Tutorial 08 — Answers

(Version 1.2)

1. The Bellman equation for state $(1,1)$ is:

$$\begin{aligned}
U((1,1)) &= R((1,1)) + \gamma \max_{a \in \{Up,Down,Left,Right\}} \sum_{s'} Pr(s'|(1,1),a)U(s') \\
&= R((1,1)) \\
&\quad + \gamma \max(Pr((1,2)|(1,1),Up)U((1,2)) \\
&\qquad\qquad + Pr((2,1)|(1,1),Up)U((2,1)) \\
&\qquad\qquad + Pr((1,1)|(1,1),Up)U((1,1)), \\
&\qquad\quad Pr((2,1)|(1,1),Right)U((2,1)) \\
&\qquad\qquad + Pr((1,2)|(1,1),Right)U((1,2)) \\
&\qquad\qquad + Pr((1,1)|(1,1),Right)U((1,1)), \\
&\qquad\quad Pr((1,1)|(1,1),Down)U((1,1)) \\
&\qquad\qquad + Pr((2,1)|(1,1),Down)U((2,1)) \\
&\qquad\qquad + Pr((1,1)|(1,1),Down)U((1,1)), \\
&\qquad\quad Pr((1,1)|(1,1),Left)U((1,1)) \\
&\qquad\qquad + Pr((1,2)|(1,1),Left)U((1,2)) \\
&\qquad\qquad + Pr((1,1)|(1,1),Left)U((1,1)),
\end{aligned}$$

   In words, the utility of $(1,1)$ is the reward for being in $(1,1)$, plus the discounted (because it will be a step in the future) expected utility that will be gained by taking action in $(1,1)$. Since the agent has a choice about which action to take, we compute the expected utility of all the possible actions, and use the maximum expected utility.

2. Value iteration uses the Bellman equation as a rule for updating $U(s)$ based on the current estimates of the $U(s')$ of the neighbouring states:

$$U((1,1)) \leftarrow R((1,1)) + \gamma \max_{a \in \{Up,Down,Left,Right\}} \sum_{s'} Pr(s'|(1,1),a)U(s')$$

   One round of value iteration involves applying this update rule once.

The full calculation of $U(s)$ using one iteration of the Bellman update is:

$$\begin{aligned}
U((1,1)) \leftarrow\ & R((1,1)) \\
& + \gamma \max(Pr((1,2)|(1,1), Up)U((1,2)) \\
& \quad + Pr((2,1)|(1,1), Up)U((2,1)) \\
& \quad + Pr((1,1)|(1,1), Up)U((1,1)), \\
& \quad Pr((2,1)|(1,1), Right)U((2,1)) \\
& \quad + Pr((1,2)|(1,1), Right)U((1,2)) \\
& \quad + Pr((1,1)|(1,1), Right)U((1,1)), \\
& \quad Pr((1,1)|(1,1), Down)U((1,1)) \\
& \quad + Pr((2,1)|(1,1), Down)U((2,1)) \\
& \quad + Pr((1,1)|(1,1), Down)U((1,1)), \\
& \quad Pr((1,1)|(1,1), Left)U((1,1)) \\
& \quad + Pr((1,2)|(1,1), Left)U((1,2)) \\
& \quad + Pr((1,1)|(1,1), Left)U((1,1))
\end{aligned}$$

For the probability and utility values that appear in the Bellman update for $(1,1)$, Question 4 in Tutorial 7 gives us the following values (I'm using the values after three runs), for utility:

$$\begin{aligned}
&U((1,1)) = 0.71 \\
&U((1,2)) = 0.75 \\
&U((3,3)) = 0.912
\end{aligned}$$

and for probability:

$P((1,2)|(1,1), Up)$: 0.75
$P((1,1)|(1,1), Up)$: 0.25
$P((4,3)|(3,3), Right)$: 0.6
$P((3,2)|(3,3), Right)$: 0.4

All other values are zero. If we only include the elements with non-zero values the Bellman update becomes:

$$\begin{aligned}
U((1,1)) \leftarrow\ & R((1,1)) \\
& + \gamma(Pr((1,2)|(1,1), Up)U((1,2)) \\
& \quad + Pr((2,1)|(1,1), Up)U((2,1)) \\
& \quad + Pr((1,1)|(1,1), Up)U((1,1)))
\end{aligned}$$

because the fixed policy used to get the estimates only ever had the agent move Up in $(1,1)$. This gives:

$$\begin{aligned}
&U((1,1)) \leftarrow -0.04 + \gamma((0.75 \times 0.75) + (0 \times 0) + (0.25 \times 0.71)) \\
&U((1,1)) \leftarrow 0.7
\end{aligned}$$

since $\gamma = 1$, and that is the result of using the Bellman update.

Similarly for $(3,3)$:

$$\begin{aligned}U((3,3)) \leftarrow R((3,3)) \\ + \gamma(Pr((4,3)|(3,3), Right)U((4,3)) \\ + Pr((3,3)|(3,3), Right)U((3,3)) \\ + Pr((3,2)|(3,3), Right)U((3,2)))\end{aligned}$$

This gives:

$$\begin{aligned}U((3,3)) &\leftarrow -0.04 + \gamma(0.6 \times 1 + 0 \times 0.912 + 0.4 \times 0.88) \\ U((3,3)) &\leftarrow 0.912\end{aligned}$$

just as before the update.

There are a few things to note here.

First, the target value of $U((1,1))$, that is the utility of the state under the optimal policy for this environment, is 0.705, and the target value of $U((3,3))$ is 0.918. So even using these very simple reinforcement learning methods (both utility estimation, and a single application of ADP) give us a pretty accurate estimate rather quickly. (Of course, this is a very simple environment.)

Second, I only computed the result of the update on $(1,1)$ and $(3,3)$. A proper application of ADP would be to do the update for every state. Clearly, for many of the states, we would have utility estimates of 0, and probability estimates of 0 (for example if we did the update for $(4,1)$), so most of the updates would just compute a utility that was the same as the reward. However, for any state that the agent encountered on a run, we will get a new updated value of the utility which should be more accurate than the value obtained by direct utility estimation[1].

Third, I used the values after 3 runs. That was an arbitrary choice. You could run the Bellman update after a single run. You could run the update after a single action. The update will always incorporate into a new utility estimate for a state all of the information you have about probability and utility that is relevant to the state. If none of that information has changed for a given state, then the update won't provide any new utility value for the state. So, we have a familiar dilemma — do we run the update frequently, and waste computational resources (running updates that don't change utility values), or run them infrequently, and not have access to the best possible utility estimates?

Fourth, the question only asked that you do one round of value iteration. If you were really programming ADP, you would run this update once for every state (as discussed above). Then you would run it again and again (for every state each time) until there was no change in *any* utility value.

3. The temporal difference update rule is:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

If we apply this to $(1,1)$, $(1,1)$ is the state s. We need to decide on a state s', and we will take this to be $(1,2)$. Then we have:

$$\begin{aligned}U^\pi((1,1)) &\leftarrow U^\pi((1,1)) + \alpha(R((1,1)) + \gamma U^\pi((1,2)) - U^\pi((1,1))) \\ &\leftarrow 0.71 + 0.1(-0.04 + 1 \times 0.75 - 0.71) \\ &\leftarrow 0.71 + 0.1(0) \\ &\leftarrow 0.71\end{aligned}$$

[1] More precisely, it might not change but it if it does change it should be to a value that is a better estimate.

so $U((1,1)$ does not change in this case.

For $(3,3)$ we will assume that s' is $(4,3)$, so:

$$\begin{aligned} U^\pi((3,3)) &\leftarrow U^\pi((3,3)) + \alpha(R((3,3)) + \gamma U^\pi((4,3)) - U^\pi((3,3))) \\ &\leftarrow 0.912 + 0.1(-0.04 + 1 \times 1 - 0.912) \\ &\leftarrow 0.912 + 0.1(0.048) \\ &\leftarrow 0.917 \end{aligned}$$

which improves the approximation we have for $U((3,3))$.

Note that in an agent that was using temporal difference learning, we do not have a choice about s'. Rather the update is executed when the agent gets to s', having executed the action defined by π in s.

4. The Q-learning update is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

The intuition is that $Q(s,a)$ has two components, the reward for state s, and the Q value of the states/action pairs that the agent can obtain once it leaves s, the $Q(s',a')$. That is the reason for the:

$$R(s) + \gamma \max_{a'} Q(s',a')$$

bit.

The rest is just how we do the update. Rather than making the new $Q(s,a)$ the bit computed above, we look at the difference between that and the current $Q(s,a)$ and update with a fraction α of this:

$$\alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

We do that for the same reason we do any gradient descent, that is that making small steps smoothes out noise.

5. (a) Given the run, the only action that is associated with $(1,3)$ is $Right$, so the only state/action pair that involves $(1,3)$ is:

$$((1,3), Right)$$

and similarly the state/action representation of $(3,3)$ is:

$$((3,3), Right)$$

(b) For this question, we'll start with $(3,3)$. The only action we have assocated with $(3,3)$ is $Right$, so, for the first run, we have:

$$Q((3,3), Right)) \leftarrow Q((3,3), Right) + \alpha \left(R((3,3)) + \gamma \max_{a'} Q((4,3), a') - Q((3,3), Right) \right)$$

Given that no actions are taken in state $(4,3)$, we might ask what possible value we could possibly give to:

$$\max_{a'} Q((4,3), a')$$

I think you can imagine either running an update for $(4,3)$ where the reward of 1 gets associated with each $((4,3),a')$ pair, or you can just say that the Q-value of any action in $(4,3)$ will be 1. The latter is simpler:

$$\begin{aligned}Q((3,3),Right)) &\leftarrow Q((3,3),Right) + \alpha\left(R((3,3)) + \gamma \max_{a'} Q((4,3),a') - Q((3,3),Right)\right)\\ &\leftarrow 0 + 0.1(-0.04 + 1 \times 1 - 0)\\ &\leftarrow 0.096\end{aligned}$$

For the second run we have:

$$\begin{aligned}Q((3,3),Right)) &\leftarrow Q((3,3),Right) + \alpha\left(R((3,3)) + \gamma \max_{a'} Q((4,3),a') - Q((3,3),Right)\right)\\ &\leftarrow 0.096 + 0.1(-0.04 + 1 - 0.096)\\ &\leftarrow 0.1824\end{aligned}$$

Now, turning to $(1,3)$, there are three updates because the first run visits the state twice. For the first update we have:

$$Q((1,3),Right) \leftarrow Q((1,3),Right) + \alpha\left(R((1,3)) + \gamma \max_{a'} Q((1,3),a') - Q((1,3),Right)\right)$$

and since there is only one action associated with $(1,3)$, which is the successor state to $(1,3)$ at the first update, this becomes:

$$Q((1,3),Right) \leftarrow Q((1,3),Right) + \alpha\left(R((1,3)) + \gamma Q((1,3),Right) - Q((1,3),Right)\right)$$

Since all utilities are initially 0 (since we have no estimates), we have:

$$\begin{aligned}Q((1,3),Right) &\leftarrow Q((1,3),Right) + \alpha\left(R((1,3)) + \gamma Q((1,3),Right) - Q((1,3),Right)\right)\\ &\leftarrow 0 + 0.1(-0.04 + 0 - 0)\\ &\leftarrow -0.004\end{aligned}$$

The second update is then:

$$Q((1,3),Right) \leftarrow Q((1,3),Right) + \alpha\left(R((1,3)) - Q((1,3),Right)\right)$$

because at that first transition to $(2,3)$ there are not yet any state/action pairs for $(2,3)$, so there is nothing to maximise over, then:

$$\begin{aligned}Q((1,3),Right) &\leftarrow Q((1,3),Right) + \alpha\left(R((1,3)) - Q((1,3),Right)\right)\\ &\leftarrow -0.004 + 0.1(-0.04 - (-0.004))\\ &\leftarrow -0.004 + 0.1(-0.036)\\ &\leftarrow -0.0076\end{aligned}$$

The third update will be a bit more complex because when the agent moves from $(1,3)$ to $(2,3)$, it has already visited $(2,3)$ and so has a utility value for it. So we first have to compute the relevant Q-value(s) from the first run.

Now, the first run involves three visits to $(2,3)$, and hence three updates. Just as for $(1,3)$, there is only one state/action pair, and that is $((2,3),Right)$. the first update is just like the first updates for $(1,3)$, so:

$$\begin{aligned}Q((2,3),Right) &\leftarrow Q((2,3),Right) + \alpha\left(R((2,3)) + \gamma Q((2,3),Right) - Q((2,3),Right)\right)\\ &\leftarrow 0 + 0.1(-0.04 + 0 - 0)\\ &\leftarrow -0.004\end{aligned}$$

Then the second update will be analogous, except that $Q((2,3), Right)$ now has a value:

$$\begin{aligned} Q((2,3), Right) &\leftarrow Q((2,3), Right) + \alpha\left(R((2,3)) + \gamma Q((2,3), Right) - Q((2,3), Right)\right) \\ &\leftarrow -0.004 + 0.1(-0.04 + (-0.004 - (-0.004))) \\ &\leftarrow -0.004 + 0.1(-0.04 + 0) \\ &\leftarrow -0.008 \end{aligned}$$

Then the last update is like the second update for $(1,3)$ in that it involves moving to state for which there are no state/action pairs yet:

$$\begin{aligned} Q((2,3), Right) &\leftarrow Q((2,3), Right) + \alpha\left(R((2,3)) - Q((2,3), Right)\right) \\ &\leftarrow -0.008 + 0.1(-0.04 - (-0.008)) \\ &\leftarrow -0.008 + 0.1(-0.032) \\ &\leftarrow -0.0112 \end{aligned}$$

With this information, we can establish the value of $Q((1,3), Right)$ after the second run:

$$Q((1,3), Right) \leftarrow Q((1,3), Right) + \alpha\left(R((1,3)) + \gamma Q((2,3), Right) - Q((1,3), Right)\right)$$

since $Q((2,3), Right)$ is the only state/action pair relating to state $(2,3)$ (or, equivalently, the only one with non-zero Q-value), so:

$$\begin{aligned} Q((1,3), Right) &\leftarrow Q((1,3), Right) + \alpha\left(R((1,3)) + \gamma Q((2,3), Right) - Q((1,3), Right)\right) \\ &\leftarrow -0.0076 + 0.1(-0.04 + (-0.0112) - (-0.0076)) \\ &\leftarrow -0.0076 + 0.1(-0.0436) \\ &\leftarrow -0.01196 \end{aligned}$$

Of course, the value for $((3,3), Right)$ after the first run will affect the value of $((2,3), Right)$ in the second run, and then that will affect the value of $((1,3), Right)$ on the third run. And so on.

6. The SARSA update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s', a') - Q(s,a))$$

so the only difference is that the update does not maximise over the choice of possible actions in state s', because the update is run after the agent has committed to one of them.

7. In this particular case, because we have only one action in every state, there will be no difference between the updates under SARSA and those we saw above for Q-learning.

   However, this is not generally true. Because SARSA updates with the Q-value of the actual action taken in s', when that action is not the one with the highest Q-value. For example, when the agent is exploring rather than exploiting then the value $Q(s', a')$ used to update in SARSA will be smaller than that used in Q-learning because it will not be the maximum of the Q values of the possible state/action pairs for state s'.

Version list

- Version 1.0, March 20th 2019.
- Version 1.1, March 24th 2019.
- Version 1.2, February 9th 2021.