Model-Free Reinforcement Learning

CS440/ECE448 Lecture 33

Mark Hasegawa-Johnson, 4/2024

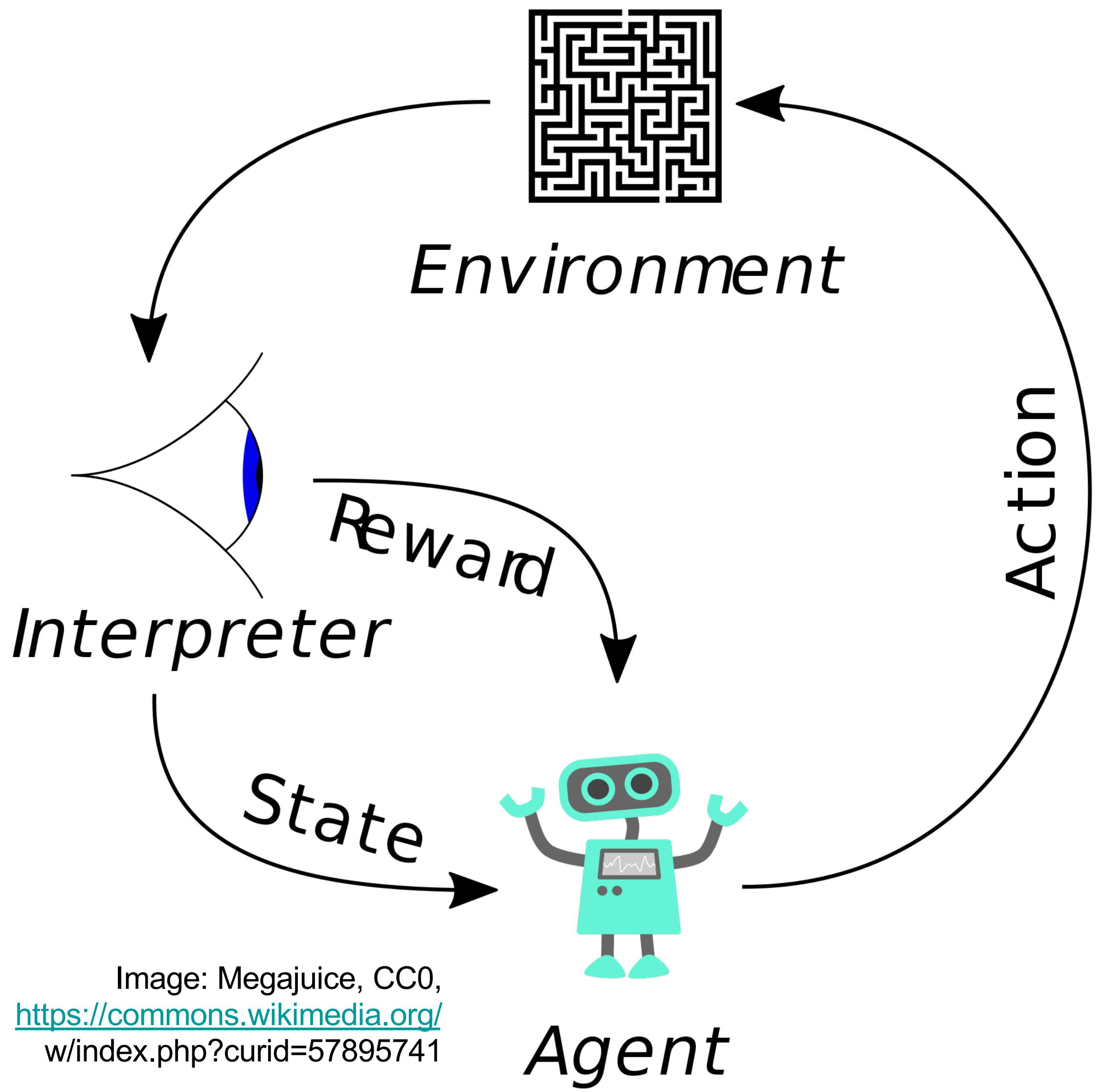


Image: Megajuice, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741

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
- Policy gradient: estimate a stochastic policy $\pi_a(s) = Pr(A_t = a|S_t = s)$; learn it by maximizing expected total return

The Quality of an Action

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

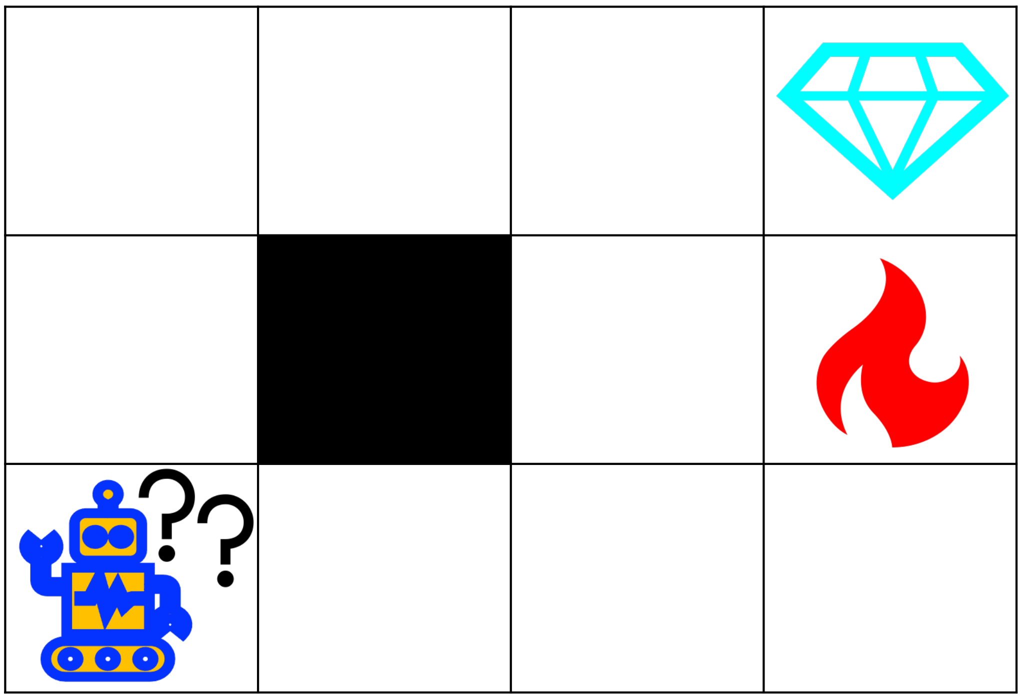
$$u(s) = r(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s,a) u(s')$$

…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



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

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

$$\gamma = 1$$

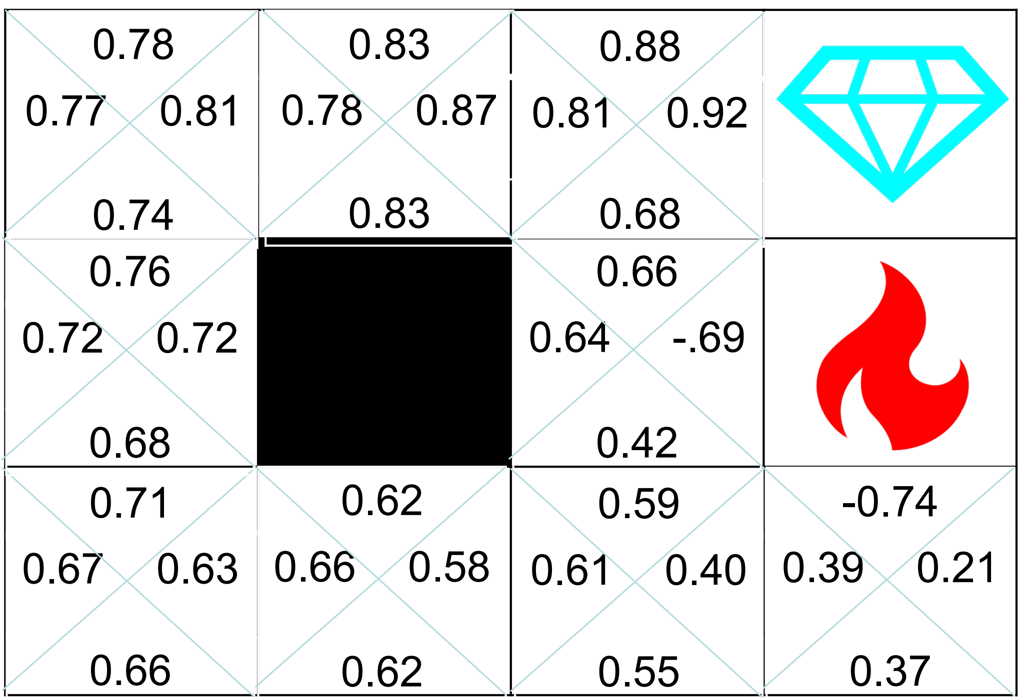
Gridworld: Utility of each state

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

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



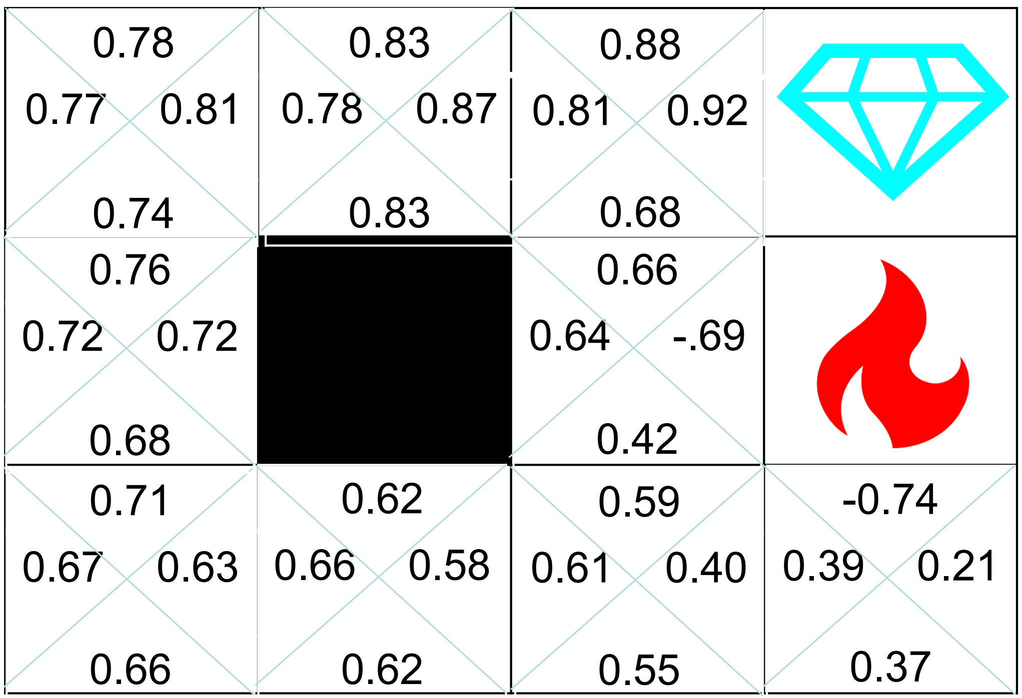
Calculated using a two-step value iteration:

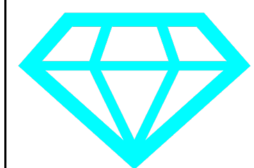
$$q(s,a) = r(s) + \gamma \sum_{s'} P(s'|s,a)u(s')$$

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

Gridworld: Relationship between Q and U

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



0.81	0.87	0.92	
0.76		0.66	
0.71	0.66	0.61	0.39

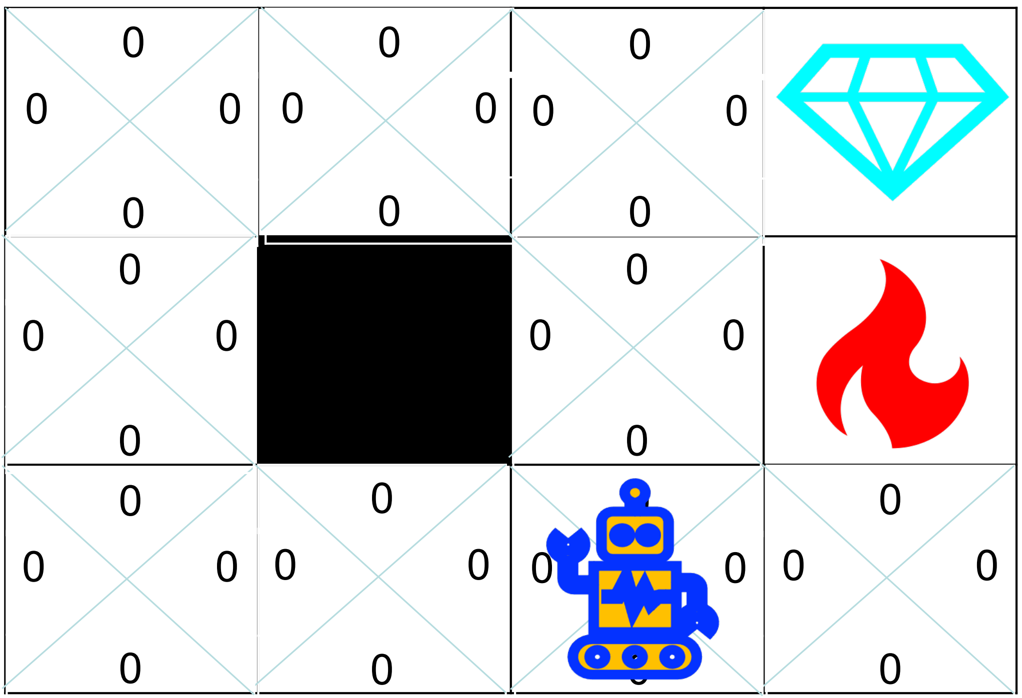
Q-learning

- 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 $u_t(s_{t+1})$.

Example: Gridworld

Suppose we start out with $\underline{\boldsymbol{q_1(s,a) = 0}}$ for all states and actions.



Robot starts out in state s_t=(3,1).

Robot receives a reward of r_t=−0.04.

Robot tries to move UP, ends up in $s_{t+1} = (4,1)$.

Now we update $q_{local}((3,1),\mathrm{UP})$:

$$q_{local}((3,1),\mathrm{UP}) = r((3,1)) + \gamma u_t\big((4,1)\big)$$
$$= -0.04 + 0 = -0.04$$

q-local, the short-time estimate

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

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

Q-local approximates the true quality of an action as:

- 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 $u(s)$, use our current estimate, $u_t(s,a) = \max_a q_t(s,a)$.

TD learning

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

Problem: NOISY!

- s_{t+1} is random, and
- $u_t(s_{t+1})$ is not the real value of q, only our current estimate, therefore
- $q_{local}(s_t, a_t)$ might be very far away from $q(s, a)$!

TD learning

Solutions:

1. If we're measuring using a table: interpolate, using a small learning rate η that's $0 < \eta < 1$:
$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \eta\big(q_{local}(s_t, a_t) - q_t(s_t, a_t)\big)$$

2. If we're measuring using a neural net, with parameters θ: use just one gradient update step, so that θ becomes the average over many successive gradient steps:
$$\theta_{t+1} = \theta_t - \eta \frac{\partial}{\partial \theta} \frac{1}{2} \big(q_t(s_t, a_t) - q_{local}(s_t, a_t)\big)^2$$

TD learning

$q_{local}(s_t, a_t) - q_t(s_t, a_t)$ is called the "time difference" or TD.

1. If the TD is positive, it means action a_t was **<u>better</u>** 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 **<u>worse</u>** 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:

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

3. Calculate the time difference, and update:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \eta\left(q_{local}(s_t, a_t) - q_t(s_t, a_t)\right)$$

or:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial}{\partial \theta} \frac{1}{2}\left(q_t(s_t, a_t) - q_{local}(s_t, a_t)\right)^2$$

TD learning is an off-policy learning algorithm

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

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

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

- Sometimes off-policy learning doesn't converge, 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_{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:

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

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}):
$$q_{local}(s_t, a_t) = r_t + \gamma \max_{a' \in \mathcal{A}} q_t(s_{t+1}, a')$$
- In order to compute the SARSA version of q_{local}, you need to have already picked out $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$:
$$q_{local}(s_t, a_t) = r_t + \gamma q_t(s_{t+1}, a_{t+1})$$

Quiz

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2417564

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
- Policy gradient: estimate a stochastic policy $\pi_a(s) = Pr(A_t = a|S_t = s)$; learn it by maximizing expected total return

Stochastic Policy

- Until now, we've mostly used deterministic policies, $\pi(s_t) = a_t$
- Now we need to a random policy. Say that the agent chooses action a with probability $\pi_a(s)$, thus

$$\boldsymbol{\pi}(s) = \begin{bmatrix} \pi_1(s) \\ \vdots \\ \pi_{|\mathcal{A}|}(s) \end{bmatrix} = \begin{bmatrix} P(A_t = 1|S_t = s) \\ \vdots \\ P(A_t = |\mathcal{A}||S_t = s) \end{bmatrix}$$

Stochastic Policy

$$\boldsymbol{\pi}(s) = \begin{bmatrix} \pi_1(s) \\ \vdots \\ \pi_{|\mathcal{A}|}(s) \end{bmatrix} = \begin{bmatrix} P(A_t = 1|S_t = s) \\ \vdots \\ P(A_t = |\mathcal{A}||S_t = s) \end{bmatrix}$$

- Notice this automatically includes a kind of epsilon-greedy exploration, as long as $\pi_a(s_t) > 0$ for every action
- Usually we calculate $\pi_a(s)$ as the softmax output of a neural network
- … but how do we train the neural network?

Utility = Expected discounted sum of all future rewards

- The policy $\pi_a(s)$ chooses an action at random, then the unknown transition probabilities $P(s'|s,a)$ choose a new state at random, and so on… call this sequence the "trajectory," $\tau = (a_t, s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \dots)$.
- The utility $u(s_t)$ is the expected discounted sum of future rewards:

$$u(s_t) = E[v(\tau)] = \sum_{\tau} P(\mathrm{T} = \tau) v(\tau)$$

…where $v(\tau) = r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \cdots$ is the discounted sum of future rewards corresponding to a particular trajectory τ.

Maximum-utility policy

Suppose $\pi_a(s)$ is a neural net with trainable parameters θ. We'd like to learn θ to maximize utility. Can we do that? Notice that $v(\tau)$ doesn't depend directly on the probabilities, only the probability $P(\tau)$ does:

$$\theta \leftarrow \theta + \eta \frac{\partial u(s_t)}{\partial \theta} = \theta + \eta \sum_{\tau} \frac{\partial P(\tau)}{\partial \theta} v(\tau)$$

Unfortunately, $P(\tau)$ is not so easy to differentiate:

$$P(\tau) = \pi_{a_t}(s_t) P(s_{t+1}|s_t, a_t) \pi_{a_{t+1}}(s_{t+1}) P(s_{t+2}|s_{t+1}, a_{t+1}) \cdots$$

Log probabilities are easier to differentiate than probabilities

Life would be much better if we were differentiating $\ln P(\tau)$:

$$\ln P(\tau) = \ln \pi_{a_t}(s_t) + \ln P(s_{t+1}|s_t, a_t) + \ln \pi_{a_{t+1}}(s_{t+1}) + \cdots$$

Then the solution would be:

$$\frac{\partial \ln P(\tau)}{\partial \theta} = \frac{\partial \ln \pi_{a_t}(s_t)}{\partial \theta} + 0 + \frac{\partial \ln \pi_{a_{t+1}}(s_{t+1})}{\partial \theta} + 0 + \cdots$$

…and if $\pi_a(s)$ is a softmax, then $\ln \pi_a(s)$ is easy to differentiate.

The derivative of a logarithm

If we need to calculate $\theta \leftarrow \theta + \eta \sum_\tau \frac{\partial P(\tau)}{\partial \theta} v(\tau)$, but we only know how to calculate $\frac{\partial \ln P(\tau)}{\partial \theta}$, what can we do?

Here's the trick. Remember that:

$$\frac{\partial \ln P(\tau)}{\partial \theta} = \frac{1}{P(\tau)} \frac{\partial P(\tau)}{\partial \theta}$$

Therefore…

$$\frac{\partial u(s_t)}{\partial \theta} = \sum_\tau \frac{\partial P(\tau)}{\partial \theta} v(\tau) = \sum_\tau P(\tau) \frac{\partial \ln P(\tau)}{\partial \theta} v(\tau) = E\left[\frac{\partial \ln P(\tau)}{\partial \theta} v(\tau)\right]$$

The policy gradient algorithm

1. Play the game k times, and store k different trajectories, $\tau_i = (a_{i,t}, s_{i,t+1}, a_{i,t+1}, s_{i,t+2}, a_{i,t+2}, \dots)$
2. Approximate the expected loss by its average over the minibatch:
$$\mathcal{L} = -\frac{1}{k}\sum_{i=1}^{k} v(\tau_i)\ln P(\tau_i) \approx -E[v(\tau)\ln P(\tau)]$$
3. Backpropagate to maximize utility:
$$\theta \leftarrow \theta - \eta\frac{\partial\mathcal{L}}{\partial\theta} \approx \theta + \eta\frac{\partial u(s_t)}{\partial\theta}$$

Summary: Model-free RL

- Q-learning:

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

then

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \eta\big(q_{local}(s_t, a_t) - q_t(s_t, a_t)\big)$$

or

$$\theta_{t+1} = \theta_t - \eta \frac{\partial}{\partial \theta} \frac{1}{2} \big(q_t(s_t, a_t) - q_{local}(s_t, a_t)\big)^2$$

- Policy gradient:

$$\frac{\partial u(s_t)}{\partial \theta} = \sum_{\tau} \frac{\partial P(\tau)}{\partial \theta} v(\tau) = \sum_{\tau} P(\tau) \frac{\partial \ln P(\tau)}{\partial \theta} v(\tau) = E\left[\frac{\partial \ln P(\tau)}{\partial \theta} v(\tau)\right]$$