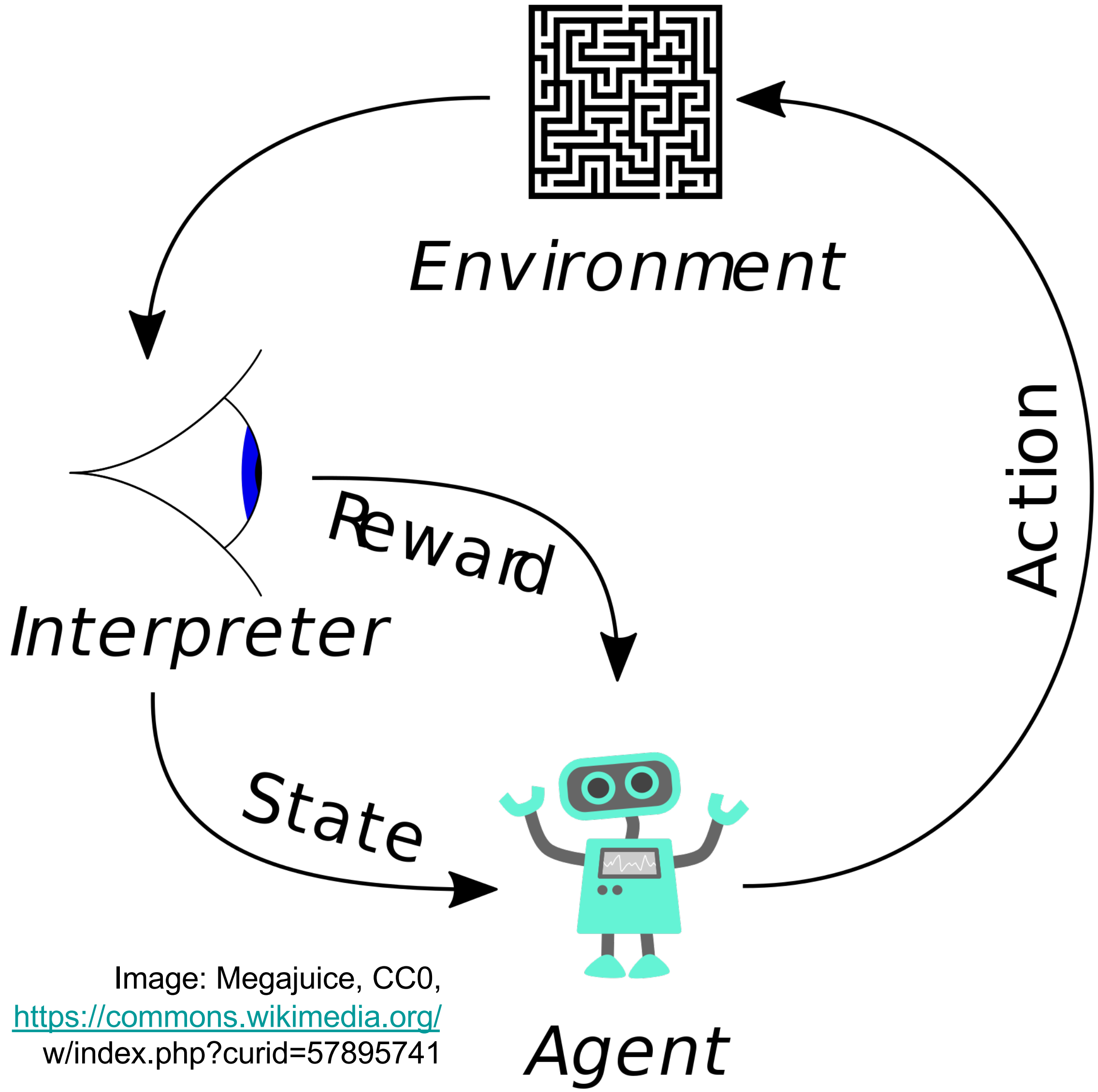
Model-Free Reinforcement Learning

CS440/ECE448 Lecture 34

Mark Hasegawa-Johnson, 4/2022,
including slides by Svetlana Lazebnik, 11/2017

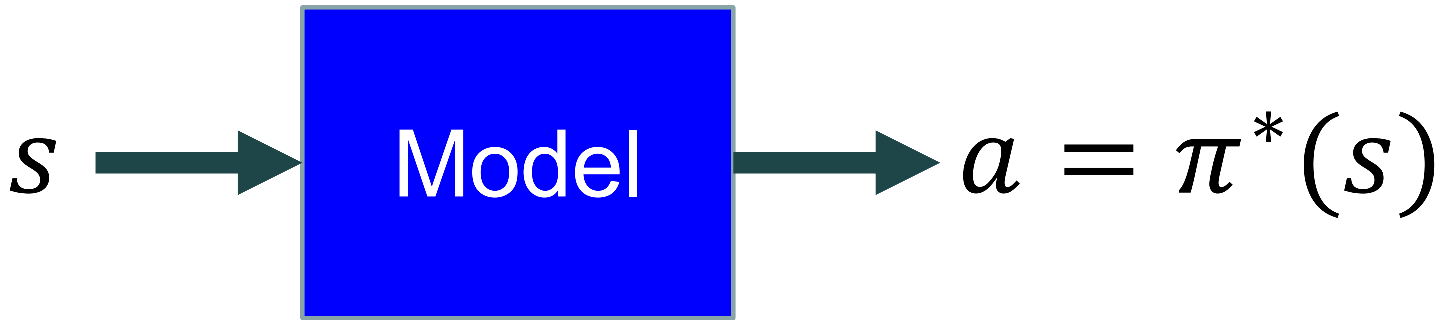


What we've learned so far

- Markov Decision Process (MDP): Given P(s'|s,a) and R(s), you can solve for $\pi^*(s)$, the optimal policy, by finding U(s), the value of each state, using either value iteration or policy iteration.
- Model-Based Reinforcement Learning: If P(s'|s,a) and R(s) are unknown, you can find for $\pi(s)$ by using the observation-model-policy loop:
  - Observation: Create a training dataset by trying n consecutive actions, using an exploration-exploitation tradeoff like epsilon-first or epsilon-greedy
  - Model: Estimate P(s'|s,a) and R(s) using maximum likelihood estimation or Laplace smoothing
  - Policy: Find the optimum policy using value iteration or policy iteration.

Today: Model-Free Learning

Why can't we just learn a model (neural net, or even a table lookup) that does this:



Outline

- Q(s,a) – the “quality” of an action
- Q-learning
- Off-policy learning: TD
- On-policy learning: SARSA

Bellman's Equation

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

When we talked about solving Bellman's equation before, we said that the optimum policy is given by the "max" operation: the action that gives you that maximum is the action you should take.

The Quality of an Action

The goal of Q-learning is to learn a function, Q(s,a), such that the best action to take is the action that maximizes Q:

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

How about if we define Q(s,a) to be “The expected future reward I will achieve if I take action a in state s?”

The Quality of an Action

Suppose we know everything: we know P(s'|s,a), R(s), γ, and U(s). Then we collect our total expected future reward by doing these things:

- Collect our current reward, R(s)
- Discount all future rewards by γ
- Make a transition to a future state, s', according to P(s'|s,a)
- Then collect all future rewards, U(s')

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a)U(s')$$

The Quality of an Action

…so the Q-function splits Bellman's equation into two parts:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

…becomes…

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a)U(s')$$

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

The Q-function without U

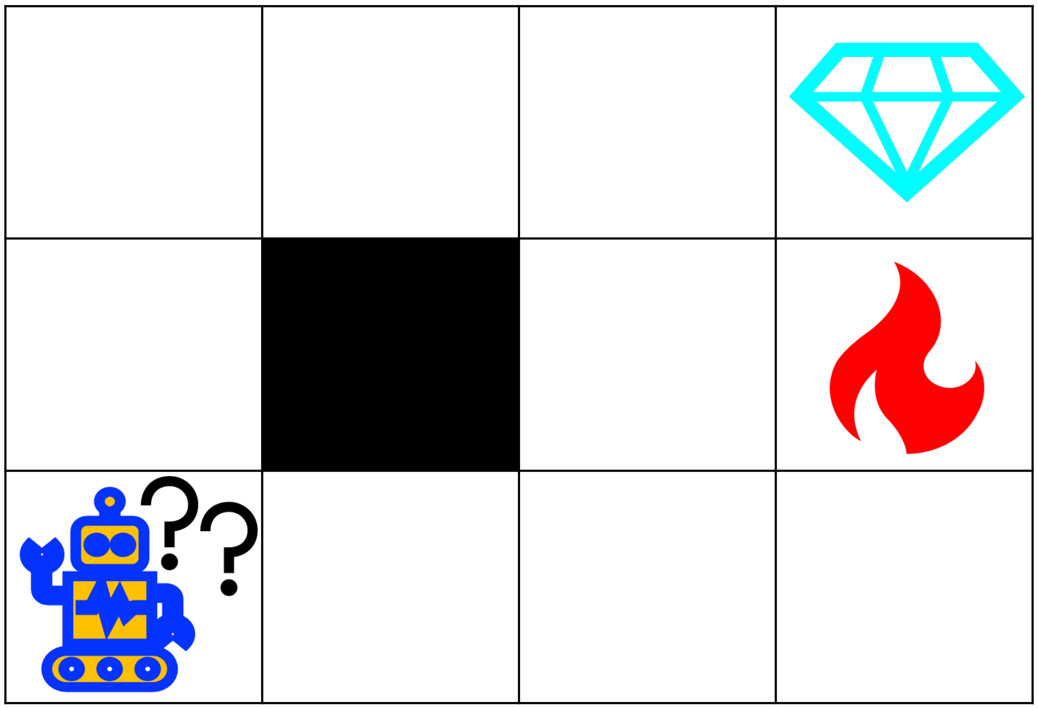
Suppose we just want Q(s,a), and we don't want to have to calculate U(s). Then we can plug $U(s') = \max_{a' \in A(s')} Q(s', a')$ into the RHS to get

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a' \in A(s')} Q(s', a')$$

It has these steps:

- Collect our current reward, R(s)
- Discount all future rewards by γ
- Make a transition to a future state, s', according to P(s'|s,a)
- Choose the optimum action, a', from state s', and collect all future rewards.

Example: Gridworld



$$R(s) = \begin{cases} +1 & s = (4,3) \\ -1 & s = (4,2) \\ -0.04 & \text{otherwise} \end{cases}$$

$$P(s'|s,a) = \begin{cases} 0.8 & \text{intended} \\ 0.1 & \text{fall left} \\ 0.1 & \text{fall right} \end{cases}$$

$$\gamma = 1$$

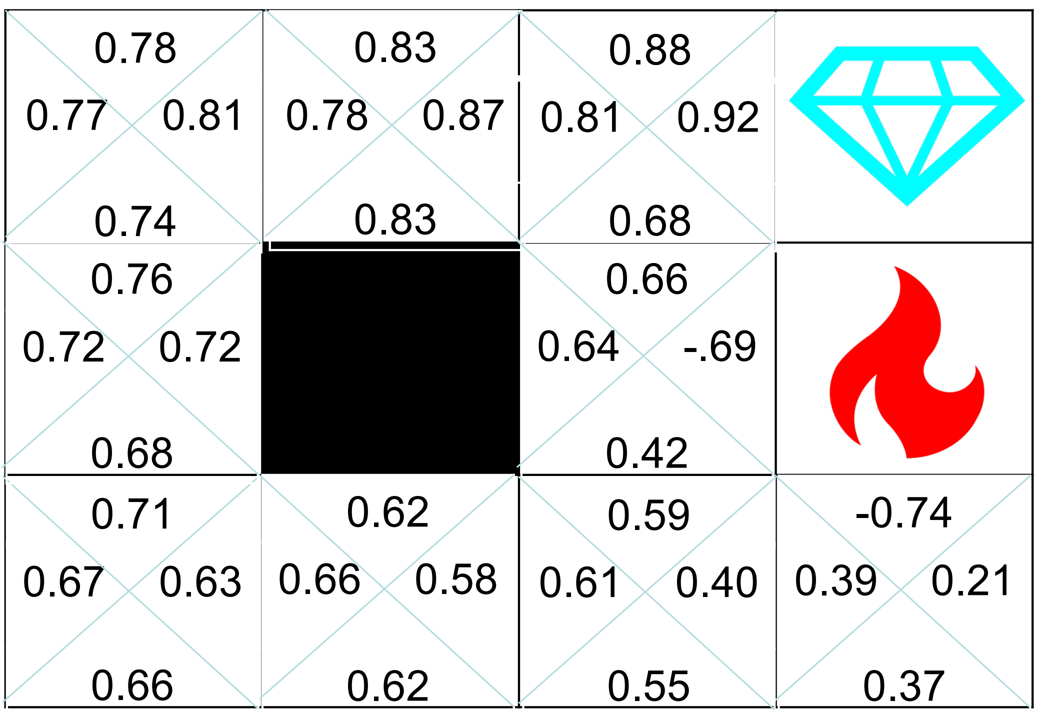
Gridworld: Utility of each state

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

0.81	0.87	0.92	
0.76		0.66	
0.71	0.66	0.61	0.39

(Calculated using value iteration.)

Gridworld: The Q-function



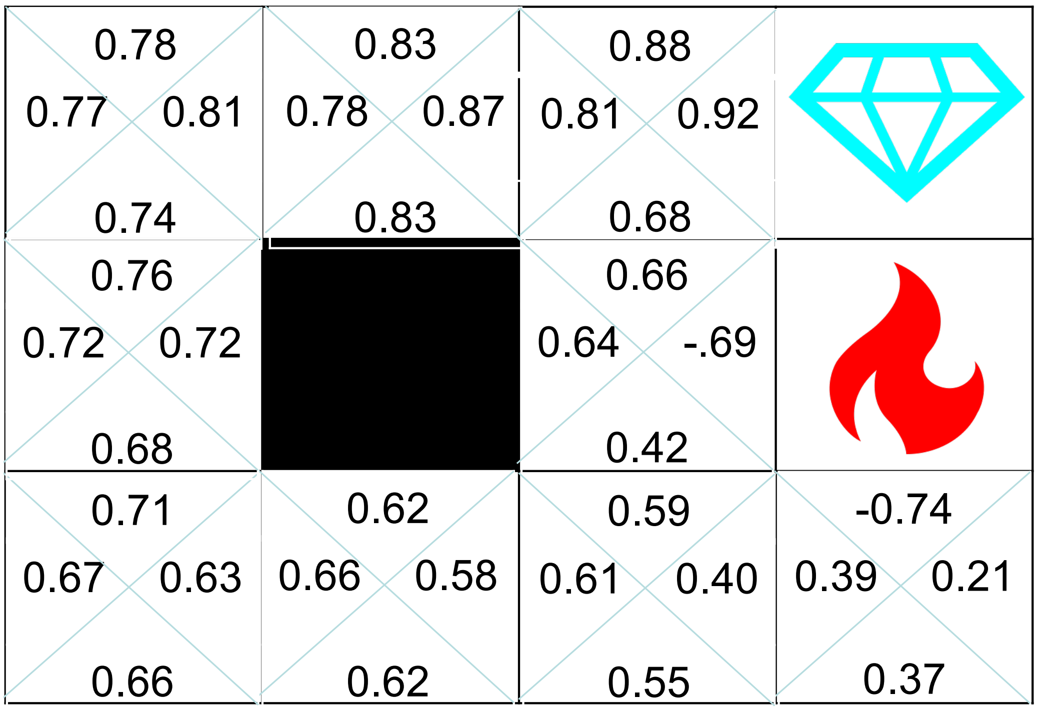
Calculated using a two-step value iteration:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a)U(s')$$

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

Gridworld: Relationship between Q and U

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



Outline

- Q(s,a) – the “quality” of an action
- Q-learning
- Off-policy learning: TD
- On-policy learning: SARSA

Reinforcement learning: Key concepts

Key concept: What if you don't know P(s'|s,a) and R(s)? Can you still estimate Q(s,a)?

1. Method #1: Model-based learning. Estimate P(s'|s,a) and R(s), then use them to compute Q(s,a).
2. Method #2 (today): Model-free learning. Try some stuff, observe the results, use the results to estimate Q(s,a).

Q-learning

Q(s,a) is the total of all current & future rewards that you expect to get if you perform action a in state s.

…so how about this strategy…

1. Play the game an infinite number of times.
2. Each time you try action a in state s, measure the reward that you receive from that point onward for the rest of the game.
3. Average.

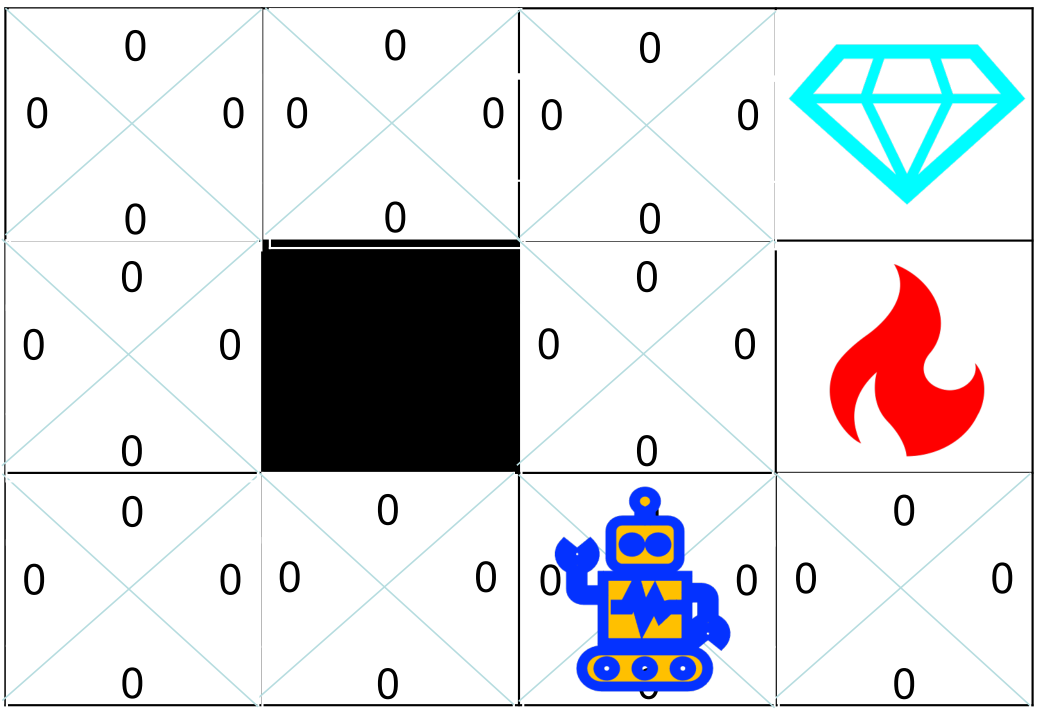
Q-learning: a slightly more practical version

Q(s,a) is the total of all current & future rewards that you expect to get if you perform action a in state s.

…so how about this strategy…

1. Play the game ~~an infinite~~ **finite** number of times. **Keep track of $Q_t(s,a)$, the estimate of Q after the tth iteration.**
2. Each time you try action a in state s, measure the reward that you receive ~~from that point onward for the rest of the game.~~ **in the current state, plus γ times $Q_t(s',a')$.**
3. Average **Q_t with #2 in order to get Q_{t+1}**.

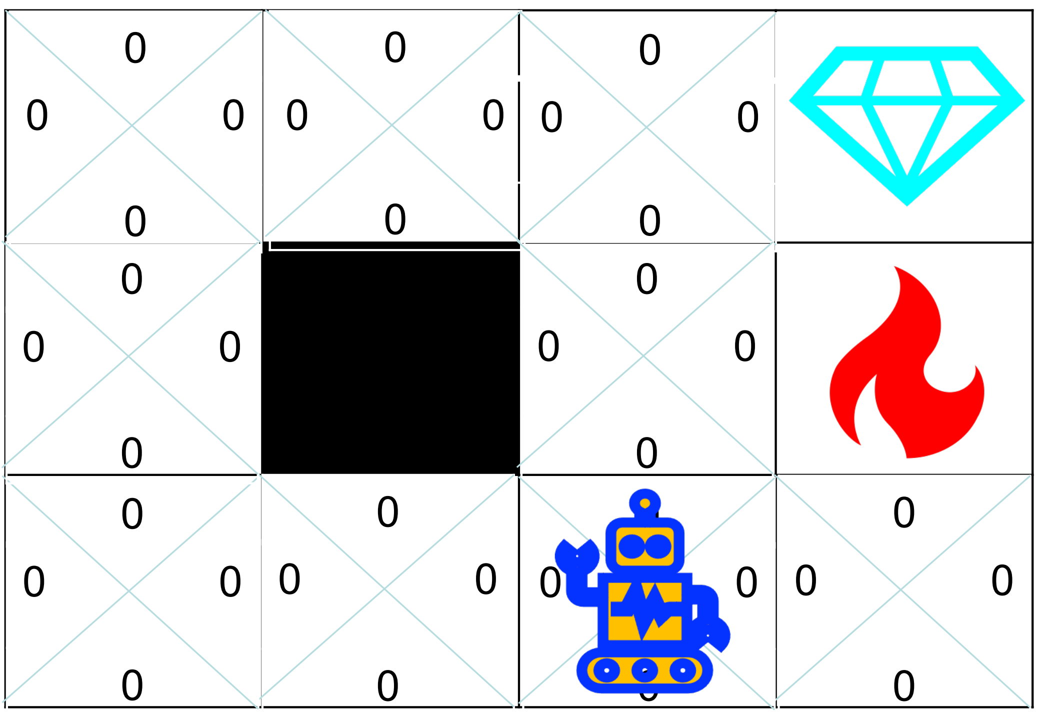
Example: Gridworld



Suppose we start out with $\underline{Q_1(s,a) = 0}$ for all states and actions.

Robot starts out in state (3,1).

Example: Gridworld



Suppose we start out with $\underline{Q_1(s,a) = 0}$ for all states and actions.

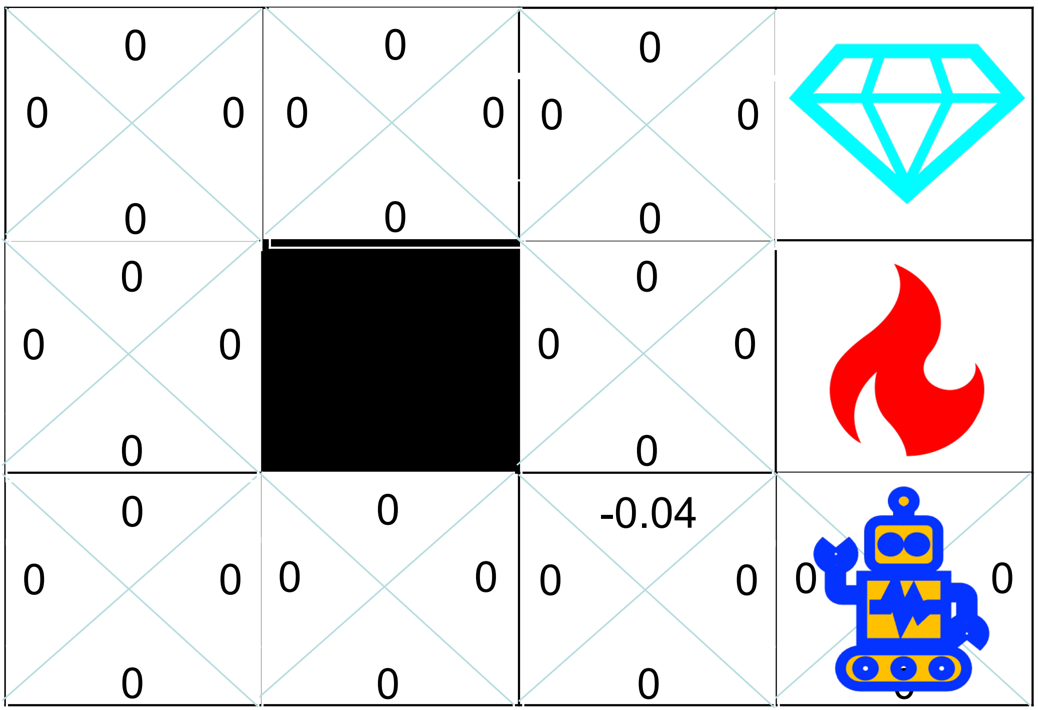
Robot starts out in state (3,1).

Robot receives a reward of -0.04.

Robot tries to move UP...

but falls right, to state (4,1).

Example: Gridworld



Now we update the Q((3,1),UP) as:

$$Q((3,1),\mathrm{UP}) = R((3,1)) + \gamma U\big((4,1)\big)$$

$$= -0.04$$

The three main problems with reinforcement learning

1. We don’t know the reward function. All we know is the reward we got this time around.
2. We don’t know the transition probabilities. All we know is the state that we reached this time around.
3. We don’t know the utility of the state we reached. All we know is our current (noisy) estimate of Q(s,a).

Outline

- Q(s,a) – the “quality” of an action
- Q-learning
- Off-policy learning: TD
- On-policy learning: SARSA

TD learning

$$Q(s,a) = R(s) + \gamma \sum_{s\prime} P(s'|s,a) \max_{a\prime \in A(s\prime)} Q(s',a')$$

Let's solve these problems as follows:

- Instead of $R(s)$, use $R_t(s)$, the reward we got this time.
- Instead of summing over $P(s'|s,a)$, just set s' equal to whatever state followed s this time.
- Instead of the true value of $Q(s,a)$, use our current estimate, $Q_t(s,a)$.

TD learning

$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma \max_{a' \in A(s_{t+1})} Q_t(s_{t+1}, a')$$

Let's solve these problems as follows:

- Instead of $R(s)$, use $R_t(s_t)$, the reward we got this time.
- Instead of summing over $P(s'|s, a)$, just set $s' = s_{t+1}$, i.e., whatever state followed s_t.
- Instead of the true value of $Q(s, a)$, use our current estimate, $Q_t(s, a)$.

TD learning

$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma \max_{a' \in A(s_{t+1})} Q_t(s_{t+1}, a')$$

Problem: NOISY!

- s_{t+1} is random, and
- $Q_t(s_{t+1}, a')$ is not the real value of Q, only our current estimate, therefore
- $Q_{local}(s_t, a_t)$ might be very far away from $Q(s, a)$. It might even be worse than $Q_t(s, a)$.

TD learning

Solution: interpolate, using a small interpolation constant α that's $0 < \alpha < 1$:

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha Q_{local}(s,a)$$

$$= Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

TD learning

$Q_{local}(s,a) - Q_t(s,a)$ is called the "time difference" or TD.

1. If the TD is positive, it means action a was **<u>better</u>** than we expected, so $Q_{t+1}(s,a) = Q_t(s,a) + \alpha TD$ is an increase.
2. If the TD is negative, it means action a was **<u>worse</u>** than we expected, so $Q_{t+1}(s,a) = Q_t(s,a) + \alpha TD$ is a decrease.

Exploration versus exploitation

- TD-learning has one gap, still: when you reach state s, how do you choose an action?
- You might think that you just choose $a^* = \max_{a \in A(s)} Q_t\,(s, a)$, but that has the following problem: what if $Q_t\,(s, a)$ is wrong?
- The solution is to use an exploration strategy. For example,
  - Epsilon-first strategy: if there's an action we've chosen less than $1/\epsilon$ times, then choose that. Otherwise, choose a^*.
  - Epsilon-greedy strategy: with probability $1 - \epsilon$, choose a^*. With probability ϵ, choose an action uniformly at random.

TD learning

Putting it all together, here's the whole TD learning algorithm:

1. When you reach state s, use your current exploration versus exploitation policy, $\pi_t(s)$, to choose some action $a = \pi_t(s)$.
2. Observe the state s' that you end up in, and the reward you receive, and then calculate Qlocal:

$$Q_{local}(s,a) = R_t(s) + \gamma \max_{a' \in A(s')} Q_t(s',a')$$

3. Calculate the time difference, and update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

Repeat.

TD learning

The action you actually perform

Putting it all together, here's the whole TD learning algorithm:

1. When you reach state s, use your current exploration versus exploitation policy, $\pi_t(s)$, to choose some action $a = \pi_t(s)$.
2. Observe the state s' that you end up in, and the reward you receive, and then calculate Qlocal:

$$Q_{local}(s,a) = R_t(s) + \gamma \max_{a' \in A(s')} Q_t(s',a')$$

3. Calculate the time difference, and update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

The action TD-learning assumes you will perform

Repeat.

TD learning is an off-policy learning algorithm

TD learning is called an off-policy learning algorithm because it assumes an action

$$\underset{a' \in A(s')}{\operatorname{argmax}} Q_t(s', a')$$

…which is different from the action dictated by your current exploration versus exploitation policy

$$a' = \pi_t(s')$$

Sometimes off-policy learning converges slowly, for example, because the TD-learning update is not taking advantage of your exploration.

Outline

- Q(s,a) – the “quality” of an action
- Q-learning
- Off-policy learning: TD
- On-policy learning: SARSA

On-policy learning: SARSA

We can create an “on-policy learning” algorithm by deciding in advance which action (a') we’ll perform in state s', and then using that action in the update equation:

1. Assume that you’re currently in state s_t, and you’ve already chosen action a_t.
2. Observe the state s_{t+1} that you end up in, and then use your current policy to choose $a_{t+1} = \pi_t(s_{t+1})$.
3. Calculate Qlocal and the update equation as:

$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma Q_t(s_{t+1}, a_{t+1})$$

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\big(Q_{local}(s_t, a_t) - Q_t(s_t, a_t)\big)$$

4. Go to step 2.

On-policy learning: SARSA

This algorithm is called SARSA (state-action-reward-state-action) because:

- In order to compute the TD-learning version of Q_{local}, you only need to know the tuple (s_t, a_t, R_t, s_{t+1}):
$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma \max_{a\prime \in A(s_{t+1})} Q_t(s_{t+1}, a')$$
- In order to compute the SARSA version of Q_{local}, you need to have already picked out $(s_t, a_t, R_t, s_{t+1}, a_{t+1})$:
$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma Q_t(s_{t+1}, a_{t+1})$$

Summary

- Q(s,a) – the "quality" of an action

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a)U(s')$$

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

- Q-learning
- Off-policy learning: TD

$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma \max_{a' \in A(s_{t+1})} Q_t(s_{t+1}, a')$$

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\big(Q_{local}(s_t, a_t) - Q_t(s_t, a_t)\big)$$

- On-policy learning: SARSA

$$a_{t+1} = \pi_t(s_{t+1})$$

$$Q_{local}(s_t, a_t) = R_t(s_t) + \gamma Q_t(s_{t+1}, a_{t+1})$$