Reinforcement Learning Notes

Notation:

- π represents a *policy*. It is a function from states to actions that tells you what action to take in every possible state. In general, policies do not necessarily have to be optimal.
- π^* is an *optimal policy*. It is a function that tells you the *best* action to take in any state in order to maximize the expected value of the discounted sum of all future rewards.
- $V^\pi(s)$ is represents the *value of a state*, a number that represents how "valuable" it is to be in state s of the MDP. Mathematically, this is the expected value of the discounted sum of all future rewards we would see if we are in state s and follow policy π.

  $V^*(s)$ is value of state s under the optimal policy π^*.
- $Q^\pi(s,a)$ is the *value of a state-action pair*, a number that represents how "valuable" it is to be in state s and then take action a in the MDP. Mathematically, this is the expected value of the discounted sum of all future rewards we would see if we are in state s, take action a, and then follow policy π. Note that taking action a from state s may not be what π says to do.

  $Q^*(s,a)$ is the expected value of the pair (s,a) under the optimal policy π^*.

Recursive relationship of V and Q functions to each other:

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

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma V^*(s')\big]$$

These equations lead to the Bellman equations:

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma V^*(s')\big]$$

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\big]$$

Value iteration is an algorithm that learns an optimal policy, given that you know $P(s' \mid s,a)$ and $R(s,a,s')$. It uses a table of V values to store estimates of the true V^* function. These estimates converge (in the limit) to the true values of V^*.

Initialize V arbitrarily, e.g., $V[s] = 0$ for all states s.
Repeat
    for each state s:
        $V_{\text{new}}[s] \leftarrow \max_a \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma V[s']]$
    $V \leftarrow V_{\text{new}}$ (copy new table over old)
until the maximum difference in new and old values is smaller than some threshold
Output a policy π where $\pi(s) = \text{argmax}_a \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma V[s']]$

Q-learning is an algorithm that learns an optimal policy where you don't have to know $P(s' \mid s, a)$ or $R(s, a, s')$. It works in episodes, where in each episode you start in the start state, and repeatedly take actions and get rewards until you reach a final state. Every time you take an action, get a reward, and land in a new state, you update the table of Q values.

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 $\pi(s) = \operatorname{argmax}_a Q[s, a]$

Note: When choosing an action a using a policy *derived from* Q, the action should **not** always be the best action determined by the policy you are learning (that's why it says **derived** from Q, not following Q exactly). Instead, you must balance the conflicting goals of **exploration** (following a random action to see if it is better than what you believe at the moment is the best action) and **exploitation** (following what you believe at the moment is the best action in order to refine your estimate of $Q(s, a)$). For instance, the policy derived from Q might be ε-greedy.