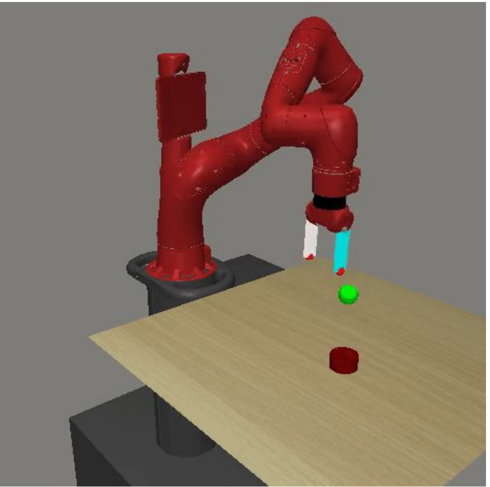
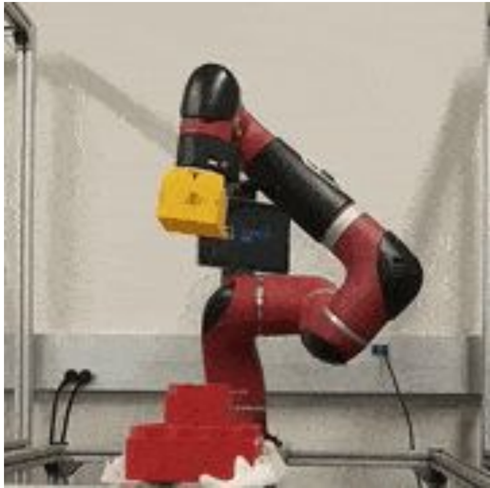
A Brief Tour of Reinforcement Learning



Ben Eysenbach

11/18/20



Learning Objectives

- Recognize when RL is a good approach to solve a problem
- Understand what makes RL challenging
- Be able to implement three RL algorithms

What is RL all about?

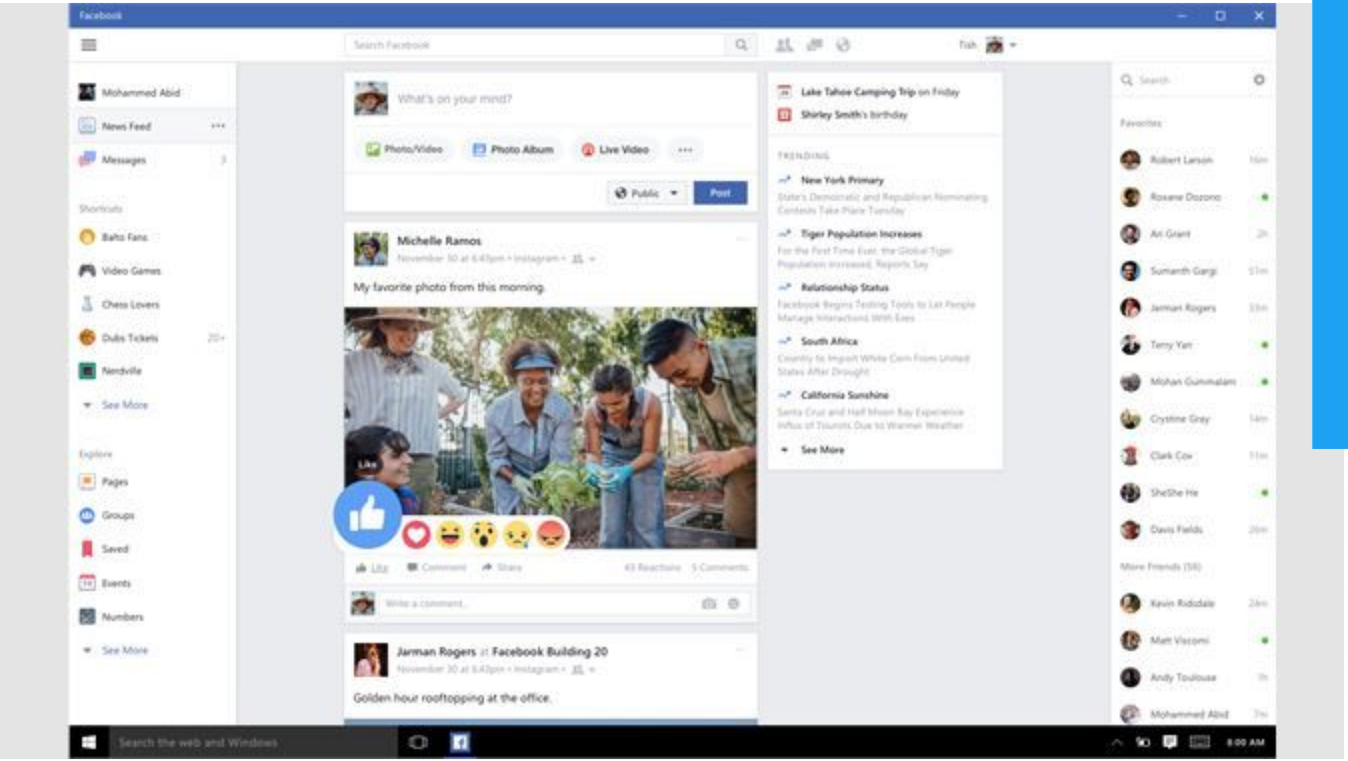
(Hint: It's not just about video games)


Shaker Spoon
Shaker Spoon
Shaker Spoon
Shaker Spoon


INVESTIGACIÓN
Ford
SELF-DRIVING
TEST VEHICLE

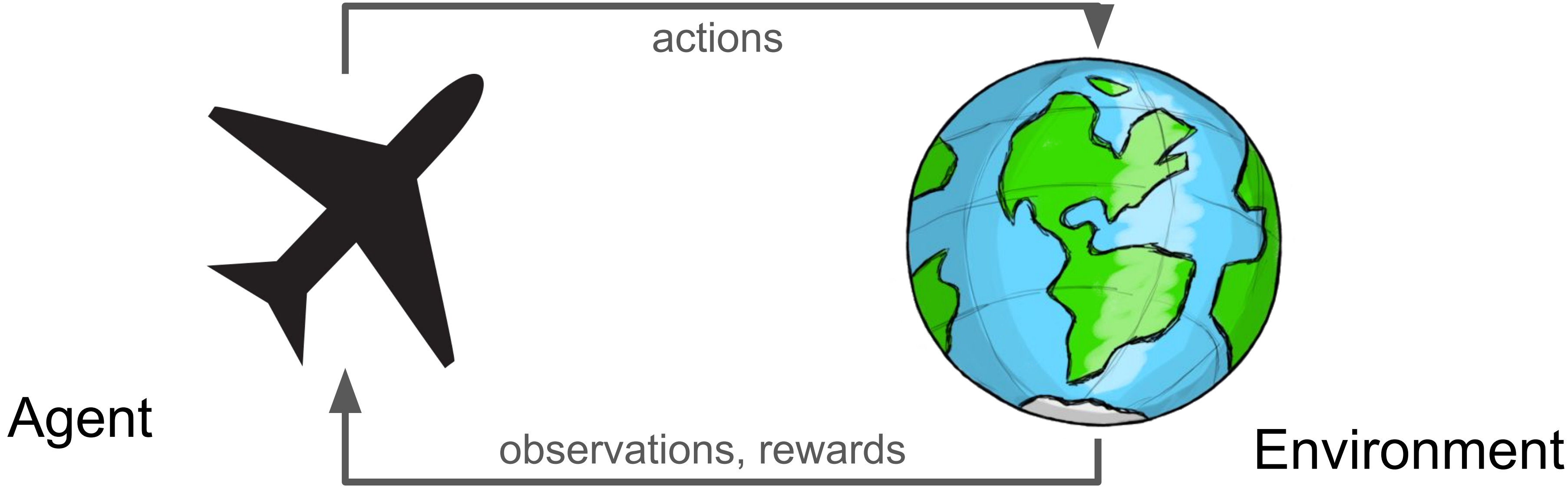

Cloverdale Av
Cochran
Dunsmuir Av
DO NOT BLOCK INTERSECTION
DO NOT BLOCK INTERSECTION
PARKING
Looking East
RUBENS'S


LAIX
LISTED
NYSE
LISTED
NYSE
LAIX
LISTED
NYSE
流利说
LISTED
NYSE
CNBC
GTS
Liulishuo
imc
BARCLAYS
CITADEL | Securities
ice
Intercontinental Exchar
Ra Medical
RMED
NYSE
UTP


Facebook
Mohammed Abid
News Feed
Messages
What's on your mind?
Photo/Video
Photo Album
Live Video
Public
Post
Michelle Ramos
My favorite photo from this morning.
Jarman Rogers at Facebook Building 20
Golden hour rooftopping at the office.
Lake Tahoe Camping Trip on Friday
Shirley Smith's birthday
New York Primary
Tiger Population Increases
Relationship Status
South Africa
California Sunshine
See More

Instagram

Markov Decision Processes (MDPs): The Language of RL



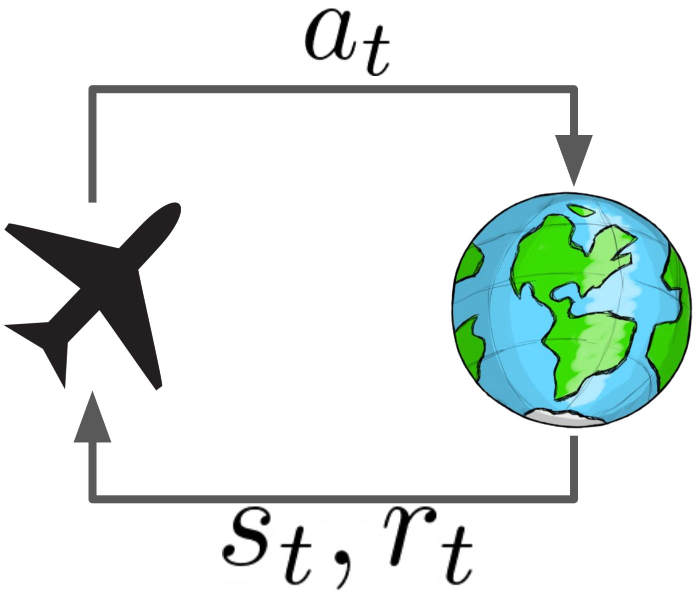
Markov Decision Processes (MDPs): The Language of RL

Rewards: $r(s_t, a_t)$

Dynamics: $p(s_{t+1} \mid s_t, a_t)$

Policy: $\pi_\theta(a_t \mid s_t)$

Objective: $\max_\pi E_\pi \left[\sum_t \gamma^t r(s_t, a_t) \right]$



Markov Decision Processes (MDPs): The Language of RL

Markov Assumption (informal): The current observation contains all information necessary to predict the next state and reward. (Memory isn't useful.)

Examples?

- Deterministic dynamics?
- Stochastic dynamics?
- Frames from Atari games?
- Medical records?
- Egocentric camera?

What makes RL harder than supervised learning?

- Dynamics are not known; can't differentiate through them.
- Supervised learning tells us the correct label ("action") for each state.
- Credit assignment: which actions contributed to future reward/penalties?
- Exploration: how do you find states with high reward?
- Data distribution depends on the policy.

Algorithms for RL

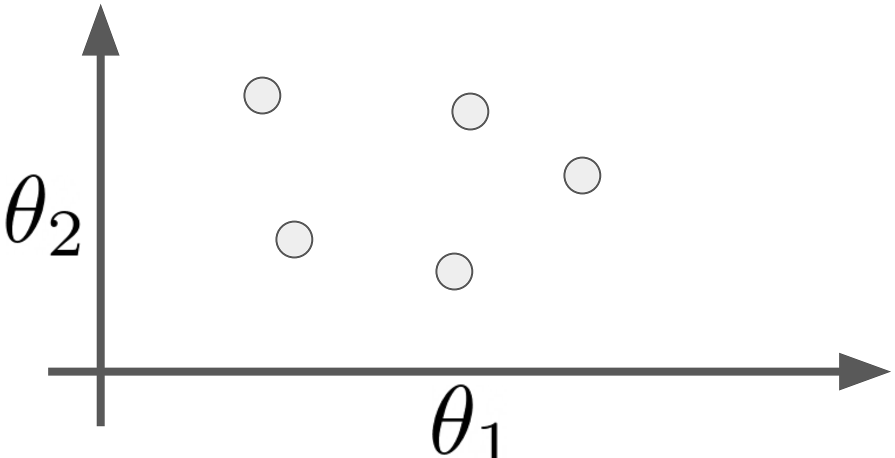
1. Black box optimization
2. REINFORCE
3. Q-Learning

Approach 1: Black-box optimization

- Expensive to evaluate
- Can't take gradients

$$f(\theta) = E_{\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right]$$

One black-box optimizer: CMA-ES [Hansen 16] (Covariance matrix adaptation evolution strategy)

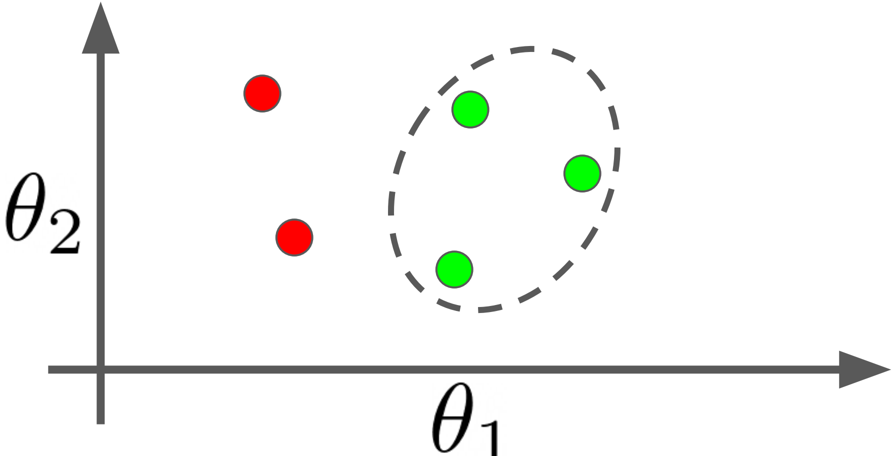


Approach 1: Black-box optimization

- Expensive to evaluate
- Can't take gradients

$$f(\theta) = E_{\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right]$$

One black-box optimizer: CMA-ES [Hansen 16] (Covariance matrix adaptation evolution strategy)

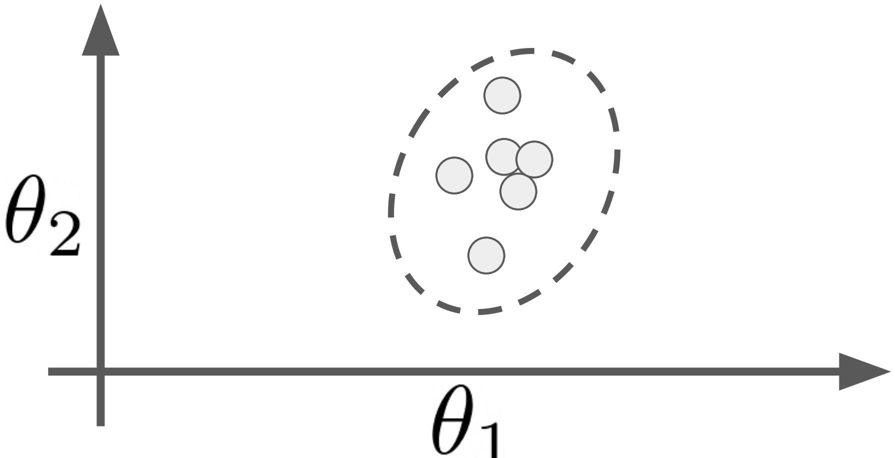


Approach 1: Black-box optimization

- Expensive to evaluate
- Can't take gradients

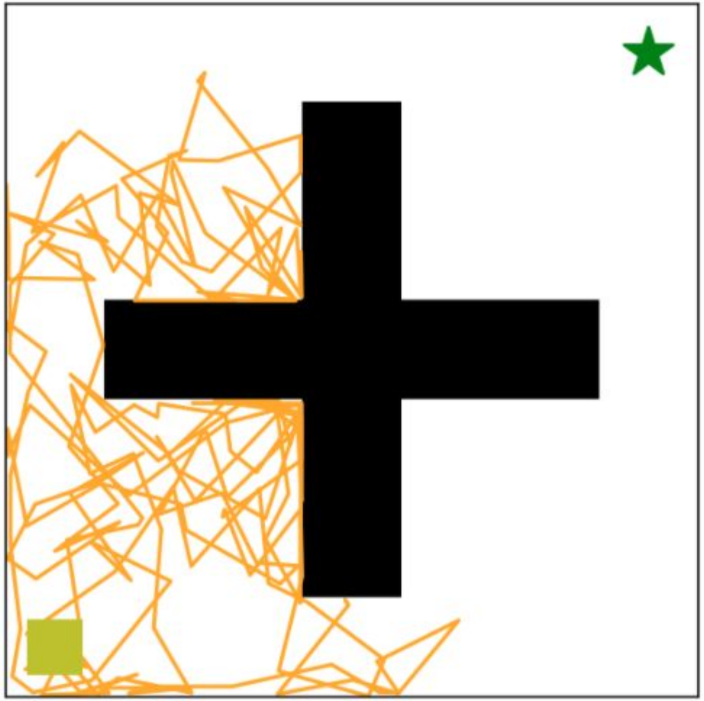
$$f(\theta) = E_{\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right]$$

One black-box optimizer: CMA-ES [Hansen 16] (Covariance matrix adaptation evolution strategy)



Approach 1: Black-box optimization

$$r(s_t, a_t) = -d(s_t, \bigstar)$$



```
def objective_fn(theta):
  s = get_initial_state()
  total_r = 0
  for t in range(NUM_STEPS):
    a = policy(s, theta)
    r = reward_fn(s, a)
    total_r += GAMMA**t * r
    s = dynamics_fn(s, a)
  return total_r

theta_opt = CMAES(objective_fn)
```

Approach 1: Black-box optimization

CMA-ES Random Search

Approach 2: REINFORCE

Let's just do gradient descent! $\nabla_\theta E_{\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right] = ???$

$$E_{p_\theta(x)}[f(x)] = \int p_\theta(x) f(x) dx$$

$$\nabla_\theta E_{p_\theta(x)}[f(x)] = \int \nabla_\theta p_\theta(x) f(x) dx$$

Approach 2: REINFORCE

$$\nabla_\theta E_{p_\theta(x)}[f(x)] = \int \boxed{\nabla_\theta p_\theta(x)} f(x) dx$$

$$\boxed{\text{Useful identity}: \quad \nabla_\theta \log p_\theta(x) = \frac{1}{p_\theta(x)} \boxed{\nabla_\theta p_\theta(x)}}$$

$$\nabla_\theta E_{p_\theta(x)}[f(x)] = \int p_\theta(x) \nabla_\theta \log p_\theta(x) f(x) dx = E_{p_\theta(x)}[f(x) \nabla_\theta \log p_\theta(x)]$$

```
for _ in range(num_iter):
  x_vec = [p(theta).sample() for __ in range(num_samples)]
  grad_vec = [log(p(x, theta)).grad(theta) for x in x_vec]
  grad = mean([f(x) * grad for (x, grad) in zip(x_vec, grad_vec)]
  theta += learning_rate * grad
```

Approach 2: REINFORCE

REINFORCE gradient

$$E_{p_\theta(x)}[f(x)\nabla_\theta \log p_\theta(x)]$$

Maximum likelihood gradient

$$E_{p^*(x)}[\nabla_\theta \log p_\theta(x)]$$

$$p_\theta(x)f(x) \approx p^*(x)$$

Intuition: REINFORCE is just maximum likelihood (i.e., supervised learning) on *weighted* data

Approach 2: REINFORCE

$$\nabla_\theta E_{p_\theta(x)}[f(x)] = \int p_\theta(x)\nabla_\theta \log p_\theta(x) f(x)dx = E_{p_\theta(x)}[f(x)\nabla_\theta \log p_\theta(x)]$$

Applying to the RL problem:

$$x \leftarrow \tau = (s_1, a_1, s_2, a_2, \cdots)$$

$$f(x) \leftarrow \sum_t \gamma^t r(s_t, a_t)$$

$$p_\theta(x) \leftarrow p_\theta(\tau) = \prod_t p(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t \mid s_t)$$

What about $\nabla_\theta \log p_\theta(\tau)$?

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \left(\cancel{\sum_t \log p(s_{t+1} \mid s_t, a_t)} + \sum_t \log \pi_\theta(a_t \mid s_t) \right)$$

Approach 2: REINFORCE

REINFORCE gradient

$$E_{\pi_\theta}\left[\left(\sum_t \gamma^t r(s_t, a_t)\right)\left(\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

Maximum likelihood gradient

$$E_{\pi^*}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

$$R(\tau)\pi_\theta(\tau) \approx \pi^*(\tau)$$

Intuition: REINFORCE is just maximum likelihood (i.e., supervised learning) on ***reward***-*weighted* data

Approach 2: REINFORCE

$$E_{\pi_\theta}\left[\left(\sum_t \gamma^t r(s_t, a_t)\right)\left(\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

```
for _ in range(num_iter):
  traj_vec = [p(theta).sample() for __ in range(num_samples)]
  grad_vec = []
  r_vec = []
  for traj in traj_vec:
    grad = sum([log(policy(s, a, theta)).grad(theta) for (s, a, r) in traj])
    grad_vec.append(grad)
    r_vec.append(sum([gamma**t * r for (t, (s, a, r)) in enumerate(traj)])
  grad = mean([r * grad for (r, grad) in zip(r_vec, grad_vec)]
  theta += learning_rate * grad
```

Review of RL algorithms so far

Black-box optimization

- Stochastic and deterministic policies
- Ignores gradients of the *policy*
- Challenging to scale to high-dim problems.
- Collects new data at each iteration

REINFORCE

- Requires stochastic policies (why?)
- Utilizes policy gradient
    - Algorithms based on REINFORCE are known as **policy gradient methods.**
- Collects new data at each iteration

Can we do RL without collecting more data at each iteration?

Approach 3: Q-learning

Idea: Estimate the expected reward *from each state and action*

$$Q_\phi(s, a) = E_{\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t) \mid s, a\right]$$

If we could learn Q(s, a), getting the optimal policy is easy!

$$\pi(s) = \arg\max_a Q_\phi(s, a)$$

Approach 3: Q-learning

So, how are we going to learn the Q function?

$$\text{Regression}: \left\{ \left(s, a, \sum_t \gamma^t r(s_t, a_t)\right) \right\}$$

What reward will we get in the future if we act according to our *current* policy?

How can we estimate the Q-function of a *different* policy?

Approach 3: Q-learning

Recursive relationship between Q values:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma E_{\substack{p(s_{t+1}|s_t,a_t), \\ \pi(a_{t+1}|s_{t+1})}} [Q(s_{t+1}, a_{t+1})]$$

$$\left\{(s, a, \sum_t \gamma^t r(s_t, a_t))\right\} \Rightarrow \{(s, a, r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}))\}$$

We're using our own predictions as *labels*!
("Bootstrapping")

Summary of today's lecture

- What is RL?
- Why is RL hard?
- What are some approaches to solving RL?
    1. Black-box optimization
    2. REINFORCE
    3. Q-learning
- [Next] AMA with RL PhD student (me)