girafe
ai

Lecture 11: Model free learning

Radoslav Neychev

Outline

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/

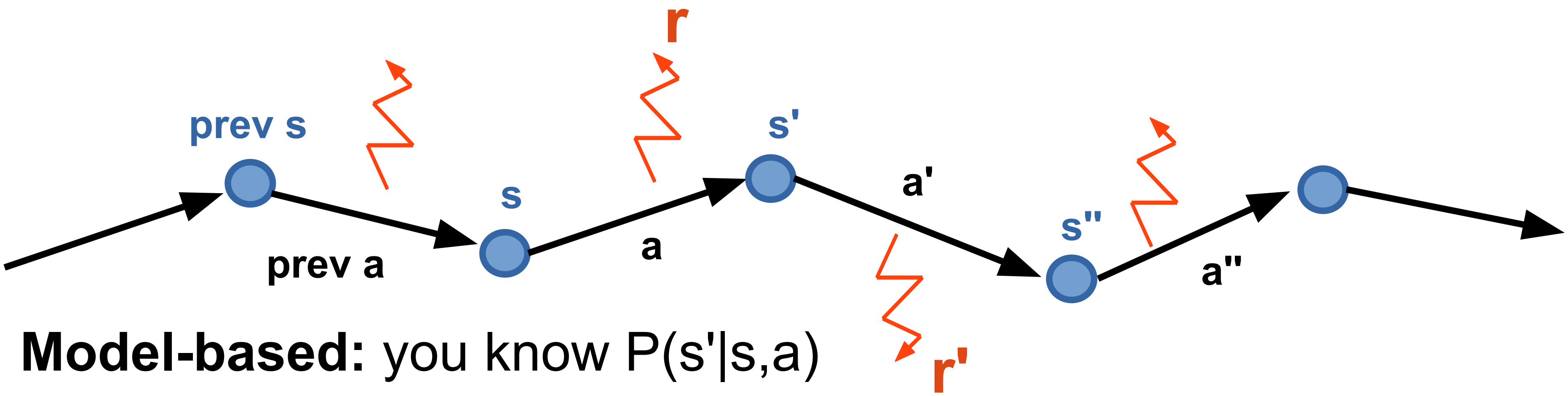
Recap

- **Vπ(s)** – expected G from state **s** if you follow **π**
- **V*(s)** – expected G from state s if you follow **π*** – optimal
- **Qπ(s,a)** – expected G from state **s**
  - if you start by taking action **a**
  - and follow **π** from next state on
- **Q*(s,a)** – same as Qπ(s,a) where **π** = **π*** – optimal policy

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

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

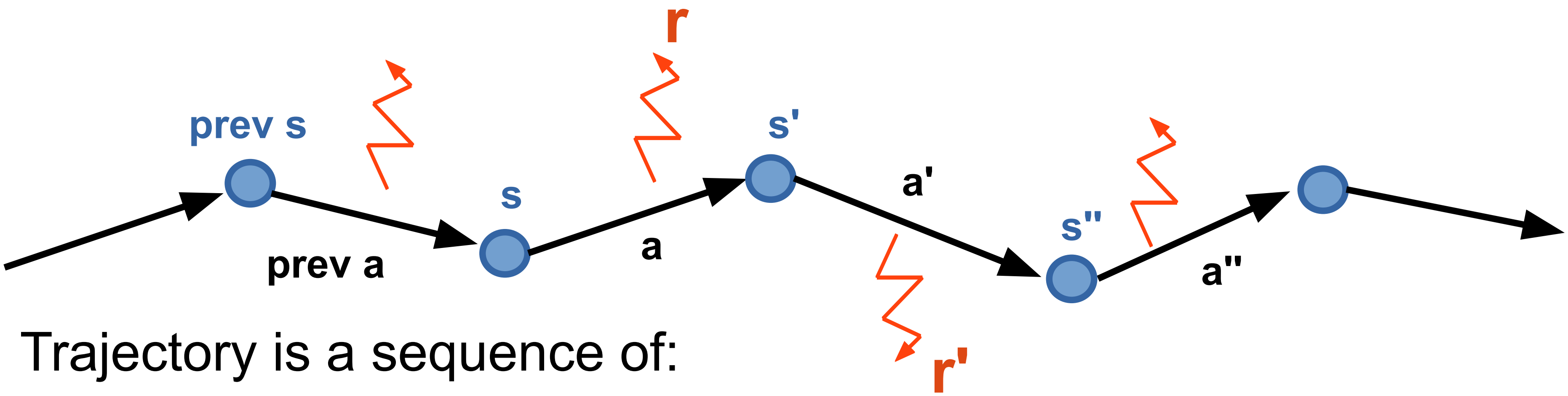
Learning from trajectories



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)

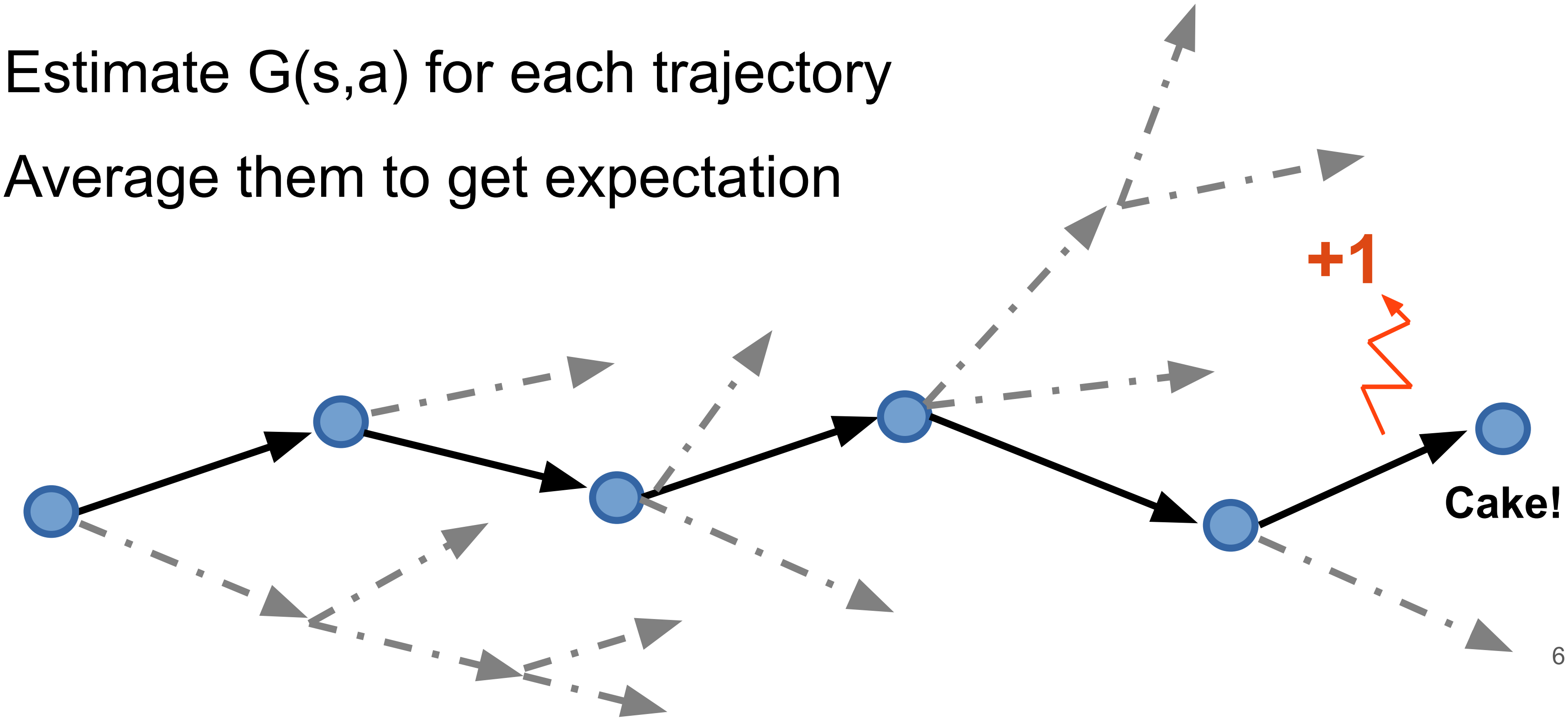
We can only sample trajectories

Q: What to learn?
V(s) or $\boxed{Q(s,a)}$

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!

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

How to get the expected value?

Idea 2: Temporal difference

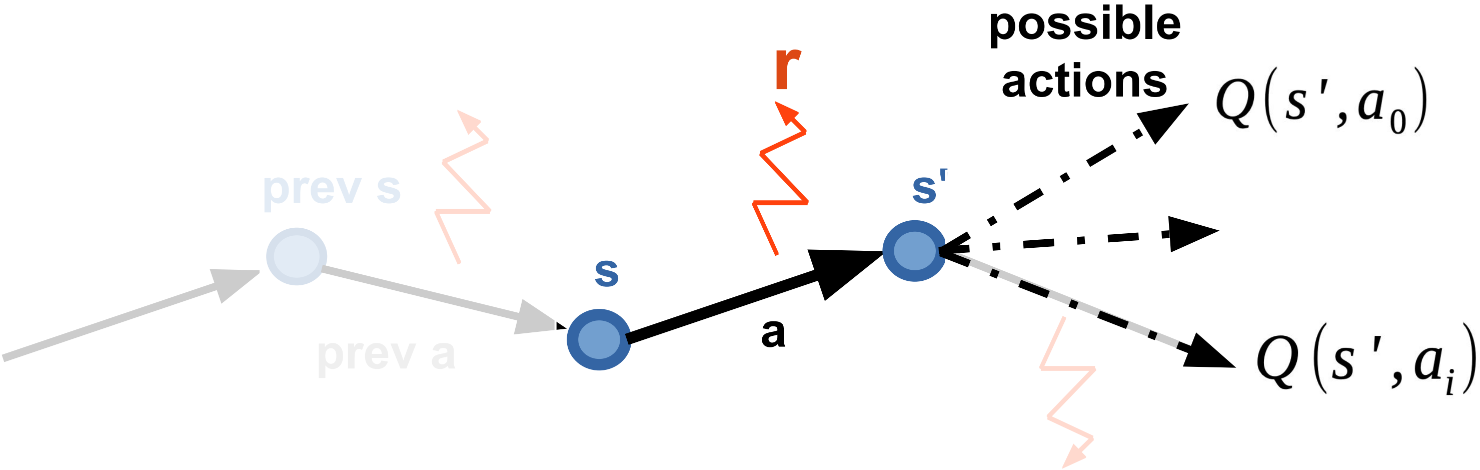
- Q(s, a) can be improved iteratively!

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

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

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

Q-learning



Initialize Q(s, a) with zeros

- Sample <s, a, r, s’> from the environment
- Compute new Q(s, a) eslimation:

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

- Update Q(s, a):

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

Q-learning

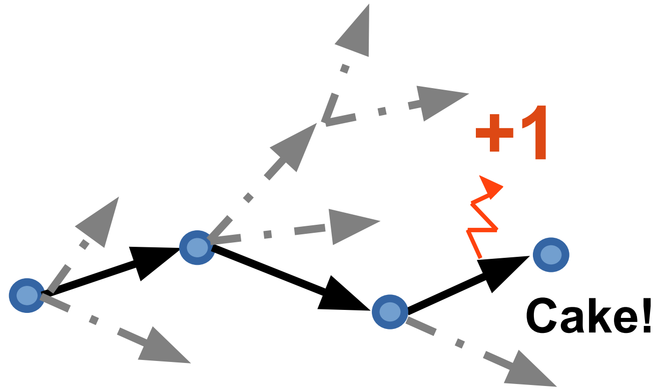
$$Q^*(s,a) = \underset{s',r}{E}\, r(s,a) + \gamma \cdot V^*(s')$$

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

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

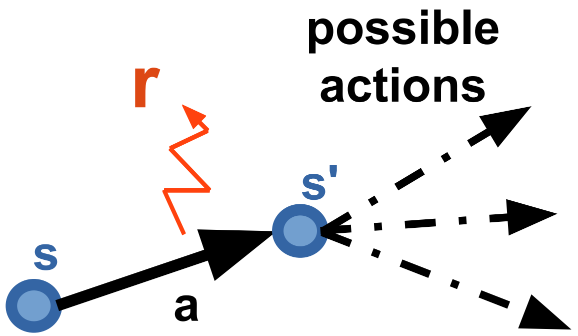
Monte-carlo

- 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-learning

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

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

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

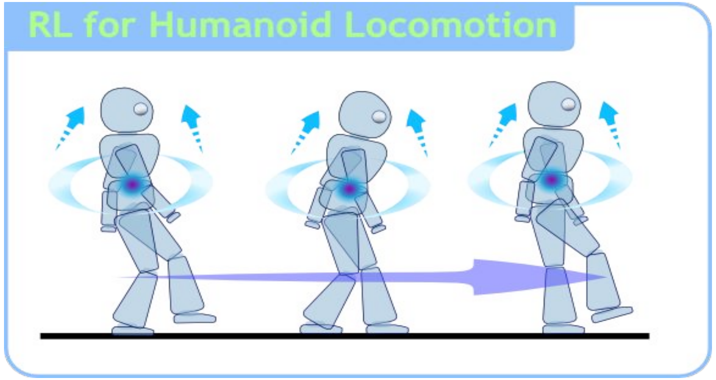
Exploration and exploitation

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 argmax Q(s,a)

It has just learned to crawl with positive reward

Now it has no chance to learn how to walk

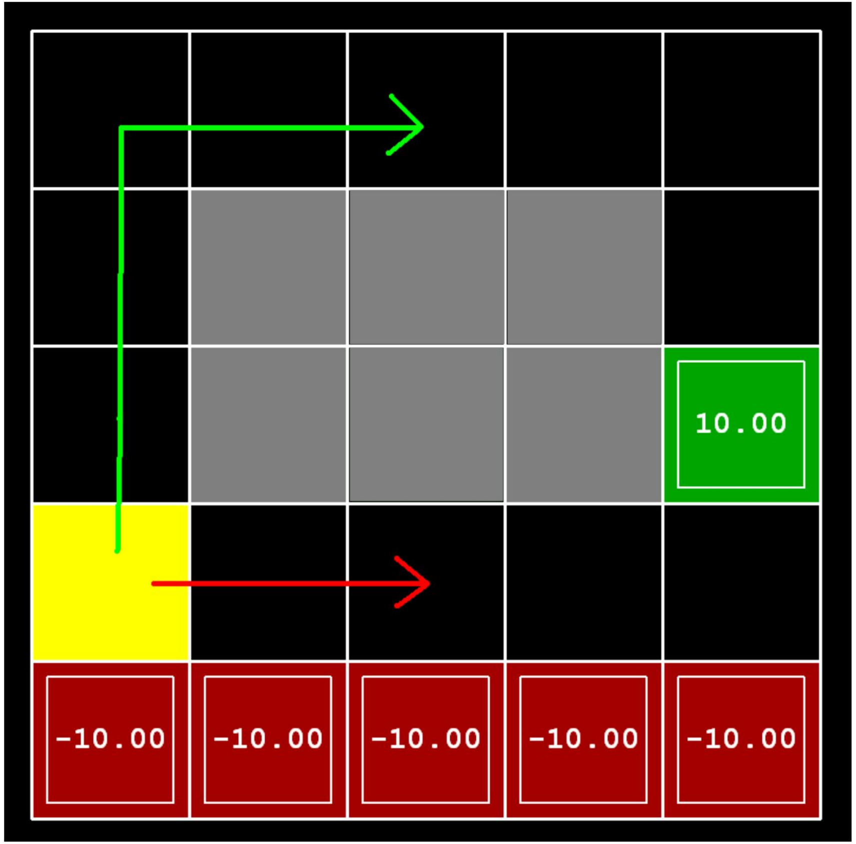
Exploration-exploitation tradeoff

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:

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

Example: Cliff-world



No stochasticity in the environment, ε-greedy policy

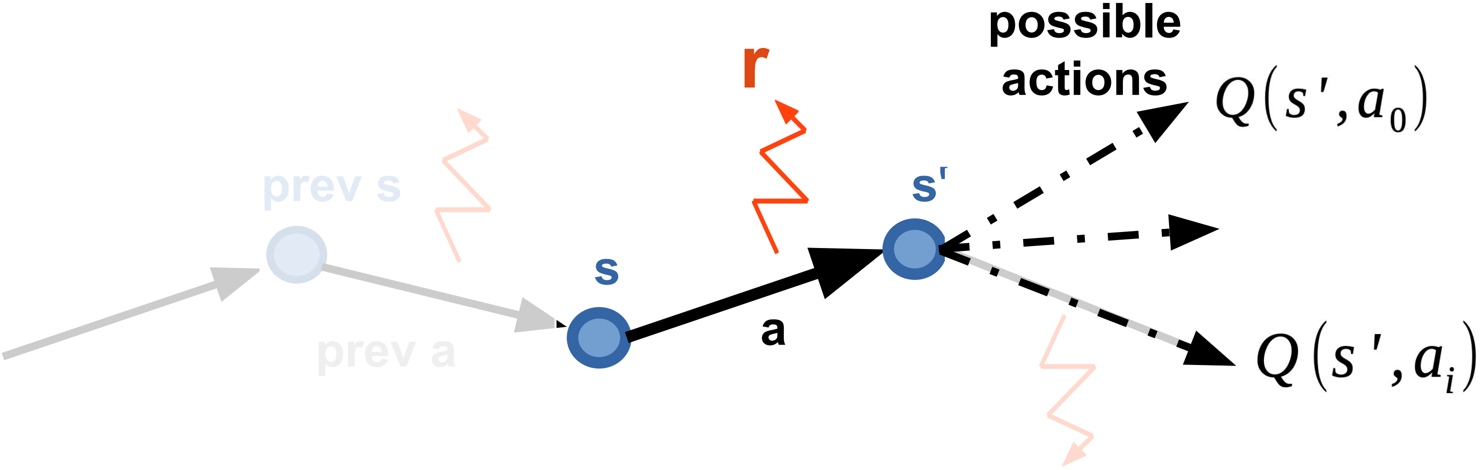
Let $\varepsilon = 0.15$, $\gamma = 0.99$

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

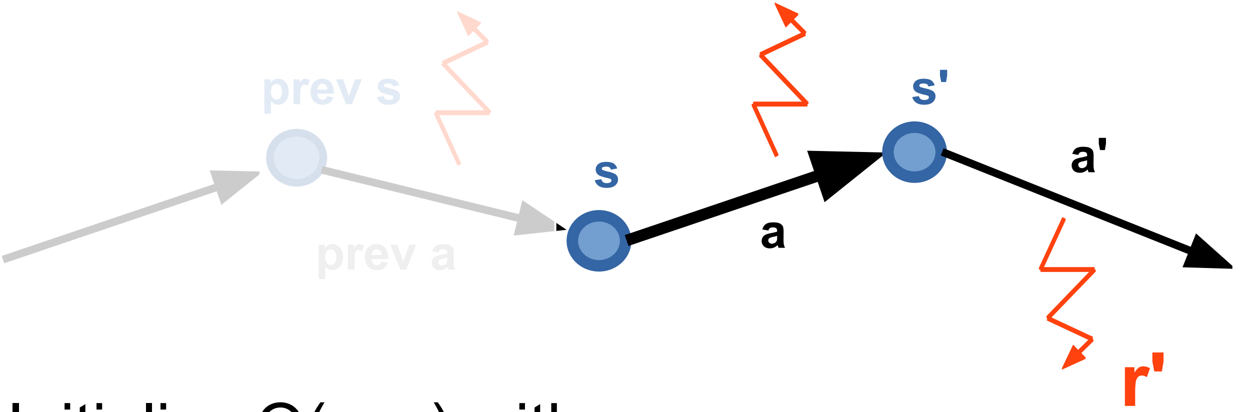
- Sample <s, a, r, s'> from the environment
- Compute new Q(s, a) eslimation:

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

- Update Q(s, a):

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

SARSA



Initialize Q(s, a) with zeros

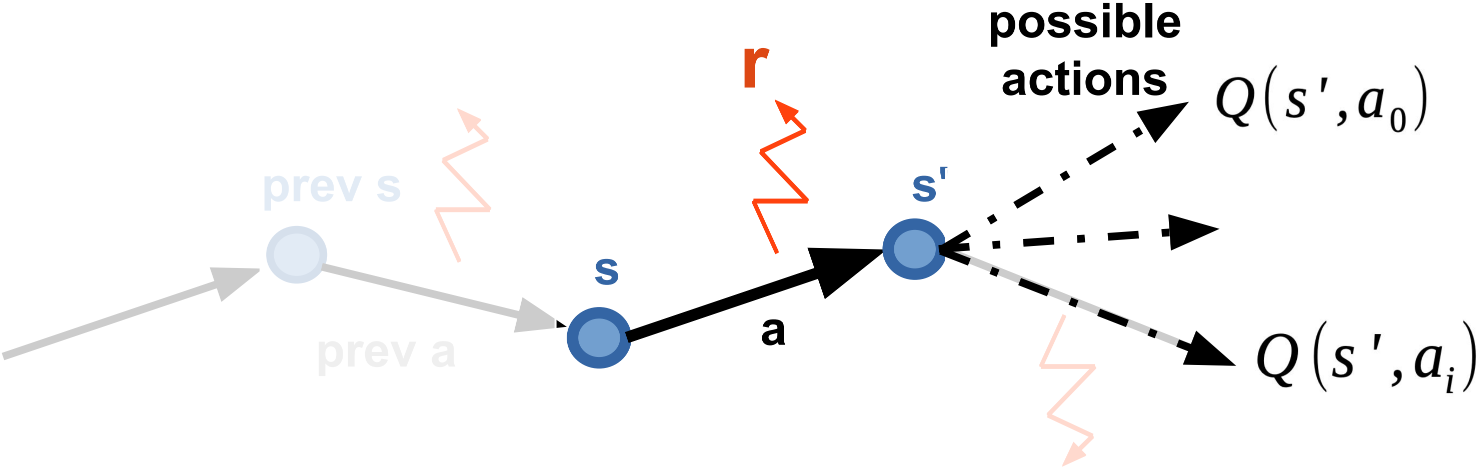
- Sample <s, a, r, s', a'> from the environment
- Compute new Q(s, a) eslimation:

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

- Update Q(s, a):

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

EV-SARSA



Initialize Q(s, a) with zeros

- Sample <s, a, r, s’> from the environment
- Compute new Q(s, a) eslimation:
- Update Q(s, a):

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

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

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

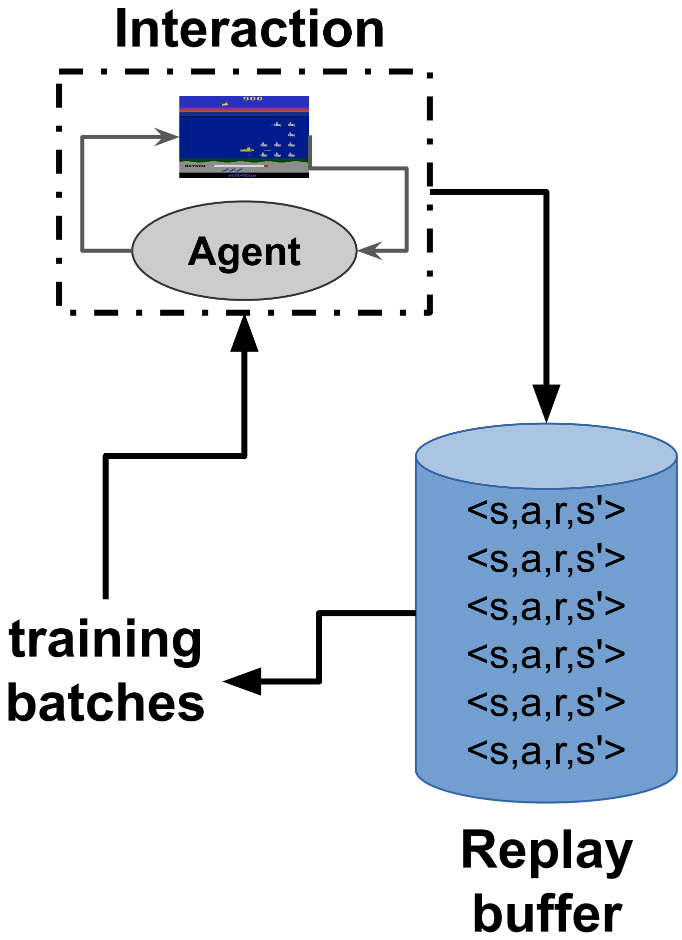
Examples:

- Q-learning
- EV-SARSA

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

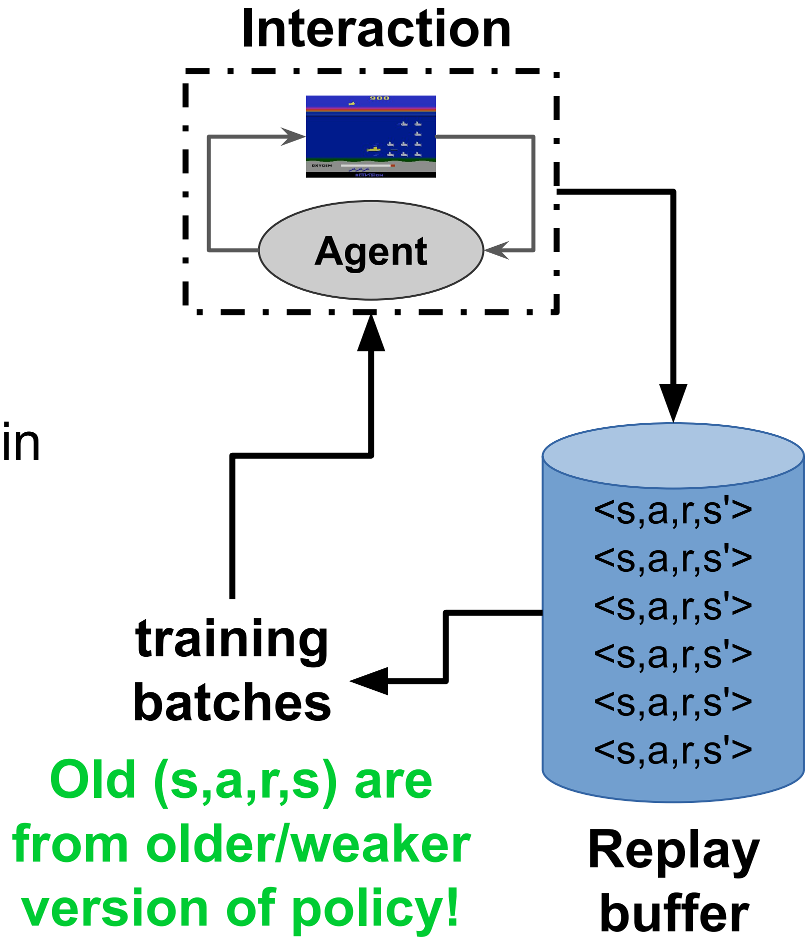
Training curriculum:

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

Profit:

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

Only works with off-policy algorithms!



Outro

- 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!