RL@Coursera
week 3

Model-free reinforcement learning

Yandex Data Factory

LAMBDA

What we learned

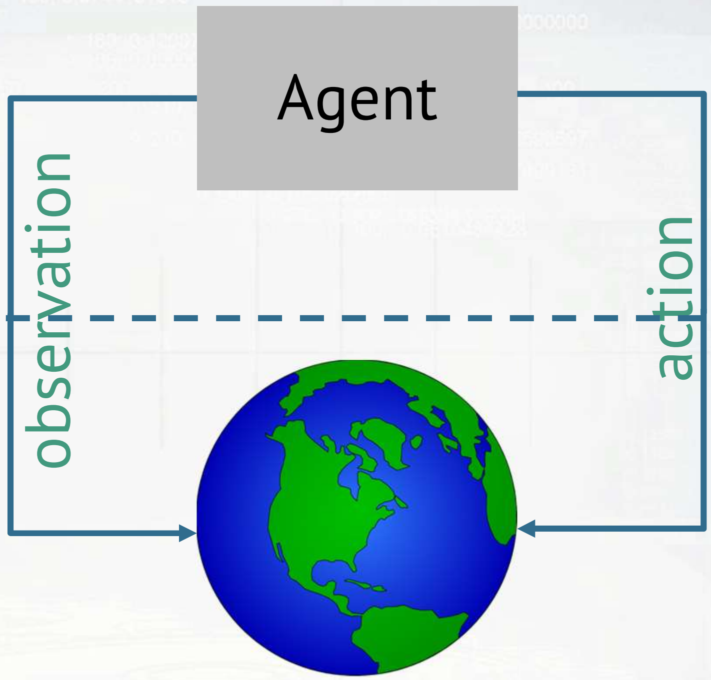
- V(s) and Q(s,a)
- If we know V or Q → we have optimal policy
- We can learn them with dynamic programming

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

Decision process in the wild

Can do anything

Can't even see
(worst case)



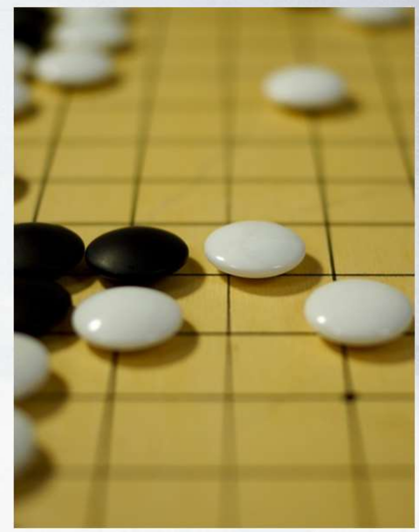
Environment

Decision process in the wild

Agent

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?

What we learned

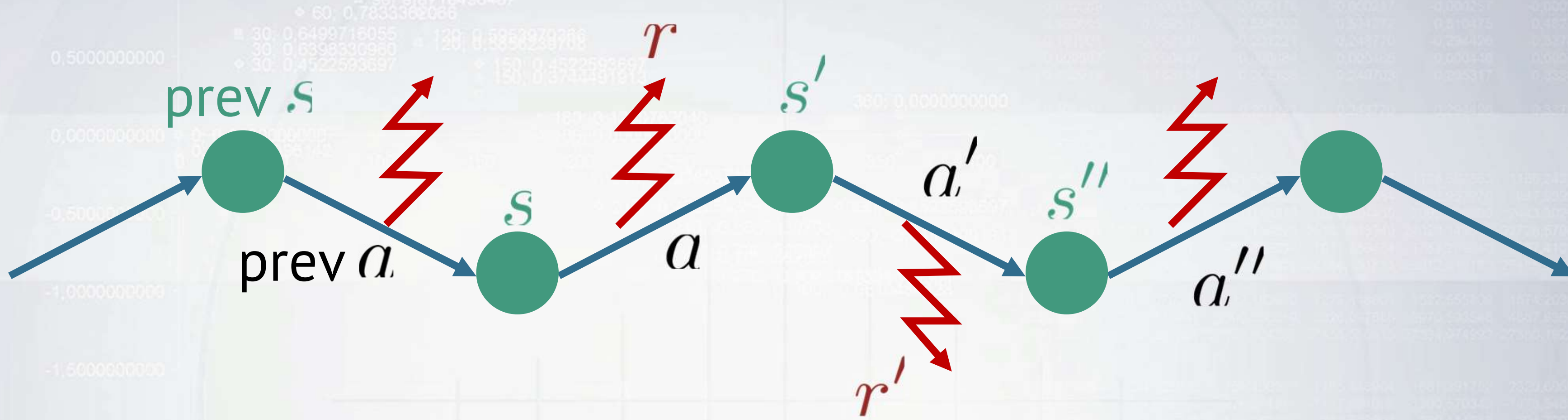
- **V(s)** and **Q(s,a)**
- If we know V or Q → we have optimal policy
- We can learn them with dynamic programming

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

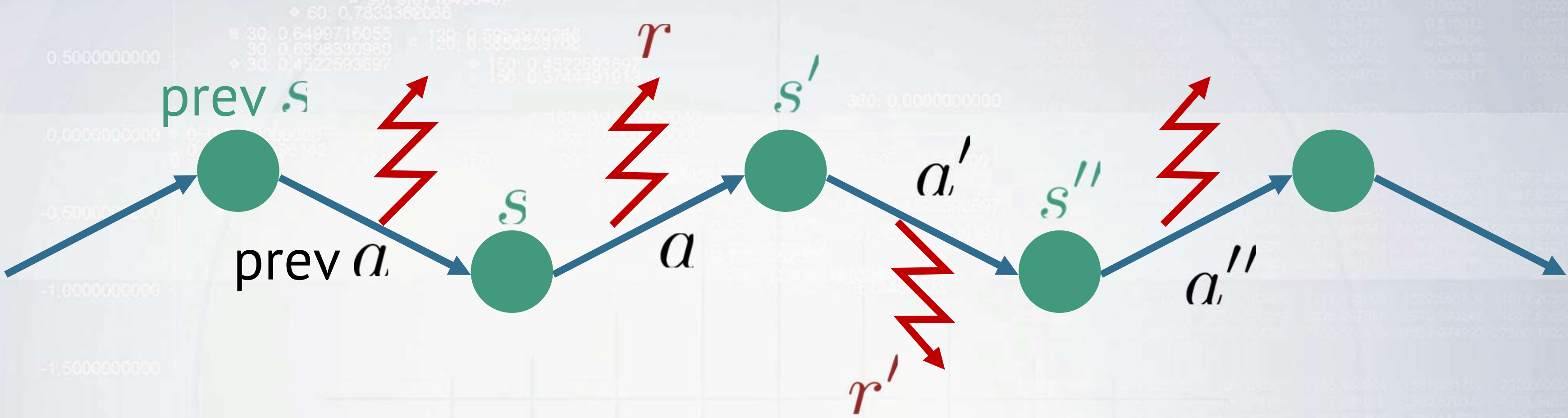
over both terms

Can't enumerate all

MDP trajectory

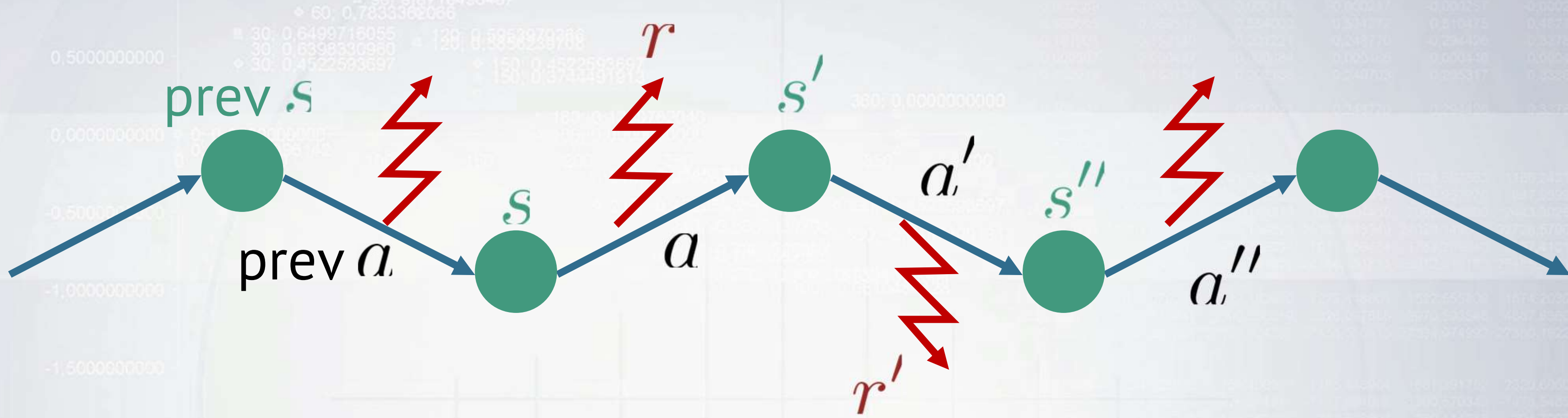


MDP trajectory



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

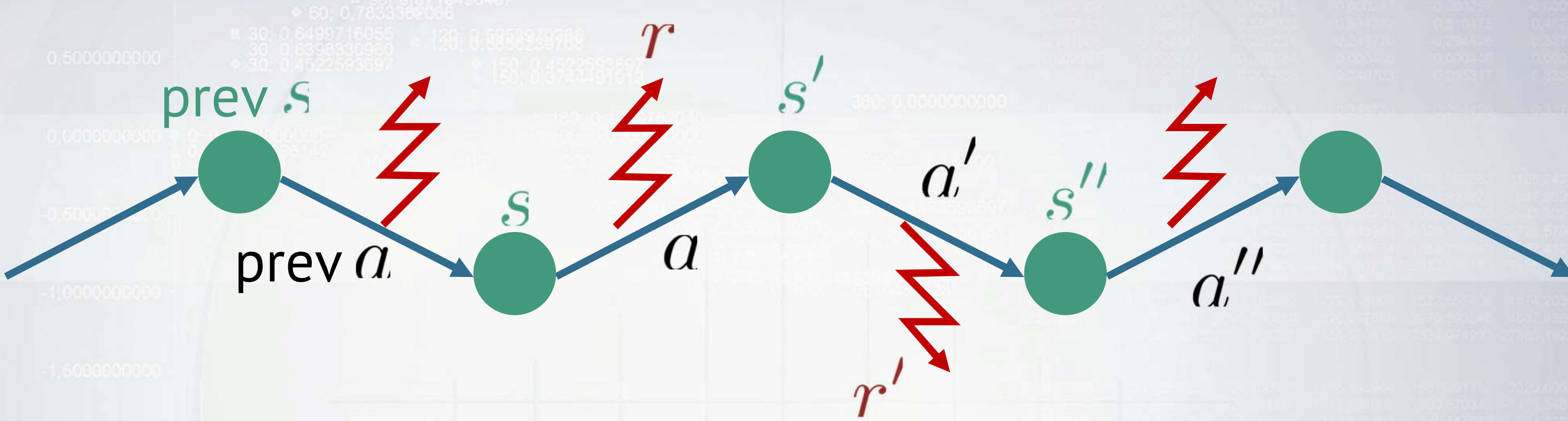
MDP trajectory



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

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

MDP trajectory



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

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

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

From V to Q

One approach: action Q- values

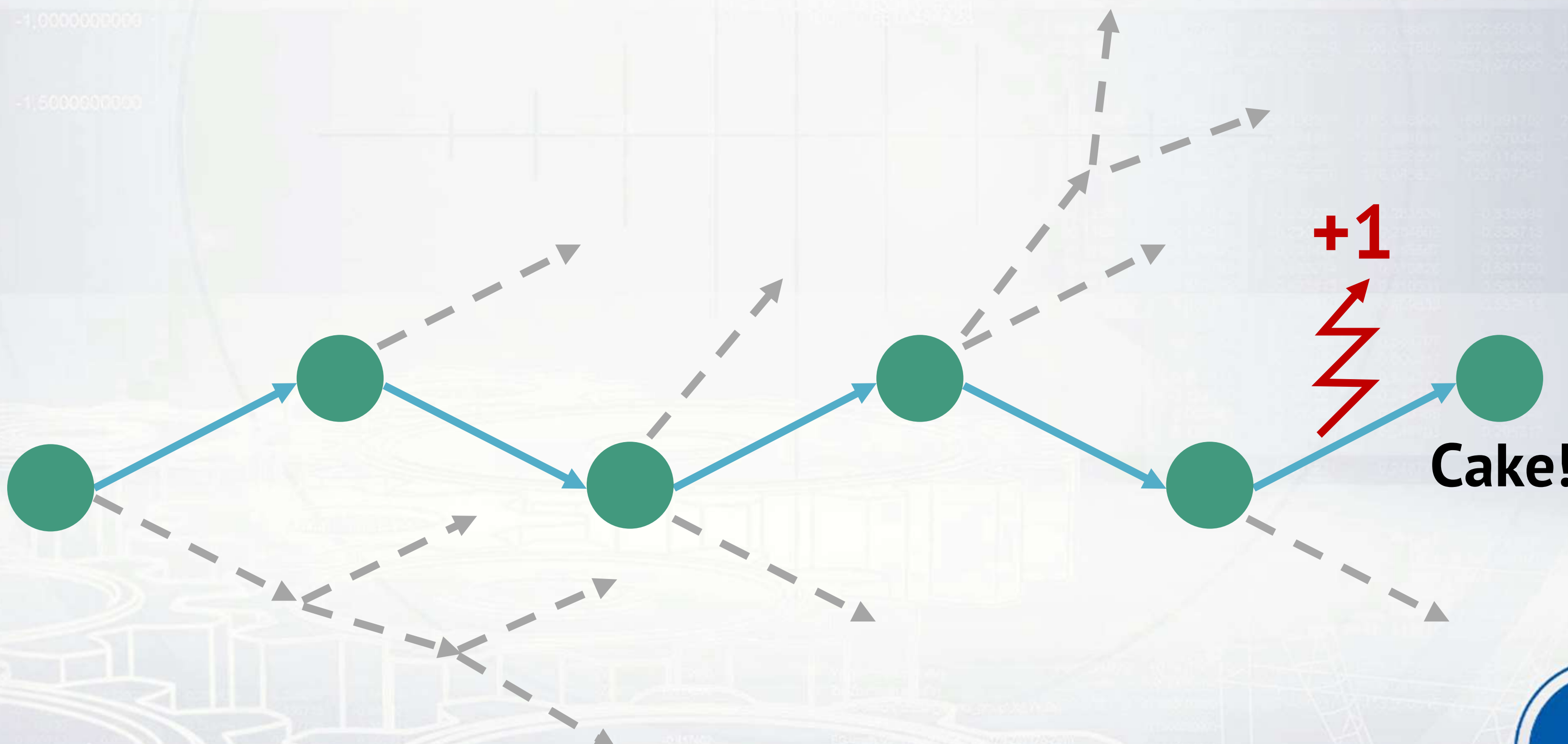
$$Q(s, a) = r(s, a) + \gamma \cdot V(s')$$

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

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

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

not always practical
to implement

takes a lot of sessions

Авторство

Idea 1: 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 1: temporal difference

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

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

$Q^*(s,a)$

That's value for $\pi*$ aka optimal policy

Idea 1: 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 $\pi*$ aka optimal policy

That's something <u>we don't have</u>

What do we do?

Idea 1: temporal difference

Авторство

Idea 1: temporal difference

- Replace expectation with sampling

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

however, since it will be very noisy, we can not directly use it as Q-value.

Idea 1: temporal difference

- Replace expectation with sampling

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

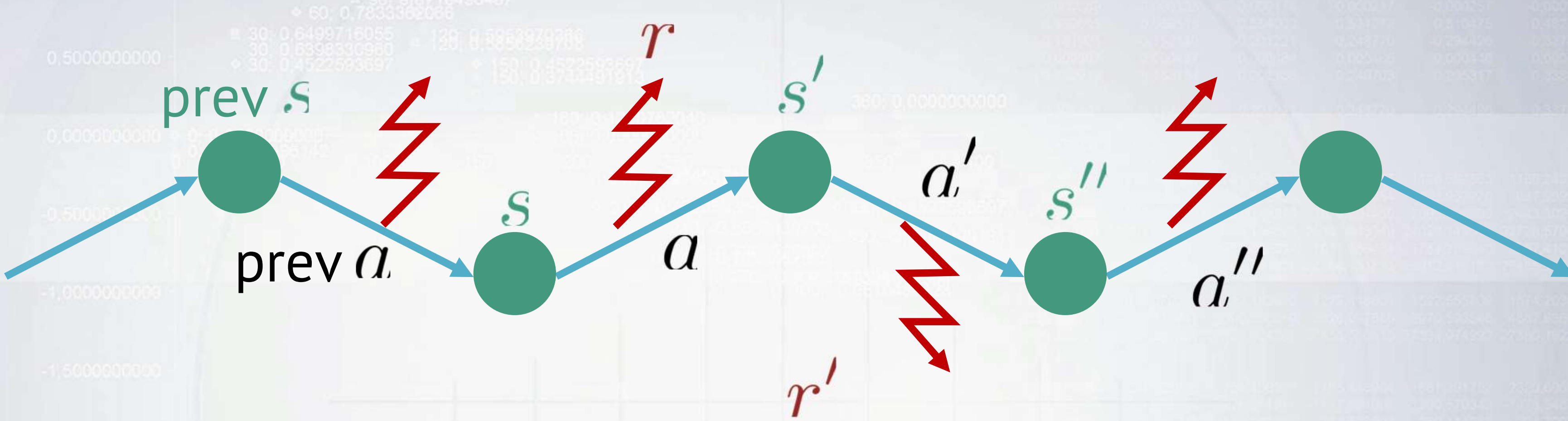
- Use moving average with just one sample!

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

Exponentially weighted moving average

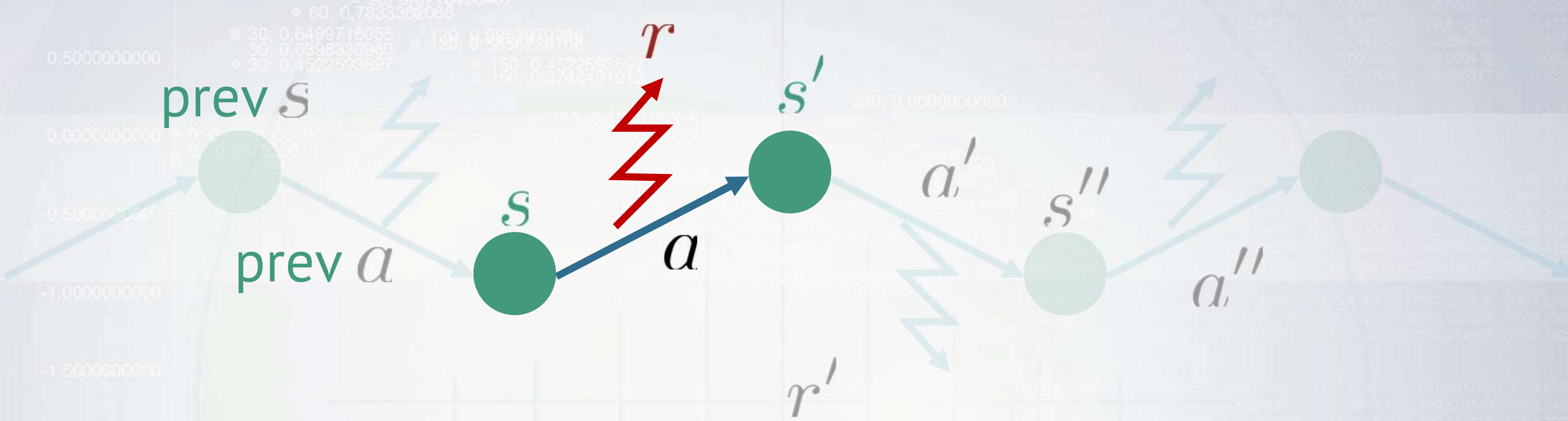
↳ a smoother version of the prev noisy version

MDP trajectory



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

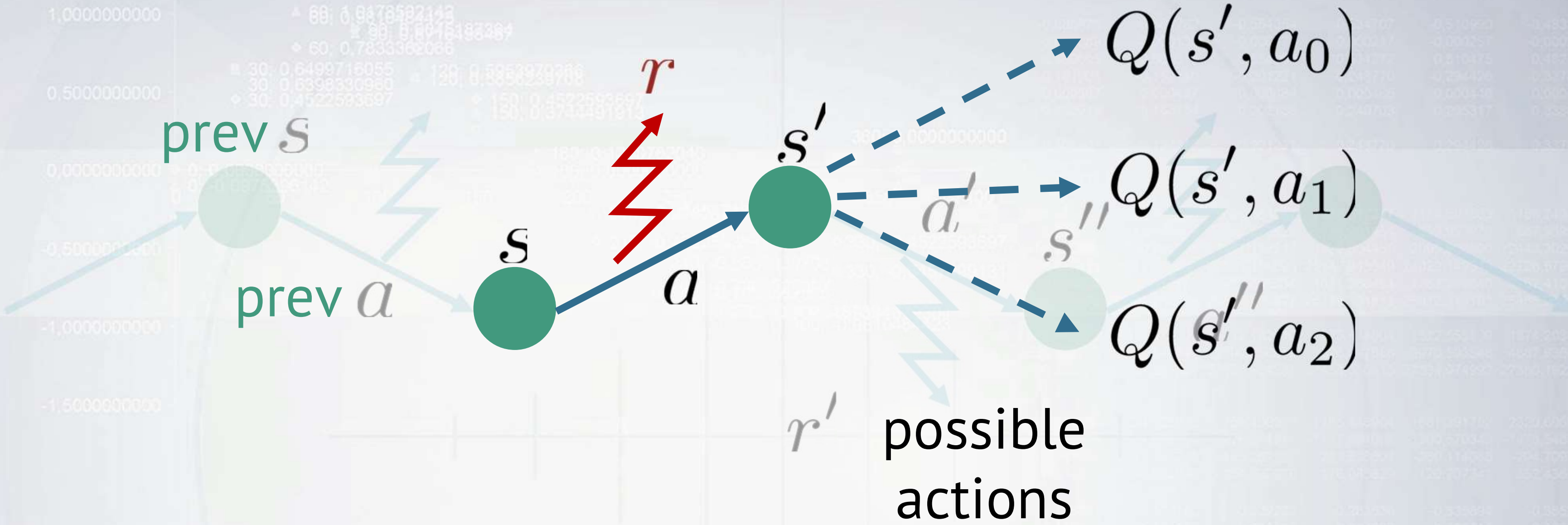
Q-learning



Initialize Q(s,a) with zeros

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

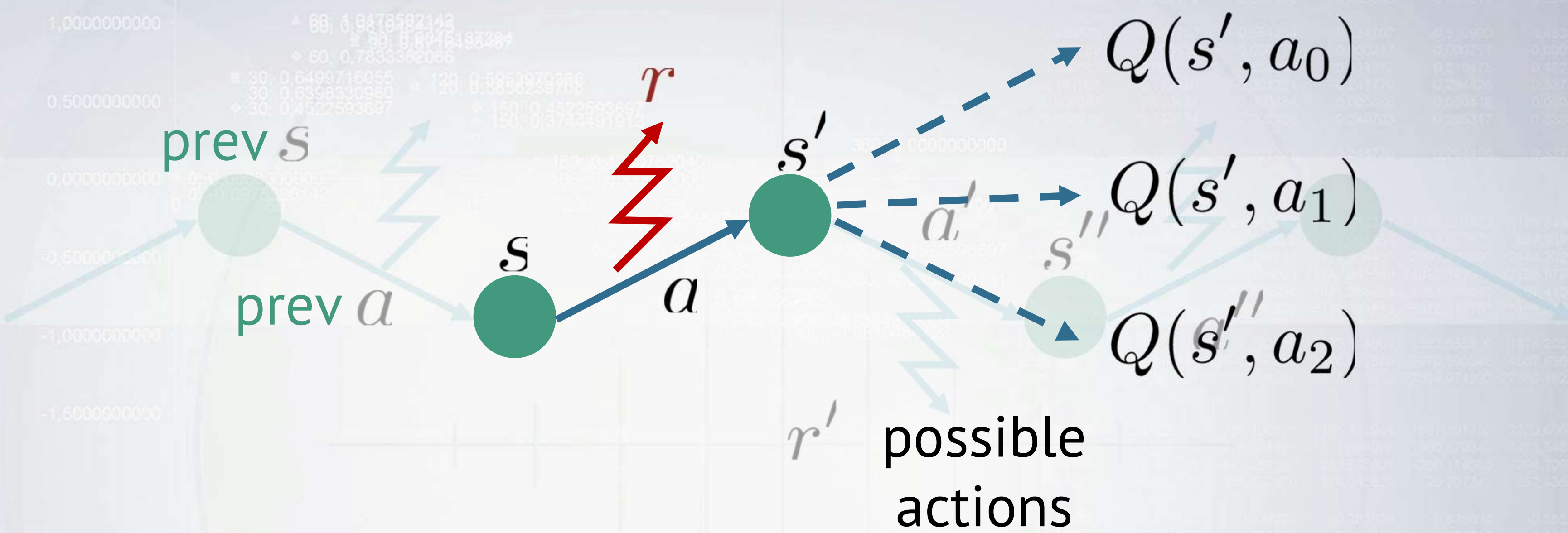
Q-learning



Initialize Q(s,a) with zeros

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

Q-learning

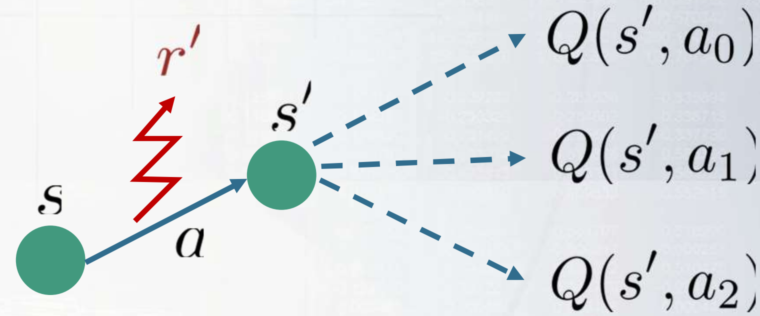


Initialize Q(s,a) with zeros

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

Recap

Monte-carlo	Temporal Difference
Averages Q over sampled paths	Uses recurrent formula for Q

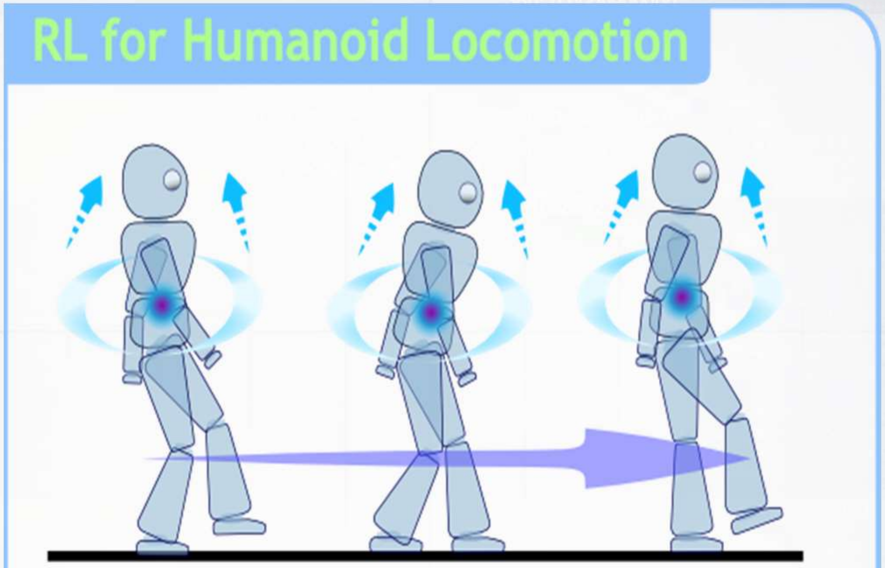


Nuts and bolts: MC vs TD

Monte-carlo	Temporal Difference
Averages Q over sampled paths	Uses recurrent formula for Q
Needs full trajectory to learn	Learns from partial trajectory Works with infinite MDP
Less reliant on markov property	Needs less experience to learn

What could possibly go wrong?

Our mobile robot learns to walk.



Авторство

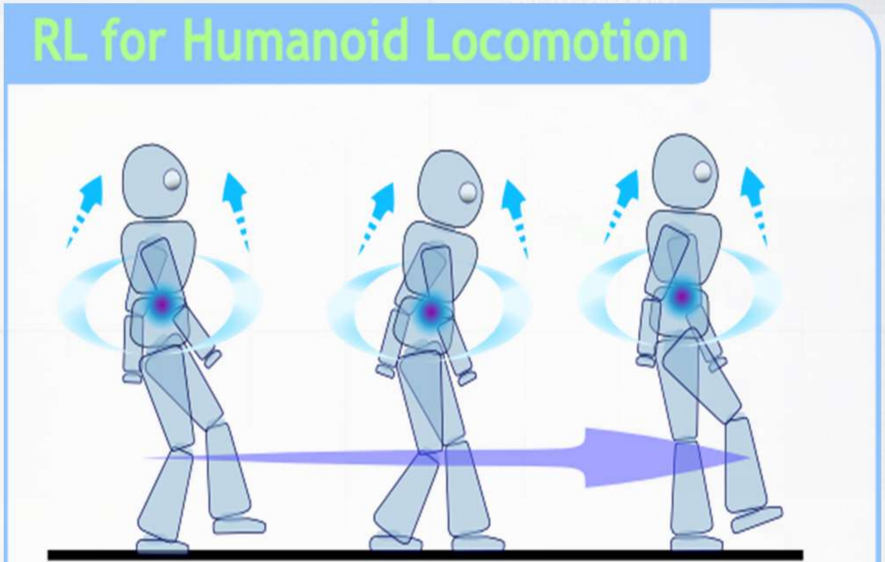
Initial $Q(s,a)$ are zeros

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.



Авторство

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,

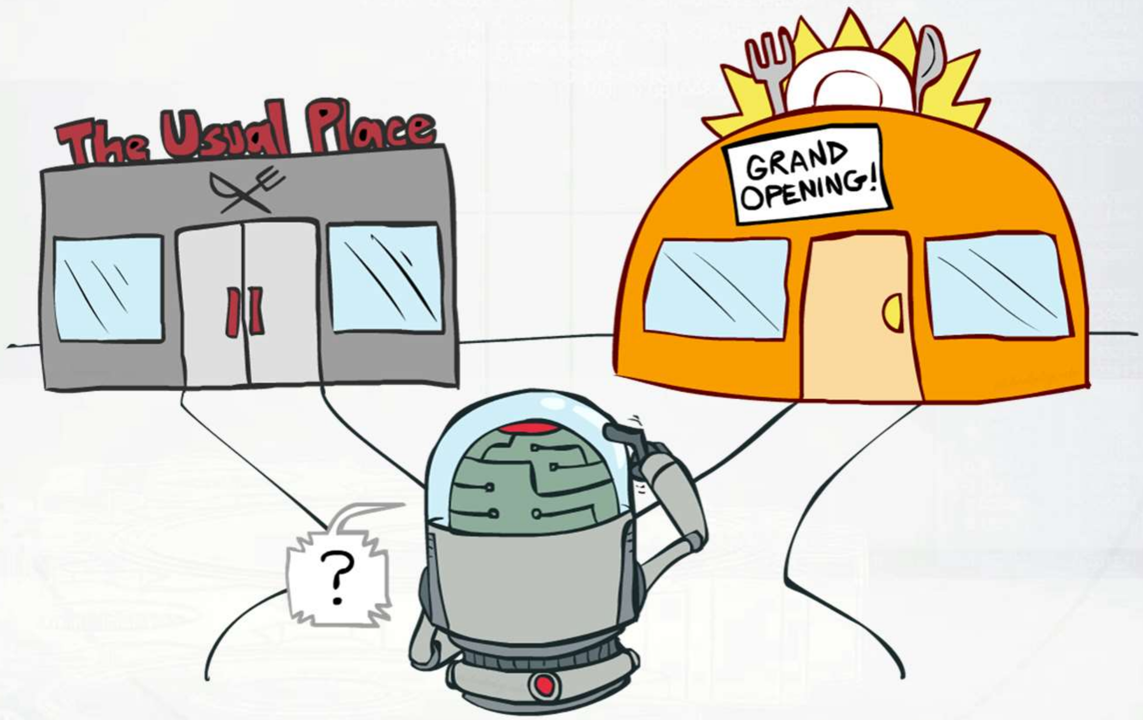
How will he ever learn that other actions
may be better than his current best one?

Stuck in
local optima

Ideas?

Exploration Vs Exploitation

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



Авторство

Exploration Vs Exploitation

Strategies:

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

One drawback :

All other actions , except the optimal , will be explored with the same prob i.e equally attractive

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.

$$P(a) = software(\frac{Q(a)}{\tau})$$

$\tau \to$ large

all Q-values will be approx equal

$\to \infty$

greedy-like approach

- More cool stuff coming later

Exploration over time

Idea:

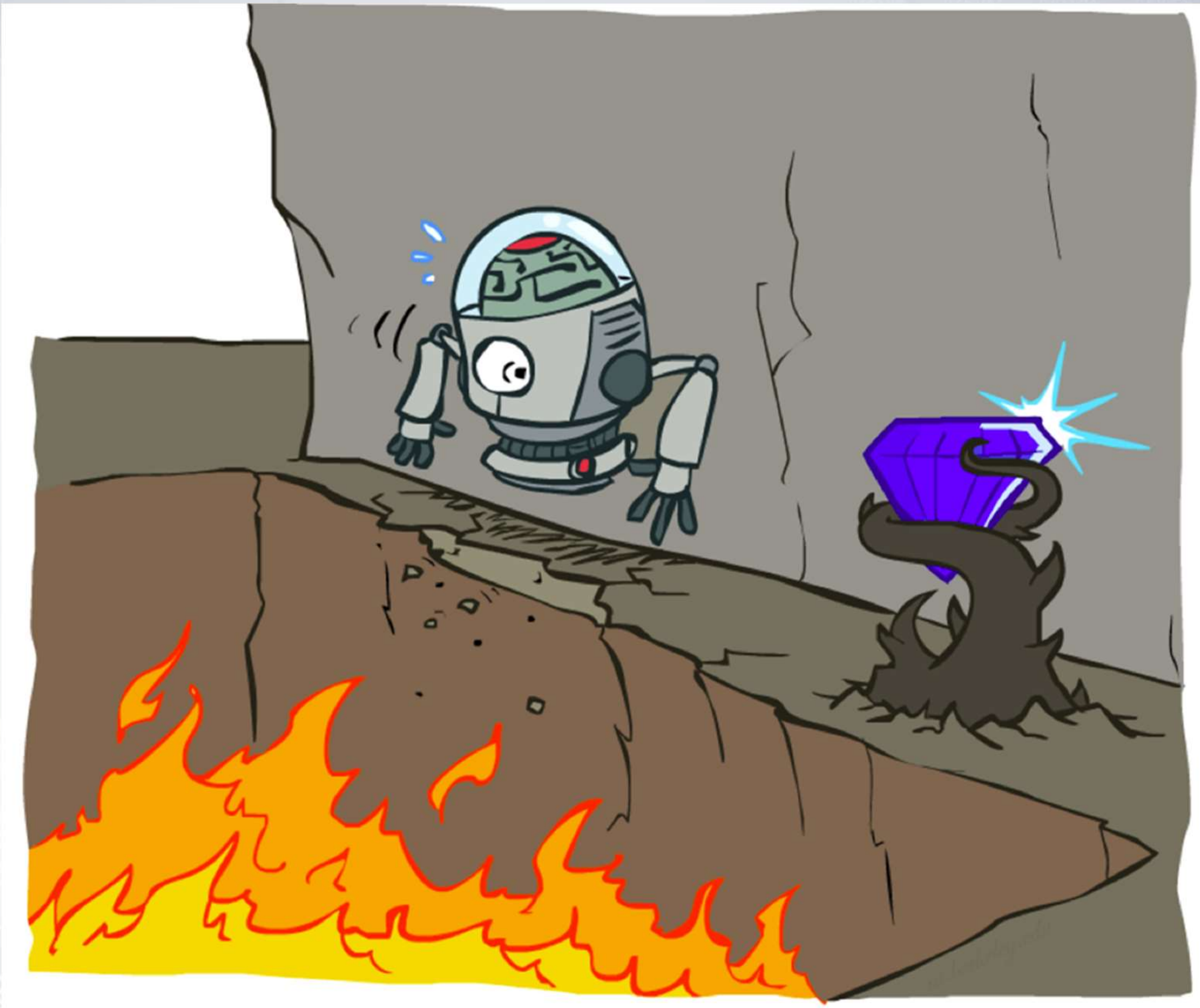
If you want to converge to optimal policy, you need to gradually reduce exploration

Example:

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

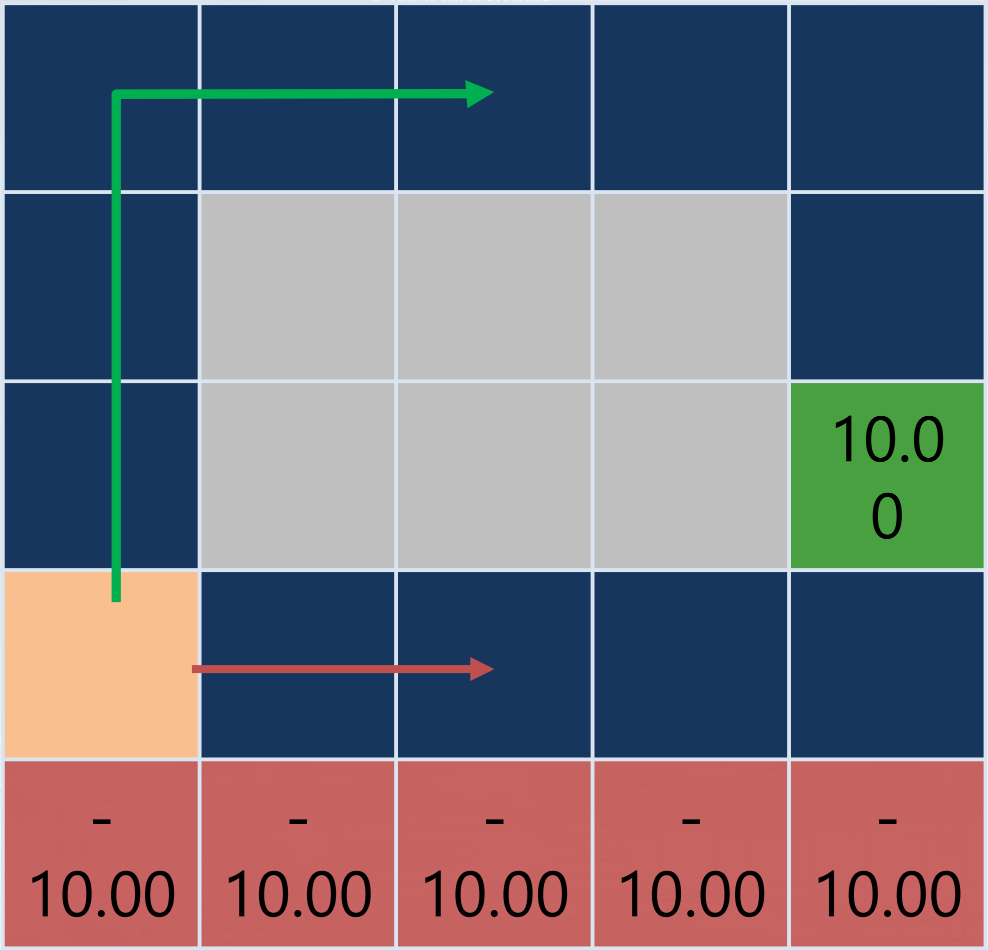
- If $\varepsilon \to 0$, it's **greedy in the limit**
- Be careful with non-stationary environments

Cliff world



Picture from Berkeley CS188x

Cliff world

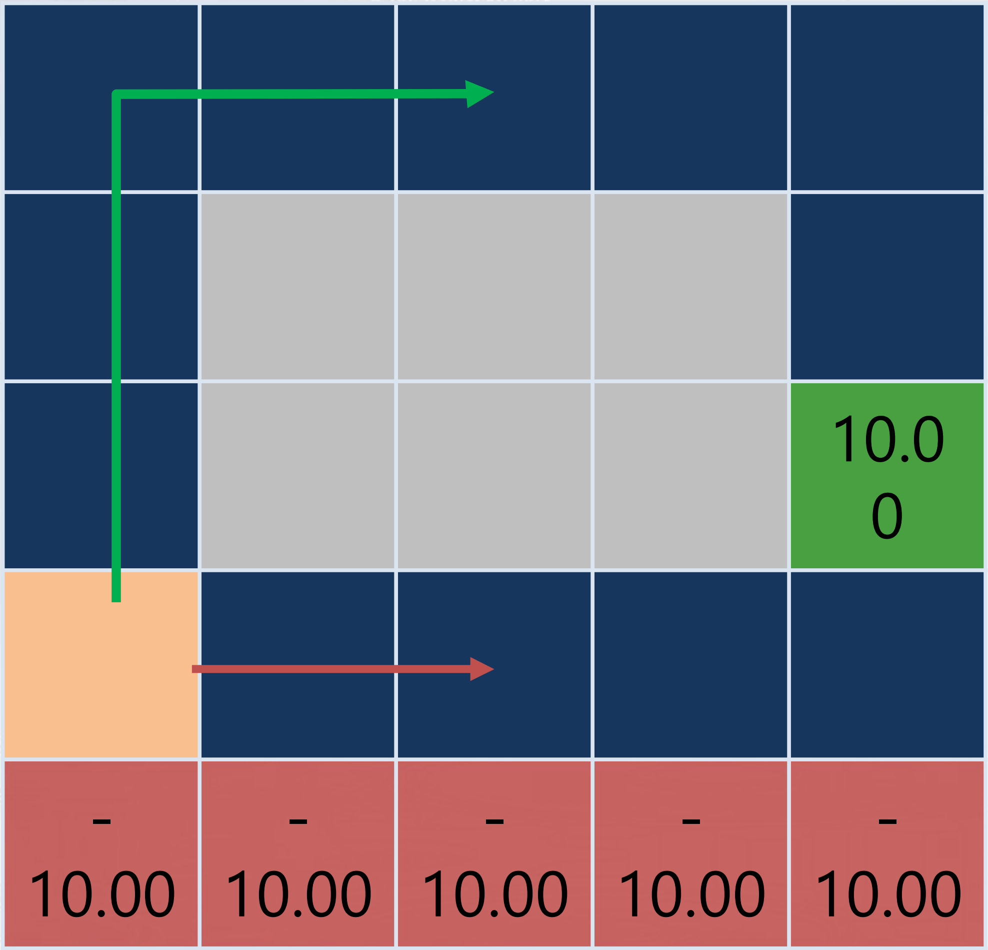


Conditions

- Q-learning
- $\gamma = 0.99 \ \epsilon = 0.1$
- no slipping

Trivia: What will q-learning learn?

Cliff world



Conditions

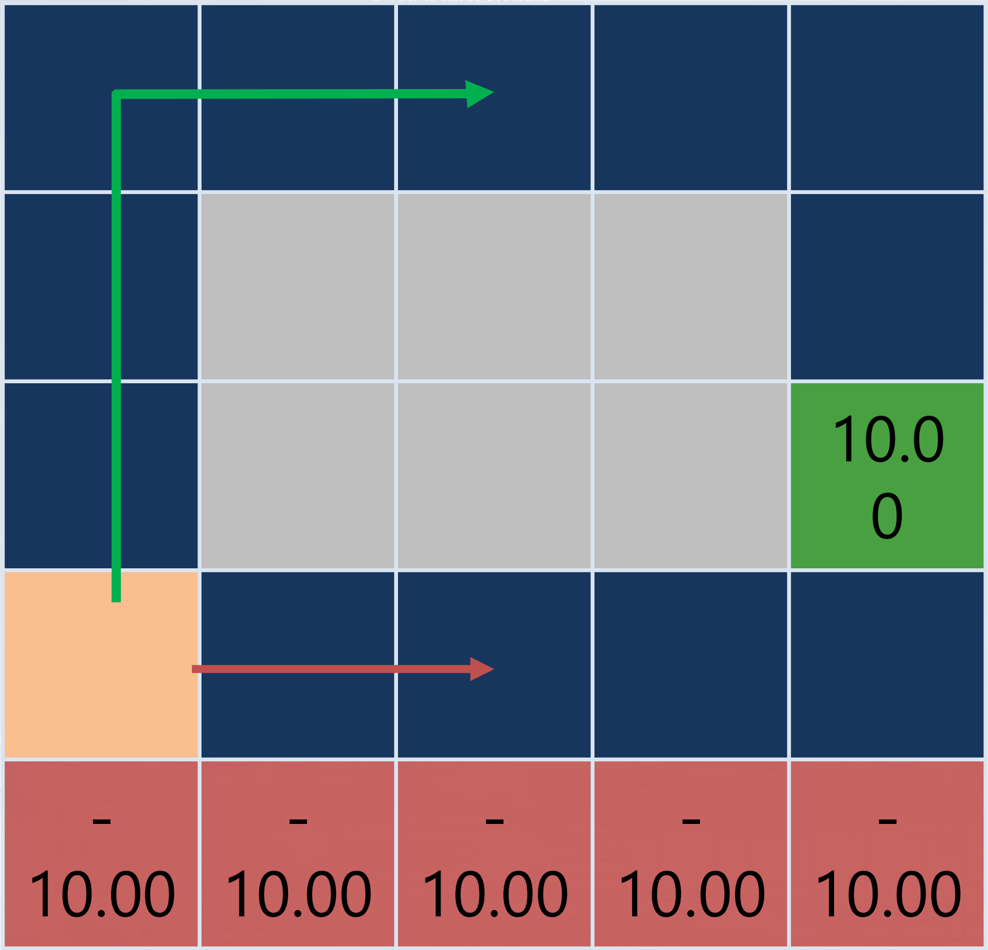
- Q-learning
- $\gamma = 0.99 \ \epsilon = 0.1$
- no slipping

Trivia: What will q-learning learn?

follow the short path

Will it maximize reward?

Cliff world



Conditions

- Q-learning
- $\gamma = 0.99 \ \epsilon = 0.1$
- no slipping

Trivia: What will q-learning learn?

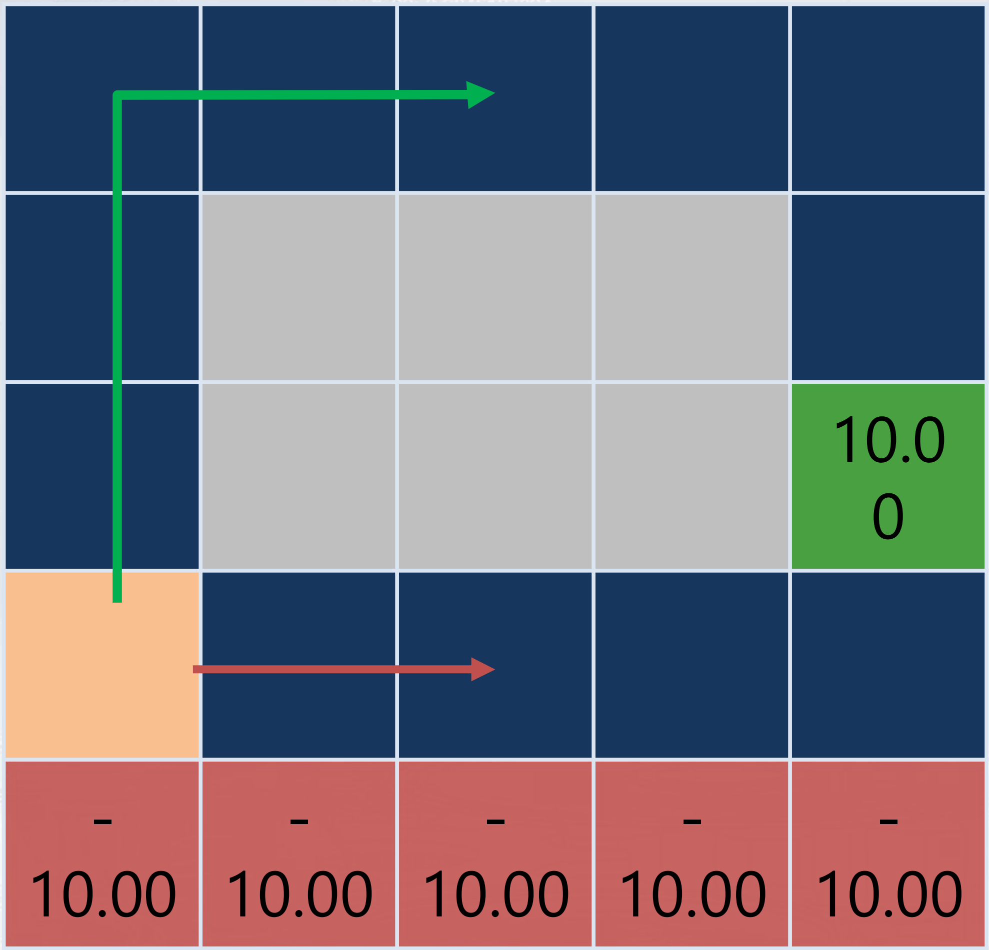
follow the short path

Will it maximize reward?

no, robot will fall due to epsilon-greedy "exploration"

no matter how long you train

Cliff world



Conditions

- Q-learning
- $\gamma = 0.99 \;\; \epsilon = 0.1$
- no slipping

Decisions must account for actual policy!

e.g. ε - greedy policy

* longer path in this case will be better, using ε-greedy

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

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

won't fit in

with exploration

Exploration over time

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

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

SARSA

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

Exploration over time

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

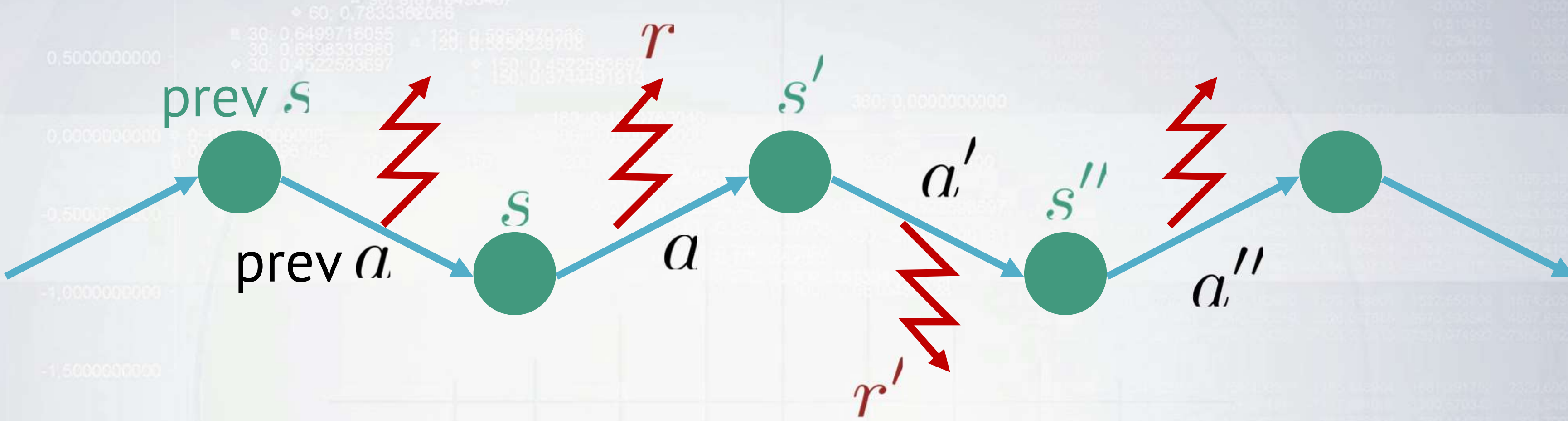
$$\hat{Q}(s,a) = r(s,a) + \gamma \cdot \max_{a'} Q(s', a')$$

Expected from $s' \sim P(s'|s,a)$

SARSA

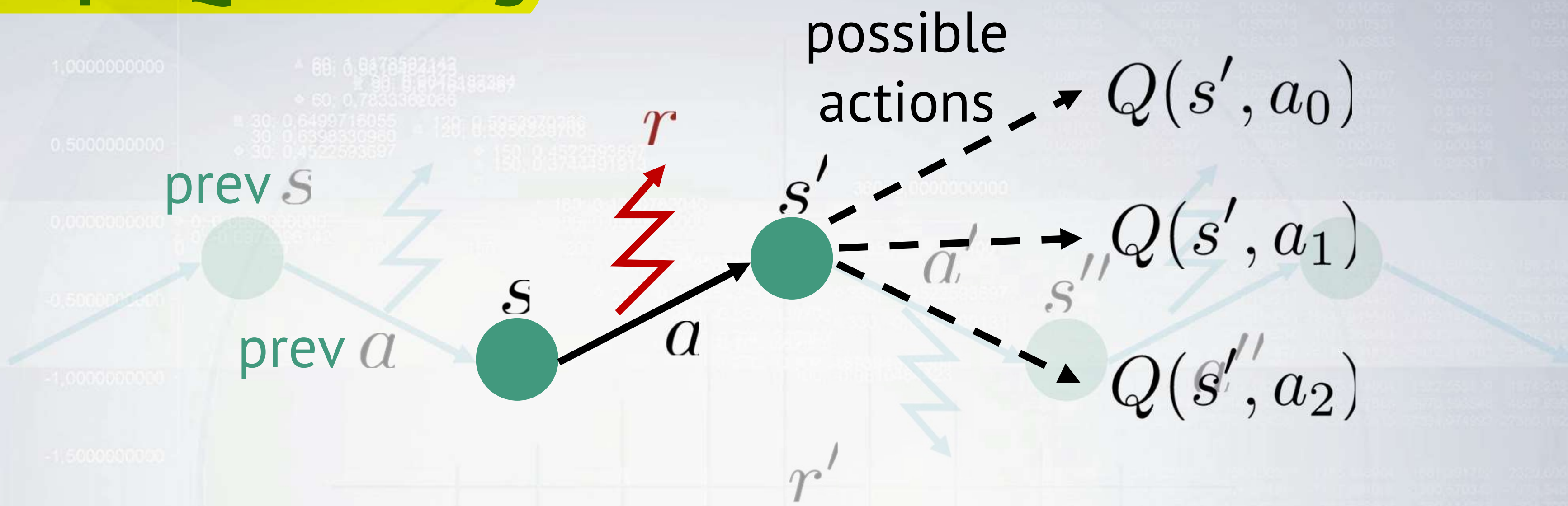
$$\hat{Q}(s,a) = r(s,a) + \gamma \cdot \mathop{E}_{a' \sim \pi(a'|s')} Q(s', a')$$

MDP trajectory



- Trajectory is a sequence of
  - states (s)
  - actions (a)
  - rewards (r)
- Can be infinite, we can't wait that long

Recap: Q-learning



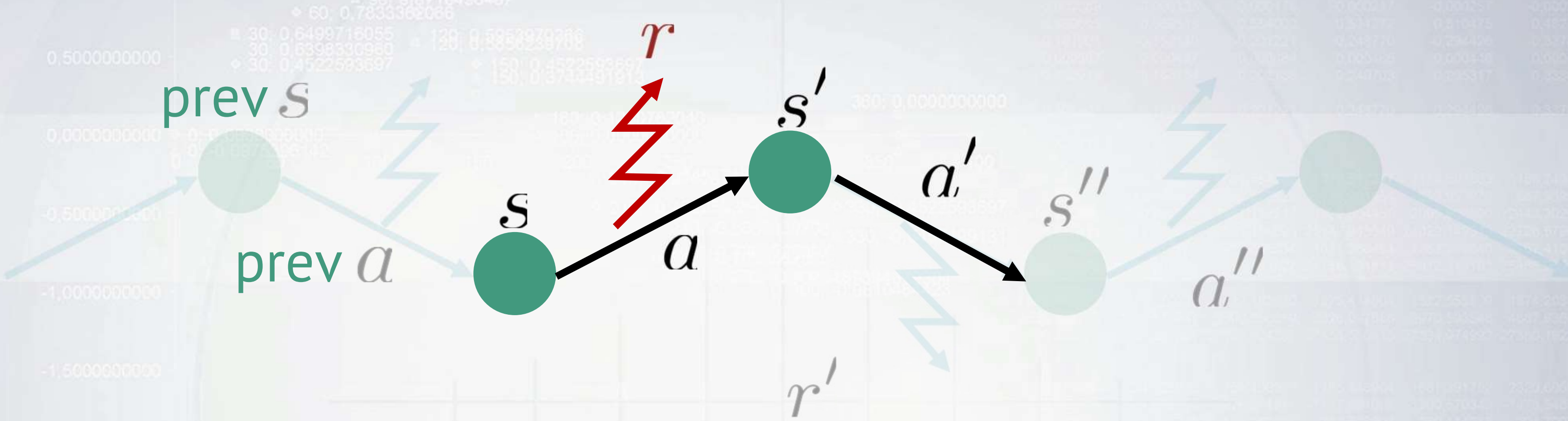
$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Initialize Q(s,a) with zeros

Loop:

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



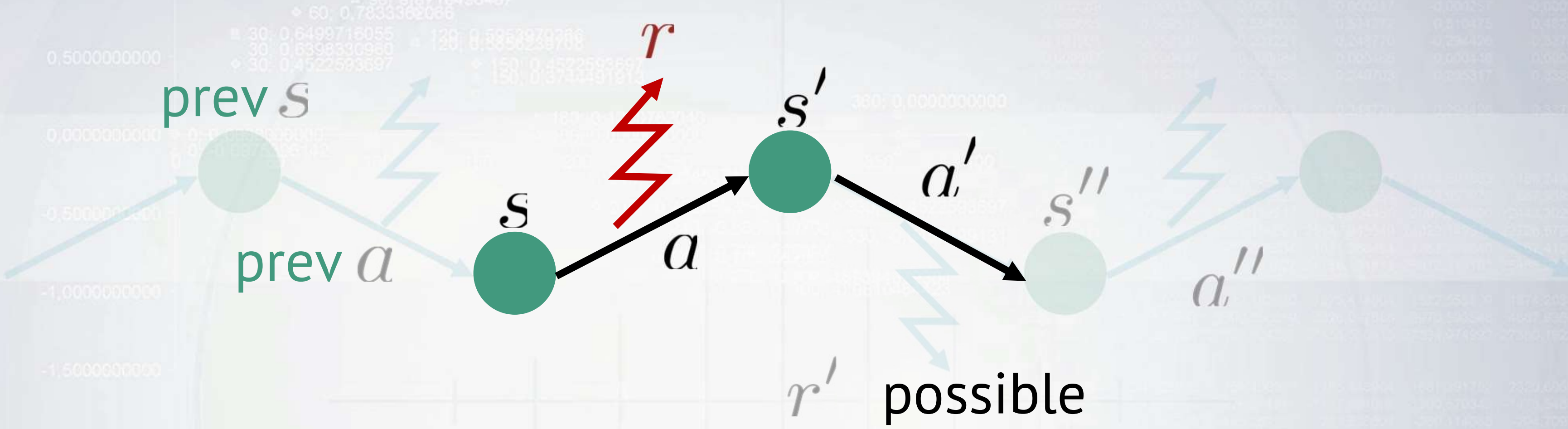
$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Initialize Q(s,a) with zeros

Loop:

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

SARSA



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

Initialize Q(s,a) with zeros

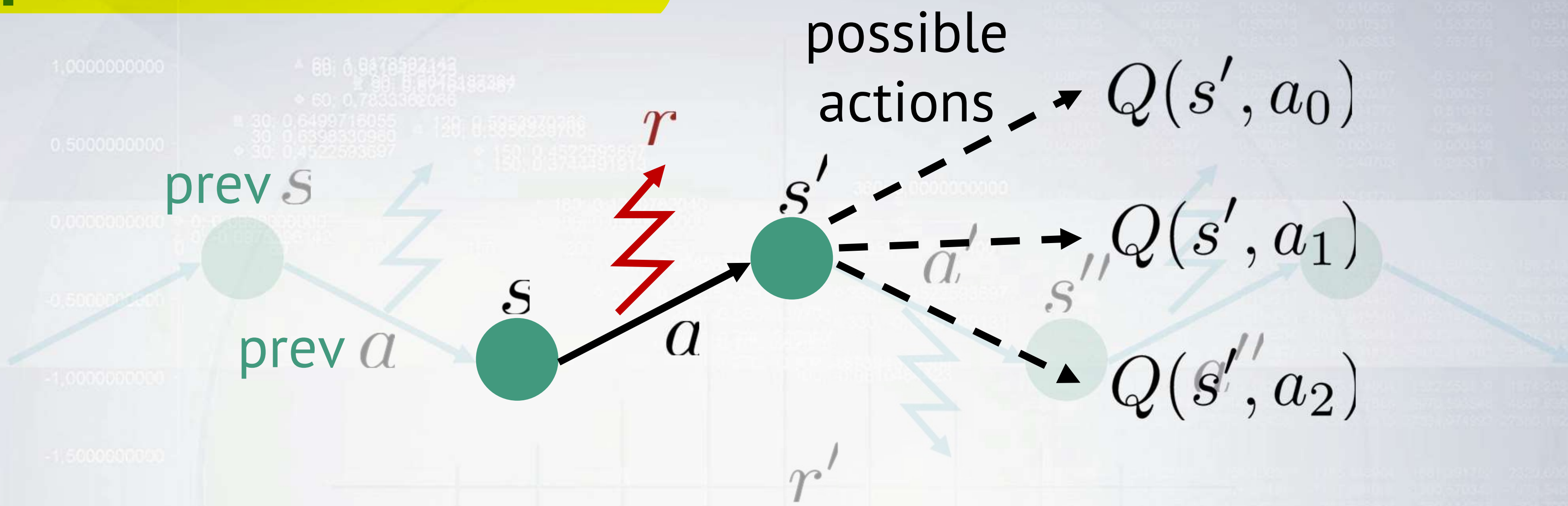
Loop:

hence “SARSA”

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

next action (not max)

Expected value SARSA



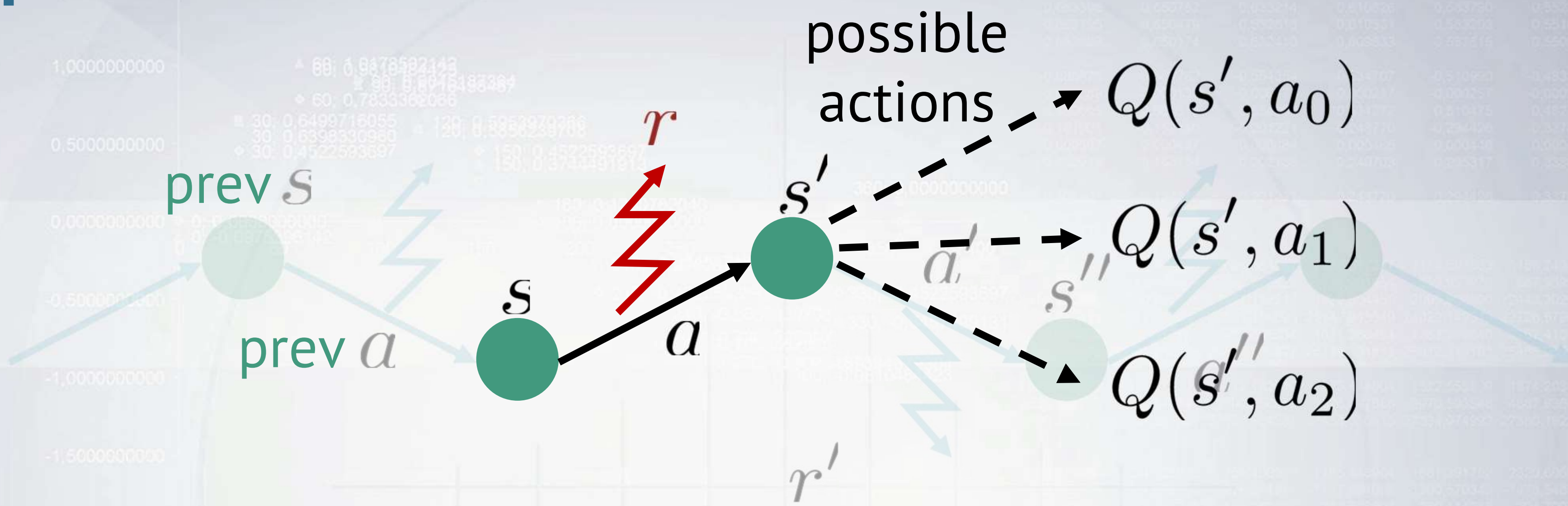
$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Initialize Q(s,a) with zeros

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



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

Initialize Q(s,a) with zeros

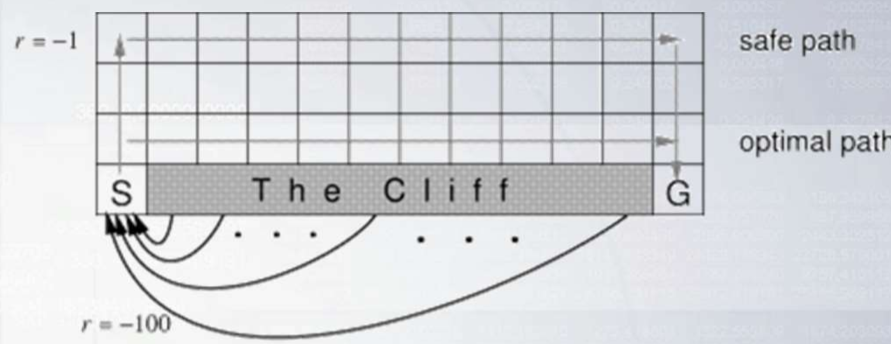
Loop:

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

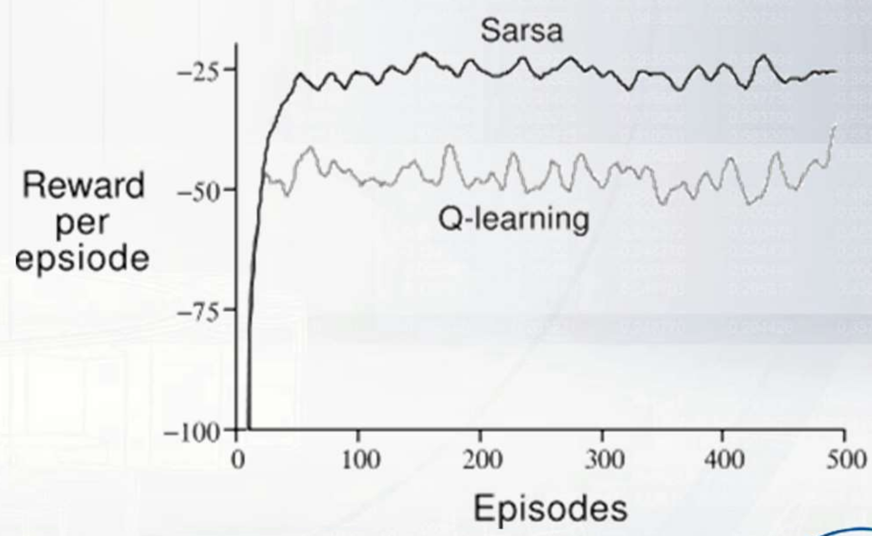
Expected value

Difference

- SARSA gets optimal rewards under current exploration strategy



- Q-learning policy **would be** optimal without exploration



Авторство

On-policy vs Off-policy

Two problem setups

on-policy	off-policy
Agent **can** pick actions	Agent **can't** pick actions
Most obvious setup :)	Learning with exploration, playing without exploration
Agent always follows his **own** policy	Learning from expert (expert is imperfect)
	Learning from sessions (recorded data)

* agent is trained here on diff policy than the current one

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
(but they be faster/better)	learn optimal policy even if agent takes random actions

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

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

Trivia: will Crossentropy Method converge if it learns off-policyfrom agent that takes random actions?

Well, no!

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

Trivia: will Crossentropy Method converge if it learns off-policyfrom agent that takes random actions?

Well, no :)

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

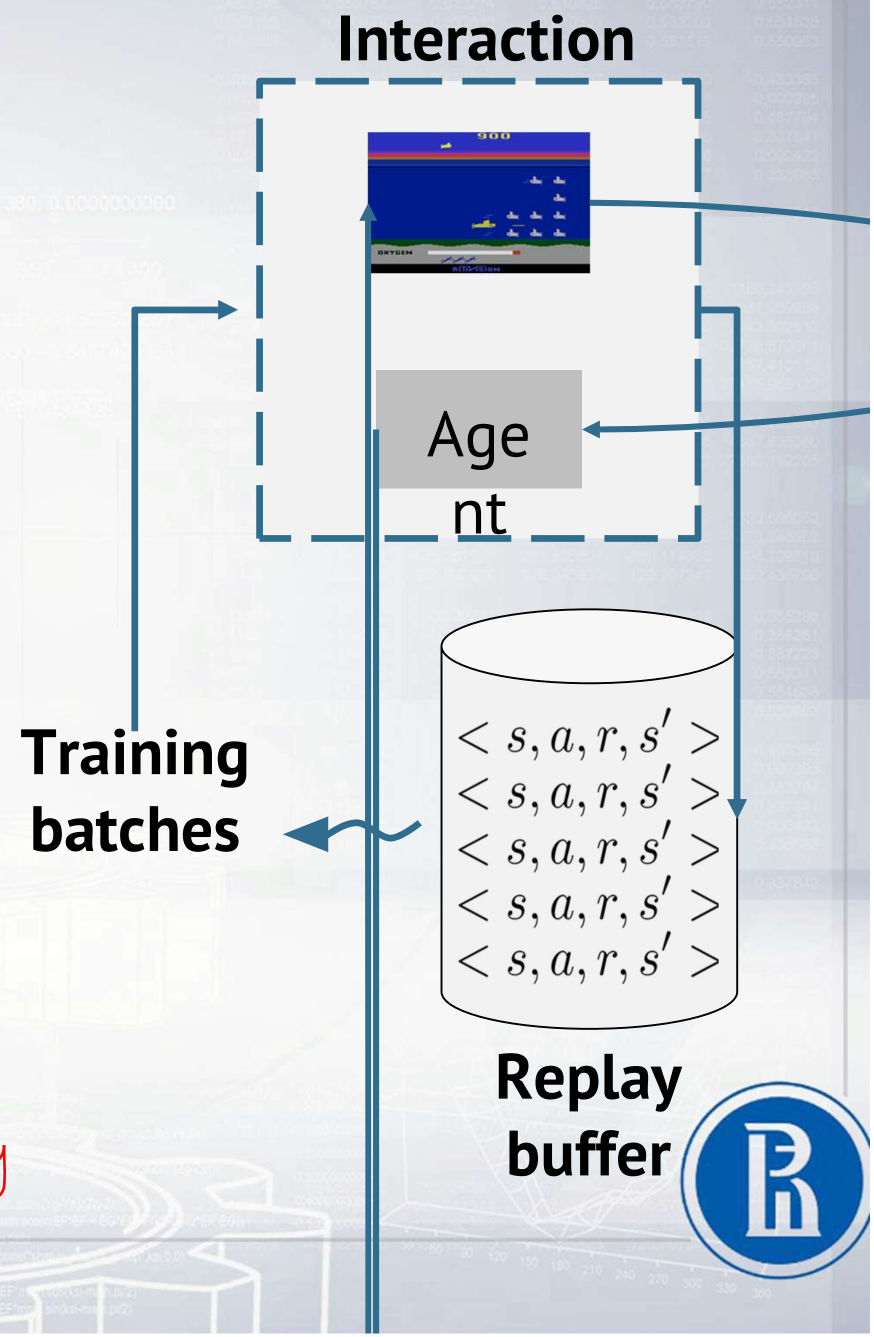
Experience replay

Idea: store several past interactions $< s, a, r, s' >$
Train on random subsamples

* Agent is trained not only on immediate state actual rewards, but it can actually store its prev interactions

Hence, sampling can be done from this large pool.

Obviously, however, such samples are probably worse than your current policy



Experience replay

Idea: store several past interactions $< s, a, r, s' >$
Train on random subsamples

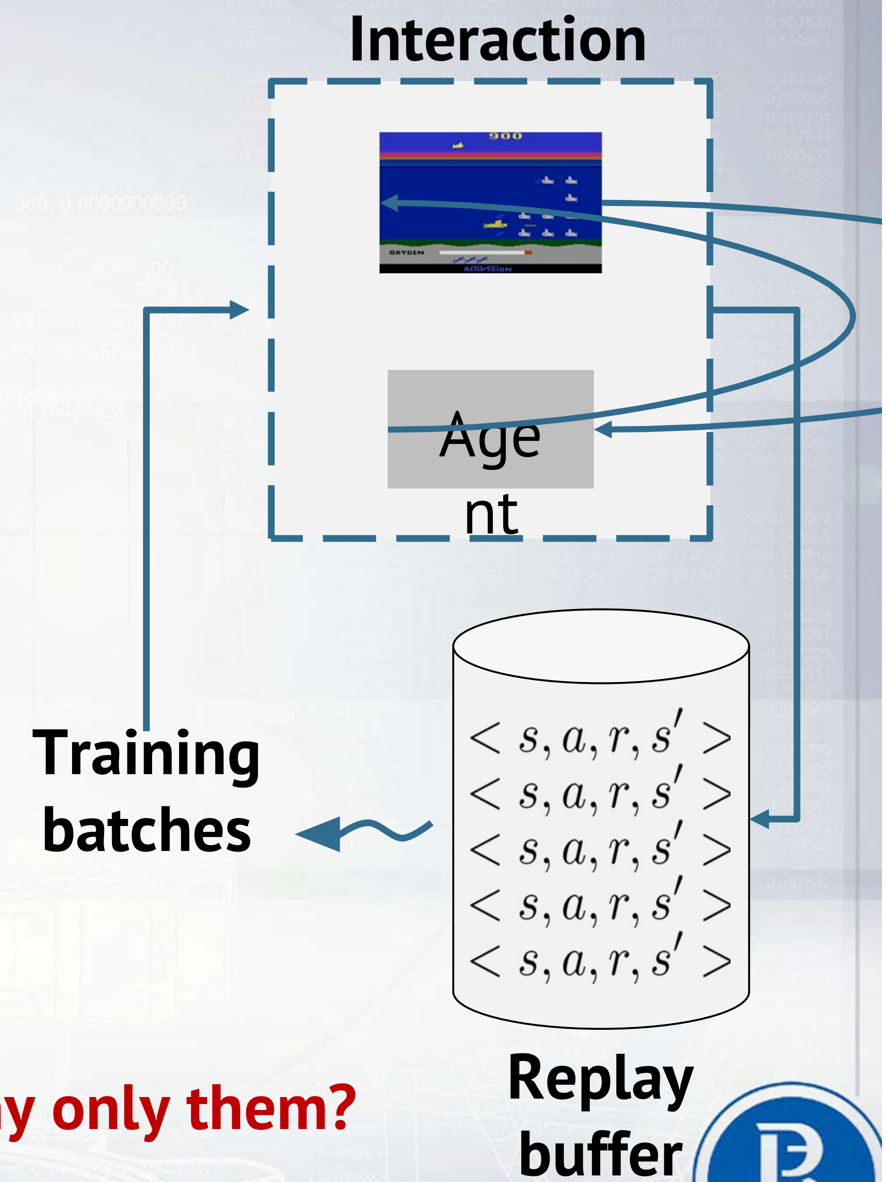
Training curriculum:

- 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?

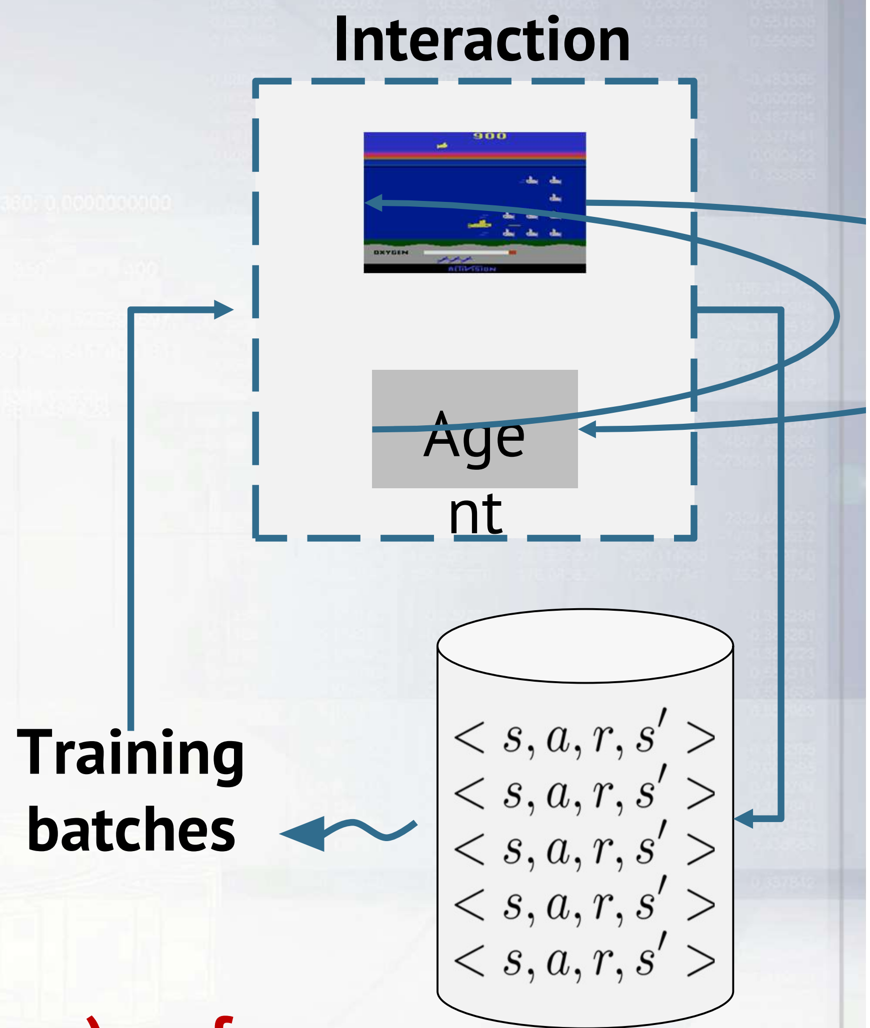


Experience replay

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) arefrom older/weakerversion of policy!

Pasha's TD(lambda) here

Let's write some code!