Q-Learning

Reinforcement Learning

- What if we don't know P(s', | s, a) or R(s)?
- These could be very tedious to specify.
- Two possibilities:
 - Learn $P(s', \mid s, a)$ and R(s), then apply value iteration.
 - Model based reinforcement learning.
 - Don't bother to learn $P(s' \mid s,a)$ and R(s). Just learn U(s) directly.
 - Model free reinforcement learning.

Model Based RL

- Learn P(s' | s, a) and R(s) and apply value iteration.
 - Since there is no hidden state $P(s' \mid s, a)$ and R(s) are easy to learn.
 - Take random actions, then:

$$\hat{P}(s'|s,a) = \frac{\text{\# transitions from s to s' given a}}{\text{total \# of transitions from s given a}}$$

 $\hat{R}(s) = \text{average reward observed in state s}$

Choosing Actions While Learning

- Previous slide suggested "random actions".
- Two problems with that:
 - Waste time exploring bad actions.
 - Lose reward while learning.

Greedy Policies

- One possible solution always choose the action that looks best so far.
 - Good idea?

Exploration vs. Exploitation

- We need to find a trade off between choosing actions that appear good now (exploitation), and taking actions that might turn out to be good later (exploration).
- Reasonable solutions are GLIE Greedy in the Limit with Infinite Exploration.

Model Free Reinforcement Learning

- The goal: learn U(s) without bothering to learn $P(s' \mid s, a)$ or R(s).
- Helpful?
- Even if we know the optimal utility function, we can't choose actions if we don't know the transition function:

$$\pi^{*}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a) U(s')$$

Q-Learning

- Instead we will learn something slightly different:
 - Q(s,a) = the expected value of taking action a in state s, and acting optimally thereafter.
- Q(s,a) has a simple relationship with U(s):

$$U(s) = \max_{a} Q(s, a)$$

• If we have Q(s,a), we don't need $P(s' \mid s,a)$:

$$\pi^*(s) = \operatorname{argmax} Q(s, a)$$

Q-Learning Update Rule

• Recall the Bellman equation:

$$U(s) = R(s) + \gamma \max \sum P(s'|s,a)U(s')$$

• From this we can derive:

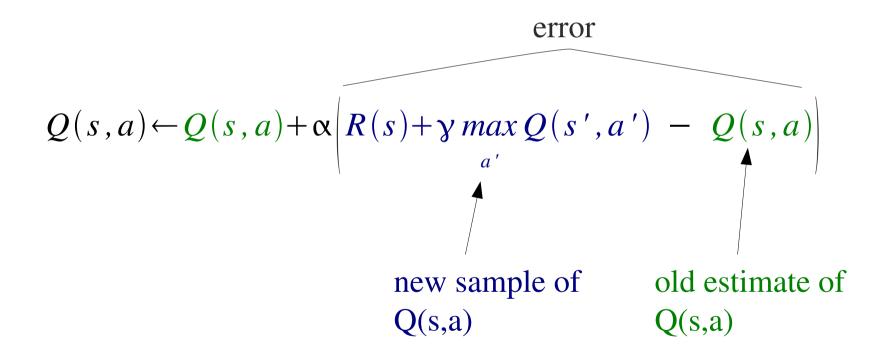
$$Q(s,a)=R(s)+\gamma\sum_{s'}P(s'|s,a)\max_{a'}Q(s',a')$$
• From this we can get the following update rule:

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

• s' is the next state (arrived at after action a), a' is the next action, α is a learning rate.

Unpacking Q-Learning

• The update moves the value of the old estimate in the direction of the new sample:



The Q-Learning Algorithm

- Initialize Q(s,a) randomly.
- Choose actions according to a GLIE policy.
- After every action, perform an update:

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

- Convergence to the optimal policy is guaranteed.
- This is really easy to program.

Q-Learning Efficiency

- Q-Learning is efficient in terms of the computation required per action.
- However, Q-learning does not make efficient use of experience.
- For example, if rewards are sparse, a Q-learning agent can run for a long time without learning anything.

Problems in Reinforcement Learning

- RL (and MDP) algorithms scale reasonably well with the number of states.
- Unfortunately, the number of the states does not scale well with the complexity of the problem.
- An example of the curse of dimensionality.