowledge) [2].

e

First of all, reinforcement learning there is no assumed knowledge of a certain mathematical model of the MDP. Also, reinforcement learning deal with large MDP (Markov Decision Process), where exact methods are not applicable [2]. In dynamic programming, a finite MDP is needed to describe the environment. With reinforcement learning, the agent can work by considering which actions give a bigger reward, it doesn't need a MDP. Policy iteration is a dynamic programming algorithm where a transition and a reward function are given and they compute in an iterative way a value function and an optimal policy. Q-learning is an example of a model-free reinforcement learning algorithm that learns directly the policy / state value.

Exercise 4

a

We use again the expression for the expectation of a random variable X:

$$\mathbb{E}[X] = \sum_{i=1}^{n} P(x_i) \cdot x_i,$$

where $P(x_i)$ is the probability of the random variables x_i.
And we have the given expression:

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

Considering $T(s,a,s')$ as the probability and $R(s,a,s') + \gamma max_{a'} Q^*(s',a')$ as the random variable, the expression can be rewritten as:

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

b

Equation:

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

is saying that the value of a state s can be actually expressed as the expected value of the reward $R(s,a,s') + \gamma V^*(s')$ (the utility of the next state) with probability $T(s,a,s')$ under the condition that action a is optimally chosen. s is the state, s' is the next state, T is the transition function, R is the reward function, γ is the discount factor.

c

The max-operator means that the action is chosen in an optimal way.

d

We know that:

$$\pi^*(s) = argmax_a Q^*(s,a) = argmax_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma max_{a'} Q^*(s',a')]$$

And we also have that:

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

Therefore, one can say that:

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

or

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

e

As it can be seen from the expression, V^* is the expected utility of a state s, therefore the action a is not stored. For Q^*, both the action a and the state s are stored, so Q^* is the expected utility of a action-state pair.

Exercise 5

a

The code that follows the policy improvement algorithm presented in the assignment description can be seen in Listing 1 and Listing 2.

Policy evaluation:

Listing 1: Policy evaluation

```
V = values(state_idx);
action = policy(state_idx);
s_new = next_state_idxs(state_idx, action);
if s_new == -1
    values(state_idx) = rewards.apple;
elseif s_new == 0
    values(state_idx) = rewards.death;
else
    values(state_idx) = rewards.default + gamm*values(s_new);
end
Delta = max(Delta, abs(V - values(state_idx)));
```

Policy improvement:

Listing 2: Policy improvement

```
all_actions = next_state_idxs(state_idx, :);
k = 1;

for action = all_actions
    if action == -1
        V(k) = rewards.apple;
    elseif action == 0
        V(k) = rewards.death;
    else
        V(k) = rewards.default + gamm*values(action);
    end
   k = k + 1;
end

[~, action] = max(V);

if action ~= policy(state_idx)
    policy_stable = false;
end
policy(state_idx) = action;;
```

b

The number of policy iterations and evaluations for each γ can be analyzed in Table 1.

γ	policy iterations	policy evaluations
0	2	4
1	doesn't converge	doesn't converge
0.95	6	38

Table 1: Number of policy iterations and evaluations for each γ

The first option, $\gamma = 0$ means that the future rewards are actually not taken into account. The snake gets stuck into an infinite loop and doesn't get to eat the apple.

For $\gamma = 1$, the game doesn't even start because the policy iteration doesn't converge. This is because when $\gamma = 1$, all future rewards are taken into account, but the snake doesn't consider also the least number of steps to get to the apple (the future rewards are not discounted). The code is stuck in an infinite loop again.

For $\gamma = 0.95$, the game is played optimally and the snake eats the apple.

c

The number of policy iterations and evaluations for each ϵ can be analyzed in Table 2.

For all the considered values of ϵ, the snake plays optimally and eats the apple. But for $\epsilon \leq 1$, the number of policy iterations is smaller, but more policy evaluations are taken because it is harder to converge with a small tolerance. Which means that the value vector has a better estimate for $\epsilon \leq 1$. On the other hand, for $\epsilon > 1$, more policy iterations are needed to compensate the lower number of policy evaluations. That is because with a higher tolerance ϵ, the estimate is not so accurate.

ϵ	policy iterations	policy evaluations
10^{-4}	6	204
10^{-3}	6	158
10^{-2}	6	115
10^{-1}	6	64
10^{0}	6	38
10^{1}	19	19
10^{2}	19	19
10^{3}	19	19
10^{4}	19	19

Table 2: Number of policy iterations and evaluations for each ϵ

Exercise 6

a

Terminate update:

Listing 3: Terminal Q-update

```
sample                    = reward; % replace nan with something appropriate.
pred                      = Q_vals(state_idx, action); % replace nan with something
    appropriate.
td_err                    = sample - pred; % don't change this.
Q_vals(state_idx, action) = Q_vals(state_idx, action) + alph*td_err;% + ... (fill
    in blanks)
```

Non-terminate update:

Listing 4: Non terminal Q-update

```
sample                    = reward + gamm*max(Q_vals(next_state_idx, :)); % replace
    nan with something appropriate
pred                      = Q_vals(state_idx, action); % replace nan with something
    appropriate
td_err                    = sample - pred; % don't change this!
Q_vals(state_idx, action) = Q_vals(state_idx, action) + alph*td_err; % + ... (fill
    in blanks)
```

b

ϵ	α	apple reward	death reward	Score
0.1	0.1	1	-1	0
0.2	0.2	1	-1	2
0.2	0.5	1	-1	158
0.5	0.5	1	-1	1
0.2	0.5	5	-5	2400

Table 3: Attempts for training the snake playing agent

First of all, the default values for ϵ and α were too low, giving 0 score. So I tried to increase them. A low value of ϵ prevents the snake from trying new actions, making it get stuck. A low value of α (the learning rate) makes it harder for the snake to make a progress, as the updates of the Q-values are too slow. The score got better when both ϵ and α were 0.2 (see line 2 from Table 3), but the results were still not satisfactory, meaning that the optimal solution was not reached yet. The best balance was on attempt with $\epsilon = 0.2$ and $\alpha = 0.5$ (see line 3 from Table 3). Even so, the score was just 158. I tried bigger values for ϵ and α (both of them being 0.5 - see line 4 from Table 3), but the score was lower, which means I passed over the optimal value. So another thing that had to be fixed was the reward function. The rewards values needed to be higher, as they make the optimal actions have greater rewards. Therefore, I tried the values 5 for the apple reward and -5 for the death reward and the result was satisfactory (see the last line from Table 3).

c

ϵ	α	apple reward	death reward	Score
0.2	0.5	15	-15	4148

Table 4: Final setting for training the snake playing agent

The best score (4148) was achieved when using $\epsilon = 0.2$ and $\alpha = 0.5$ and the values 15 for the apple reward and -15 for the death reward. The changes in the reward values improved a lot the performance (the bigger the values, the better the score). Considering this, with these values I was able to train a good policy for the Snake game.

d

First of all, there are a lot of combinations that can be done when tuning parameters ϵ and α. And sometimes it can be not very intuitive to find the right combination, as you could have passed over a local maximum, but notice that the score is lower, so you could wrongly try to increase the values, instead of decreasing them. Then there is also the problem that there are 4136 possible states (therefore the number of action-states pairs is even larger) and the number of episodes is only 5000. So it is possible that the training does not include all possible states.

Exercise 7

a

Terminate update:

Listing 5: Terminal Q-weight update

```
target = reward; % replace nan with something appropriate
pred   = Q_fun(weights, state_action_feats, action); % replace nan with something
    appropriate
td_err = target - pred; % don't change this
```

```
weights = weights + alph*td_err*state_action_feats(:, action); % + ... (fill in
    blanks)
```

Non-terminate update:

Listing 6: Non-terminal Q-weight update

```
target = reward + gamm*max(Q_fun(weights, state_action_feats_future)); % replace
    nan with something appropriate
pred   = Q_fun(weights, state_action_feats, action); % replace nan with something
    appropriate
td_err = target - pred; % don't change this
weights = weights + alph*td_err*state_action_feats(:,action);% + ... (fill in
    blanks)
```

b

First of all, one needs to engineer the state-action features. The first feature tells if the agent is getting closer to the apple by taking the action. The L1 distance from the apple to the previous head location is computed and the distance from the apple to the new head location is computed. The difference between these distances is computed, normalized and stored in the first state-action feature. Here, if the snake gets closer to the action, than the state-feature should be positive. Therefore, $w_1 = -1$.

Listing 7: First feature

```
[next_head_loc, next_move_dir] = get_next_info(action, movement_dir, head_loc);
next_head_loc_m = next_head_loc(1);
next_head_loc_n = next_head_loc(2);

% see if the action get the agent closer to the apple
distance = (norm(apple_loc - prev_head_loc) - norm(apple_loc - next_head_loc));
state_action_feats(1, action) = distance/norm(size(grid), 1);
```

The second feature tells is the snake dies after taking the action. The snake dies in the next state if its head hits a wall or if its head is placed on itself. In other words, the snake dies by taking the action if the pixel value of the next head location is 1. So, if that is the case, than the state-action feature will have a value of -1, otherwise it will be 0. Since 0 is the good value (because we don't want the snake to die), then the state-action feature should be positive. Which means that $w_2 = -1$.

Listing 8: Second feature

```
 % see if the snake dies taking the action
if(grid(next_head_loc_m,next_head_loc_n) == 1)
       state_action_feats(2, action) = -1;
else
       state_action_feats(2,action) = 0;
end
```

The last feature says if snake gets to the apple after taking the action. This happens if the pixel value of the next head location is -1. If this is the case, than the feature-state is 1, otherwise it's 0. The good value is 1, so $w_3 = -1$.

Listing 9: Third feature

```
% see if the agent gets the apple by taking the action
if (grid(next_head_loc_m,next_head_loc_n) == -1)
        state_action_feats(3,action) = 1;
else
        state_action_feats(3,action) = 0;
end
```

- $\epsilon = 0.5$, $\alpha = 0.7$, apple reward $= 1$, death reward $= -1$

  This attempt improved the score, as I obtained a value of **28.72**. This means that the agent needed a slightly higher learning rate in order to not get stuck on a local optimum.

- $\epsilon = 0.5$, $\alpha = 0.9$, apple reward $= 1$, death reward $= -1$

  This attempt didn't improved the score, as I obtained still the value **28.72**. This means that a learning rate of 0.7 is good for the agent.

- $\epsilon = 0.2$, $\alpha = 0.7$, apple reward $= 1$, death reward $= -1$

  This attempt didn't improved either the score, as I obtained still the value **28.72**. This means that lowering the greediness factor doesn't make a difference.

- $\epsilon = 0.9$, $\alpha = 0.7$, apple reward $= 1$, death reward $= -1$

  This attempt didn't improved either the score, as I obtained still the value **28.72**. This means that increasing the greediness factor doesn't make a difference either.

- $\epsilon = 0.5$, $\alpha = 0.7$, apple reward $= 50$, death reward $= -50$

  This attempt didn't improved either the score, as I obtained still the value **28.72**. So the values for the reward function seem to not have a great impact on the performance of the agent.

- The first state-action feature

  After seeing that tuning the parameters didn't make any change, I thought that maybe I should change the first state-action feature, as it was the only one that had a value bigger outside the interval [-1, 1]. Therefore, after computing the difference between the 2 distances, I just check if it is bigger than 0. If it is, the state-action feature becomes 0, otherwise it becomes -1. This attempt gave a score of **42.52**.

Listing 10: First feature - modified version

```
distance = (norm(apple_loc - prev_head_loc) - norm(apple_loc - next_head_loc));
if distance > 0
       state_action_feats(1,action) = 0;
else
       state_action_feats(1,action) = -1;
end
```

c

The final setting for the experiment can be seen in Table 5. The state-action features are the ones from Listing 10 (first feature), Listing 8 (second feature) and Listing 9 (third feature). Fine-tuning the parameters didn't really make a difference. What made a difference was engineering the state-action features. This actually makes sense because the state-action features summarize the most important concept of the state.

ϵ	α	apple reward	death reward	Score
0.5	0.7	50	-10	42.52

Table 5: Final setting for training the snake playing agent

w_{01}	w_{02}	w_{03}	w_1	w_2	w_3	f_1	f_2	f_3
-1	-1	-1	2.079	5.149	50.000	[-1, -1, 0]	[-1, -1, 0]	[0, -1, 0]

Table 6: Values for initial weights, final weights and state-action features

References

[1] FMAN45: Machine learning (2022), Lecture 10, Lund University
[2] Reinforcement learning, https://en.wikipedia.org/wiki/Reinforcement_learning