



Lecture 03: Model free learning

Radoslav Neychev

1. Value-function and Q-function recap
2. Model-free and model-based learning
3. Q-learning
4. Temporal difference
5. SARSA and EV-SARSA
6. Experience replay

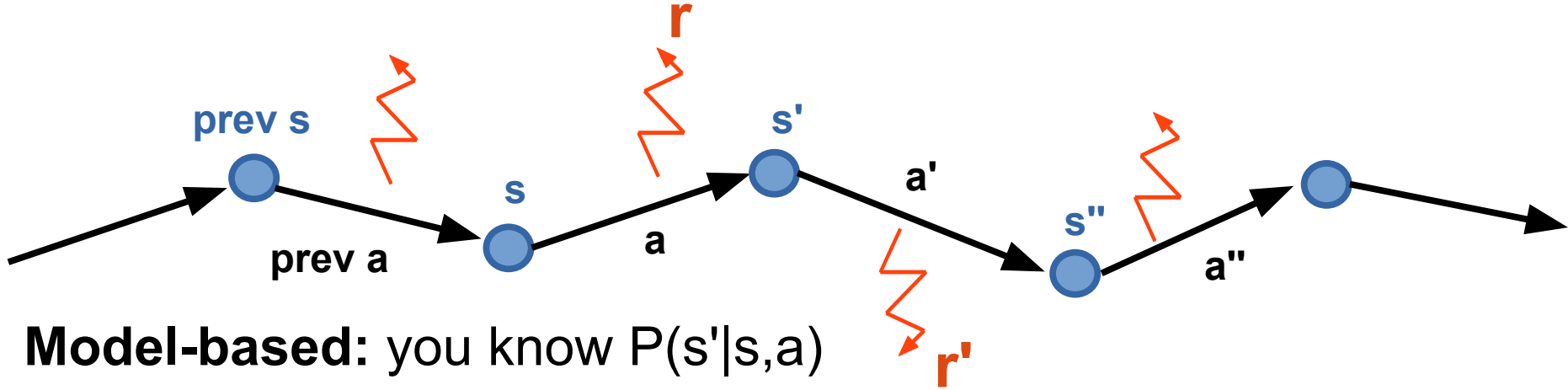
Based on: https://github.com/yandexdataschool/Practical_RL/

- $V_{\pi}(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π
- $V^*(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π^* – optimal
- $Q_{\pi}(\mathbf{s}, \mathbf{a})$ – 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 $\pi = \pi^*$ – optimal policy

$$Q^*(s, a) = E_{s', r} [r(s, a) + \gamma \cdot V^*(s')]$$

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

Learning from trajectories



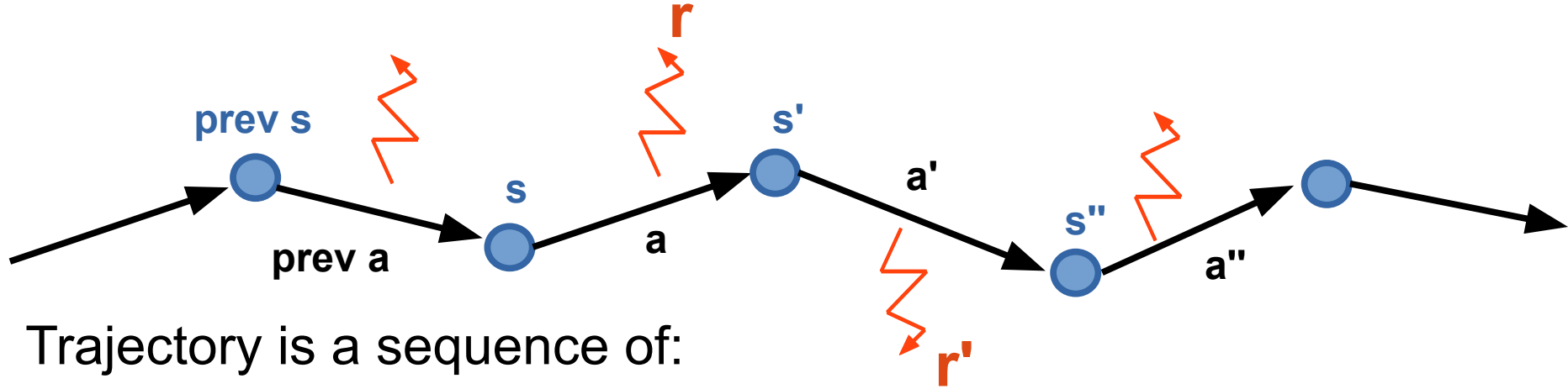
Model-based: you know $P(s'|s,a)$

- can apply dynamic programming
- can plan ahead

Model-free: you can sample trajectories

- can try stuff out
- insurance not included

Learning from trajectories



Trajectory is a sequence of:

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

Q: What to learn?

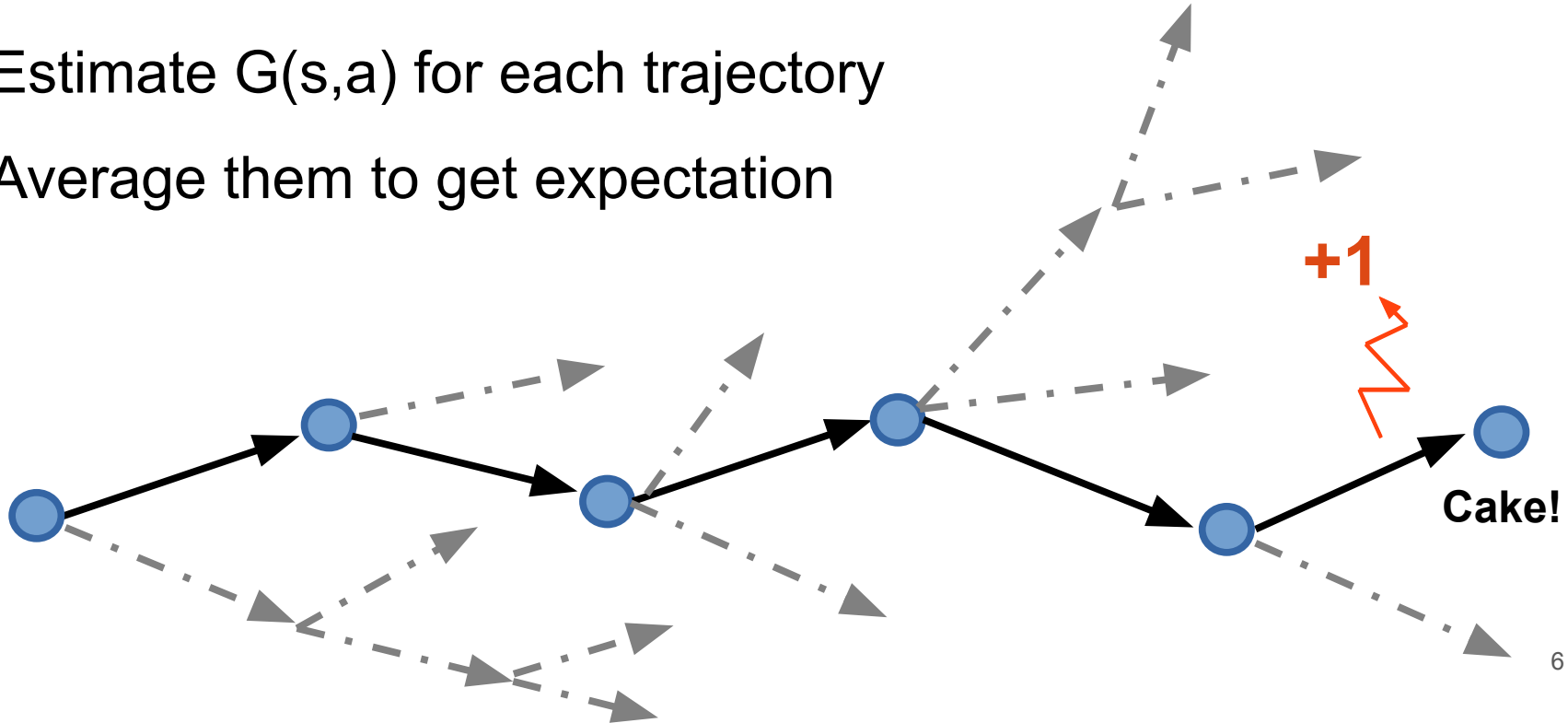
$V(s)$ or $Q(s,a)$

We can only sample trajectories

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

Idea 1: Monte-carlo

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



Idea 2: Temporal difference

- $Q(s, a)$ can be improved iteratively!

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

How to get the
expected value?

Idea 2: Temporal difference

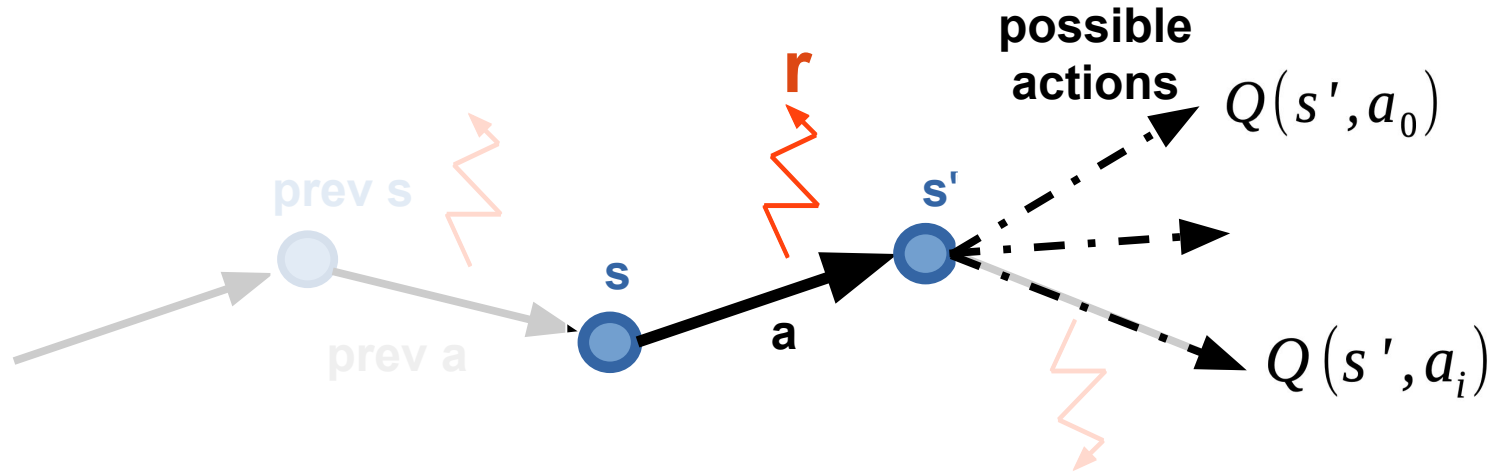
- $Q(s, a)$ can be improved iteratively!

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

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

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

Q-learning



Initialize $Q(s, a)$ with zeros

- Sample $\langle s, a, r, s' \rangle$ from the environment
- Compute new $Q(s, a)$ estimation:

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

- Update $Q(s, a)$:

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

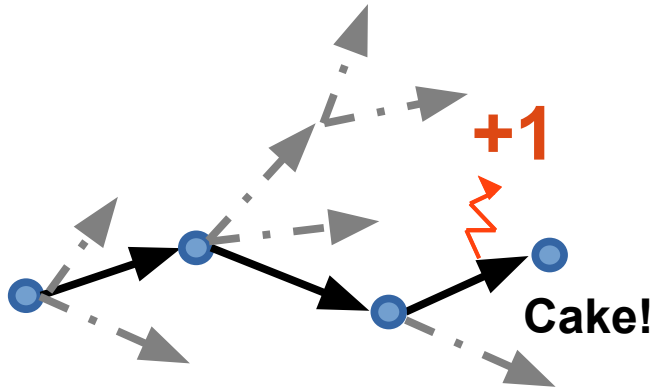
$$Q^*(s, a) = E_{s', r} [r(s, a) + \gamma \cdot V^*(s')]$$

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

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

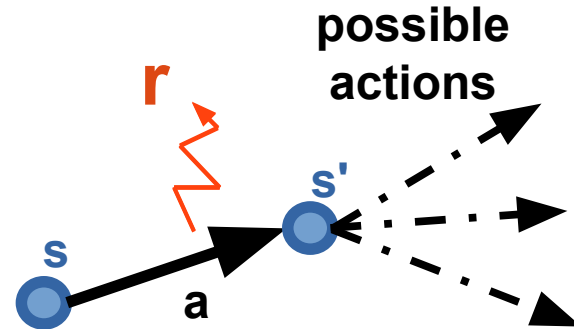
Monte-carlo

- Averages Q over sampled paths



Temporal Difference

- Uses recurrent formula for Q



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 trajectories
- Works with infinite MDP
- Requires less experience to learn

$$Q^*(s, a) = E_{s', r} [r(s, a) + \gamma \cdot V^*(s')]$$

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

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

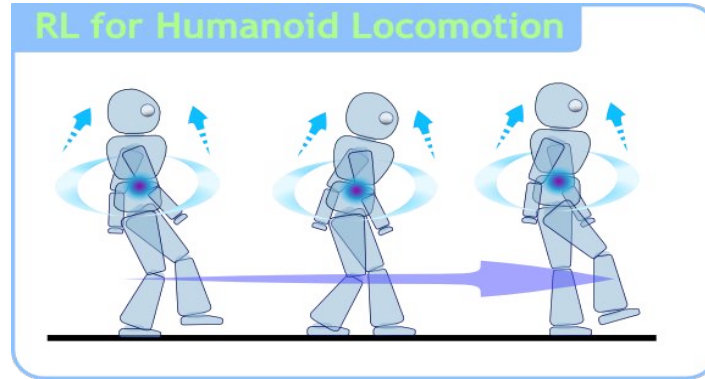
Exploration and exploitation

What if agent is greedy and it always selects the “best” (according to the Q-value) action?

It will not be able to find better actions!

Q-learning: problems

Imagine a robot learning to walk



Initial $Q(s,a)$ are zeros

Robot uses $\operatorname{argmax} Q(s,a)$

It has just learned to crawl with positive reward

Now it has no chance to learn how to walk

Exploration-exploitation tradeoff

Two potential approaches (for now):

- ϵ -greedy policy:
 - Select random action with ϵ probability, otherwise select best according to the current Q-function
- Softmax:
 - Sample action from a distribution generated by softmax normalization of Q-values:

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

Example: Cliff-world

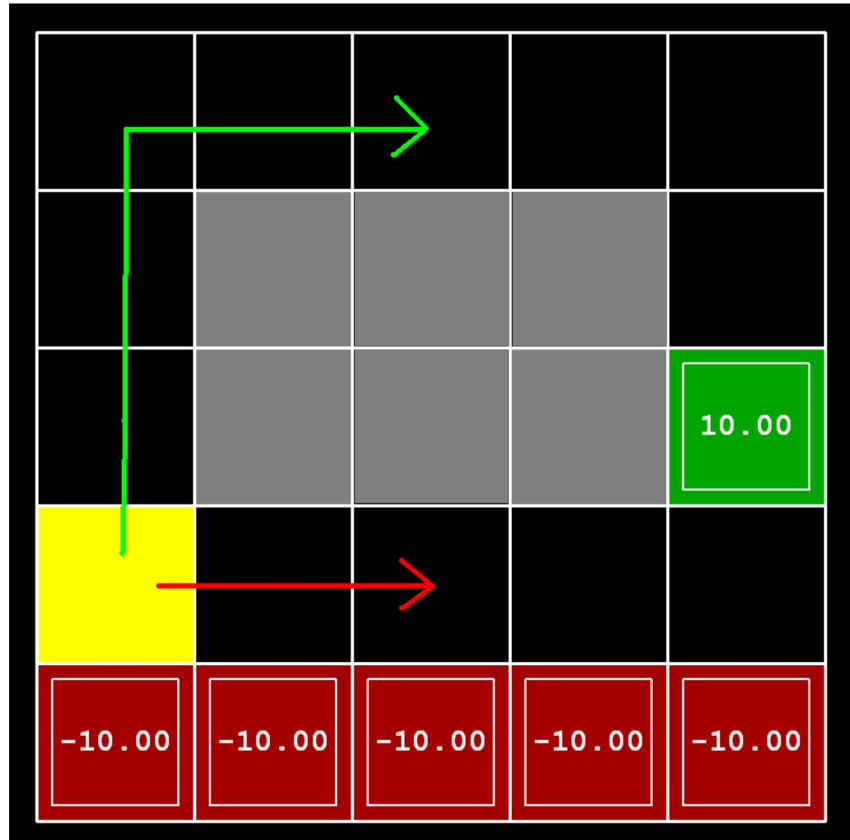
No stochasticity in the environment, ϵ -greedy policy

Let $\epsilon = 0.15$, $\gamma = 0.99$

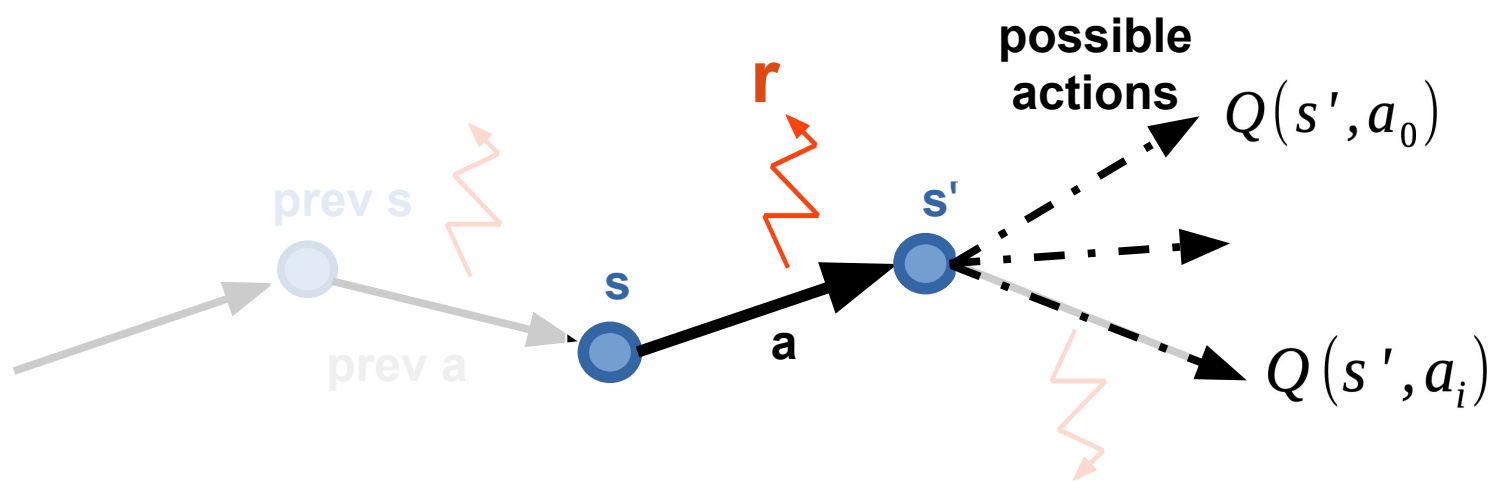
Which trajectory will Q-learning prefer?

The red one!

Despite it will not maximize the reward (due to ϵ -greedy behaviour)



Q-learning

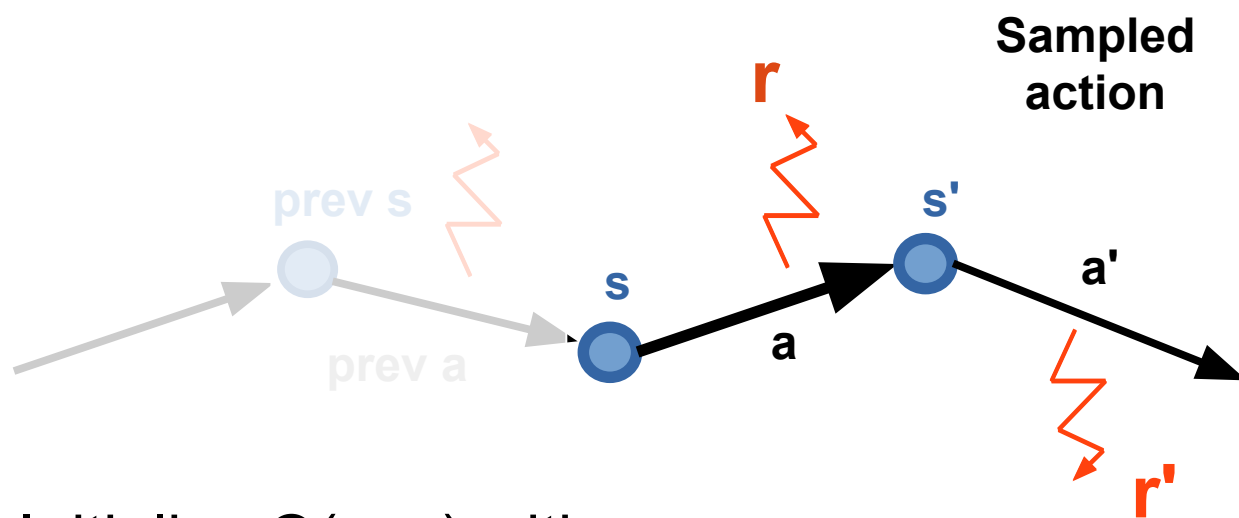


Initialize $Q(s, a)$ with zeros

- Sample $\langle s, a, r, s' \rangle$ from the environment
- Compute new $Q(s, a)$ estimation:

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

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



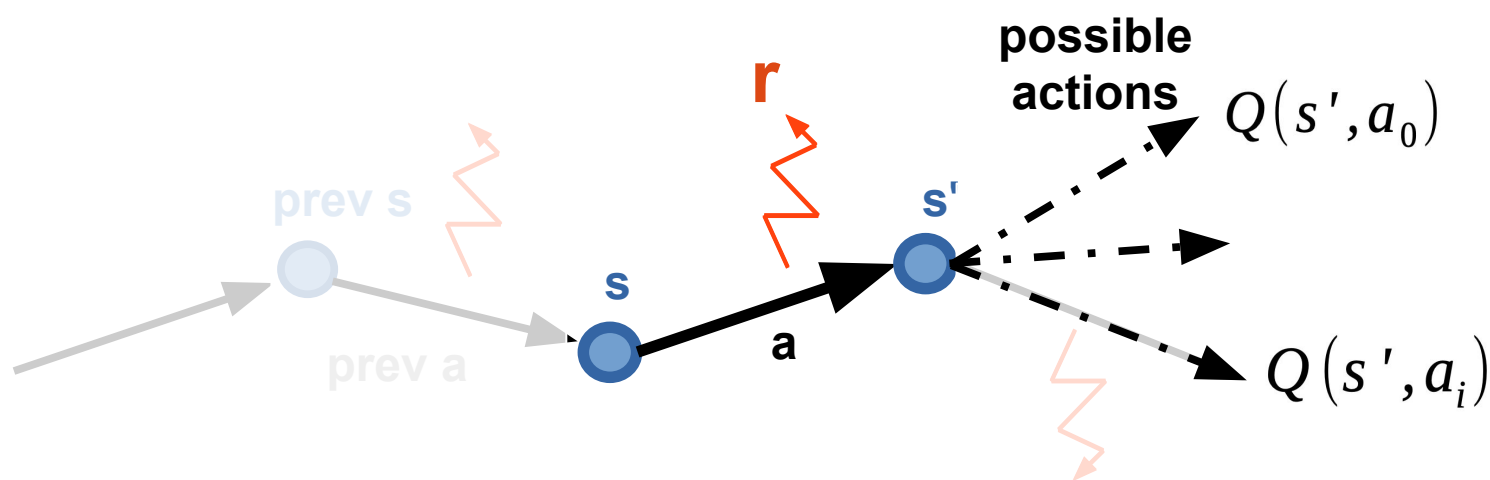
Initialize $Q(s, a)$ with zeros

- Sample $\langle s, a, r, s', a' \rangle$ from the environment
- Compute new $Q(s, a)$ estimation:

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

- Update $Q(s, a)$:

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



Initialize $Q(s, a)$ with zeros

- Sample $\langle s, a, r, s' \rangle$ from the environment
- Compute new $Q(s, a)$ estimation:

- Update $Q(s, a)$:

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

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

On-policy

- Agent trains on experience generated with its own policy
- Can't learn off-policy

Examples:

- Cross-entropy method
- SARSA

Off-policy

- Agent trains on any kind of experience
- Can still learn on-policy

Examples:

- Q-learning
- EV-SARSA

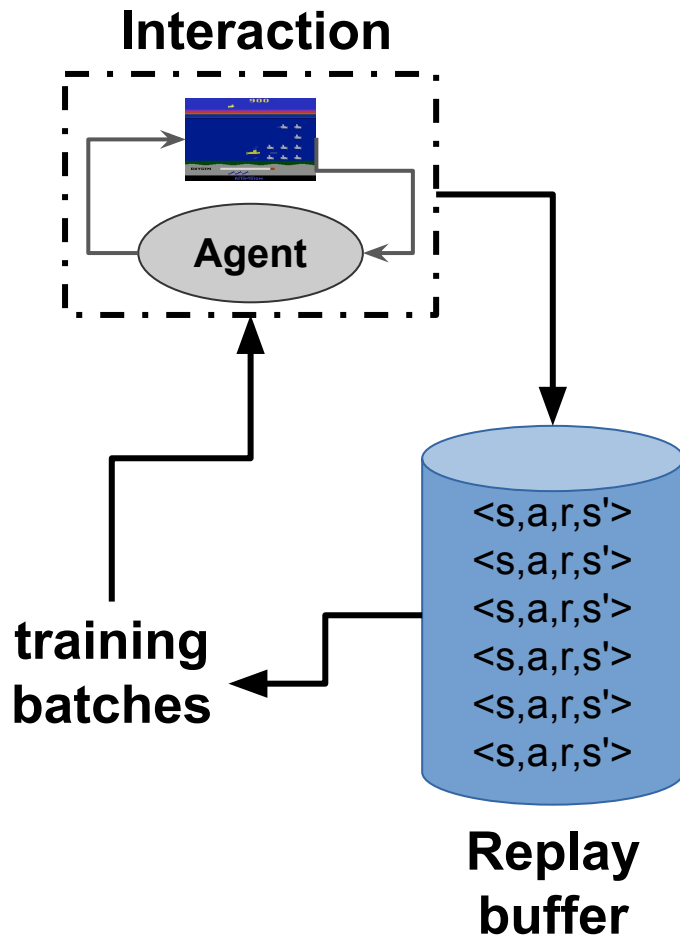
Idea:

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

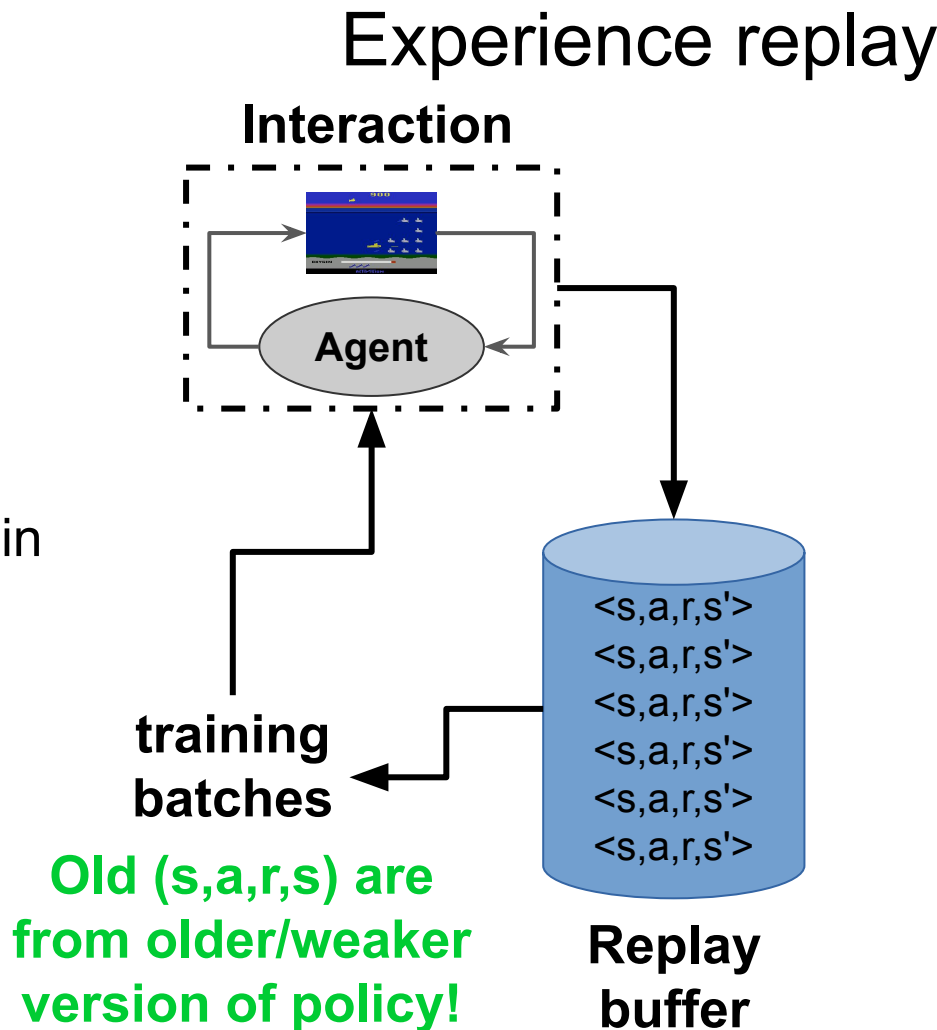
Training curriculum:

- Play 1 step and record it
- Pick N random transitions to train

Profit:

you don't need to revisit same (s, a)
many times to learn it.

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



- Q-learning allows to learn some approximation of the reward function and the environment model
 - So we can use it to solve the desired problem
- 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!