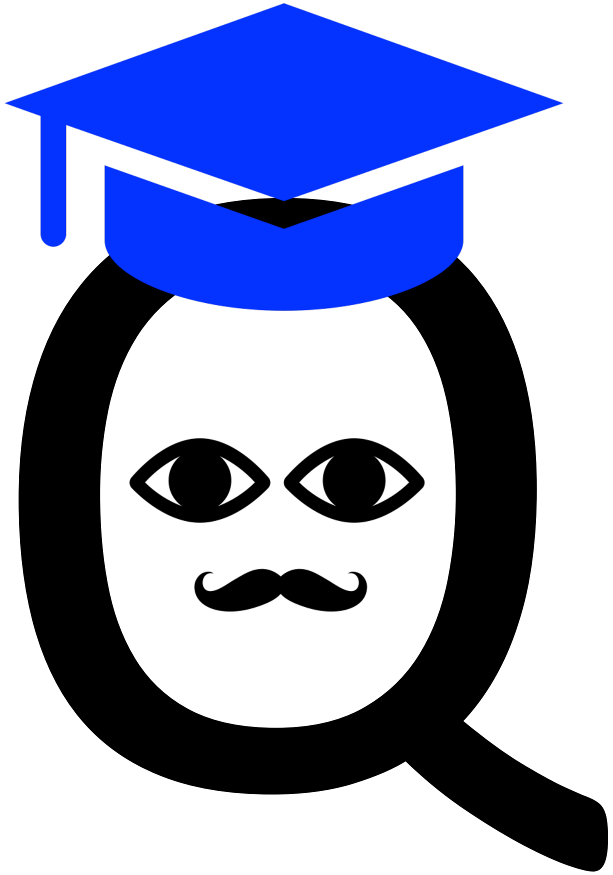
CS 440/ECE448 Lecture 31: Q-Learning

Mark Hasegawa-Johnson, 4/15/2020



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 observations-model-policy loop:
  - Observations: 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: Q-Learning

- If you knew P(s’|s,a) and R(s), how would you define the quality of an action, Q(s,a)?
- Q-learning
    - Key concepts
    - TD-learning: a practical algorithm for Q-learning
- Off-policy vs. on-policy learning: TD vs. SARSA
- Batch learning

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

Whoa! So Bellman's equation is actually just a simplified version of the Q-function:

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

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

The Q-function: recursive definition

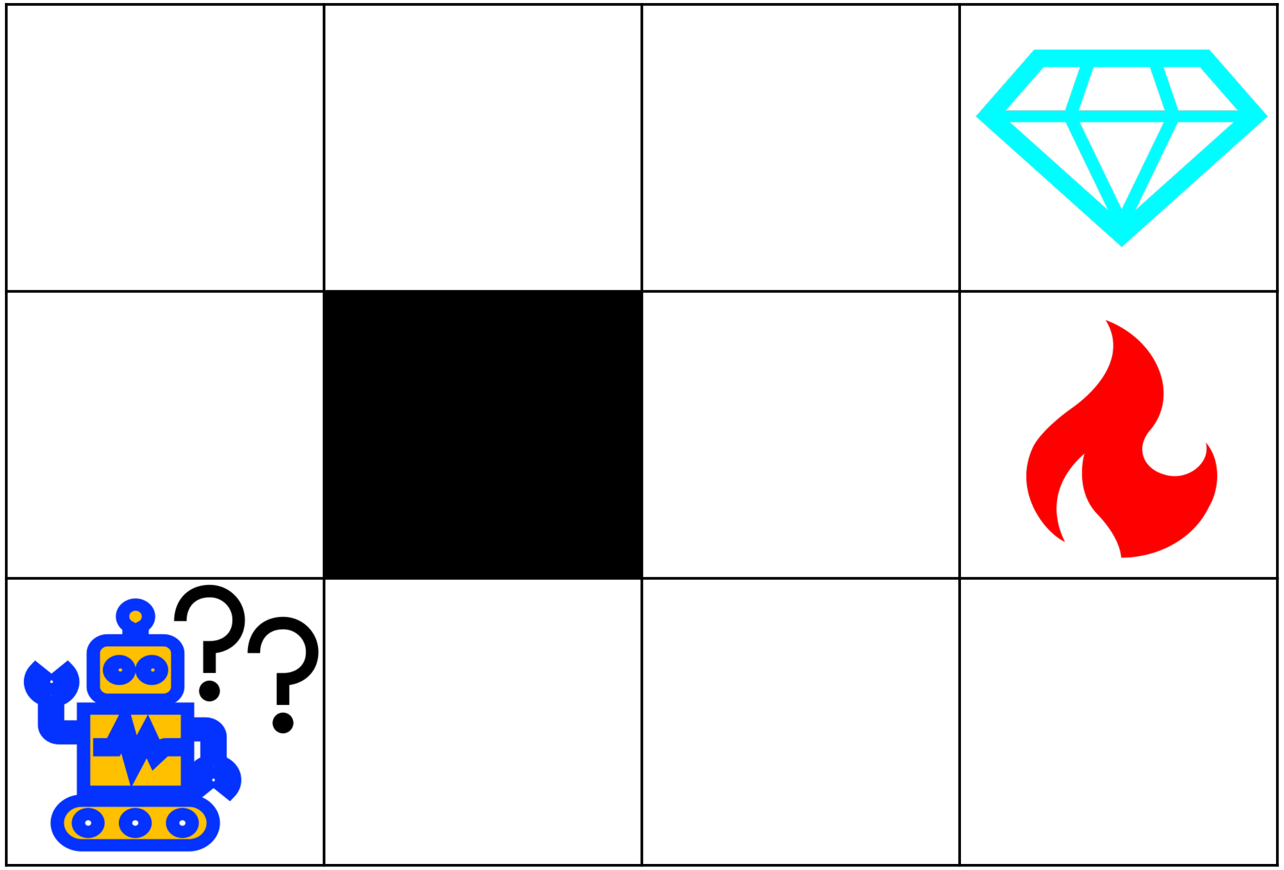
Or, to look at it another way, we could plug $U(s') = \max_{a' \in A(s')} Q(s', a')$ into the definition of the Q-function in order to get

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

Remember, 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



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

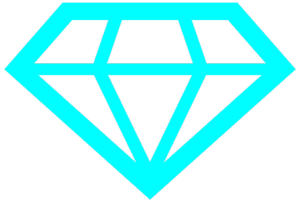
$$P(s'|s,a) = \begin{cases} 0.8 & \text{intended} \\ 0.1 & \text{fall left} \\ 0.1 & \text{fall right} \end{cases}$$

$$\gamma = 1$$

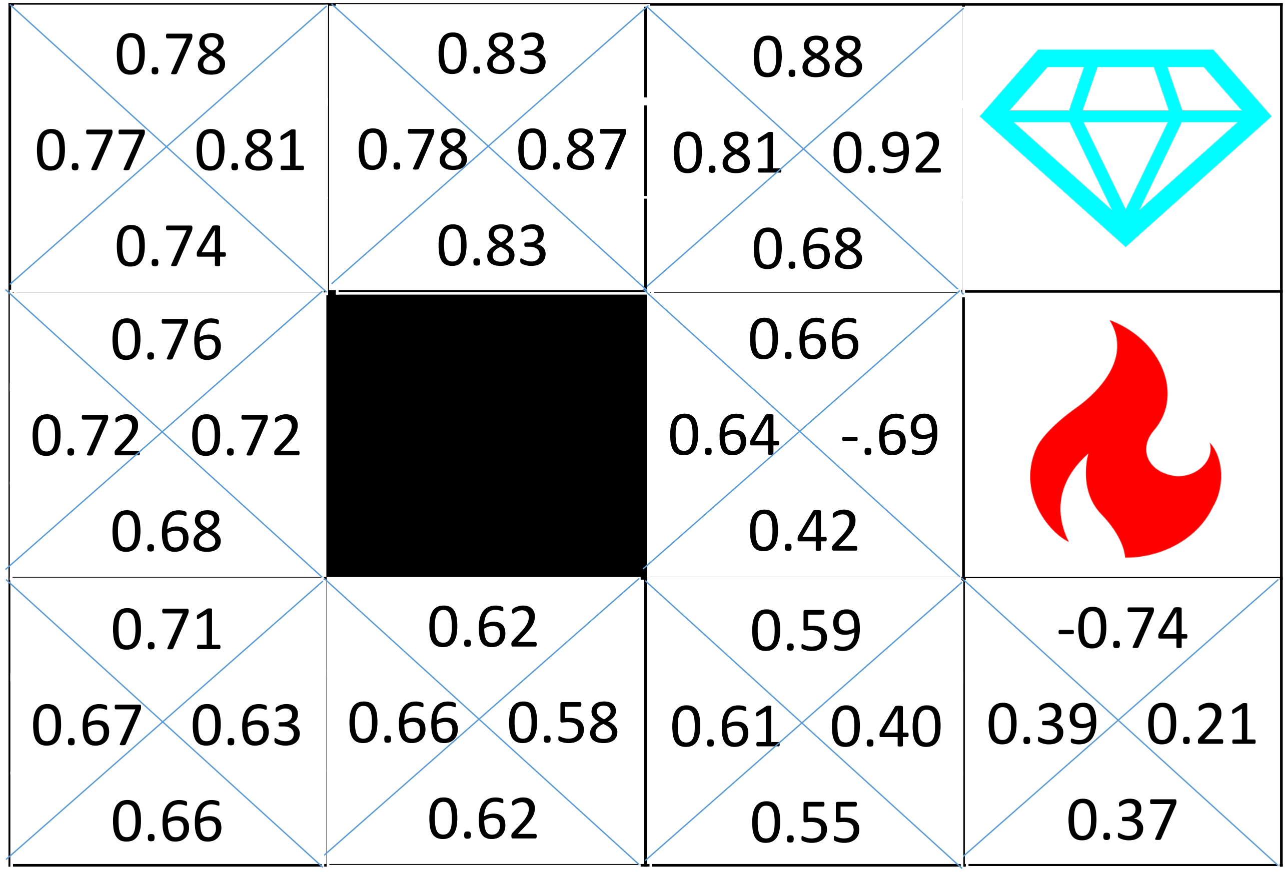
Gridworld: Utility of each state

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

(Calculated using value iteration.)

0.81	0.87	0.92	
0.76		0.66	
0.71	0.66	0.61	0.39

Gridworld: The Q-function



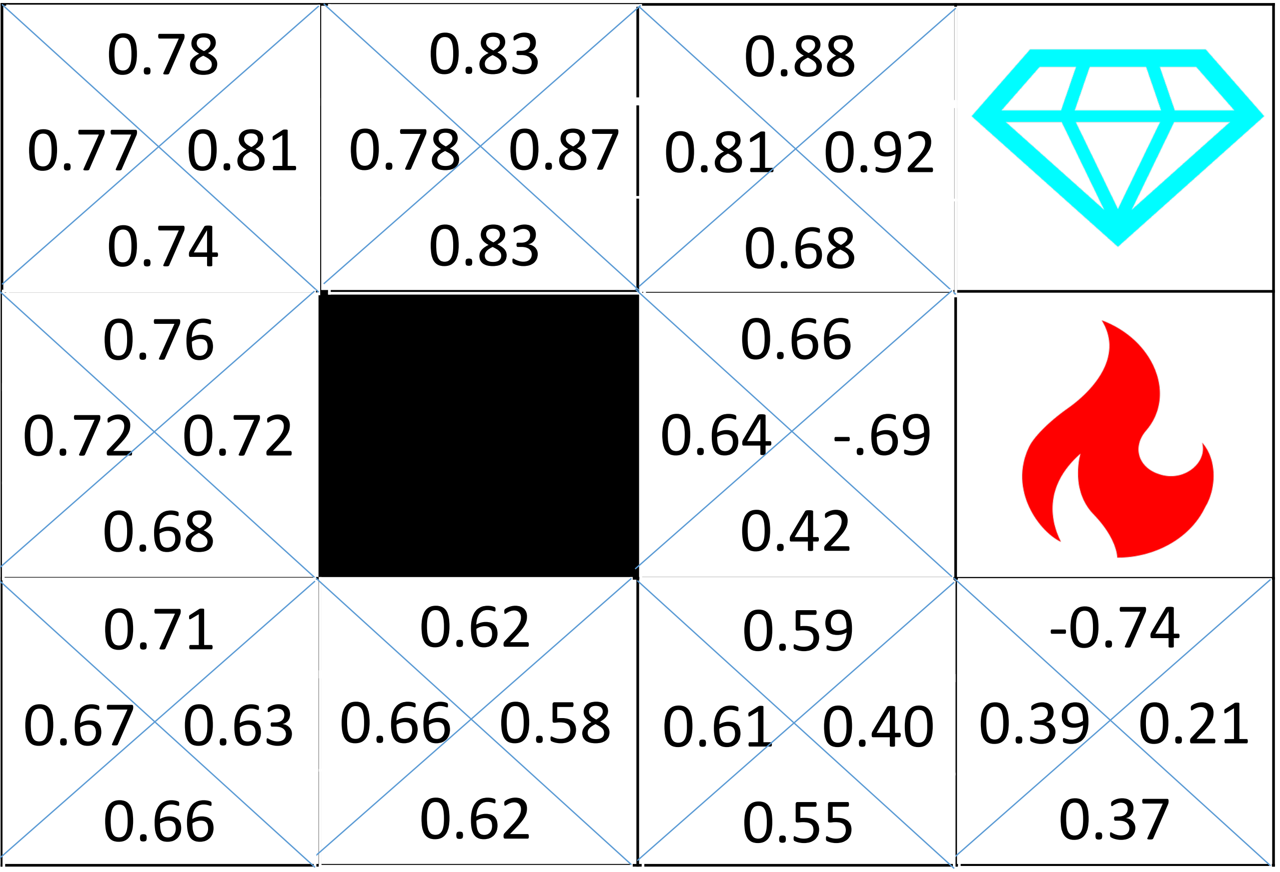
Calculated using a two-step value iteration:

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

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

Gridworld: Relationship between Q and U

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



0.81	0.87	0.92	
0.76		0.66	
0.71	0.66	0.61	0.39

Today: Q-Learning

- If you knew P(s'|s,a) and R(s), how would you define the quality of an action, Q(s,a)?
- Q-learning
  - Key concepts
  - TD-learning: a practical algorithm for Q-learning
- Off-policy vs. on-policy learning: TD vs. SARSA
- Batch learning

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 a~~n infinite~~ **finite** number of times. **Keep track of $Q_t(s,a)$, the estimate of Q after the t^th 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}**.

Q-learning

Remember that the true Q-function is given by Bellman's equation to be:

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

But in Q-learning, we have the following problems:

- We don't know $R(s)$
- We don't know $P(s'|s,a)$
- We don't yet know $Q(s,a)$.

TD learning

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

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

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

The problem with this solution is that it's noisy. s' was chosen completely at random, so $Q_{local}(s,a)$ might be very far away from $Q(s,a)$. It might even be worse than $Q_t(s,a)$.

We can solve this problem by interpolating, using an 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

This quantity, $dQ_t(s,a) = Q_{local}(s,a) - Q_t(s,a)$, is called the "time difference." The whole algorithm is therefore called "time difference learning" (TD learning). It goes like this:

1. When you reach state s, try some action a. Observe the state s' that you end up in, and the reward you receive, and then calculate Qlocal:

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

2. Calculate the time difference, and update:

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

Repeat.

Today: Q-Learning

- If you knew P(s'|s,a) and R(s), how would you define the quality of an action, Q(s,a)?
- Q-learning
  - Key concepts
  - TD-learning: a practical algorithm for Q-learning
- Off-policy vs. on-policy learning: TD vs. SARSA
- Batch learning

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 ϵN 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.

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. 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 then use your current policy to choose $a' = \pi_t(s')$.
3. Calculate Qlocal and the update equation as:

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

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

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, a, R, s'):

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

- In order to compute the SARSA version of Q_{local}, you need to have already picked out (s, a, R, s', a'):

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

Today: Q-Learning

- If you knew P(s'|s,a) and R(s), how would you define the quality of an action, Q(s,a)?
- Q-learning
  - Key concepts
  - TD-learning: a practical algorithm for Q-learning
- Off-policy vs. on-policy learning: TD vs. SARSA
- Batch learning

Batch learning

Both TD learning and SARSA can be performed in batch mode:

1. Play the game several times, using a fixed policy $\pi_t(s)$.
2. Collect a training database of SARS or SARSA tuples:

$$\mathcal{D} = \{(s_1, a_1, R_1, s_1'), \dots, (s_n, a_n, R_n, s_n')\}$$

…or…

$$\mathcal{D} = \{(s_1, a_1, R_1, s_1', a_1'), \dots, (s_n, a_n, R_n, s_n', a_n')\}$$

3. Compute the updates, $\{Q_{local,1}, \dots, Q_{local,1}\}$, and update Q:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \sum_{s_i,a_i=s,a} \left(Q_{local,i} - Q_t(s,a)\right)$$

Batch learning

- Both TD-learning and SARSA work pretty well when you have a discrete state space, s, and Q(s,a) is just a lookup table.
- When your state space is continuous-valued, then you have to use a neural network to estimate Q(s,a). In that case, batch learning becomes much more important, to help you learn smoothly.
- … so I’ll talk more about batch learning when I talk about deep Q-learning, on Friday.

Conclusions: Q-learning

- $Q(s,a)$ is the expected reward you get by choosing action a in state s. Its true value is given by Bellman's equation as:

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

- If you don't know $P(s'|s,a)$ or $R(s)$, you can learn $Q(s,a)$ using TD-learning:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \left(R_t(s) + \gamma \max_{a' \in A(s')} Q_t(s',a') - Q_t(s,a) \right)$$

- TD-learning is an off-policy algorithm. SARSA is an example of an on-policy algorithm:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(R_t(s) + \gamma Q_t(s',a') - Q_t(s,a)\big)$$

- Batch learning collects a large number of SARS or SARSA tuples before each update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \sum_{s_i,a_i=s,a} \left(Q_{local,i} - Q_t(s,a) \right)$$

The robots of the world thank you for helping them find blue diamonds.