COMS 4030A

Adaptive Computation and Machine Learning

12.8. **Value Iteration Algorithm.**

The Value Iteration Algorithm given below is used to find an optimal policy for an MDP in the case that we know the transition function P and the reward function R.

```
Value Iteration

Given MDP with state transition function $P$ and reward function $R$.

Declare an array $V$ indexed by all the states $s \in S$.
Initialise $V(s) = 0$ for each $s \in S$ (or initialise to small random number).
Choose a discount rate $\gamma$ (usually 1 or 0.9).

Repeat until $\Delta < \epsilon$     (where $\epsilon$ is some small number)
    $\Delta \leftarrow 0$
    for each $s \in S$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s,a,s')\,(R(s,a,s') + \gamma V(s'))$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
```

The output of the Value Iteration Algorithm is the final array V from which the following policy π can be obtained (using also P and R).
For every state s in S,

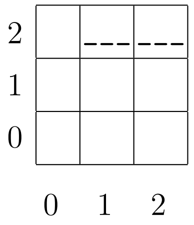
$$\pi(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} P(s,a,s')\,(R(s,a,s') + \gamma V(s'))\,.$$

The final array V should be a close approximation to V^{π^*} and the final policy π should be close to, or exactly, the optimal policy π^*.

The above Value Iteration Algorithm is a method for solving Bellman's optimality equation that uses dynamic programming. For each state s, the value $V(s)$ is updated as in the update formula, using the existing values for $V(s')$.
Note that the values of $P(s,a,s')$ and $R(s,a,s')$ are known for all s, a and s'.

Once all $V(s)$ have been updated, the process starts again: update each $V(s)$, and so on.
The algorithm terminates when the values of $V(s)$ converge, in other words, when updating the values of $V(s)$ there is no change to any $V(s)$, or only a very small change.
This is the purpose of the Δ which keeps track of the greatest change in any of the $V(s)$'s through each iteration.

Example 1. Consider the following 3×3 gridworld:



Each cell in the grid is a state. Let state s_{ij} be the cell in position i horizontally and j vertically.
The sets of states is then $S = \{s_{ij} \mid 0 \leq i \leq 2,\ 0 \leq j \leq 2\}$.
There is a barrier between s_{11} and s_{12} and also between s_{21} and s_{22}.
The set of actions is $A = \{\ell, u, r, d\}$ corresponding to left, up, right, down.
The transition rules are as before: the agent moves 1 cell left if action ℓ is take, 1 cell up if action u is taken, 1 cell right if action r is taken and 1 cell down if action d is taken, except that if an action would take the agent off the grid or into a barrier, then the agent stays in its current state.

The transition function is as follows:
$P(s_{00}, u, s_{01}) = 1$, $P(s_{00}, u, s_{ij}) = 0$ for any s_{ij} other than s_{01}.
$P(s_{00}, r, s_{10}) = 1$, $P(s_{00}, r, s_{ij}) = 0$ for any s_{ij} other than s_{10}.
$\vdots$
$P(s_{11}, u, s_{11}) = 1$, $P(s_{11}, r, s_{ij}) = 0$ for any s_{ij} other than s_{11}.
$\vdots$
$P(s_{12}, r, s_{22}) = 1$, $P(s_{12}, d, s_{ij}) = 0$ for any s_{ij} other than s_{22}.
$\vdots$

There are no probabilistic transitions here, so the MDP is deterministic.

Suppose that the goal of the agent is to start at s_{00} and get to s_{22}. Then a reward function could be the following. Any action that gets the agent to state s_{22} gets a reward of 100, which is written as $R(s_{12}, r, s_{22}) = 100$. Any other action that moves the agent to a new state gets a reward of -1 and any action that would take the agent off the grid or into a barrier gets a reward of -5.

The following table gives the values in the array V, which is initialised to 0:

0	0	0
0	0	0
0	0	0

After one iteration of the Value iteration algorithm, the values of V would be the following:

-1	100	0
-1	-1	-1
-1	-1	-1

Note that the value of s_{22} will always be 0 since the episode ends when the agent arrives in s_{22} and so the agent will never take an action when in s_{22}.

The values in the above table were updated using the value iteration algorithm. For example, the value in state $s_{12} = 100$ was obtained as follows (using $\gamma = 1$):

$$V(s_{12}) \leftarrow \max_{a \in \{\ell,u,r,d\}} \sum_{s' \in S} P(s_{12}, a, s') \left(R(s_{12}, a, s') + \gamma V(s')\right)$$

We must consider all 4 actions: ℓ, u, r, d.

If the agent takes action ℓ, then $P(s_{12}, \ell, s_{02}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{12}, \ell, s') \left(R(s_{12}, \ell, s') + \gamma V(s')\right) = P(s_{12}, \ell, s_{02}) \left(R(s_{12}, \ell, s_{02}) + \gamma V(s_{02})\right) = 1(-1{+}1(0)) = -1.$$

If the agent takes action u, then $P(s_{12}, u, s_{12}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{12}, u, s') \left(R(s_{12}, u, s') + \gamma V(s')\right) = P(s_{12}, u, s_{12}) \left(R(s_{12}, u, s_{12}) + \gamma V(s_{12})\right) = 1(-5{+}1(0)) = -5.$$

If the agent takes action r, then $P(s_{12}, u, s_{22}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{12}, r, s') \left(R(s_{12}, r, s') + \gamma V(s')\right) = P(s_{12}, r, s_{22}) \left(R(s_{12}, r, s_{22}) + \gamma V(s_{22})\right) = 1(100{+}1(0)) = 100.$$

If the agent takes action d, then $P(s_{12}, d, s_{12}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{12}, d, s') \left(R(s_{12}, d, s') + \gamma V(s')\right) = P(s_{12}, d, s_{12}) \left(R(s_{12}, d, s_{12}) + \gamma V(s_{12})\right) = 1(-5{+}1(0)) = -5.$$

The four values are: -1, -5, 100, -5. Clearly, the maximum is 100, so $V(s_{12})$ is updated to 100.

After the next iteration of the Value iteration algorithm, the values of V would be the following:

99	100	0
-2	-2	-2
-2	-2	-2

The remaining iterations would be:

99	100	0
98	-3	-3
-3	-3	-3

99	100	0
98	97	-4
97	-4	-4

99	100	0
98	97	96
97	96	-5

99	100	0
98	97	96
97	96	95

The next iteration of the algorithm would not change any of the V values, so the algorithm would stop. The optimal policy can be determined from the final V table. For example, let's compute the optimal action to take in state s_{00}:

$$\pi(s_{00}) = \operatorname*{argmax}_{a \in \{\ell,u,r,d\}} \sum_{s' \in S} P(s_{00}, a, s')\,(R(s_{00}, a, s') + \gamma V(s'))$$

To evaluate $\pi(s_{00})$, we must consider all 4 actions: ℓ, u, r, d.

If the agent takes action ℓ, then $P(s_{00}, \ell, s_{00}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{00}, \ell, s')\,(R(s_{00}, \ell, s') + \gamma V(s')) = P(s_{00}, \ell, s_{00})\,(R(s_{00}, \ell, s_{00}) + \gamma V(s_{00})) = 1(-5{+}1(97)) = 92.$$

If the agent takes action u, then $P(s_{00}, u, s_{01}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{00}, u, s')\,(R(s_{00}, u, s') + \gamma V(s')) = P(s_{00}, u, s_{01})\,(R(s_{00}, u, s_{01}) + \gamma V(s_{01})) = 1(-1{+}1(98)) = 97.$$

If the agent takes action r, then $P(s_{00}, u, s_{10}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{00}, r, s')\,(R(s_{00}, r, s') + \gamma V(s')) = P(s_{00}, r, s_{10})\,(R(s_{00}, r, s_{10}) + \gamma V(s_{10})) = 1(-1{+}1(96)) = 95.$$

If the agent takes action d, then $P(s_{00}, d, s_{00}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{00}, d, s')\,(R(s_{00}, d, s') + \gamma V(s')) = P(s_{00}, d, s_{00})\,(R(s_{00}, d, s_{00}) + \gamma V(s_{00})) = 1(-5{+}1(97)) = 92.$$

For the 4 actions, the values for the summation are: 92, 97, 95, 92. Clearly, the maximum is 97, hence the argmax is the action u. (The argmax is the argument which gives the maximum value.) Therefore, $\pi(s_{00}) = u$.

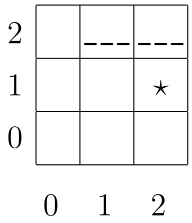
Note that when choosing the optimal action for any state, the formula considers both the reward obtained for taking that action, and the value of being in the resulting state. For example., actions u and r both have reward -1, but being in state s_{01} is better than being in state s_{10}.

Example 2. Consider the 3×3 gridworld of Example 1 with the following change to the state transition function. If the agent moves in the direction of state s_{21} from a neighbouring state, then with probability 0.8 the agent ends up in state s_{12} and with probability 0.2 the agent ends up in state s_{00}. That is,

$P(s_{20}, u, s_{12}) = 0.8$ and $P(s_{20}, u, s_{00}) = 0.2$

$P(s_{11}, r, s_{12}) = 0.8$ and $P(s_{11}, r, s_{00}) = 0.2$

The other transitions are as in Example 1 and the reward function is the same.



Suppose we use Value iteration to find the optimal policy. As before we start with a table for the V function, which is initialised to 0:

0	0	0
0	0	0
0	0	0

After one iteration of the algorithm, the values of V would be the following:

-1	100	0
-1	-1	-1
-1	-1	-1

After the next iteration, the values of V would be the following:

99	100	0
-2	78.8	-2
-2	-2	78.8

To see how we get these values, let's calculate the update for state s_{11}, i.e.,

$$V(s_{11}) \leftarrow \max_{a \in \{\ell,u,r,d\}} \sum_{s' \in S} P(s_{11}, a, s')\,(R(s_{11}, a, s') + \gamma V(s'))$$

We must consider all 4 actions: ℓ, u, r, d. (We use $\gamma = 1$.)

If the agent takes action ℓ, then $P(s_{11}, \ell, s_{01}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{11}, \ell, s')\,(R(s_{11}, \ell, s') + \gamma V(s')) = P(s_{11}, \ell, s_{01})\,(R(s_{11}, \ell, s_{01}) + \gamma V(s_{01})) = 1(-1{+}1(-1)) = -2.$$

If the agent takes action u, then $P(s_{11}, u, s_{11}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{11}, u, s')\,(R(s_{11}, u, s') + \gamma V(s')) = P(s_{11}, u, s_{11})\,(R(s_{11}, u, s_{11}) + \gamma V(s_{11})) = 1(-5{+}1(-1)) = -6.$$

If the agent takes action r, then $P(s_{11}, r, s_{12}) = 0.8$, $P(s_{11}, r, s_{00}) = 0.2$ and all other probabilities are 0, so

$$\begin{aligned}
&\sum_{s' \in S} P(s_{11}, r, s')\,(R(s_{11}, r, s') + \gamma V(s')) \\
&= P(s_{11}, r, s_{12})\,(R(s_{11}, r, s_{12}) + \gamma V(s_{12})) + P(s_{11}, r, s_{00})\,(R(s_{11}, r, s_{00}) + \gamma V(s_{00})) \\
&= 0.8(-1 + 1(100)) + 0.2(-1 + 1(-1)) \\
&= 78.8.
\end{aligned}$$

If the agent takes action d, then $P(s_{11}, d, s_{10}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{11}, d, s')\,(R(s_{11}, d, s') + \gamma V(s')) = P(s_{11}, d, s_{10})\,(R(s_{11}, d, s_{10}) + \gamma V(s_{10})) = 1(-1{+}1(-1)) = -2.$$

The four values are: -2, -6, 78.8, -2. The maximum is 78.8, so $V(s_{11})$ is updated to 78.8.

The remaining iterations would be:

99	100	0
98	78.6	77.8
-3	77.8	78.6

99	100	0
98	97	77.6
97	77.6	78.4

99	100	0
98	98.4	96
97	96	98.4

99	100	0
98	98.4	97.4
97	97.4	98.4

The next iteration of the algorithm should not change any of the V values, so the algorithm would stop. The optimal policy can be determined from the final V table. Let's calculate the optimal action to take in state s_{11}, i.e.,

$$\pi(s_{11}) = \underset{a \in \{\ell, u, r, d\}}{\operatorname{argmax}} \sum_{s' \in S} P(s_{11}, a, s')\,(R(s_{11}, a, s') + \gamma V(s'))$$

To evaluate $\pi(s_{11})$, we must consider all 4 actions.
If the agent takes action ℓ, then $P(s_{11}, \ell, s_{01}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{11}, \ell, s')\,(R(s_{11}, \ell, s') + \gamma V(s')) = P(s_{11}, \ell, s_{01})\,(R(s_{11}, \ell, s_{01}) + \gamma V(s_{01})) = 1(-1{+}1(98)) = 97.$$

If the agent takes action u, then $P(s_{11}, u, s_{11}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{11}, u, s')\,(R(s_{11}, u, s') + \gamma V(s')) = P(s_{11}, u, s_{11})\,(R(s_{11}, u, s_{11}) + \gamma V(s_{11})) = 1(-5{+}1(98.4)) = 93.4.$$

If the agent takes action r, then $P(s_{11}, r, s_{12}) = 0.8$, $P(s_{11}, r, s_{00}) = 0.2$ and all other probabilities are 0, so

$$\begin{aligned}
&\sum_{s' \in S} P(s_{11}, r, s')\left(R(s_{11}, r, s') + \gamma V(s')\right) \\
&= P(s_{11}, r, s_{12})\left(R(s_{11}, r, s_{12}) + \gamma V(s_{12})\right) + P(s_{11}, r, s_{00})\left(R(s_{11}, r, s_{00}) + \gamma V(s_{00})\right) \\
&= 0.8(-1 + 1(100)) + 0.2(-1 + 1(97)) \\
&= 98.4.
\end{aligned}$$

If the agent takes action d, then $P(s_{11}, d, s_{10}) = 1$ and all other probabilities are 0, so

$$\sum_{s' \in S} P(s_{11}, d, s')\left(R(s_{11}, d, s') + \gamma V(s')\right) = P(s_{11}, d, s_{10})\left(R(s_{11}, d, s_{10}) + \gamma V(s_{10})\right) = 1(-1{+}1(97.4)) = 96.4.$$

For the 4 actions, the values for the summation are: 97, 93.4, 98.4, 96.4. The maximum is 98.4, hence the argmax is the action r. Therefore, $\pi(s_{11}) = r$.

12.9. **Sarsa and Q-learning.**

Consider the case of an agent acting in some environment for which it does not know the state transition function or reward function. Each time the agent takes an action, the environment will return it a reward and new state, but the function that determines these is unknown to the agent.

To find an optimal policy in this case, we make use of a Q-**function**.
The Q-function is similar to the V-function used in value iteration.

Suppose we have some MDP and a policy π.
Then $Q^\pi : S \times A \to \mathbb{R}$ is defined as follows:
For each state $s \in S$ and action $a \in A$, $Q^\pi(s, a)$ is the expected return for the agent if it takes action a first, and then follows policy π until the episode ends (or continuously).
(Recall that the return is the sum of all rewards obtained in the episode.)

The algorithms Sarsa and Q-learning below use a Q-function to find the optimal policy π^* for an MDP. Both algorithms require an array Q, indexed by the set of all states and all actions, that stores the value of $Q(s, a)$ for all s, a. The array is initialised to zero (or small random values).

The idea is that the agent generates a number of episodes in the environment, taking actions and collecting rewards as it goes along, until eventually it reaches a terminal state. For each state s that the agent visits, if it takes action a, then the value of $Q(s, a)$ is updated according to the reward it gets and the resulting state. In each subsequent episode the agent uses the information it has learned about which actions are good in which states. However, the agent is still allowed to take some random actions in order to explore the environment.

Once the algorithm has been run over a large number of episodes, the values in Q can be used to define a policy as follows:

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q(s, a).$$

That is, if the agent is in state s, it takes the action a that gives the maximum value of $Q(s, a)$. If the algorithm has been run on enough episodes, then this action should be optimal.

We shall need the notion of an **ϵ-greedy** policy determined by Q, which is the following:

$$\pi(s) = \begin{cases} \operatorname*{argmax}_{a \in A} Q(s, a), & \text{with probability } 1 - \epsilon, \\ \text{random action}, & \text{with probability } \epsilon, \end{cases}$$

In the ϵ-greedy policy above, ϵ is a small value, usually $\epsilon = 0.1$.
If the agent is in state s, the ϵ-greedy policy chooses the best action determined by Q most of the time, but with some small probability (namely, ϵ) it chooses a random action.
This randomness allows the agent to explore different actions in different states, in the hopes of finding a better action and thereby a better path to the goal.

An episode looks like this:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots, r_{n-1}, s_{n-1}, a_{n-1}, r_n, s_n$$

s_0 is the start state, and a_0 is the first action taken.
The environment returns reward r_1 and new state s_1.
Based on the state s_1, a new action a_1 is chosen using the ϵ-greedy policy.
The algorithm Sarsa below uses the information s_0, a_0, r_1, s_1, a_1 to update the value of $Q(s_0, a_0)$ (hence the name Sarsa).

After the above update, the action a_1 is taken, resulting in reward r_2 and new state s_2.
Based on s_2 a new action a_2 is chosen. Then, using the information in s_1, a_1, r_2, s_2, a_2, the value of $Q(s_1, a_1)$ is updated. This continues until the end of the episode, and then a new episode is started.

```
Sarsa

Declare an array $Q$ indexed by all states $s \in S$ and actions $a \in A$.
Initialise array $Q(s,a) = 0$ for all $s \in S$ and $a \in A$.
Choose a discount rate $\gamma$ (usually 1 or 0.9).
Choose a learning rate $\alpha$ (where $0 \leq \alpha \leq 1$).

Repeat (for each episode)
    Initialise $s$
    Choose action $a$ using $\epsilon$-greedy policy determined by $Q$
    Repeat (for each step of episode until $s$ is a terminal state)
        Take action $a$ and observe reward $r$ and new state $s'$.
        Choose action $a'$ from $s'$ using $\epsilon$-greedy policy determined by $Q$.
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s', \; a \leftarrow a'$
```

In the Sarsa algorithm, if an agent is in state s, takes action a and ends up in a terminal state, then we set $Q(s', a') = 0$ in the update for $Q(s, a)$, as follows:

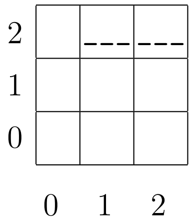
$$Q(s,a) \leftarrow Q(s,a) + \alpha[r - Q(s,a)].$$

This is necessary because in the terminal state no further actions are possible.

The output of Sarsa is the final array Q, from which the following policy π is determined. For every state s in S,

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q(s,a).$$

Example 3. Consider again the 3×3 grid-world MDP of Example 1.



Suppose the first episode of Sarsa starts as follows: $s_{00}, u, -1, s_{01}$.

Before we can update $Q(s_{00}, u)$ we need to choose the next action.

By the ϵ-greedy policy this could be any action, let's suppose it's ℓ.

Thus, our episode starts as: $s_{00}, u, -1, s_{01}, \ell$.

The update to $Q(s_{00}, u)$ is:

$$Q(s_{00}, u) \leftarrow Q(s_{00}, u) + \alpha[-1 + \gamma Q(s_{01}, \ell) - Q(s_{00}, u)].$$

If we take $\alpha = 0.3$, $\gamma = 0.9$, then $Q(s_{00}, u) \leftarrow 0 + 0.3(-1 + 0.9(0) - 0) = -0.3$.

Now the agent takes the action ℓ, which results in a reward of -5 and new state s_{01}, so the episode is now:

$$s_{00}, u, -1, s_{01}, \ell, -5, s_{01}$$

Again, before we can update $Q(s_{01}, \ell)$ we must choose a new action using the ϵ-greedy policy, say action r is chosen, so the episode is currently:

$$s_{00}, u, -1, s_{01}, \ell, -5, s_{01}, r$$

The update to $Q(s_{01}, \ell)$ is $Q(s_{01}, \ell) \leftarrow 0 + 0.3(-5 + 0.9(0) - 0) = -1.5$.

Continue this way until the episode ends and repeat with a large number of episodes.

The following algorithm is very similar to Sarsa, but uses a different update rule for $Q(s, a)$.

Q-LEARNING

```
Declare an array $Q$ indexed by all states $s \in S$ and actions $a \in A$.
Initialise array $Q(s,a) = 0$ for all $s \in S$ and $a \in A$.
Choose a discount rate $\gamma$ (usually 1 or 0.9).
Choose a learning rate $\alpha$ (where $0 \leq \alpha \leq 1$).

**Repeat** (for each episode)
    Initialise $s$
    **Repeat** (for each step of episode until $s$ is a terminal state)
        Choose action $a$ using $\epsilon$-greedy policy determined by $Q$.
        Take action $a$ and observe reward $r$ and new state $s'$.
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a)]$
        $s \leftarrow s'$
```

In the Q-LEARNING algorithm, if an agent is in state s, takes action a and ends up in a terminal state, then we set $\max\limits_{a' \in A} Q(s', a') = 0$ in the update for $Q(s, a)$, as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r - Q(s, a)].$$

As for SARSA, this is necessary because in the terminal state no further actions are possible.

The output of the Q-LEARNING algorithm is the final array Q, from which the following policy π determined. For every state s in S,

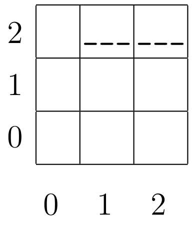
$$\pi(s) = \underset{a \in A}{\operatorname{argmax}}\, Q(s, a).$$

The difference between Q-LEARNING and SARSA is in the update of $Q(s, a)$ values.
Suppose a new episode is started in Q-LEARNING. The agent starts in state s_0, takes action a_0, gets reward r_1 and the new state is s_1, giving the sequence: s_0, a_0, r_1, s_1.
Based on this information the update for $Q(s_0, a_0)$ in the Q-LEARNING algorithm is:

$$Q(s_0, a_0) \leftarrow Q(s_0, a_0) + \alpha[r_1 + \gamma \max_{a' \in A} Q(s_1, a') - Q(s_0, a_0)]$$

For this update it is necessary to calculate $\max\limits_{a' \in A} Q(s_1, a')$, using the current Q-values.

Example 4. Consider once more the 3×3 grid-world MDP in Example 1 above.



Suppose the first episode of Q-LEARNING starts $s_{00}, u, -1, s_{01}$.
We update $Q(s_{00}, u)$ as follows:

$$Q(s_{00}, u) \leftarrow Q(s_{00}, u) + \alpha[-1 + \gamma \max_{a' \in \{\ell, u, r, d\}} Q(s_{01}, a') - Q(s_{00}, u)]$$

Since this is the first episode, we have $Q(s_{00}, u) = 0$, $Q(s_{01}, \ell) = 0$, $Q(s_{01}, u) = 0$, $Q(s_{01}, r) = 0$ and $Q(s_{01}, d) = 0$, so $\max\limits_{a' \in \{\ell, u, r, d\}} Q(s_{01}, a') = 0$.

If we use $\alpha = 0.3$ and $\gamma = 0.9$, then $Q(s_{00}, u) \leftarrow 0 + 0.3(-1 + 0.9(0) - 0) = -0.3$.

Suppose a number of episodes have been run, and a new episode starts at s_{00} with first action r.
This gives a reward of -1 and new state of s_{10}. Thus, the episode starts: $s_{00}, r, -1, s_{10}$.
To update $Q(s_{00}, r)$ requires the current value for $Q(s_{00}, r)$; let us suppose that $Q(s_{00}, r) = -2$, and also all the current Q-values for s_{10}, say:

$$Q(s_{10}, \ell) = -0.3, \quad Q(s_{10}, u) = -0.1, \quad Q(s_{10}, r) = -0.3, \quad Q(s_{10}, d) = -1.5.$$

The update for $Q(s_{00}, r)$ is then:

$$\begin{aligned} Q(s_{00}, r) &\leftarrow Q(s_{00}, r) + \alpha[r + \gamma \max_{a' \in \{\ell, u, r, d\}} Q(s_{10}, a') - Q(s_{00}, r)] \\ &= -2 + 0.3[-1 + 0.9(-0.1) - (-2)] \\ &= -1.727. \end{aligned}$$

EXERCISES

(1) Consider Example 1 above. Try to verify some of the values given in the V tables. Also, use the final table to find $\pi(s_{10})$.

(2) Consider Example 2 above. Try to verify some of the values given in the V tables. Also, use the final table to find $\pi(s_{20})$.

(3) In Example 2, the optimal policy for an agent starting in state s_{00} is to go up, then up again, then right and right. Suppose the state transition probabilities are changed slightly to:
$P(s_{20}, u, s_{12}) = 0.9$ and $P(s_{20}, u, s_{00}) = 0.1$
$P(s_{11}, r, s_{12}) = 0.9$ and $P(s_{11}, r, s_{00}) = 0.1$
What do you think the optimal policy would be in this case? Try do some calculations to show this.

(4) Consider the following MDP: the set of states is $S = \{s_0, s_1, s_2, s_3\}$ and the set of actions available at each state is $A = \{a_0, a_1\}$. Each action taken has a reward of -1, i.e., $R(s_i, a_0, s_j) = -1$ and $R(s_i, a_1, s_j) = -1$ for all s_i, s_j. The start state is s_0 and the terminal state is s_3.
The transition probabilities for state s_0 are:

$$\begin{aligned} P(s_0, a_0, s_0) &= 0.2 \qquad & P(s_0, a_1, s_0) &= 0.3 \\ P(s_0, a_0, s_1) &= 0.8 & P(s_0, a_1, s_1) &= 0.1 \\ P(s_0, a_0, s_2) &= 0 & P(s_0, a_1, s_2) &= 0.6 \\ P(s_0, a_0, s_3) &= 0 & P(s_0, a_1, s_3) &= 0 \end{aligned}$$

Suppose you are using the Value Iteration algorithm and the current values for V are: $V(s_0) = -5$, $V(s_1) = -2$, $V(s_2) = -1$ and $V(s_3) = 0$.
(a) Do the Value Iteration update for $V(s_0)$, using $\gamma = 1$.
(b) Suppose this update was the last one done. What is $\pi(s_0)$?

(5) Consider the following MDP: the set of states is $S = \{s_0, s_1, s_2, s_3\}$ and the set of actions available at each state is $A = \{a_0, a_1\}$. Each episode of the MDP starts in s_0 and terminates in s_3. You do not know the transition probabilities or the reward function of the MDP, so you are using SARSA and Q-LEARNING to find the optimal policy. Suppose the current Q-values obtained from previous episodes are:

$$Q(s_0, a_0) = 2.6 \qquad Q(s_0, a_1) = 2.5$$

$$Q(s_1, a_0) = -1 \qquad Q(s_1, a_1) = -2$$

$$Q(s_2, a_0) = 1.5 \qquad Q(s_2, a_1) = 1.7$$

$$Q(s_3, a_0) = 0 \qquad Q(s_3, a_1) = 0$$

Suppose the next episode is as follows:

$$s_0, a_0, 2, s_1, a_1, -1, s_1, a_1, -2, s_0, a_1, 3, s_2, a_0, 2, s_3$$

(a) Do all the SARSA updates for this episode, using $\alpha = 0.3$ and $\gamma = 0.9$.

(b) Based on the updated Q-values in (a), give the final policy π determined by Q, i.e, give $\pi(s_0)$, $\pi(s_1)$, $\pi(s_2)$ and $\pi(s_3)$.

(c) Do all the Q-LEARNING updates for this episode, using $\alpha = 0.3$ and $\gamma = 0.9$.

(d) Based on the updated Q-values in (c), give the final policy π determined by Q.