Lecture 1.3

Reinforcement Learning: Model-Free RL methods

Alexey Gruzdev
alexey.s.gruzdev@gmail.com
HSE, Winter 2019

Recap: Dynamic Programming

- $v_\pi(s)$, $v_*(s)$

- If you know $v_*(s)$, $p(r, s' \mid s, a)$ → **know optimal policy**

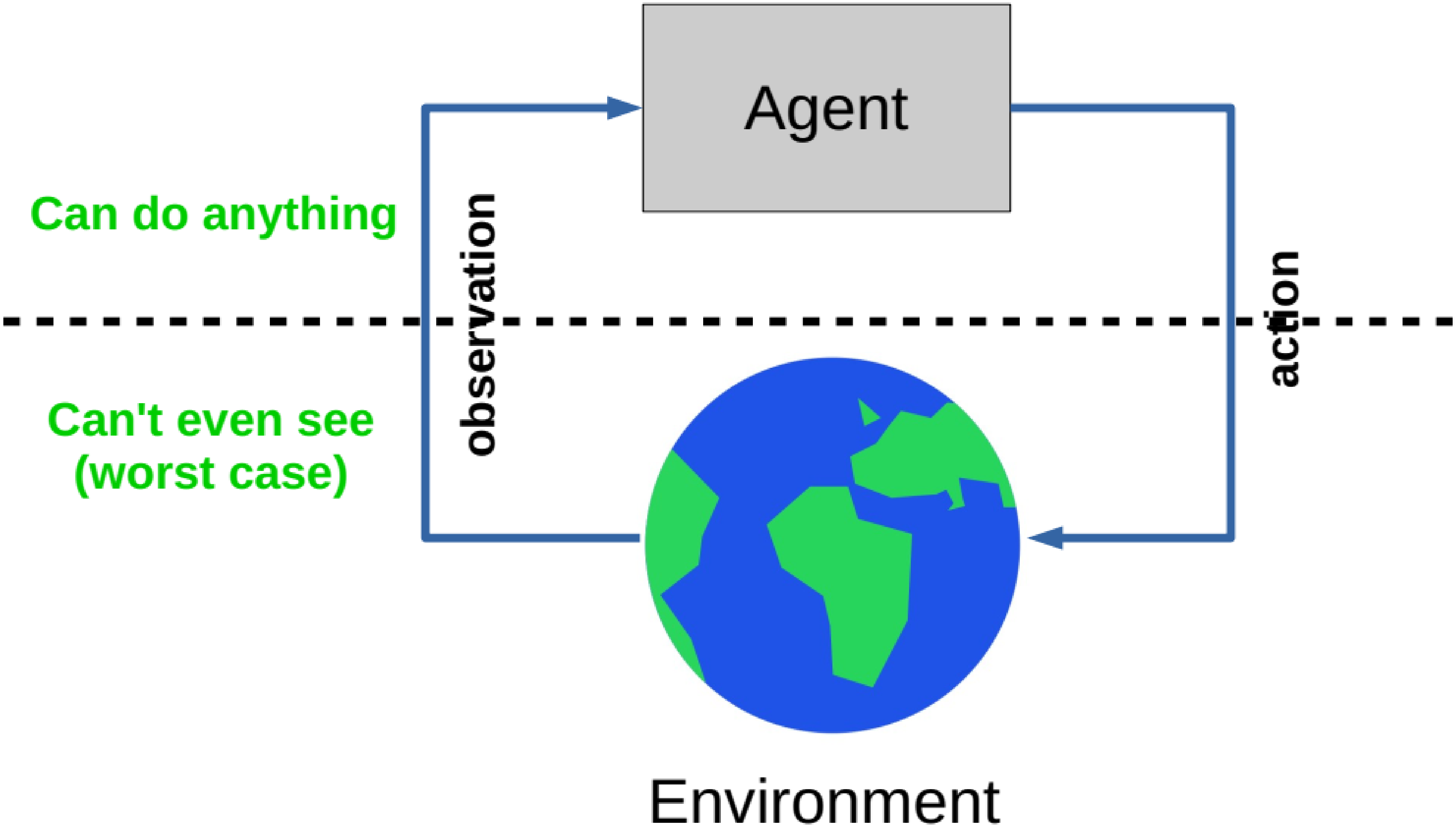
- We can learn $v_*(s)$ with Dynamic Programming:

$$v_*(s) = \max_a \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma v_*(s')]$$

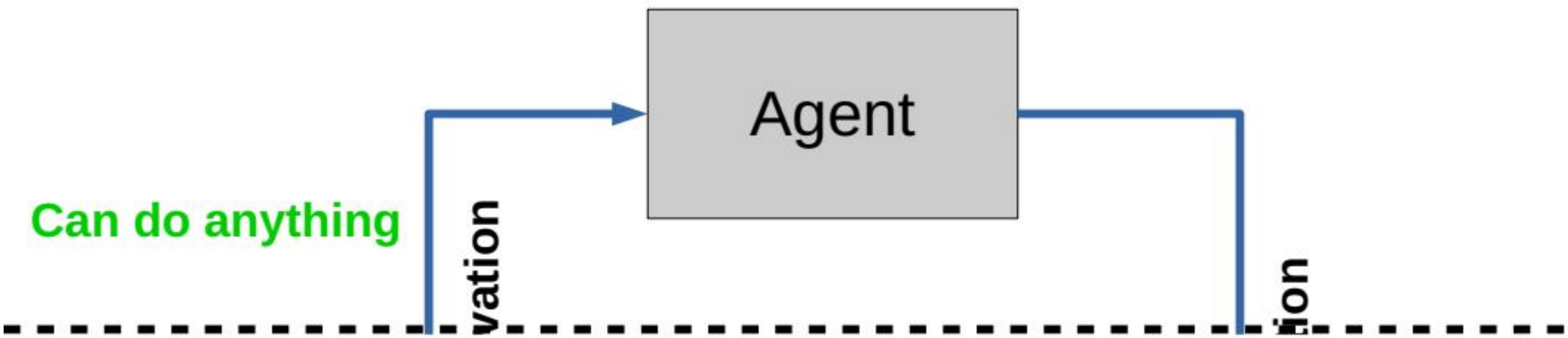
- $q_\pi(s, a)$, $q_*(s, a)$

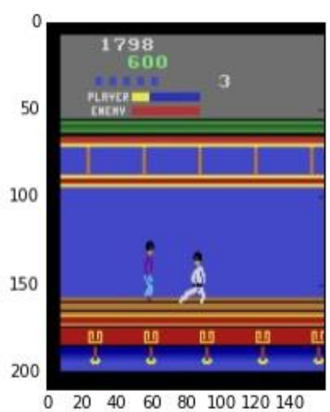
$$q_*(s, a) = \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma \max_{a'} q_*(s', a')]$$

Decision making: reality check



Decision making: reality check





Model-Free Setup

- We don't know internal environment representation, e.g.

$$\boldsymbol{p}(\boldsymbol{r}, \boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \text{ - unknown}$$

What should we do?

Let's count all letters that we introduced

- $a, r, s, p(r, s' | s)$

- $G_t(s)$

- $v_\pi(s), v_*(s)$

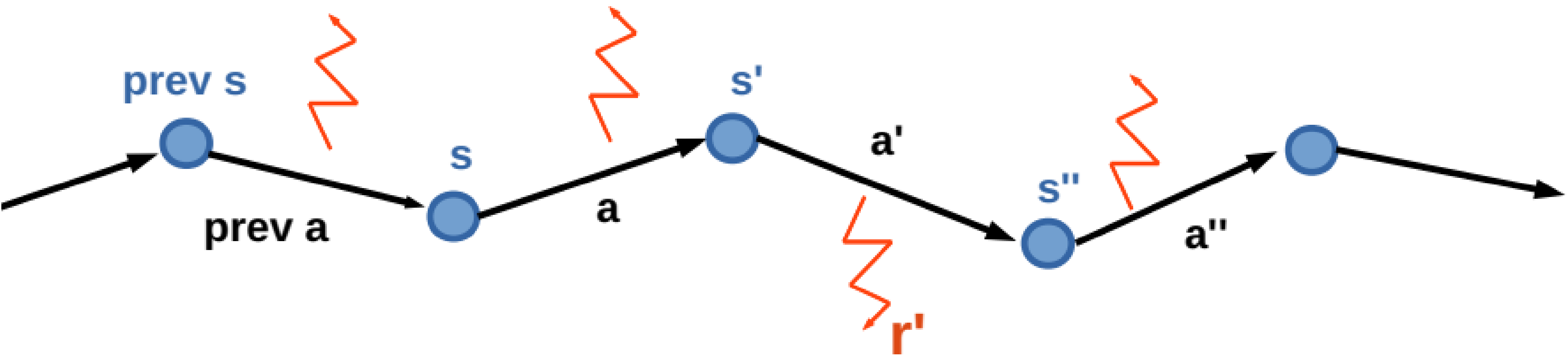
- $q_\pi(s, a), q_*(s, a)$

- π, π^*

- *Not enough? Next lectures we'll fix that!*

Learning from trajectories

- $s_1 \text{ -> } a_1 \text{ -> } r_1 \text{ -> } s_2 \text{ -> } \ldots \text{ -> } s_n$ – *trajectory*



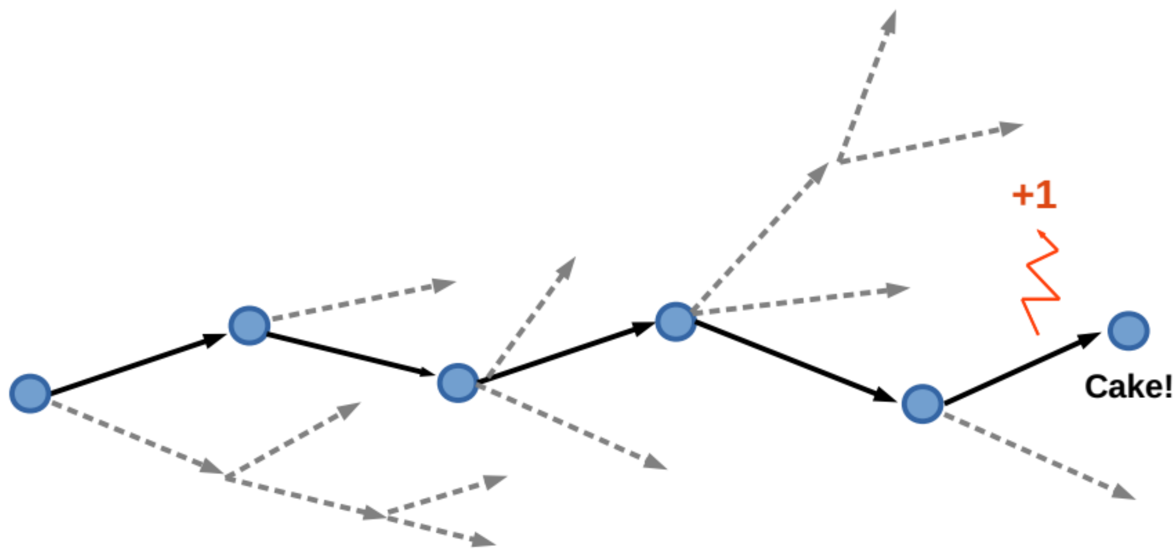
- Model-based setup:
  - you can apply Dynamic Programming
  - you can plan (!)
- Model-free setup:
  - you can experiment with different actions
  - no guaranties (!!!)

Learning from trajectories

- $s_1 \text{ -> } a_1 \text{ -> } r_1 \text{ -> } s_2 \text{ -> } \ldots. \text{ -> } s_n$ – *trajectory*
- *We can sample trajectories (a lot of trajectories!)*
- *What should we learn ?*
  - $p(r, s' \mid s, a)$
  - $v_\pi(s)$
  - $q_\pi(s, a)$

Monte-Carlo RL

- Just like N+1 heuristic:
  - Get all trajectories containing particular (s, a)
  - Estimate $G_t(s, a)$ for each trajectory
  - Average them to get *estimation* of expectation



Monte-Carlo RL

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- <u>Note:</u> can only apply MC to *episodic* MDPs
  - All episodes must terminate

Incremental Mean

- The mean $\mu_1, \mu_2, \ldots, \mu_k$ of a sequence $x_1, x_2, \ldots, x_k$ can be computed incrementally:

$$
\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} \left(x_k + (k-1)\mu_{k-1} \right) \\
&= \mu_{k-1} + \frac{1}{k} \left(x_k - \mu_{k-1} \right)
\end{aligned}
$$

Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

Temporal Difference

- Just like in the 'incremental mean' example we can improve $q_\pi(s, a)$ iteratively:

$$q_*(s, a) = \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma \max_{a'} q_*(s', a')]$$

- We don't have $p(r, s' \mid s, a)$ to compute 'fair' expectation, so what should we do?

Temporal Difference

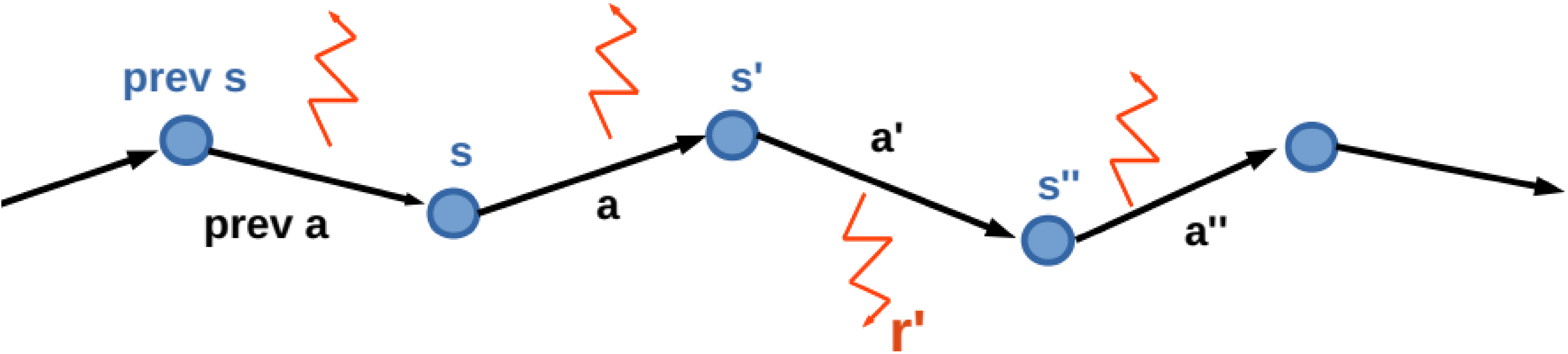
$$\sum_{r,s'} p(r, s' \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \approx$$

$$\approx \frac{1}{N}\sum_i r_i + \gamma \max_{a'} Q(s_i', a')$$

- One more trick: use incremental averaging with 1 sample.

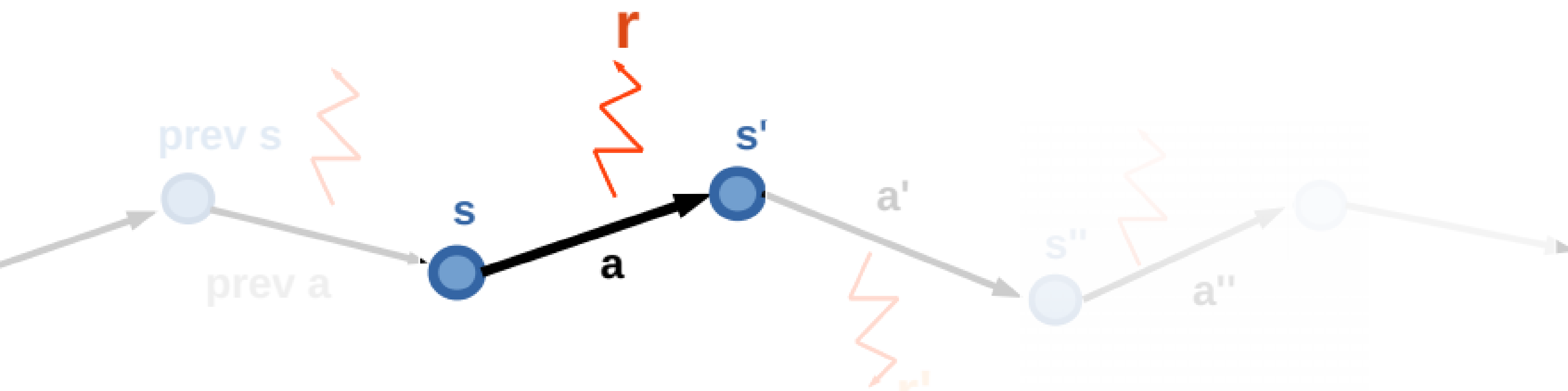
$$Q(s_t, a_t) = \alpha\, (r_t + \gamma\, max_{a'}\, Q(s_{t+1}, a')) + (1 - \alpha)\, Q(s_t, at)$$

Q-learning



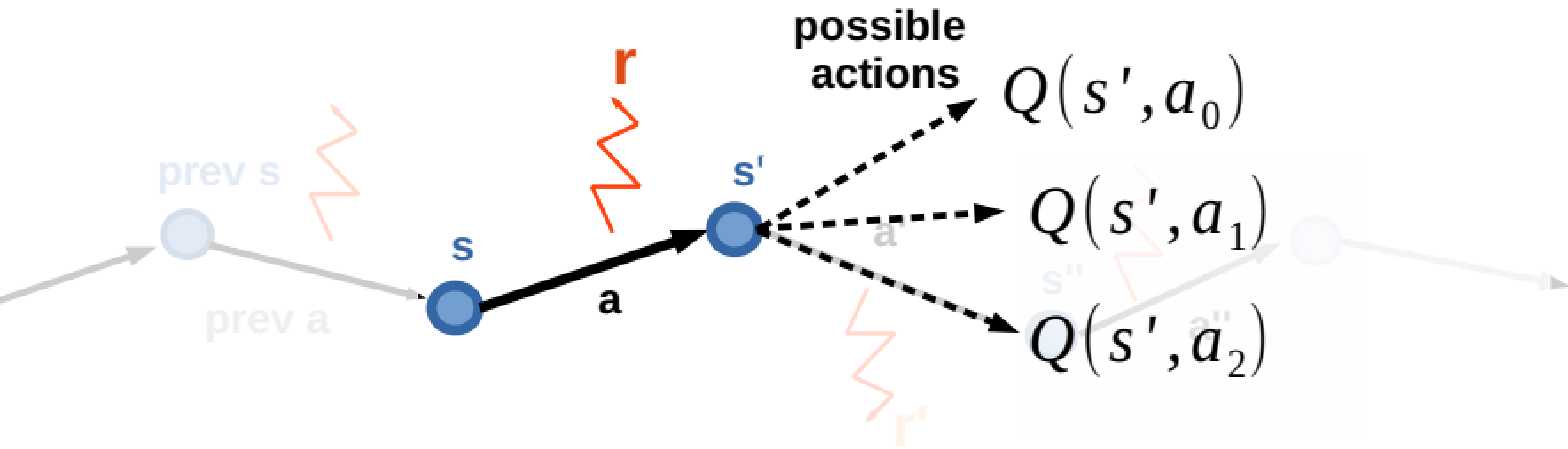
- Works on a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

Q-learning



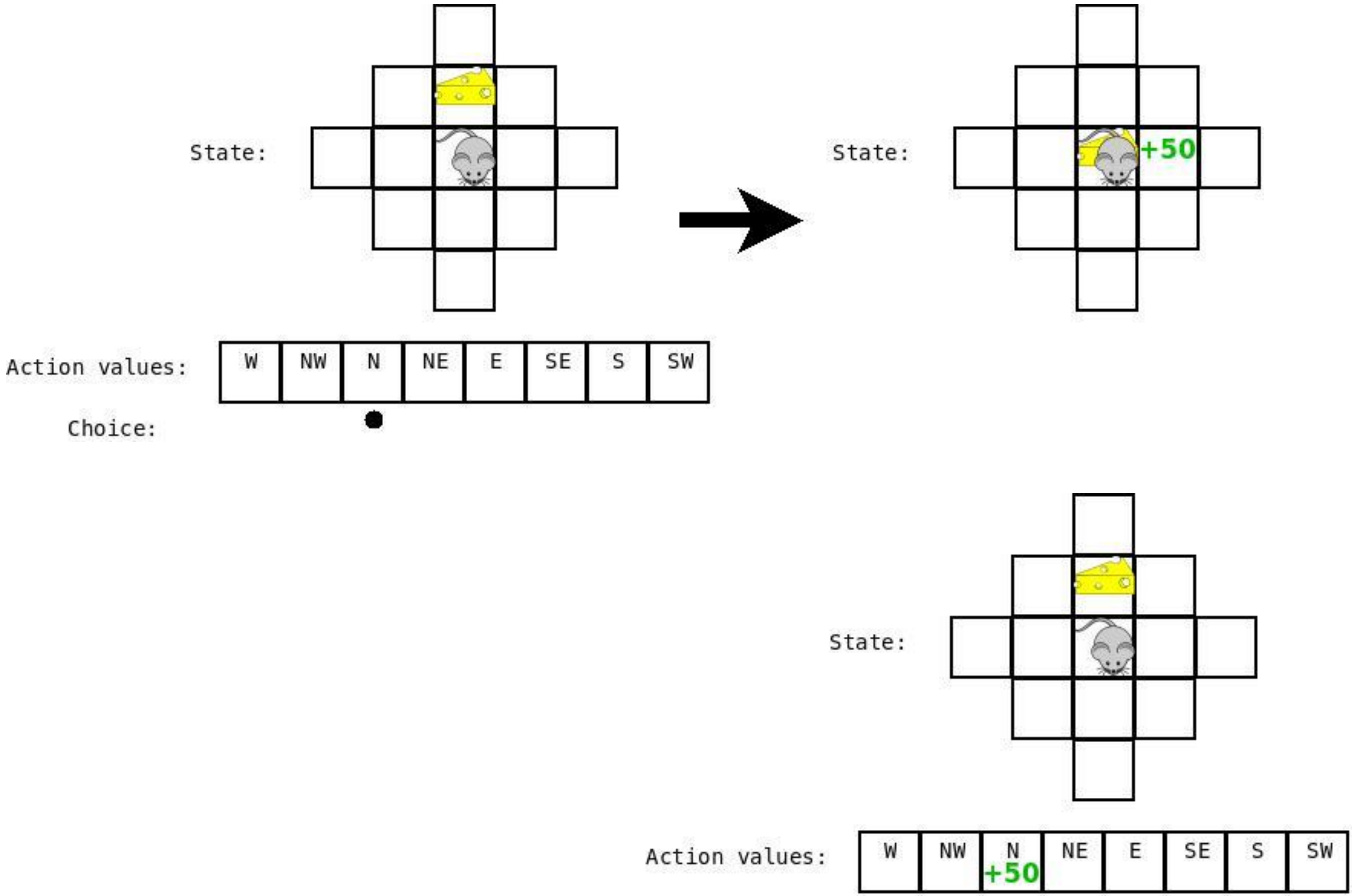
- Initialize $\boldsymbol{Q(s, a)}$ with zeros
- Cycle:
  - Sample **<s, a, r, s’>** from environment

Q-learning

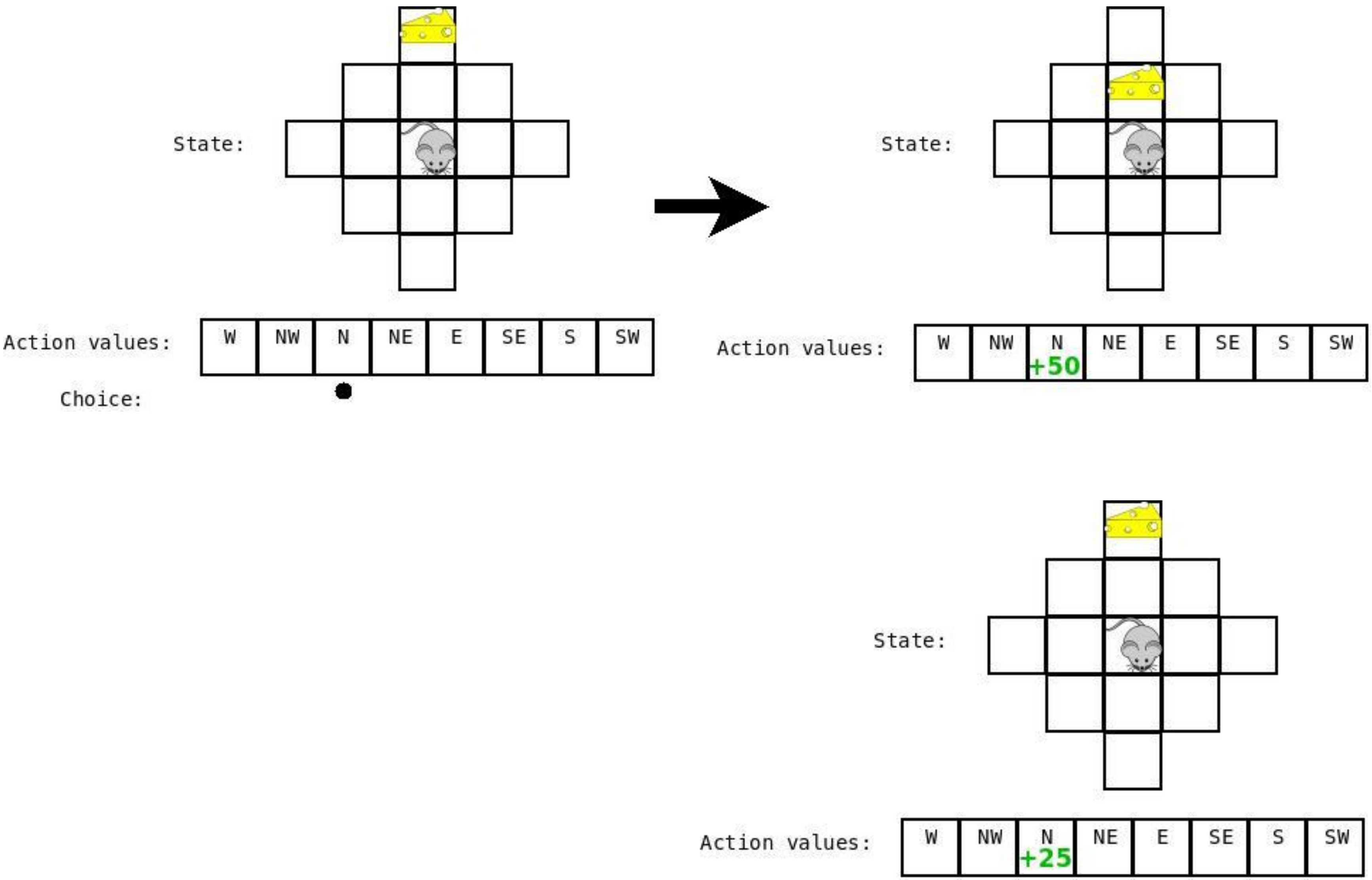


- Initialize $\boldsymbol{Q(s,a)}$ with zeros
- Cycle:
  - Sample **<s, a, r, s'>** from environment
  - Compute $\boldsymbol{\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',ai)}$
  - Update: $\boldsymbol{Q(s_t,at) = \ \alpha\, \hat{Q}(s,a)}$ **+ (1 -** $\boldsymbol{\alpha}$**)** $\boldsymbol{Q(s_t,at)}$

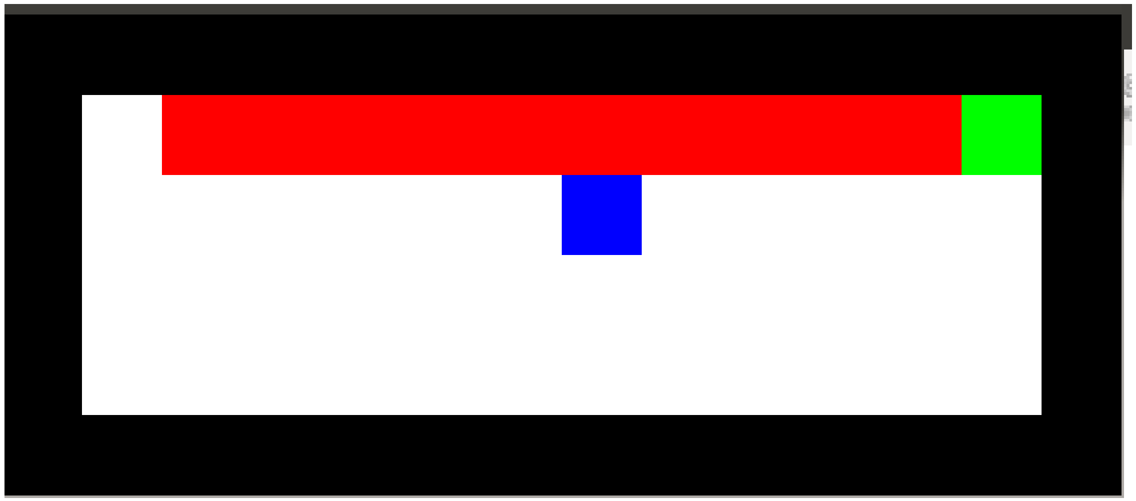
Q-learning mouse example



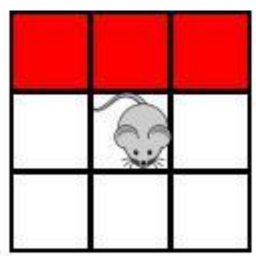
Q-learning mouse example



Mouse cliff example



State:

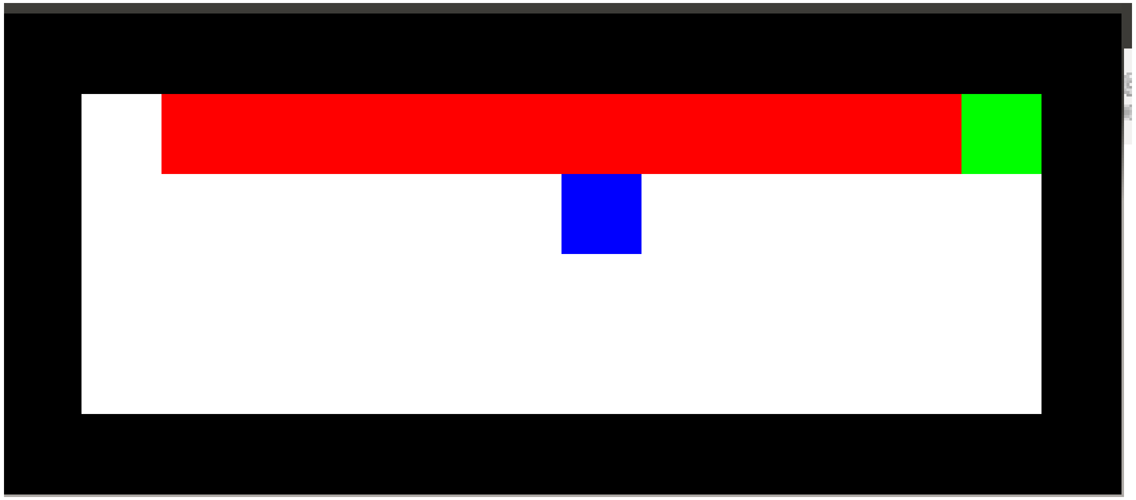


Action values:

W	NW	N	NE	E	SE	S	SW
	-50	-50	-50	+50			

Mouse cliff example

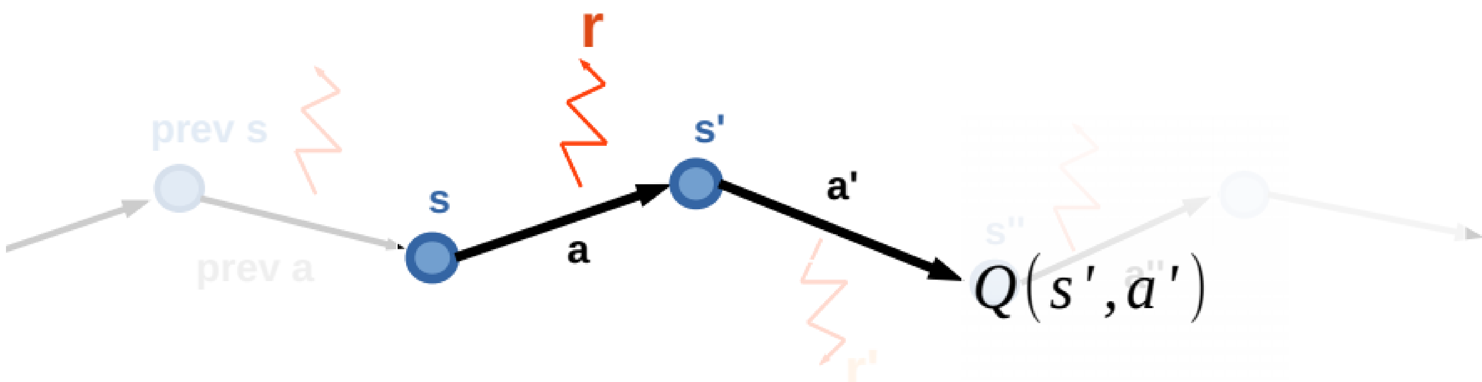
- Q - learning results:



Mouse cliff example

What's wrong with Q-learning?

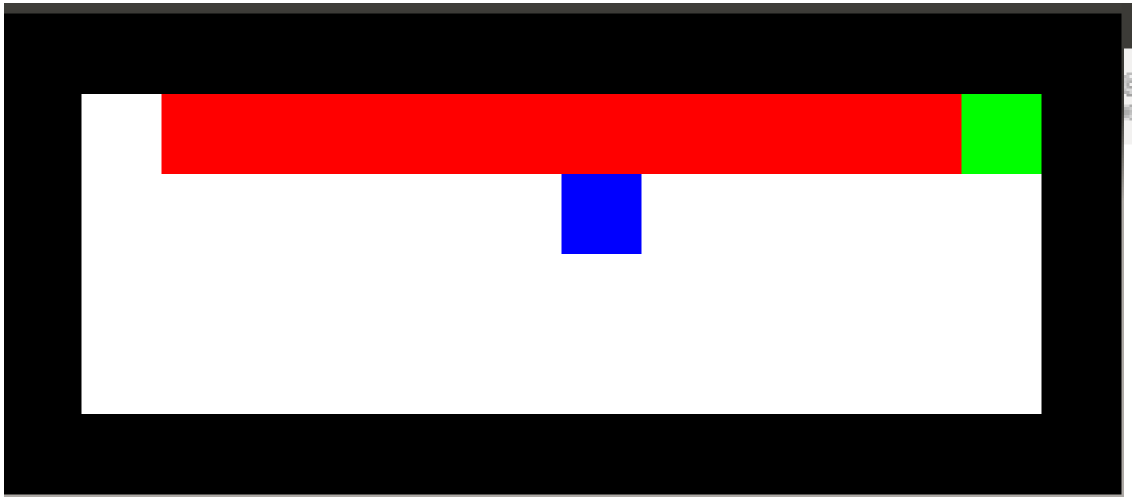
S-A-R-S-A

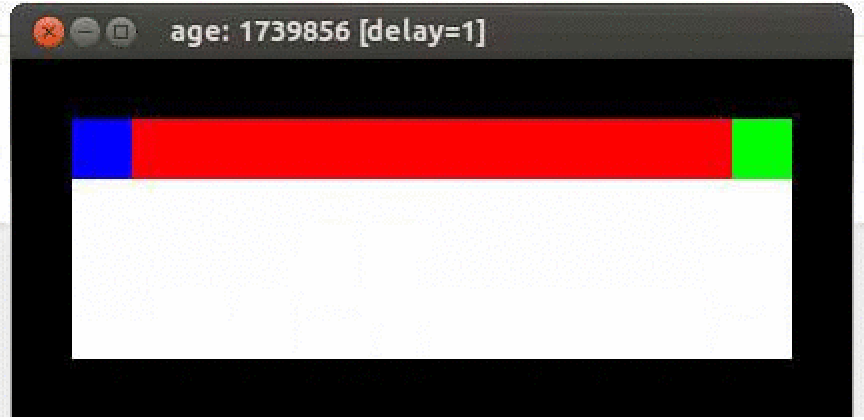


- Initialize $\boldsymbol{Q(s,a)}$ with zeros
- Cycle:
  - Sample **<s, a, r, s', a'>** from environment
  - Compute $\widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a}) = \boldsymbol{r}(\boldsymbol{s},\boldsymbol{a}) + \boldsymbol{\gamma Q}(\boldsymbol{s}',\boldsymbol{a}')$
  - Update: $\boldsymbol{Q(s_t, at)} = \ \boldsymbol{\alpha}\,\widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a})$ **+ (1 -** $\boldsymbol{\alpha}$**)** $\boldsymbol{Q(s_t, at)}$

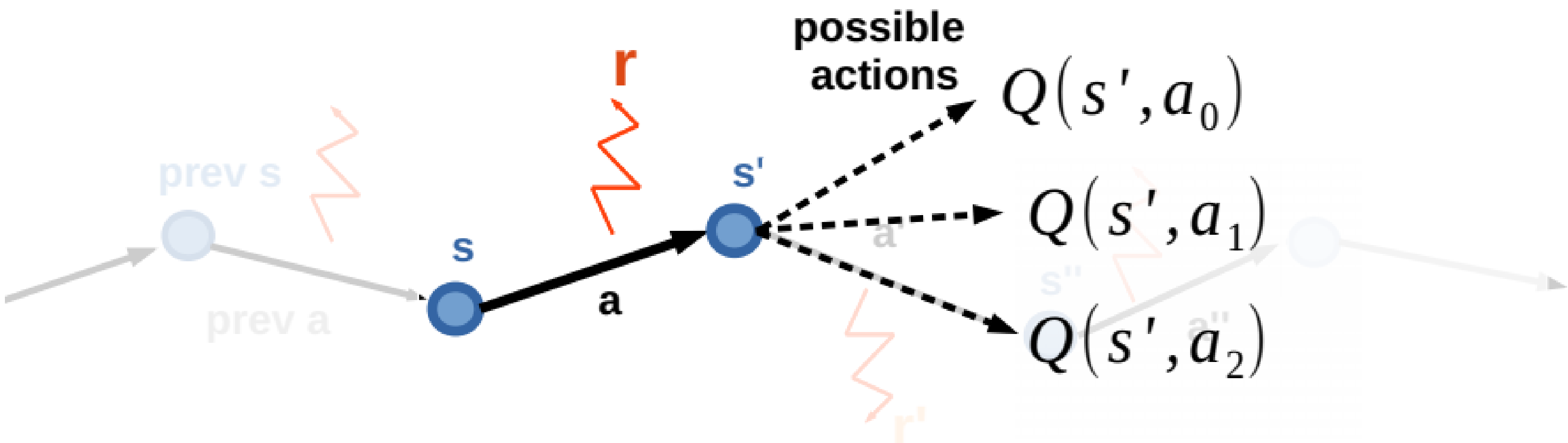
Mouse cliff example

- SARSA - learning results:



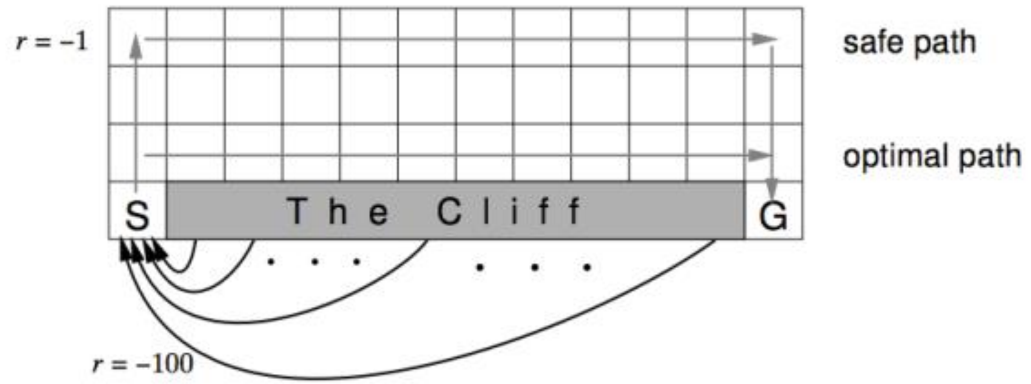


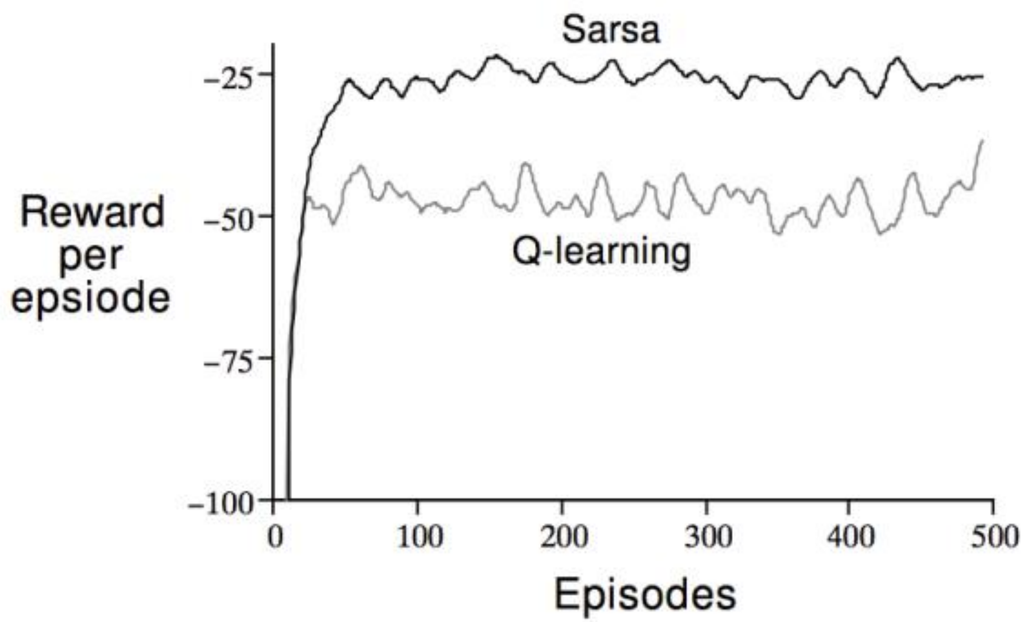
Expected value S-A-R-S-A



- Initialize $\boldsymbol{Q(s, a)}$ with zeros
- Cycle:
  - Sample **<s, a, r, s'>** from environment
  - Compute $\hat{\boldsymbol{Q}}(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{r}(\boldsymbol{s}, \boldsymbol{a}) + \boldsymbol{\gamma} \boldsymbol{E}_{\boldsymbol{a''} \sim \pi}[\boldsymbol{Q}(\boldsymbol{s'}, \boldsymbol{a''})]$
  - Update: $\boldsymbol{Q(s_t, at)} = \ \boldsymbol{\alpha}\, \hat{\boldsymbol{Q}}(\boldsymbol{s}, \boldsymbol{a})$ **+ (1 -** $\boldsymbol{\alpha}$**)** $\boldsymbol{Q(s_t, at)}$

Mouse cliff example





S-A-R-S-A vs Q-learning

- SARSA gets optimal rewards under current policy
- Q-learning policy **would be** optimal (converges to Q*)

S-A-R-S-A vs Q-learning

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

MC vs TD

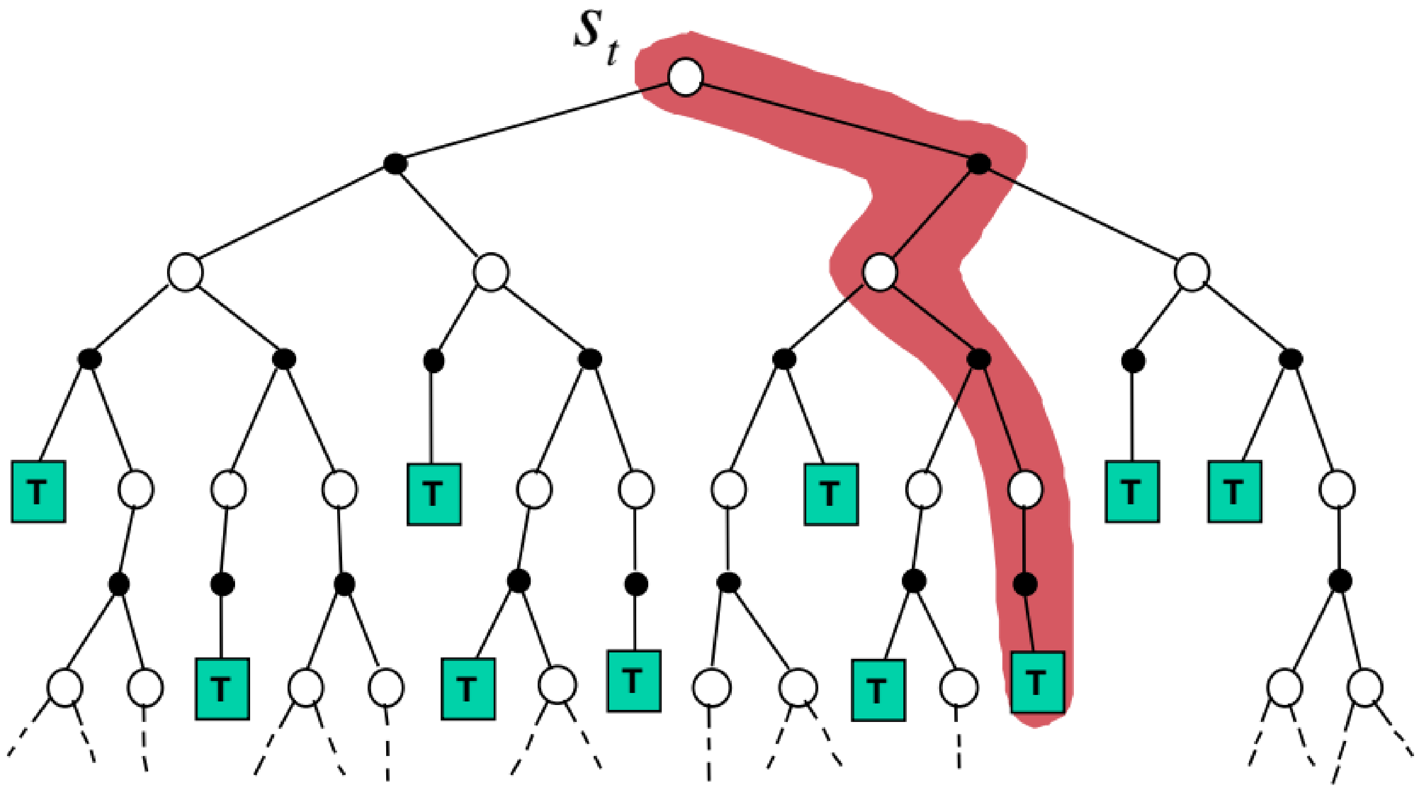
- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

MC vs TD: Bias/Variance Trade-off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is *unbiased* estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is *unbiased* estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is *biased* estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return:
  - Return depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random action, transition, reward

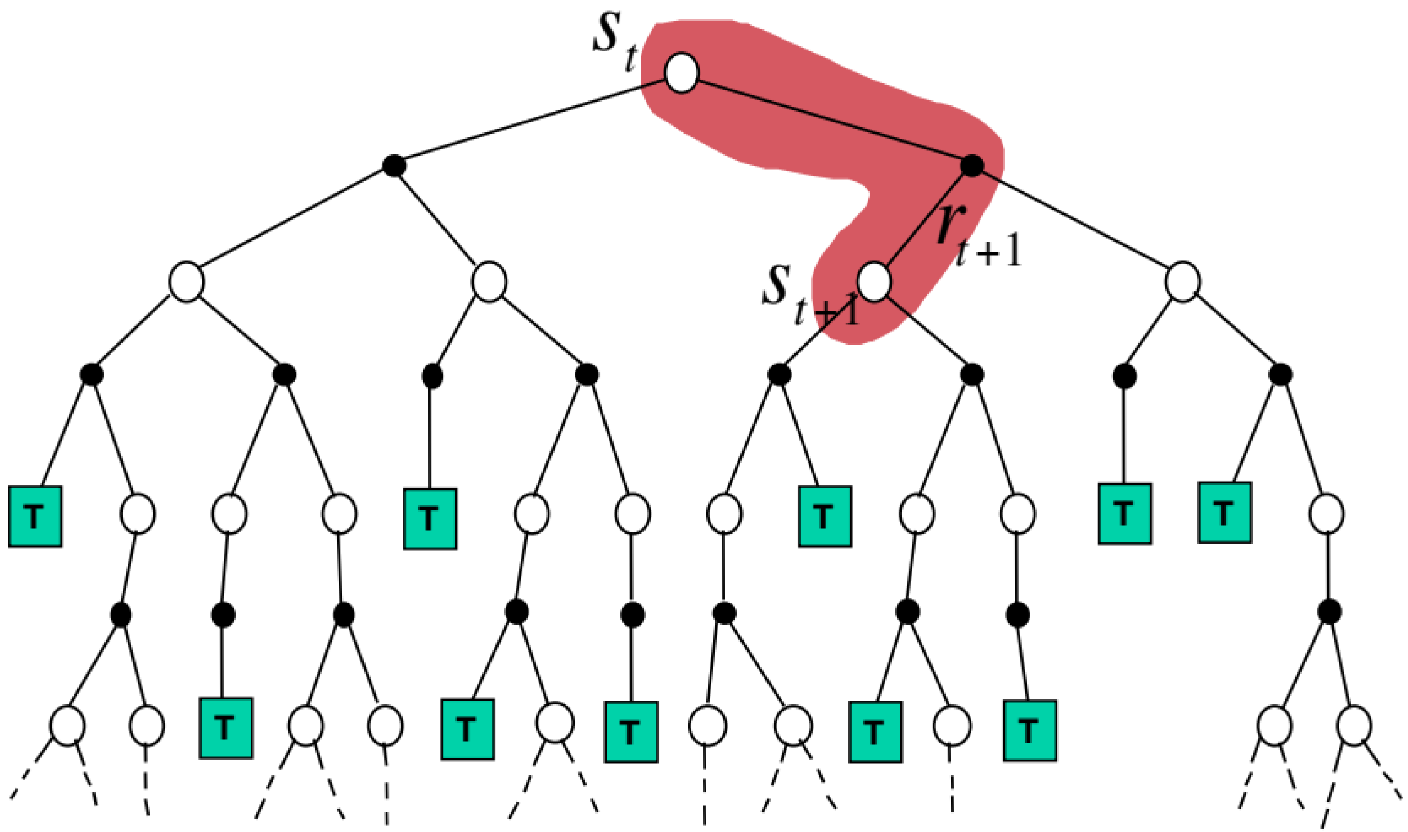
Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$



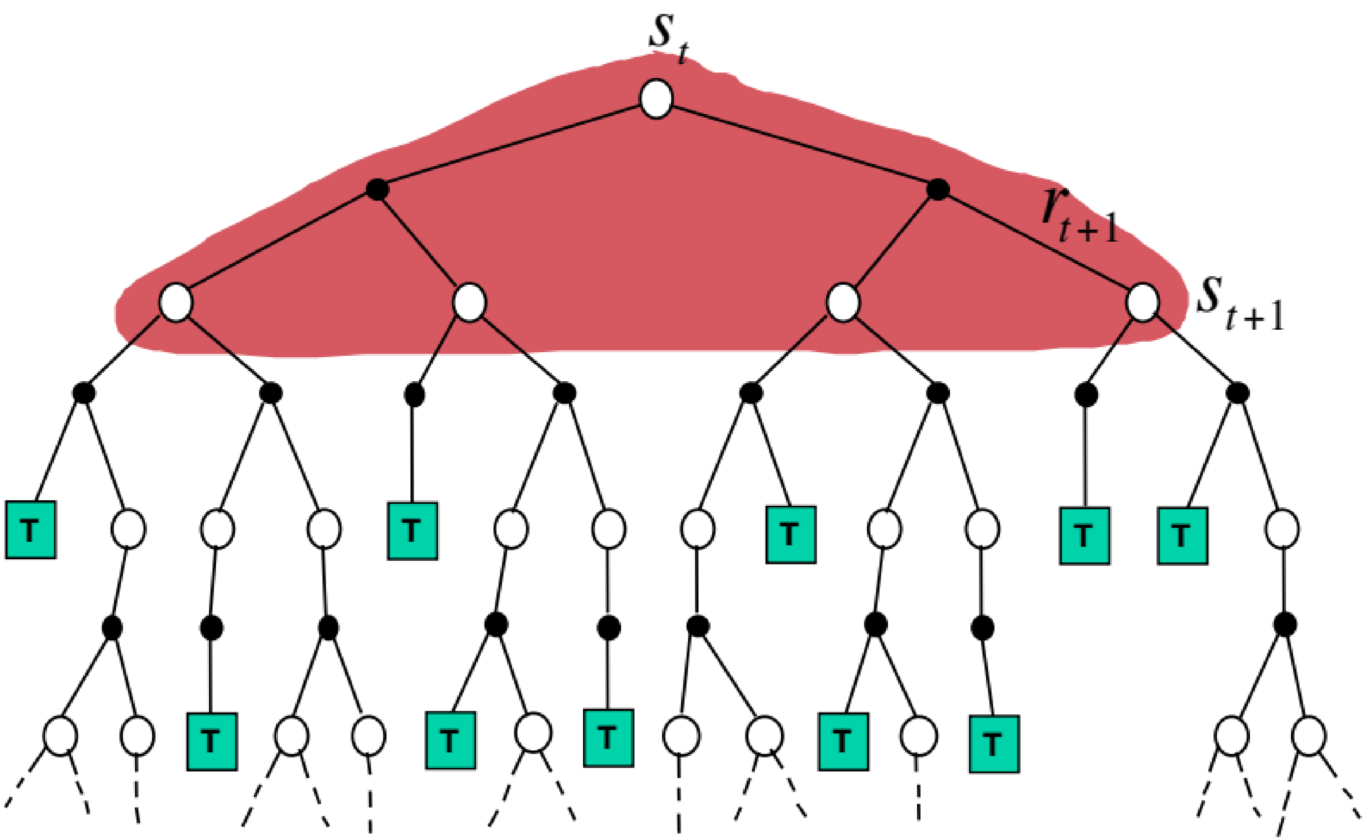
Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



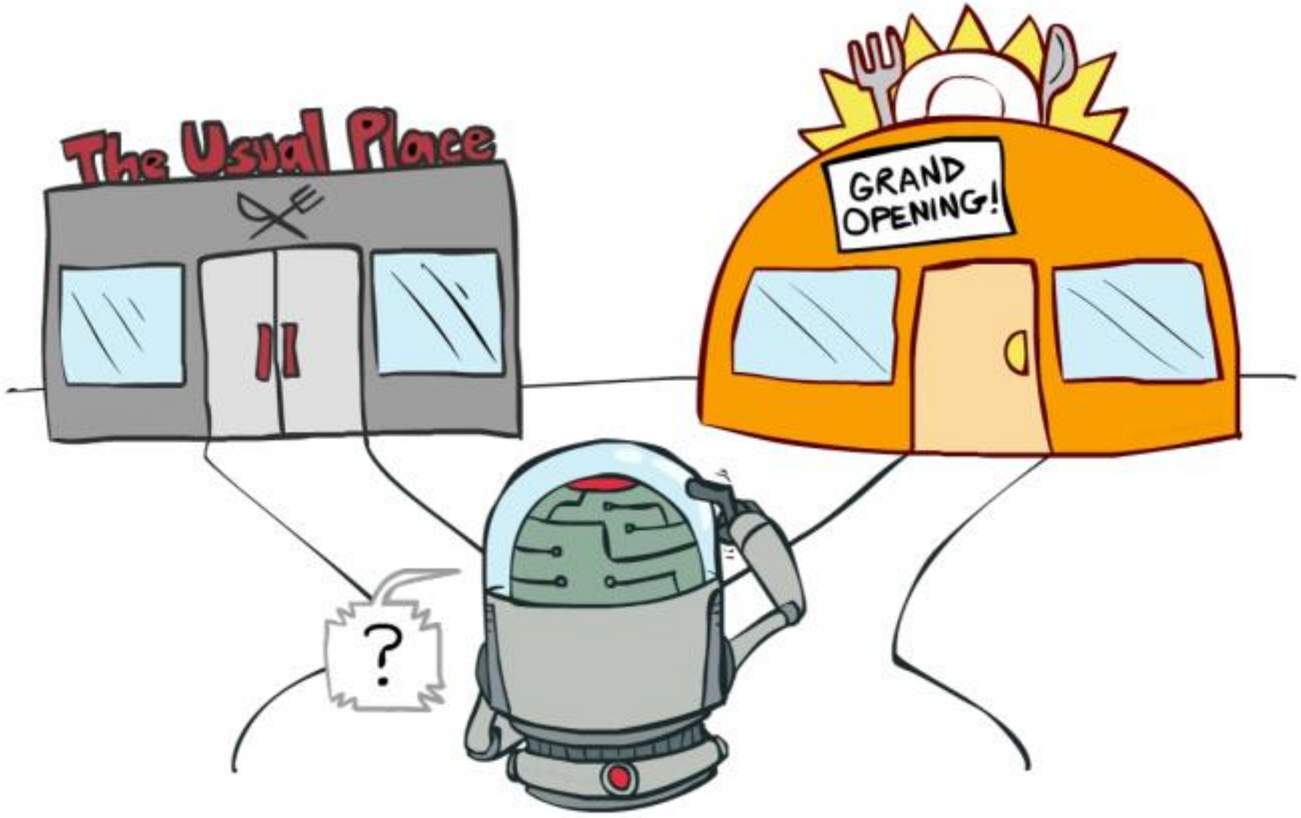
Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



Exploration/Exploitation Revisited

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



Exploration/Exploitation Revisited

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

$$\pi(a \mid s) = softmax(\,Q(s,a) \,/\, \tau)$$

  - ε- dithering
    Adding random noise to Q-values with ε probability

Exploration/Exploitation over time

- 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

Resume

What have we learned today?

Questions?