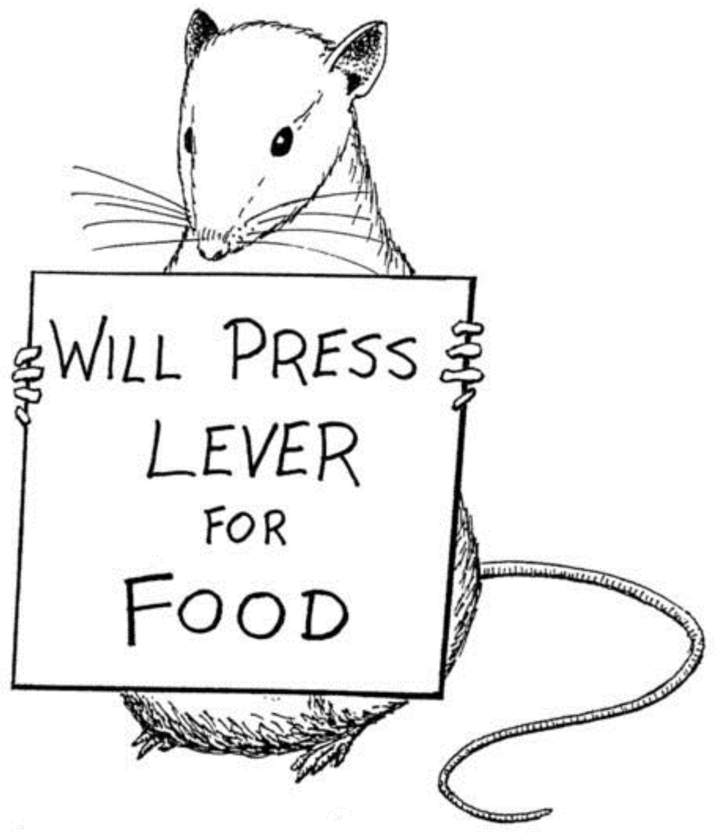
CS 277: Control and Reinforcement Learning

Winter 2026

Lecture 4: Deep Q-Learning

Roy Fox
Department of Computer Science
School of Information and Computer Sciences
University of California, Irvine



Logistics

assignments

- Exercise 1 due tomorrow
- Quiz 2 due next Monday

Q function

- To approach V_π when we update $V(s) \to r + \gamma V(s')$, we need on-policy data
  - Roll out π to see transition $(s, a) \to s'$ with reward r
- On-policy data is expensive: need more every time π changes
- Action-value function: $Q_\pi(s, a) = \mathbb{E}_{\xi \sim p_\pi}[R \,|\, s_0 = s, a_0 = a]$
  - Compare: $V_\pi(s) = \mathbb{E}_{\xi \sim p_\pi}[R \,|\, s_0 = s] = \mathbb{E}_{(a|s) \sim \pi}[Q_\pi(s, a)]$

- Action-value backward recursion: $Q_\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{(s'|s,a) \sim p}[V_\pi(s')]$
  - Broke down $V_\pi(s) = \mathbb{E}_{(a|s) \sim \pi}[r(s, a) + \gamma \mathbb{E}_{(s'|s,a) \sim p}[V_\pi(s')]]$ into two parts

MF

θ

DP

TD from off-policy data

- Backward recursion in two parts:

$$V_\pi(s) = \mathbb{E}_{(a|s)\sim\pi}[Q_\pi(s,a)] \qquad Q_\pi(s,a) = r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V_\pi(s')]$$

- This should hold in every state and action
  - (s,a) can be sampled from any distribution $p_{\pi'}$ for any alternative π'
- Put together, we update $Q(s,a) \to r + \gamma\mathbb{E}_{(a'|s')\sim\pi}[Q(s',a')]$
  - For any distribution of (s,a), giving reward r and following state $s' \sim p(\cdot\,|\,s,a)$
  - In other words: $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma\mathbb{E}_{(a'|s')\sim\pi}[Q(s',a')] - Q(s,a))$

temporal difference

MF

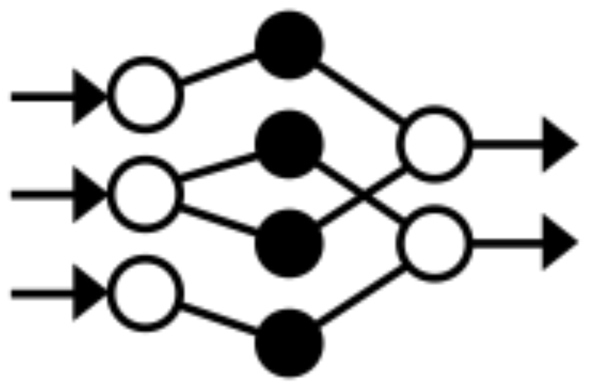
θ

DP

π'

TD with function approximation

- With large state space: represent $V_\theta : S \to \mathbb{R}$ or $Q_\theta : S \times A \to \mathbb{R}$



- Instead of the update $V(s) \to r + \gamma V(s')$
  - Descend on square loss $\mathscr{L}_\theta = (r + \gamma V_{\bar{\theta}}(s') - V_\theta(s))^2$
  - On on-policy experience (s, a, r, s')

only learn $V_\theta(s)$
$V_{\bar{\theta}}(s')$ is the target
$\Rightarrow$ don't take its gradient!

MF
θ
DP
π'

- Instead of the update $Q(s, a) \to r + \gamma \mathbb{E}_{(a'|s') \sim \pi}[Q(s', a')]$
  - Descend on square loss $\mathscr{L}_\theta = (r + \gamma \mathbb{E}_{(a'|s') \sim \pi}[Q_{\bar{\theta}}(s', a')] - Q_\theta(s, a))^2$
  - On off-policy experience (s, a, r, s')

only learn $Q_\theta(s, a)$
$Q_{\bar{\theta}}(s', a')$ is the target
$\Rightarrow$ don't take its gradient!

MF
θ
DP
π'

Today's lecture

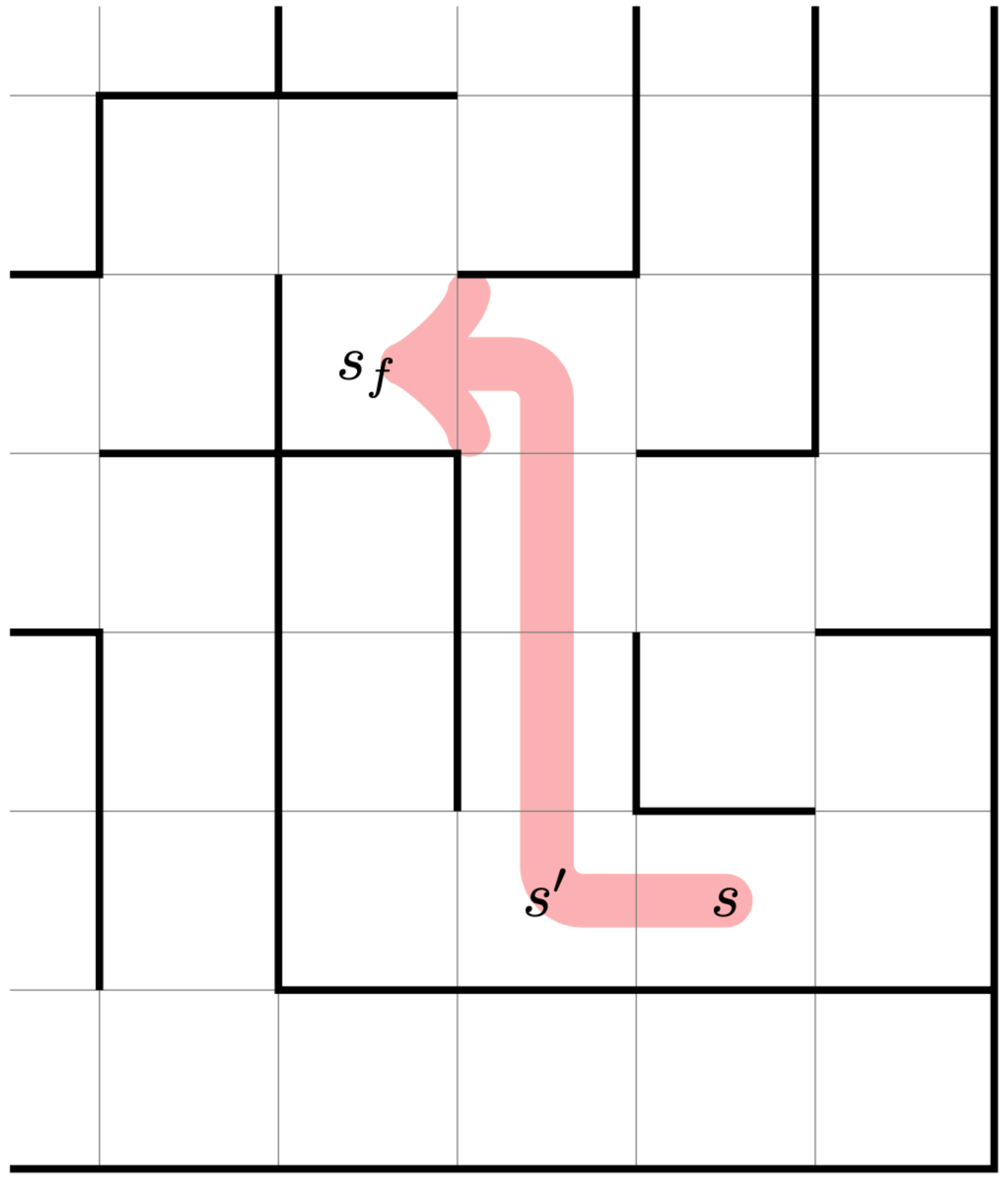
Policy Improvement

Fitted Q-Iteration

Deep Q-Learning

DQN tricks

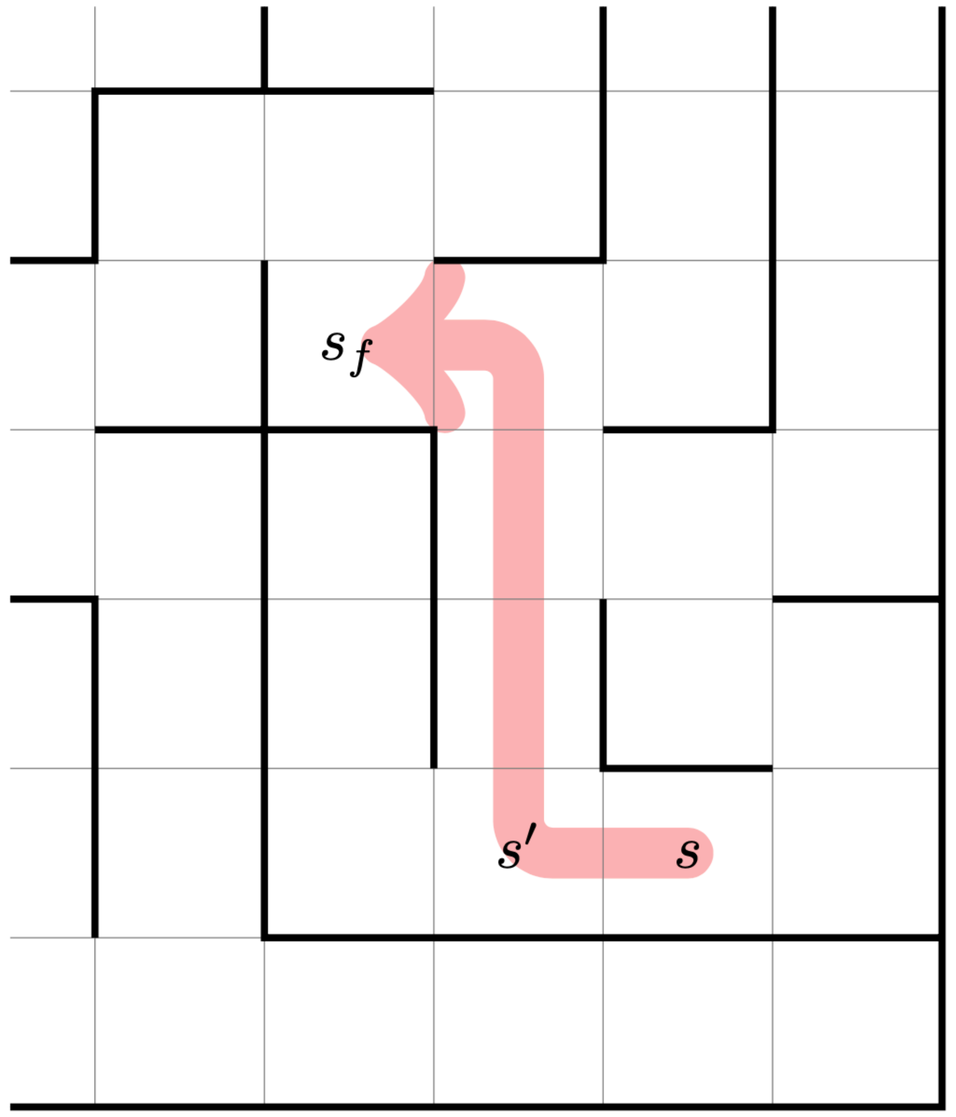
Special case: shortest path



- Deterministic dynamics: in state s, take action a to get to state $s' = f(s, a)$
  - Example above: $s' = f(s, a_{\text{left}})$
- Reward: (-1) in each step (until the goal s_f is reached)

Shortest path: optimality principle

- Proposition: ξ is shortest from s to s_f through $s' \Rightarrow$ suffix of ξ is shortest from s' to s_f
- Proof: otherwise, let ξ' be a shorter path from s' to s_f, then take $s \xrightarrow{\xi} s' \xrightarrow{\xi'} s_f$
- The proposition is "if" but not "only if", because we don't know which s' is best
  - Try them all: for each a, try $s' = f(s, a)$



- Let $V(s)$ be the shortest path length from s to s_f
  - For each candidate s', the shortest path through it is $1 + V(s')$
  - For all $s \neq s_f$, we have $V(s) = \min_a (1 + V(f(s, a)))$

Bellman-Ford shortest path algorithm

MF

θ

DP

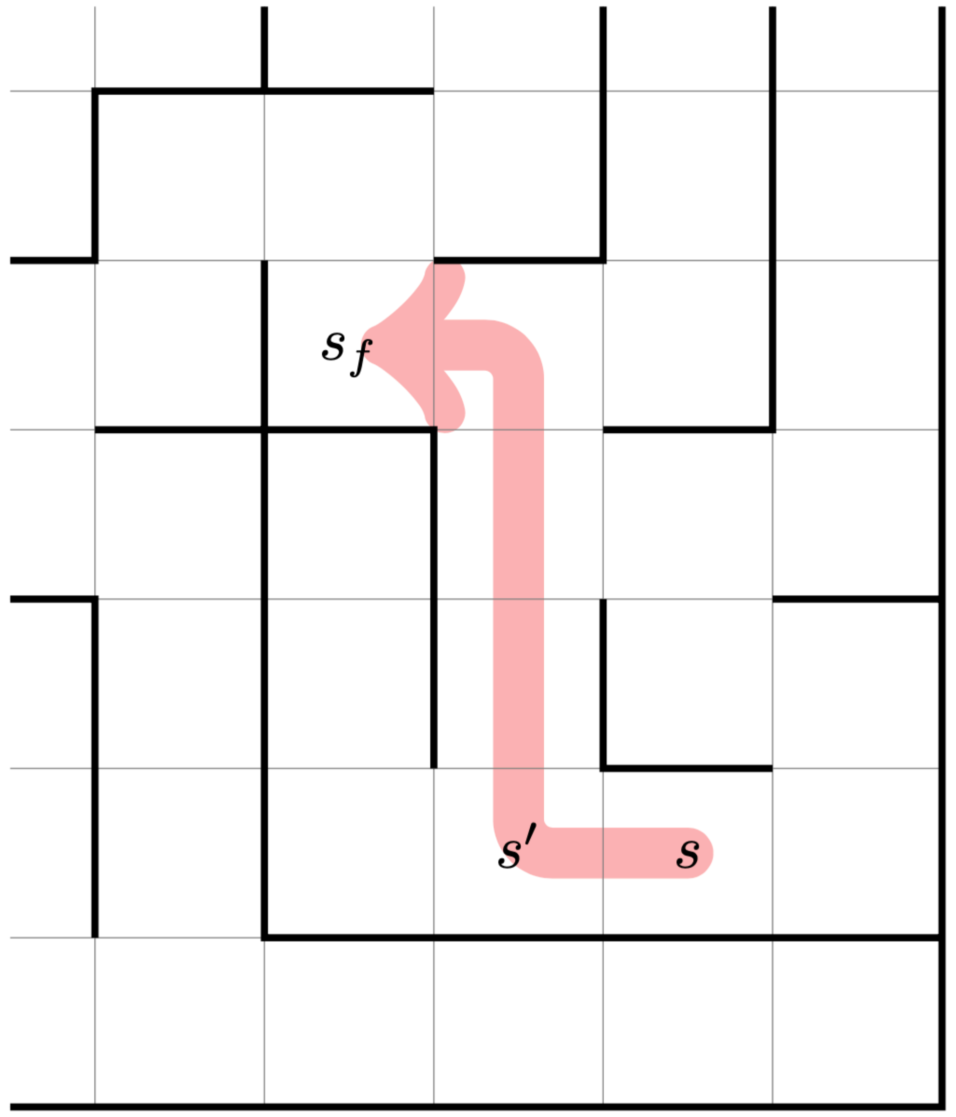
π'

max

- For all $s \neq s_f$, we have $V(s) = \min_a (1 + V(f(s,a)))$

Algorithm Bellman-Ford

$V(s_f) \leftarrow 0$
$V(s) \leftarrow \infty$ for each non-terminal state s
for $|S| - 1$ iterations
  for each non-terminal state s
    $V(s) \leftarrow \min_{a \in A}(1 + V(f(s,a)))$



- The optimal policy is $\pi(s) = \arg\min_a (1 + V(f(s,a)))$

[Ford and Fulkerson, 1962]

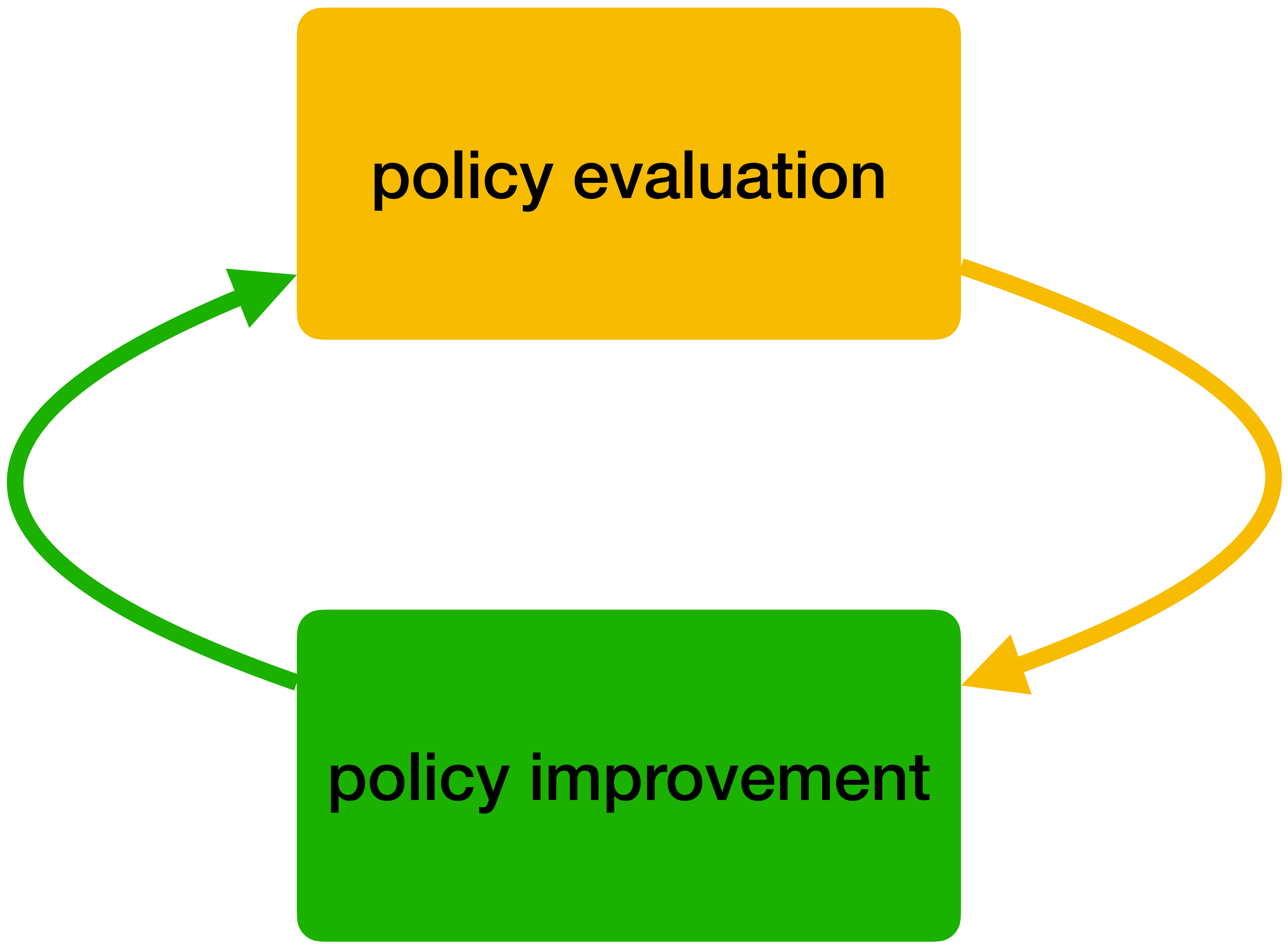
Policy improvement

- A value function suggests the greedy policy:

$$\pi(s) = \arg\max_a Q(s,a) = \arg\max_a (r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V(s')])$$

- The greedy policy may not be the optimal policy $\pi^* = \arg\max_\pi J_\pi$
  - But is the greedy policy always an improvement?
- Proposition: the greedy policy for Q_π (value of π) is never worse than π
- Corollary (Bellman optimality): if π is greedy for its value Q_π then it is optimal
  - In a finite MDP, the iteration $\pi \xrightarrow{\text{evaluate}} Q_\pi \xrightarrow{\text{greedy}} \pi$ converges, and then π is optimal

The RL scheme



Policy Iteration

MF

θ

DP

π'

max

- If we know the MDP (model-based), we can just alternate evaluate/greedy:

Algorithm Policy Iteration

Initialize some policy π
repeat
  Evaluate the policy $Q(s, a) \leftarrow \mathbb{E}_{\xi \sim p_\pi}[R | s_0 = s, a_0 = a]$
  Update to the greedy policy $\pi(s) \leftarrow \arg\max_a Q(s, a)$

- Upon convergence, $\pi = \pi^*$ and $Q = Q^*$

Value Iteration

MF
θ
DP
π'
max

- We can also alternate evaluate/greedy inside the loop over states:

Algorithm Value Iteration

Initialize some value function V
repeat
  for each state s
    Update $V(s) \leftarrow \max_a(r(s,a)+\gamma\,\mathbb{E}_{(s'|s,a)\sim p}[V(s')])$

- Must update each state repeatedly until convergence

- Upon convergence, $\pi^*(s) = \arg\max_a(r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V(s')])$

Generalized Policy Iteration

MF

θ

DP

π'

max

- We can even alternate in any order we wish:

$$V(s) \leftarrow \mathbb{E}_{(a|s)\sim\pi}[r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V(s')]]$$

$$\pi(s) \leftarrow \arg\max_a (r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V(s')])$$

- As long as each state gets each of the two update without starvation
  - The process will eventually converge to V^* and π^*

Model-free reinforcement learning

- We can be model-free using MC policy evaluation:

Algorithm MC model-free RL

$$
\begin{array}{l}
\text{Initialize some policy } \pi \\
\textbf{repeat} \\
\quad \text{Initialize some value function } Q \\
\quad \textbf{repeat} \text{ to convergence} \\
\qquad \text{Sample } \xi \sim p_\pi \\
\qquad \text{Update } Q(s_t, a_t) \to R_{\geq t}(\xi) \text{ for all } t \geq 0 \\
\quad \pi(s) \leftarrow \arg\max_a Q(s, a) \text{ for all } s
\end{array}
$$

- On-policy policy evaluation in the inner loop — very inefficient

- We could also do this with function approximation

MF
θ
DP
π'
max
MF
θ
DP
π'
max

Off-policy model-free reinforcement learning

- Value iteration is model-based: $V(s) \leftarrow \max_a (r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V(s')])$

- Action-value version: $Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[\max_{a'} Q(s',a')]$

- A model-free (data-driven) version — Q-Learning:
  - On off-policy data (s,a,r,s'), update

$$Q(s,a) \to r + \gamma \max_{a'} Q(s',a')$$

MF

θ

DP

π'

max

[Watkins and Dayan, 1992]

Recap

- RL is a (policy evaluation ↔ policy improvement) loop
- Policy evaluation: model-based, Monte Carlo, or Temporal-Difference
  - Temporal-Difference exploits the sequential structure using dynamic programming
- TD can be off-policy by considering the action-value Q function
  - Off-policy data can be thrown out less often as the policy changes
- Policy improvement can be greedy
  - Arbitrarily alternated with policy evaluation of any kind (MB, MC, or TD)
- Many approaches can be made differentiable for Deep RL

Today's lecture

Policy Improvement

Fitted Q-Iteration

Deep Q-Learning

DQN tricks

Fitted Value-Iteration (FVI)

Algorithm Value Iteration

Initialize some value function V

repeat

for each state s

Update $V(s) \leftarrow \max_a (r(s,a) + \gamma\, \mathbb{E}_{(s'|s,a)\sim p}[V(s')])$

- Fitted Value-Iteration (FVI):

$$\theta^{i+1} \leftarrow \arg\min_\theta \mathbb{E}_{s\sim\mu}[(\max_a(r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V_{\theta^i}(s')]) - V_\theta(s))^2]$$

square error

  - For some state distribution μ
  - Can use losses other than square

MF
θ
DP
π'
max

MF
θ
DP
π'
max

Fitted Q-Iteration (FQI)

- Fitted Value-Iteration (FVI):

$$\theta^{i+1} \leftarrow \arg\min_\theta \mathbb{E}_{s\sim\mu}[(\max_a(r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V_{\theta^i}(s')]) - V_\theta(s))^2]$$

- Action-value iteration: $Q(s,a) \leftarrow r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[\max_{a'} Q(s',a')]$

- Fitted Q-Iteration (FQI):

$$\theta^{i+1} \leftarrow \arg\min_\theta \mathbb{E}_{(s,a)\sim\mu}[(r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[\max_{a'} Q_{\theta^i}(s',a')]) - Q_\theta(s,a))^2]$$

  - For some state-action distribution μ

MF θ DP π' max

MF θ DP π' max

Q-Learning

MF θ DP π' max

Algorithm Q-Learning

Initialize Q

$s \leftarrow$ reset state

repeat

  Take some action a

  Receive reward r

  Observe next state s'

  Update $Q(s,a) \to \begin{cases} r & s' \text{ terminal} \\ r + \gamma \max_{a'} Q(s',a') & \text{otherwise} \end{cases}$

  $s \leftarrow$ reset state if s' terminal, else $s \leftarrow s'$

[Watkins and Dayan, 1992]

Sampling-based Fitted Q-Iteration

- FQI can be model-free by sampling from p
  - We can sample using environment interaction with some π', if $\mu = p_{\pi'}$
  - Or sample using a simulator we can reset to any state $s \sim \mu$
  - Anyway, this is off-policy from the greedy policy $\arg\max_a Q_\theta(s, a)$

MF

θ

DP

π'

max

Algorithm Sampling-based Fitted Q-Iteration

Initialize θ
repeat
Sample a batch $(\vec{s}, \vec{a}) \sim \mu$
Feed to simulator to get batch $(\vec{r}, \vec{s}')$
Descend $\mathcal{L}_\theta = (\vec{r} + \gamma \max_{\vec{a}'} Q_{\bar{\theta}}(\vec{s}', \vec{a}') - Q_\theta(\vec{s}, \vec{a}))^2$

[Munos and Szepesvári, 2008]

Today's lecture

Policy Improvement

Fitted Q-Iteration

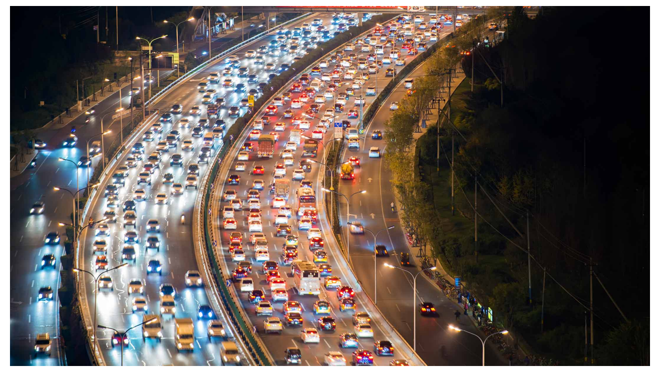
Deep Q-Learning

DQN tricks

Experience policy

- Which distribution should the training data have?
  - The policy may not be good on other distributions / unsupported states
  - $\Rightarrow$ ideally, the test distribution p_π for the final π
- On-policy methods (e.g. MC): must use on-policy data (from the current π)
- Off-policy methods (e.g. Q) can use different policy (or even non-trajectories)
  - But both should eventually use p_π or suffer train–test distribution mismatch

Exploration policies



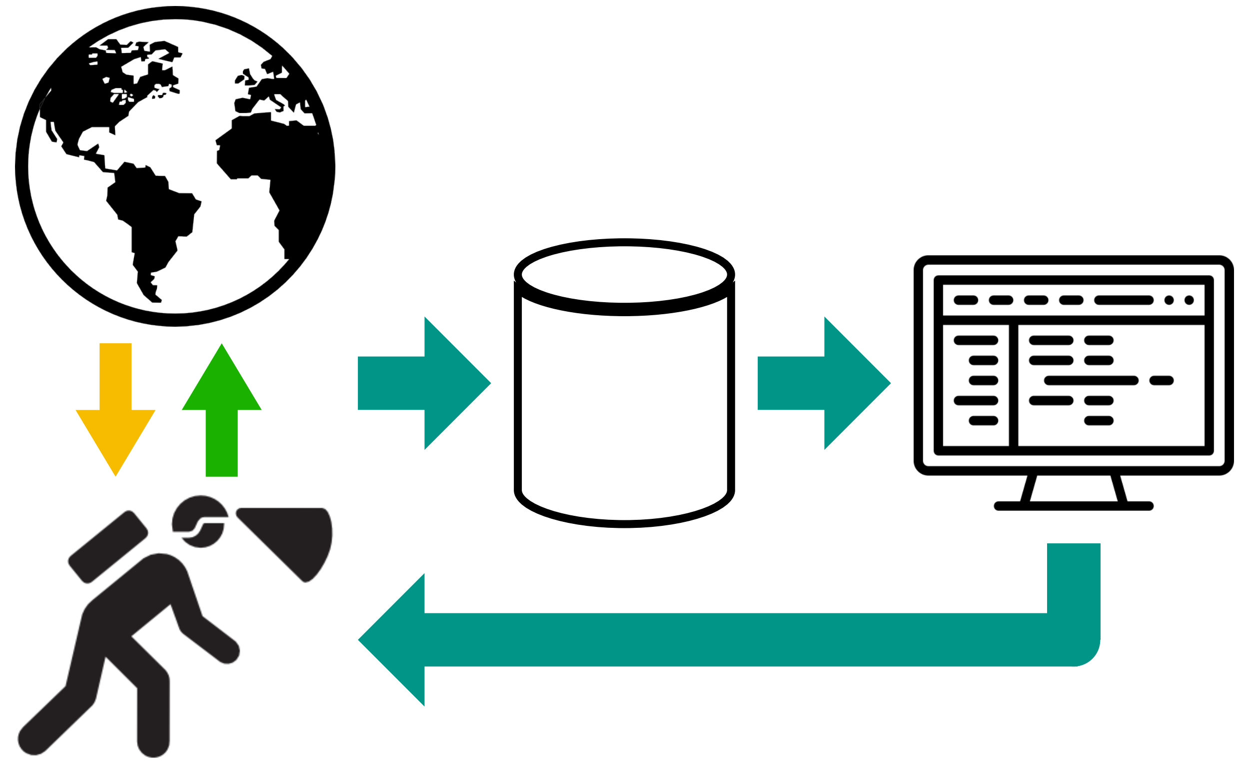
- Example: I tried route 1: {40, 20, 30}; route 2: {30, 25, 40}
  - Suppose route 1 really has expected time 30min, should you commit to it forever?
- To avoid overfitting, we must try all actions infinitely often
- ϵ-greedy exploration: select uniform action with prob. ϵ, otherwise greedy
- Boltzmann exploration:

$$\pi(a \mid s) = \operatorname*{soft\,max}_a(Q(s,a);\beta) = \frac{\exp(\beta Q(s,a))}{\sum_{\bar{a}} \exp(\beta Q(s,\bar{a}))}$$

  - Becomes uniform as the inverse temperature $\beta \to 0$, greedy as $\beta \to \infty$

Experience replay

- On-policy methods are inefficient: throw out all data with each policy update
- Off-policy methods can keep the data = experience replay
  - Replay buffer: dataset of past experience
  - Diversifies the experience (beyond current trajectory)
- Variants differ on
  - How often to add data vs. sample data
  - How to sample from the buffer
  - When to evict data from the buffer, and which

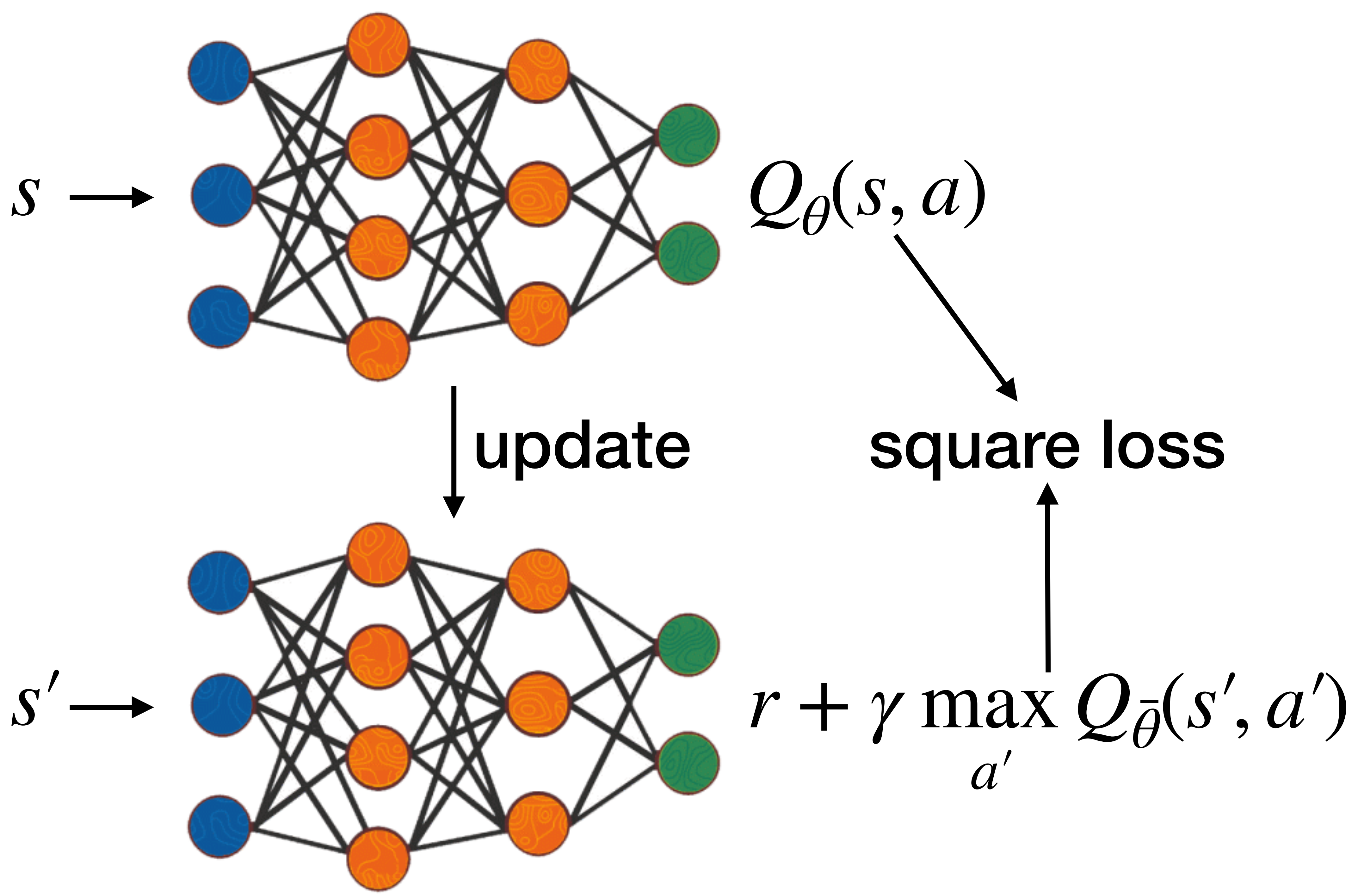


Why use target network?

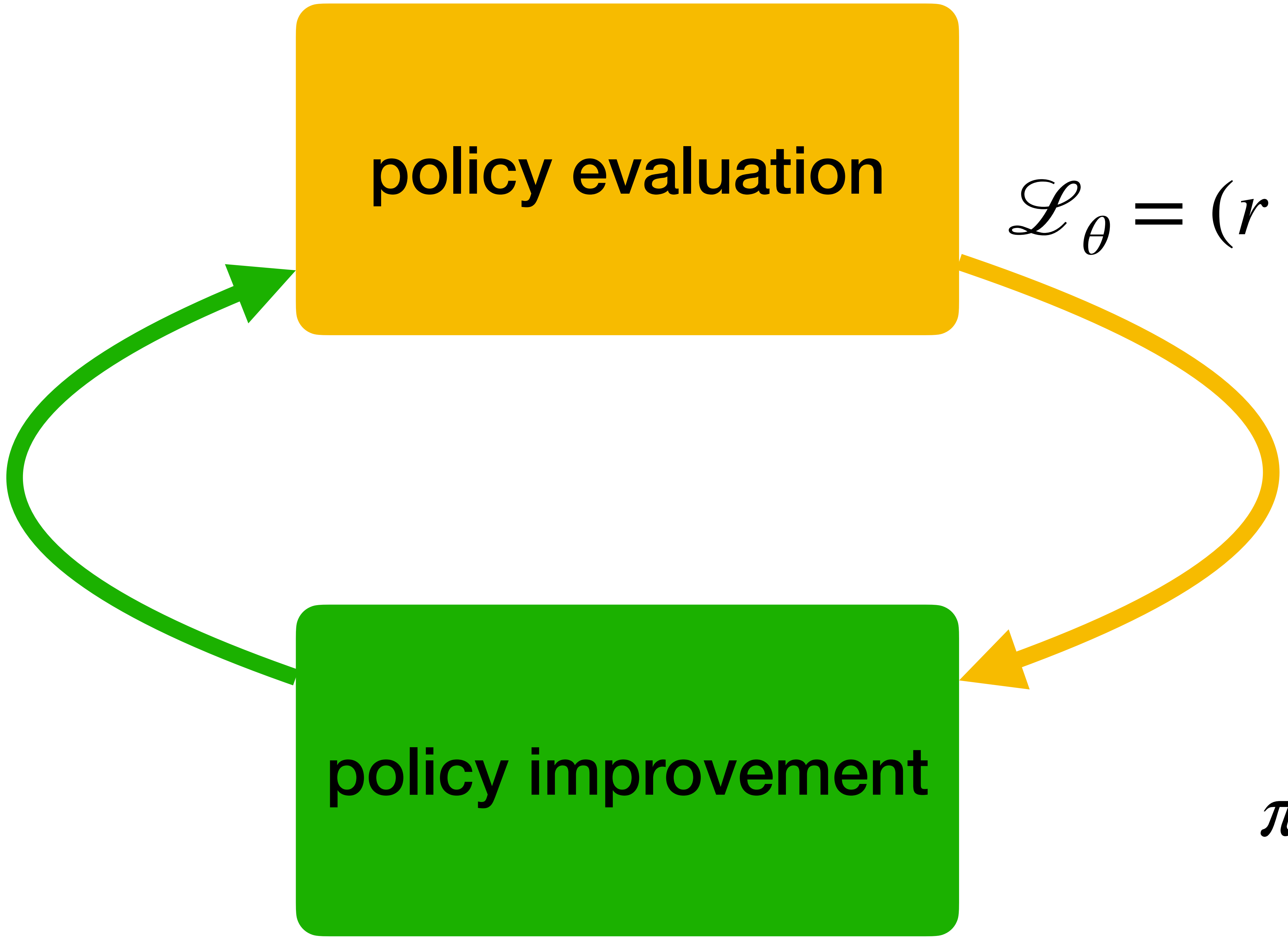
- Fitted-Q loss: $\mathscr{L}_\theta = (r + \gamma \max_{a'} Q_{\bar\theta}(s', a') - Q_\theta(s, a))^2$

  no gradient from the target term

- Target network = lagging copy of $Q_\theta(s, a)$
  - Periodic update: $\bar\theta \leftarrow \theta$ every T_{target} steps
  - Exponential update: $\bar\theta \leftarrow (1 - \eta)\bar\theta + \eta\theta$
- $Q_{\bar\theta}$ is more stable
  - Less of a moving target
  - Less sensitive to data $\Rightarrow$ less variance
- But $\bar\theta \neq \theta$ introduces bias



Putting it all together: DQN



fitted-Q loss

$$\mathscr{L}_\theta = (r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') - Q_\theta(s, a))^2$$

greedy policy

$$\pi(s) = \arg\max_a Q(s, a)$$

exploration

e.g. ϵ-greedy

Deep Q-Learning (DQN)

Algorithm DQN

Initialize θ, set $\bar{\theta} \leftarrow \theta$

$s \leftarrow$ reset state

for each interaction step

Sample $a \sim \epsilon$-greedy for $Q_\theta(s, \cdot)$

Get reward r and observe next state s'

Add (s, a, r, s') to replay buffer $\mathcal{D}$

Sample batch $(\vec{s}, \vec{a}, \vec{r}, \vec{s}') \sim \mathcal{D}$

$$y_i \leftarrow \begin{cases} r_i & s'_i \text{ terminal} \\ r_i + \gamma \max_{a'} Q_{\bar{\theta}}(s'_i, a') & \text{otherwise} \end{cases}$$

Descend $\mathcal{L}_\theta = (\vec{y} - Q_\theta(\vec{s}, \vec{a}))^2$

every T_{target} steps, set $\bar{\theta} \leftarrow \theta$

$s \leftarrow$ reset state if s' terminal, else $s \leftarrow s'$

MF

θ

DP

π'

max

[Mnih et al., 2015]

Today's lecture

- Policy Improvement
- Fitted Q-Iteration
- Deep Q-Learning
- **DQN tricks**

Value estimation bias

- Q-value estimation is optimistically biased
- Jensen's inequality: for a random vector Q

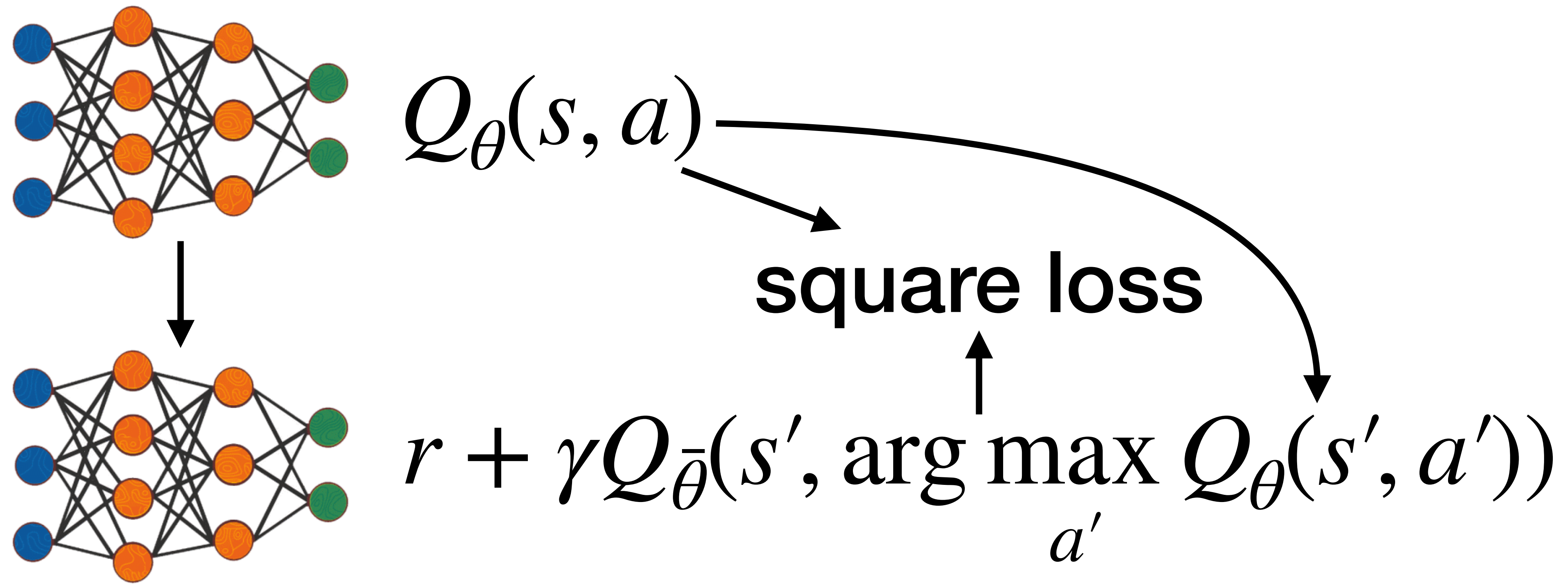
$$\mathbb{E}[\max_a Q_a] \geq \max_a \mathbb{E}[Q_a]$$

- While there's uncertainty in $Q_{\bar{\theta}}$, $\max_{a'} Q_{\bar{\theta}}(s', a')$ is positively biased
- So how can this converge?
  - As certainty increases, the bias of each update decreases
  - Existing bias attenuates with repeated discounting by γ

Double Q-Learning

- Idea: keep two value estimates Q_1 and Q_2
  - Update: $Q_i(s,a) \to r + \gamma Q_{-i}(s', \arg\max_{a'} Q_i(s',a'))$

    $-i$ = the other
- How to use this with DQN?
- Idea 1: use target network as the other estimate



- Idea 2: Clipped Double Q-Learning

$$Q_{\theta_i}(s,a) \to r + \gamma \min_{i=1,2} Q_{\bar\theta_i}(s', \arg\max_{a'} Q_{\theta_i}(s',a'))$$

[van Hasselt, 2010]

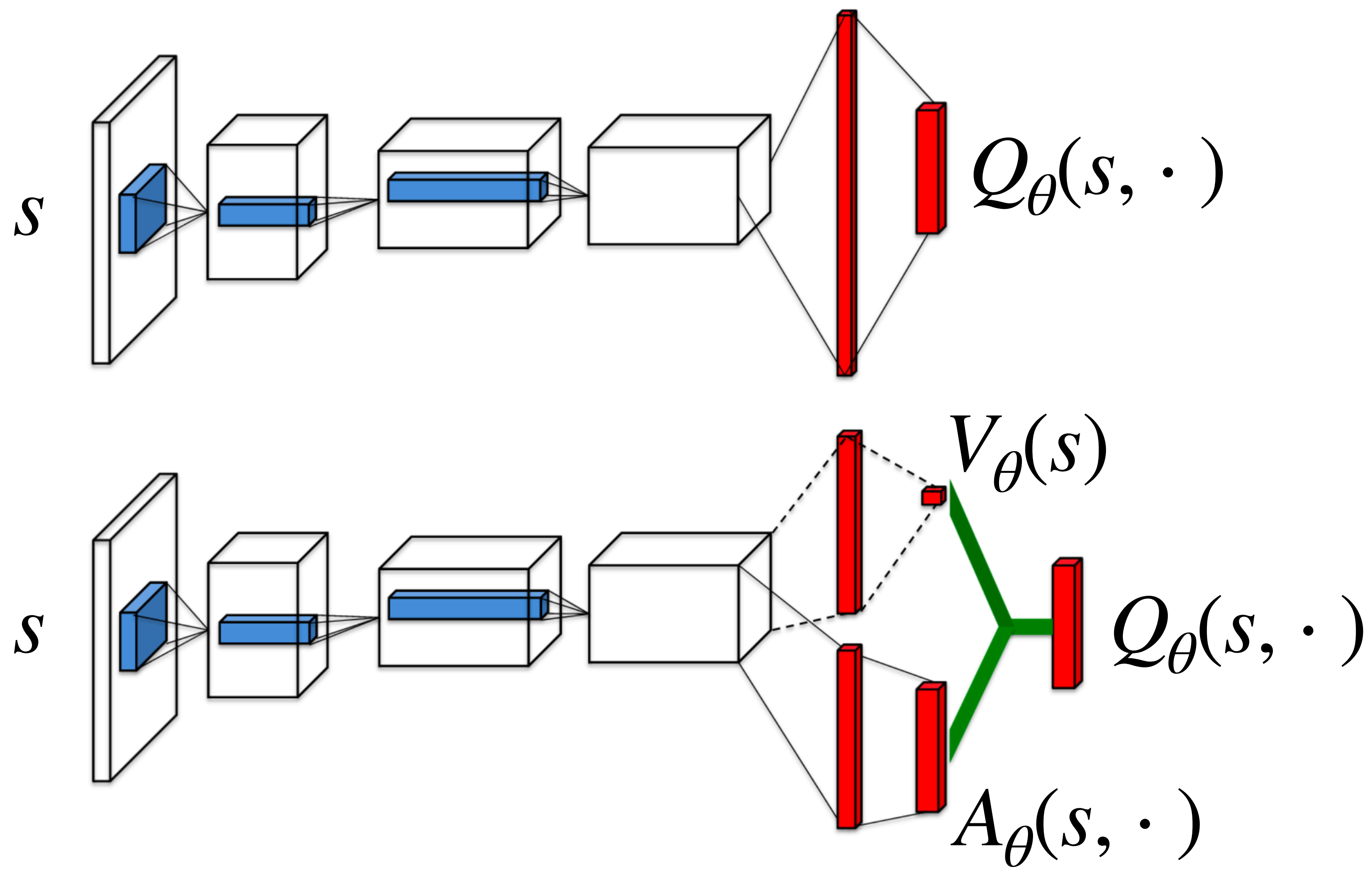
Prioritized Experience Replay

- Bellman error (= TD error): $\delta(s,a,r,s') = r + \gamma \max_{a'} Q(s',a') - Q(s,a)$
  - Optimality: $\delta \equiv 0$; that's why we usually descend the square loss δ^2
- Experience with high error ⇒ more important to see
- Prioritized Experience Replay:
  - Sample instance i with prob. $p_i \propto \delta_i^\omega$; e.g. $\omega = 0.6$
  - Update with Importance Sampling (IS) weight $(m \cdot p_i)^{-\beta}$; e.g. $\beta = 0.4$
- δ is computed during the updates; new experience is weighted $\max_i \delta_i^\omega$

[Schaul et al., 2016]

Dueling Networks

- Advantage function: $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$
- $A_\pi(s,a)$ can be more consistent across states with similar effect of actions
    - Even if their value $V_\pi(s)$ is very different
- $V_\pi(s)$ is a scalar, which can be easier to learn



- Issue: $Q = (V + c) + (A - c)$ is underdetermined
    - Stabilize with $Q(s,a) = V(s) + \left(A(s,a) - \operatorname{mean}_{\bar{a}} A(s,\bar{a})\right)$

[Wang et al., 2016]

Multi-step Q Learning

- MC is high variance but unbiased: $Q(s_t, a_t) \to R_{\geq t} = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}$

- TD is lower variance but biased: $Q(s_t, a_t) \to r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$
  - Because $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ isn't really the next-step value, while still learning

- Let's trade them off, n-step Q-Learning:

$$Q(s_t, a_t) \to r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_{a_{t+n}} Q(s_{t+n}, a_{t+n})$$

Rainbow DQN

- Rainbow DQN = DQN + a powerful combination of tricks
  - Double Q-Learning
  - Prioritized Experience Replay
  - Dueling Networks
  - Multi-step Q-Learning
  - Distributional RL
  - Noisy Nets

[Hessel et al., 2018]

Recap

- RL algorithms can be implemented with function approximation
- There are (at least) 2 important policies
  - The learner policy — should be the best possible (e.g. greedy)
  - The experience policy — should explore (e.g. ϵ-greedy)
- Replay buffer: store data for longer (off-policy), diversify
- Target network: reduce variance, stabilize the target
- In practice, add lots of tricks and heuristics to the theory

Logistics

assignments

- Exercise 1 due tomorrow
- Quiz 2 due next Monday