Page 1, section "Value, state and optimality"

We defined value as an expected total reward obtainable from the state. In formal way, value of the state is: $V(s) = \mathbb{E}[\sum_{t=0}^{\infty} r_t \gamma^t]$, where r_t is the local reward obtained at the step t of the episode.

Page 2, section "Bellman equation of optimality"

Lets start with deterministic case, when all our actions have 100% guaranteed outcome. Imagine our agent observes state s_0 and has N available actions. Every action leads to another state $s_1 \dots s_N$ with respective reward $r_1 \dots r_N$. Also assume that we know the values V_i of all states connected to the state s_0.

If we fix the action and calculate the value given to this action, the value will be $V_0(a = a_i) = r_i + V_i$. So, to choose the best possible action, the agent needs to calculate resulting values for every action and choose the maximum possible outcome. In other words: $V_0 = \max_{a \in 1 \dots N}(r_a + V_a)$. If were using discount factor γ, we need to multiply value of the next state by gamma: $V_0 = \max_{a \in 1 \dots N}(r_a + \gamma V_a)$.

Page 3

Its not very complicated to extend it for a stochastic case, when our actions can have chance to end up in different states. What we need to do is to calculate the expected value for every action instead of just taking the value of the next state. To illustrate this, lets consider one single action available from state s_0 with three possible outcomes.

Here we have one action available from the state s_0, which can lead to three different states with different probabilities: with probability p_1 it can end up in state s_1, p_2 in state s_2 and p_3 in state s_3 ($p_1 + p_2 + p_3 = 1$, of course). Every target state has its own reward r_1, r_2 or r_3. To calculate the expected value after issuing action 1, we need to sum all values multiplied by their probabilities:

$$V_0(a = 1) = p_1(r_1 + \gamma V_1) + p_2(r_2 + \gamma V_2) + p_3(r_3 + \gamma V_3)$$

or, more formal

$$V_0(a) = \mathbb{E}_{s \sim S}[r_{s,a} + \gamma V_s] = \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s)$$

By combining the Bellman equation for a deterministic case with value for stochastic actions, we get Bellman equation for general case:

$$V_0 = \max_{a \in A} \mathbb{E}_{s \sim S}[r_{s,a} + \gamma V_s] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s)$$

(Notation $p_{a,i \to j}$ means probability of action a issued in state i to end up in state j)

Page 4, Value of action

To make our life slightly easier, we can define different quantity in addition to value of state $V(s)$: value of action $Q(s, a)$. Basically, it equals total reward we can get by executing action a in state s, and could be defined via $V(s)$. Being much less fundamental entity than $V(s)$, this quantity gave a name to the whole family of methods Q-learning, because it is slightly more convenient in practice.

In those methods, our primary objective is to get values of Q for every pair of state and action.

$$Q(s,a) = \mathbb{E}_{s' \sim S}[r(s,a) + \gamma V(s')] = \sum_{s' \in S} p_{a,s \to s'}(r(s,a) + \gamma V(s'))$$

Which means: Q for this state s and action a equals the expected immediate reward plus discounted long-term reward of destination state. We also can define $V(s)$ via $Q(s,a)$:

$$V(s) = \max_{a \in A} Q(s,a)$$

And, finally, we can express Q(s, a) via itself, which will be used in the next chapters topic of Q-learning:

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in A} Q(s',a')$$

To give you a concrete example, lets consider a simple environment which is similar to FrozenLake, but has much simpler structure: we have one initial state s_0 surrounded by four target states s_1, s_2, s_3, s_4 with different rewards.

Lets calculate the values of actions to begin with. Terminal states $s_1 \dots s_4$ have no outbound connections, so Q for those states is zero for all actions. Due to this, the values of the Terminal states are equal to their immediate reward (once we get there, our episode ends without any subsequent states): $V_1 = 1, V_2 = 2, V_3 = 3, V_4 = 4$. The values of actions for state 0 are a bit more complicated. Lets start with the up action. Its value, according to the definition, is equal to the expected sum of the immediate reward plus long-term value for subsequent steps. We have no subsequent steps for any possible transition for the up action, so

$$Q(s_0, up) = 0.33 \cdot V_1 + 0.33 \cdot V_2 + 0.33 \cdot V_4 = 0.33 \cdot 1 + 0.33 \cdot 2 + 0.33 \cdot 4 = 2.31$$

.

Repeating this for the rest of s_0 actions results in the following:

$$Q(s_0, left) = 0.33 \cdot V_1 + 0.33 \cdot V_2 + 0.33 \cdot V_3 = 1.98$$
$$Q(s_0, right) = 0.33 \cdot V_4 + 0.33 \cdot V_1 + 0.33 \cdot V_3 = 2.64$$
$$Q(s_0, down) = 0.33 \cdot V_3 + 0.33 \cdot V_2 + 0.33 \cdot V_4 = 2.97$$

The final value for state s_0 is the maximum of those actions values, which is 2.97.

Page 6, Value iteration method

We start from state s_1 and the only action we can do leads us to state s_2. We get reward r=1 and the only transition from s_2 is an action which brings us back to the s_1. So, the life of our agent is an infinite sequence of states [

$s_1, s_2, s_1, s_2, s_1, s_2, s_1, s_2, \ldots]$. To deal with this infinity, we can use a discount factor $\gamma = 0.9$. Now, the question: whats the values for both states?

The answer is not very complicated, though. Every transition from s_1 to s_2 gives us reward of 1 and every back transition gives us 2. So, our sequence of rewards will be [1, 2, 1, 2, 1, 2, 1, 2, .]. As there is only one action available in every state, our agent has no choice, so, we can omit max operation in formulas (there is only one alternative). Value for every state will be equal to the infinite sum:

$$V(s_1) = 1 + \gamma(2 + \gamma(1 + \gamma(2 + \ldots))) = \sum_{i=0}^{\infty} 1\gamma^{2i} + 2\gamma^{2i+1}$$

$$V(s_2) = 2 + \gamma(1 + \gamma(2 + \gamma(1 + \ldots))) = \sum_{i=0}^{\infty} 2\gamma^{2i} + 1\gamma^{2i+1}$$

Strictly speaking, we cannot calculate the exact values for our states, but with $\gamma = 0.9$, contribution of every transition quickly decreases over time. For example, after 10 steps, $\gamma^{10} = 0.910 = 0.349$, but after 100 steps it becomes just 0.0000266. Due to this, we can stop after 50 iterations and still get quite precise estimation.

1. Initialize values of all states V_i to some initial value, usually zero.
2. For every state s in the MDP perform Bellman update: $V_s \leftarrow \max_a \sum_{s'} p_{a,s\to s'}(r_{s,a} + \gamma V_{s'})$
3. Repeat step 2 for some large amount of steps or until changes become too small.

Only minor modifications to the above procedure are required in case of action values (i.e. Q):

1. Initialize all $Q_{s,a}$ to zero
2. For every state s and every action a in this state perform update: $Q_{s,a} \leftarrow \sum_{s'} p_{a,s\to s'}(r_{s,a} + \gamma \max_{a'} Q_{s',a'})$
3. Repeat step 2

The second practical problem arises from the fact that we rarely know the transition probability for the actions and rewards matrix. Remember what interface provides Gym to the agents writer: we observe the state, decide on an action and only then do we get the next observation and reward for the transition. We dont know (without peeking into Gyms environment code) what the probability is to get into state s_1 from state s_0 by issuing action a_0. What we do have is just the history from the agents interaction with the environment. However, in Bellmans update, we need both a reward for every transition and the probability of this transition. So, the obvious answer to this issue is to use

our agents experience as an estimation for both unknowns. Rewards could be used as they are. We just need to remember what reward weve got on transition from s_0 to s_1, using action a, but to estimate probabilities we need to maintain counters for every tuple (s_0, s_1, a) and normalize them.