Learning from experience

- What if we don't know the exact model of the environment, but we are allowed to ***sample*** from it?
  - That is, we are allowed to "practice" the MDP as much as we want.
  - This echoes real-life experience.
- One way to do this is ***temporal difference learning.***

Temporal difference learning

- We want to compute V(s) or Q(s, a).
- TD learning uses the idea of taking lots of samples of V or Q (from the MDP) and averaging them to get a good estimate.
- Let's see how this works for estimating the probability of a coin flip being heads.

Q-learning

- Q-learning is a temporal difference learning algorithm that learns optimal values for Q (instead of V, as value iteration did).
- The algorithm works in episodes, where the agent "practices" (aka *samples*) the MDP to learn which actions obtain the most rewards.
- Like value iteration, table of Q values eventually converge to Q*.

  (under certain conditions)

Initialize $Q[s,a]$ arbitrarily, e.g., $Q[s,a]=0$ for all (s,a) pairs.
Repeat (for each episode):
  Set s to the start state
  Repeat (for each step of the episode):
    Choose action a from state s using policy derived from Q (see note below)
    Take action a, observe reward r, new state s'
    $Q[s,a] \leftarrow Q[s,a] + \alpha\,[r + \gamma \max_{a'} Q[s',a'] - Q[s,a]]$
    $s \leftarrow s'$
  until s is a final state
Output a policy π where $\pi(s) = \operatorname{argmax}_a Q(s,a)$

- Notice the Q[s, a] update equation is very similar to the coin probability update equation.
  - (The extra γ $\max_{a'}$ Q[s', a'] piece is to handle future rewards.)
  - alpha ($0 < \alpha <= 1$) is called the learning rate; it controls how fast the algorithm learns. In stochastic environments, alpha is usually small, such as 0.1.

Initialize $Q[s,a]$ arbitrarily, e.g., $Q[s,a] = 0$ for all (s,a) pairs.
Repeat (for each episode):
    Set s to the start state
    Repeat (for each step of the episode):
        Choose action a from state s using policy derived from Q (see note below)
        Take action a, observe reward r, new state s'
        $Q[s,a] \leftarrow Q[s,a] + \alpha \left[r + \gamma \max_{a'} Q[s',a'] - Q[s,a]\right]$
        $s \leftarrow s'$
    until s is a final state
Output a policy π where $\pi(s) = \text{argmax}_a Q(s,a)$

- Note: The "choose action" step does not mean you choose the best action according to your table of Q values.
- You must balance exploration and exploitation; like in the real world, the algorithm learns best when you "practice" the best policy often, but sometimes explore other actions that may be better in the long run.

Initialize $Q[s, a]$ arbitrarily, e.g., $Q[s, a] = 0$ for all (s, a) pairs.
Repeat (for each episode):
    Set s to the start state
    Repeat (for each step of the episode):
        Choose action a from state s using policy derived from Q (see note below)
        Take action a, observe reward r, new state s'
        $Q[s, a] \leftarrow Q[s, a] + \alpha \left[r + \gamma \max_{a'} Q[s', a'] - Q[s, a]\right]$
        $s \leftarrow s'$
    until s is a final state
Output a policy π where $\pi(s) = \text{argmax}_a\, Q(s, a)$

- Often the "choose action" step uses policy that mostly exploits but sometimes explores.
- One common idea: (epsilon-greedy policy)
  - With probability 1 - ε, pick the best action (the "a" that maximizes Q[s, a].
  - With probability ε, pick a random action.
- Also common to start with large ε and decrease over time while learning.

Initialize $Q[s,a]$ arbitrarily, e.g., $Q[s,a] = 0$ for all (s,a) pairs.
Repeat (for each episode):
    Set s to the start state
    Repeat (for each step of the episode):
        Choose action a from state s using policy derived from Q (see note below)
        Take action a, observe reward r, new state s'
        $Q[s,a] \leftarrow Q[s,a] + \alpha\,[r + \gamma \max_{a'} Q[s',a'] - Q[s,a]]$
        $s \leftarrow s'$
    until s is a final state
Output a policy π where $\pi(s) = \mathrm{argmax}_a\, Q(s,a)$

- What makes Q-learning so amazing is that the Q-values still converge to the optimal Q* values even though the algorithm itself is not following the optimal policy!

Q-learning with Ebola!

- Update formula:

$$Q[s,a] \leftarrow Q[s,a] + \alpha \left[r + \gamma \max_{a'} Q[s',a'] - Q[s,a] \right]$$

- Sample episodes (states and actions):

Sick-X ➔ A ➔ Sick-A ➔ A ➔ Healthy

Sick-X ➔ A ➔ Sick-A ➔ B ➔ Healthy

Sick-X ➔ A ➔ Sick-A ➔ A ➔ Sick-A ➔ B ➔ Healthy