Machine Learning course
advanced track

Lecture 8: Model free learning

Radoslav Neychev

MIPT
25.10.2019, Moscow, Russia

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

Outline

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

Previously...

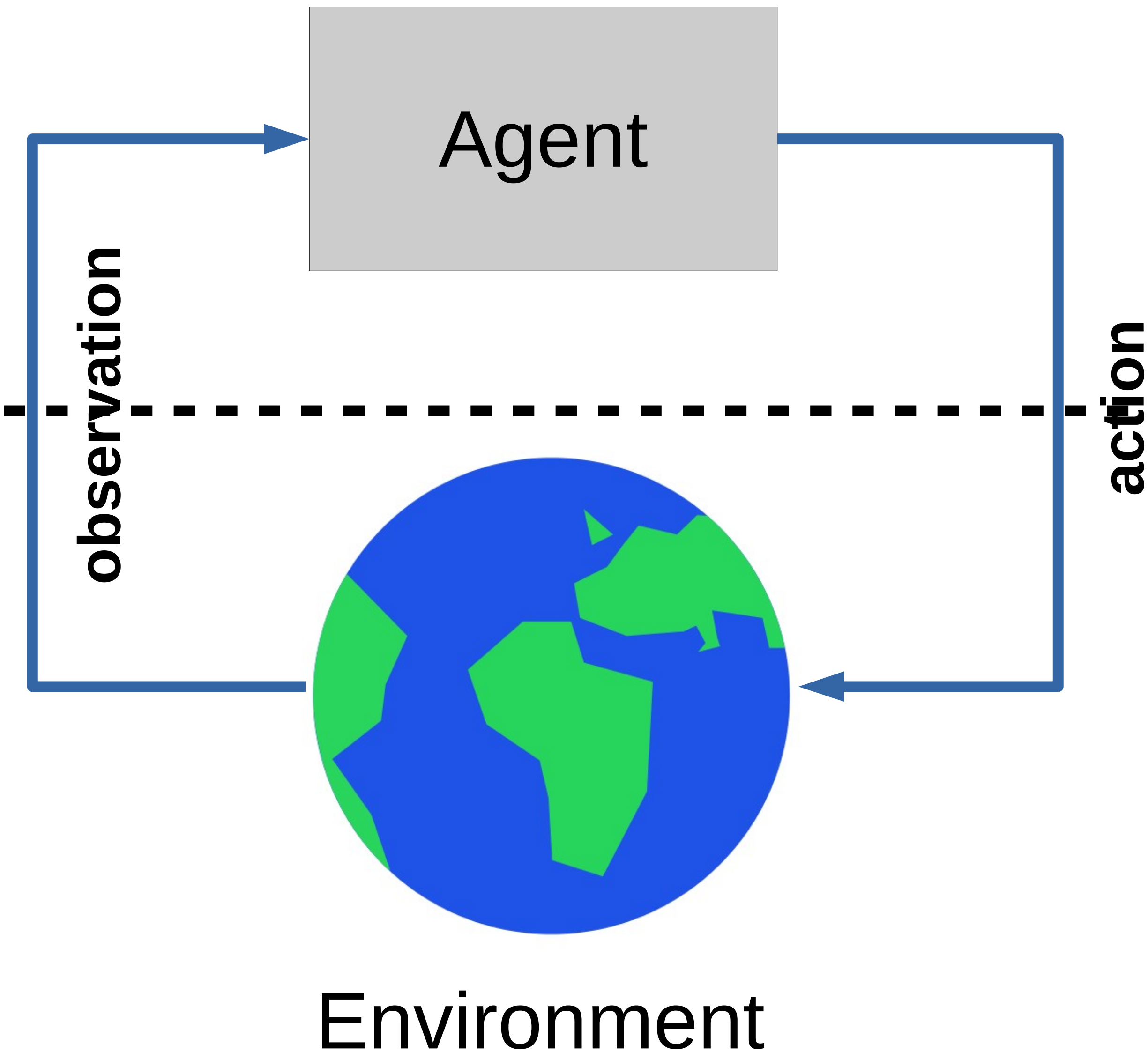
- **V(s)** and **V*(s,a)**
- know V* and P(s'|s,a) → know optimal policy
- We can learn V* with dynamic programming

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

Decision process in the wild

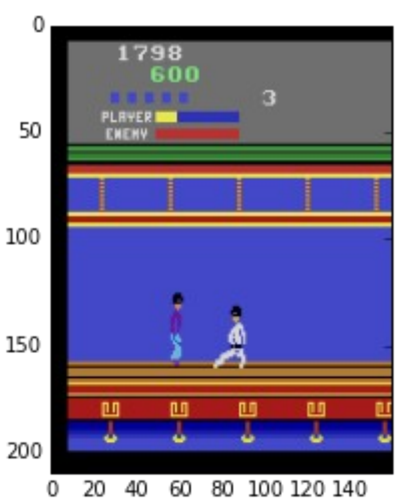
Can do anything

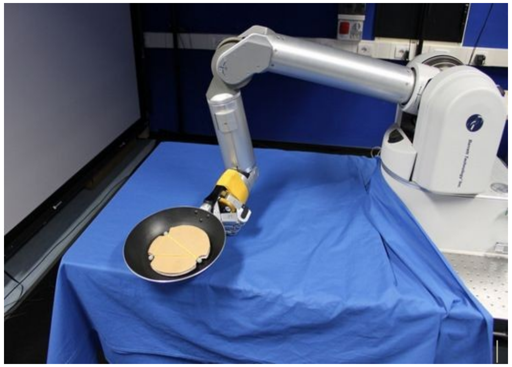
Can't even see
(worst case)



Decision process in the wild

Can do anything





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_π(s)** – expected G from state **s** if you follow **π**
- **V*(s)** – expected G from state s if you follow **π***

More new letters

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

- $\mathbf{Q_\pi(s,a)}$ – expected G from state **s**
  - if you start by taking action **a**
  - and follow $\boldsymbol{\pi}$ from next state on

- $\mathbf{Q^*(s,a)}$ – guess what it is :)

More new letters

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

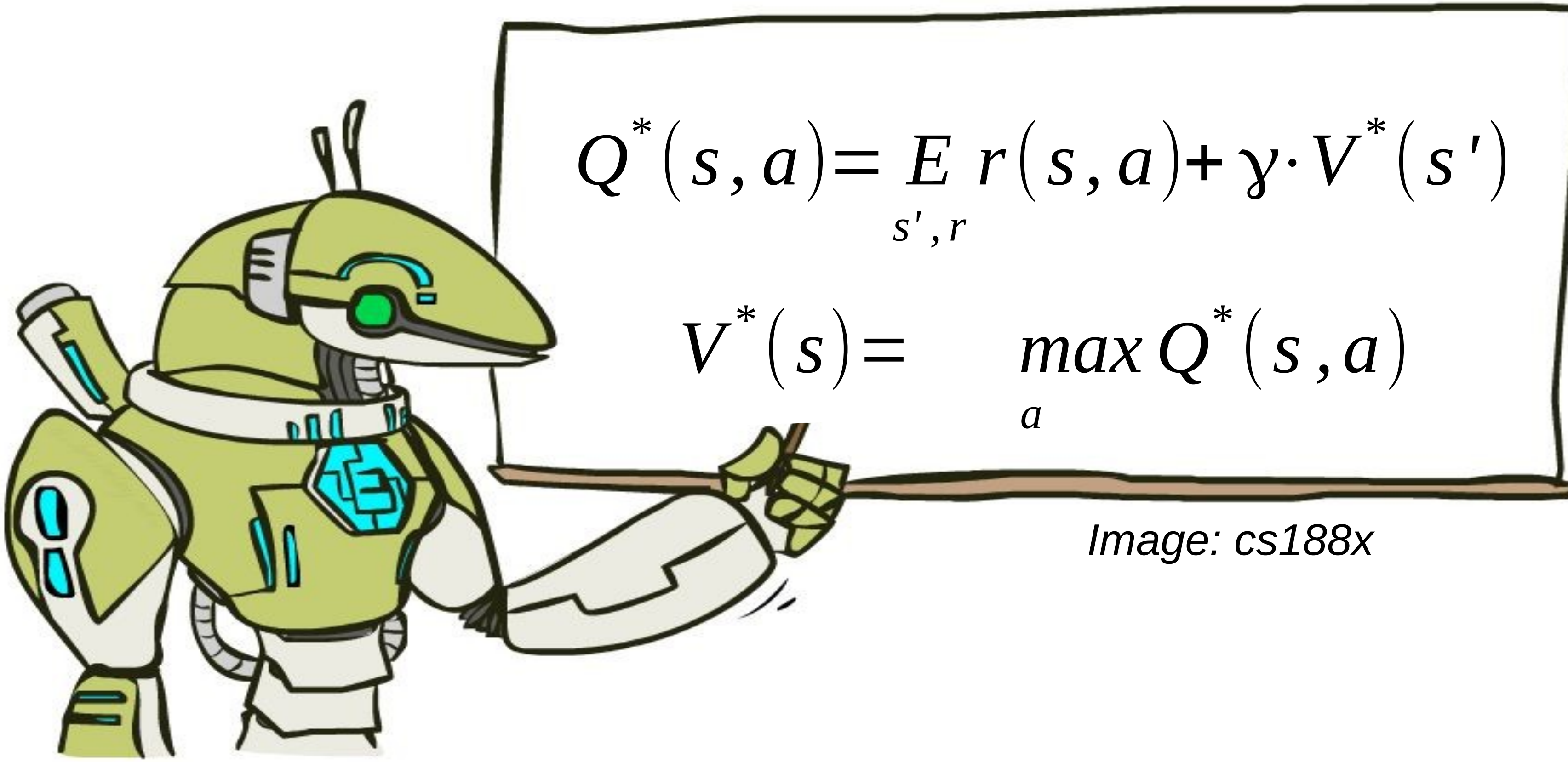
- $\mathbf{Q_\pi(s,a)}$ – expected G from state **s**
  - if you start by taking action **a**
  - and follow $\boldsymbol{\pi}$ from next state on

- $\mathbf{Q^*(s,a)}$ – same as $\mathbf{Q_\pi(s,a)}$ where $\boldsymbol{\pi = \pi^*}$

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



$$Q^*(s,a) = \mathop{E}_{s',r} r(s,a) + \gamma \cdot V^*(s')$$

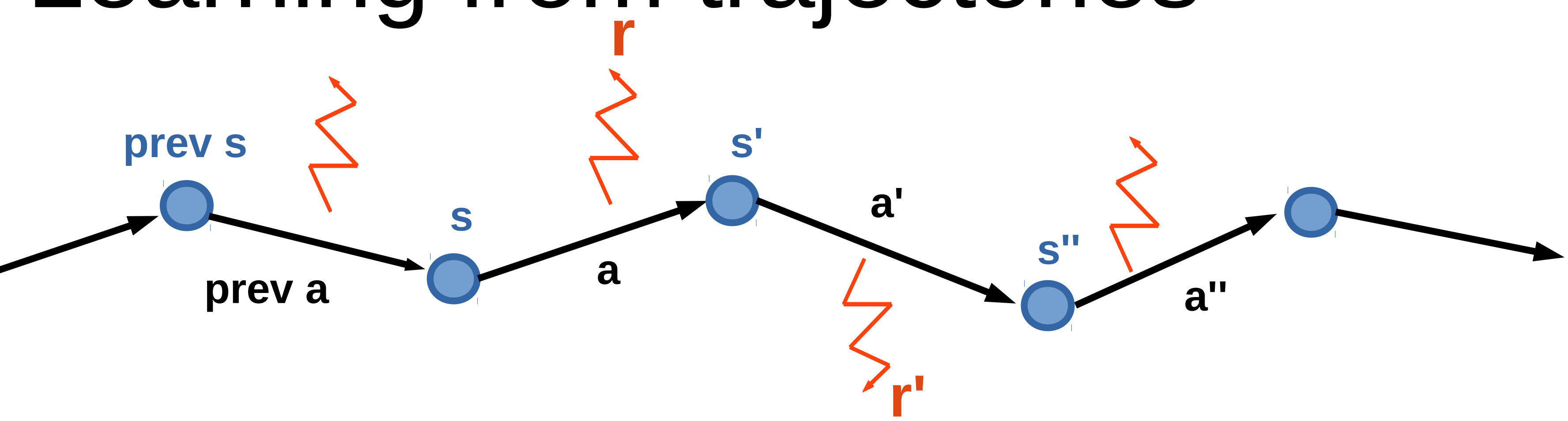
$$V^*(s) = \max_a Q^*(s,a)$$

Image: cs188x

Action value Q_π(s,a) is the expected total reward **G** agent gets from state **s** by taking action **a** and following policy **π** from next state.

$$\pi(s) : argmax_a Q(s,a)$$

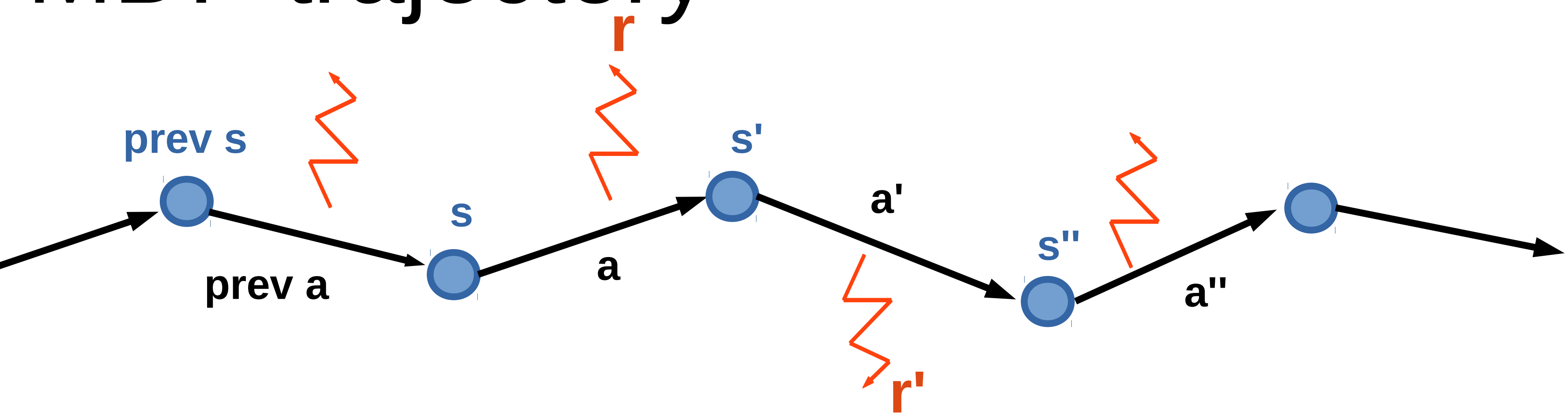
Learning from trajectories



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

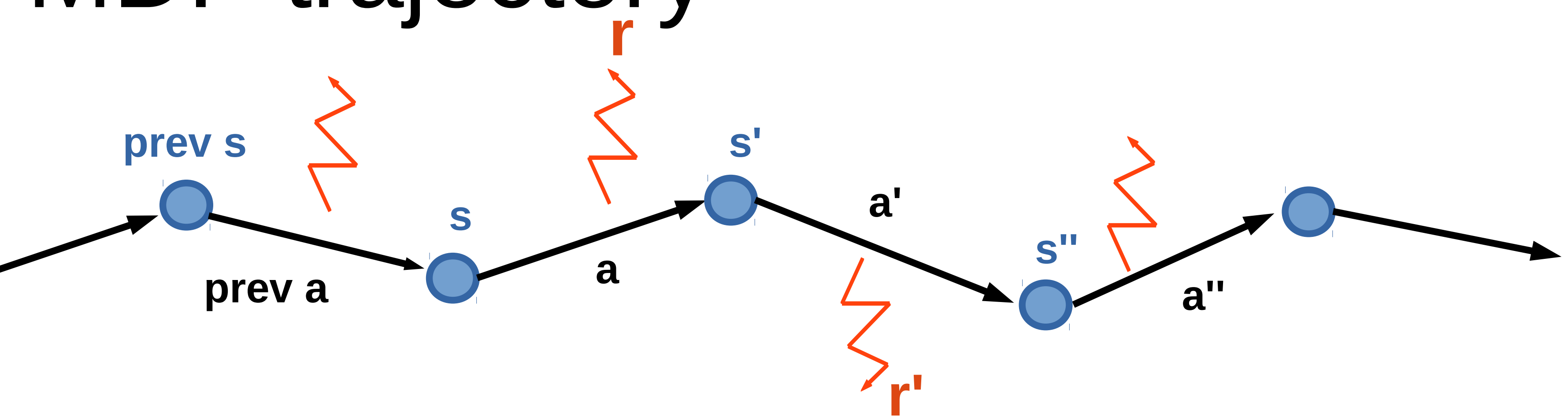
MDP trajectory



- Trajectory is a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

- We can only sample trajectories

MDP trajectory

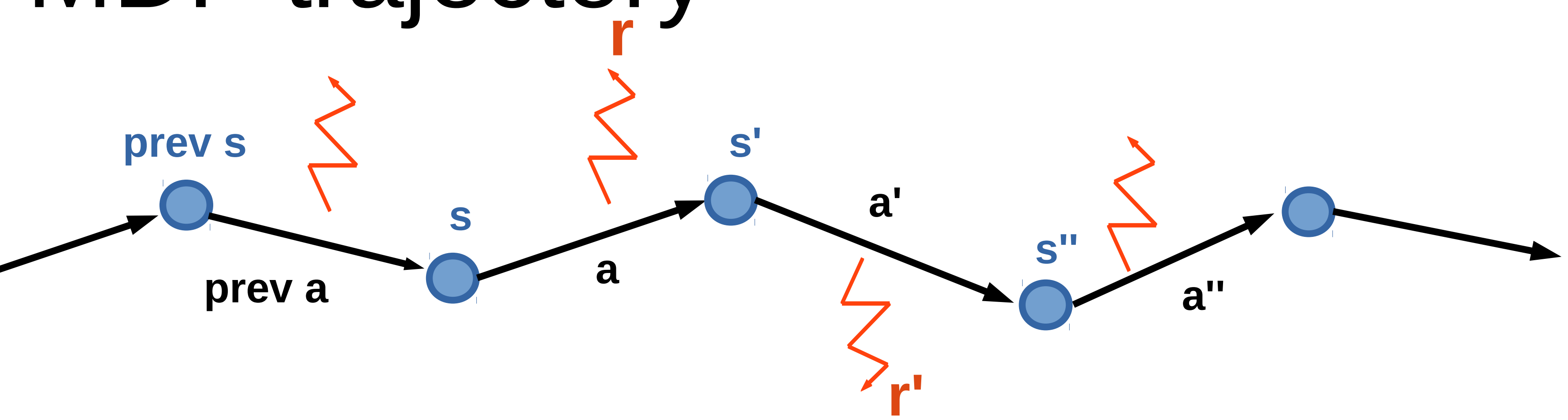


- Trajectory is a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

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)

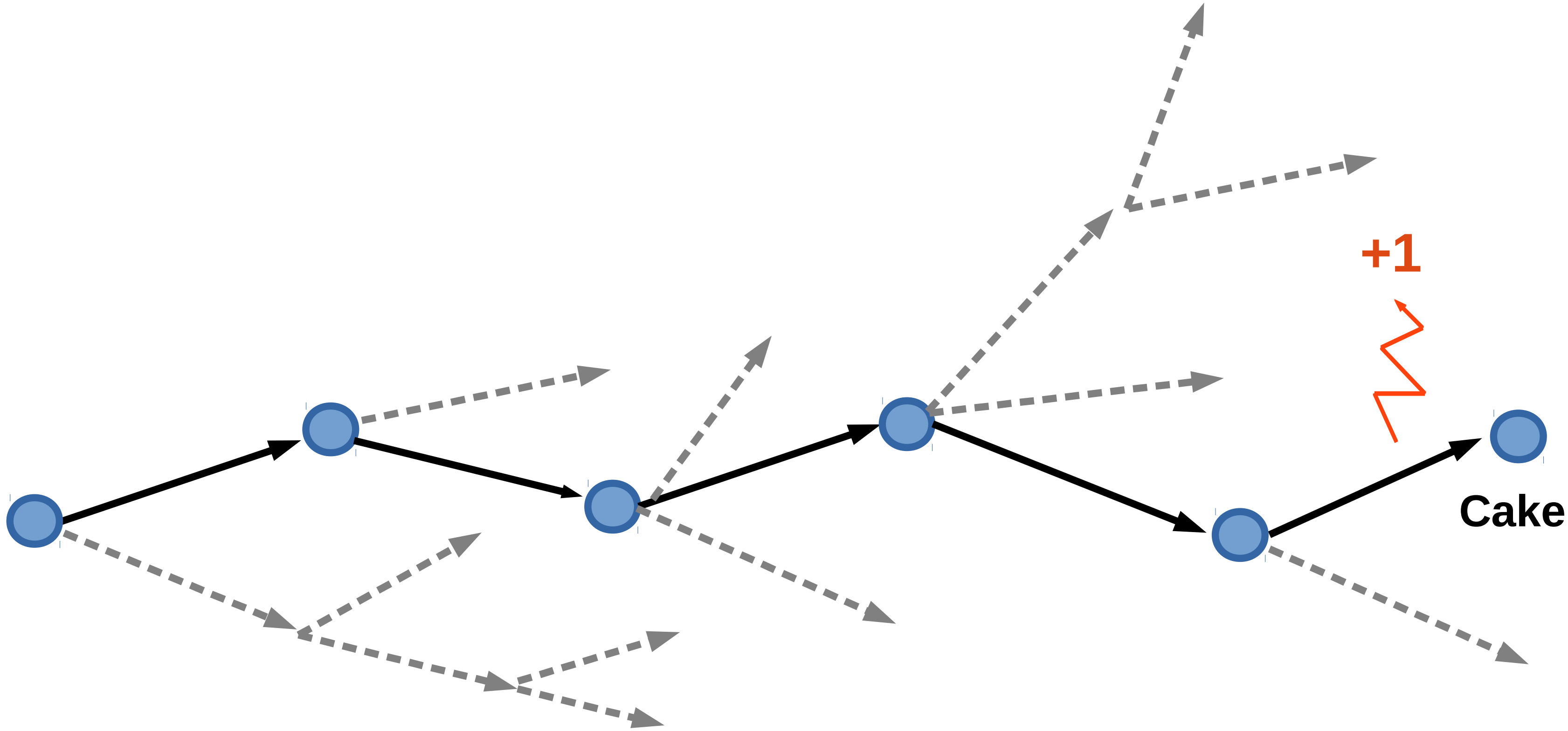
Q: What to learn?
V(s) or Q(s,a)

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!

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

Idea 2: temporal difference

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

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

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!

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

That's Q*(s,a)

That's something we don't have

What do we do?

That's value for π*
aka optimal policy

Idea 2: temporal difference

Idea 2: temporal difference

- Replace expectation with sampling

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

Idea 2: temporal difference

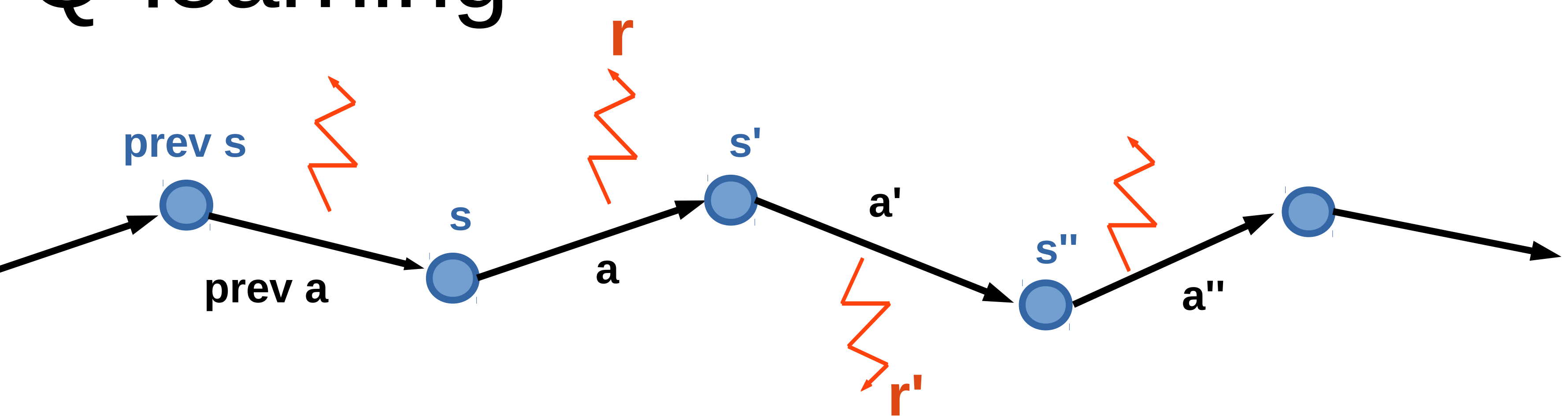
- Replace expectation with sampling

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

- Use moving average with just one sample!

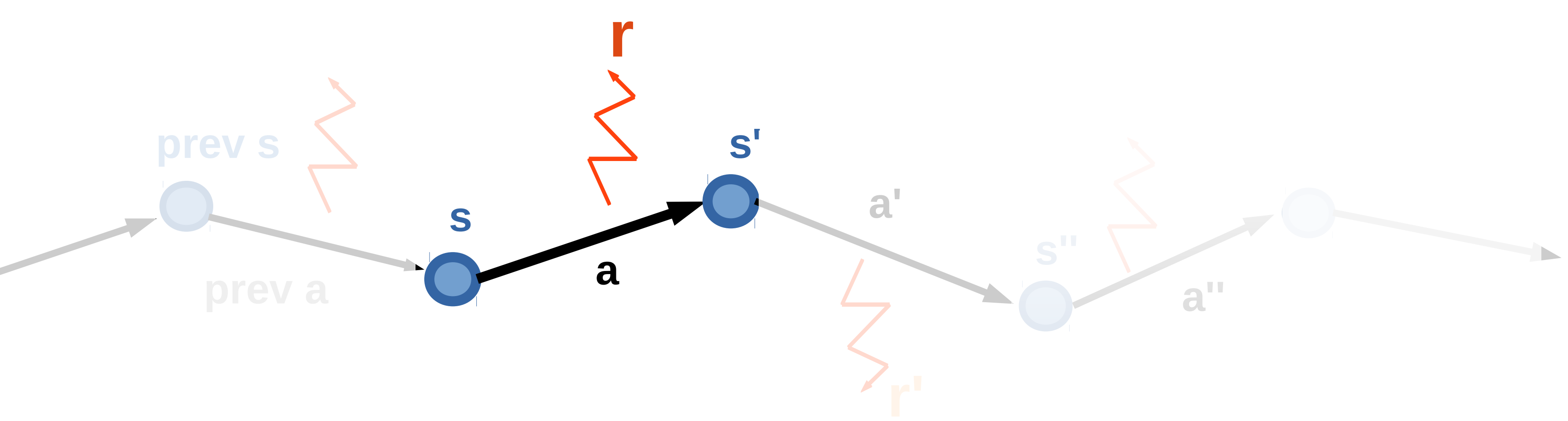
$$Q(s_t, a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a')) + (1 - \alpha) Q(s_t, a_t)$$

Q-learning



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

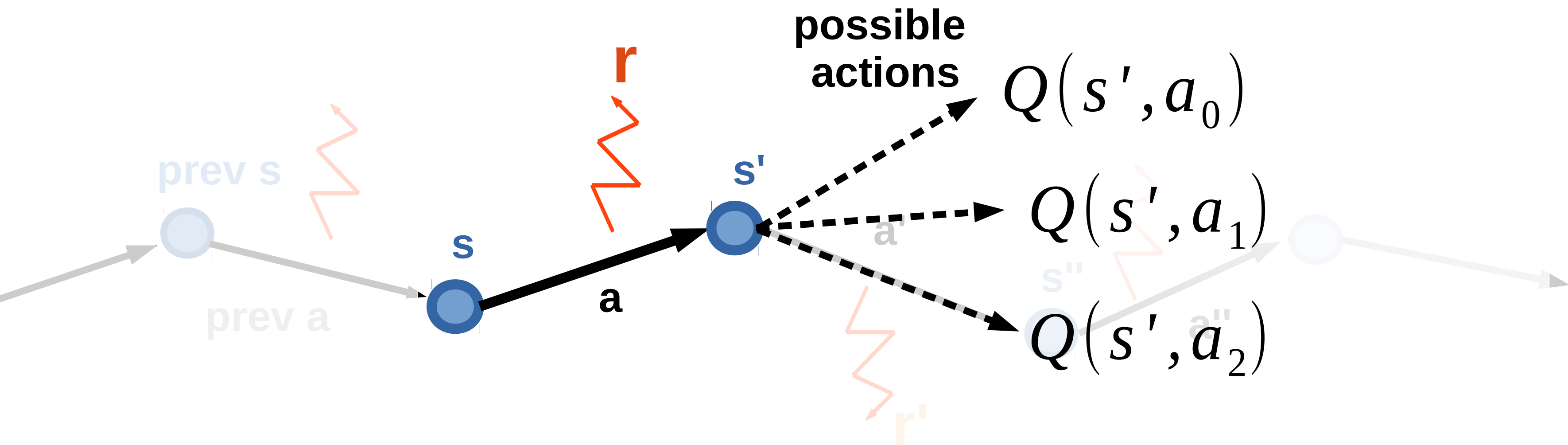
Q-learning



Initialize Q(s,a) with zeros

- Loop:
  - Sample <**s**,**a**,**r**,**s'**> from env

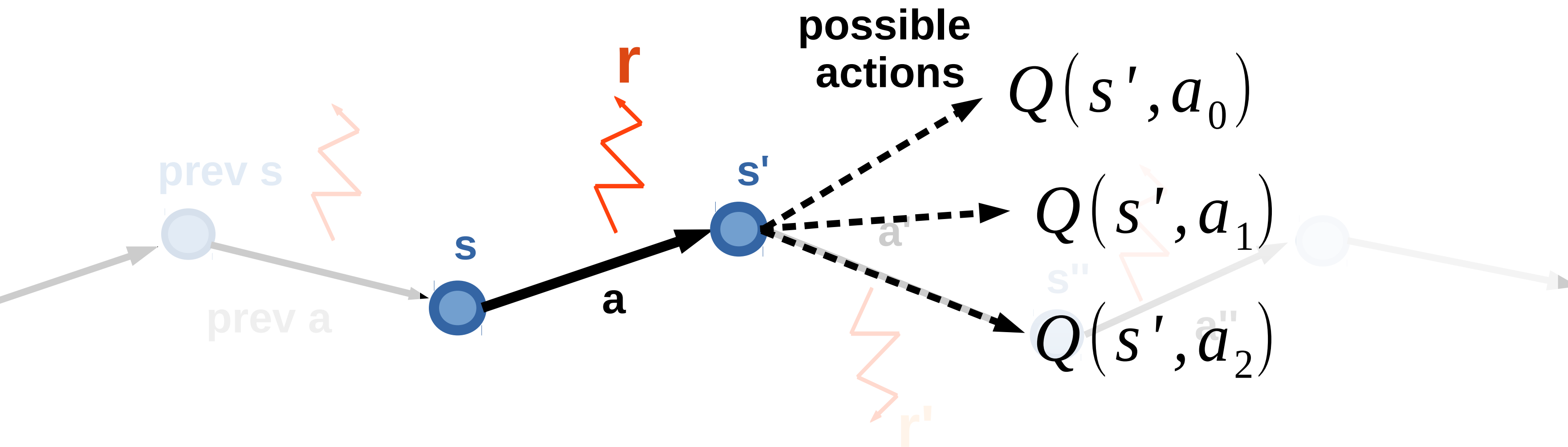
Q-learning



Initialize Q(s,a) with zeros

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

Q-learning



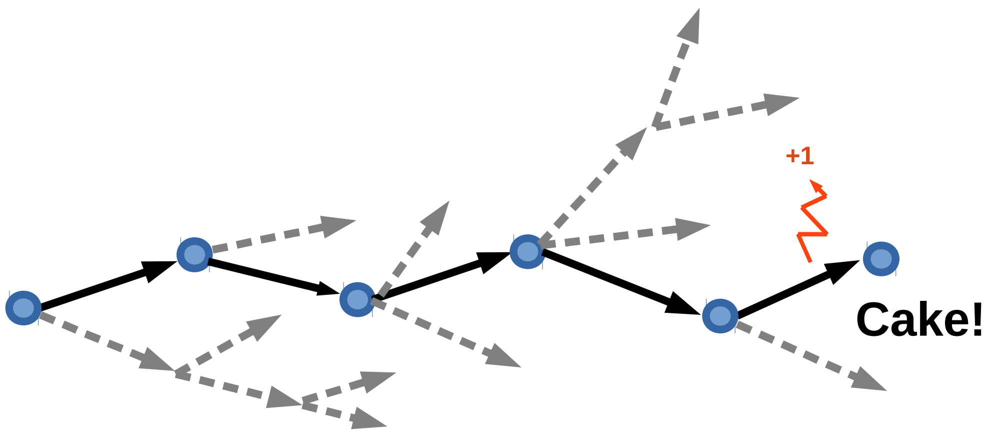
Initialize Q(s,a) with zeros

- Loop:
  - Sample <**s**,**a**,**r**,**s'**> 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

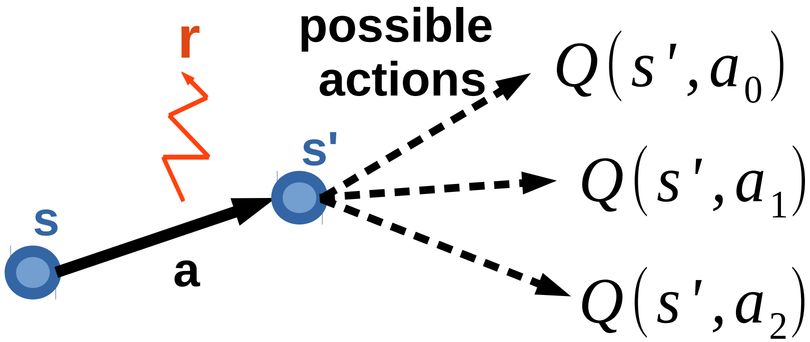
Monte-carlo

- Averages Q over sampled paths



Temporal Difference

- Uses recurrent formula for Q



Nuts and bolts: MC vs TD

Monte-carlo

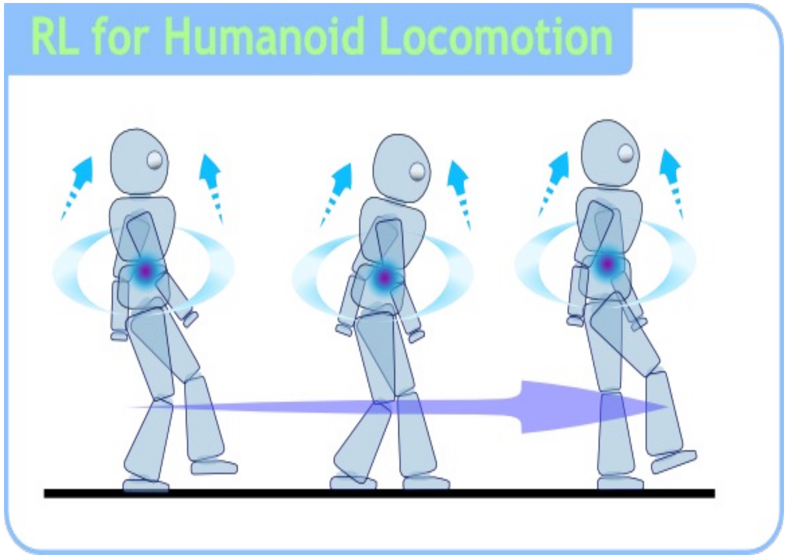
- Averages Q over sampled paths
- Needs full trajectory to learn
- Less reliant on markov property

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.

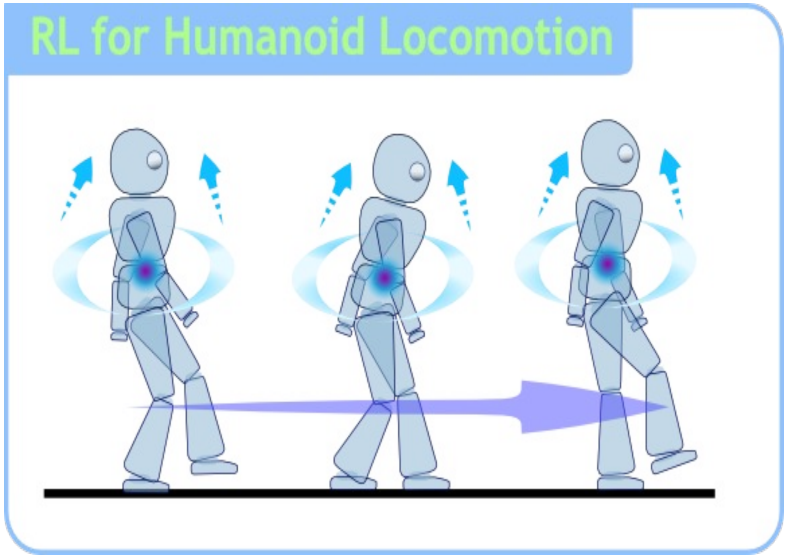


Initial Q(s,a) are zeros
robot uses argmax Q(s,a)

He has just learned to crawl with positive reward!

What could possibly go wrong?

Our mobile robot learns to walk.



Initial Q(s,a) are zeros
robot uses argmax Q(s,a)

Too bad, now he will never learn to walk upright =(

What could possibly go wrong?

New problem:

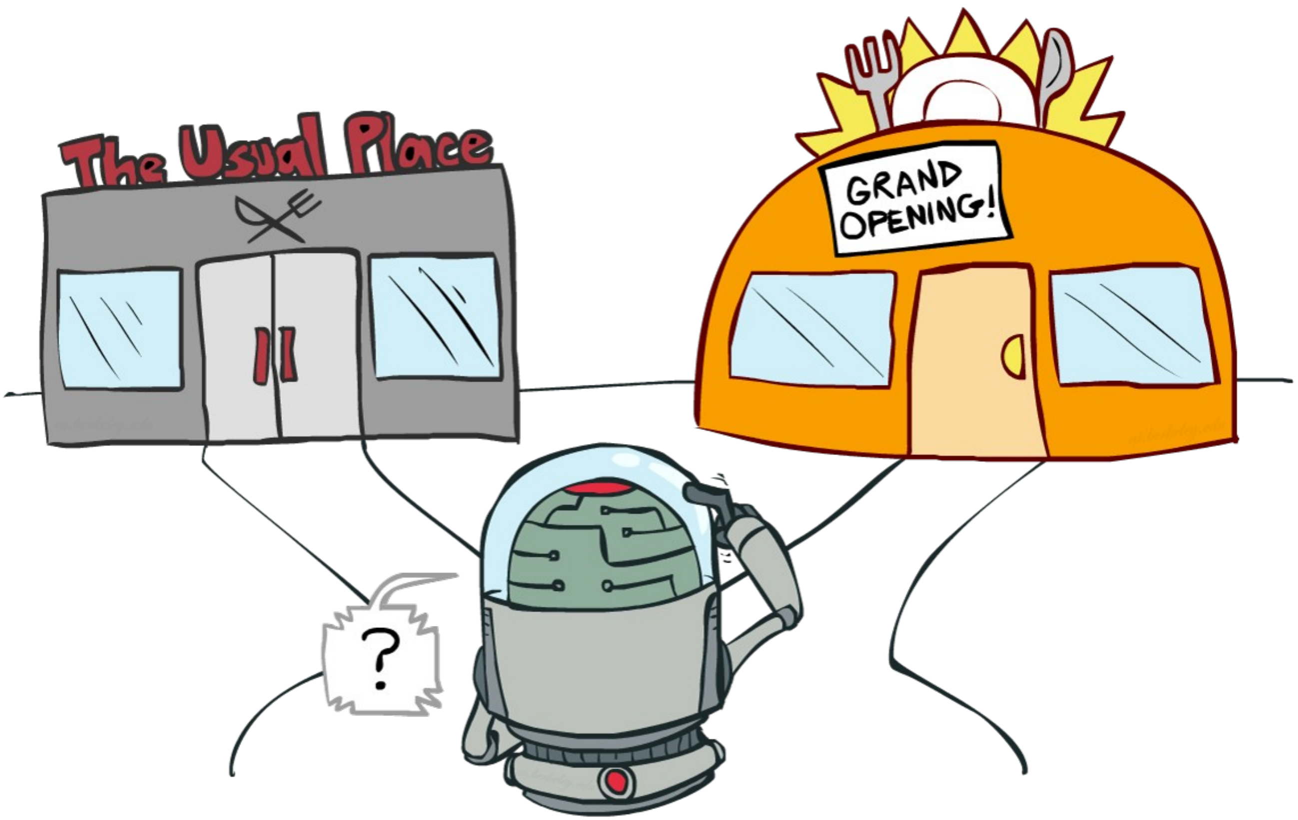
If our agent always takes “best” actions
from his current point of view,

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.

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

- More cool stuff coming later

Exploration over time

Idea:

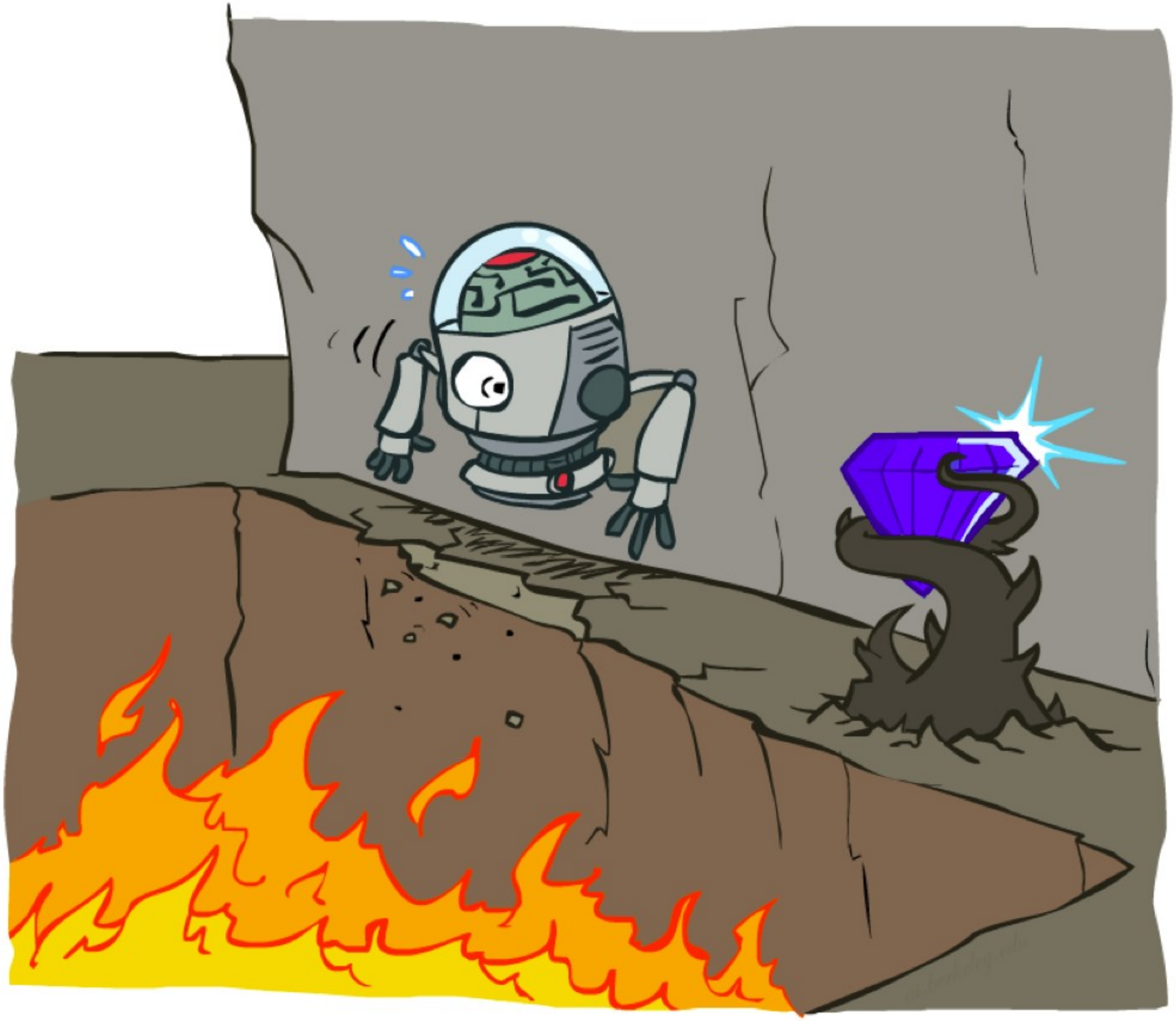
If you want to converge to optimal policy,
you need to gradually reduce exploration

Example:

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

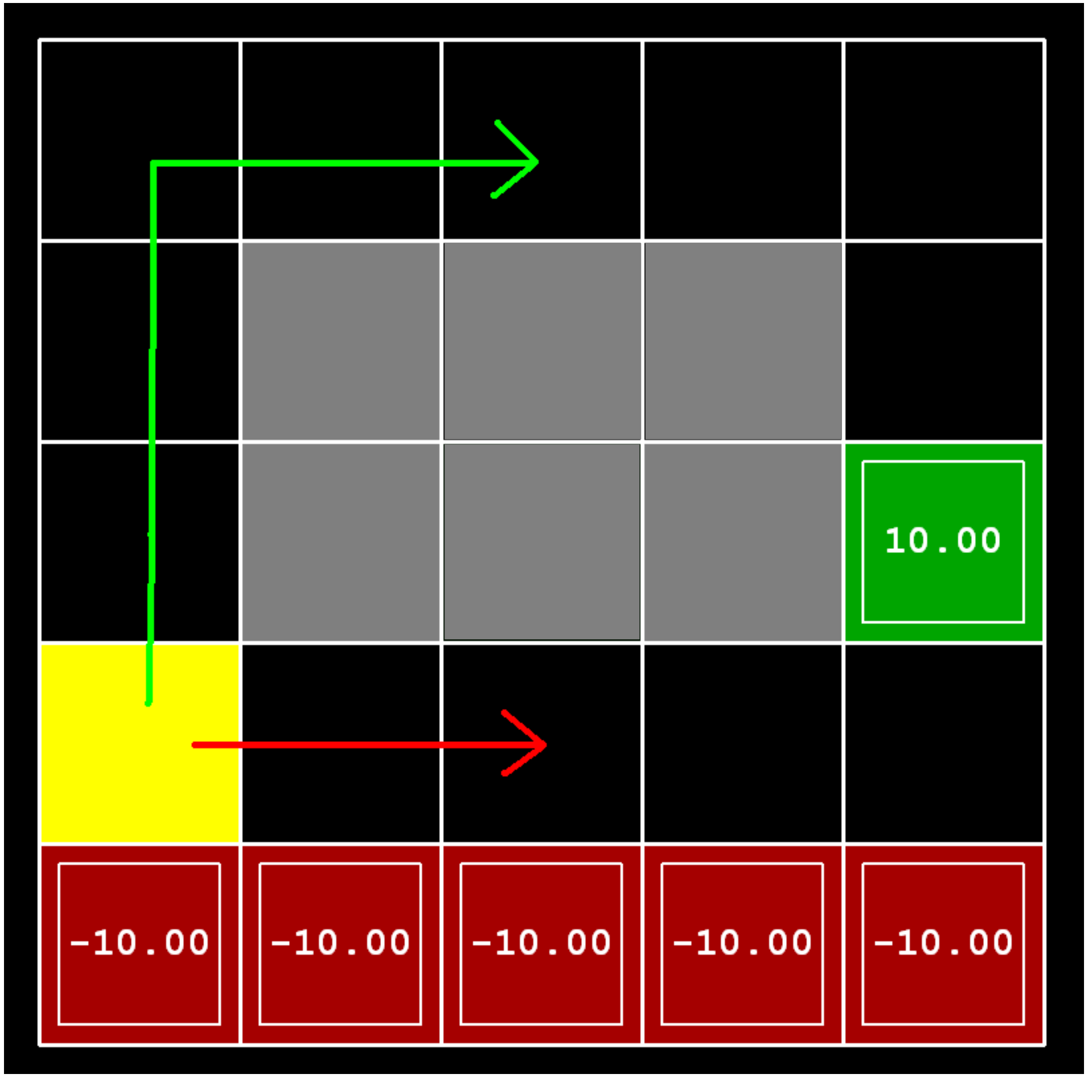
- If $\varepsilon \rightarrow 0$, it's **greedy in the limit**
- Be careful with non-stationary environments

Cliff world



Picture from Berkeley CS188x

Cliff world



Conditions

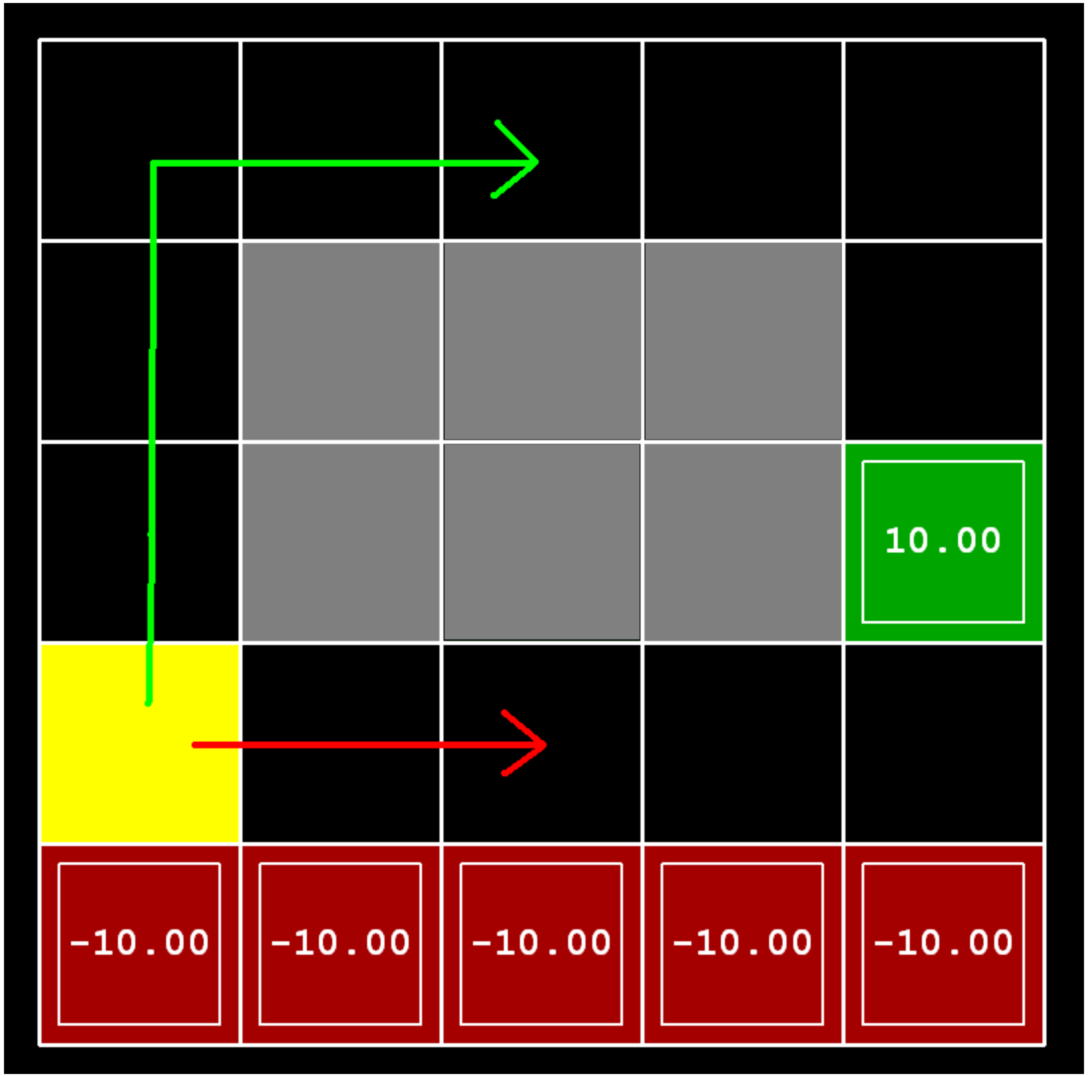
- Q-learning

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

- no slipping

Trivia:
What will q-learning learn?

Cliff world



Conditions

- Q-learning

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

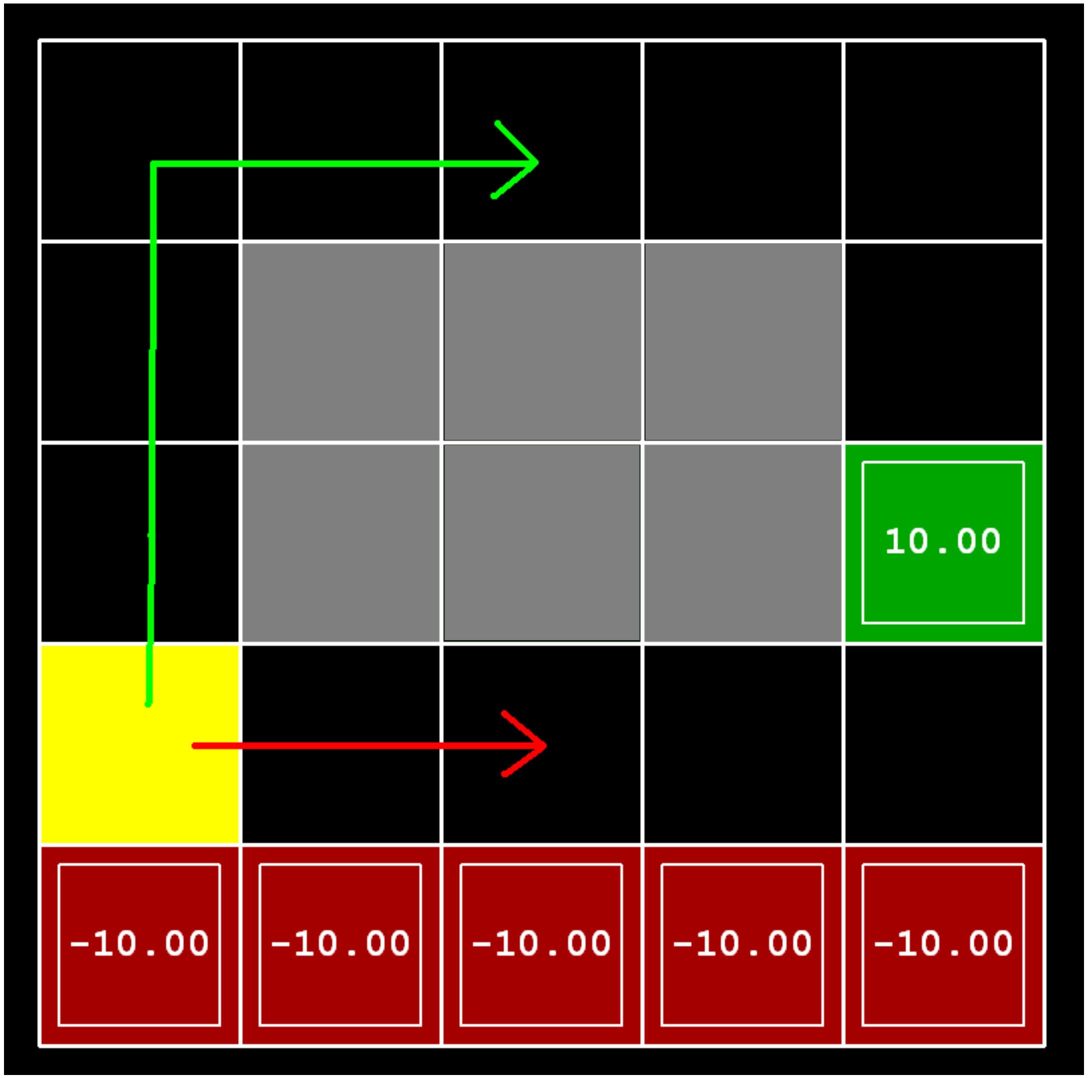
- no slipping

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

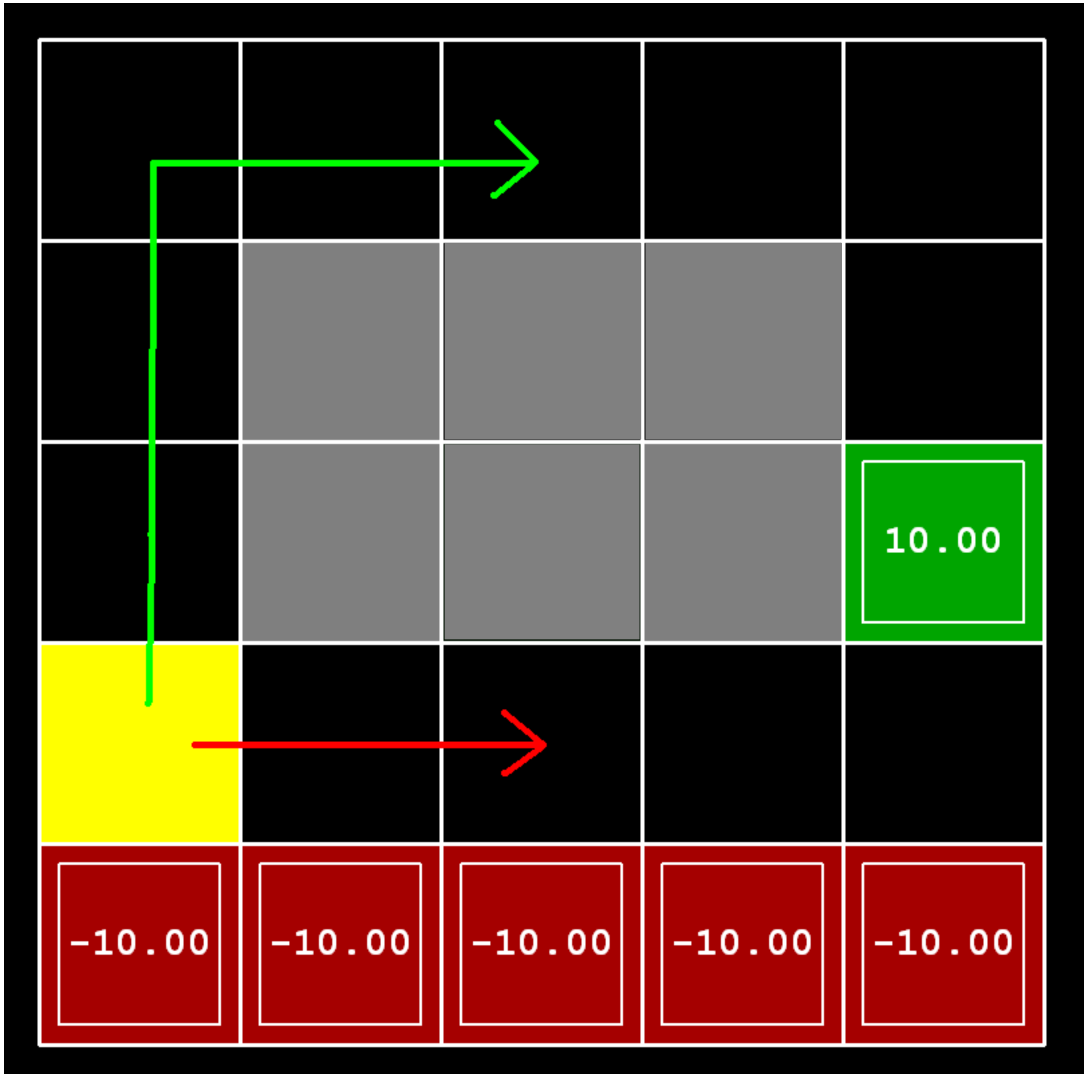
Trivia:
What will q-learning learn?

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.)

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

"better Q(s,a)"

Q-learning

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

Q-learning VS SARSA

Update rule (from Bellman eq.)

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

“better Q(s,a)”

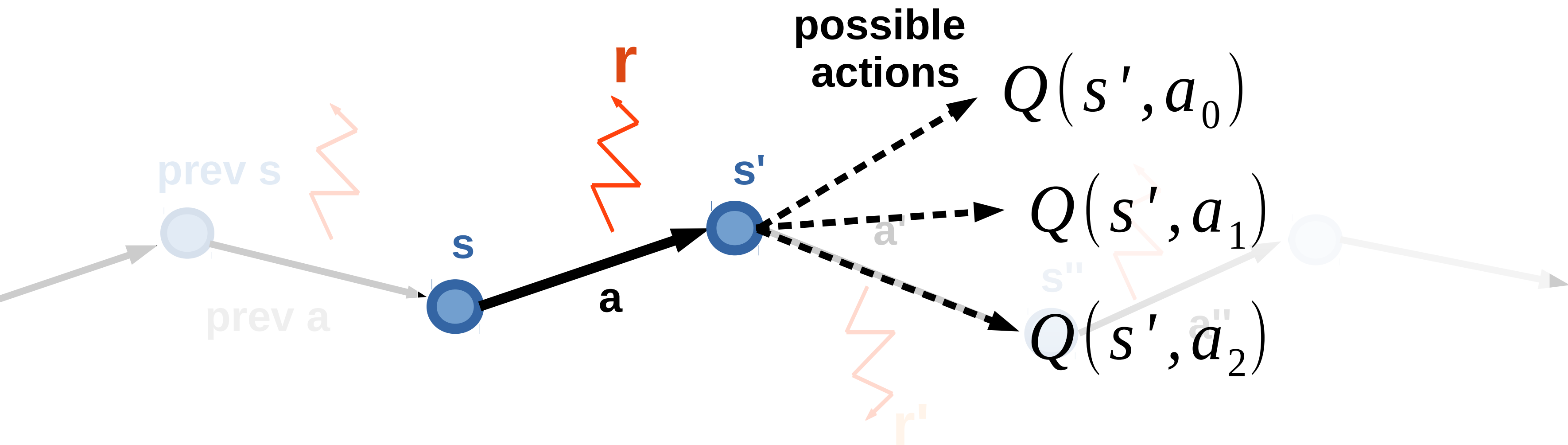
Q-learning

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

EV-SARSA

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

Recap: Q-learning

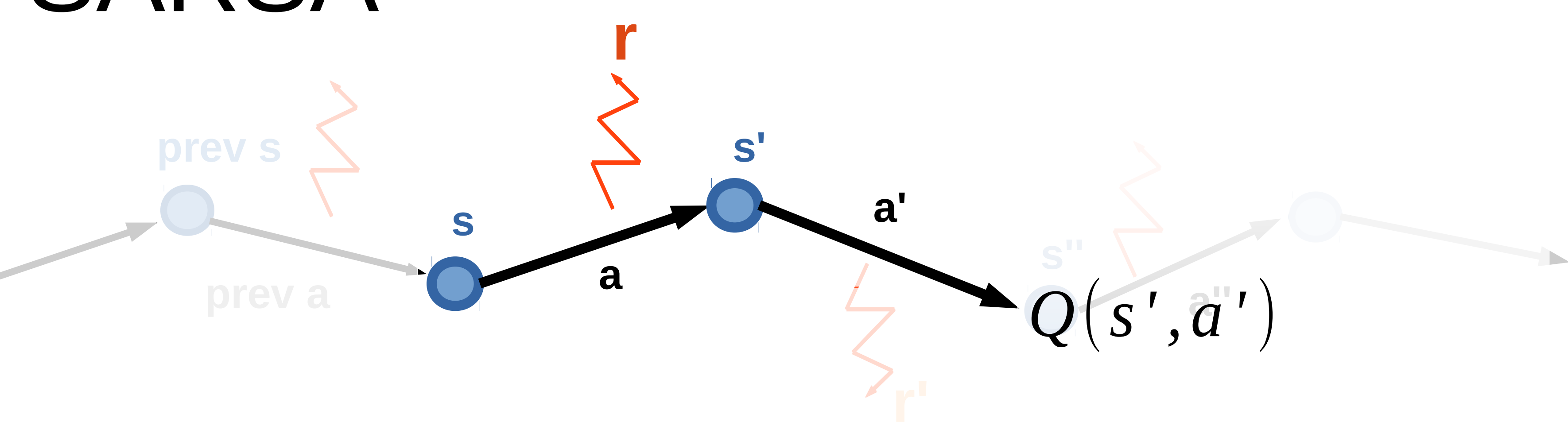


$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s**,**a**,**r**,**s'**> 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

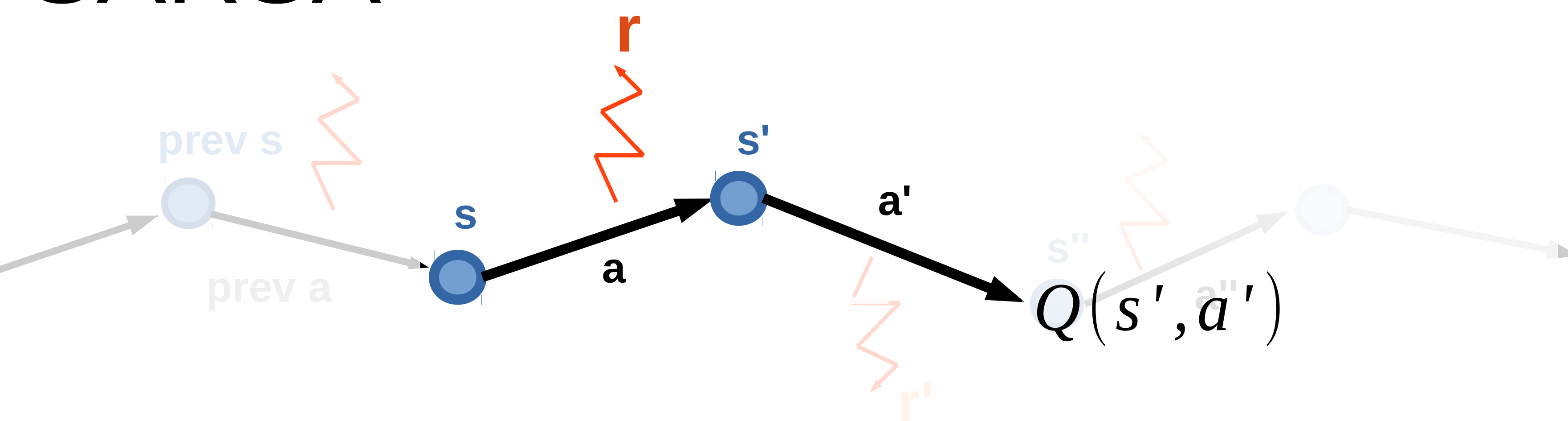


$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s**,**a**,**r**,**s'**,**a'**> 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 <**s**,**a**,**r**,**s'**,**a'**> from env

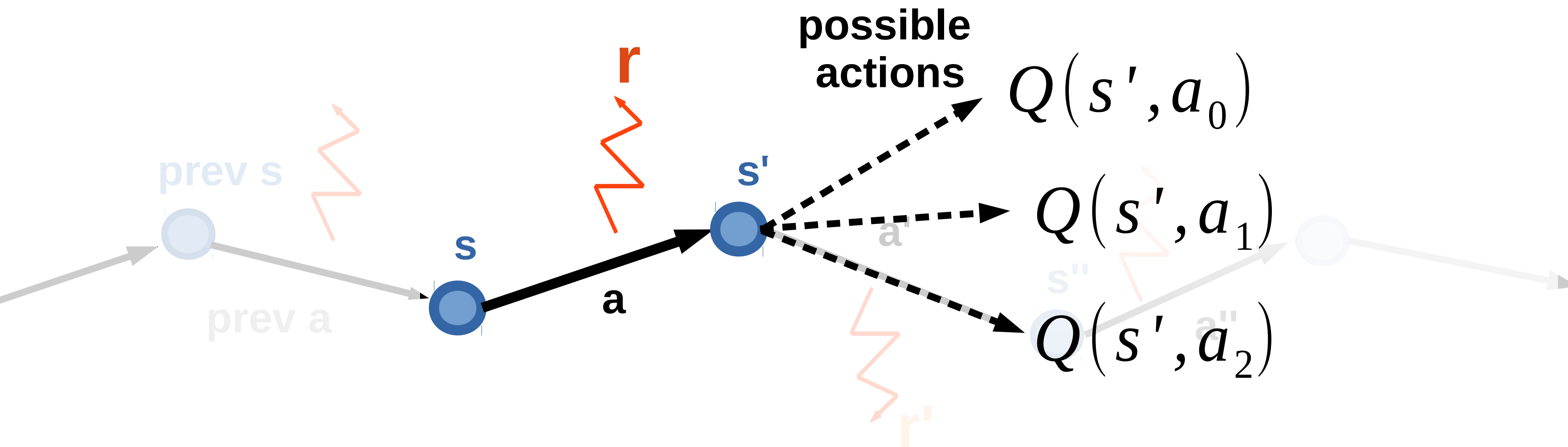
hence "SARSA"

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

next action (not max)

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

Expected value SARSA

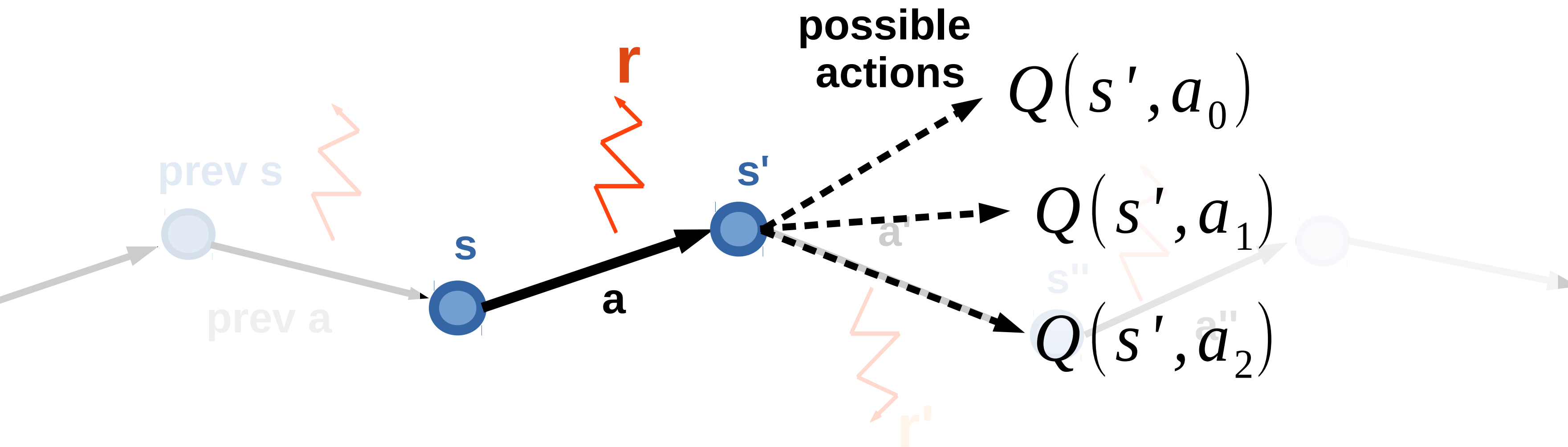


$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s**,**a**,**r**,**s'**> from env
- Compute $\hat{Q}(s,a) = r(s,a) + \gamma \underset{a_i \sim \pi(a|s')}{E} Q(s',a_i)$
- Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

Expected value SARSA



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

Loop:

- Sample <**s**,**a**,**r**,**s'**> from env

Expected value

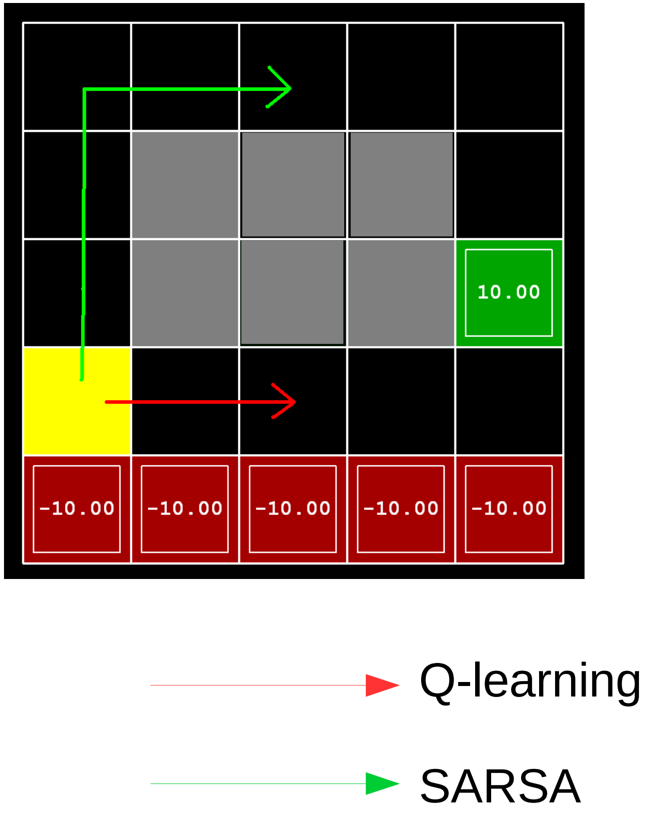
- Compute $\hat{Q}(s,a) = r(s,a) + \gamma \underset{a_i \sim \pi(a|s')}{E} Q(s', a_i)$

- 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



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

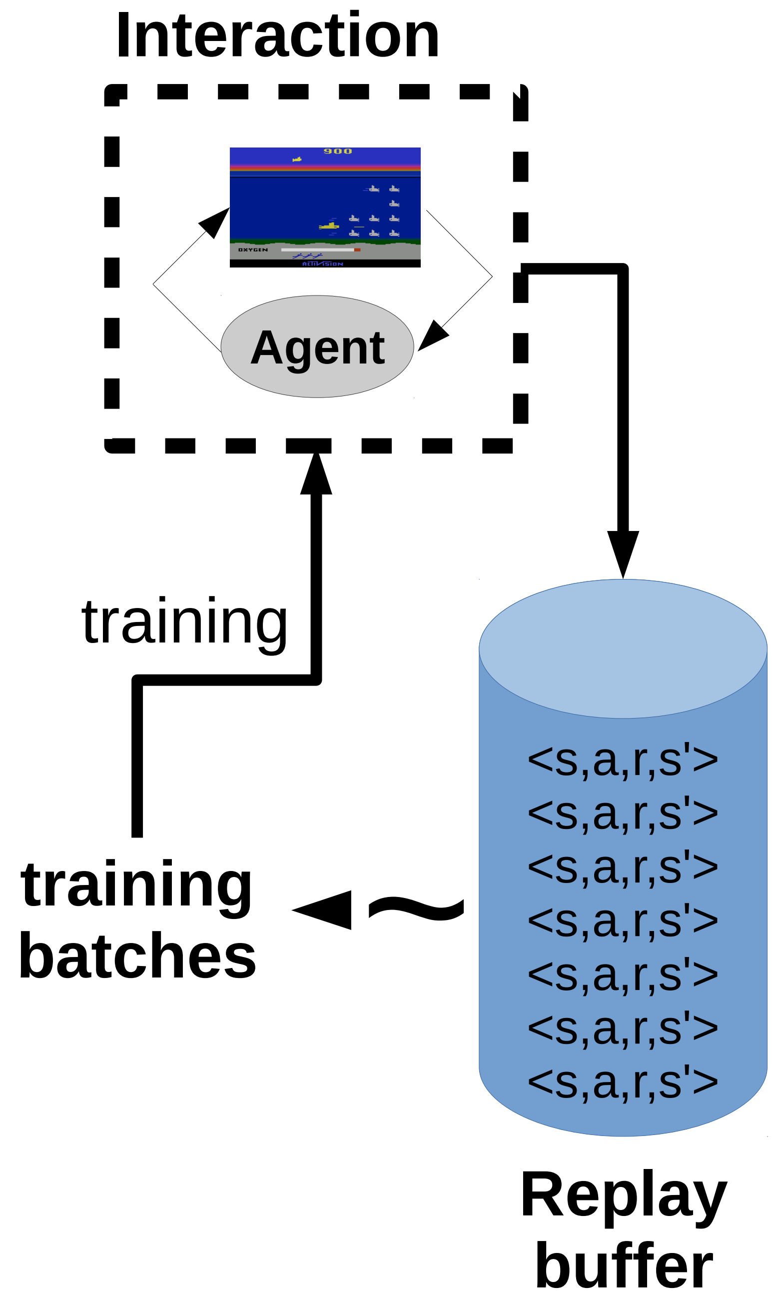
on-policy	**off-policy**
Agent **can** pick actions	Agent **can't** pick actions
– On-policy algorithms **can't** learn off-policy	– Off-policy algorithms **can** learn on-policy
– SARSA	– Q-learning
– more coming soon	– Expected Value SARSA

Experience replay

Idea: store several past interactions

<s,a,r,s'>

Train on random subsamples

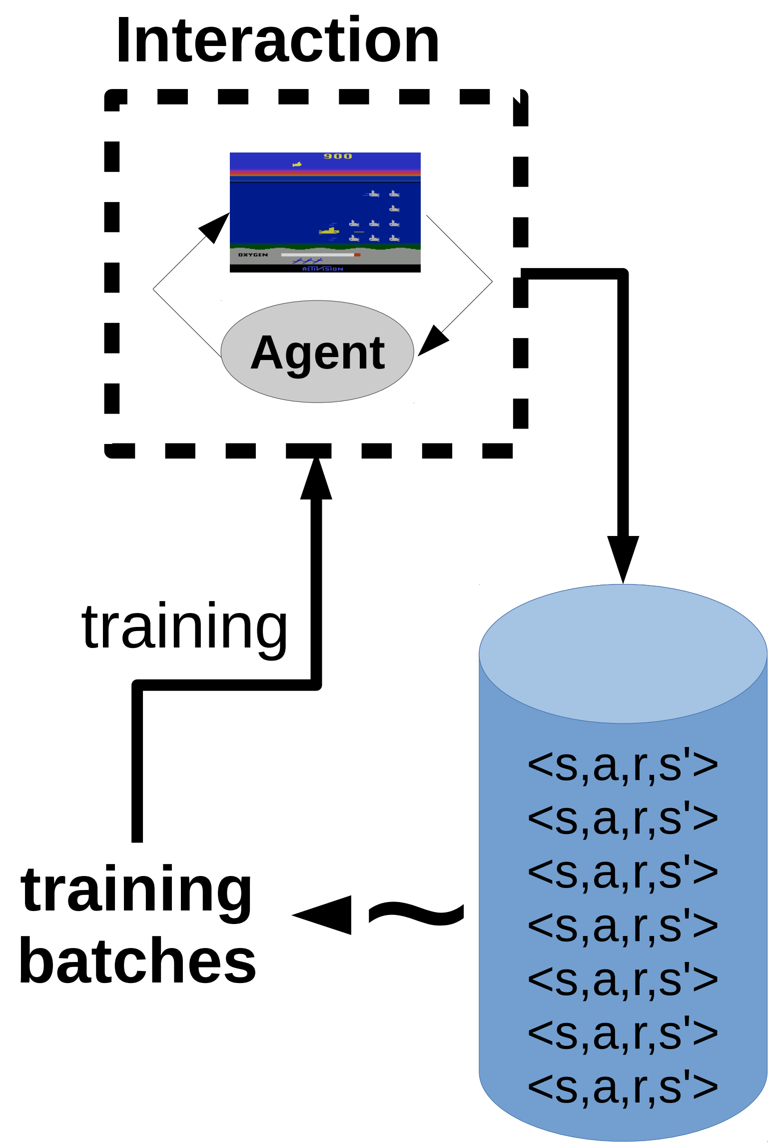


Experience replay

Idea: store several past interactions
<s,a,r,s'>
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?

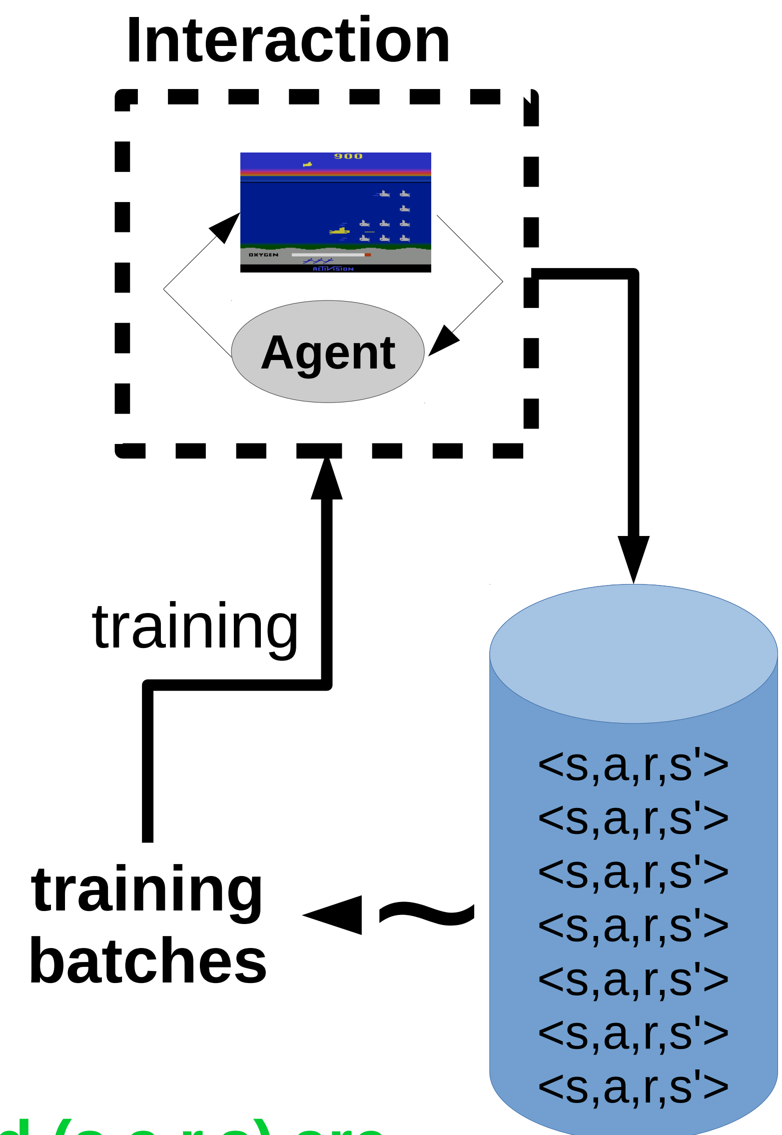
Experience replay

Idea: store several past interactions
<s,a,r,s'>
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!



Old (s,a,r,s) are from older/weaker version of policy!

Replay buffer

New stuff we learned

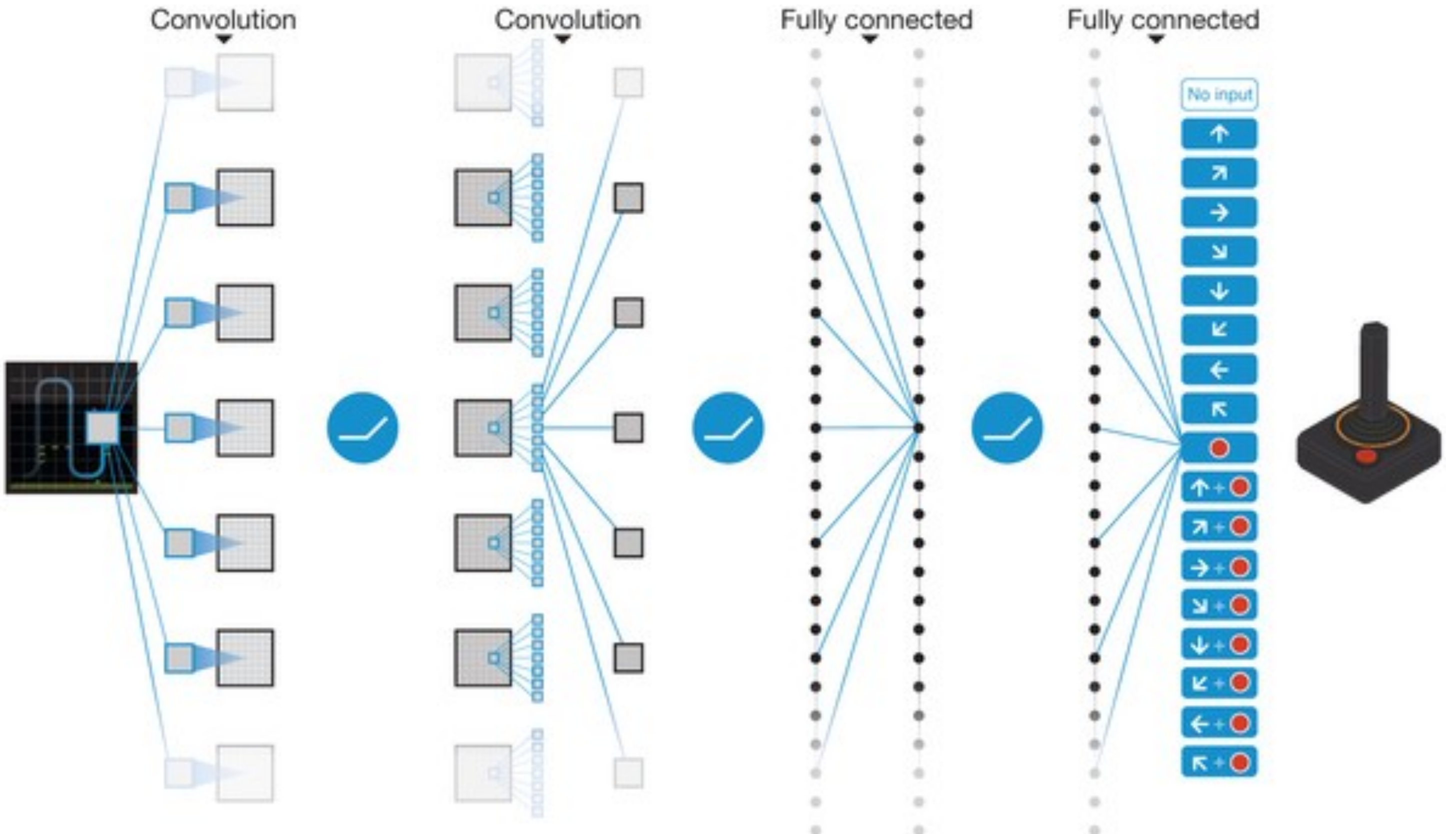
- Anything?

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



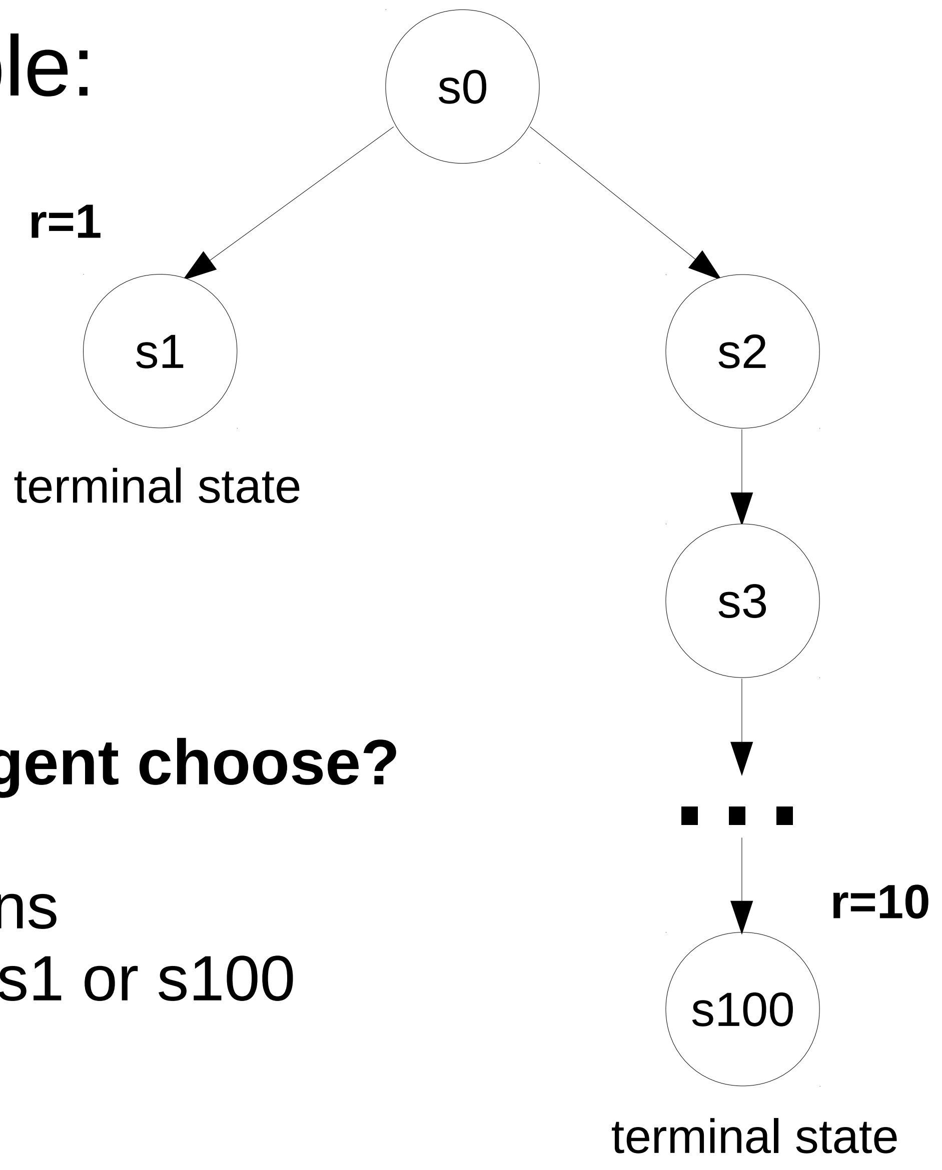
Outro and Q & A

- 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?

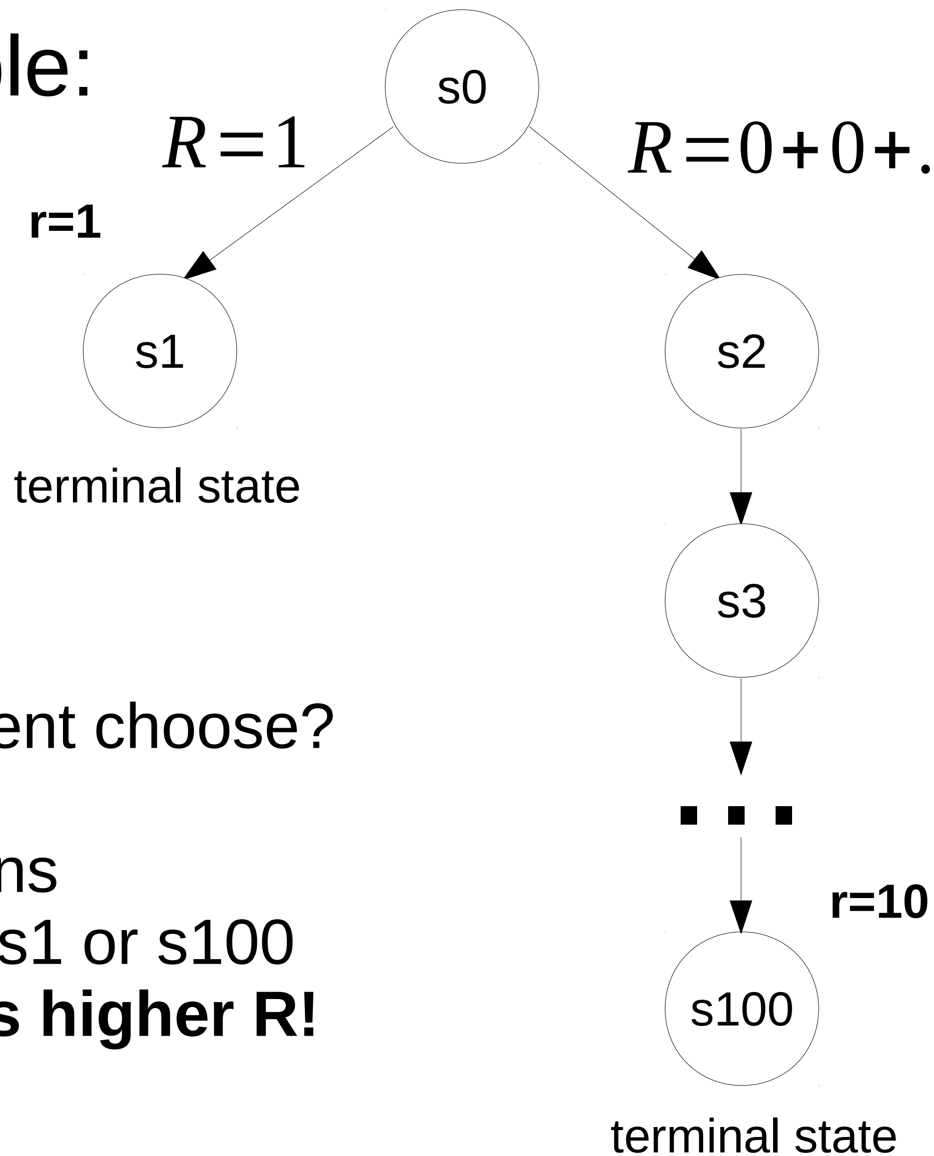
- γ=0.9
- arrows = actions
- game ends at s1 or s100

Discounted reward fails #1

Trivial example:

$R = 1$

$R = 0 + 0 + \ldots + \gamma^{99} \cdot 10 \approx 10^{-4}$



What path will agent choose?

- γ=0.9
- arrows = actions
- game ends at s1 or s100
- **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?

http://rl.deephack.me/

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

http://rl.deephack.me/

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

https://openai.com/blog/faulty-reward-functions/

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 $$1+\gamma+\gamma^2+\ldots=\frac{1}{(1-\gamma)}$$

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

  Typical values:
  - γ=0.9, 10 turns
  - γ=0.95, 20 turns
  - γ=0.99, 100 turns
  - γ=1, infinitely long

  Higher γ = less stable algorithm.
  γ=1 only works for episodic MDP (finite amount of turns).

Nuts and bolts: discount

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

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

- Atari Skiing, reward was delayed by in 5k steps

- γ=0.99 is not enough

- γ=1 and a few hacks works better

- Or use a better reward function