

Lecture 10: Model-free learning

Radoslav Neychev

Harbour.Space University
19.06.2020, Barcelona, Spain

References

These slides are almost the exact copy of Practical RL course week 3 slides.
Special thanks to YSDA team for making them publicly available.

Original slides link: [week03_model_free](#)

- Value iteration brief overview
- Learning from trajectories
 - MC approach
 - Temporal difference
- Q-learning
- Exploration-exploitation tradeoff
- SARSA
- Experience replay
- Practice

Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal

Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal

Cumulative reward is called a return:

$$G_t \triangleq R_t + R_{t+1} + R_{t+2} + \dots + R_T$$

E.g.: reward in **chess** – value of taken opponent's piece

Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal

Cumulative reward is called a return:

$$\begin{array}{c} \text{end of an episode} \end{array} \quad \begin{array}{c} \text{immediate reward} \end{array}$$
$$G_t \triangleq R_t + R_{t+1} + R_{t+2} + \dots + R_T$$

E.g.: reward in **chess** – value of taken opponent's piece

E.g.: data center non-stop cooling system

- **States** – temperature measurements
- **Actions** – different fans speed
- **R = 0** for exceeding temperature thresholds
- **R = +1** for each second system is cool

What could go wrong with such a design?

E.g.: data center non-stop cooling system

- **States** – temperature measurements
- **Actions** – different fans speed
- **R = 0** for exceeding temperature thresholds
- **R = +1** for each second system is cool

What could go wrong with such a design?

Infinite return for **non optimal** behaviour!

$$G_t = 1 + 1 + 0 + 1 + 1 + 0 + \dots = \sum_{t=1}^{\infty} R_t = \infty$$

E.g.: cleaning robot

- States – dust sensors, air
- Actions – cleaning / rest / conditioning on or off
- $R = 100$ for long tedious floor cleaning task done
- $R = 1$ for turning air conditioning on-off
- Episode ends each day

What could go wrong with such a design?

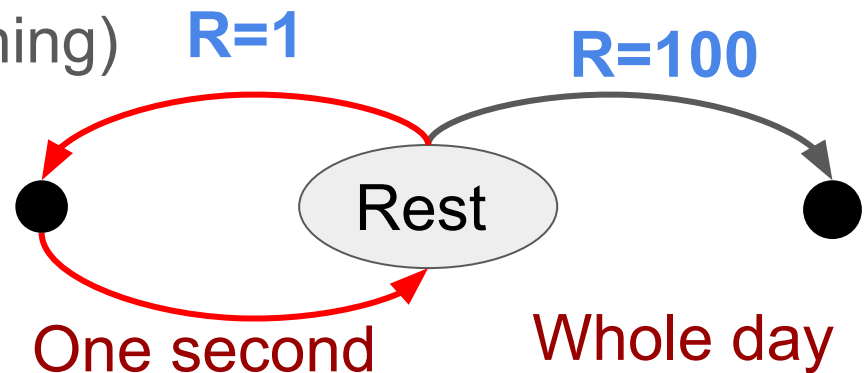
E.g.: cleaning robot

- **States** – dust sensors, air
- **Actions** – cleaning / rest / conditioning on or off
- **R = 100** for long tedious floor cleaning task done
- **R = 1** for turning air conditioning on-off
- Episode **ends** each **day**

What could go wrong with such a design?

Reward(air) < Reward(cleaning)
Time(air) << Time(cleaning)

Positive feedback loop!





OpenAI blog post about faulty rewards: <https://openai.com/blog/faulty-reward-functions/>

Reward discounting

Reward discounting

Get rid of infinite sum by **discounting** $0 \leq \gamma < 1$

$$G_t \triangleq R_t + \underset{\substack{\text{discount factor} \quad \nearrow}}{\gamma} R_{t+1} + \gamma^2 R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The same cake compared to today's one worth

- γ times less tomorrow
- γ^2 times less the day after tomorrow



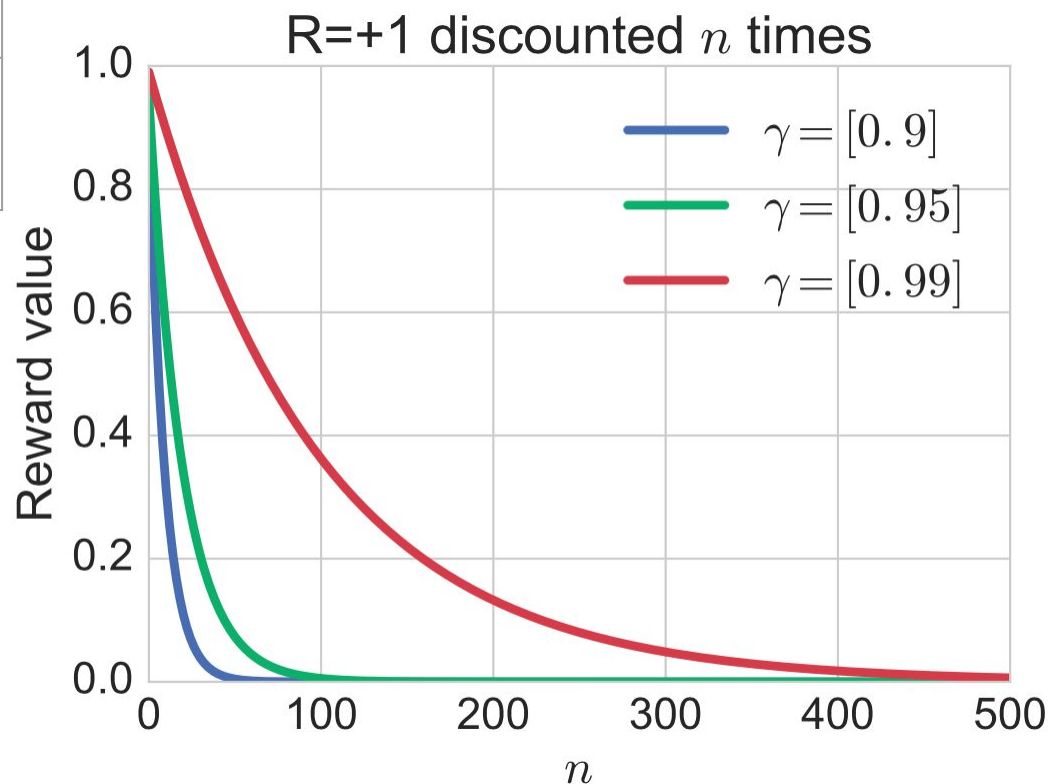
γ will eat it day by day

Discounting makes sums finite

Maximal return for **R = +1**

$$G_0 = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma}$$

γ	0.9	0.95	0.99
$\frac{1}{1-\gamma}$	10	20	100



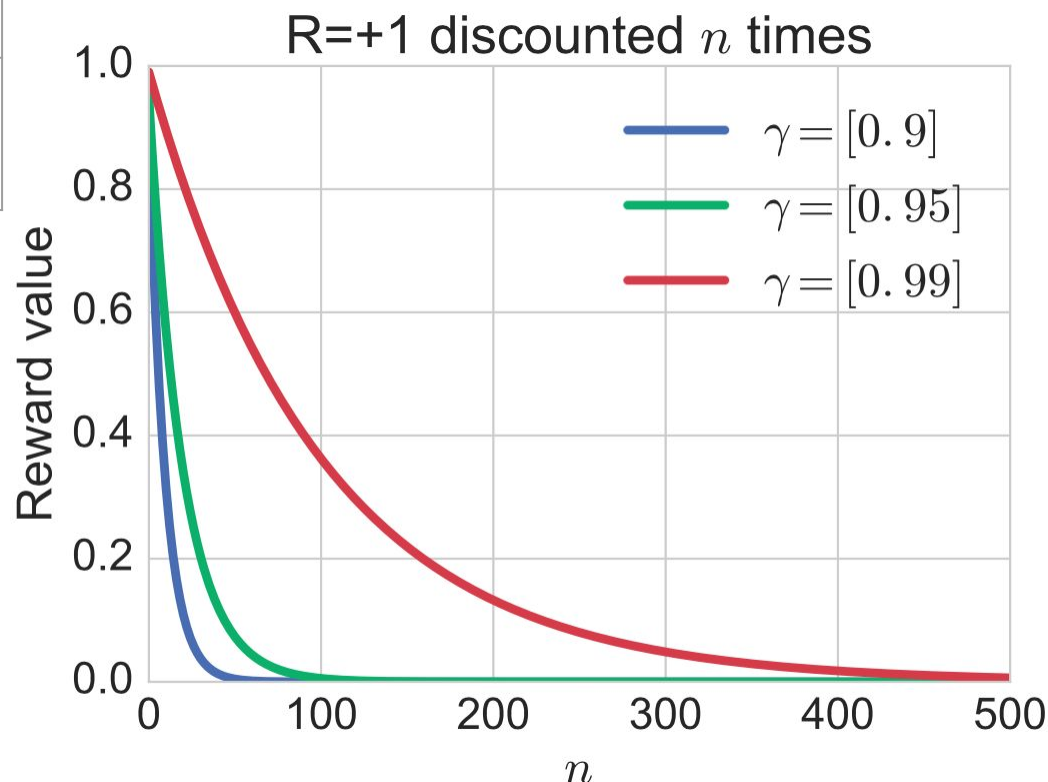
Discounting makes sums finite

Maximal return for **R = +1**

$$G_0 = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma}$$

γ	0.9	0.95	0.99
$\frac{1}{1-\gamma}$	10	20	100

Any **discounting**
changes optimisation
task and its solution!



Discounting is inherent to humans

- Quasi-hyperbolic $f(t) = \beta\gamma^t$
- Hyperbolic discounting $f(t) = \frac{1}{1 + \beta t}$

Discounting is inherent to humans

- Quasi-hyperbolic $f(t) = \beta\gamma^t$
- Hyperbolic discounting $f(t) = \frac{1}{1 + \beta t}$

Mathematical convenience

$$\begin{aligned} G_t &= R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \dots) \\ &= \boxed{R_t + \gamma G_{t+1}} \end{aligned}$$

Remember this one!
We will need it later

Discounting is a stationary end-of-effect model

Any action affects (1) immediate reward (2) next state

Discounting is a stationary end-of-effect model

Any action affects (1) immediate reward (2) next state

Action indirectly affects future rewards 

But how long does this effect lasts?

$$\begin{aligned} G_0 &= R_0 + \gamma R_1 + \gamma^2 R_2 + \dots + \gamma^T R_T \\ &= (1 - \gamma) R_0 \\ &\quad + (1 - \gamma) \gamma (R_0 + R_1) \\ &\quad + (1 - \gamma) \gamma^2 (R_0 + R_1 + R_2) \\ &\quad \dots \\ &\quad + \gamma^T \cdot \sum_{t=0}^T R_t \end{aligned}$$

G is expected return under stationary end-of-effect model

Discounting is a stationary end-of-effect model

Any action affects (1) immediate reward (2) next state

Action indirectly affects future rewards _____↑

But how long does this effect lasts?

$$\begin{aligned} G_0 &= R_0 + \gamma R_1 + \gamma^2 R_2 + \dots + \gamma^T R_T \\ &= \boxed{(1 - \gamma)} R_0 + \boxed{(1 - \gamma)} \gamma (R_0 + R_1) \\ &\quad + \boxed{(1 - \gamma)} \gamma^2 (R_0 + R_1 + R_2) \\ &\quad \dots \\ &\quad + \gamma^T \cdot \sum_{t=0}^T R_t \end{aligned}$$

“End of effect” probability → → →

“Effect continuation” probability →

G is expected return under stationary end-of-effect model

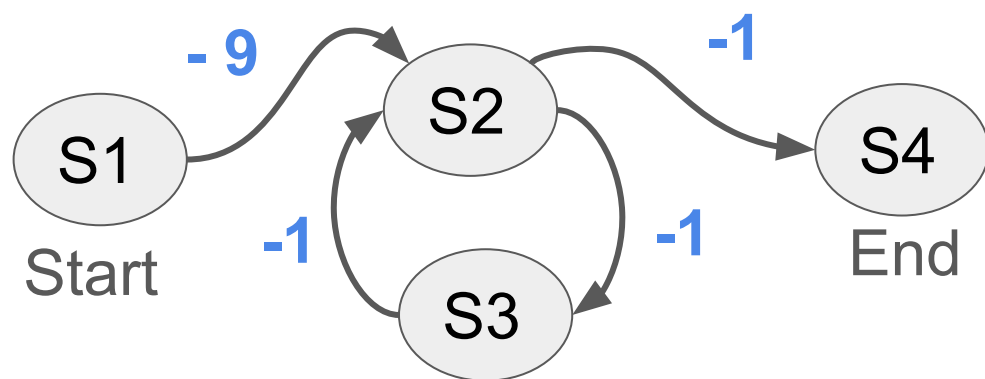
Reward design – don't shift, reward for WHAT

- E.g.: chess – value of taken opponent's piece
 - Problem: agent will not have a desire to win!
- E.g.: cleaning robot, **+100** (cleaning), **+0.1** (on-off)
 - Problem: agent will not bother cleaning the floor!

Reward design – don't shift, reward for WHAT

- **E.g.:** chess – value of taken opponent's piece
 - **Problem:** agent will not have a desire to win!
- **E.g.:** cleaning robot, **+100** (cleaning), **+0.1** (on-off)
 - **Problem:** agent will not bother cleaning the floor!

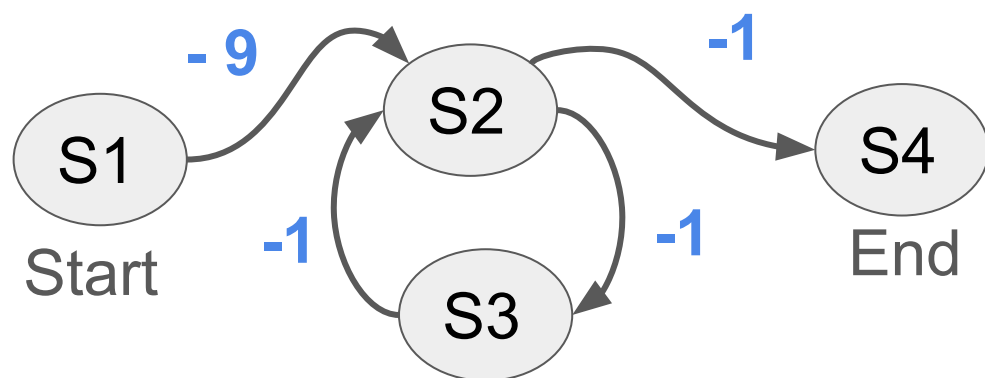
Take away: reward only for **WHAT**, but never for **HOW**



Reward design – don't shift, reward for WHAT

- **E.g.:** chess – value of taken opponent's piece
 - **Problem:** agent will not have a desire to win!
- **E.g.:** cleaning robot, **+100** (cleaning), **+0.1** (on-off)
 - **Problem:** agent will not bother cleaning the floor!

Take away: reward only for **WHAT**, but never for **HOW**



Take away: do not **subtract** mean from rewards

Reward design – scaling, shaping

What transformations do not change optimal policy?

- Reward **scaling** – division by positive constant
 - May be useful in practise for approximate methods

Reward design – scaling, shaping

What transformations do not change optimal policy?

- Reward **scaling** – division by positive constant
 - May be useful in practise for approximate methods
- Reward **shaping** – we could add to all rewards in MDP values of **potential-based shaping function** $F(s, a, s')$ without changing an optimal policy:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s)$$

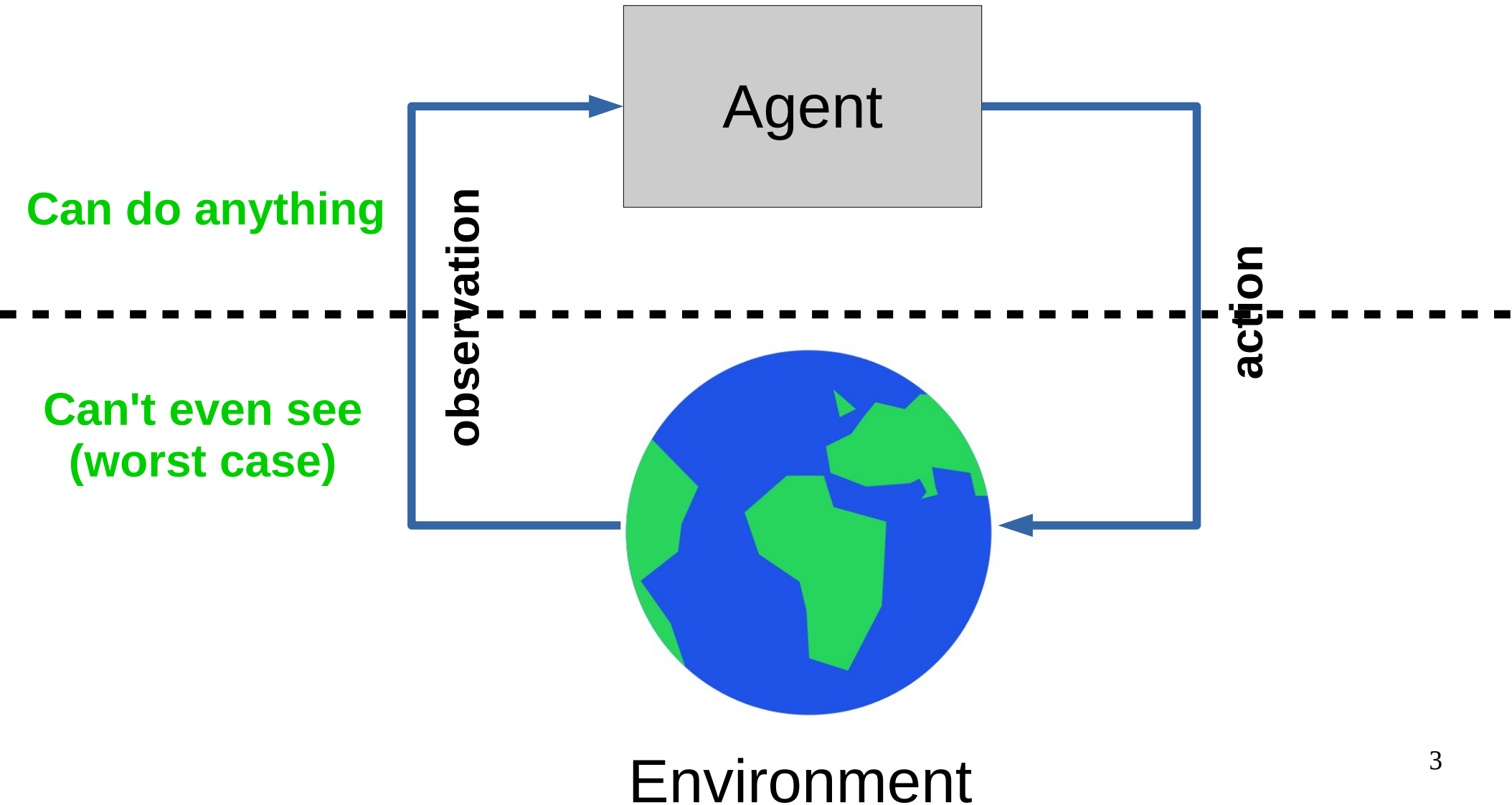
Intuition: when no discounting F adds as much as it subtracts from the total return

Previously...

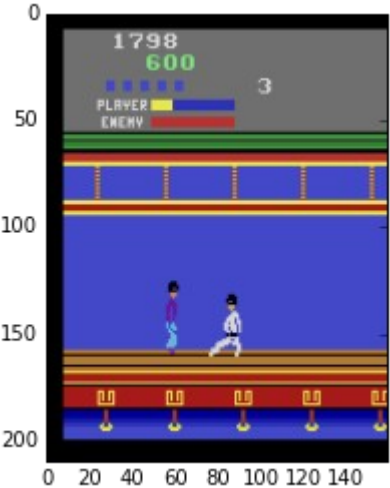
- $V(s)$ and $V^*(s,a)$
- know V^* and $P(s'|s,a) \rightarrow$ know optimal policy
- We can learn V^* with dynamic programming

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$

Decision process in the wild



Decision process in the wild



Model-free setting:

We don't know actual

$$P(s',r|s,a)$$

Whachagonnado?

Model-free setting:

We don't know actual

$$P(s',r|s,a)$$

Learn it?

Get rid of it?

More new letters

- $V_{\pi}(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π
- $V^*(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π^*

More new letters

- $V_{\pi}(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π
- $V^*(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π^*
- $Q_{\pi}(\mathbf{s}, \mathbf{a})$ – expected G from state \mathbf{s}
 - if you start by taking action \mathbf{a}
 - and follow π from next state on
- $Q^*(\mathbf{s}, \mathbf{a})$ – guess what it is :)

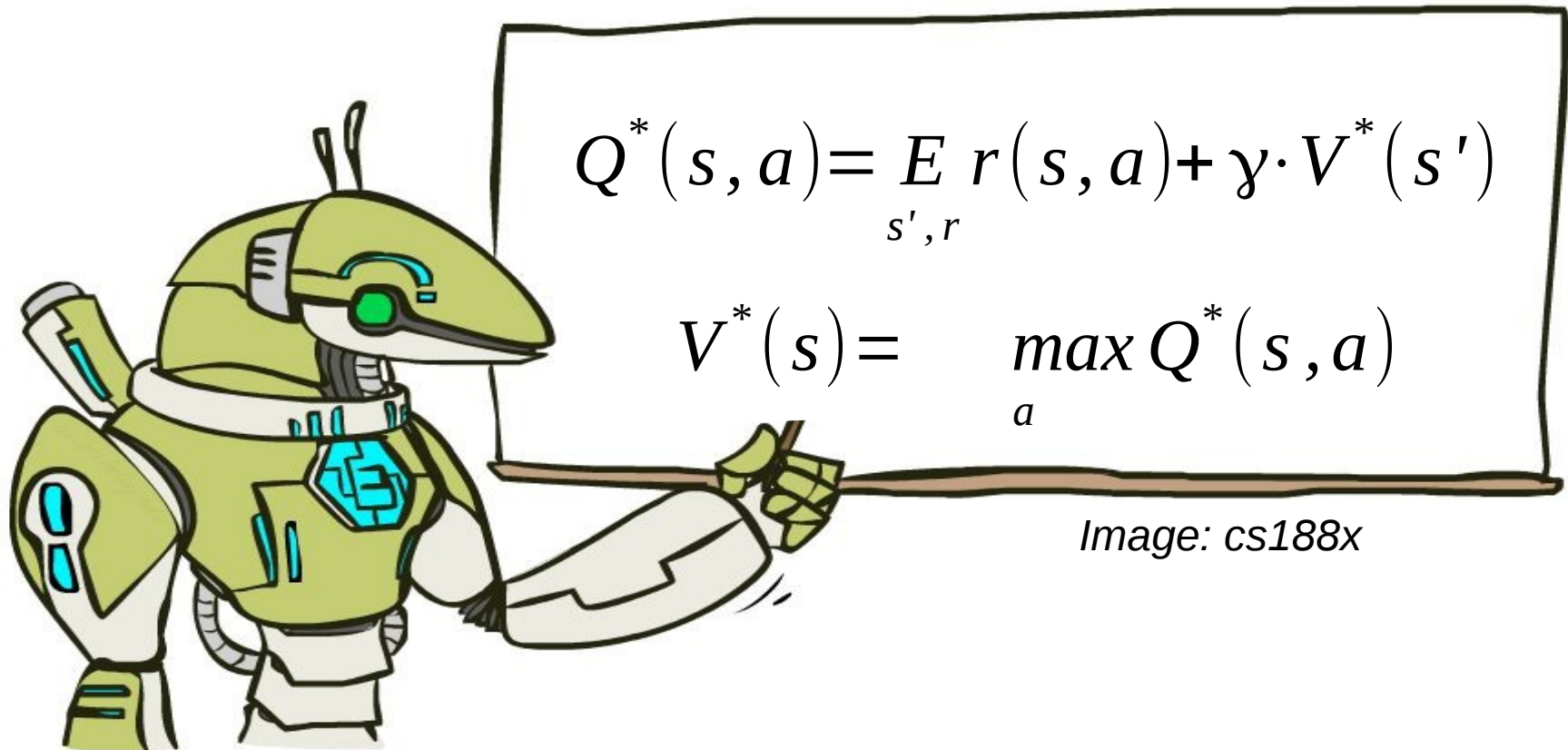
More new letters

- $V_{\pi}(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π
- $V^*(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π^*
- $Q_{\pi}(\mathbf{s}, \mathbf{a})$ – expected G from state \mathbf{s}
 - if you start by taking action \mathbf{a}
 - and follow π from next state on
- $Q^*(\mathbf{s}, \mathbf{a})$ – same as $Q_{\pi}(\mathbf{s}, \mathbf{a})$ where $\pi = \pi^*$

Trivia

- Assuming you know $Q^*(s,a)$,
 - how do you compute π^*
 - how do you compute $V^*(s)$?
- Assuming you know $V(s)$
 - how do you compute $Q(s,a)$?

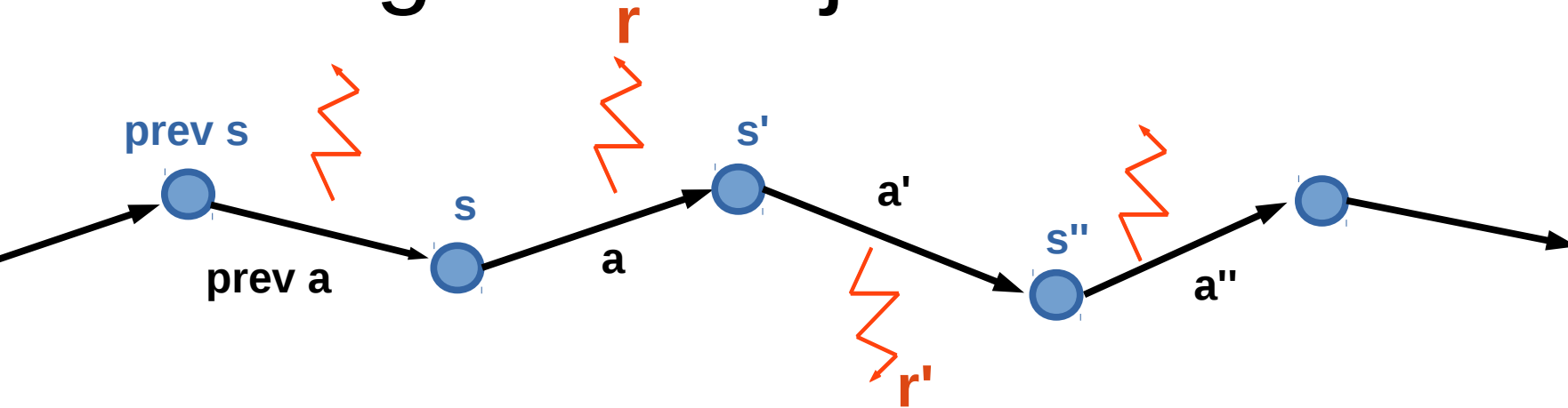
To sum up



Action value $Q_{\pi}(s, a)$ is the expected total reward **G** agent gets from state **s** by taking action **a** and following policy **π** from next state.

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

Learning from trajectories



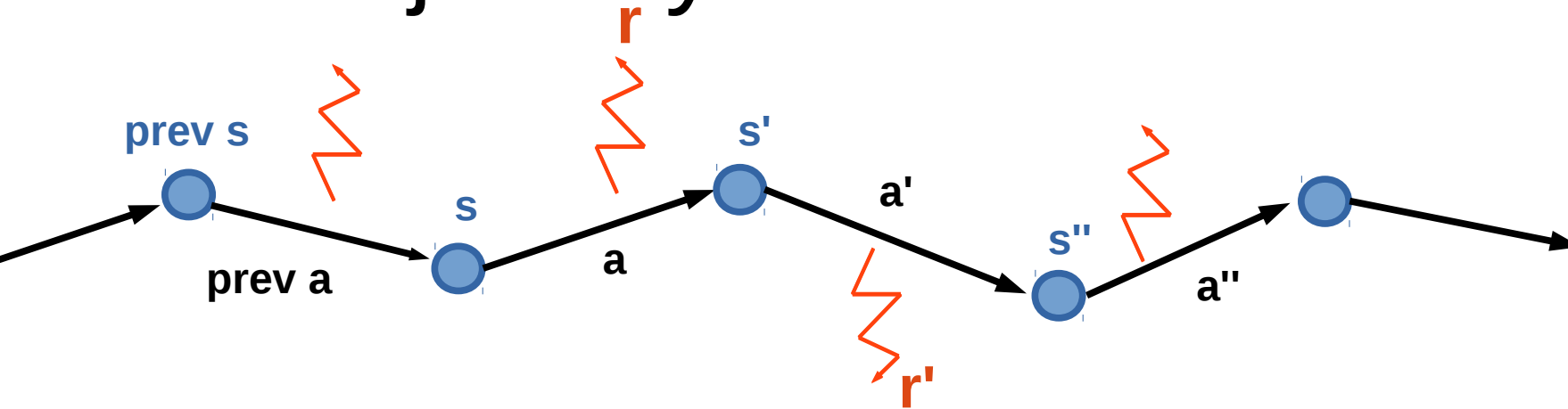
Model-based: you know $P(s'|s,a)$

- can apply dynamic programming
- can plan ahead

Model-free: you can sample trajectories

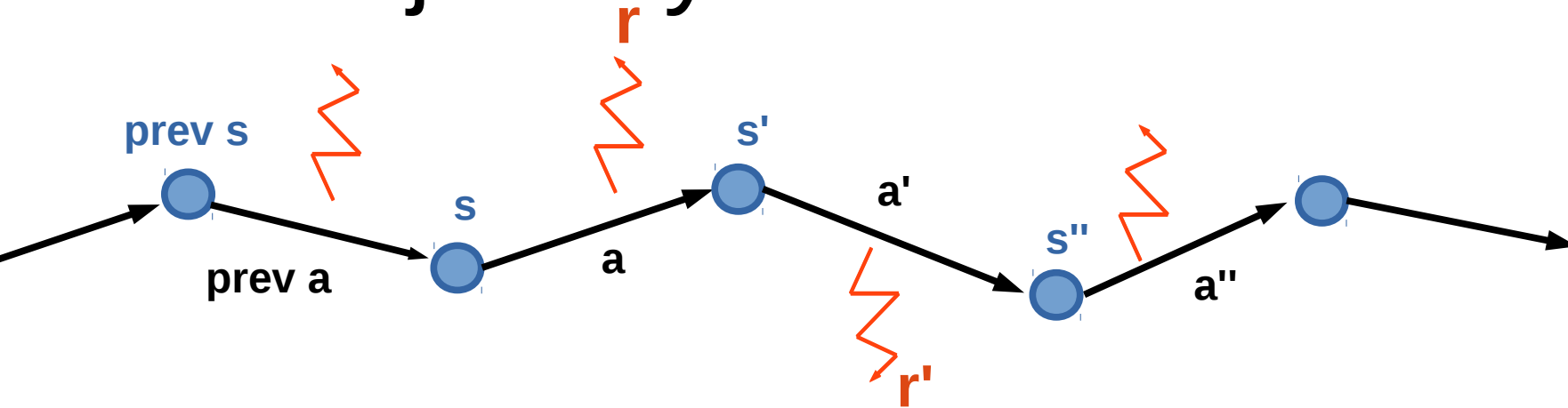
- can try stuff out
- insurance not included

MDP trajectory



- Trajectory is a sequence of
 - states (s)
 - actions (a)
 - rewards (r)
- We can only sample trajectories

MDP trajectory



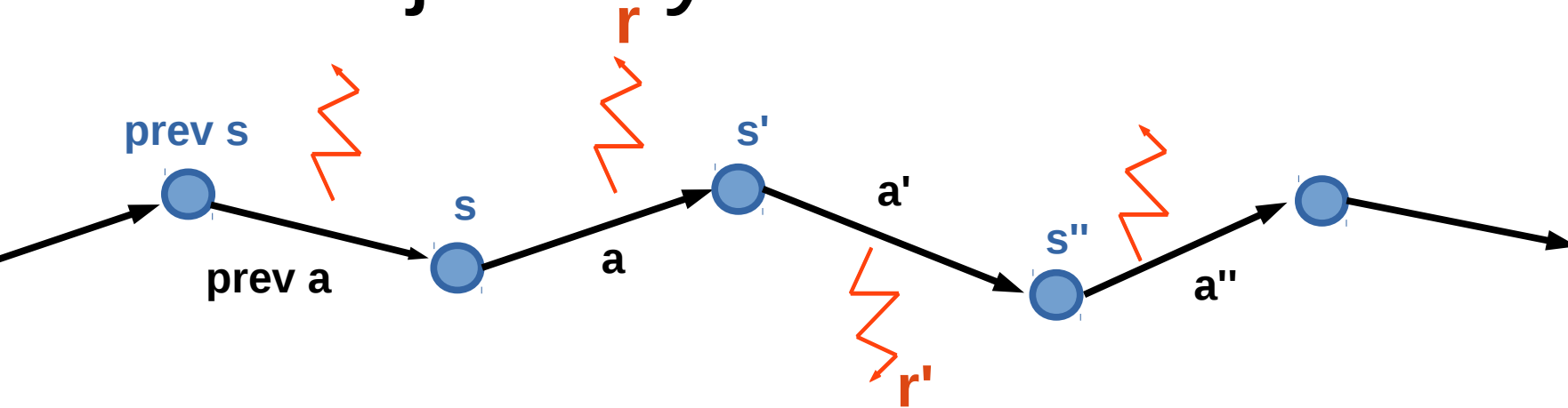
- Trajectory is a sequence of

- states (s)
- actions (a)
- rewards (r)

Q: What to learn?
 $V(s)$ or $Q(s,a)$

- We can only sample trajectories

MDP trajectory



- Trajectory is a sequence of

- states (s)
- actions (a)
- rewards (r)

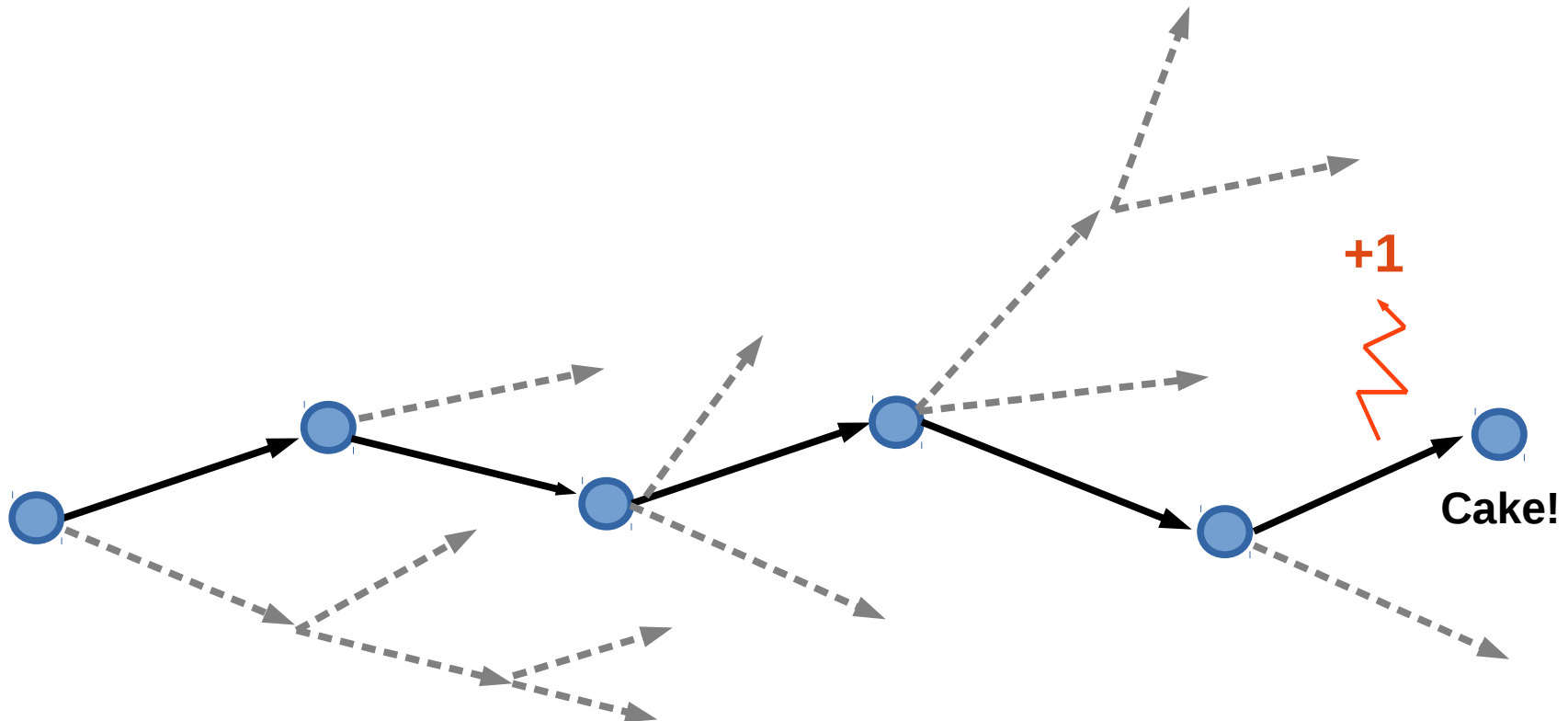
Q: What to learn?
 $V(s)$ or $Q(s,a)$

$V(s)$ is useless
without $P(s'|s,a)$

- We can only sample trajectories

Idea 1: monte-carlo

- Get all trajectories containing particular (s,a)
- Estimate $G(s,a)$ for each trajectory
- Average them to get expectation



Idea 1: monte-carlo

- Get all trajectories containing particular (s,a)
- Estimate $G(s,a)$ for each trajectory
- Average them to get expectation

takes a lot of sessions



Image: super meat boy

Idea 2: temporal difference

- Remember we can improve $Q(s,a)$ iteratively!

$$Q(s_t, a_t) \leftarrow E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

Idea 2: temporal difference

- Remember we can improve $Q(s,a)$ iteratively!

$$Q(s_t, a_t) \leftarrow E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

↑
That's $Q^*(s,a)$

↑
That's value for π^*
aka optimal policy

Idea 2: temporal difference

- Remember we can improve $Q(s,a)$ iteratively!

$$Q(s_t, a_t) \leftarrow E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

↑
That's $Q^*(s,a)$

↑
That's value for π^*
aka optimal policy

↑
That's something
we don't have

What do we do?

Idea 2: temporal difference



Idea 2: temporal difference

- Replace expectation with sampling

$$E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a') \approx \frac{1}{N} \sum_i r_i + \gamma \cdot \max_{a'} Q(s_i^{\text{next}}, a')$$

Idea 2: temporal difference

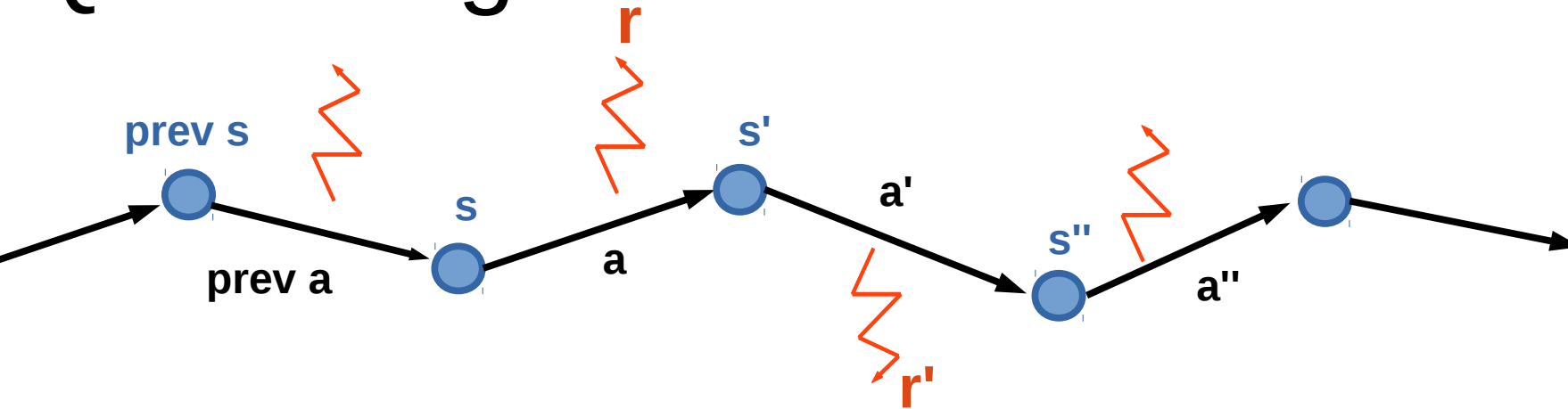
- Replace expectation with sampling

$$E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a') \approx \frac{1}{N} \sum_i r_i + \gamma \cdot \max_{a'} Q(s_i^{next}, a')$$

- Use moving average with just one sample!

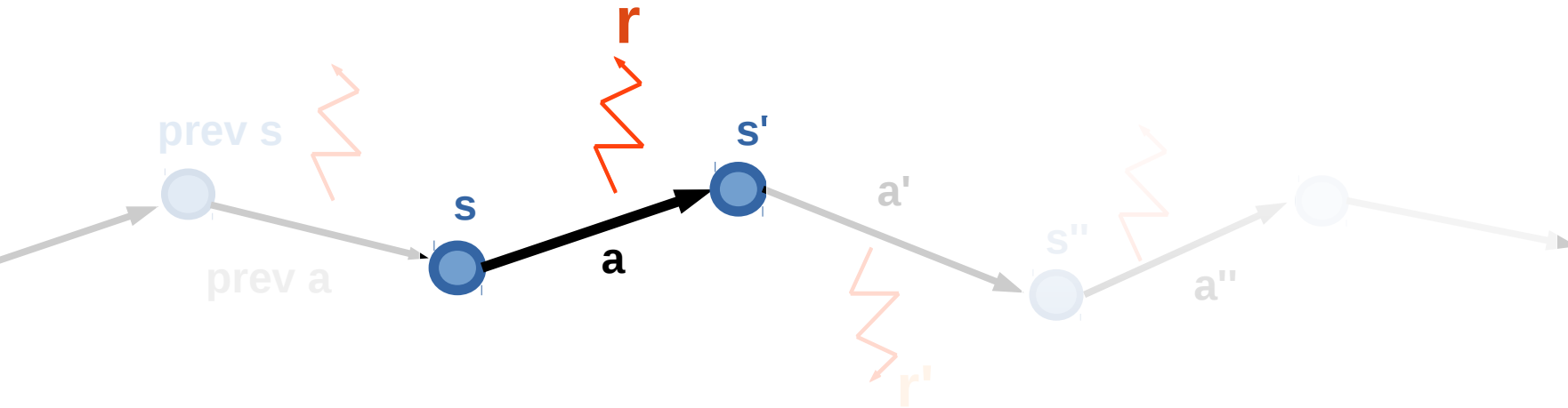
$$Q(s_t, a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')) + (1 - \alpha) Q(s_t, a_t)$$

Q-learning



- Works on a sequence of
 - states (s)
 - actions (a)
 - rewards (r)

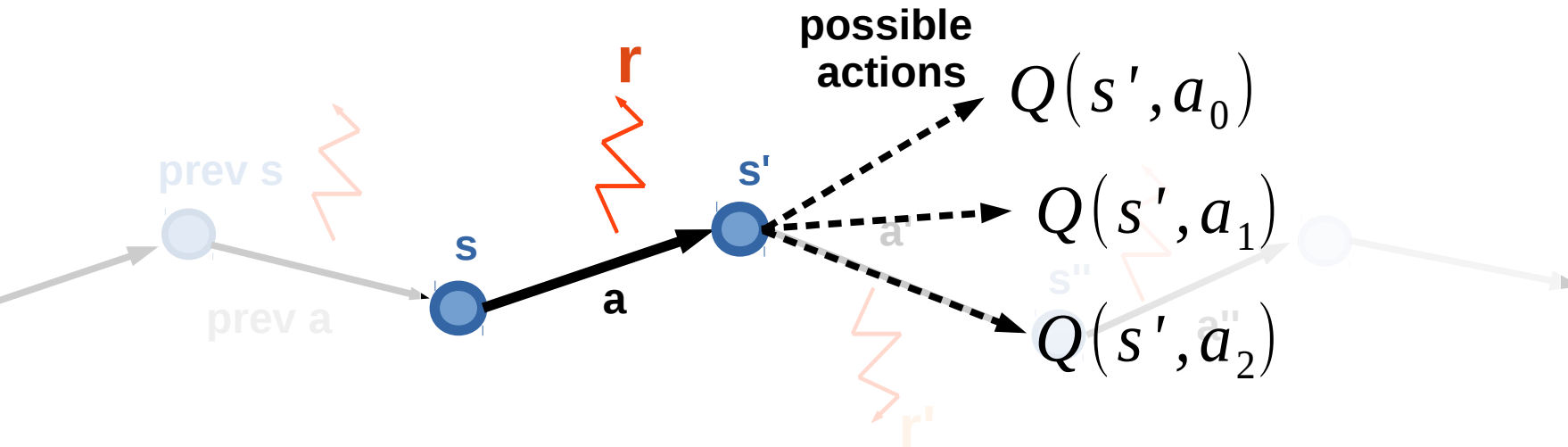
Q-learning



Initialize $Q(s,a)$ with zeros

- Loop:
 - Sample $\langle s, a, r, s' \rangle$ from env

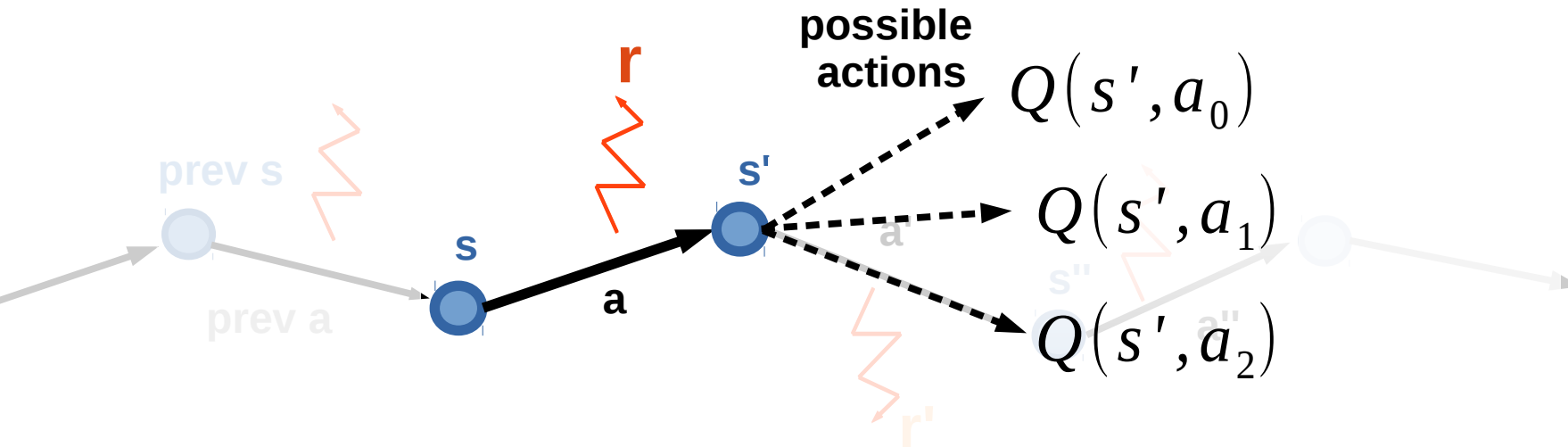
Q-learning



Initialize $Q(s,a)$ with zeros

- Loop:
 - Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$ from env
 - Compute
$$\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',a_i)$$

Q-learning



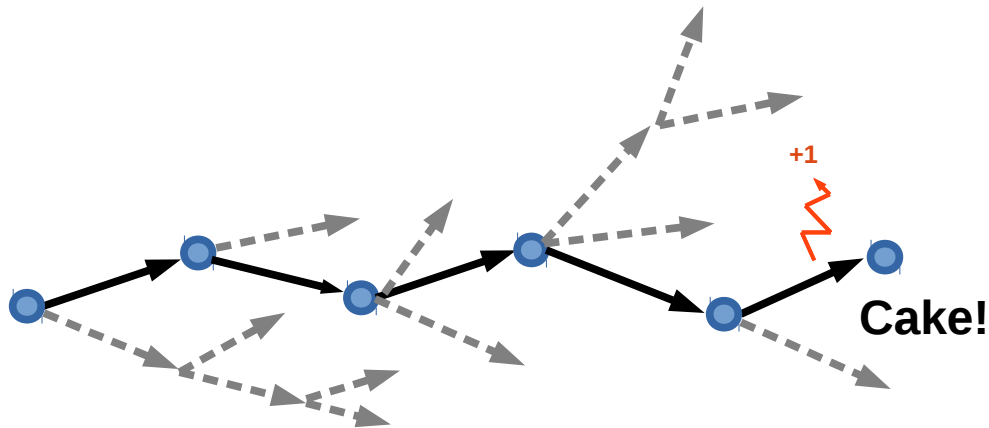
Initialize $Q(s,a)$ with zeros

- Loop:
 - Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$ from env
 - Compute $\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',a_i)$
 - Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

Recap

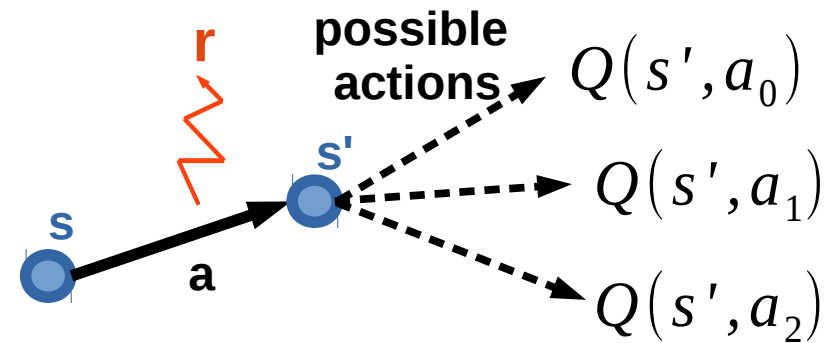
Monte-carlo

- Averages Q over sampled paths



Temporal Difference

- Uses recurrent formula for Q



Nuts and bolts: MC vs TD

Monte-carlo

- Averages Q over sampled paths
- Needs full trajectory to learn
- Less reliant on markov property

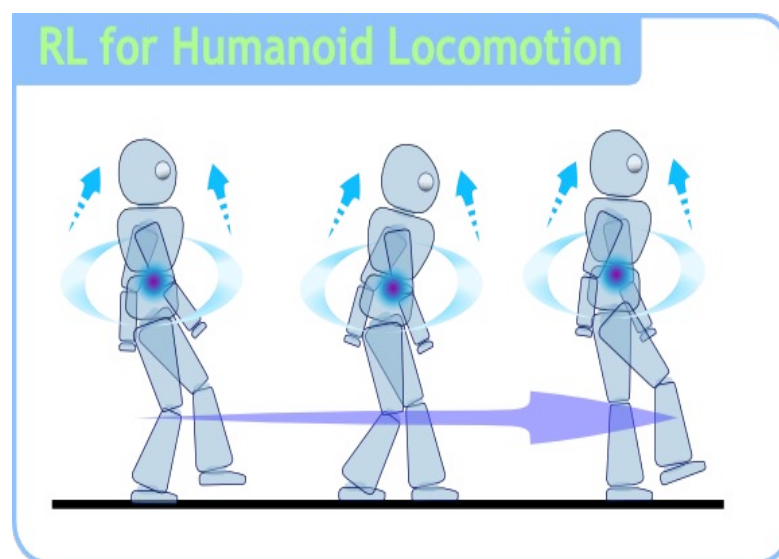
Temporal Difference

- Uses recurrent formula for Q
- Learns from partial trajectory
Works with infinite MDP
- Needs less experience to learn



What could possibly go wrong?

Our mobile robot learns to walk.

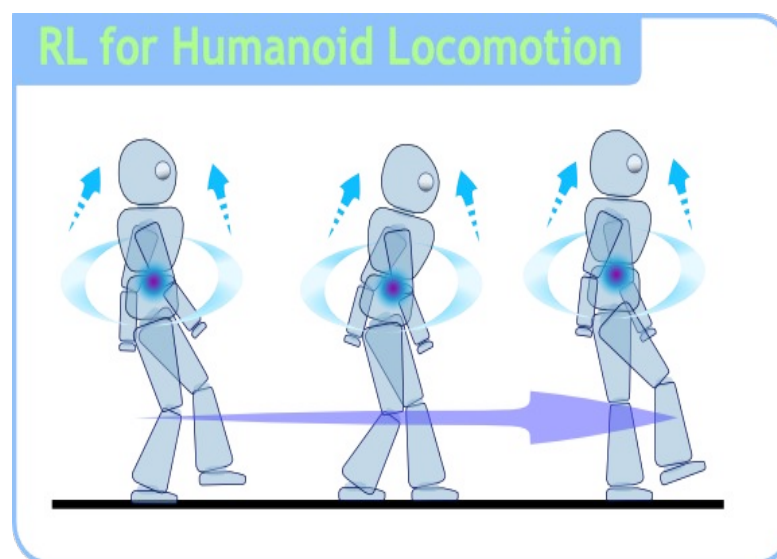


Initial $Q(s,a)$ are zeros
robot uses $\operatorname{argmax} Q(s,a)$

He has just learned to crawl with positive reward! ³⁰

What could possibly go wrong?

Our mobile robot learns to walk.



Initial $Q(s,a)$ are zeros
robot uses $\operatorname{argmax} Q(s,a)$

Too bad, now he will never learn to walk upright = ℓ^1

What could possibly go wrong?

New problem:

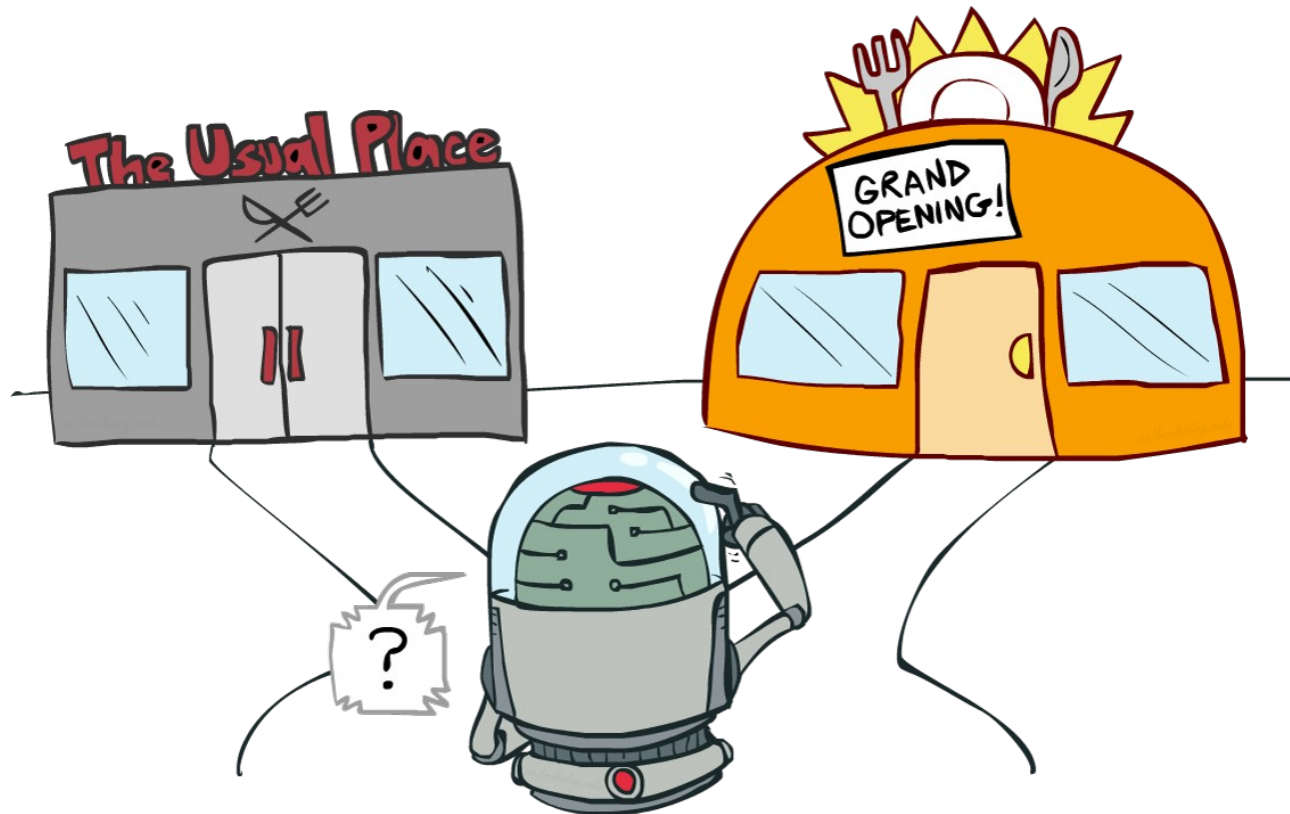
If our agent always takes “best” actions
from his current point of view,

How will he ever learn that other actions
may be better than his current best one?

Ideas?

Exploration Vs Exploitation

Balance between using what you learned and trying to find something even better



Exploration Vs Exploitation

Strategies:

- ϵ -greedy
 - With probability ϵ take random action; otherwise take optimal action.

Exploration Vs Exploitation

Strategies:

- ϵ -greedy
 - With probability ϵ take random action; otherwise take optimal action.
- Softmax
 - Pick action proportional to softmax of shifted normalized Q-values.

$$\pi(a|s) = \text{softmax}\left(\frac{Q(s, a)}{\tau}\right)$$

- More cool stuff coming later

Exploration over time

Idea:

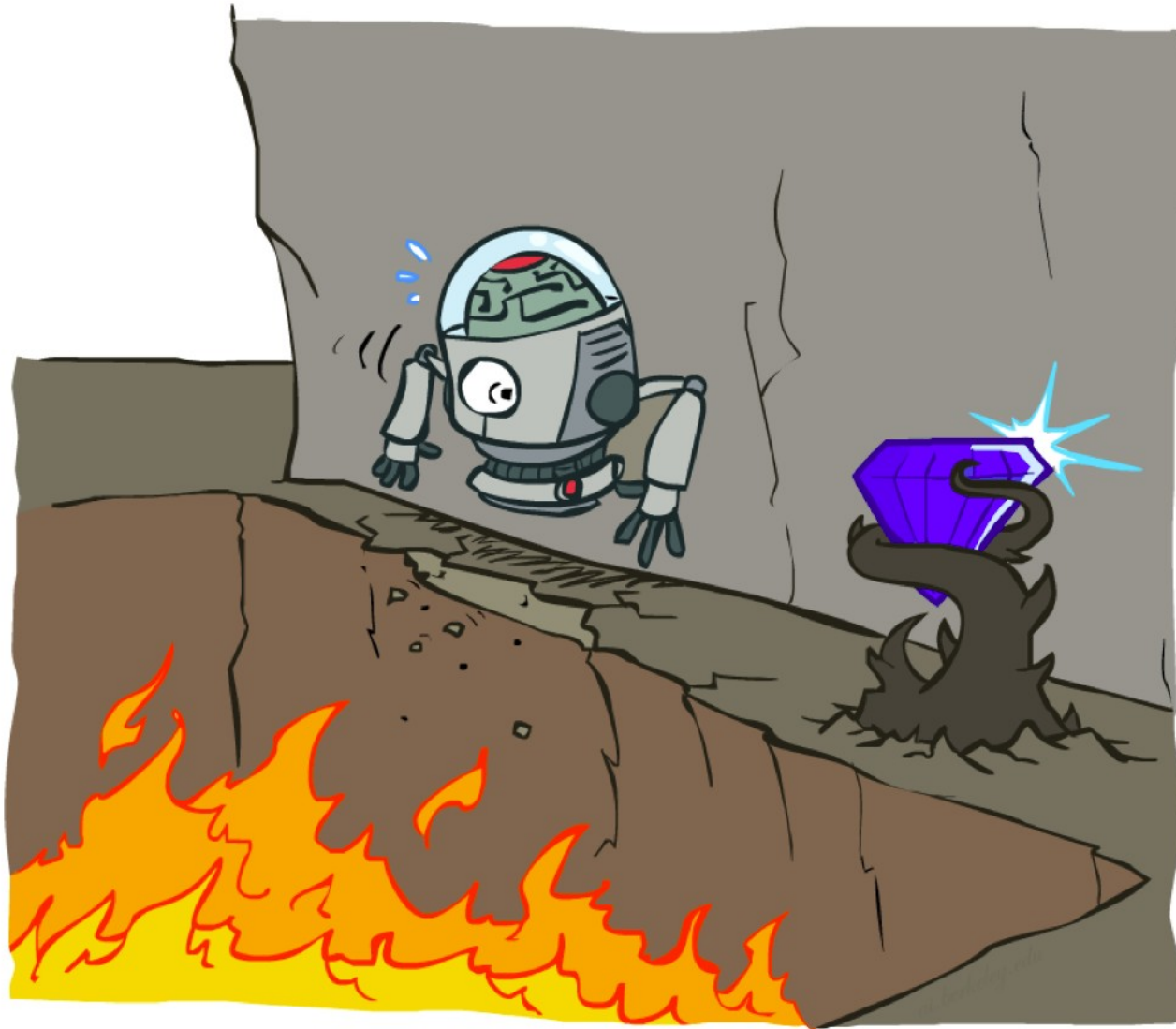
If you want to converge to optimal policy, you need to gradually reduce exploration

Example:

Initialize ϵ -greedy $\epsilon = 0.5$, then gradually reduce it

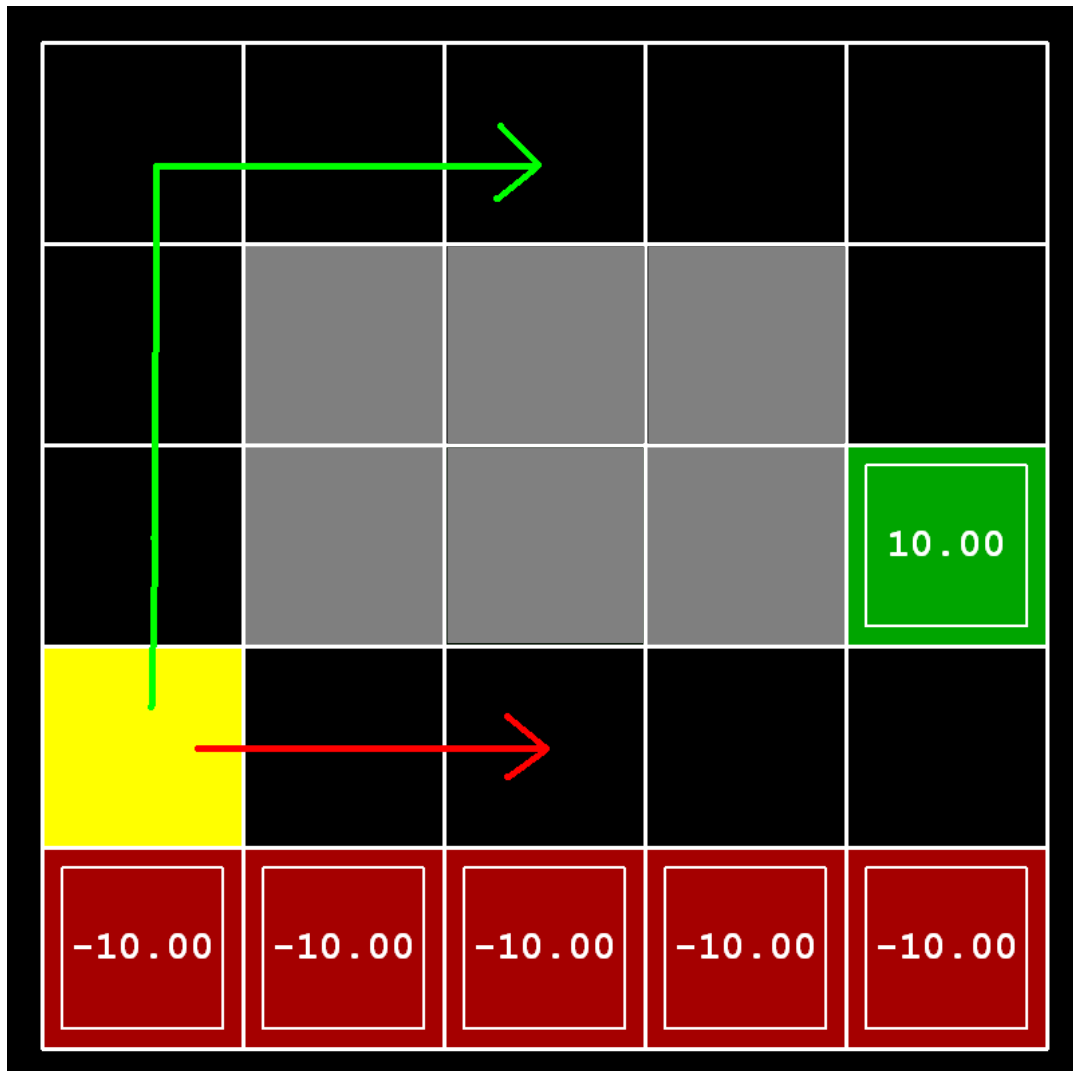
- If $\epsilon \rightarrow 0$, it's **greedy in the limit**
- Be careful with non-stationary environments

Cliff world



Picture from Berkeley CS188x

Cliff world



Conditions

- Q-learning

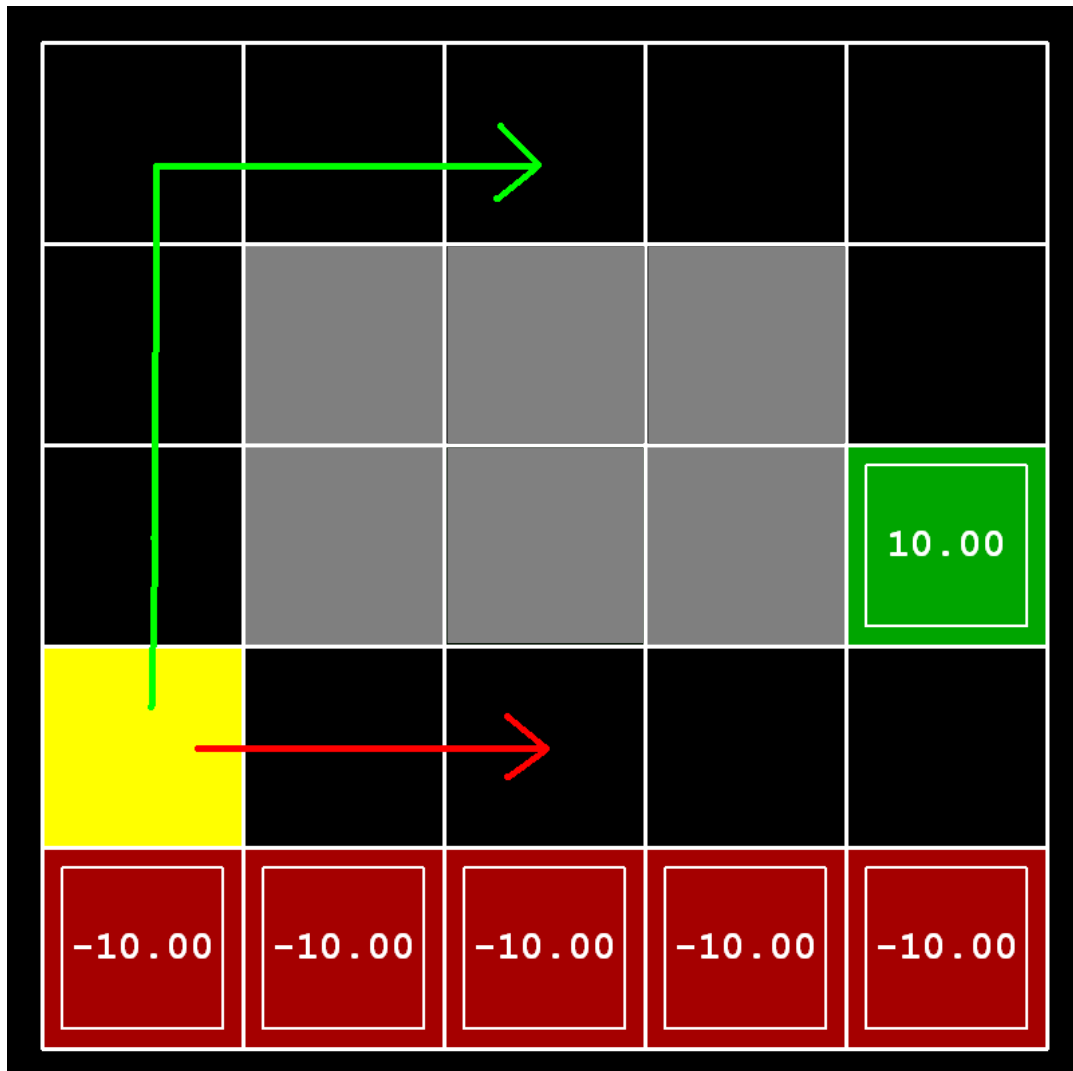
$$\gamma = 0.99 \quad \epsilon = 0.1$$

- no slipping

Trivia:

What will q-learning learn?

Cliff world



Conditions

- Q-learning

$$\gamma = 0.99 \quad \epsilon = 0.1$$

- no slipping

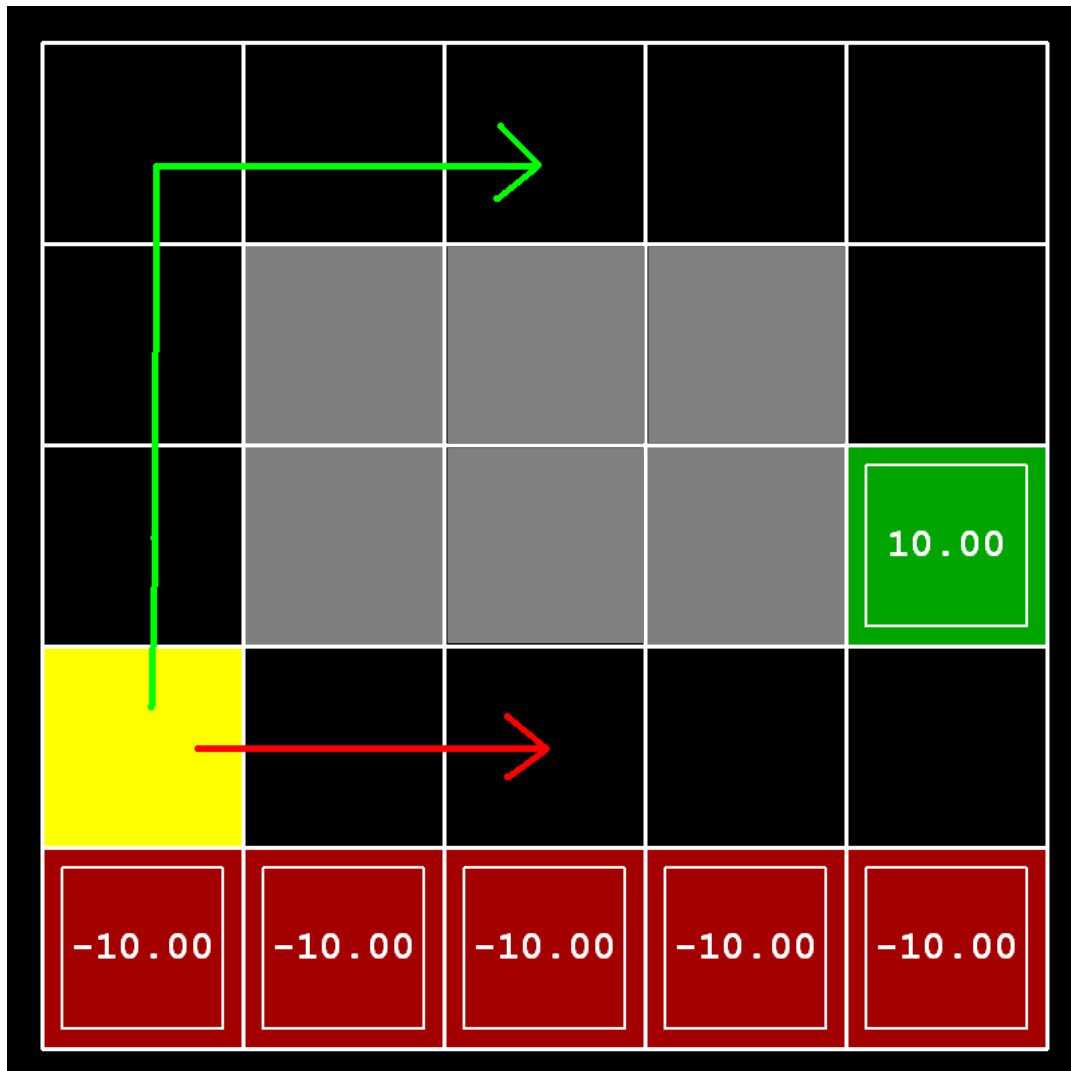
Trivia:

What will q-learning learn?

follow the short path

Will it maximize reward?

Cliff world



Conditions

- Q-learning

$$\gamma = 0.99 \quad \epsilon = 0.1$$

- no slipping

Trivia:

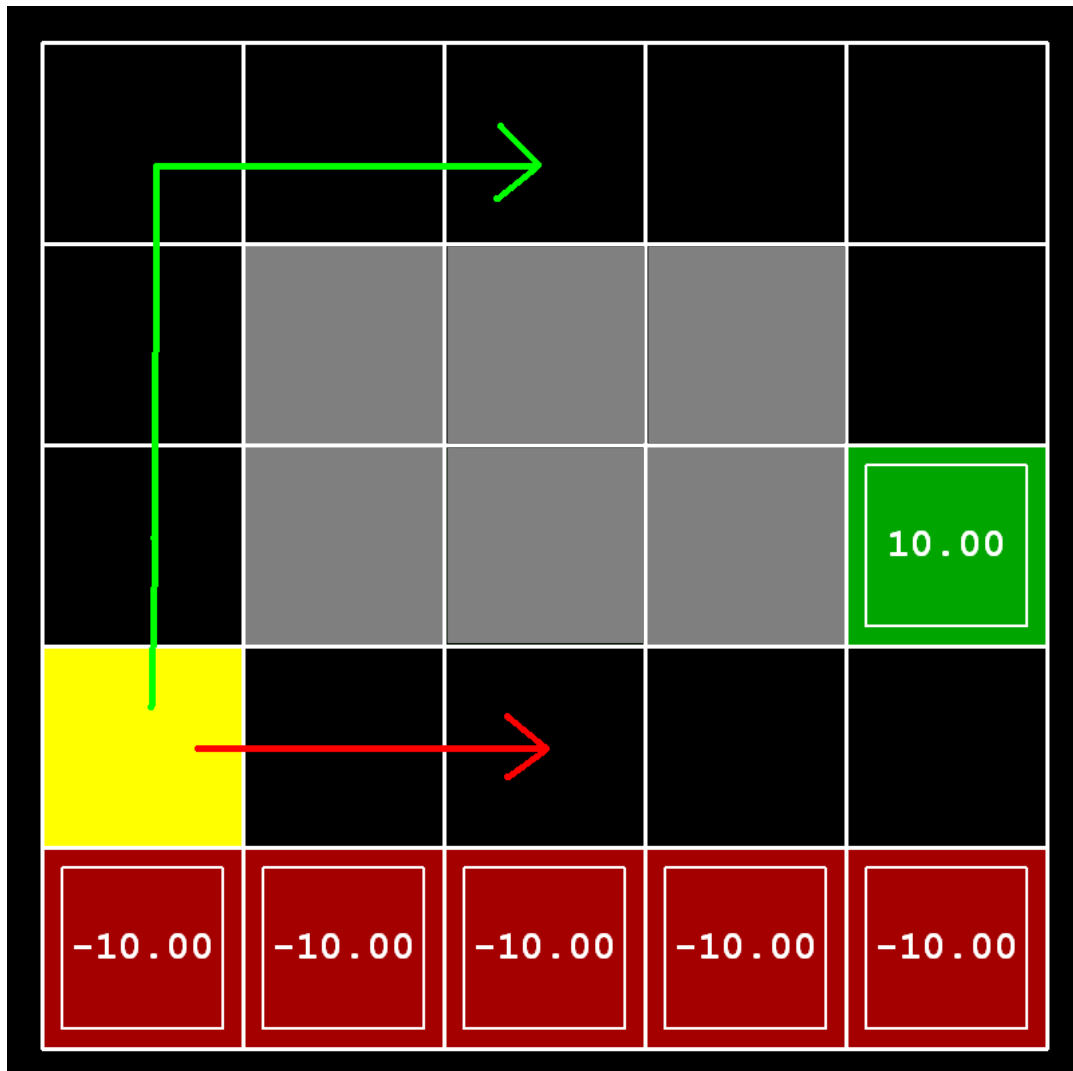
What will q-learning learn?

follow the short path

Will it maximize reward?

**no, robot will fall due to
epsilon-greedy “exploration”**

Cliff world



Conditions

- Q-learning

$$\gamma = 0.99 \quad \epsilon = 0.1$$

- no slipping

**Decisions must account
for actual policy!**

e.g. ϵ -greedy policy

Generalized update rule

Update rule (from Bellman eq.)

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1 - \alpha) Q(s_t, a_t)$$

 “better $Q(s,a)$ ”

Q-learning VS SARSA

Update rule (from Bellman eq.)

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1 - \alpha) Q(s_t, a_t)$$

Q-learning

“better $Q(s, a)$ ”



$$\hat{Q}(s, a) = r(s, a) + \gamma \cdot \max_{a'} Q(s', a')$$

Q-learning VS SARSA

Update rule (from Bellman eq.)

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1 - \alpha) Q(s_t, a_t)$$

Q-learning

“better $Q(s, a)$ ”



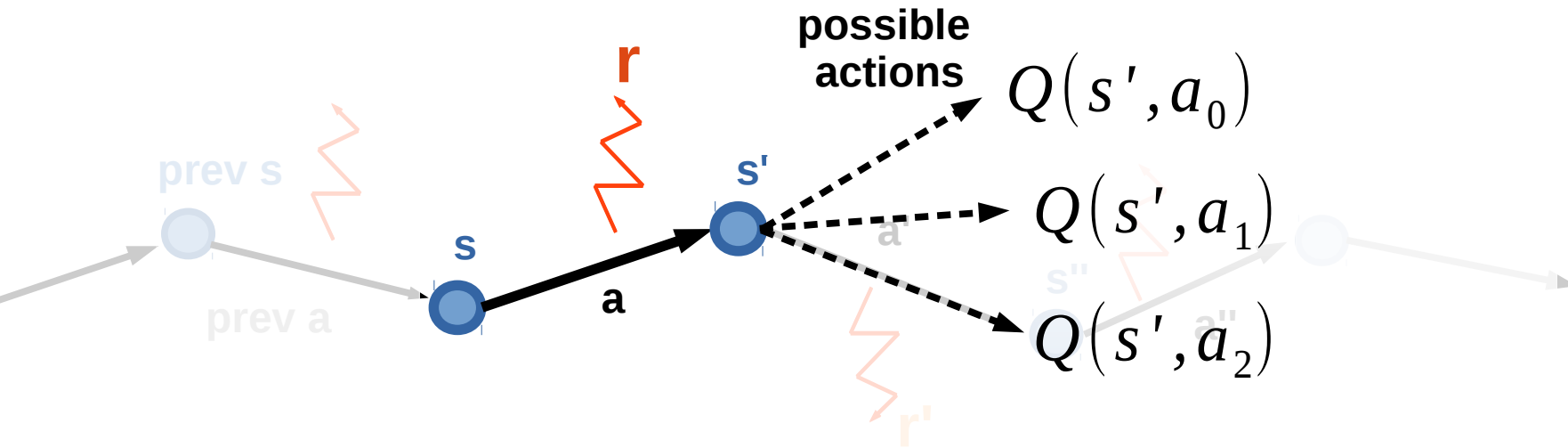
$$\hat{Q}(s, a) = r(s, a) + \gamma \cdot \max_{a'} Q(s', a')$$

EV-

SARSA

$$\hat{Q}(s, a) = r(s, a) + \gamma \cdot E_{a' \sim \pi(a'|s')} Q(s', a')$$

Recap: Q-learning



$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

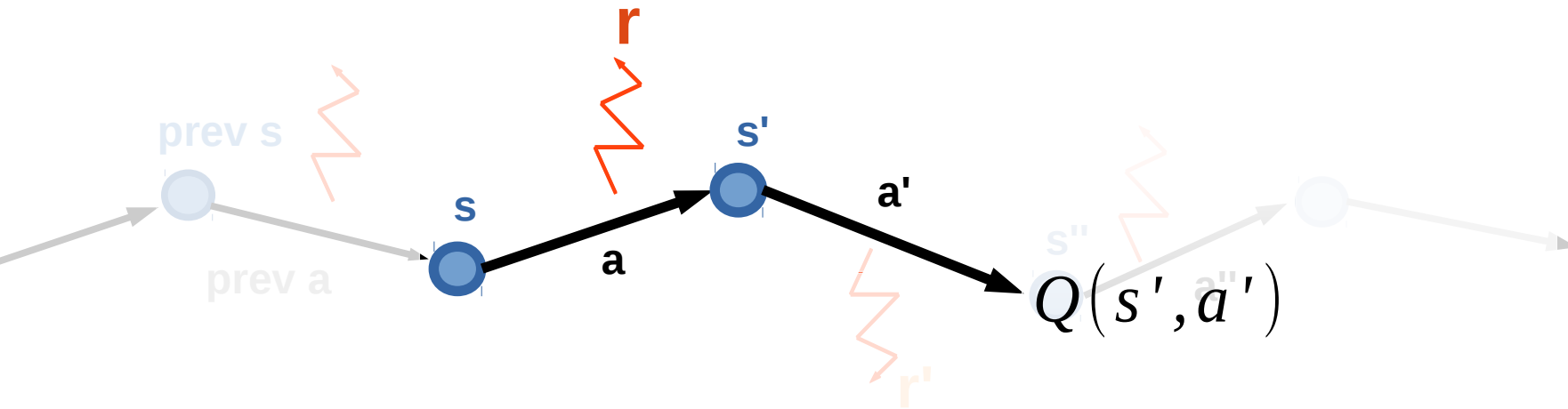
Loop:

- Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$ from env

- Compute $\hat{Q}(s, a) = r(s, a) + \gamma \max_{a_i} Q(s', a_i)$

- Update $Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

SARSA



$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

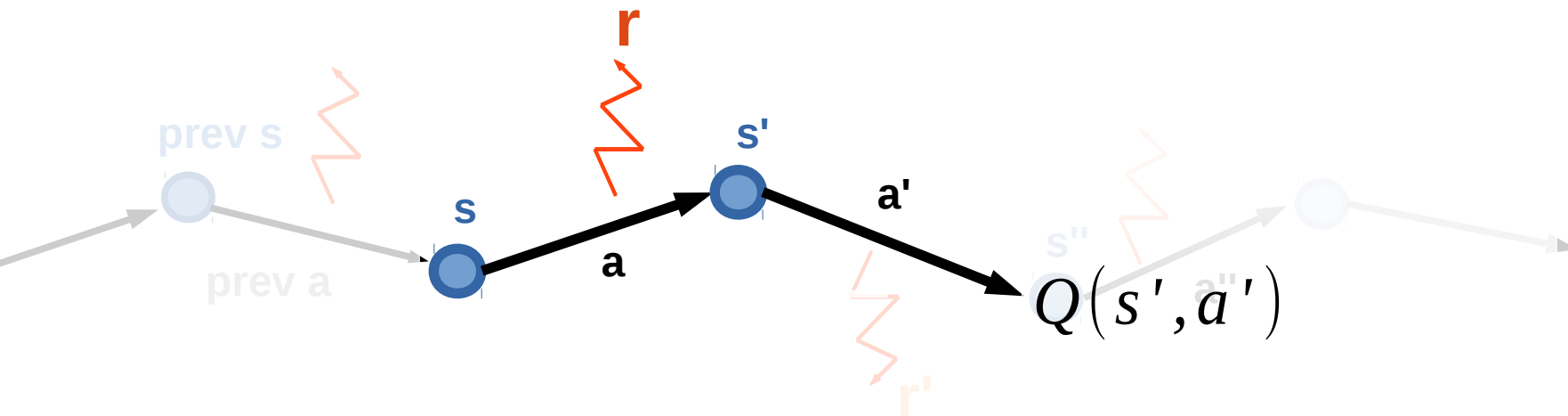
Loop:

- Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}', \mathbf{a}' \rangle$ from env

- Compute $\hat{Q}(s, a) = r(s, a) + \gamma Q(s', a')$

- Update $Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

SARSA



$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

Loop:

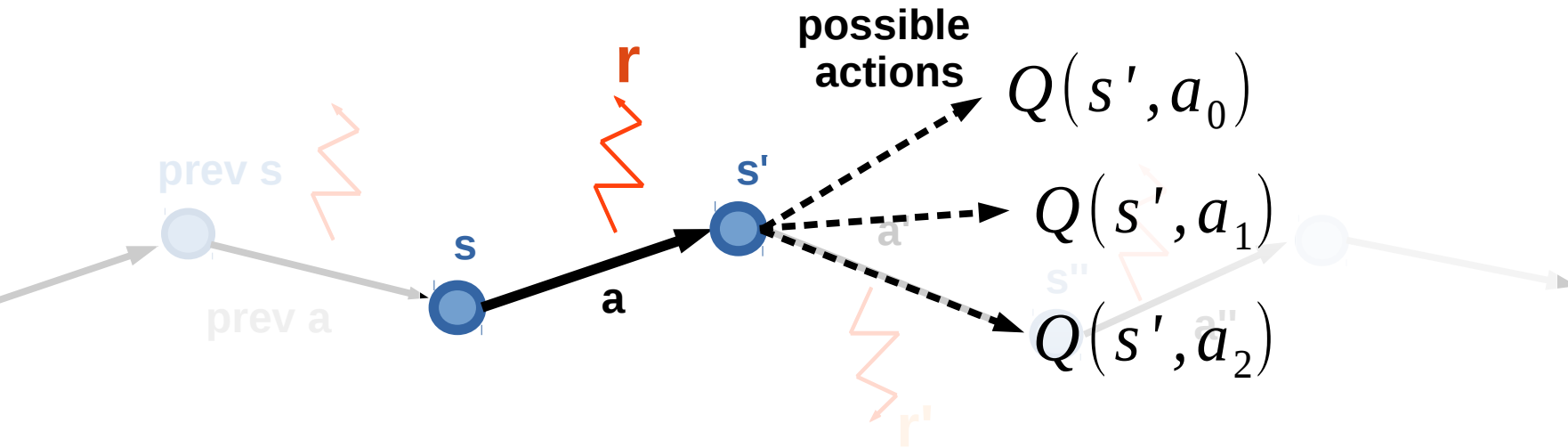
– Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}', \mathbf{a}' \rangle$ from env

hence “SARSA”

– Compute $\hat{Q}(s, a) = r(s, a) + \gamma \underline{Q(s', a')}$ **next action (not max)**

– Update $Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

Expected value SARSA



$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

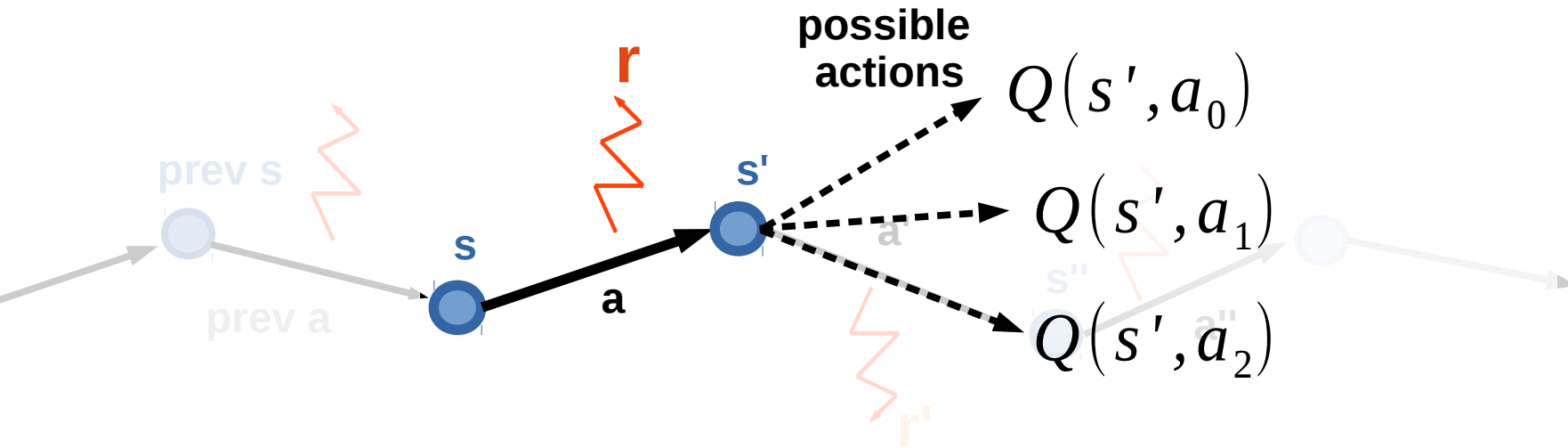
Loop:

- Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$ from env

- Compute $\hat{Q}(s, a) = r(s, a) + \gamma \mathop{E}_{a_i \sim \pi(a|s')} Q(s', a_i)$

- Update $Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

Expected value SARSA



$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

Loop:

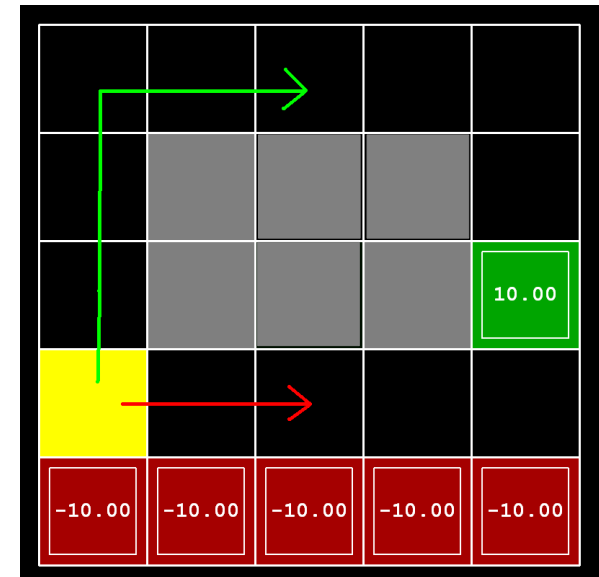
- Sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$ from env

- Compute $\hat{Q}(s, a) = r(s, a) + \gamma \underset{a_i \sim \pi(a|s')}{E} Q(s', a_i)$

- Update $Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

Difference

- SARSA gets optimal rewards under current policy
- Q-learning policy **would be** optimal under



→ Q-learning
→ SARSA

On-policy vs Off-policy

Two problem setups

on-policy

Agent **can** pick actions

- Most obvious setup :)
- Agent always follows his **own** policy

off-policy

Agent **can't** pick actions

- Learning with exploration,
playing without exploration
- Learning from expert
(expert is imperfect)
- Learning from sessions
(recorded data)

On-policy vs Off-policy

Two problem setups

on-policy

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy

off-policy

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy

learn optimal policy even if agent takes random actions

Q: which of Q-learning, SARSA and exp. val. SARSA will **only** work on-policy?

On-policy vs Off-policy

Two problem setups

on-policy

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy
- SARSA
- more later

off-policy

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy
- Q-learning
- Expected Value SARSA

On-policy vs Off-policy

Two problem setups

on-policy

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy
- SARSA
- more coming soon

off-policy

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy
- Q-learning
- Expected Value SARSA

On-policy vs Off-policy

Two problem setups

on-policy

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy
- SARSA
- more coming soon

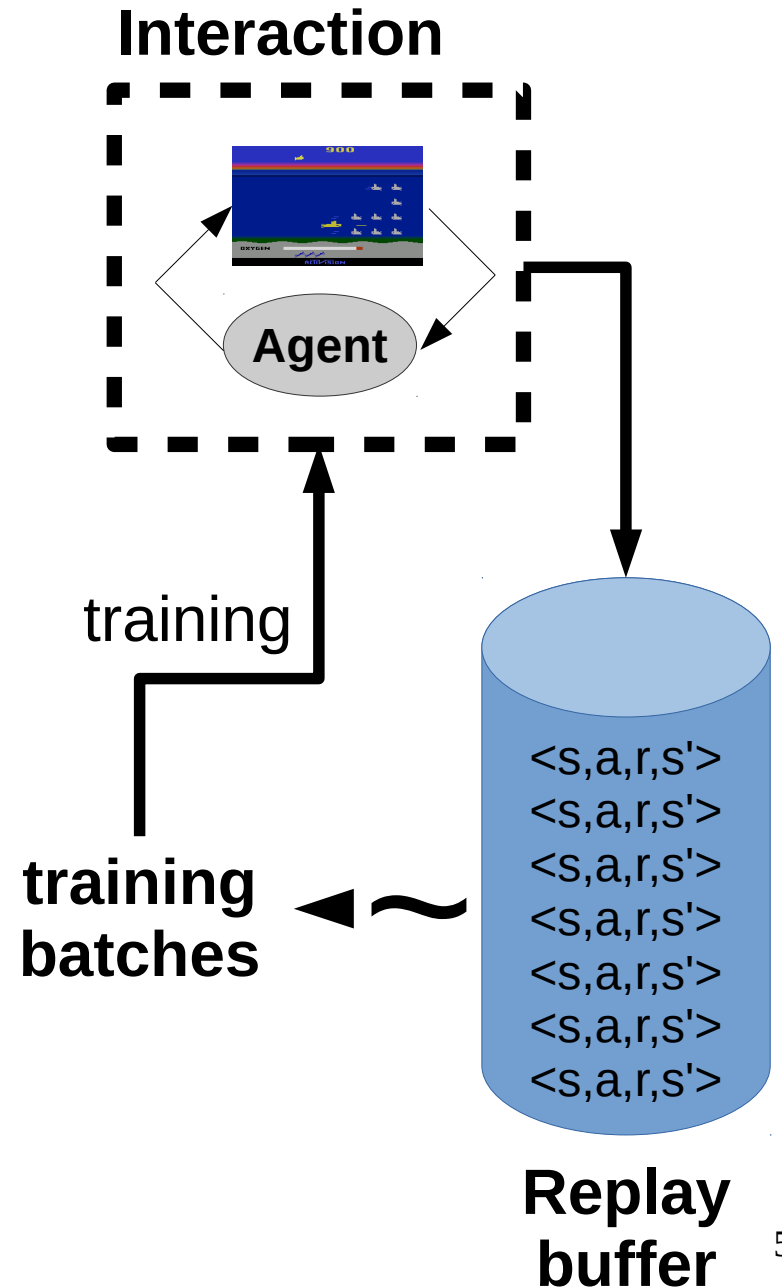
off-policy

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy
- Q-learning
- Expected Value SARSA

Experience replay

Idea: store several past interactions
 $\langle s, a, r, s' \rangle$
Train on random subsamples



Experience replay

Idea: store several past interactions
 $\langle s, a, r, s' \rangle$
Train on random subsamples

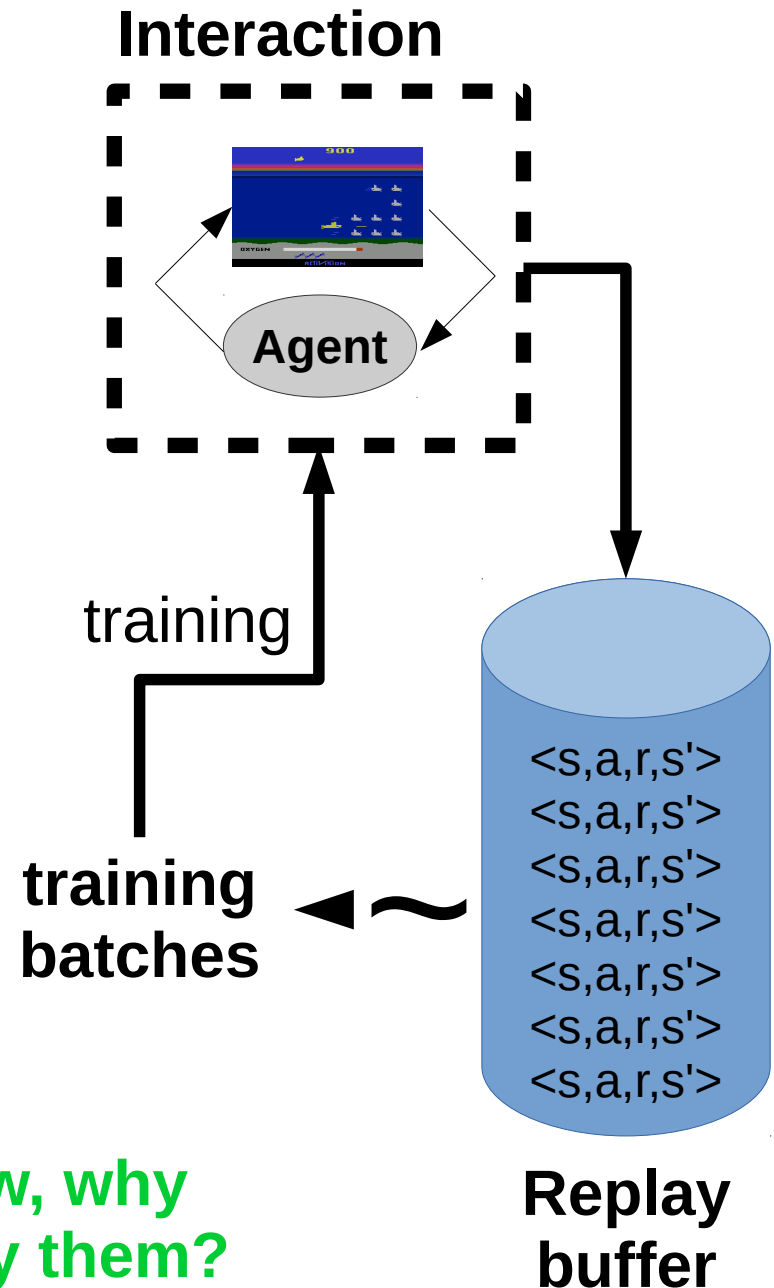
Training curriculum:

- play 1 step and record it
- pick N random transitions to train

Profit: you don't need to re-visit same
(s,a) many times to learn it.

**Only works with
off-policy algorithms!**

**Btw, why
only them?**



Experience replay

</chapter>

Idea: store several past interactions

$\langle s, a, r, s' \rangle$

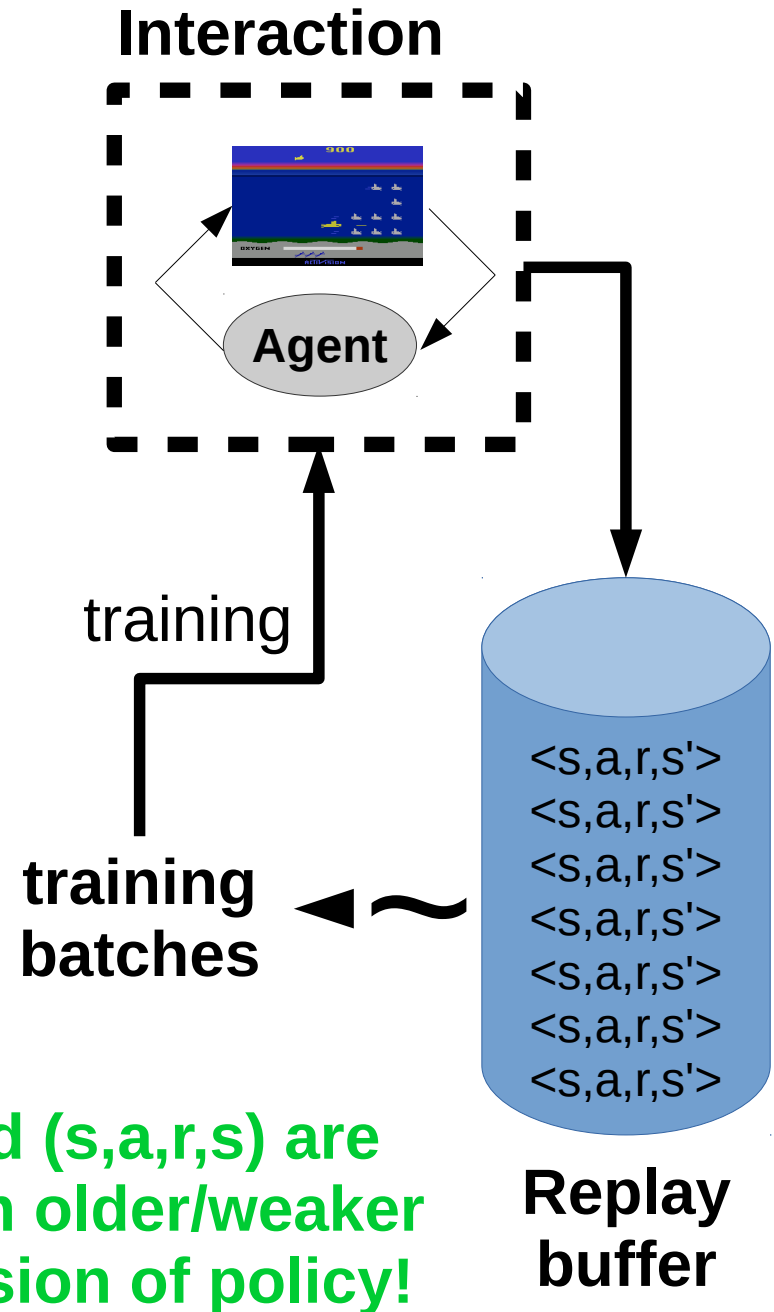
Train on random subsamples

Training curriculum:

- play 1 step and record it
- pick N random transitions to train

Profit: you don't need to re-visit same (s, a) many times to learn it.

**Only works with
off-policy algorithms!**



New stuff we learned

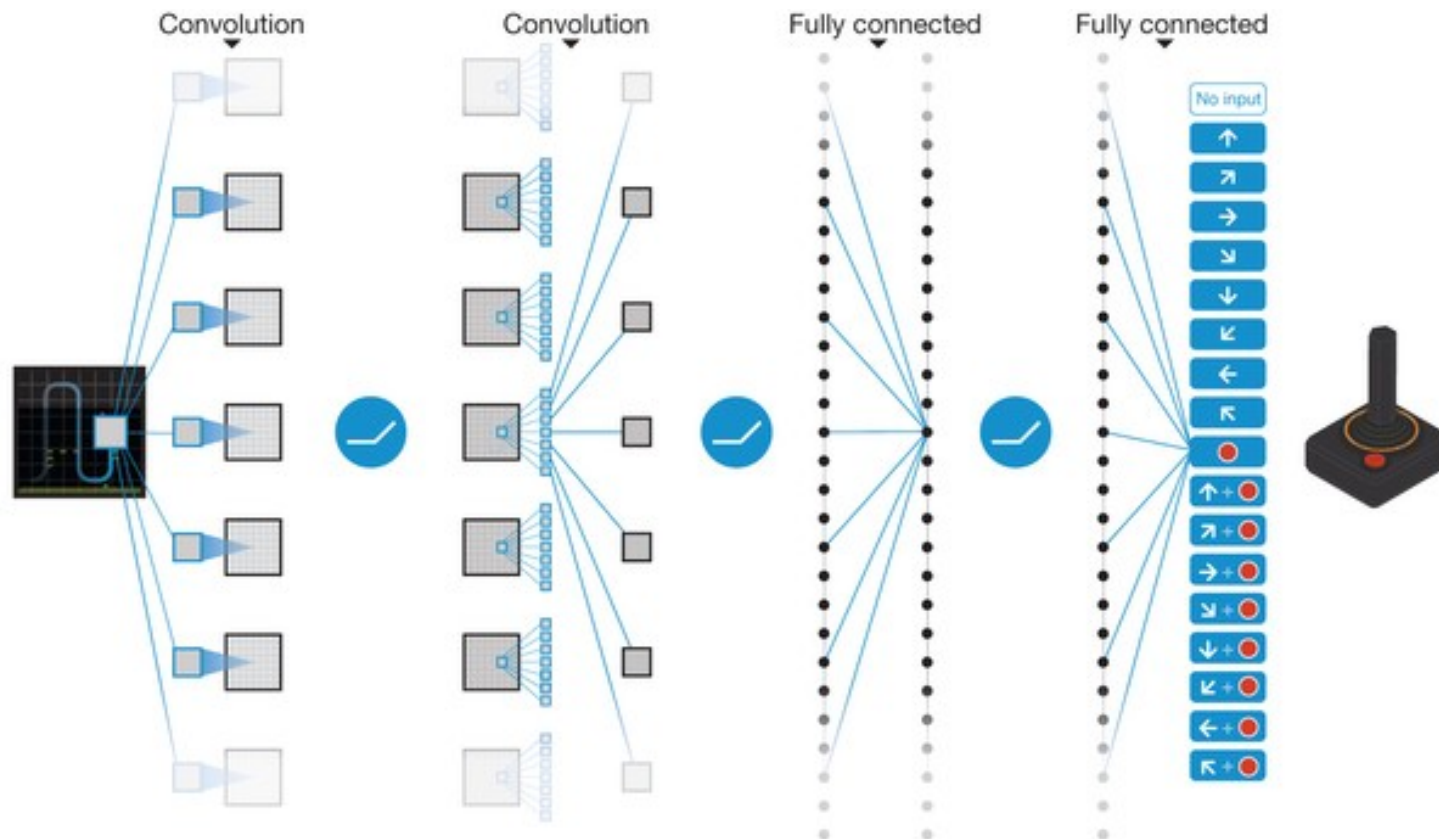
- Anything?

New stuff we learned

- $Q(s,a), Q^*(s,a)$
- Q-learning, SARSA
 - We can learn from trajectories (model-free)
- Exploration vs exploitation (basics)
- Learning On-policy vs Off-policy
 - Using experience replay

Coming next...

- What if state space is large/continuous
 - Deep reinforcement learning

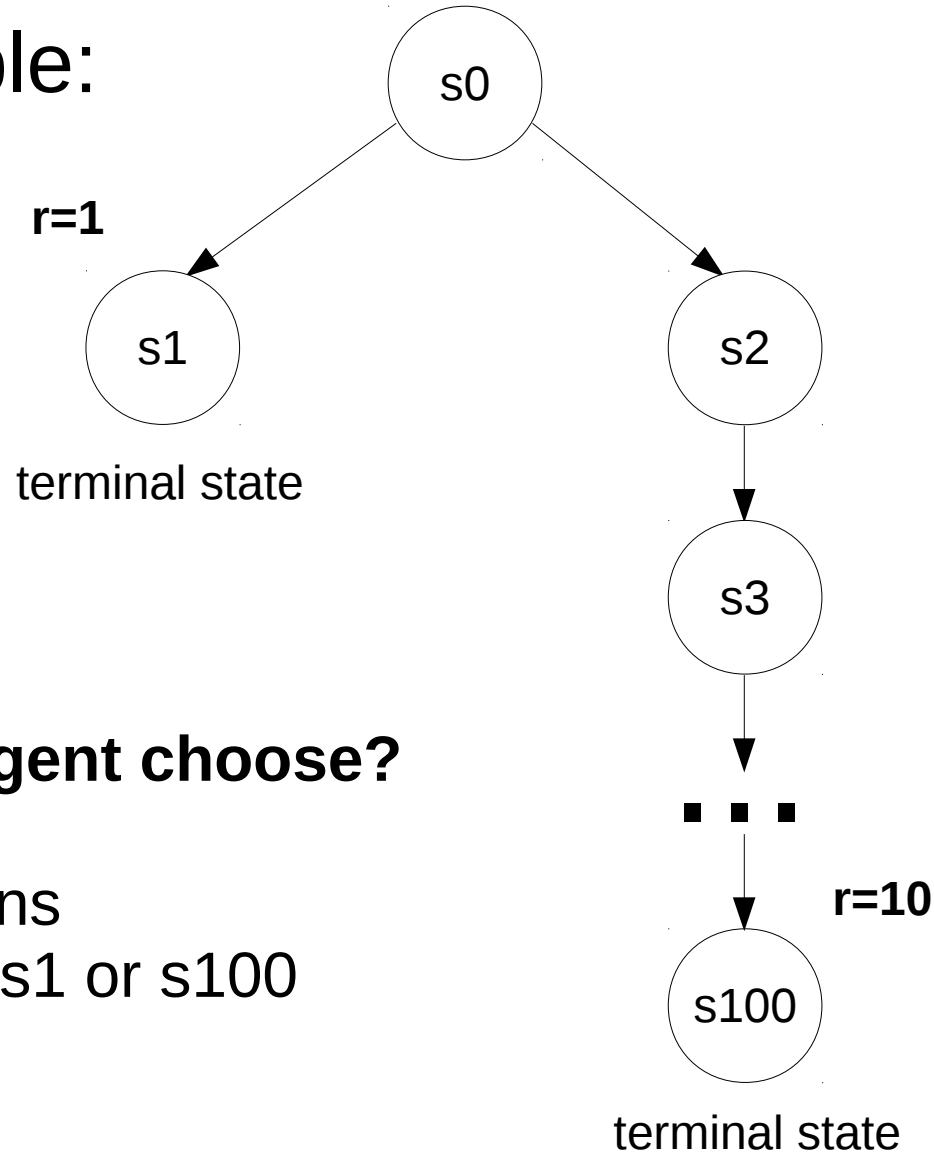


- Remember what $Q(s, a)$ and $V(s)$ functions do
- Remember both about exploration and exploitation
 - At least using greedy policy or softmax smoothing
- Remember the difference between on-policy and off-policy algorithms!
 - On-policy algorithms **can't** learn off-policy (e.g. SARSA)
 - Off-policy algorithms **can** learn on-policy (e.g. Q-learning)
- Experience replay: no need to re-visit same (s,a) many times to learn it.
 - Works only with off-policy algorithms

Remember discounted rewards?

Discounted reward **fails** #1

Trivial example:

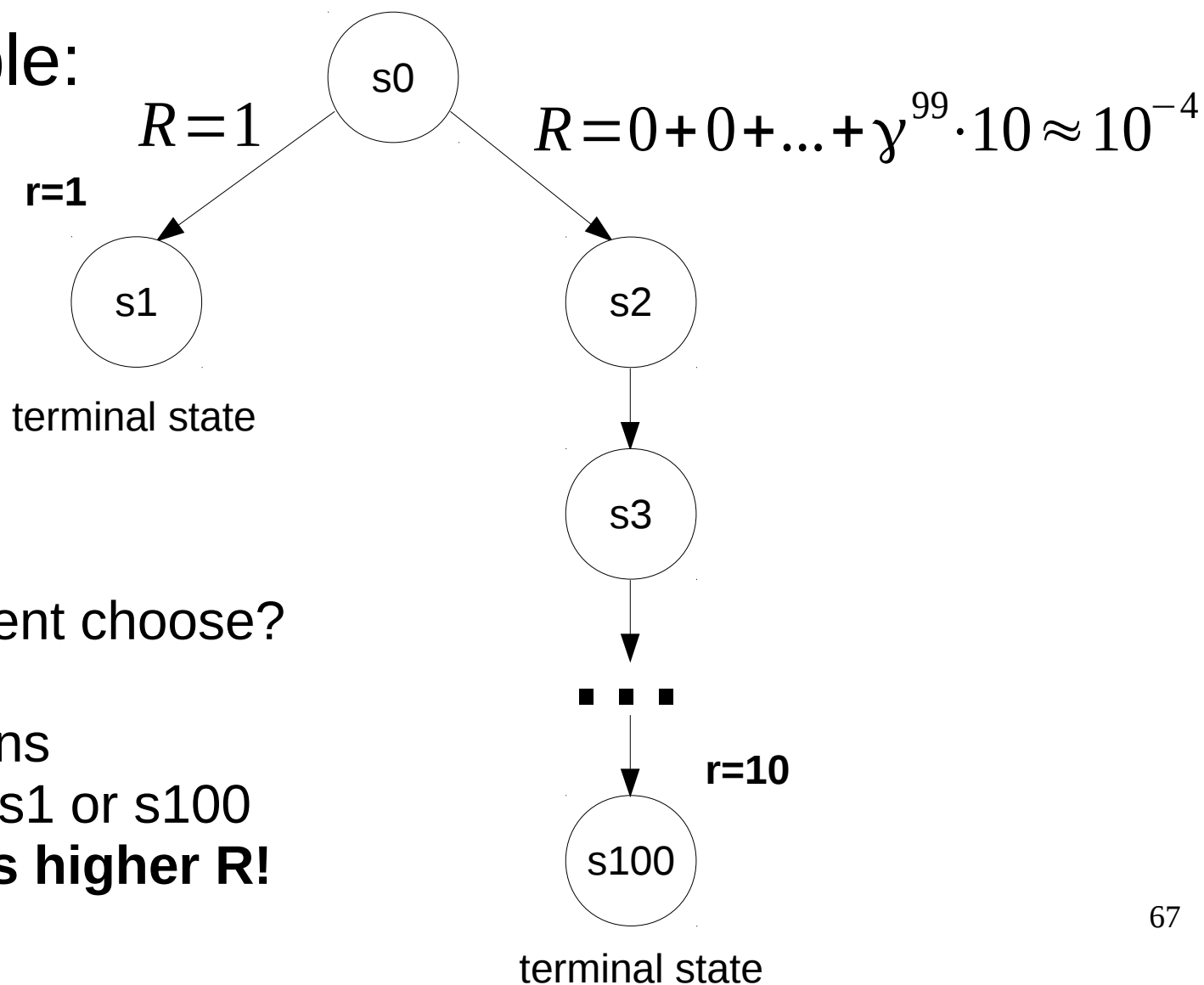


What path will agent choose?

- $\gamma=0.9$
- arrows = actions
- game ends at s1 or s100

Discounted reward **fails** #1

Trivial example:



What path will agent choose?

- $\gamma=0.9$
- arrows = actions
- game ends at s_1 or s_{100}
- **left action has higher R!**

Discounted reward **fails** #2

Deephack'17 qualification round, Atari Skiing



- You steer the red guy
- Session lasts ~5k steps
- You get -3~-7 reward each tick (faster game = better score)
- At the end of session, you get up to $r=-30k$ (based on passing gates, etc.)
- Q-learning with $\gamma=0.99$ fails it doesn't learn to pass gates

What's the problem?

Discounted reward **fails** #2

Deephack'17 qualification round, Atari Skiing



- You steer the red guy
- Session lasts ~5k steps
- You get -3~-7 reward each tick (faster game = better score)
- At the end of session, you get up to $r=-30k$ (based on passing gates, etc.)
- Q-learning with $\gamma=0.99$ fails

Discounted reward **fails** #3

CoastRunner7 experiment (openAI)



- You control the boat
- Rewards for getting to checkpoints
- Rewards for collecting bonuses
- What could possibly go wrong?
- “Optimal” policy video:
<https://www.youtube.com/watch?v=tlOIHko8ySg>

Nuts and bolts: MC vs TD

Monte-carlo

- Ignores intermediate rewards
doesn't need γ (discount)
- Needs full episode to learn
Infinite MDP are a problem
- Doesn't use Markov property
Works with non-markov envs

Temporal Difference

- Uses intermediate rewards
trains faster under right γ
- Learns from incomplete episode
Works with infinite MDP
- Requires markov property
Non-markov env is a problem



Nuts and bolts: discount

- Effective horizon $1 + \gamma + \gamma^2 + \dots = \frac{1}{(1 - \gamma)}$

Heuristic: your agent stops giving a damn in *this many* turns.

Typical values:

- $\gamma=0.9$, 10 turns
- $\gamma=0.95$, 20 turns
- $\gamma=0.99$, 100 turns
- $\gamma=1$, infinitely long

Higher γ = less stable algorithm.

$\gamma=1$ only works for episodic MDP (finite amount of turns).

Nuts and bolts: discount

- Effective horizon $1 + \gamma + \gamma^2 + \dots = \frac{1}{(1 - \gamma)}$

Heuristic: your agent stops giving a damn in *this many* turns.

- Atari Skiing, reward was delayed by in 5k steps
- $\gamma=0.99$ is not enough
- $\gamma=1$ and a few hacks works better
- Or use a better reward function

