CityU CS5491 Tutorial on MDP and RL

Markov Decision Processes Recap

- A **Markov Decision Process (MDP)** is similar to a state transition system. It has states, actions, a transition function $T(s, a, s')$ specifying the probability an agent ends up in state s' when he takes action a from state s, a distribution over start states, and possibly a set of terminal states. It also has a **reward function** $R(s, a, s')$, which represents the reward that an agent receives for performing action a in state s and ending up in state s'.
- A **q-state** is a (state, action) pair. From a state s, the agent chooses an action a, and then from that q-state (s, a), the (possibly nondeterministic) transition function chooses the resulting state s'.
- The utility of a state is not just the reward associated with that state, it also depends on what is going to happen in the future, so we need to compute utilities for sequences of rewards. We can define the utility of a sequence of rewards $[r_0, r_1, r_2, \ldots]$ as $U([r_0, r_1, r_2, \ldots]) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \ldots$ for some $0 \leq \gamma \leq 1$. If $\gamma \neq 1$, the sum will converge, so the utilities will be finite; this is called temporal **discounting**.
- The **value** of a state s is the total expected future utility given that the agent is currently in state s. Similarly, the **q-value** of a q-state (s, a) is the total expected future utility given that the agent is currently in state s and has just taken action a. Under an optimal policy, the value and q-value are denoted $V^*(s)$ and $Q^*(s, a)$, respectively.
- The **Bellman equations** specify the relationship between $V^*(s)$ and $Q^*(s, a)$:

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

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \big(R(s, a, s') + \gamma V^*(s')\big).$$

- If we plug the second Bellman equation into the first one, we get a recursive definition of $V^*(s)$. We can approximate $V^*(s)$ by $V_k^*(s)$, the optimal value considering only the next k time steps: as $k \to \infty, V_k^* \to V^*(s)$. So, to compute V^*, we can use **value iteration**: initialize $V_0^*(s)$ to 0 for all s, and then given V_i^*, plug it into the Bellman equations to compute V_{i+1}^*, and repeat until convergence. **Policy iteration** is a similar process but updates the policy instead of the values. Often, the policy will converge before the values do.

Reinforcement Learning

- In **reinforcement learning (RL)** an agent gets feedback in the form of rewards, and he wants to learn a policy to maximize his expected utility. In passive RL, the policy is given, and the agent needs to learn the states' values from observation. In active RL, the agent has to choose actions and create a policy.
- In **temporal difference (TD) learning**, the agent estimates V directly from samples, without estimating T and R. It uses an exponentially-weighted moving average: $V_{new} = \alpha V_{sampled} + (1 - \alpha)V_{old}$ for some weight α, known as the *learning rate*.
- For constructing a policy, it is more useful to know Q than to know V. **Q-learning** is sample-based q-value iteration: each time the agent takes an action, he updates his Q values based on the Bellman equations.
- An agent must trade off between **exploration** (to learn a better policy) and **exploitation** (to get the most benefit out of the existing policy). In the **ϵ-greedy** method, with probability ϵ the agent chooses a random action, otherwise he acts according to his current policy. A better alternative is to have an "optimistic" utility function, which adds more utility to states that the agent knows less about.
- **Approximate Q-Learning**. In many contexts, there are too many states to explore all of them and store their information separately. One solution is to encode states (or q-states) as feature vectors and then estimate a state's value as a linear function of its feature vector; this enables the agent to generalize from examples to new states that it has not seen before.

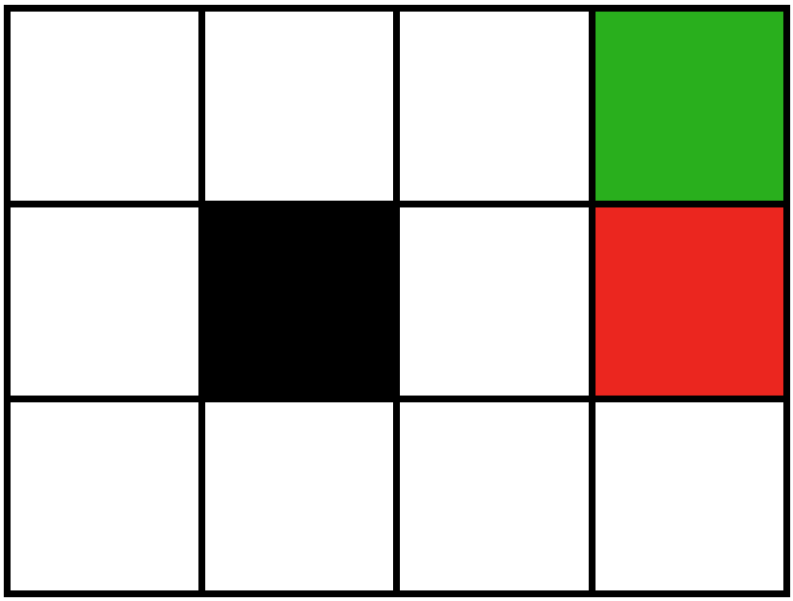
$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

Approximate Q-learning with linear Q-functions:

$$transition = (s, a, r, s')$$
$$\Delta = \left(r + \gamma \max_{a'} Q(s', a')\right) - Q(s,a)$$
$$w_i \leftarrow w_i + \alpha[\Delta] f_i(s,a)$$

1 MDP Exercise: Value Iteration

For today's class the environment we're exploring is a grid world, pictured below.



With this grid world, we define the following Markov decision process $\langle S, A, T, R, \gamma \rangle$:

- S: our states are represented as grid cells, each grid cell is one state in our world;
- A: the actions the robot can take in the world include moving *up*, *down*, *left*, and *right*; so there are 4 total actions that can be taken in this world;
- T: the robot moves as expected with action a from one state (s) to another (s') with probability 0.8. With probability 0.1, the robot moves to either left or right in perpendicular to action a. Here are some examples:
  - If a specifies that the robot should move to the *right*, with probability 0.8 the robot will move right. With probability 0.1 the robot will move up. And with probability 0.1 the robot will move down.
  - If a specifies that the robot should move to the *down*, with probability 0.8 the robot will move down. With probability 0.1 the robot will move right. And with probability 0.1 the robot will move left.
- R: Reward function only depends on the state of the world, i.e., $R(green) = +1$, $R(red) = -1$, and $R(\text{everywhere else}) = -2$.
- γ: for today's class we'll use a fixed discounting factor γ of value of 0.8.

Task today: With our grid world and the specified reward function shown above, compute the value for each state in the grid world using the Bellman equation and the value iteration approach. Before we start with value iteration, you can assume that the value for each state is 0.

To keep everyone on the same page, we're all going to follow the same set of trajectories:

> Note: To best keep track of each state, we will denote each state by it's row and column where the bottom left is the origin $[0, 0]$. For example, the green state at time k can be represented as $s_k = [2, 3]$

	Trajectory 1	Trajectory 2	Trajectory 3	Trajectory 4	Trajectory 5	Trajectory 6
s_0	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
V	$V([0, 0]) = -2$	$V([0, 0]) = -2.32$	$V([0, 0]) = -2.47$	$V([0, 0]) = -3.77$	$V([0, 0]) = -4.89$	$V([0, 0]) = -5.09$
a_0	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_1	$[1, 0]$	$[1, 0]$	$[0, 1]$	$[0, 1]$	$[1, 0]$	$[0, 1]$
V	$V([1, 0]) = -2$	$V([1, 0]) = -3.6$	$V([0, 1]) = -2$	$V([0, 1]) = -3.6$	$V([1, 0]) = -4.88$	$V([0, 1]) = -4.07$
a_1	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_2	$[2, 0]$	$[2, 0]$	$[0, 2]$	$[0, 2]$	$[2, 0]$	$[0, 2]$
V	$V([2, 0]) = -2$	$V([2, 0]) = -3.6$	$V([0, 2]) = -2$	$V([0, 2]) = -2.34$	$V([2, 0]) = -4.88$	$V([0, 2]) = -3.70$
a_2	RIGHT	RIGHT	UP	RIGHT	RIGHT	UP
s_3	$[2, 1]$	$[2, 1]$	$[1, 2]$	$[0, 3]$	$[2, 1]$	$[1, 2]$
V	$V([2, 1]) = -2$	$V([2, 1]) = -3.6$	$V([1, 2]) = -2.28$	$V([0, 3]) = -2.08$	$V([2, 1]) = -3.54$	$V([1, 2]) = -3.06$
a_3	RIGHT	RIGHT	RIGHT	UP	RIGHT	UP
s_4	$[2, 2]$	$[2, 2]$	$[1, 3]$	$[1, 3]$	$[2, 2]$	$[2, 2]$
V	$V([2, 2]) = -2$	$V([2, 2]) = -1.52$	$V([1, 3]) = -1$	$V([1, 3]) = -1$	$V([2, 2]) = -1.66$	$V([2, 2]) = -1.74$
a_4	RIGHT	RIGHT			RIGHT	RIGHT
s_5	$[2, 3]$	$[2, 3]$			$[2, 3]$	$[2, 3]$
V	$V([2, 3]) = 1$	$V([2, 3]) = 1$			$V([2, 3]) = 1$	$V([2, 3]) = 1$

- Trajectory 1
  - $V([2, 3]) = 1$
    - We can determine this value because $R(s_t, a_t) = +1$ for being in the green state and once we get into the state, the world ends (we cannot transition into any further states), meaning that for all a_t, $\left(\gamma \sum_{s' \in S} T(s, a, s')V(s')\right) = 0$.
  - $V(\text{all other states}) = -2$
    - We can determine this value because $R(s_t, a_t) = -2$ and all states initially start out with a value of 0 for all states until after we go through the first iteration.
- Trajectory 2
  - $V([0, 0]) = -2 + \gamma(0.8 \cdot 0 + 0.1 \cdot -2 + 0.1 \cdot -2) = -2 + \gamma(-0.4) = -2 + 0.8 \cdot -0.4 = -2.32$

    - The optimal action in this case would be a move to the RIGHT since the value of state $[0, 1]$ is currently 0 and the value of the state $[1, 0]$ is currently -2.
- $V([1, 0]) == -2 + \gamma(0.8 \cdot -2 + 0.1 \cdot -2 + 0.1 \cdot -2) = -2 + \gamma(-2) = -2 + 0.8 \cdot -2 = -3.6$
    - The optimal action in this case would be a move UP since the value of state $[2, 0]$ is currently -2 and the value of the state $[0, 0]$ is now -2.32.
- ...

RL Exercise: Approximate Q-learning

A self-driving car needs to decide whether to Accelerate (**A**) or Brake (**B**) so as to drive to a location without hitting other cars. It receives a reward of +1 if the car moves and does not hit another car, 0 if it does not move, and −2 if it hits another car. The discount factor $\gamma = 1$.

We want to use Approximate Q-learning to learn a good driving policy for the car. The car has sensors that allow it to observe the distance (**D**) to the nearest object and the current speed (**S**). We decide to create a set of four features: f_{AD}, f_{AS}, f_{BD}, f_{BS}. The first two features are for action A and the second two features are for action B. Q values will be approximated by a linear combination of four features, with four weights (one for each feature): w_{AD}, w_{AS}, w_{BD}, w_{BS}.

Suppose that some learning has already happened such that we have $w_{AD} = 1$, but the other weights are all 0. The learning rate $\alpha = 0.5$. Below is the stream of data the car receives as it drives. Compute the weights after each step.

Observed Data	Weights after seeing data
	$w_{AD} = 1$, $w_{AS} = w_{BD} = w_{BS} = 0$
Initial Sensors: $D = 0$, $S = 2$ Action: A Reward: -2 Final Sensors: $D = 1$, $S = 0$	$s[D = 0, S = 2]$, taken action A, $f_{AD} = 0$, $f_{AS} = 2$, $f_{BD} = f_{BS} = 0$ $Q(s, \text{A}) = w_{AD} \cdot f_{AD} + w_{AS} \cdot f_{AS} + w_{BD} \cdot f_{BD} + w_{BS} \cdot f_{BS}$ $= 1 \cdot 0 + 0 \cdot 2 + 0 \cdot 0 + 0 \cdot 0 = 0$ $s'[D = 1, S = 0]$, taken action A, $f_{AD} = 1$, $f_{AS} = 0$, $f_{BD} = f_{BS} = 0$ $Q(s', \text{A}) = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$ $s'[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0$, $f_{BD} = 1$, $f_{BS} = 0$ $Q(s', \text{B}) = 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$ $\Delta = (-2 + 1.0 \cdot max\big(Q(s', \text{A}), Q(s', \text{B})\big) - 0 = -2 + 1.0(1) - 0 = -1$ $w_{AD} \leftarrow w_{AD} + 0.5 \cdot \Delta \cdot f_{AD}(s, A) = 1 + 0.5 \cdot (-1) \cdot 0 =$ (1) $w_{AS} \leftarrow w_{AS} + 0.5 \cdot \Delta \cdot f_{AS}(s, A) = 0 + 0.5 \cdot (-1) \cdot 2 =$ (-1) $w_{BD} =$ (0), $w_{BS} =$ (0)
Initial Sensors: $D = 1$, $S = 0$ Action: B Reward: 0 Final Sensors: $D = 1$, $S = 0$	$s[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0$, $f_{BD} = 1$, $f_{BS} = 0$ $Q(s, \text{B}) = w_{AD} \cdot f_{AD} + w_{AS} \cdot f_{AS} + w_{BD} \cdot f_{BD} + w_{BS} \cdot f_{BS}$ $= 1 \cdot 0 + (-1) \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 0$ $s'[D = 1, S = 0]$, taken action A, $f_{AD} = 1$, $f_{AS} = 0$, $f_{BD} = f_{BS} = 0$ $Q(s', \text{A}) = 1 \cdot 1 + (-1) \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$ $s'[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0$, $f_{BD} = 1$, $f_{BS} = 0$ $Q(s', \text{B}) = 1 \cdot 0 + (-1) \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$ $\Delta = (0 + 1.0 \cdot max\big(Q(s', \text{A}), Q(s', \text{B})\big) - 0 = 0 + 1.0(1) - 0 = 1$ $w_{AD} =$ (1), $w_{AS} =$ (-1) $w_{BD} \leftarrow w_{BD} + 0.5 \cdot \Delta \cdot f_{BD}(s, B) = 0 + 0.5 \cdot 1 \cdot 1 =$ (0.5) $w_{BS} \leftarrow w_{BS} + 0.5 \cdot \Delta \cdot f_{BS}(s, B) = 0 + 0.5 \cdot 1 \cdot 0 =$ (0)

Given the learned weights, suppose that the sensors read $D = 1$, $S = 1$. Which action would be preferred?

- At state $s[D = 1, S = 1]$, if take action A, $Q(s, \text{A}) = 1 \cdot 1 + (-1) \cdot 1 + 0.5 \cdot 0 + 0 \cdot 0 = 0$
- At state $s[D = 1, S = 1]$, if take action B, $Q(s, \text{B}) = 1 \cdot 0 + (-1) \cdot 0 + 0.5 \cdot 1 + 0 \cdot 1 =$ (0.5)

Hence, action B will be preferred as the corresponding Q-value (i.e., $Q(s, \text{B})$) is greater.