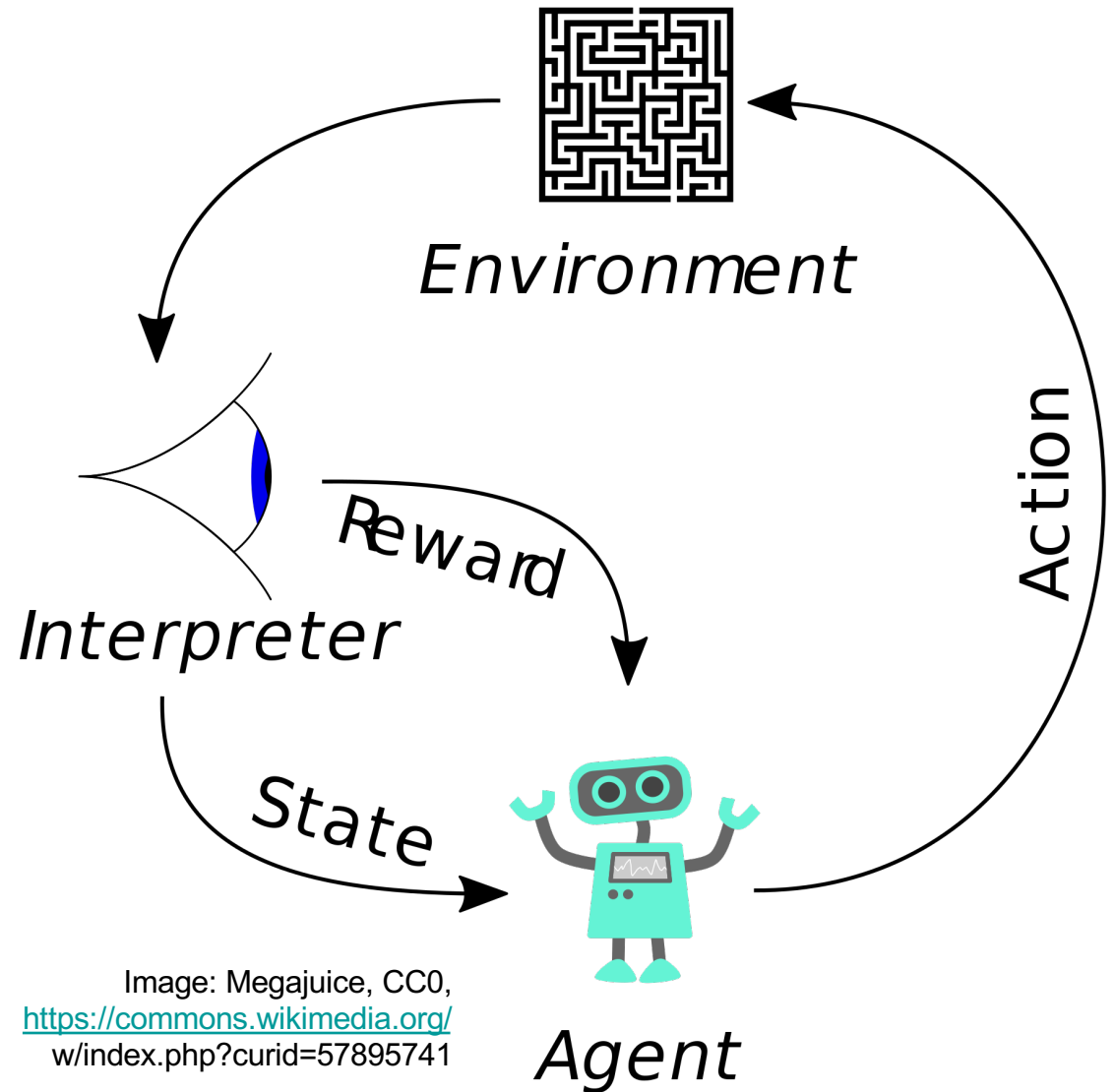


Model-Free Reinforcement Learning

CS440/ECE448



Mark Hasegawa-Johnson
This slides are in the public domain

Image: Megajuce, CC0,
<https://commons.wikimedia.org/w/index.php?curid=57895741>

Outline

- Q-learning: What to estimate
- Q-local: The information available to us
- TD-learning: Q-learning with smoothing
- SARSA: on-policy Q-learning

What should reinforcement learning learn?

Last time:

- Model-based learning: $P(s'|s, a)$

Today:

- Q-learning: $q(s, a)$, the quality of action a in state s

Monday:

- Policy gradient: estimate a stochastic policy $P(A_t = a | S_t = s)$

The Quality of an Action

Q-learning splits Bellman's equation into two parts:

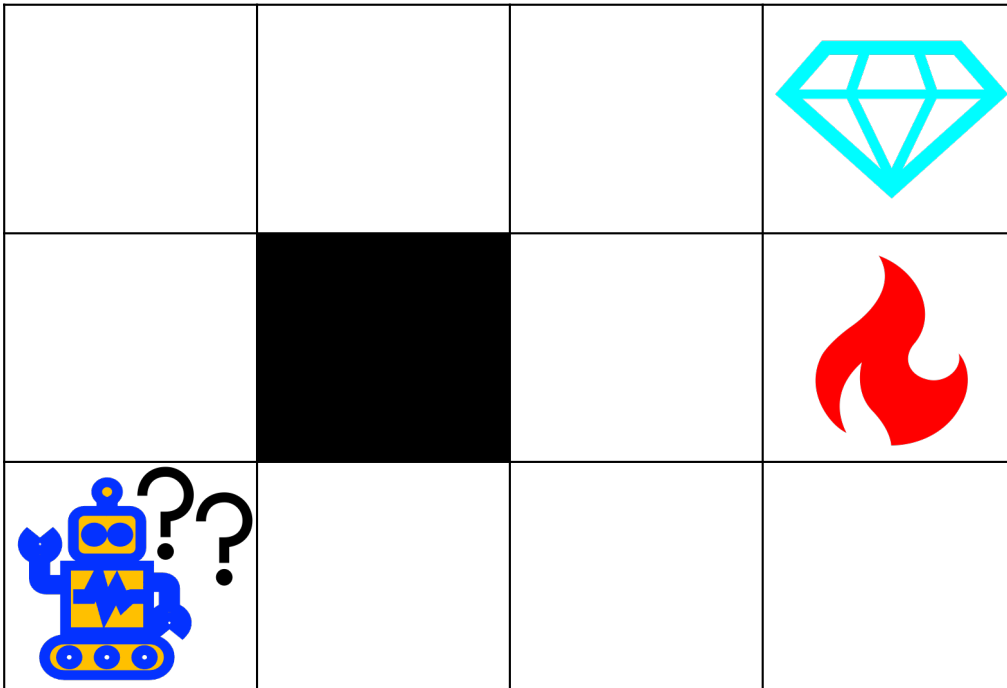
$$u(s) = r(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s, a) u(s')$$

...becomes...

$$q(s, a) = r(s) + \gamma \sum_{s'} P(s'|s, a) u(s')$$

$$u(s) = \max_{a \in \mathcal{A}} q(s, a)$$

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

$$u(s) = r(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s, a) u(s')$$

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

0.78 0.77 0.81	0.83 0.78 0.87	0.88 0.81 0.92	
0.74 0.76	0.83 [Black square]	0.68 0.66	
0.72 0.72		0.64 -0.69	
0.68 0.71	0.62 0.66 0.58	0.42 0.59	-0.74
0.67 0.63	0.66 0.58	0.61 0.40	0.39 0.21
0.66	0.62	0.55	0.37





Calculated using a two-step value iteration:

$$q(s, a) = r(s) + \gamma \sum_{s'} P(s'|s, a)u(s')$$

$$u(s) = \max_{a \in \mathcal{A}} q(s, a)$$

Gridworld: Relationship between Q and U

$$u(s) = \max_{a \in \mathcal{A}} q(s, a)$$

0.78 0.77 0.81	0.83 0.78 0.87	0.88 0.81 0.92		0.81	0.87	0.92	
0.74 0.76 0.72 0.72	0.83	0.68 0.66 0.64 -0.69		0.76		0.66	
0.68 0.71 0.67 0.63	0.62 0.66 0.58	0.59 0.61 0.40	-0.74 0.39 0.21	0.71	0.66	0.61	0.39
0.66	0.62	0.55	0.37				

Q-learning

- In the reinforcement learning scenario, we don't know $P(s'|s, a)$. We just want to play the game, and observe our earned reward, and from it, estimate $q(s, a)$.
- On the t^{th} iteration of q-learning, suppose that we have an estimate $q_t(s, a)$. We can use that as follows:

Try action a_t in state s_t . Measure the reward r_t , and observe the estimated utility of the state we end up in $u_t(s_{t+1})$.

Why?

- Notice that, if there are N states and M actions, $P(s'|s, a)$ is a table that contains MN^2 entries
 - Takes a lot of memory to store it
 - Takes a lot of training data to learn all those parameters
- Q-learning learns $q(s, a)$, which has only MN
 - For example, if $N=1000$ and $M=4$, Q-learning requires 1000X less training data

Outline

- Q-learning: What to estimate
- Q-local: The information available to us
- TD-learning: Q-learning with smoothing
- SARSA: on-policy Q-learning

But how can you learn?

Remember in model-based learning we learned:

$$P(s_{t+1}|s_t, a_t) = \frac{N(s_t, a_t, s_{t+1}) + k}{\sum_{s' \in \mathcal{S}} N(s_t, a_t, s') + k|\mathcal{S}|}$$

But how can you learn?

In Q-learning, we want to learn:

$$q(s_t, a_t) = r(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)u(s_{t+1})$$

...but we don't know $P(s_{t+1}|s_t, a_t)$!

But how can you learn?

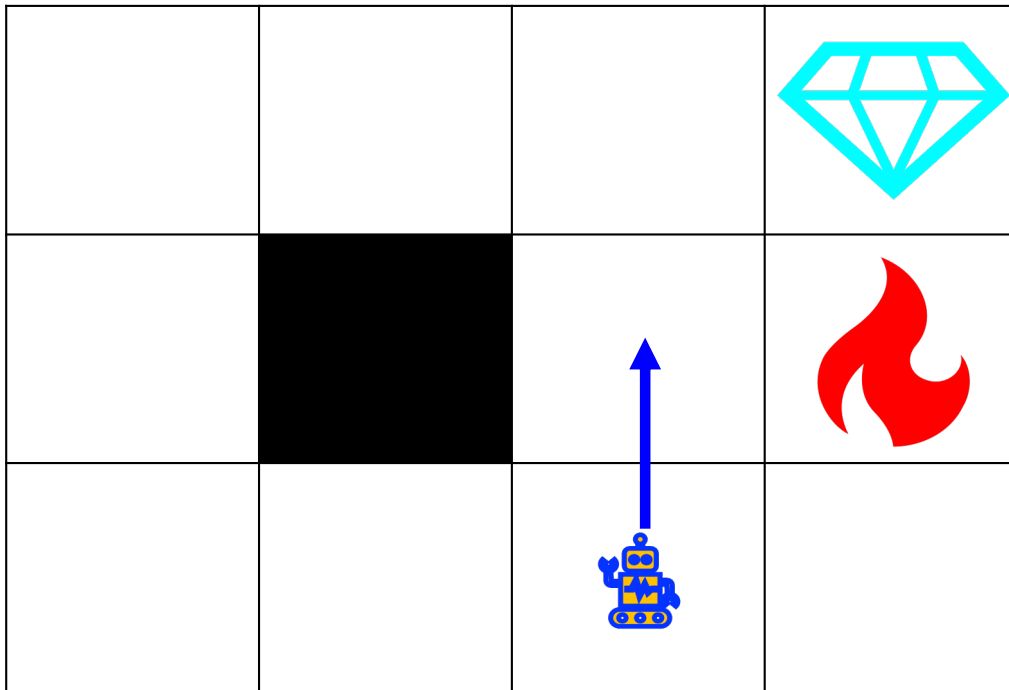
Simplifying assumption: Assume that $P(s_{t+1}|s_t, a_t) \approx 1$ for the s_{t+1} we observe:

$$\begin{aligned} q(s_t, a_t) &= r(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) u(s_{t+1}) \\ &\approx q_{\text{local}}(s_t, a_t, r_t, s_{t+1}) \end{aligned}$$

...which we define as:

$$q_{\text{local}}(s_t, a_t, r_t, s_{t+1}) = r_t + \gamma u(s_{t+1})$$

Example: Gridworld



- Suppose $s_t=(3,1)$ receives a reward of $r_t=-0.04$, tries to move U, ends in $s_{t+1} = (3,2)$.
- Update $q_{\text{local}}((3,1), U)$:

$$q_{\text{local}}((3,1), U) = r_t + \gamma u(s_{t+1}) \\ = -0.04 + \gamma u((3,2)) = ?$$

- Wait. What is $u((3,2))$?

To learn Q, you must already know Q!

We assume:

$$q(s_t, a_t) \approx q_{\text{local}}(s_t, a_t, r_t, s_{t+1}) = r_t + \gamma u(s_{t+1})$$

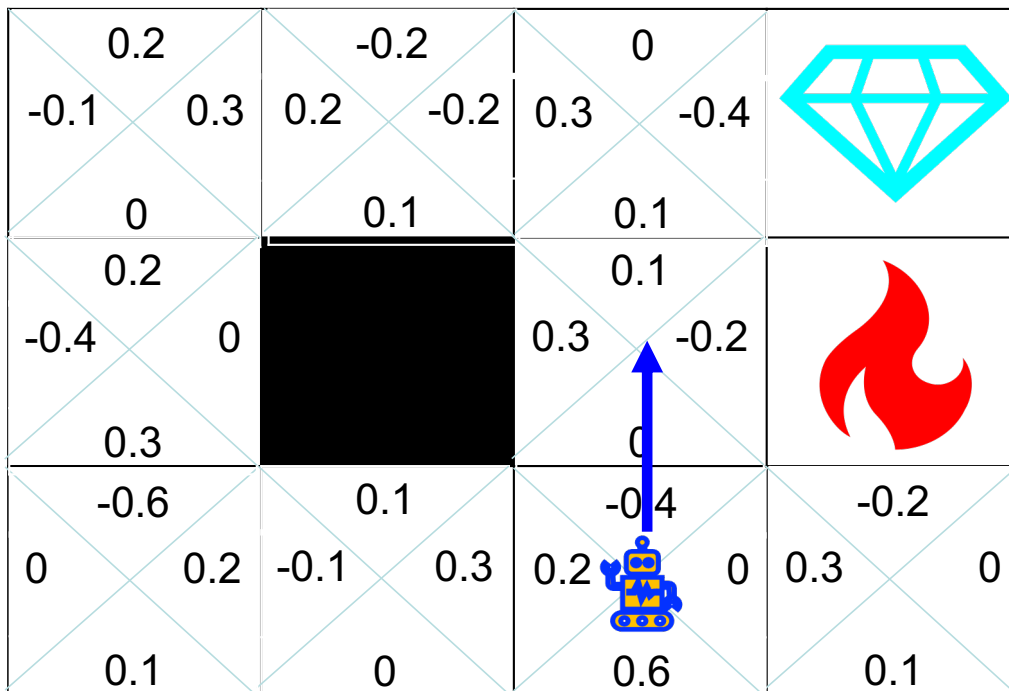
But what is $u(s_{t+1})$? Why, it is $\max_{a \in \mathcal{A}} q(s_{t+1}, a)$.

To learn Q, you must already know Q!

$$q_{\text{local}}(s_t, a_t, r_t, s_{t+1}) = r_t + \gamma \max_{a \in \mathcal{A}} q(s_{t+1}, a)$$

Example: Gridworld


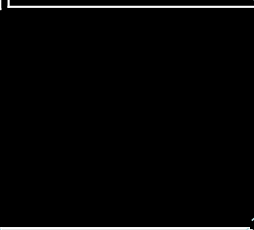

Suppose we start with some small random initial guesses of $q(s, a)$, as shown, and suppose $\gamma = 1$. Then:



$$\begin{aligned}
 q_{\text{local}}((3,1), U) &= r_t + \gamma \max_{a \in \mathcal{A}} q((3,2), a) \\
 &= -0.04 + 0.3 = 0.26
 \end{aligned}$$

Why is that answer so unsatisfactory?

$$\begin{aligned}
 q_{\text{local}}((3,1), U) &= r_t + \gamma \max_{a \in \mathcal{A}} q((3,2), a) \\
 &= -0.04 + 0.3 = 0.26
 \end{aligned}$$

0.2 -0.1 0.3	-0.2 0.2 -0.2	0 0.3 -0.4	
0 0.2 -0.4	0.1 	0.1 0.1 0.3 -0.2	
0.3 -0.6 0	0.1 -0.1 0.3	0 -0.4 0.2	-0.2 0.3 0
0.1 0.2	0	0.6 0.2	0.1

A blue robot icon is positioned at the bottom of the cell (3,2), with a blue arrow pointing upwards towards the cell (3,1).

- It completely ignores our previous estimate, $q((3,1), U) = -0.4$
- It seems to give too much credit to the random initial estimate $q((3,2), L) = 0.3$
- It assumes that $P((3,2)|(3,1), U) = 1$

Outline

- Q-learning: What to estimate
- Q-local: The information available to us
- TD-learning: Q-learning with smoothing
- SARSA: on-policy Q-learning

TD learning

Problem: Q learning is noisy.

Solution: learn slowly, with a learning rate η :

1. If $q(s_t, a_t)$ is a table:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta (q_{local}(s_t, a_t, r_t, s_{t+1}) - q(s_t, a_t))$$

2. If $q(s_t, a_t)$ is estimated using a neural net with parameters \mathbf{w} :

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial}{\partial \mathbf{w}} \frac{1}{2} (q_t(s_t, a_t) - q_{local}(s_t, a_t, r_t, s_{t+1}))^2$$

TD learning

$q_{local}(s_t, a_t, r_t, s_{t+1}) - q_t(s_t, a_t)$ is called the “time difference” or TD.

1. If the TD is positive, it means action a_t was **better** than we expected, so $q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \eta TD$ is an increase.
2. If the TD is negative, it means action a_t was **worse** than we expected, so $q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \eta TD$ is a decrease.

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 to choose some action.
2. Observe the state s_{t+1} that you end up in, and the reward you receive, and then calculate q -local:

$$q_{local}(s_t, a_t, r_t, s_{t+1}) = r_t + \gamma \max_{a' \in \mathcal{A}} q(s_{t+1}, a')$$

3. Calculate the time difference, and update:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta (q_{local}(s_t, a_t, r_t, s_{t+1}) - q(s_t, a_t))$$

or:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial}{\partial \mathbf{w}} \frac{1}{2} (q_t(s_t, a_t) - q_{local}(s_t, a_t, r_t, s_{t+1}))^2$$

Outline

- Q-learning: What to estimate
- Q-local: The information available to us
- TD-learning: Q-learning with smoothing
- **SARSA: on-policy Q-learning**

TD learning is an off-policy learning algorithm

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

$$\operatorname{argmax}_{a' \in \mathcal{A}} q(s_{t+1}, a')$$

...which may be different from your real action (e.g., you might explore instead of exploiting).

- Thus, TD-learning might not converge to real q-functions, because it focuses all the learning on the actions you think are best, and doesn't learn very much about actions whose quality you don't yet know.

On-policy learning: SARSA

We can create an “on-policy learning” algorithm by deciding in advance which action (a_{t+1}) we’ll perform in state s_{t+1} , 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 exploration vs. exploitation policy to already choose a_{t+1} !
3. Calculate q-local and the update equation as:

$$q_{\text{local}}(s_t, a_t, r_t, s_{t+1}, a_{t+1}) = r_t + \gamma q(s_{t+1}, a_{t+1})$$
$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta (q_{\text{local}}(s_t, a_t, r_t, s_{t+1}, a_{t+1}) - q(s_t, a_t))$$

Quiz

Try the quiz!

Summary: Q-learning

Q-learning:

$$q(s, a) = r(s) + \gamma \sum_{s'} P(s'|s, a) u(s')$$
$$u(s) = \max_{a \in \mathcal{A}} q(s, a)$$

TD-learning = Q-learning with smoothing

$$q_{\text{local}}(s_t, a_t, r_t, s_{t+1}) = r_t + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$$
$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta (q_{\text{local}}(s_t, a_t, r_t, s_{t+1}) - q(s_t, a_t))$$

SARSA = on-policy Q-learning

$$q_{\text{local}}(s_t, a_t, r_t, s_{t+1}, a_{t+1}) = r_t + \gamma q_t(s_{t+1}, a_{t+1})$$
$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \eta (q_{\text{local}}(s_t, a_t, r_t, s_{t+1}, a_{t+1}) - q(s_t, a_t))$$