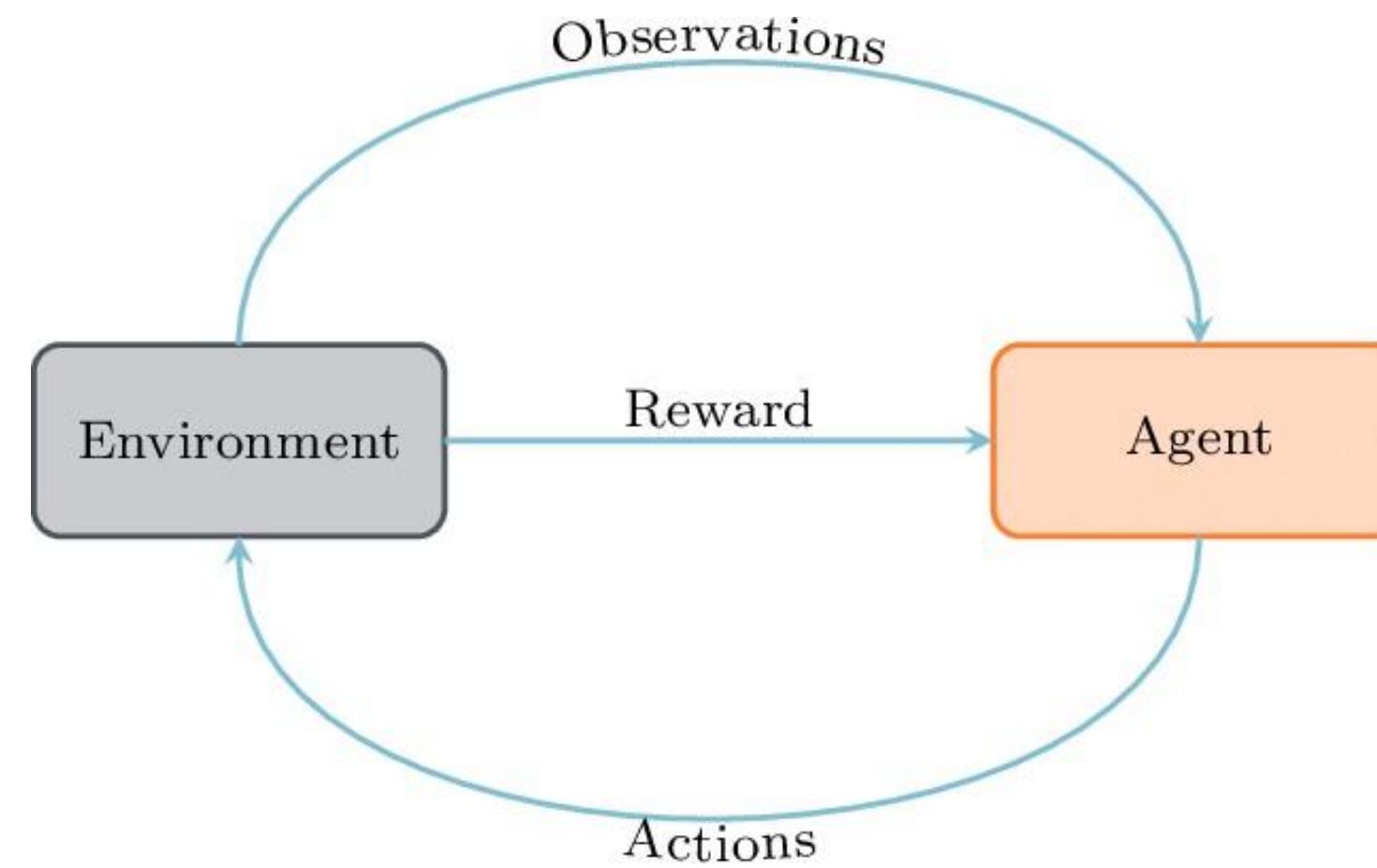


Deep Learning

Lecture 9.1

from really good course in AI masters (<https://ozonmasters.ru/reinforcementlearning>).

Reinforcement Learning: previous lecture



Recap: Value Functions

**Total Discounted
Return:**

$$R_t = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \dots = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r(s_{t+\tau}, a_{t+\tau})$$

**State Value Function /
V-Function:**

$$V_{\pi}(s) = \mathbb{E}_{\pi}[R_t | s_t = s] = \mathbb{E}_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} | s_t = s\right]$$

**State-Action Value Function /
Q-Function:**

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] = \mathbb{E}_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} | s_t = s, a_t = a\right]$$

Recap: Bellman Equations

Bellman Expectation Equations for policy π :

$$V_{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)[r + \gamma V_{\pi}(s')]$$

$$Q_{\pi}(s, a) = \sum_{s'} p(s'|s, a)[r + \gamma \sum_{a'} \pi(a'|s') Q_{\pi}(s', a')]$$

Bellman Optimality Equations:

$$V_*(s) = \max_a \sum_{s'} p(s'|s, a)[r + \gamma V_*(s')] = \max_a Q_*(s, a)$$

$$Q_*(s, a) = \sum_{s'} p(s'|s, a) [r + \gamma \max_{a'} Q_*(s', a')]$$

Recap: Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

Use Belman **Expectation** Equations to
learn V/Q for current policy

3. Policy Improvement

$policy\text{-stable} \leftarrow true$

For each $s \in \mathcal{S}$:

$$old\text{-action} \leftarrow \pi(s)$$

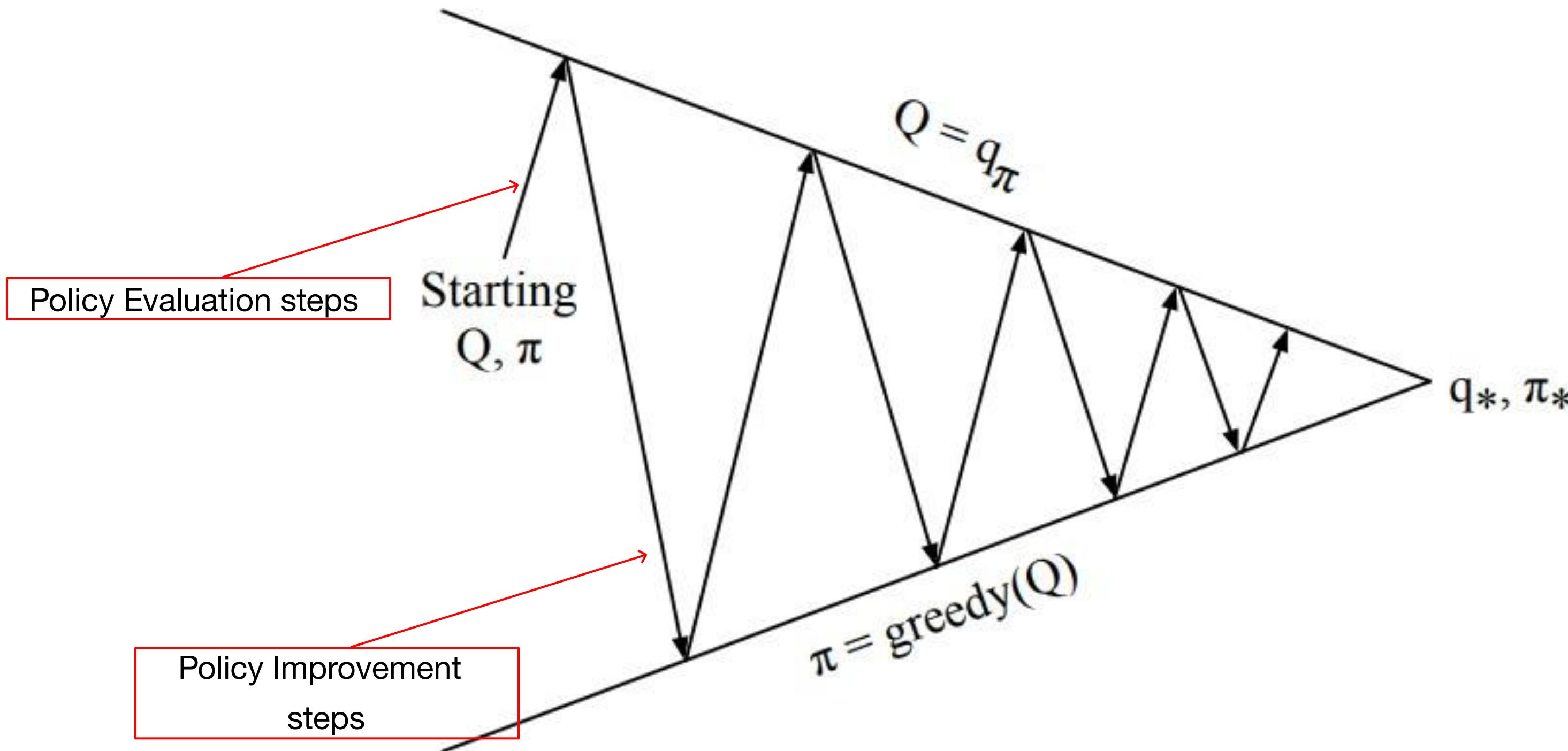
$$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

If $old\text{-action} \neq \pi(s)$, then $policy\text{-stable} \leftarrow false$

If $policy\text{-stable}$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Greedily Update policy w.r.t. V/Q-function

Recap: Policy Iteration



Recap: Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$

Use Bellman **Optimality** Equations to
learn V/Q for current policy

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

Policy Improvement is
implicitly used here

But we don't know the model

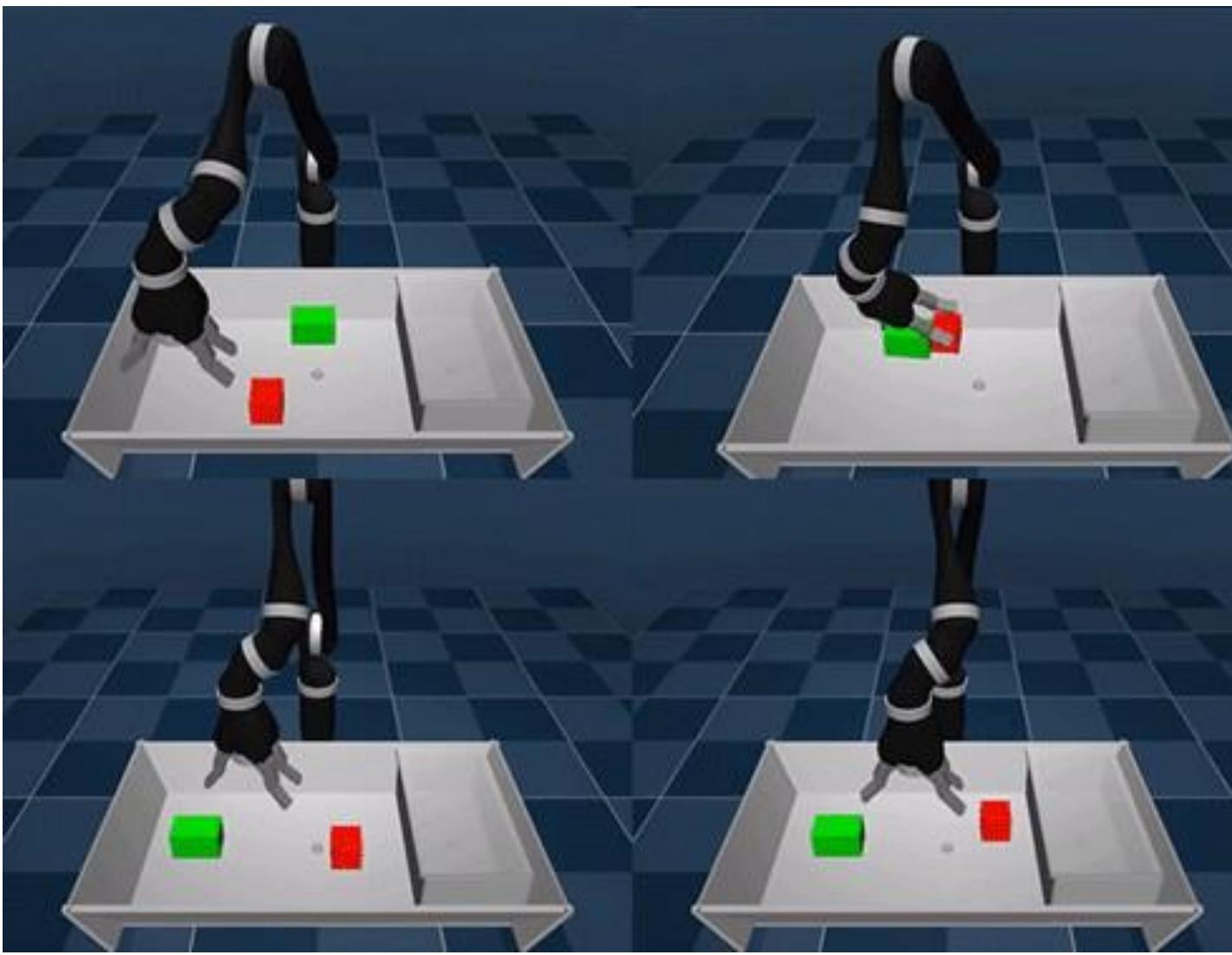


$$V_{\pi}(s) = \sum_a \pi(a | s) \sum_{s'} p(s' | s, a) [r + \gamma V_{\pi}(s')]$$

$$Q_{\pi}(s, a) = \sum_{s'} p(s' | s, a) [r + \gamma \sum_{a'} \pi(a' | s') Q_{\pi}(s', a')]$$

Solution: Use Sampling

- The model of the environment is unknown!
- :(But you still can interact with it!



Monte-Carlo Methods

GOAL: Learn value functions Q_π or V_π without knowing $p(s' | s, a)$ and $R(s, a)$

RECALL that value function is the expected return:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s, A_t = a \right]$$

First Visit Monte-Carlo

IDEA: Estimate expectation $Q_\pi(s, a)$ with empirical mean $q(s, a)$:

- Generate an episode with π
The first time-step t that state-action (s, a) is visited
- Increment counter $N(s, a) = N(s, a) + 1$
- Increment total return $S(s, a) = S(s, a) + R_t$
- Value is estimated by mean return $q(s, a) = S(s, a)/N(s, a)$

By Law of Large Numbers, $q(s, a) \rightarrow Q_\pi(s, a)$ as $N(s, a) \rightarrow \infty$

Monte-Carlo Methods: Prediction

We can update mean values incrementally:

$$\mu_{k+1} = \frac{1}{k+1} \sum_{k=1}^{k+1} x_k = \mu_k + \frac{1}{k+1} (x_k - \mu_k)$$

Old estimate Learning rate Prediction error

Incremental Monte Carlo Update:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \frac{1}{N(s_t, a_t)} (R_t - q(s_t, a_t))$$

In non-stationary problems we can fix learning rate:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha (R_t - q(s_t, a_t))$$

Monte-Carlo Methods

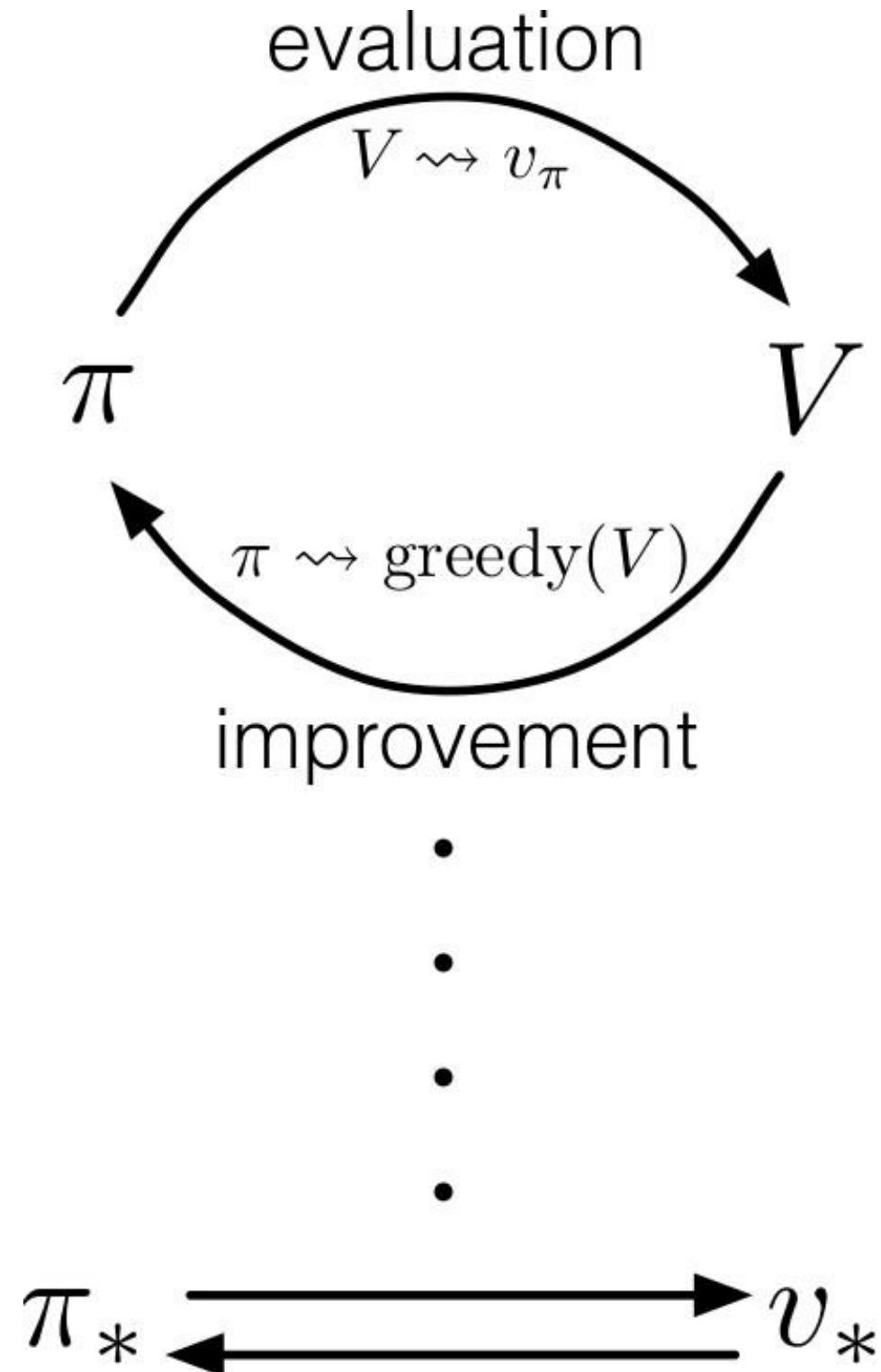
- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs:
 - All episodes must terminate

Monte-Carlo Methods: Control

Remember **Policy Iteration**?

How would look PI version with Monte-Carlo Policy Evaluation?

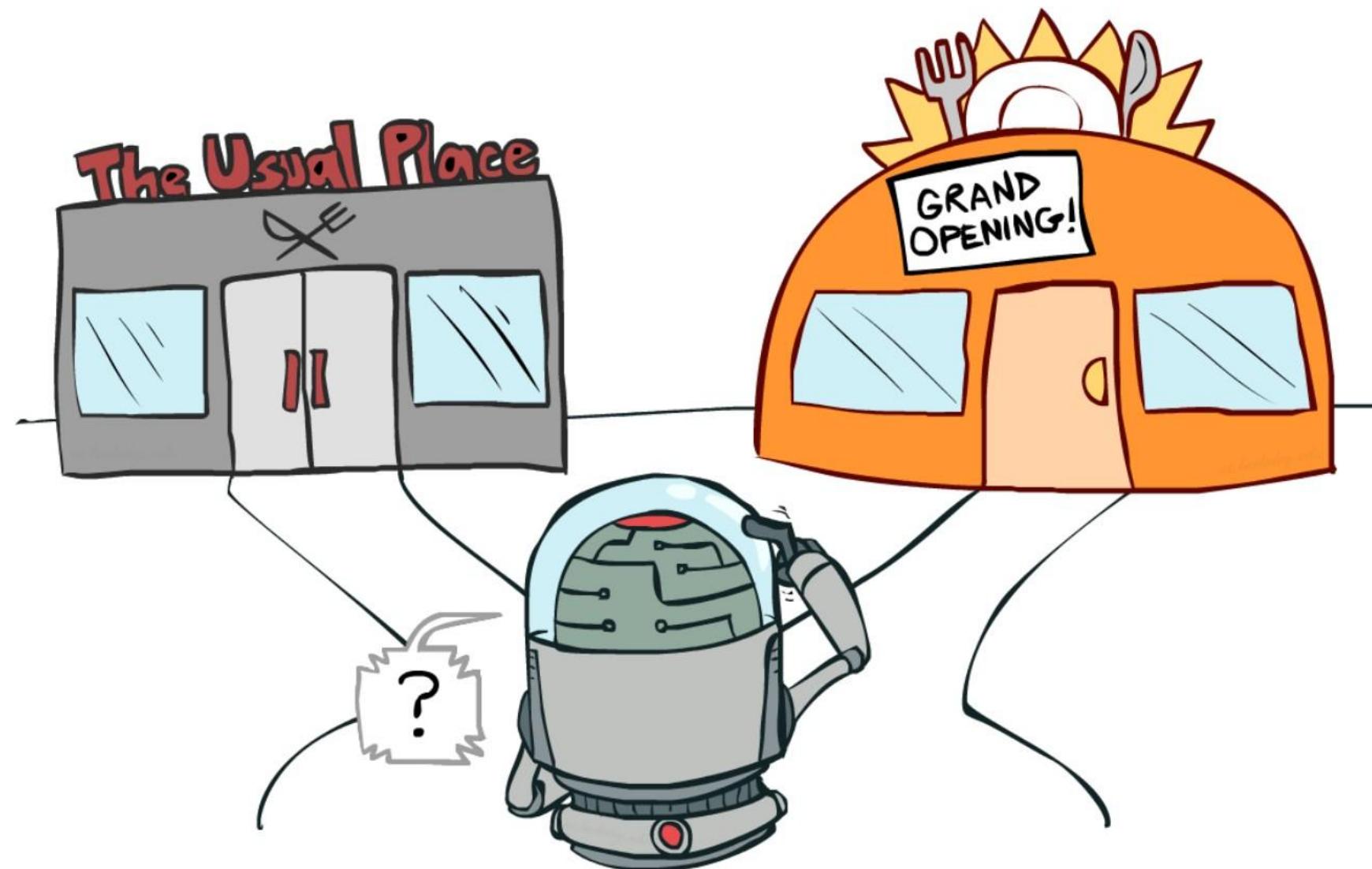
- **Policy Evaluation:** Monte-Carlo Evaluation, $q = Q_\pi$
- **Policy Improvement:** Greedy policy improvement,
i.e. $\pi'(s) = \text{argmax}_a q(s, a)$



Questions:

- Why we estimate Q_π and not V_π ?
- Do you see any problem with policy improvement step?

Exploration-Exploitation Problem



Agent can't visit every (s, a) with **greedy policy**!

Agent can't get correct $q(s, a)$ estimates without visiting (s, a) frequently!

(i.e. remember **law of large numbers**)

Monte-Carlo Methods: Control

Use ϵ -greedy policy:

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1-\epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Monte-Carlo Methods: Control

That any ε -greedy policy with respect to q_π is an improvement over any ε -soft policy π is assured by the policy improvement theorem. Let π' be the ε -greedy policy. The conditions of the policy improvement theorem apply because for any $s \in \mathcal{S}$:

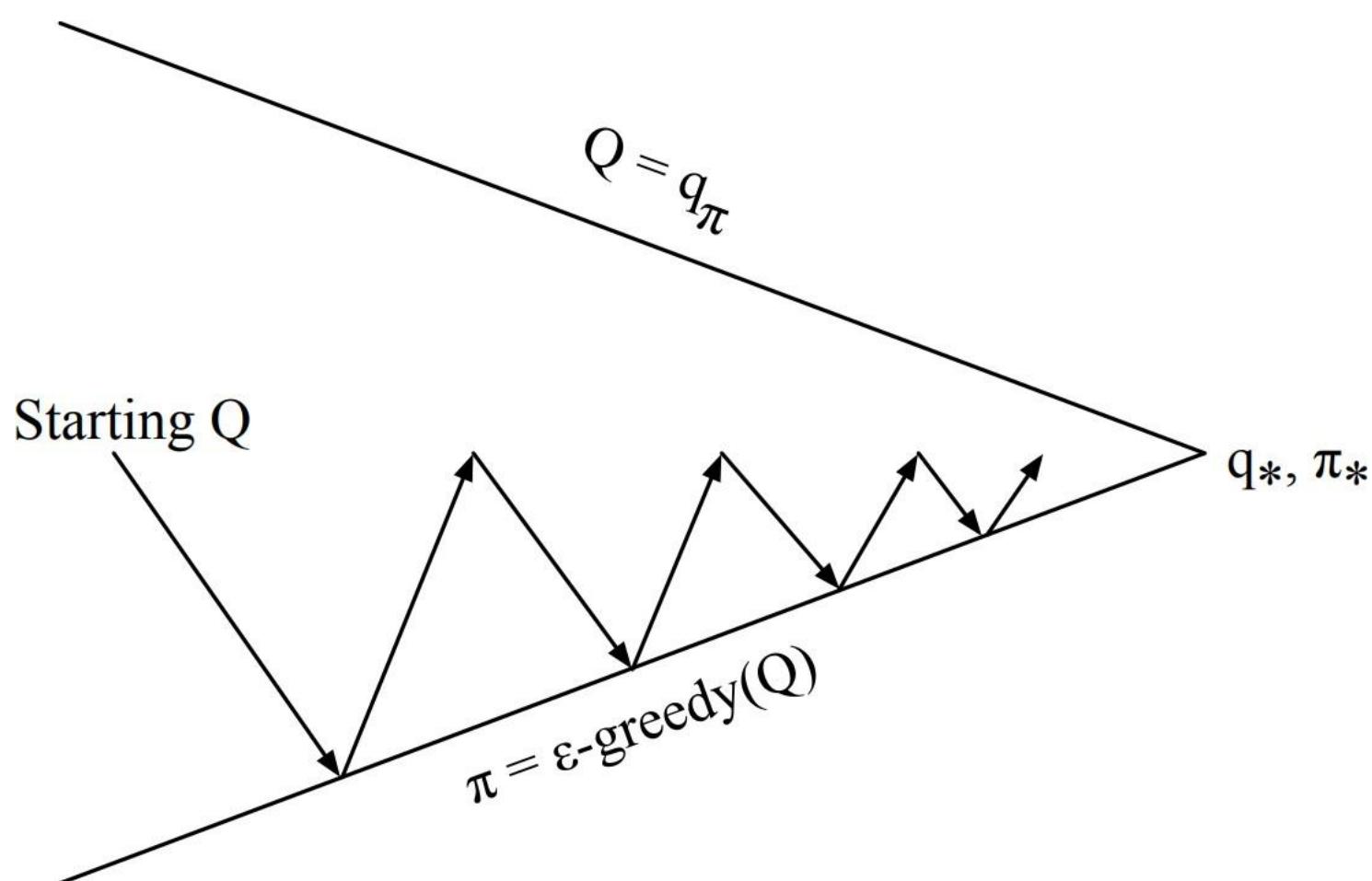
$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \max_a q_\pi(s, a) \tag{5.2} \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon} q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) \\ &= v_\pi(s). \end{aligned}$$

Monte-Carlo Method

Policy Iteration with Monte-Carlo method:

For every episode:

- **Policy Evaluation**: Monte-Carlo Evaluation, $q = Q_\pi$
- **Policy Improvement**: ϵ -greedy policy improvement



Temporal Difference Learning

Problems with Monte-Carlo method:

- Updates policy only once per episode, i.e. only episodic MDPs
- Doesn't use MDP properties

Solution:

- Recall Bellman equation:

$$Q_{\pi}(s, a) = \sum_{s'} p(s' | s, a) [r + \gamma \sum_{a'} \pi(a' | s') Q_{\pi}(s', a')]$$

- Use sampling instead of knowledge about the model:

$$q_{\pi}(s, a) = \mathbb{E}_{s'} [r + \gamma \mathbb{E}_{a'} q_{\pi}(s', a')] = \mathbb{E}_{\tau} [r + \gamma q_{\pi}(s', a')]$$

TD-learning: Prediction

Goal: learn \mathcal{Q}_π online from experience

Incremental Monte-Carlo:

- Update value $q(s_t, a_t)$ toward actual return R_t

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(R_t - q(s_t, a_t))$$

Temporal-Difference learning:

- Update value $q(s_t, a_t)$ toward estimated return $r_{t+1} + \gamma q(s_{t+1}, a_{t+1})$

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t))$$

$r_{t+1} + \gamma q(s_{t+1}, a_{t+1})$ is called the TD target

$\delta_t = r_{t+1} + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t)$ is called the TD error

Temporal Difference Learning

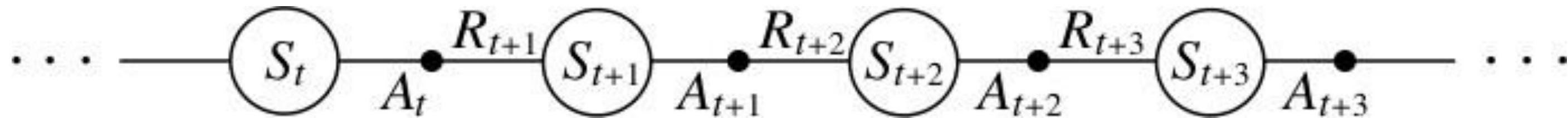
Temporal Difference Learning:

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping
- TD updates a guess towards a guess

TD-learning: SARSA update

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t))$$

This update is called **SARSA**: **S**tate, **A**ction, **RS**tate, next **A**ction

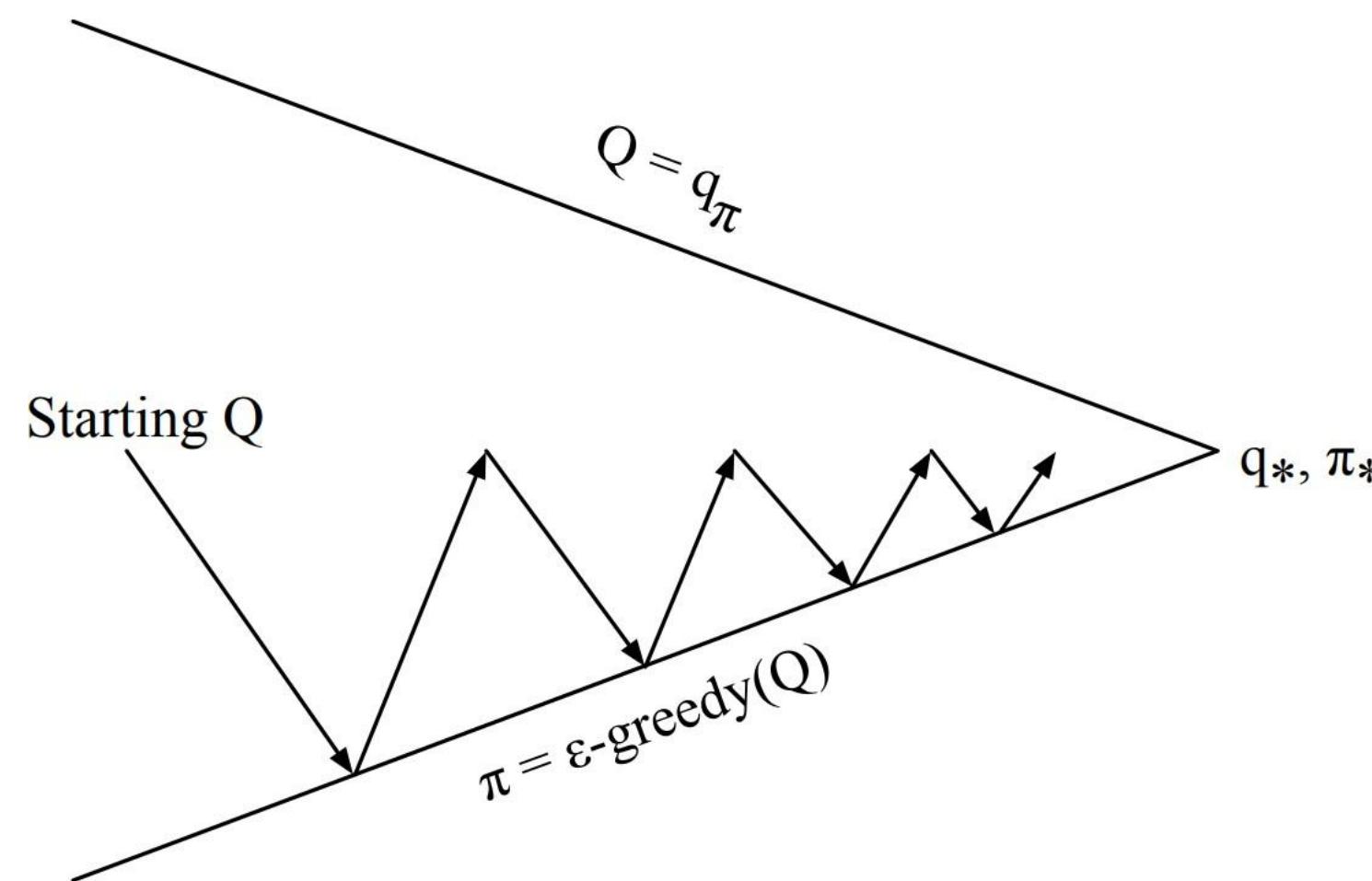


TD-learning: SARSA as Policy Iteration

Policy Iteration with Temporal Difference Learning:

For **every step**:

- **Policy Evaluation:** SARSA Evaluation, $q = Q_\pi$
- **Policy Improvement:** ϵ -greedy policy improvement



TD-learning: SARSA algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

TD-learning: Q-Learning

We approximate Bellman **Expectation** Equation with SARSA update:

$$Q_{\pi}(s, a) = \sum_{s'} p(s' | s, a)[r + \gamma \sum_{a'} \pi(a' | s') Q_{\pi}(s', a')]$$

Can we utilize Bellman **Optimality** Equation for TD-Learning?

$$Q_*(s, a) = \sum_{s'} p(s' | s, a)[r + \gamma \max_{a'} Q_*(s', a')]$$

TD-learning: Q-Learning

We approximate Bellman **Expectation** Equation with SARSA update:

$$Q_\pi(s, a) = \sum_{s'} p(s' | s, a)[r + \gamma \sum_{a'} \pi(a' | s') Q_\pi(s', a')]$$

Can we utilize Bellman **Optimality** Equation for TD-Learning?

$$Q_*(s, a) = \sum_{s'} p(s' | s, a)[r + \gamma \max_{a'} Q_*(s', a')]$$

Yes, of course:

$$q(s, a) = \mathbf{E}_{s'} [r + \gamma \max_{a'} q(s', a')]$$

TD-learning: Q-Learning vs SARSA

From Bellman Expectation Equation (SARSA) :

$$q(s, a) = \mathbb{E}_{s'} [r + \gamma \mathbb{E}_{a'} q(s', a')]$$

a' comes from the policy π
that generated this
experience!

From Bellman Optimality Equation (Q-Learning):

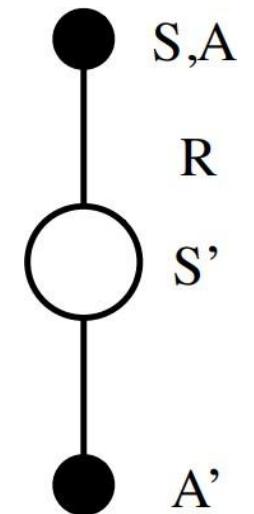
$$q(s, a) = \mathbb{E}_{s'} [r + \gamma \max_{a'} q(s', a')]$$

No connection to the actual
policy π

TD-learning: Q-Learning vs SARSA

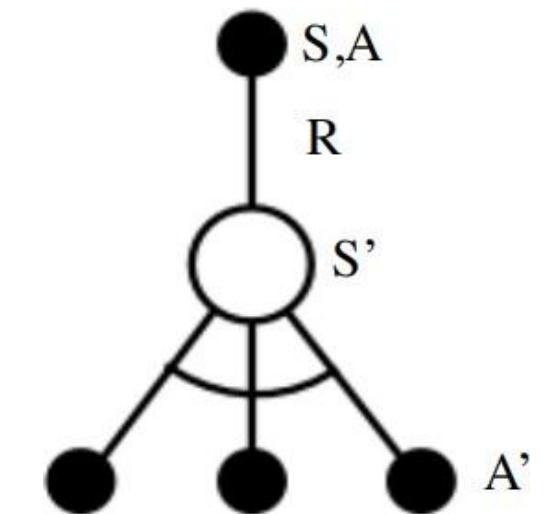
SARSA Update:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t))$$



Q-Learning Update:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} q(s_{t+1}, a') - q(s_t, a_t))$$



TD-learning: Q-Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

 until S is terminal

Theorem

Q-learning control converges to the optimal action-value function,

$$Q(s, a) \rightarrow q_*(s, a)$$

On-policy vs Off-Policy Algorithms

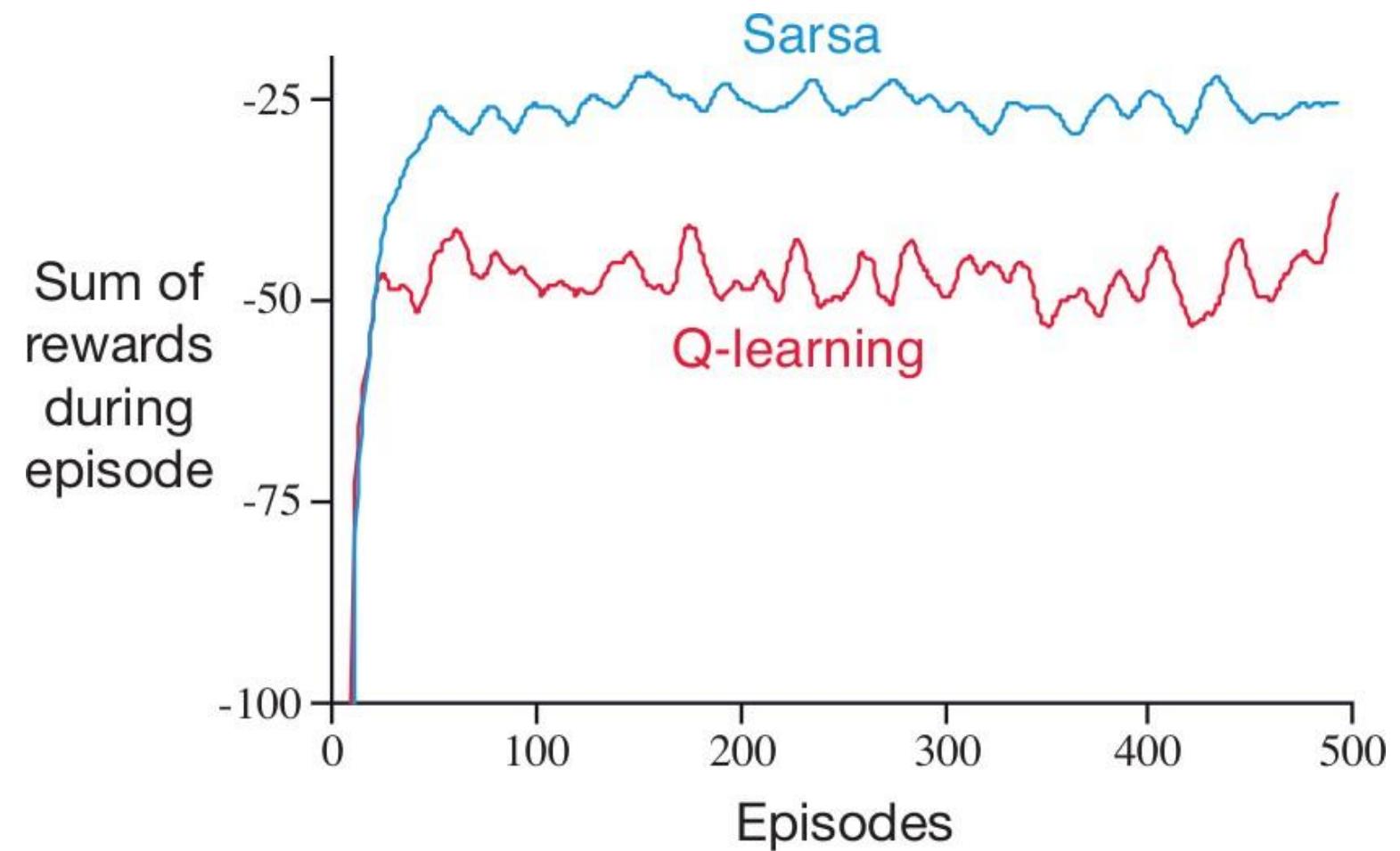
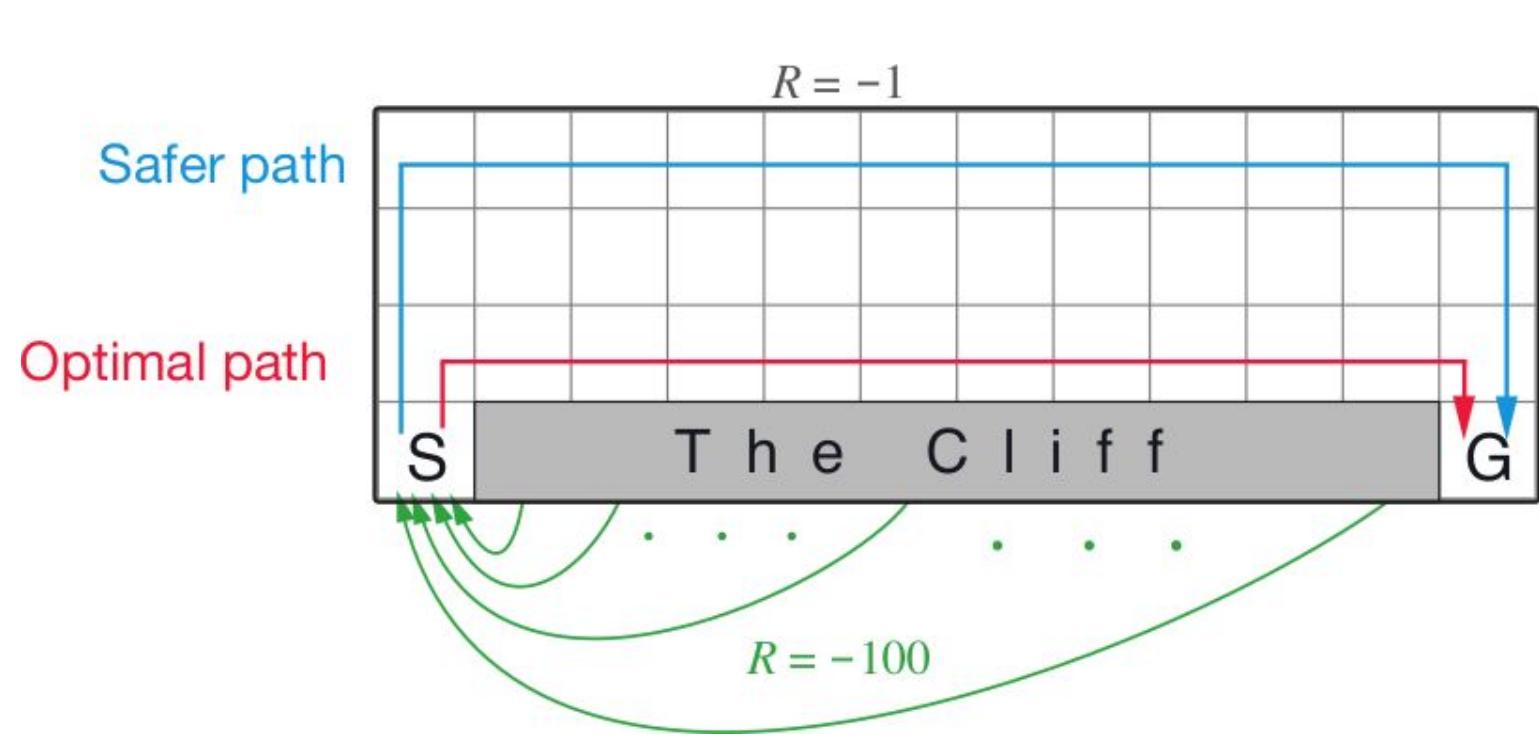
SARSA and Monte-Carlo are **on-policy** algorithms:

- Improve policy π_k only from experience sampled with this policy π_k
- Can't use old trajectories sampled with π_{k-i}

Q-Learning is **off-policy** algorithm:

- Can Learn policy π using experience generated with other policy μ
- Learn from observing humans or other agents
- Re-use experience generated from old policies
- Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following one policy

TD-learning: Cliff Example



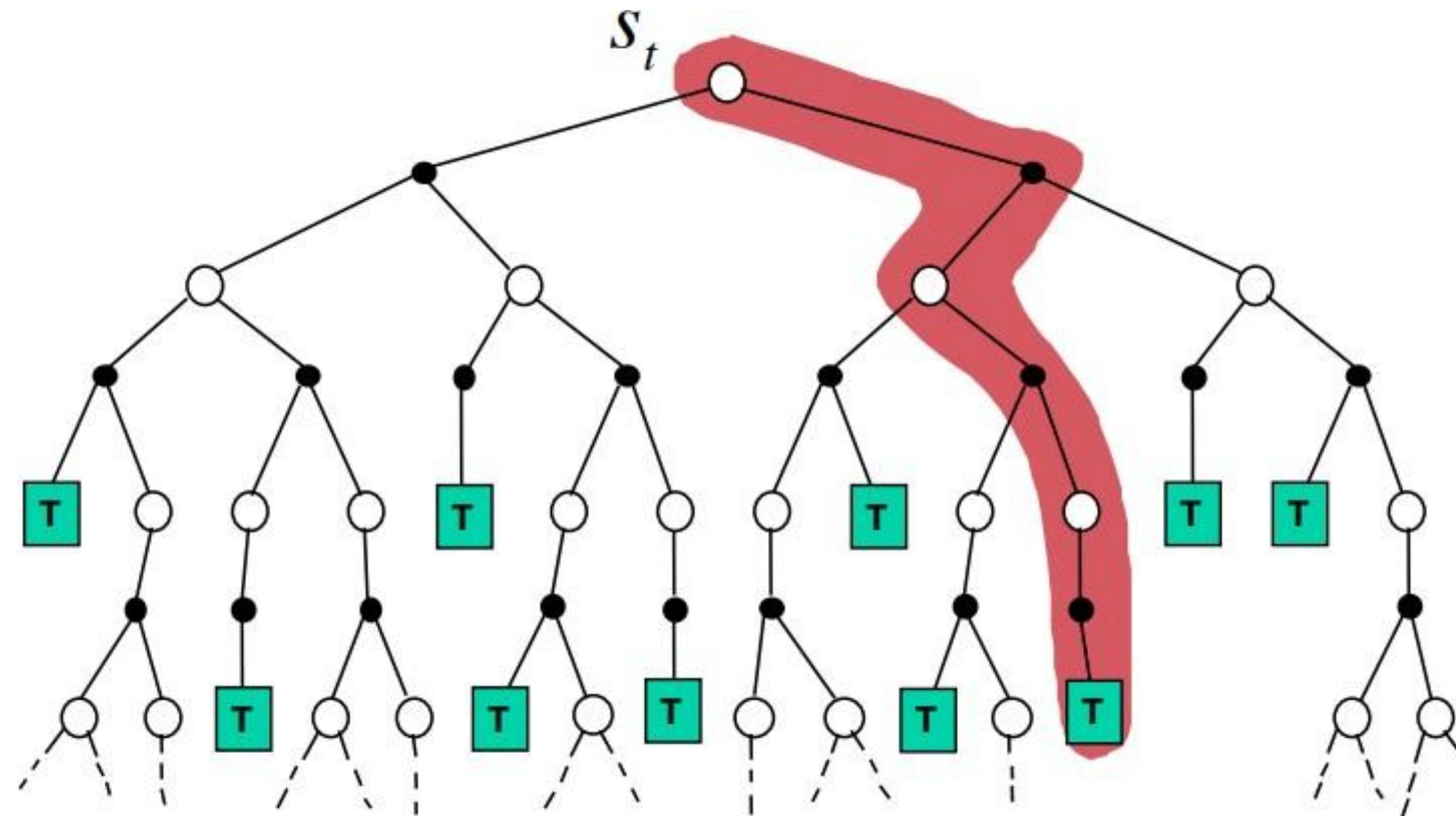
TD vs MC: Bias-Variance Tradeoff

- TD target is much lower variance than the return:
- Return $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$ is unbiased estimate of $Q_\pi(s_t, a_t)$
- True TD target $R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})$ is unbiased estimate of $Q_\pi(s_t, a_t)$
- TD target $R_{t+1} + \gamma q(s_{t+1}, a_{t+1})$ is biased estimate of $Q_\pi(s_t, a_t)$
- TD target is much lower variance than the return:
 - **Return** depends on many random actions, transitions, rewards
 - **TD target** depends on one random action, transition, reward

- Monte-Carlo Methods: **high variance, no bias**
 - TD-Обучение: **low variance, has bias**

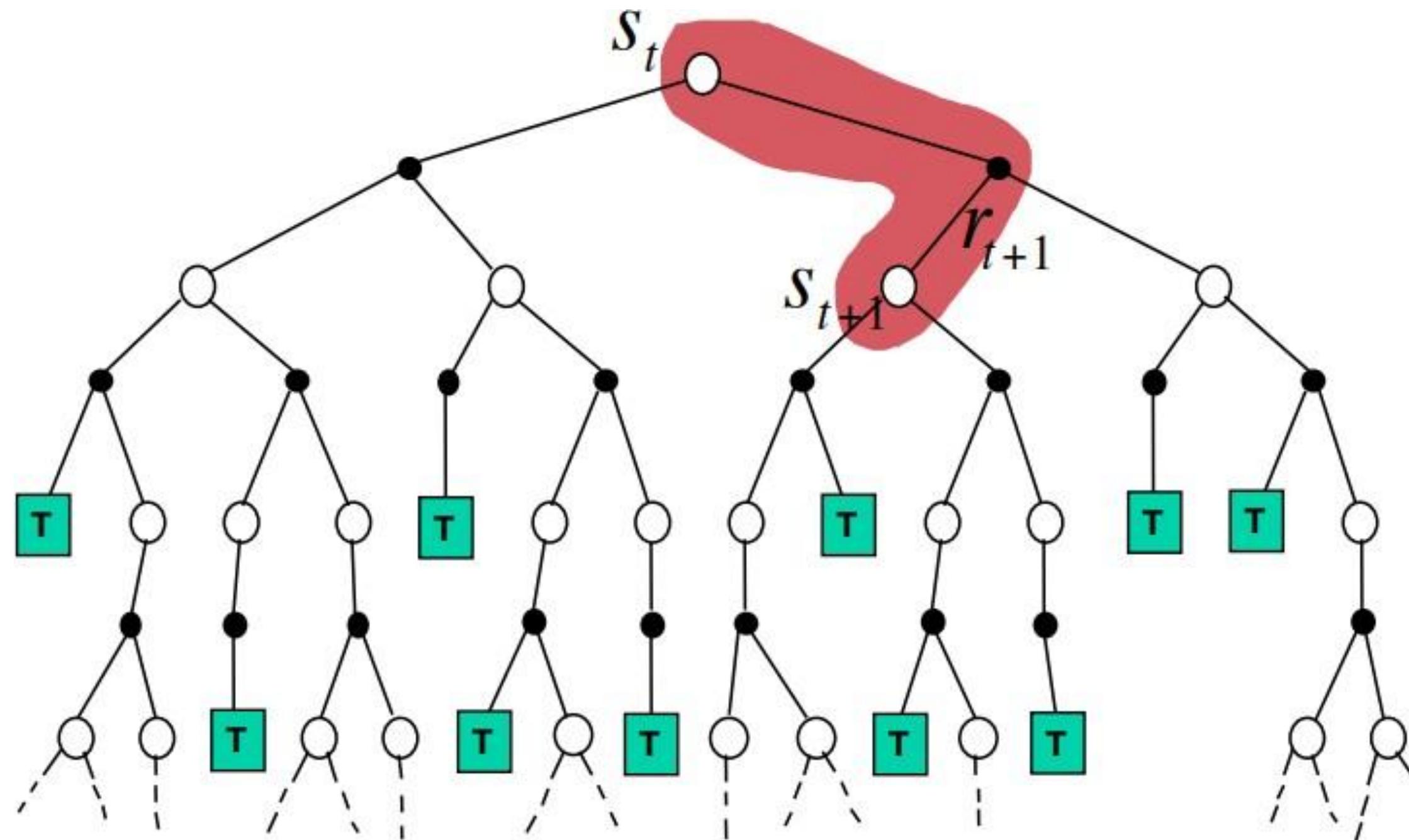
Monte-Carlo Backup

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(G_t - q(s_t, a_t))$$



Temporal-Difference Backup

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t))$$



N-step Returns

Consider the following n-step returns for $n = 1, 2, \dots$:

(TD: SARSA) $n = 1$ $G_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$n = 2$ $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$

$$\begin{array}{c} \vdots \\ \vdots \end{array}$$

(MC) $n = \infty$ $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$

n-step Temporal Difference Learning:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(G_t^{(n)} - q(s_t, a_t))$$

Recap: Q-learning

Bellman optimality equation for Q^* : $\forall s, a:$

$$Q^*(s, a) := r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} \max_{a'} Q^*(s', a')$$

Optimal policy:

$$\pi^*(s) := \operatorname{argmax}_a Q^*(s, a)$$

Q-learning can learn Q^* from trial and error in *finite* MDPs: $|S| \ll \infty, |\mathcal{A}| \ll \infty$

Important!

Q-learning is off-policy

Recap: Q-learning update intuition

For transition (s, a, r, s') :

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha_k \underbrace{\left(\overbrace{r + \gamma \max_{a'} Q_k(s', a')}^{\text{Bellman target}} - Q_k(s, a) \right)}_{\text{temporal difference}}$$

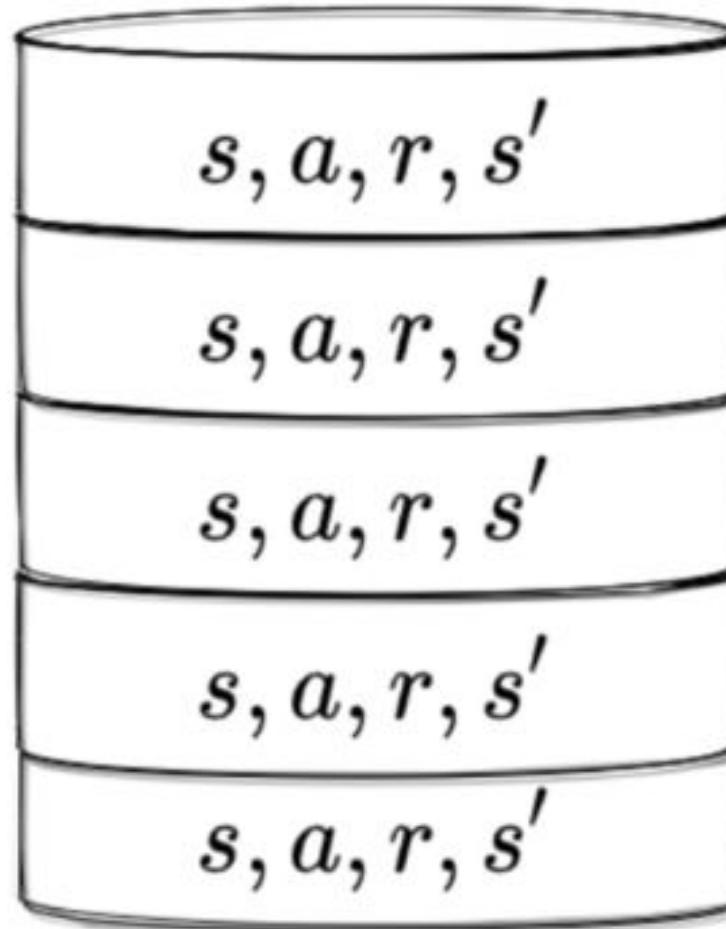
s, a — considered **fixed**, can be chosen arbitrarily;

(!) each pair must be updated infinite times;

$s' \sim p(s' | s, a)$ — **random variable** from interaction experience.

Experience Replay

**Experience
Replay**



On each training step sample **uniformly** random transition from experience replay and perform update using it.

- ✓ allows reuse of experience;
- ✓ allows training on expert data;
- ✗ sample s' comes from empirical approximation:

$$s' \sim \hat{p}(s' | s, a) \approx p(s' | s, a)$$

- ✓ smoothes out with size of experience replay!

Q-learning with experience replay

Q-learning (online)

Initialize $Q(s, a)$ arbitrarily;

observe s_0 ;

for $k = 0, 1, 2 \dots$

- take action $a_k \sim \varepsilon\text{-greedy}(Q(s_k, a))$;
- observe r_k, s_{k+1} ;
- $y := r_k + \gamma \max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k)Q(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q(s, a)$ arbitrarily, $\mathcal{D} = \emptyset$;

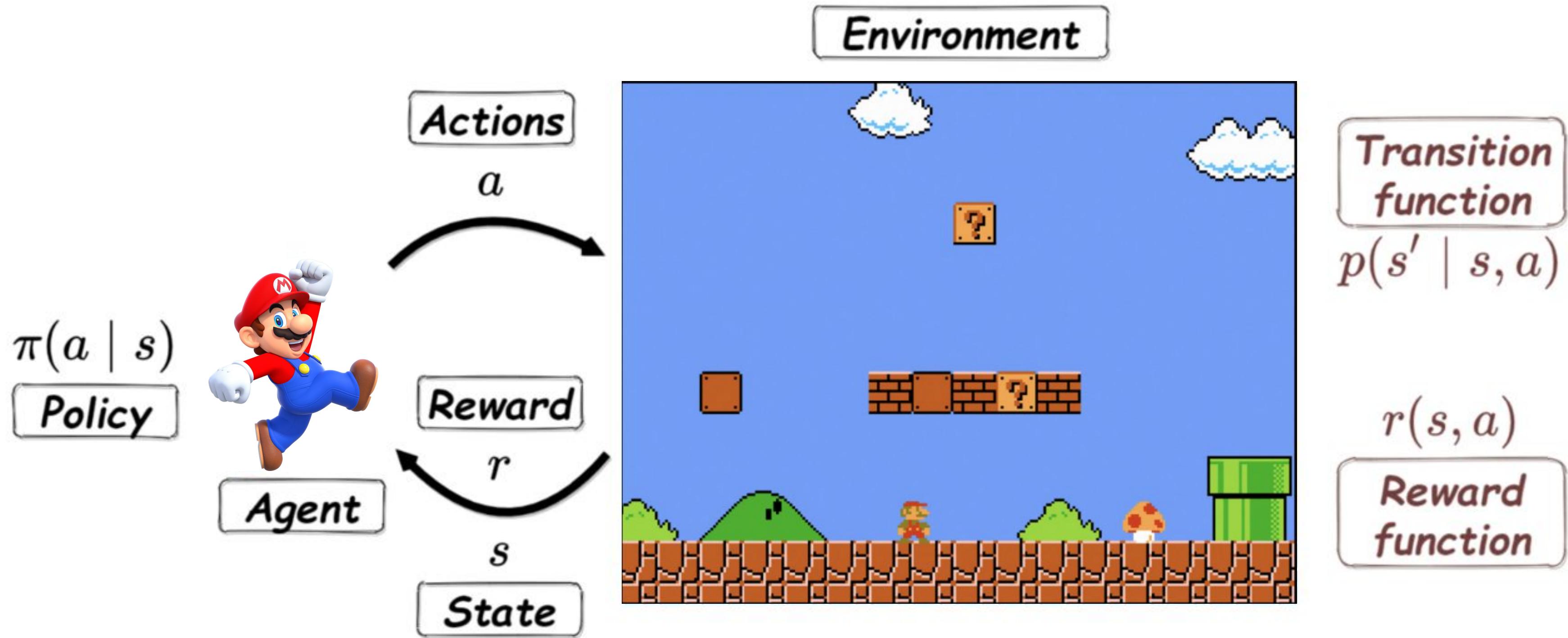
observe s_0 ;

for $k = 0, 1, 2 \dots$

- take action $a_k \sim \varepsilon\text{-greedy}(Q(s_k, a))$;
- observe r_k, s_{k+1} ;
- store (s_k, a_k, r_k, s_{k+1}) in \mathcal{D} ;
- sample (s, a, r, s') from \mathcal{D} ;
- $y := r + \gamma \max_{a'} Q(s', a')$;
- $Q(s, a) \leftarrow (1 - \alpha_k)Q(s, a) + \alpha_k y$

Deep RL

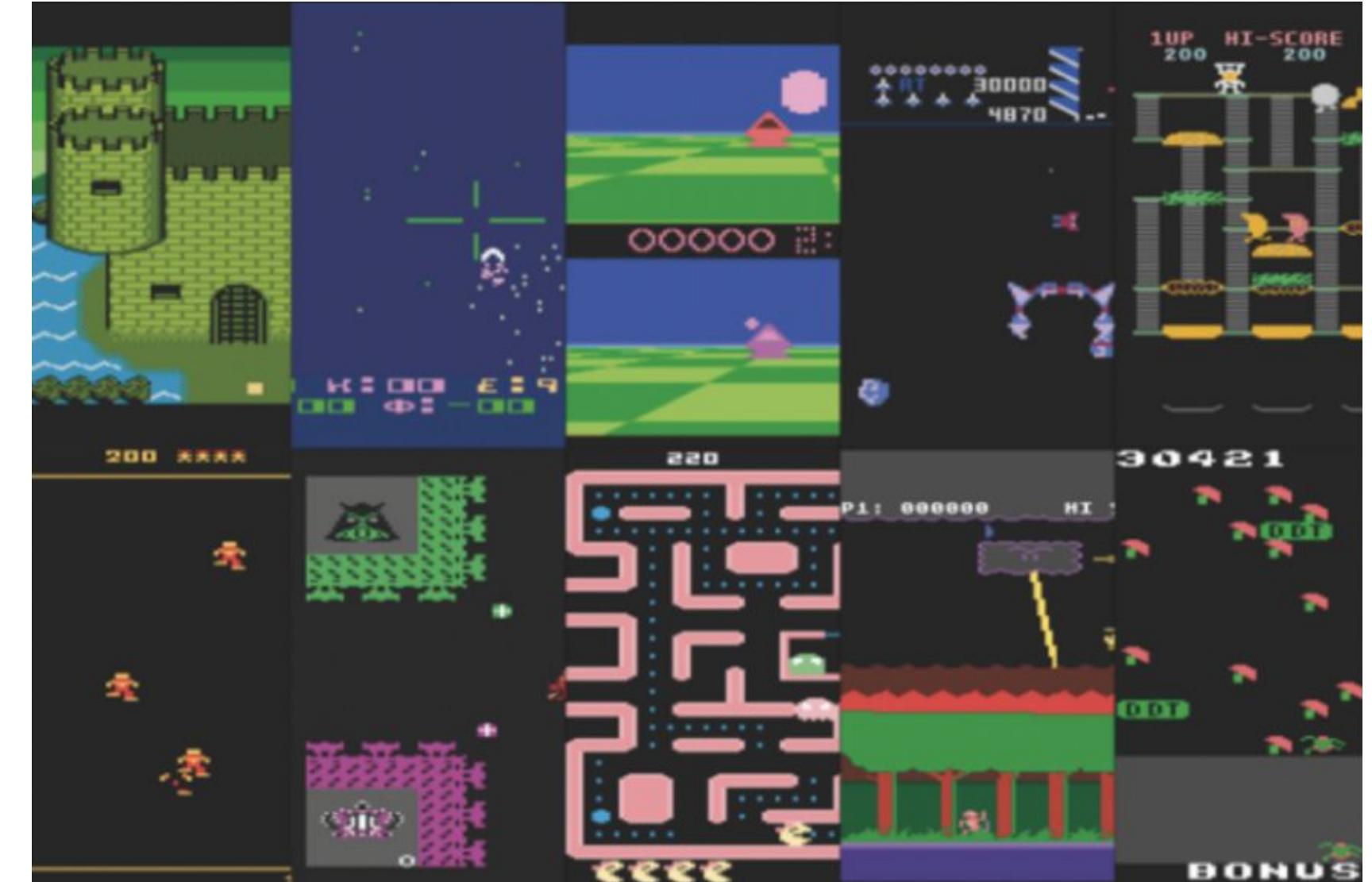
Benchmarks: Video Games



Atari Games

Setup: ~~$T \leftarrow \infty$~~ , $|A| \ll \infty$

- 57 various games
- Only screen image as input.
- No game-specific features.
- Finite-state case... not quite finite.
- $|A| \leq 18$



Approximate $Q^*(s, a)$ with neural network!

Is Atari game a MDP?

Practical notes: preprocessing

Action selection frequency:

- Framestack;
- Frameskip;
- MaxAndSkip;
- (sometimes) Sticky actions;

Atari-specific preprocessing:

- EpisodicLife;
- FireReset;

Standard tricks:

- Crop image;
- Rescale (often to 84x84);
- Grayscale;

Reward preprocessing:

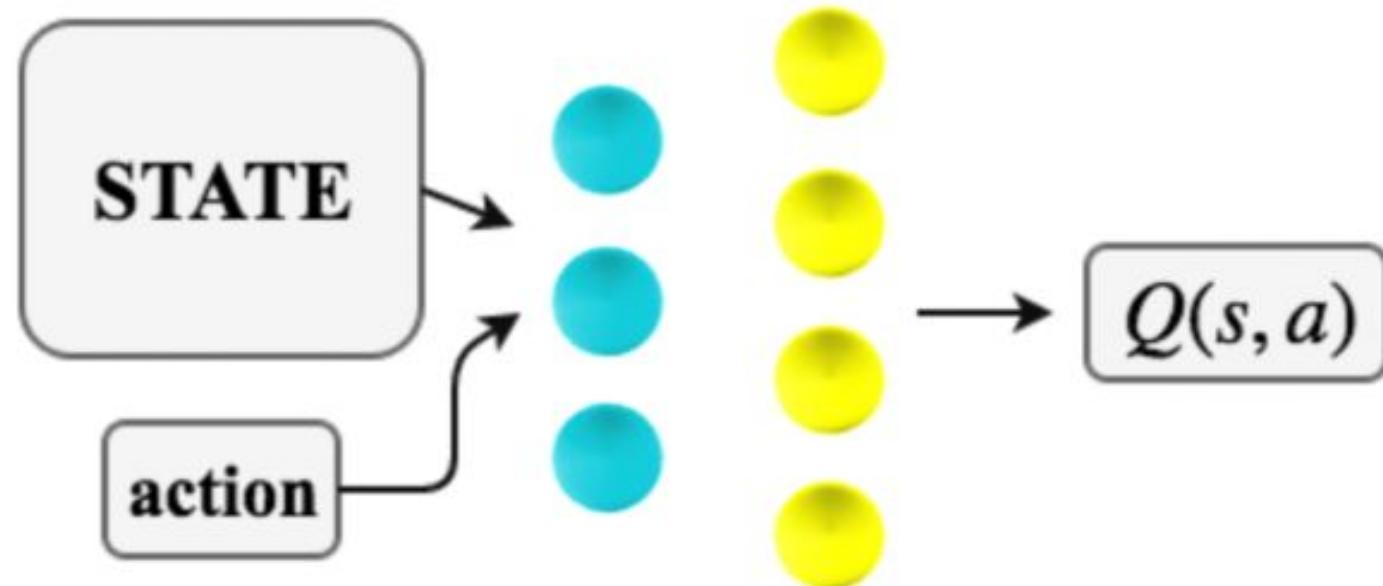
- (!) Clip reward to $\{-1, 0, 1\}$;

Important!

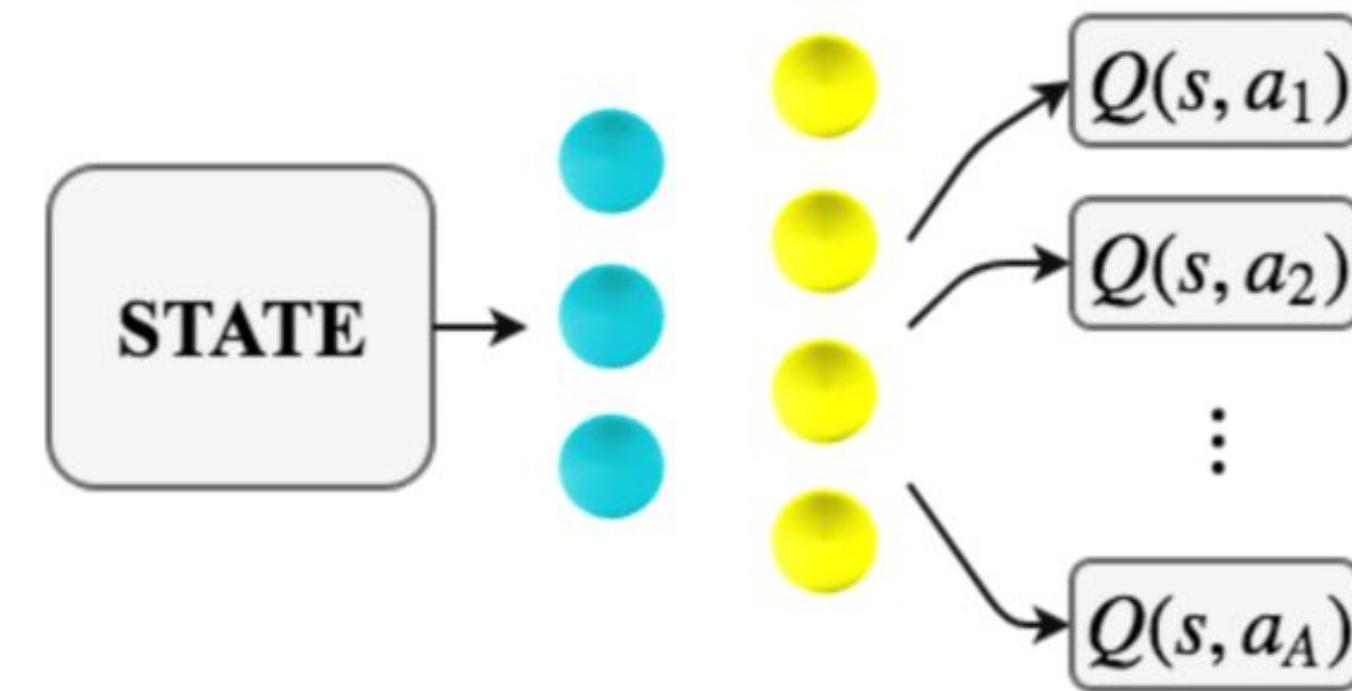
No hyperparameter tuning for each specific game!

Deep Q-network

Option 1



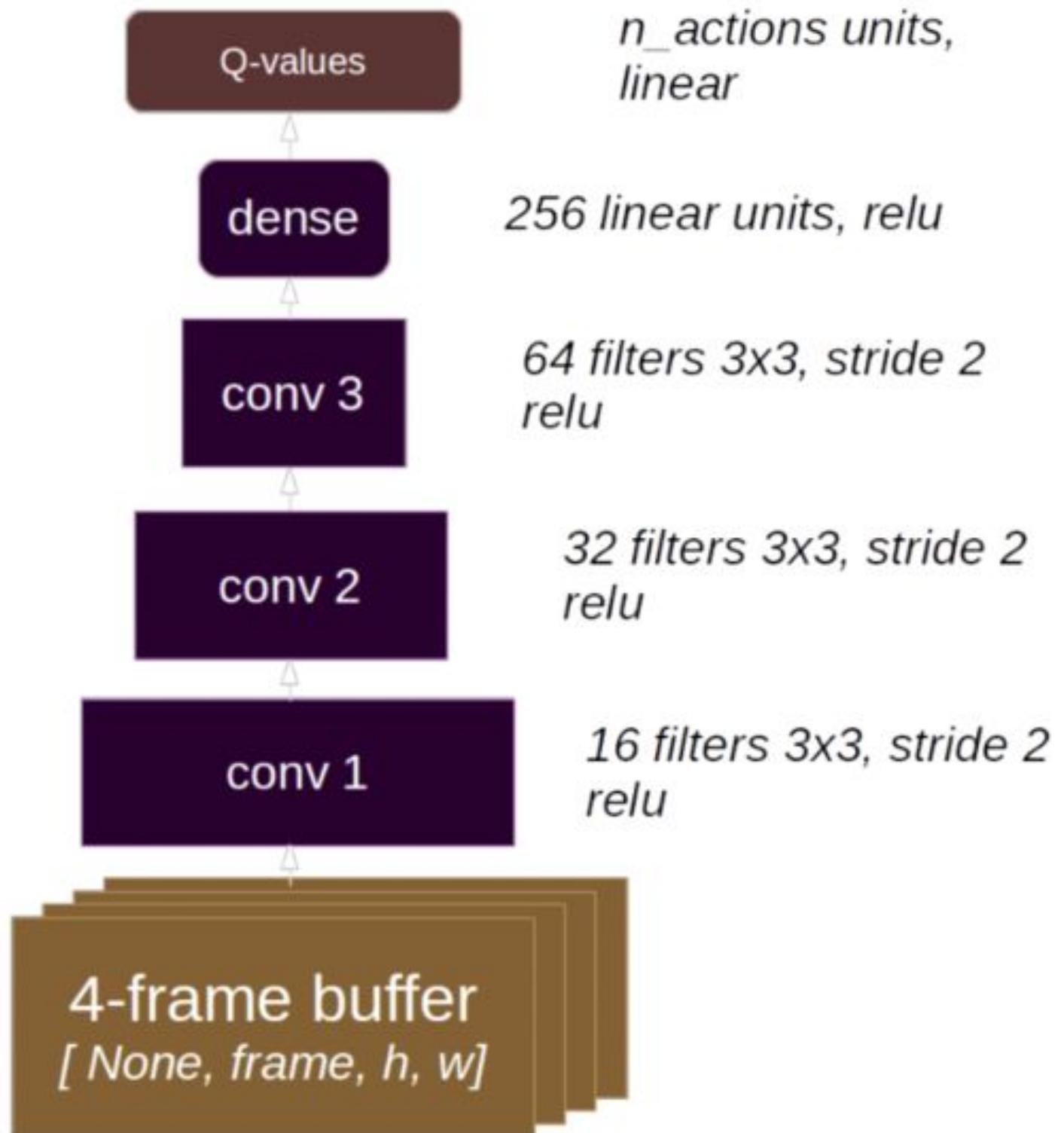
Option 2



✗ $\underset{a}{\operatorname{argmax}} Q(s, a, \theta)$ — expensive!

✓ $\underset{a}{\operatorname{argmax}} Q(s, a, \theta)$ — one forward pass.

Deep Q-network: architecture



- 3-4 convolutional layers followed by 1-2 dense layers;
- stride > 1 for size reduction; linear layers still wide;

Think twice before using:

- ✖ max pooling
- ✖ batch normalization
- ✖ dropout

How to train your network?

Given:

- dataset (x, y) , where x — input, y — output;
- loss function $\text{Loss}(y, \hat{y})$;
- parametric family $\hat{y}(x, \theta)$;

in **supervised learning** we optimized

$$\sum_{(x,y)} \text{Loss}(y, \hat{y}(x, \theta)) \rightarrow \min_{\theta}$$

using techniques based on stochastic gradient descent.

In **reinforcement learning**:

- we have to collect all data ourself
 - ▶ some exploration-exploitation technique is required, e. g. ε -greedy;
 - ▶ the data is not i.i.d.
- what is the **target** y ?
- what is the loss function?

Important

Q-learning is SGD.

Idea: Approximate Dynamic Programming

Consider the following **regression task**:

- s, a is input;
- $y \in \mathbb{R}$ is target:

$$y(s, a) := r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q(s', a', \theta_k)$$

- MSE loss function: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$



$$\frac{1}{2} \mathbb{E}_{(s, a, y)} (y - Q(s, a, \theta_{k+1}))^2 \rightarrow \min_{\theta_{k+1}}$$

Looking at the gradient

$$\begin{aligned} \nabla_{\theta} \frac{1}{2} (y - Q(s, a, \theta))^2 &= \underbrace{(y - Q(s, a, \theta))}_{\text{temporal difference}} \overbrace{\nabla_{\theta} Q(s, a, \theta)}^{\text{how to increase output value}} = \\ &= \left(r + \gamma \mathbb{E}_{s'} \max_{a'} Q(s', a', \theta_k) - Q(s, a, \theta) \right) \nabla_{\theta} Q(s, a, \theta) \approx \\ &\approx \left(r + \gamma \max_{a'} Q(s', a', \theta_k) - Q(s, a, \theta) \right) \nabla_{\theta} Q(s, a, \theta), \end{aligned}$$

where $s' \sim p(s' | s, a)$.

Deep Q-learning

Consider the following **regression task**:

- s, a is input;
- $y \in \mathbb{R}$ is target:

$$y(s, a) := r + \gamma \max_{a'} Q(s', a', \theta_k)$$

where $r = r(s, a), s' \sim p(s' | s, a)$;

- MSE loss function: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

Important!

We can do the same with other Bellman equations (e.g. for V^π)!

Intuition: why this is Q-learning

Consider the following setting:

- *tabular* parametric family $Q(s, a, \theta) := \theta_{s,a}$;
- define Bellman target as $y := r(s, a) + \gamma \max_{a'} Q(s', a', \theta)$;

$$\theta \leftarrow \theta - \alpha (y - Q(s, a, \theta)) \nabla_\theta Q(s, a, \theta)$$

$$\theta \leftarrow \theta - \alpha (y - Q(s, a, \theta)) \text{OHE}(s, a)$$

$$\theta_{s,a} \leftarrow \theta_{s,a} - \alpha (y - Q(s, a, \theta))$$

Why stop gradients from target?

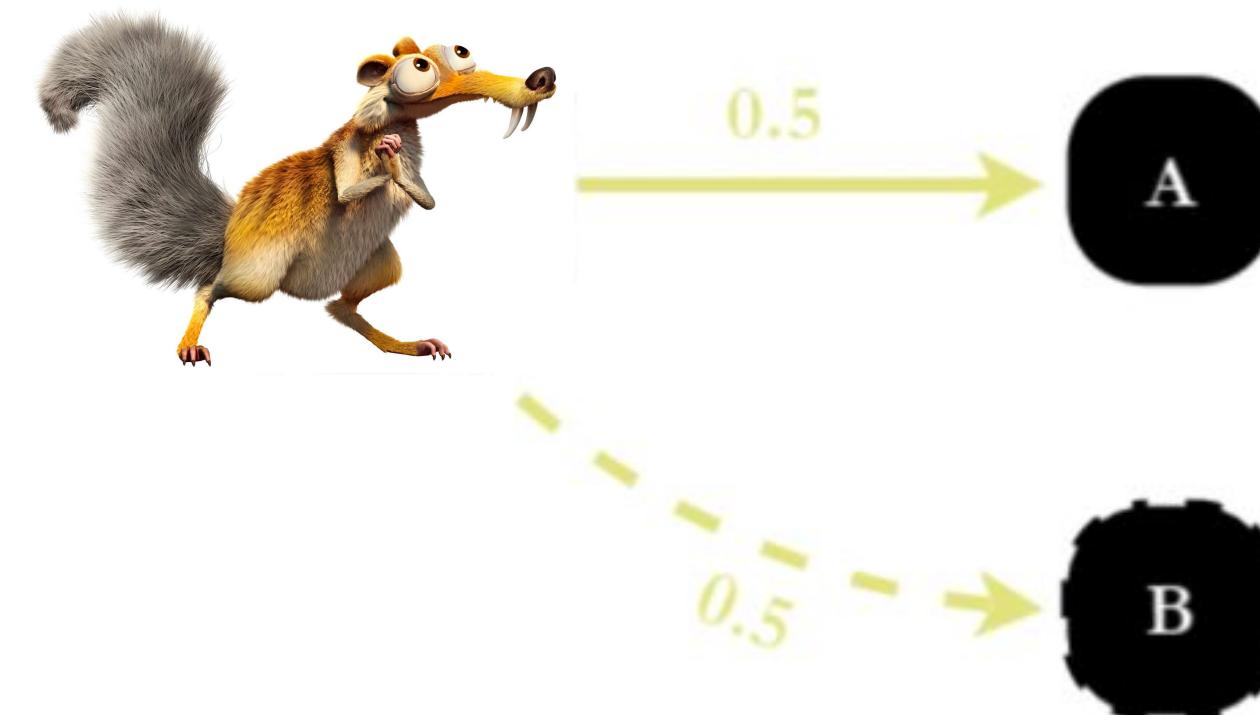
$$\mathbb{E}_{s,a,r,s'} \left(\underbrace{r + \gamma \max_a Q(s', a') - Q(s, a)}_{\text{fixed; no gradient flow!}} \right)^2 \rightarrow \min_Q$$

This is a consequence of **causality**:

- ✓ match *past* estimations to *future*;
- ✗ do not match *future* estimations to *past*;

Note that unlike supervised learning:

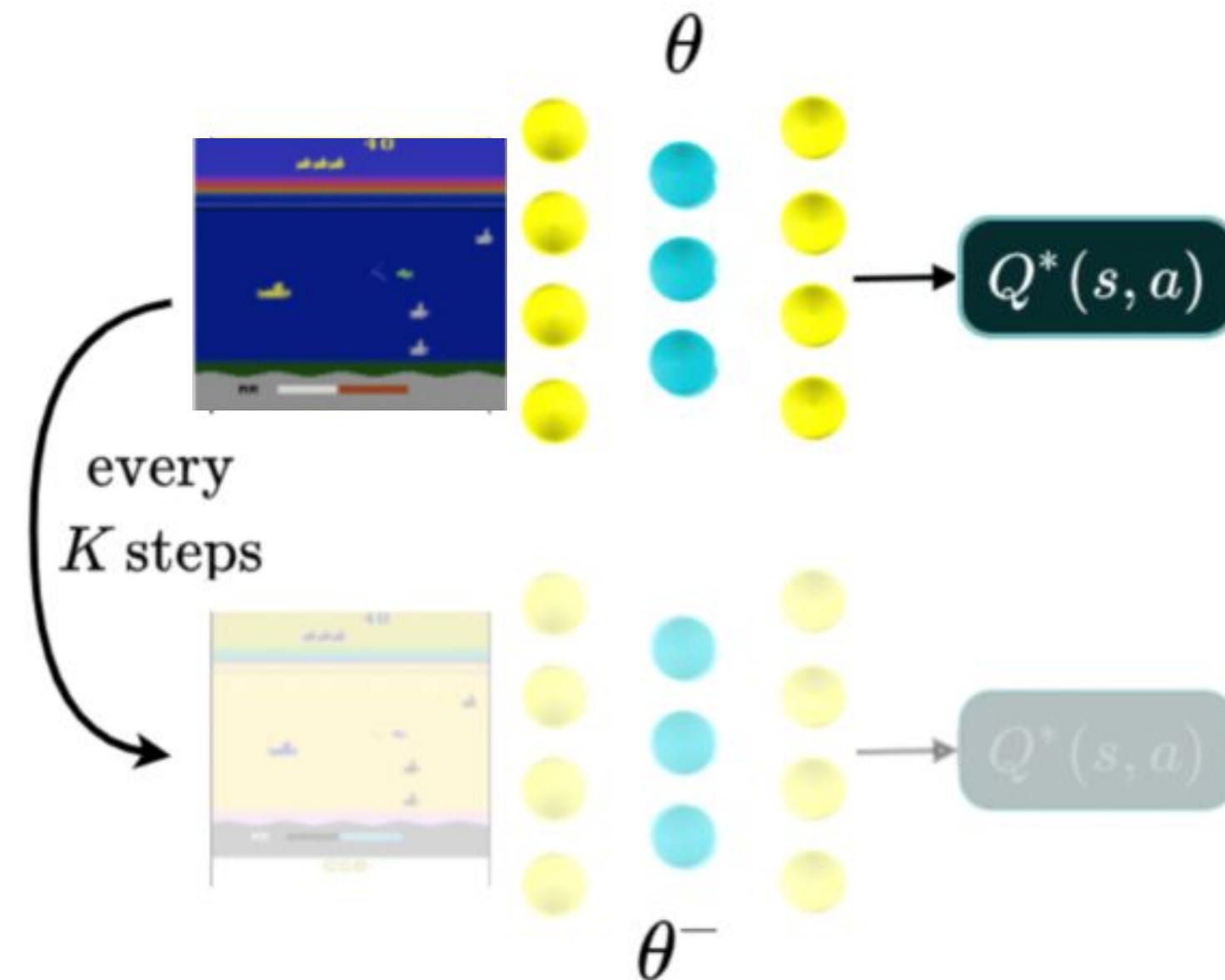
- target «depends» on our own model;
- it is still *better* than we currently have;
(«provides direction»)



Target network

For how long to solve one regression task?

- Till convergence
 - ✗ too long
- One SGD iteration
 - ✗ completely unstable :(
- ✓ ≈ 1000 SGD iterations
 - $\theta^- \leftarrow \theta$ every K SGD iterations
 - $\theta^- \leftarrow (1 - \beta)\theta^- + \beta\theta$



Store a copy of your old Q-network $Q(s, a, \theta^-)$

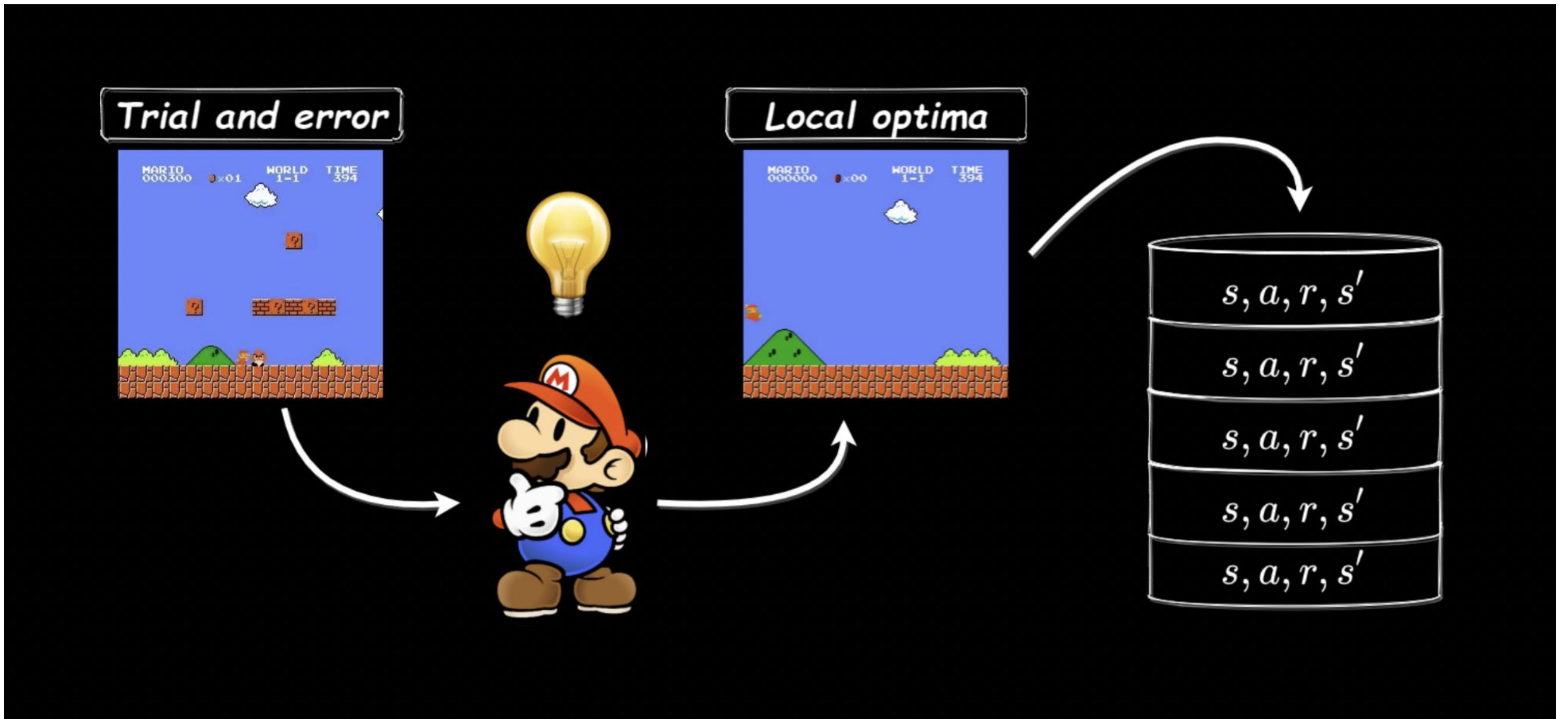
Distributional shift issue



Experience Replay



Do we still need exploration?



Full alrogithm

Deep Q-learning

Initialize $Q(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \emptyset$;

observe s_0 ;

for $k = 0, 1, 2 \dots$

- take action $a_k \sim \varepsilon\text{-greedy}(Q(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \text{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \text{done}_{k+1})$ in \mathcal{D} ;
- sample batch of transitions $\mathbb{T} := (s, a, r, s', \text{done})$ from \mathcal{D} ;
- $y(\mathbb{T}) := r + \gamma(1 - \text{done}) \max_{a'} Q(s', a', \theta^-)$;
- perform a step of gradient descent:

$$\theta \leftarrow \theta - \frac{\alpha}{B} \sum_{\mathbb{T}} \nabla_{\theta} (Q(s, a, \theta) - y(\mathbb{T}))^2$$

- update target network: if $k \bmod K = 0$: $\theta^- \leftarrow \theta$

Recap

- Monte-Carlo methods (sample + epsilon greedy)
- Temporal difference learning (SARSA)
- Q-learning
- On-policy/off-policy
- Replay buffer
- DQN