Reinforcement Learning

COMP9414: Artificial Intelligence

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

Initial ideas

Instrumental or operational conditioning.
Stimulus-behavior learning.

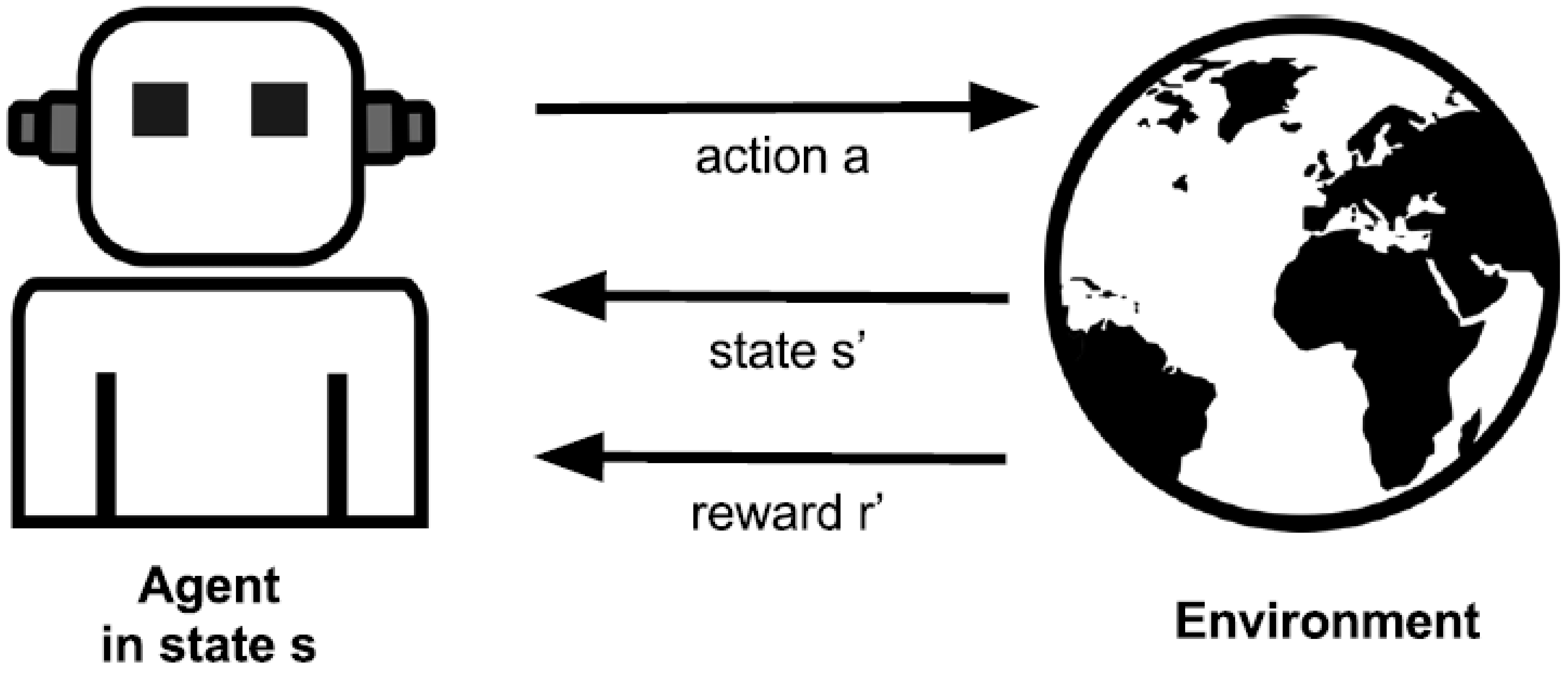
Thorndike, 1911

Deakin University CRICOS Provider Code: 00113B

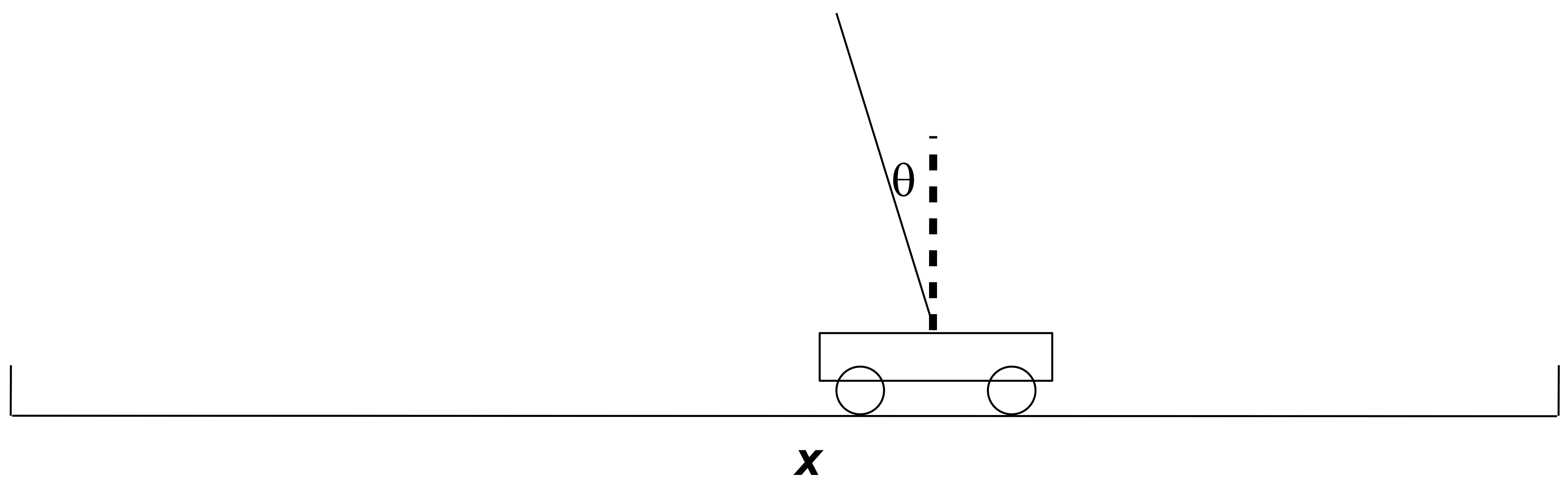
Reinforcement learning

- Idea of learning by interaction with the environment.
- With no explicit instructor but with a direct sensorimotor connection.
- Awareness of how our environment answers to what we do.

Reinforcement Learning

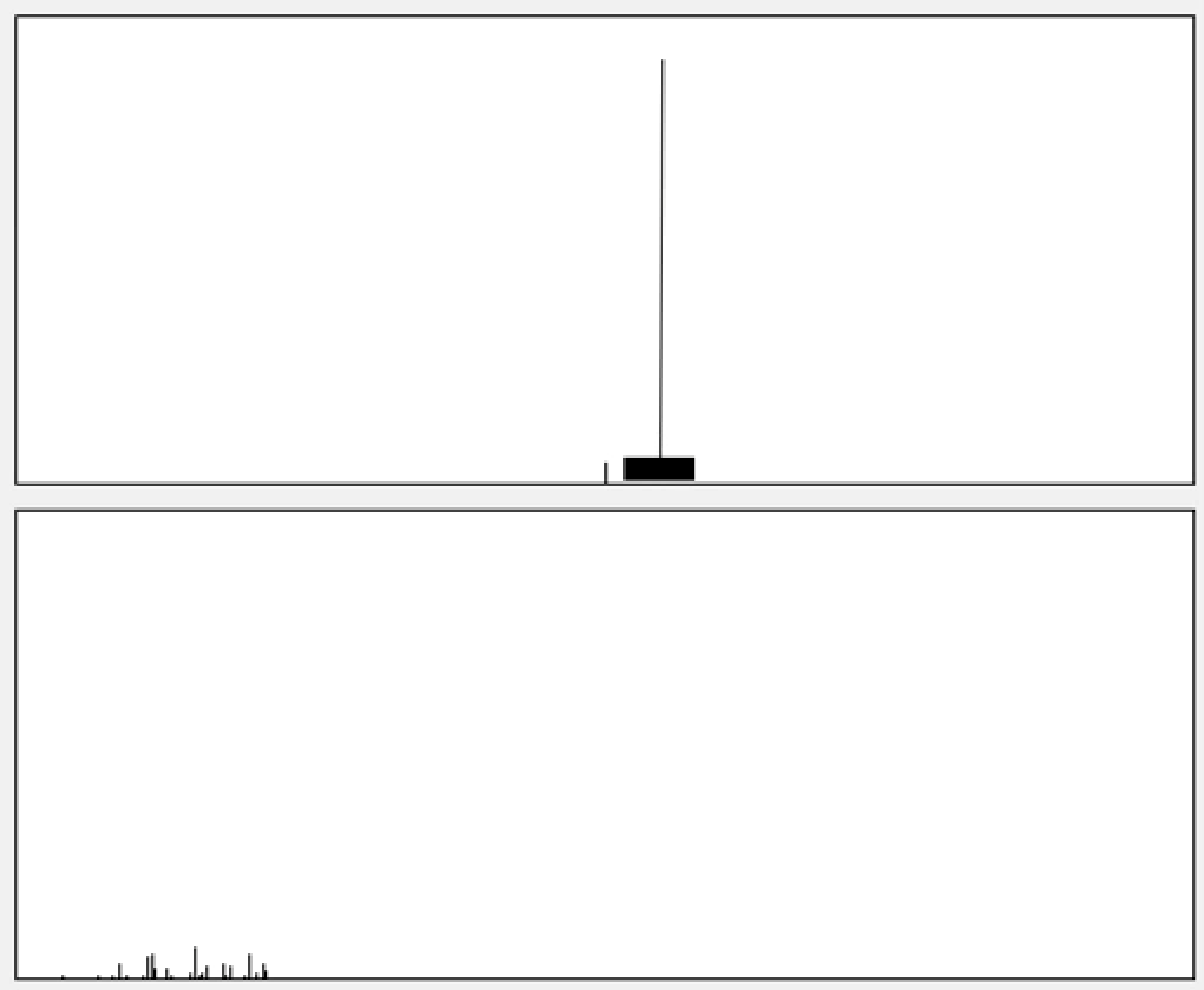


Pole balancing



- Pole balancing can be learned the same way
- Reward might be only received at the end
  - after falling or hitting the end of the track

Pole balancing



And you think pole balancing is trivial?

Types of learning

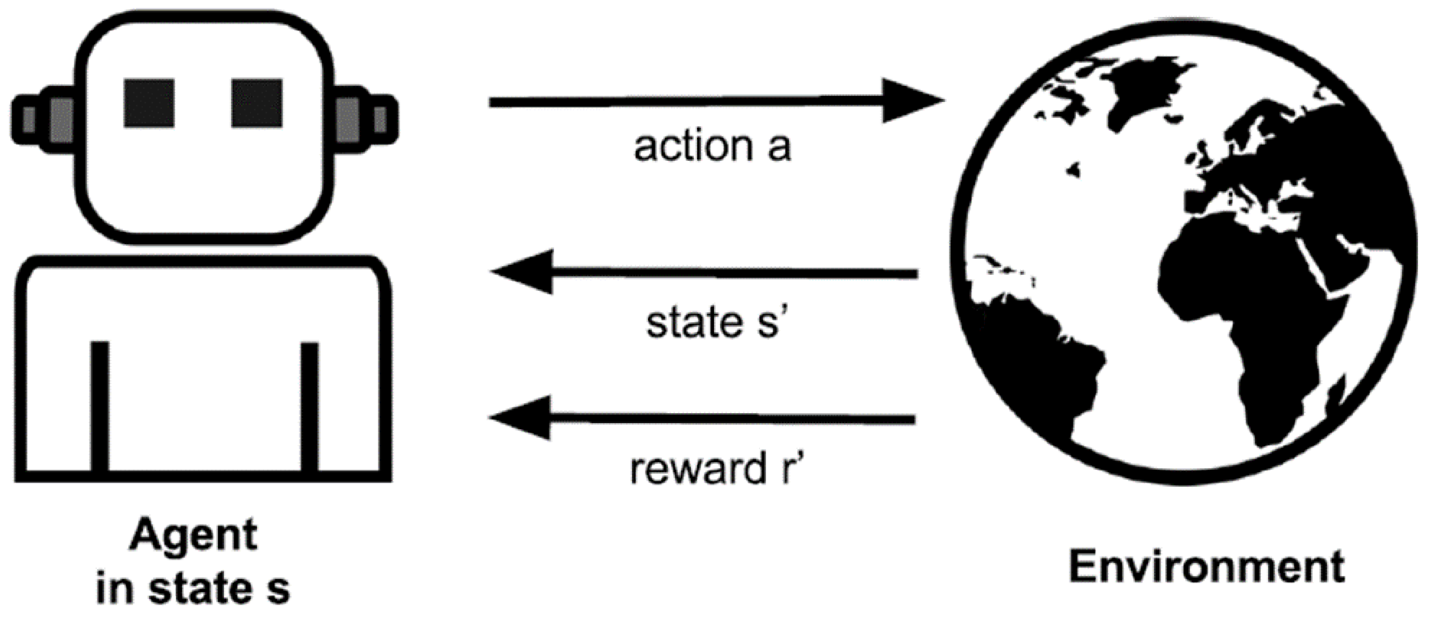
- Supervised Learning
  - Agent is given examples of input/output pairs
  - Learns a function from inputs to outputs that agrees with the training examples and generalises to new examples
- Unsupervised Learning
  - Agent is only given inputs
  - Tries to find structure in these inputs
- Reinforcement Learning
  - Training examples presented one at a time
  - Must guess best output based on a reward, tries to maximise (expected) rewards over time

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

Reinforcement Learning

- RL is to learn what to do, mapping from situations to actions.
- An agent should be able to sense the environment states and perform actions to affect such states.
- Actions might affect not only immediate reward.
- An important challenge is the exploration/ exploitation trade-off problem.



Elements of reinforcement learning

There are four essential elements:

- Policy
    - Informs how to act in a particular situation.
    - Set of stimulus-response rules or associations.
    - Can be stochastic.

Elements of reinforcement learning

There are four essential elements:

- **Reward function**

    - Defines the aim of an RL problem.
    - Maps each perceived state (or state-action pair) into a number, the reward.
    - The goal is to maximize the long-term reward.
    - In biological systems may correspond to pain and pleasure feelings.
    - Can be stochastic.

Elements of reinforcement learning

There are four essential elements:

- Value function

    - Shows what it’s good in the long run (the reward in an immediate sense).
    - In biological systems corresponds to more refined judgments o foresight about the future from one state.
    - Actions are decided based on the value.
    - It’s much harder to determine values than rewards.

Elements of reinforcement learning

There are four essential elements:

- **Optionally, a model of the environment**

    - Imitates the environment behaviour.
    - Can predict states and reward obtained.
    - The use of models of the environment is still relatively new.

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

Exploration / Exploitation Trade-off

- Most of the time, the agent chooses what it thinks the “best” action is.
- But to learn, it must occasionally choose something different from the preferred action.

Exploration / Exploitation Trade-off

Should I stay or should I go now?
Should I stay or should I go now?
If I go, there will be trouble
And if I stay it will be double

-- The Clash

Exploration / Exploitation Trade-off

- The greedy action exploits the current knowledge.
- The non-greedy action explores.
- Exploitation maximises the immediate reward and exploration in the long run.
- There is a conflict between exploration and exploitation.

Action-value estimation methods

- We denote the real action value as $q_*(a)$.
- We denote the estimated value at time-step t as $Q_t(a)$.
- **Simple Estimation:** to average received rewards when action **a** has been selected K_a times.

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{K_a}}{K_a}.$$

Action-value estimation methods

- If $K_a = 0$, $Q_t(a)$ is defined with an arbitrary value, e.g., $Q_t(a) = 0$ (not necessarily the best).
- As $K_a \rightarrow \infty$, $Q_t(a)$ converges to $q_*(a)$.

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{K_a}}{K_a}.$$

Action-value estimation methods

Greedy method

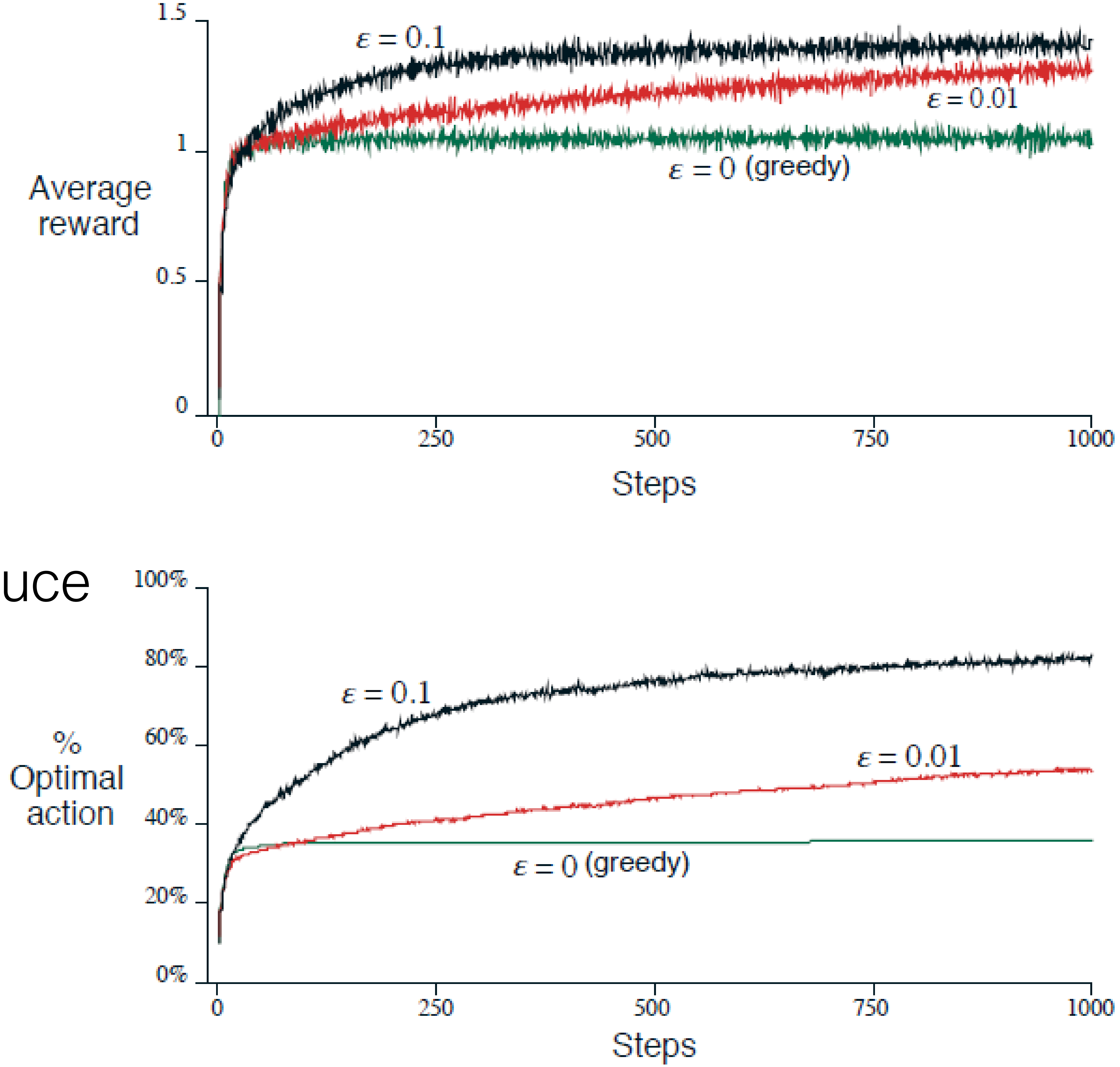
- The simplest way to choose an action: the action with the highest estimated value.
- A_t^* where $Q_t(A_t^*) = \max_a Q_t(a)$.

ε-greedy method

- A simple alternative: to choose the best action most of the time, and sometimes (with a small probability ε) a random one.
- $Q_t(a)$ converges to $q_*(a)$ with probability 1- ε.

Action-value estimation methods

- 2000 agents averaged.
- It's possible to reduce ε over time



Action-value estimation methods

Softmax method

- ε-greedy effectively trades off exploration and exploitation, but the selection is equitable (or fair) among actions.
- Sometimes, the worst action is very bad.
- High temperatures give almost equal probability for all actions.
- Low temperatures make a bigger difference in the probability.

$$\frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}}$$

Incremental implementation

- A simple implementation: record all the rewards.

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{K_a}}{K_a}$$

- Problem: growing use of memory and computational cost over time.

Incremental implementation

- Denote Q_k the estimated reward at time-step k, i.e., the average of the k-1 first rewards, then:

$$
\begin{aligned}
Q_{k+1} &= \frac{1}{k}\sum_{i=1}^{k} R_i \\
&= \frac{1}{k}\left(R_k + \sum_{i=1}^{k-1} R_i\right) \\
&= \frac{1}{k}\Big(R_k + (k-1)Q_k\Big) \\
&= \frac{1}{k}\Big(R_k + kQ_k - Q_k\Big) \\
&= Q_k + \frac{1}{k}\Big[R_k - Q_k\Big],
\end{aligned}
$$

Non-stationary problems

- Methods of average are appropriate for stationary problems.
- In non-stationary problems, make sense to give more weight to more recent rewards than the past ones.
- Using a constant parameter *step-size*, $0 < \alpha \leq 1$:

$$Q_{k+1} = Q_k + \alpha\Big[R_k - Q_k\Big]$$

Contextual or associative search

- So far, only non-associative tasks, i.e., no association between actions and situations.
- In an RL problem, there is more than one situation.
- The aim is to learn a policy: to map situations into actions.
- For instance, a set of n-armed bandits with changing colours.

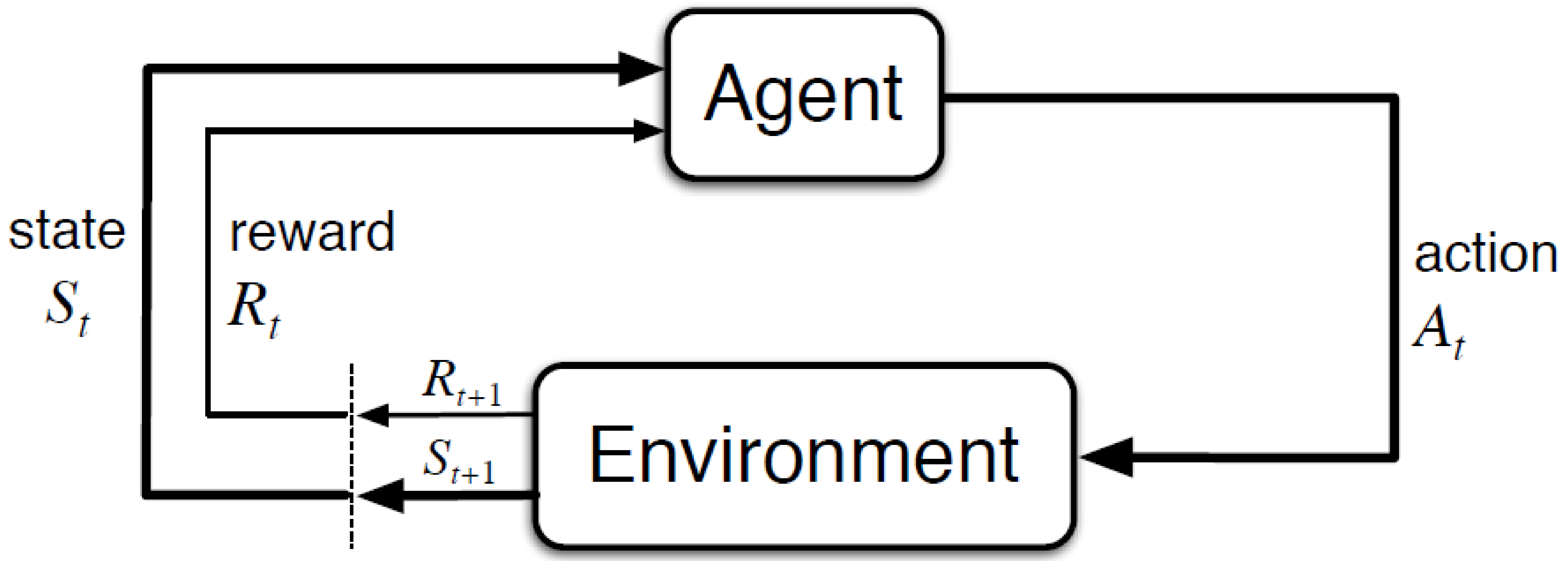
- **If actions also affect the following situation as well as the reward, then it corresponds to a full RL problem.**

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

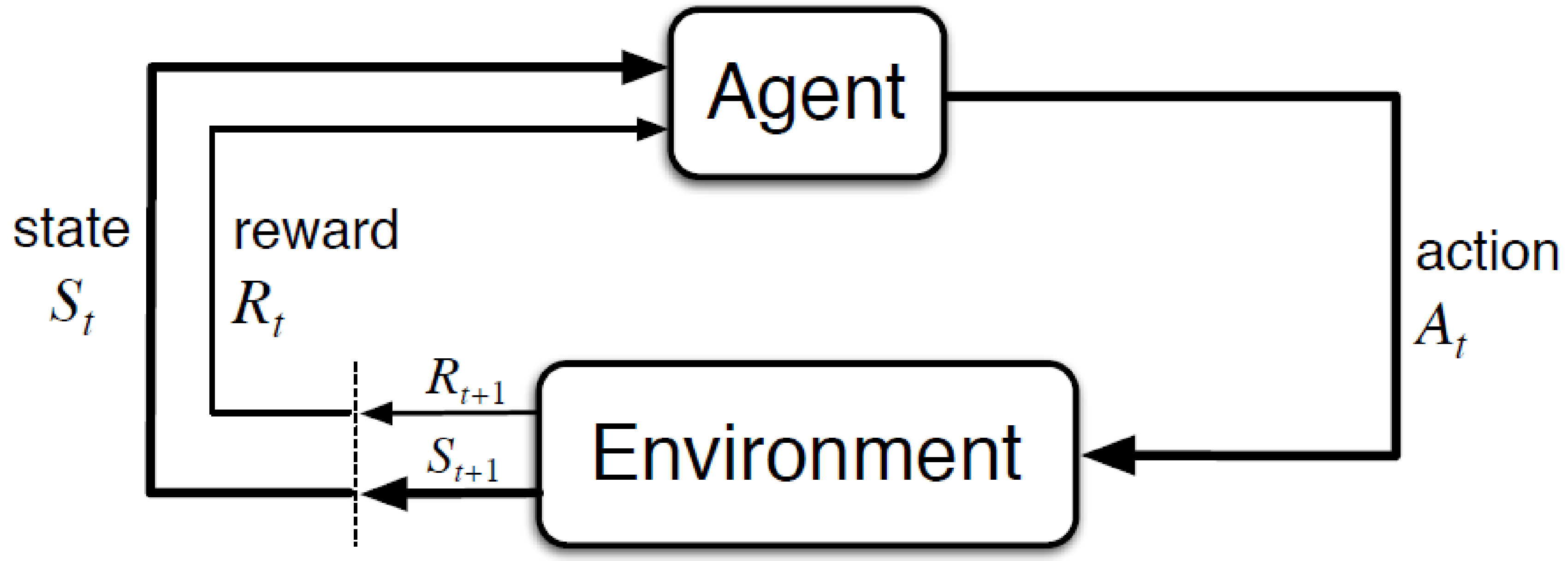
The agent-environment interface

- Any method able to solve the problem is considered an RL method.
- **Agent:** comprises the learner and the one making the decisions. (although they can be separated!).
- **Environment:** everything external to the agent that it interacts with.



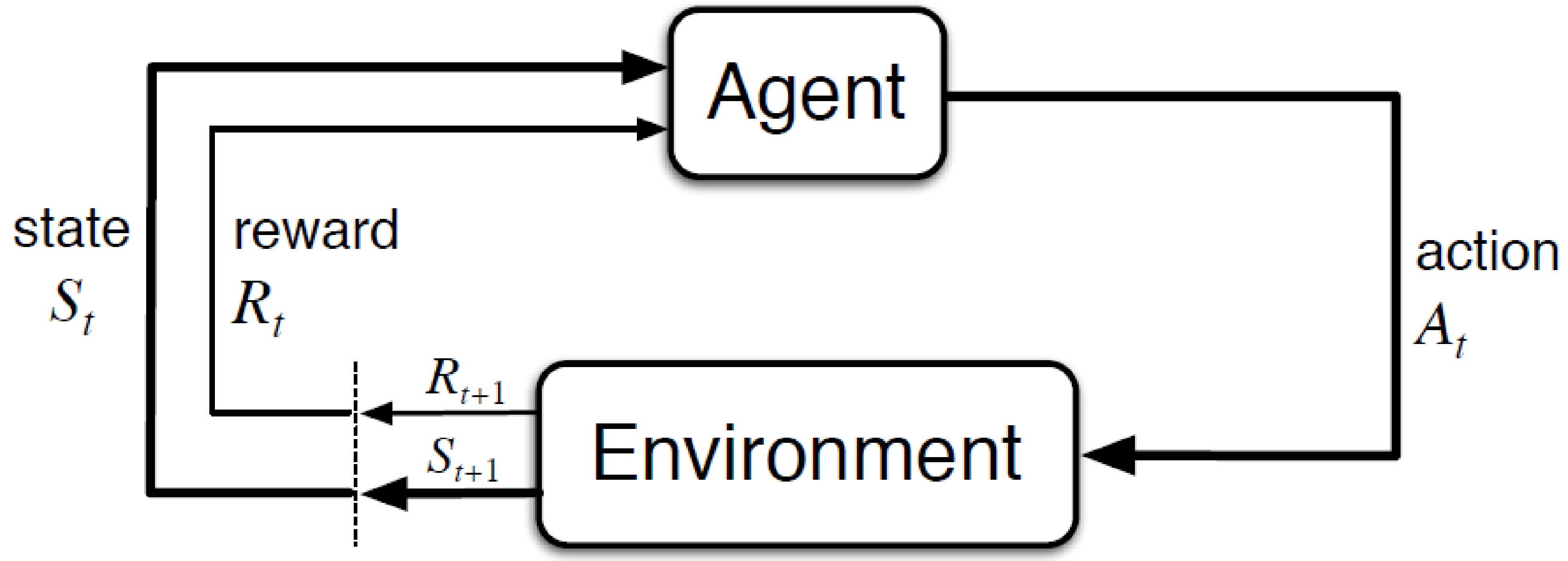
The agent-environment interface

- **Reward**: numeric value the agent tries to maximise. $R_{t+1} \in \mathbb{R}$.
- $S_t \in S$. S set of possible states.
- $A_t \in A(S_t)$. $A(S_t)$ actions available at S_t.
- The agent implements a map from the states toward the action selection probability.
- This is called agent policy π_t where $\pi_t(a|s)$ is the probability of $A_t = a$ and $S_t = s$.
- RL methods detail how an agent updates its policy as a result of its experience.



The agent-environment interface

- **Example:** recycling robot.
- The agent decides if (i) actively search for a can, (ii) remains stationary and waits for a can, or (iii) gets back to the home base to recharge (three possible actions).
- The state is determined by the battery state.
- Reward: Mostly zero, positive when collects a can and negative (much higher) when the battery runs empty.



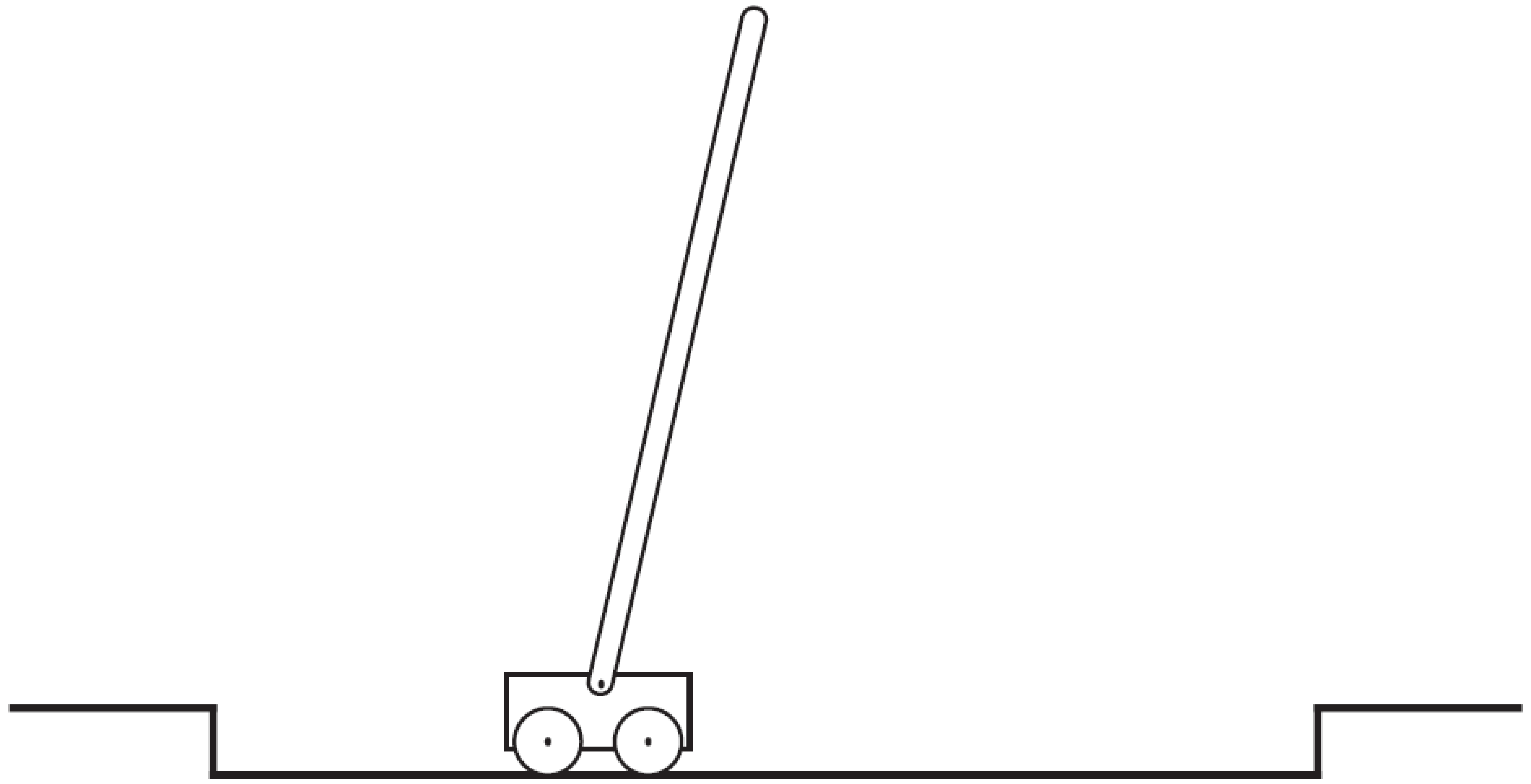
Goals and rewards

- The agent's goal is to maximise the amount of total reward, not the immediate reward.
- A robot learning to walk receives a reward proportional to the forward movement.
- A robot learning to escape from a maze receives a reward equal to zero until escapes and then receives +1.
- Another strategy is giving a reward of -1 after each movement till escaping.
- An agent learning to play chess receives +1 for winning and -1 for losing.

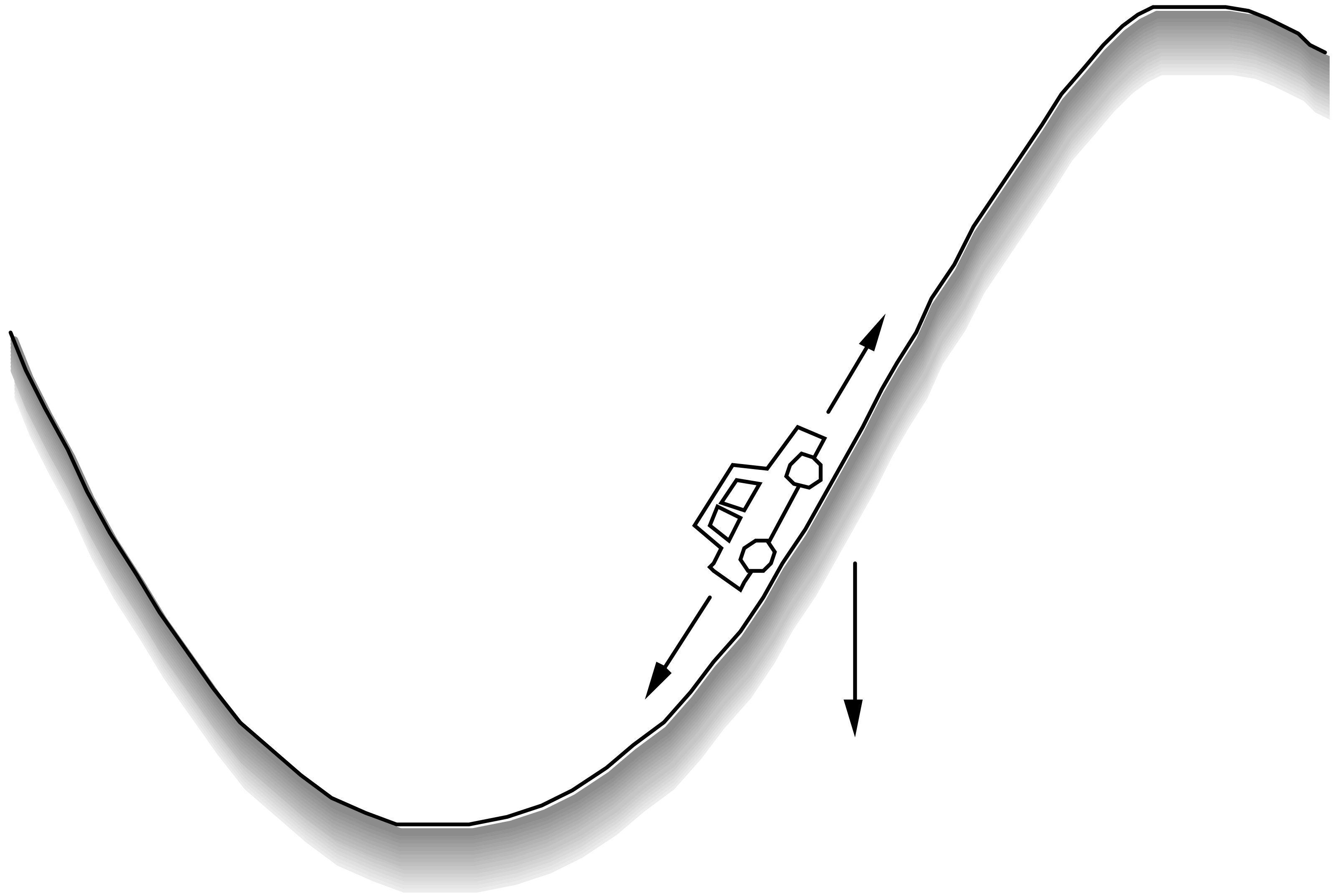
Goals and rewards

- The chess player should be rewarded only for winning and not for taking opponent's pieces.
- Otherwise, the agent will learn to maximise subgoals.
- In summary, the reward signal is the way to communicate to the agent **what** you want it to achieve, **not how** you want it achieved.

Episodic and non-episodic tasks



The pole-balancing task.



The mountain car task.

Returns

- If the reward sequence received is R_{t+1}, R_{t+2}, R_{t+3}, …. We want to maximise the expected return G_t.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- In tasks with final state and that can be divided into subsequences (episodes)
- Each episode finishes in the final state and the task starts over from an initial state.
- These tasks are known as episodic tasks.

Returns

- Tasks intended to be performed continuously without limit are referred to as continuous tasks (or non-episodic).
- The return could be infinite, given that T = ∞.
- In this case, the agent maximises the discounted rewards, choosing actions to maximise the discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad = \quad \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Discount rate $0 \leq \gamma < 1$. Determine the present value of future rewards. If $\gamma = 0$, the agent is myopic. If $\gamma \rightarrow 1$ the agent is foresighted.

Returns

3 minutes

γ	Reward sequence	Return
0.5	1 0 0 0	
0.5	0 0 2 0 0 0	
0.9	0 0 2 0 0 0	
0.5	-1 2 6 3 2 0 0 0	

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Returns

3 minutes

γ	Reward sequence	Return
0.5	1 0 0 0	1
0.5	0 0 2 0 0 0	
0.9	0 0 2 0 0 0	
0.5	-1 2 6 3 2 0 0 0	

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Returns

3 minutes

γ	Reward sequence	Return
0.5	1 0 0 0	1
0.5	0 0 2 0 0 0	0.5
0.9	0 0 2 0 0 0	
0.5	-1 2 6 3 2 0 0 0	

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Returns

4 minutes

γ	Reward sequence	Return
0.5	1 0 0 0 …..	1
0.5	0 0 2 0 0 0 …..	0.5
0.9	0 0 2 0 0 0 …..	1.62
0.5	-1 2 6 3 2 0 0 0 …..	

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Returns

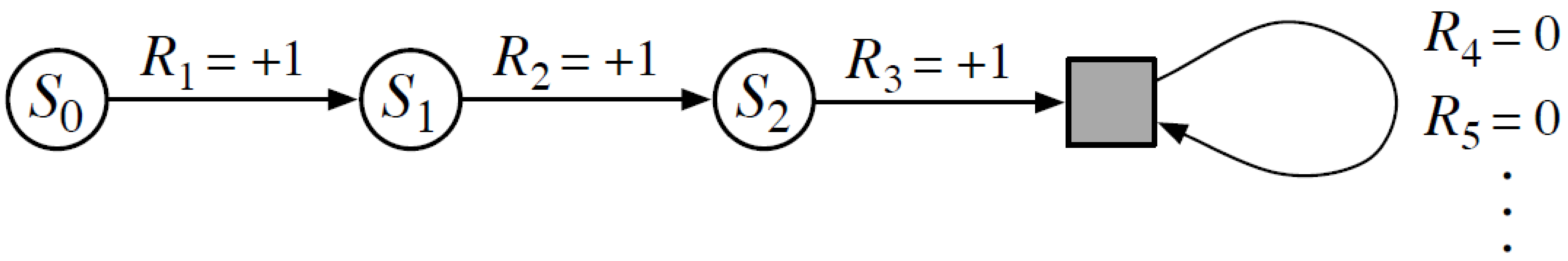
3 minutes

γ	Reward sequence	Return
0.5	1 0 0 0 …..	1
0.5	0 0 2 0 0 0 …..	0.5
0.9	0 0 2 0 0 0 …..	1.62
0.5	-1 2 6 3 2 0 0 0 …..	2

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Unified Notation

- A final absorbing state with reward equal to zero.



- It is possible that T = ∞ or γ = 1, but not both.

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

The Markov property

- In RL, state means any information available for the agent (either processed or not).
- The state should not inform everything about the environment to the agent. For instance, an agent playing blackjack should not know the next card in the deck.
- We do not blame the agent for not knowing something important, but we do for knowing something and then forgetting it.
- Ideally, a state should contain compact information about the past, retaining relevant information. This is called the Markov property. For instance, the chess board.

The Markov property

- Sometimes, the property cannot be fully satisfied.
- In pole-balancing, the state satisfies the property if the exact position and velocity of the cart are specified, and the pole angle and its change rate.
- However, there may exist distortions, such as delays and other effects as the temperature of the wheels.
- Some studies have even used a simple region division: left, right, and centre.

Markov Decision Processes

- An RL task with the Markov property is called Markov decision process (MDP).
- Markov decision process (MDP): $< S, A, \delta, r >$.
  - S is a finite set of states,
  - A is a set of actions,
  - δ is the transition function $\delta: S \times A \rightarrow S$, and,
  - r is the reward function $r: S \times A \rightarrow \mathbb{R}$.

Recycling robot MDP

- At each moment, the robot decides if (i) actively search for a can, (ii) waits for someone to bring a can, or (iii) gets back to the home base to recharge.
- The best strategy is to actively search for cans.
- In case the battery runs out, the robot needs to be rescued leading to a negative reward.
- The agent solely decides as a function of the energy level of the battery. Two levels: high, low.
- S = {high, low}.
- Possible decisions (agent's actions): wait, search, recharge.
- A(high) = {search, wait}.
- A(low) = {search, wait, recharge}.

Recycling robot MDP

- Transition probabilities and expected reward:

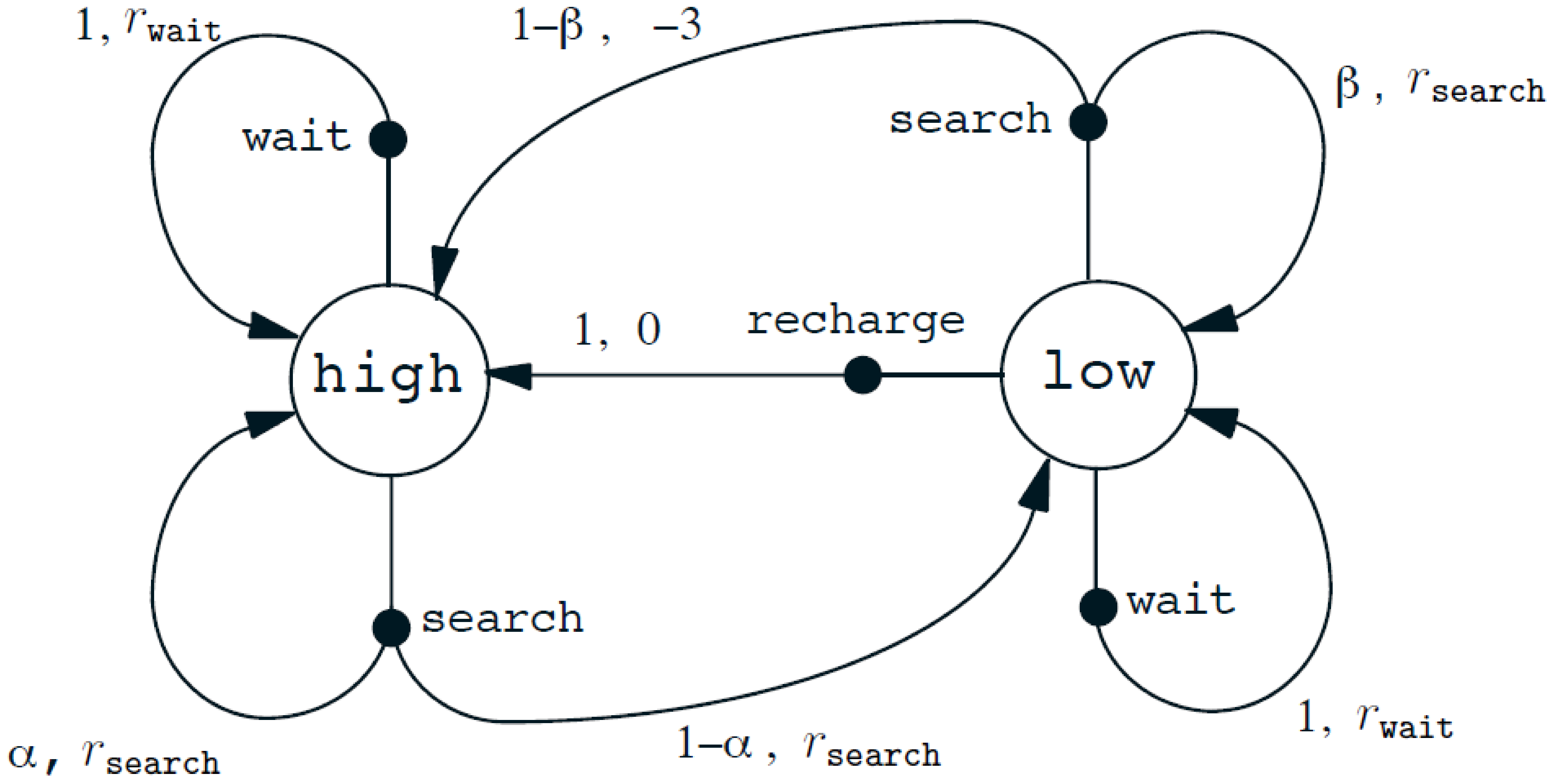
s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

$r_{search} > r_{wait}$

- Assumption: cans cannot be collected when going back to the home base or when the battery is depleted.

Recycling robot MDP

- Transition graph:



- Transition probabilities from one action always sum to 1.

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

Value function

- Value function estimations.
  - State-value function, or
  - Action-value function (for state-action pairs)

- The function estimates how good it is for the agent to be in a given state, in terms of future reward (or expected return).
- The value of a state s under a policy π, denoted $v_\pi(s)$ or $V^\pi(S)$, is the expected return when starting in s and following π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right]$$

Value function

- The value of a terminal state, if any, is zero.
- The value of taking action a in state s under policy π, is denoted $q_\pi(s,a)$ or $Q^\pi(S,A)$:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

- Value functions $v_\pi(s)$ and $q_\pi(s,a)$ can be estimated from experience.
- If there are many states, it's impractical to keep values for each state.
- In this case, parameterized function approximators are used to keep $v_\pi(s)$ and $q_\pi(s,a)$.

Value function

- Try to maximise expected future reward:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- $V^{\pi}(s_t)$ is the value of state s_t under policy π
- γ is a discount factor [0..1]

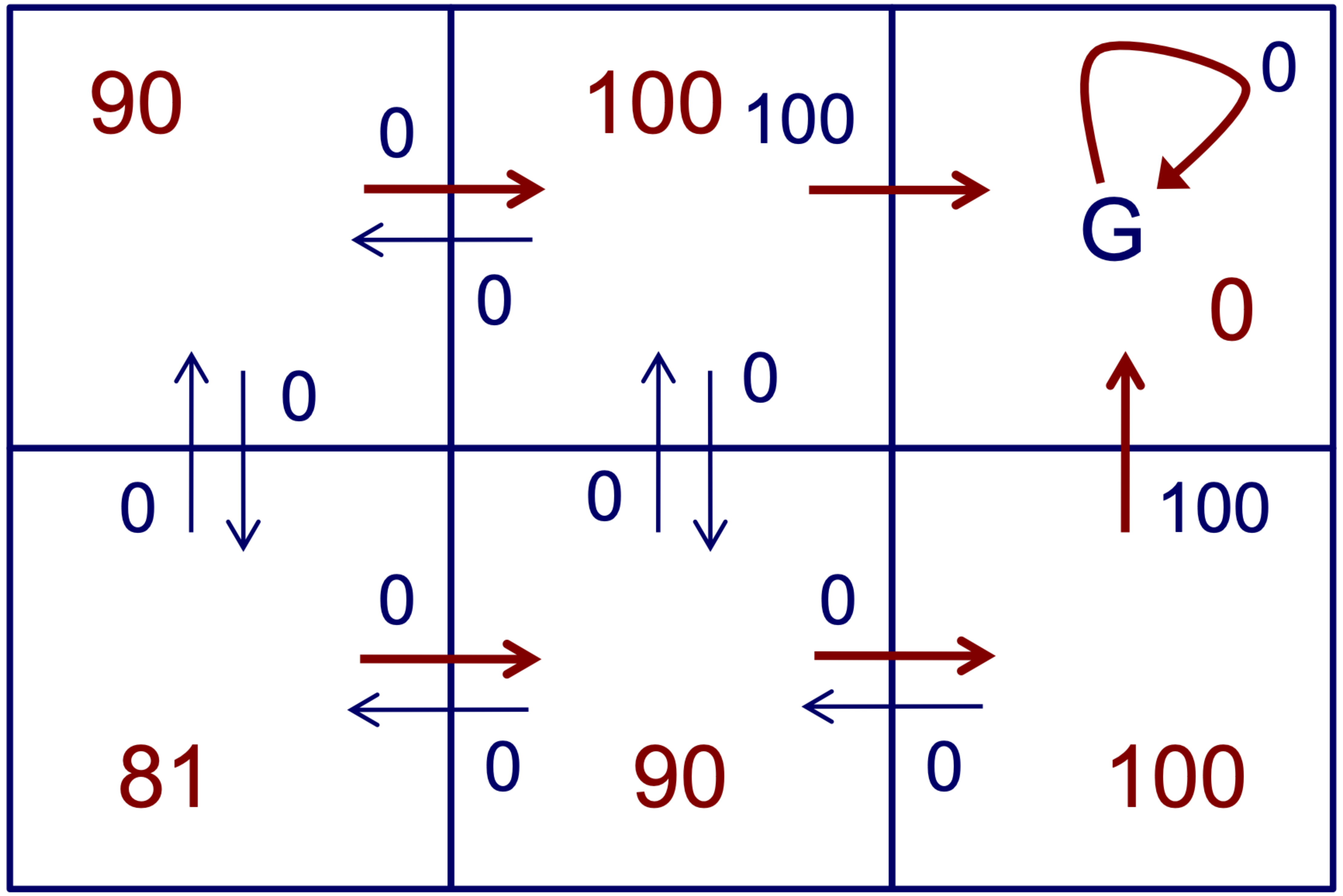
Value function

- $V^{\pi}(s)$ is the expected value of following policy π in state s
- $V^{*}(s)$ be the maximum discounted reward obtainable from s.
  - i.e. the value of following the optimal policy
- We make the simplification that actions are deterministic, but we don't know which action to take.
  - Other RL algorithms relax this assumption

Value function

- The red arrows show π^*, the optimal policy, with $\gamma = 0.9$
- $V^*(s)$ values shown in red

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$



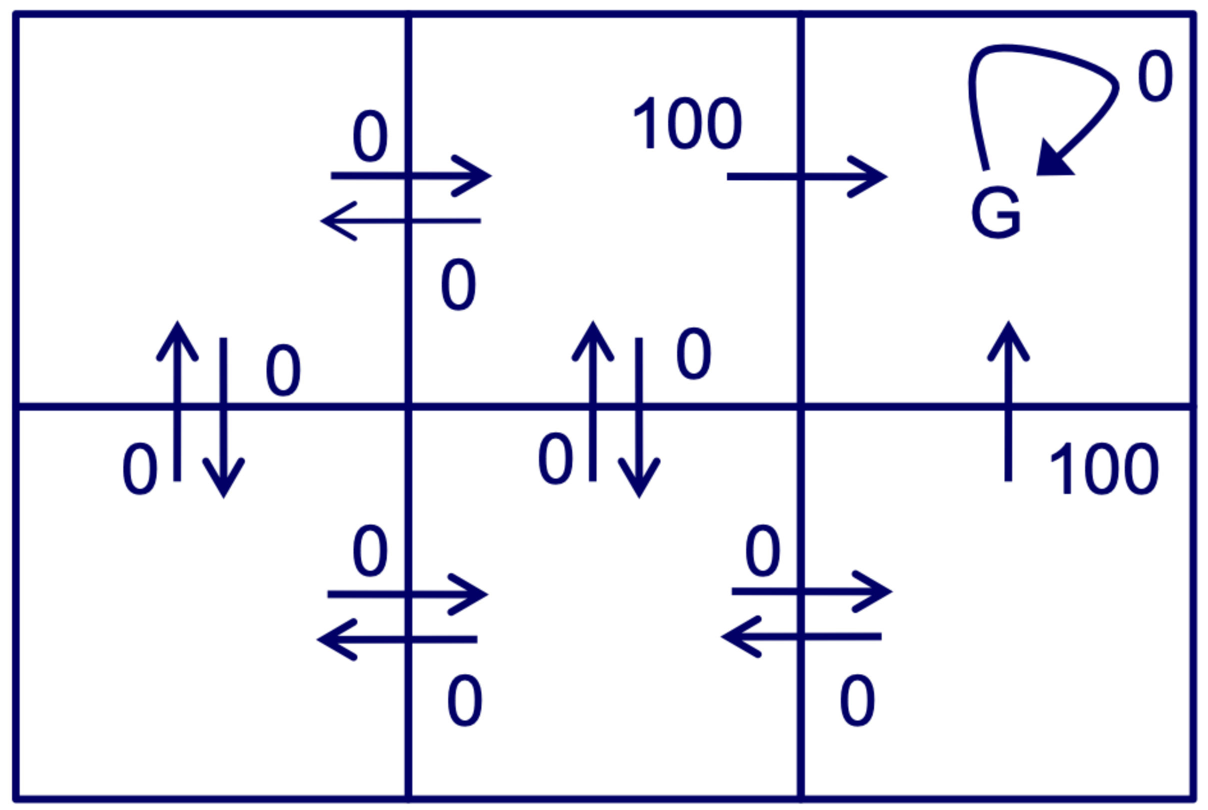
Q-values

- How to choose an action in a state?

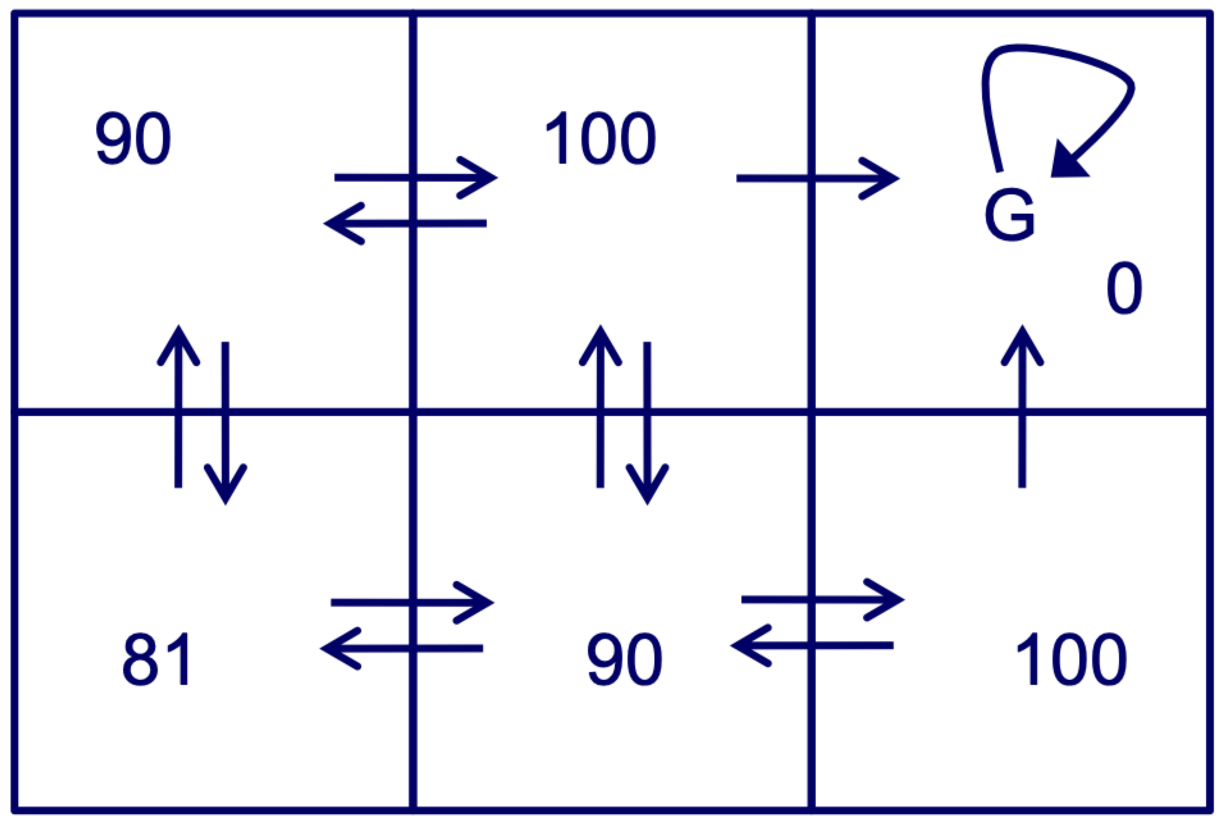
$$Q(s, a) = r(s, a) + \gamma V^*(s')$$

- The Q-value for an action, a, in a state, s, is the immediate reward for the action plus the discounted value of following the optimal policy after that action
- V^* is value obtained by following the optimal policy
- $s' = \delta(s, a)$ is the succeeding state, assuming the optimal policy

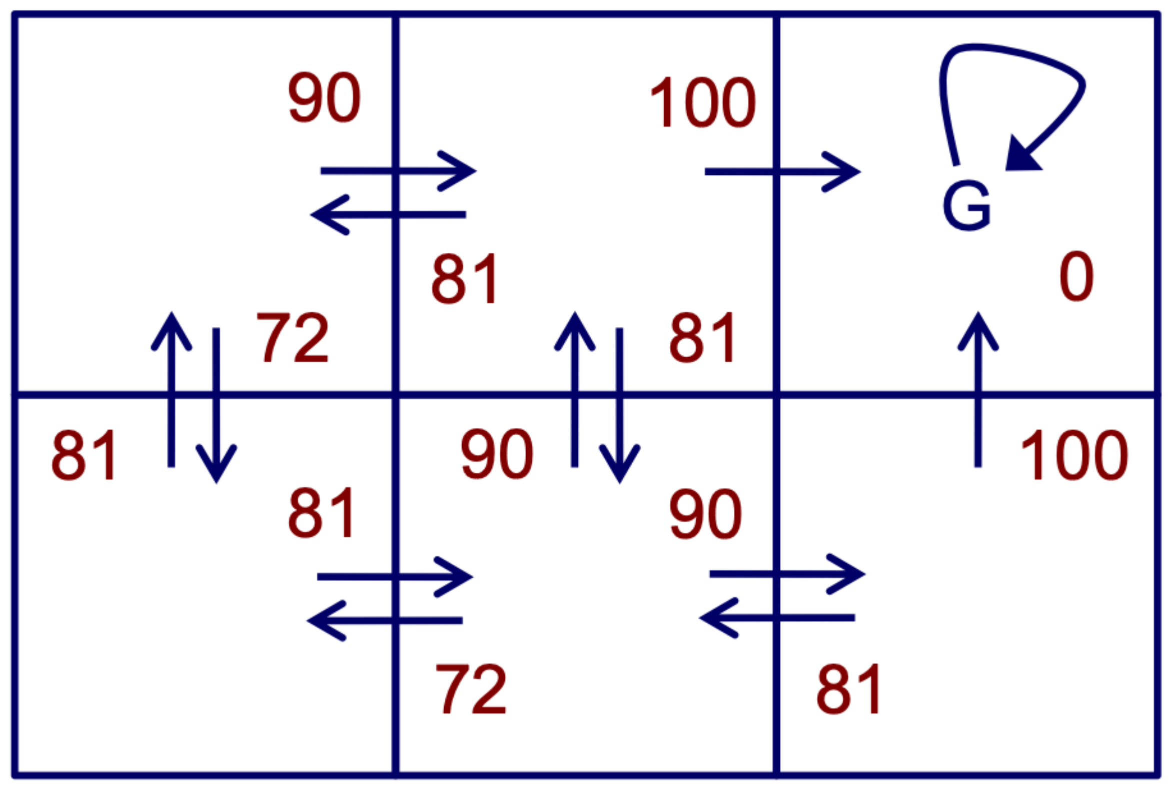
Q-values



$r(s, a)$ (immediate reward) values



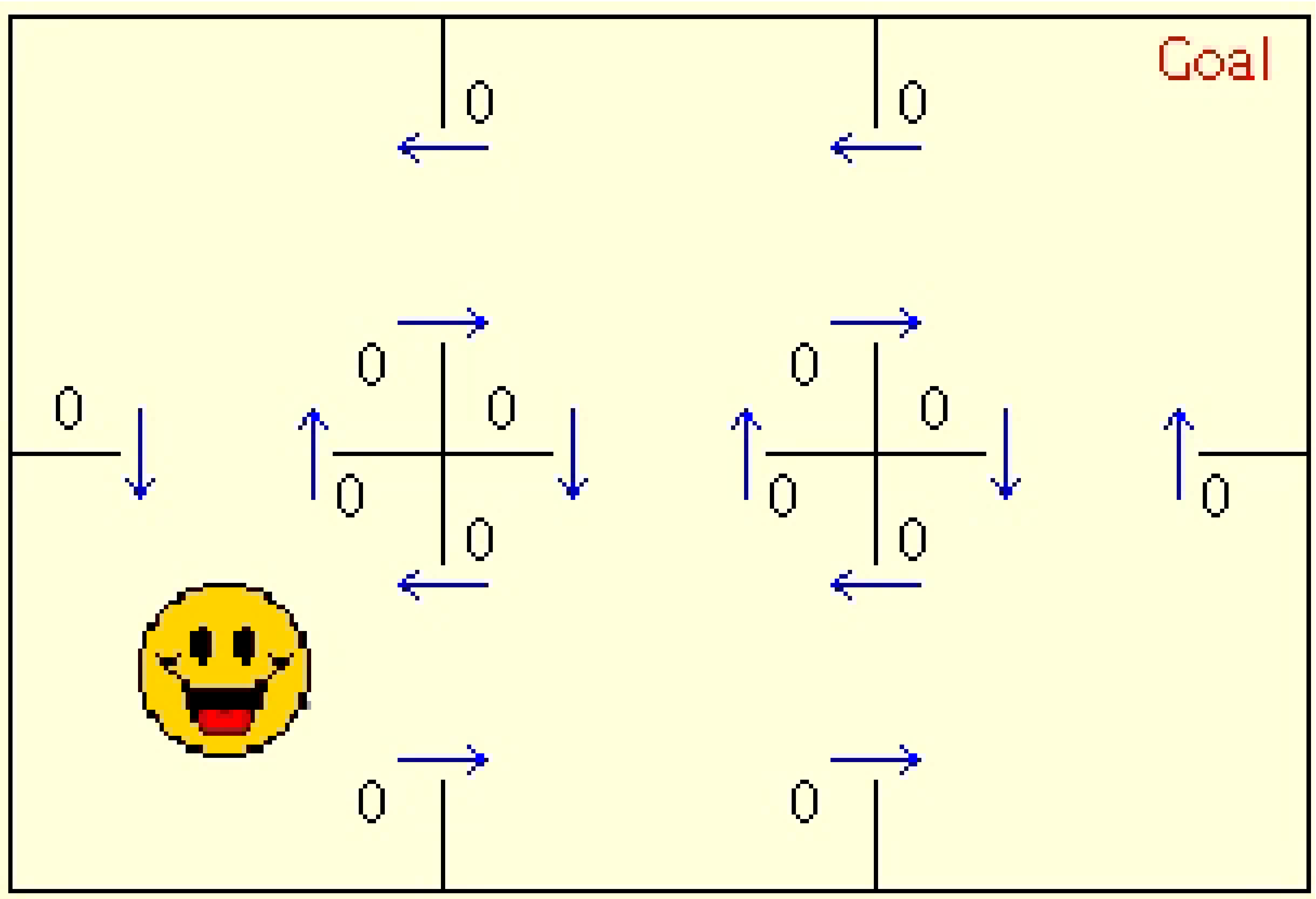
$V^*(s)$ values



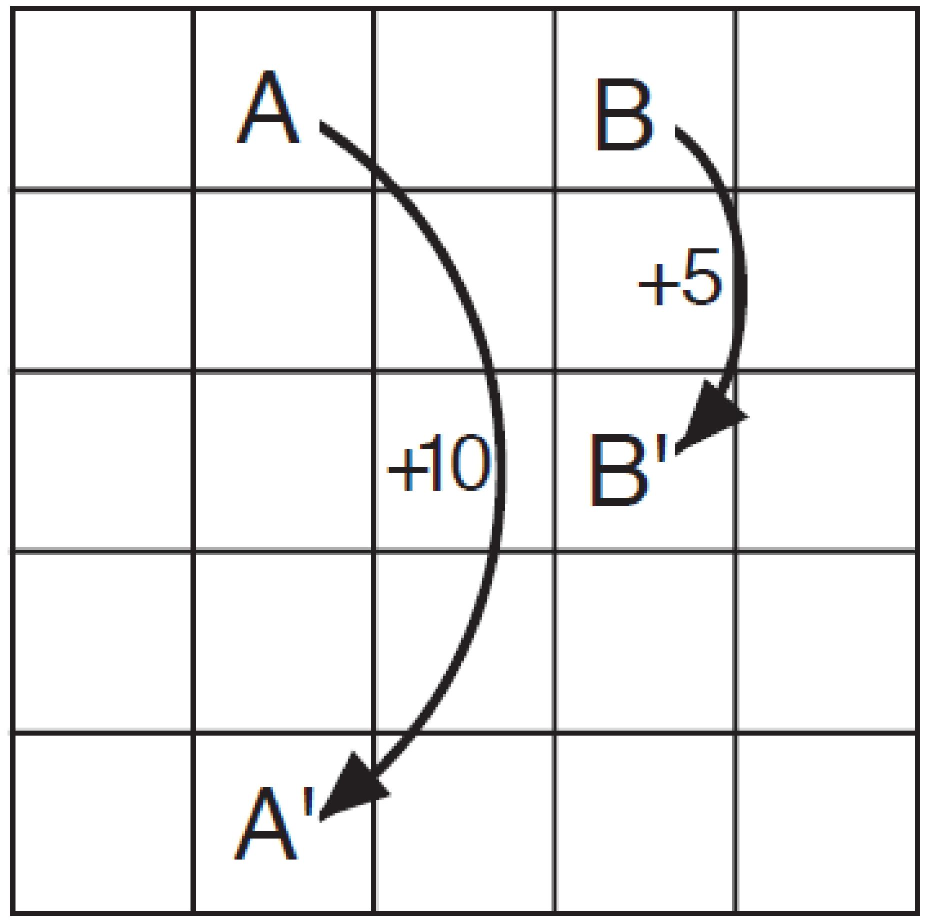
$Q(s, a)$ values

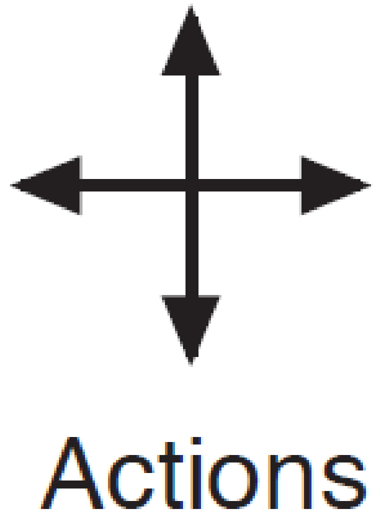
$\gamma = 0.9$

Grid world example



Another grid world example

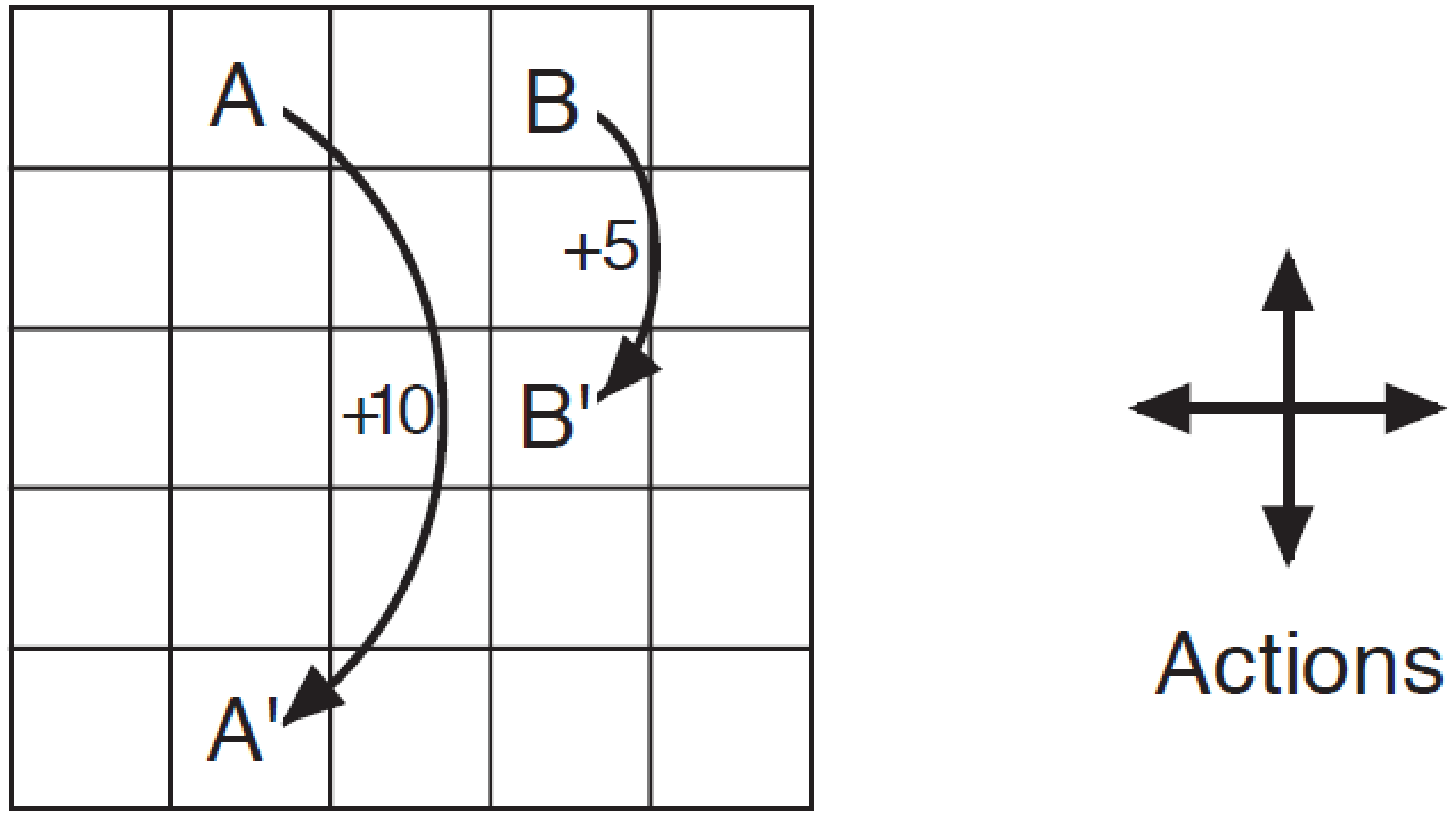




3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- Cells correspond to the states.
- 4 possible actions.
- Actions leading the agent out of the environment do not change the position but give reward = -1.
- All other actions give reward = 0, except movements from A and B.

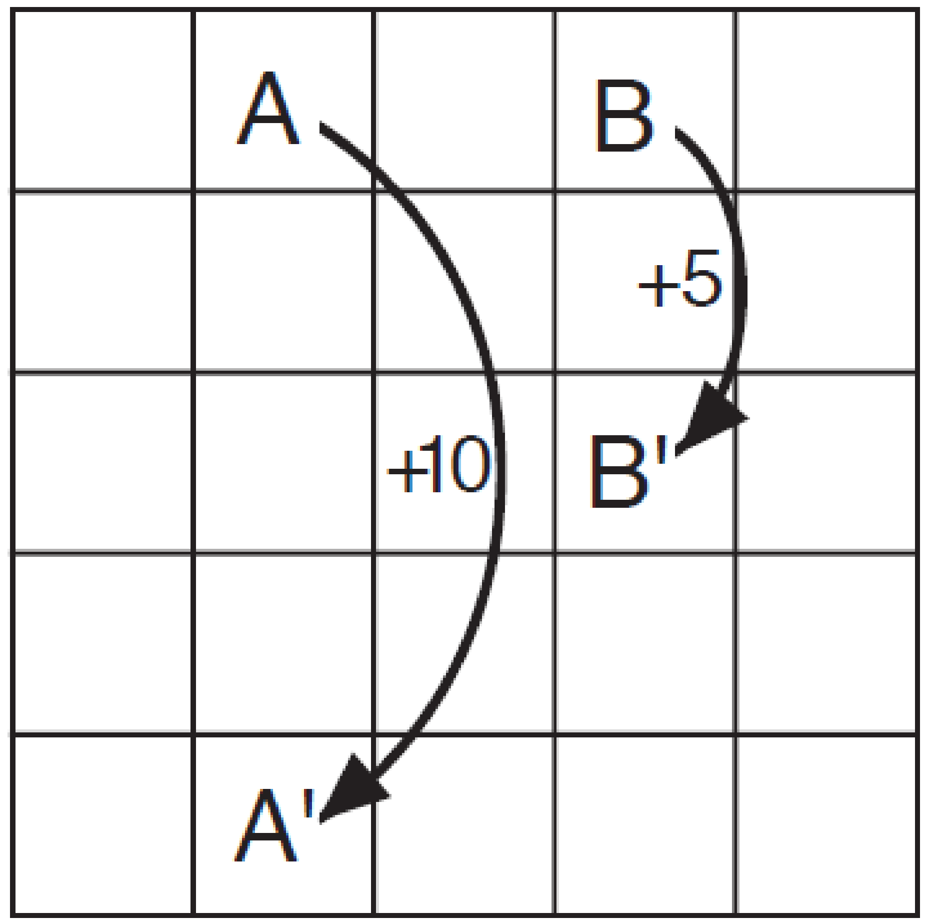
Another grid world example

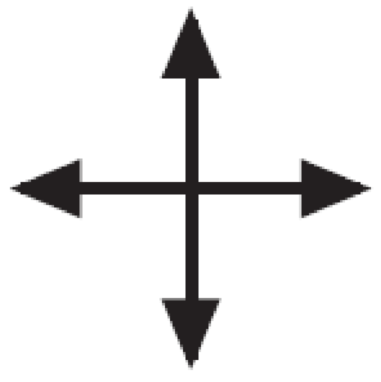


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- All actions equal probability.
- Discount factor γ = 0.9.
- Negative values near the lower edge.
- The best state is A, but expected return is lower than 10, the immediate reward.
- B is valued more than 5, the immediate rewards.

Another grid world example



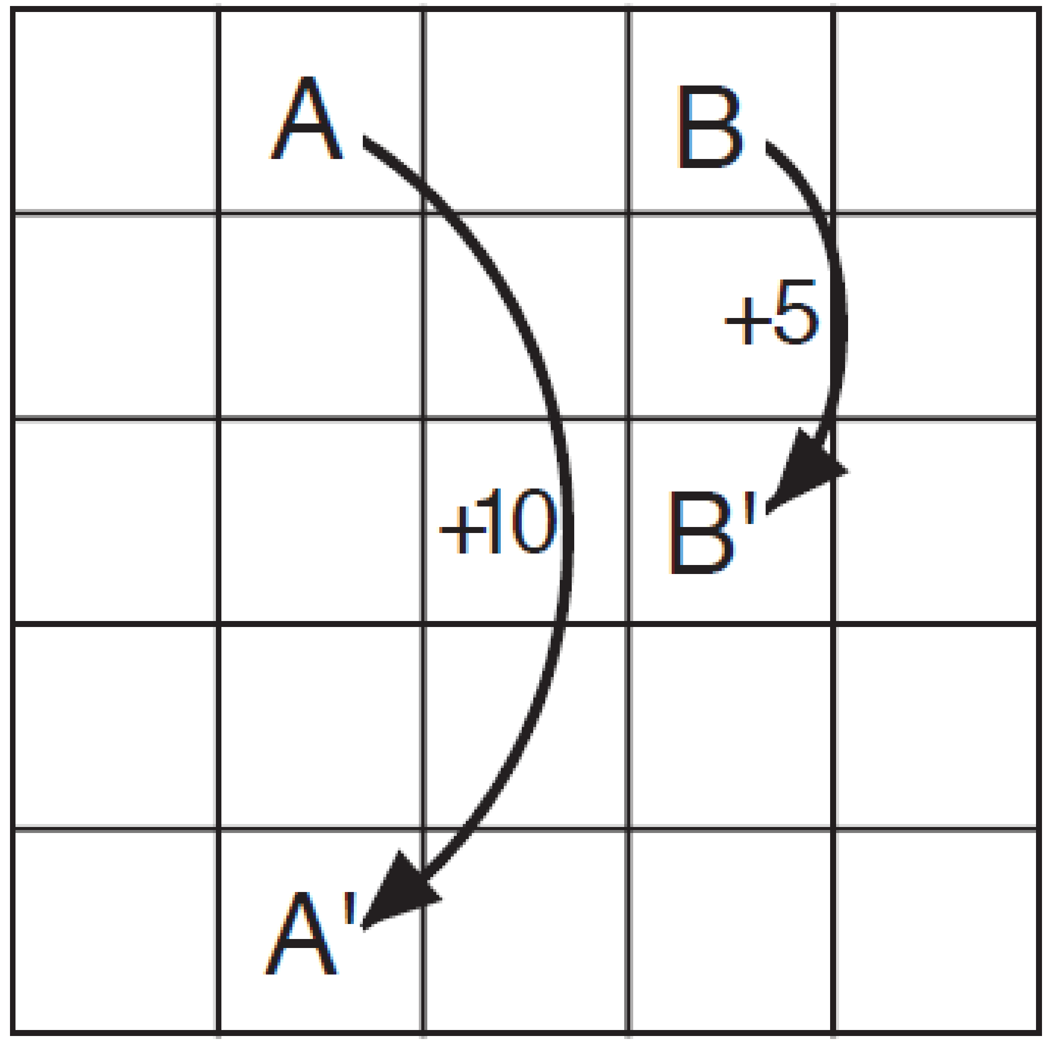


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- The best state is A, but expected return is lower than 10, the immediate reward.
- B is valued more than 5, the immediate rewards.
- Why?

2 minutes

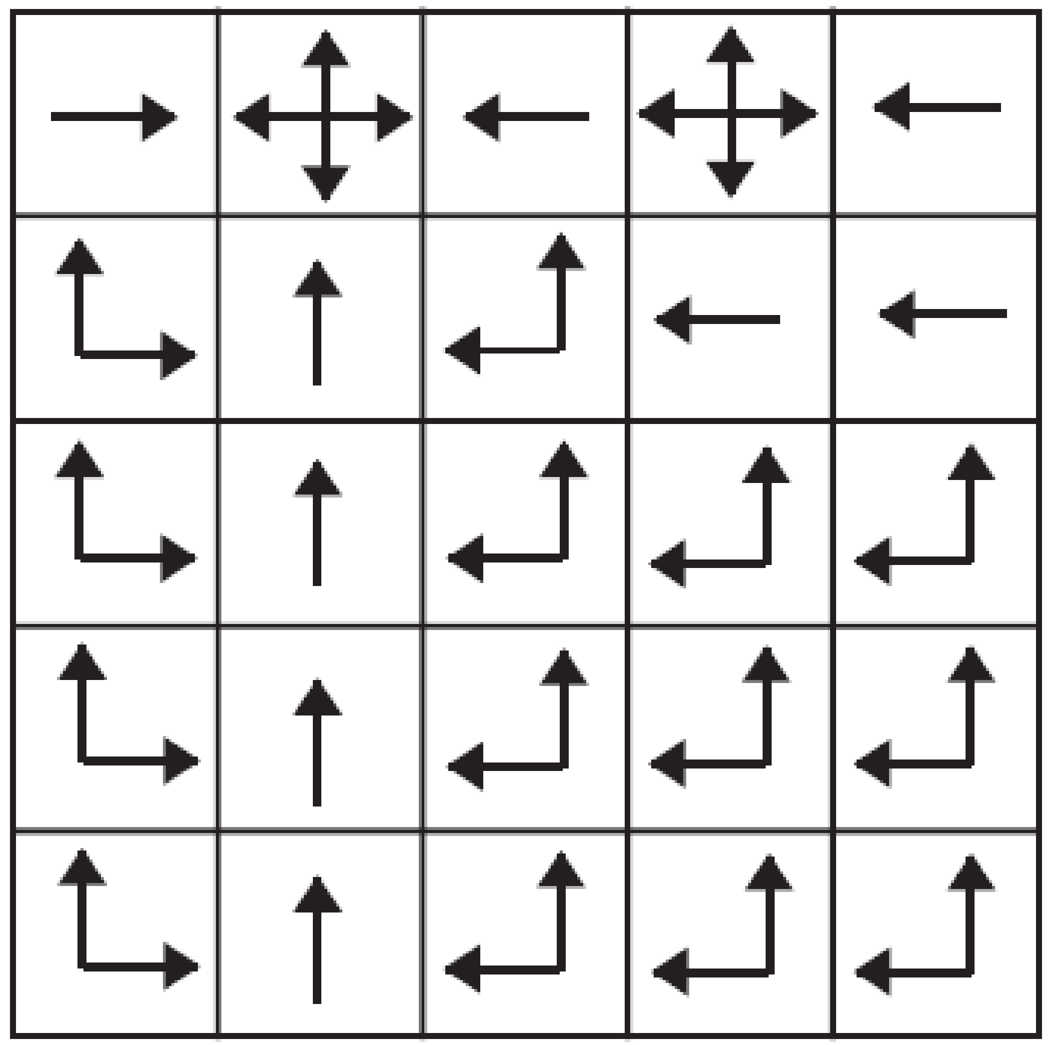
Another grid world example



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

- Optimal value function and optimal policy for the grid world.

Lecture Overview

- Introduction
- Elements of Reinforcement Learning
- Exploration vs Exploitation
- The agent-environment interface
- Value functions
- Temporal difference prediction

Temporal-difference (TD) prediction

- TD is one central and novel idea in RL.
- Monte Carlo-like methods must wait until the end of the episode to update $V(S_t)$ – (only at that point G_t is known):

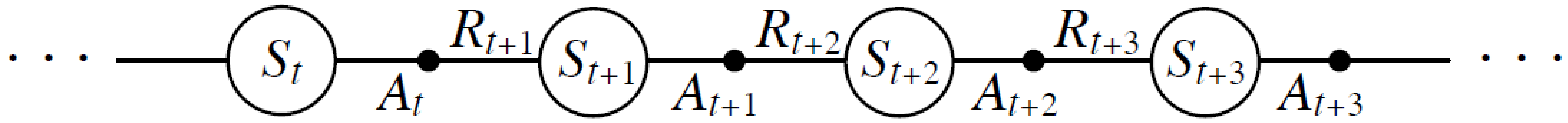
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

- The simplest TD method is called TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

Temporal-difference (TD) prediction

- Approximations can be on-policy or off-policy.
- TD control learns an action-value function instead of a state-value function.
- We estimate $q_\pi(s,a)$ for the current policy π.
- Therefore, we consider transitions from state-action pair to state-action pair.



Sarsa: On-Policy TD Control

- Updates after each transition from a non-terminal S_t.
- If S_{t+1} is terminal, $Q(S_{t+1}, A_{t+1})$ is defined as zero.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

- Each element of the 5-tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ is used, this gives the name to the algorithm.
- On-policy methods continuously estimate q_π for policy π, and at the same time change π greedily towards q_π.

Sarsa: On-Policy TD Control

- On-policy TD algorithm:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-Learning: Off-Policy TD Control

- A simple but important breakthrough is an off-policy TD algorithm.
- The simplest way is *one-step Q-learning*:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

- The learned action-value function Q directly approximates q*, the optimal action-value function, regardless the followed policy π.
- The policy still has an effect in which state-action pairs are visited and updated.

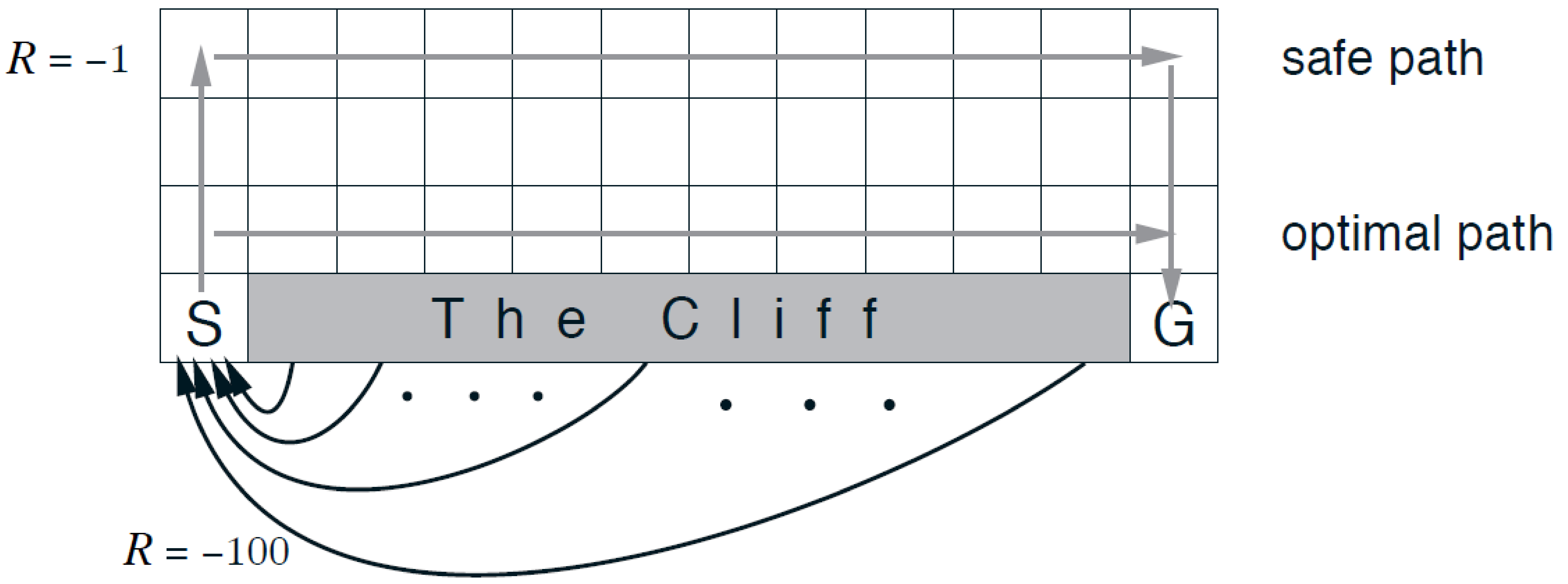
Q-Learning: Off-Policy TD Control

- Off-policy TD algorithm:

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$;
  until S is terminal

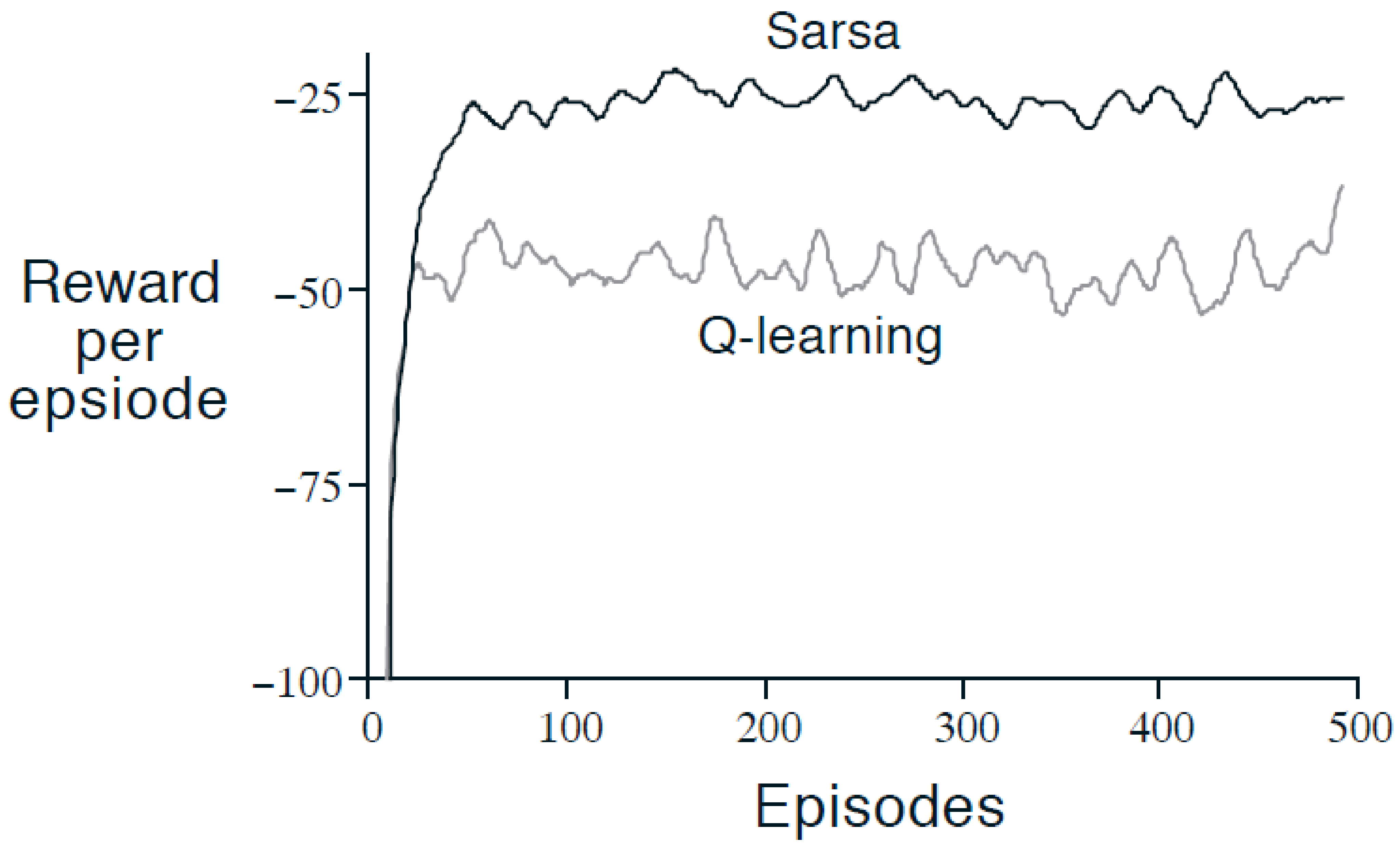
The Cliff Walking

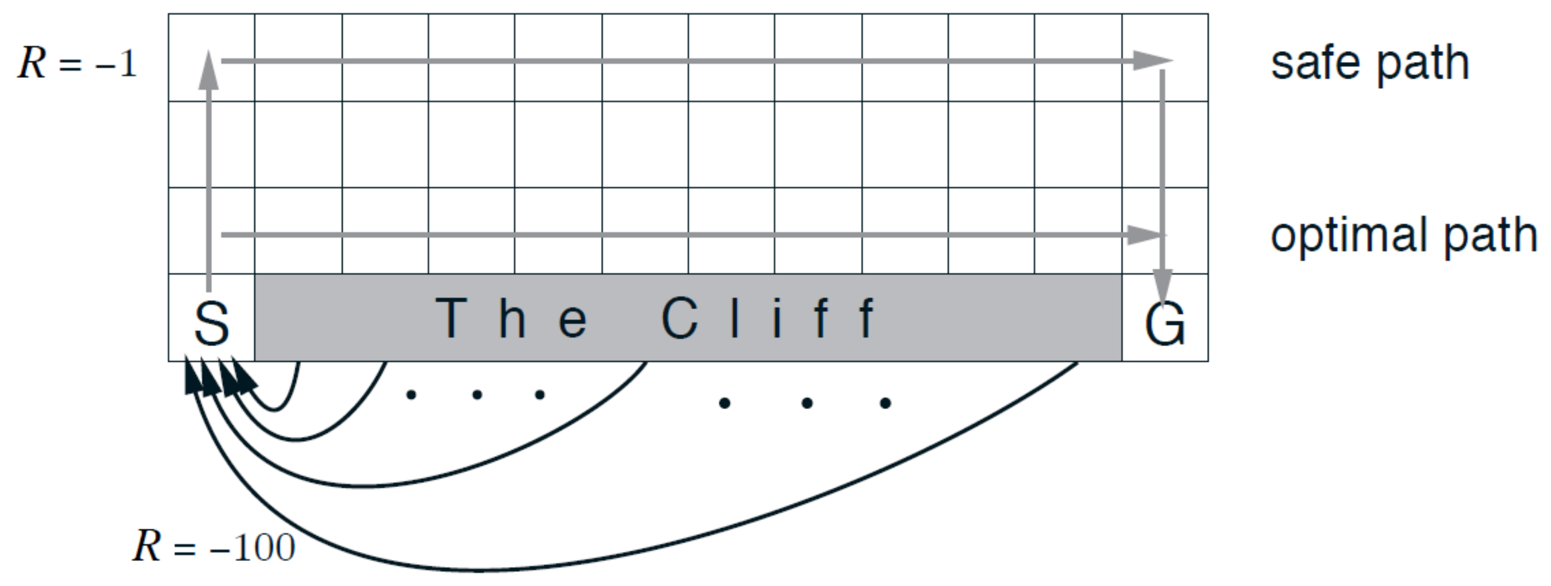
- Reward of -1 for all transitions, except in the cliff.
- The cliff gives a negative reward of -100 and sends the agent back to the start position.



The Cliff Walking

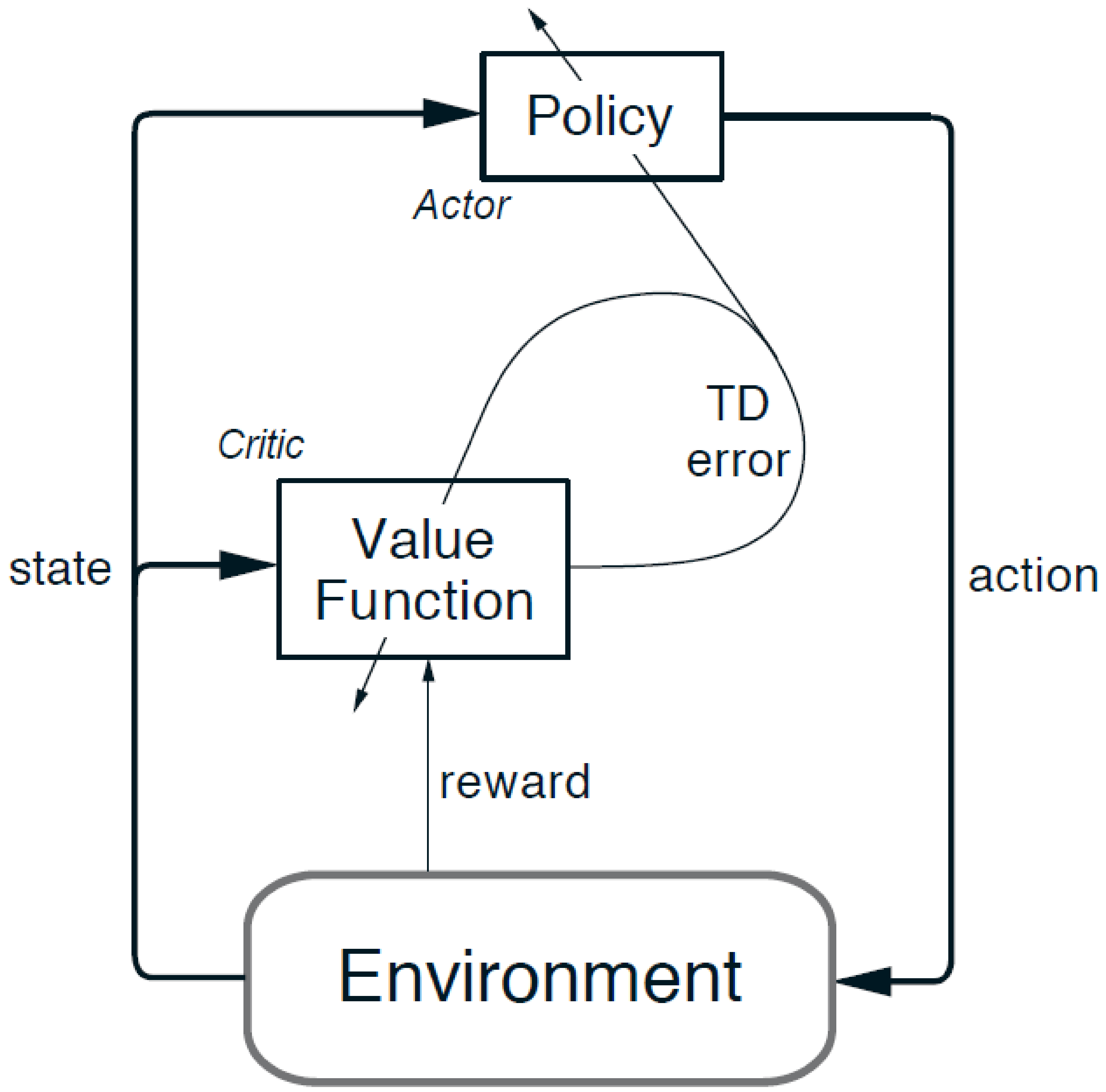
- ε-greedy, with ε=0.1.
- Q-learning learns the optimal path. Sarsa learns the longest, safest path.
- However, overall Q-learning behaviour is worse.
- If ε is gradually reduced, both methods converge asymptotically to the optimal policy.





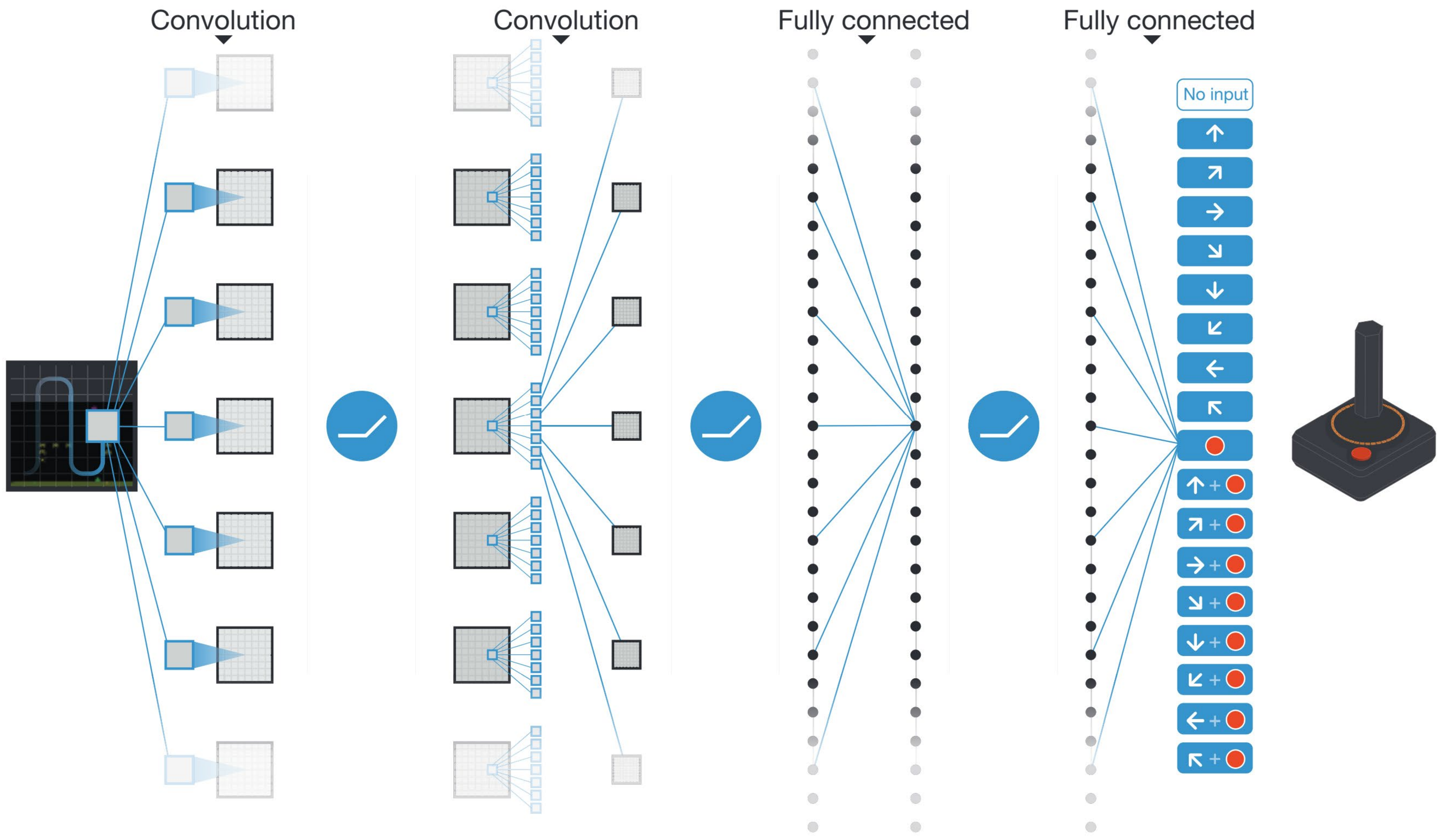
Actor-critic methods

- Policy approximation.
- Learning is always on-policy.
- The actor structures the policy.
- The critic must learn and critique the followed policy.
- Minimal computation to select actions, even in continuous-valued actions or for stochastic policies.
- The separate actor in actor-critic is more appealing as psychological and biological models.



Deep Q-Network

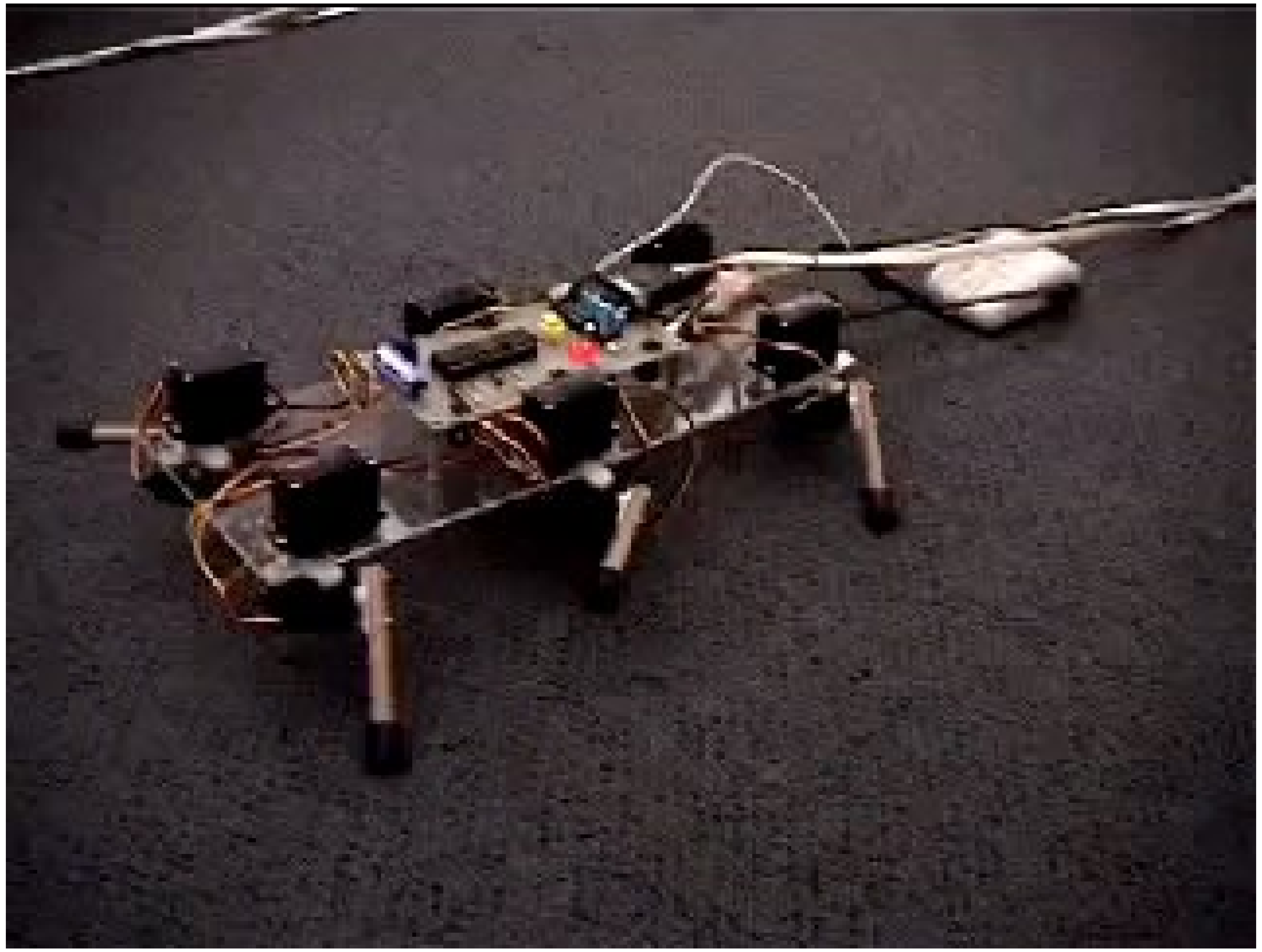
- Proposed by Mnih et al. in 2015.
- Human-level control through deep reinforcement learning.
- Tested in 49 Atari games.



Examples

- **Stumpy - A simple learning robot.**

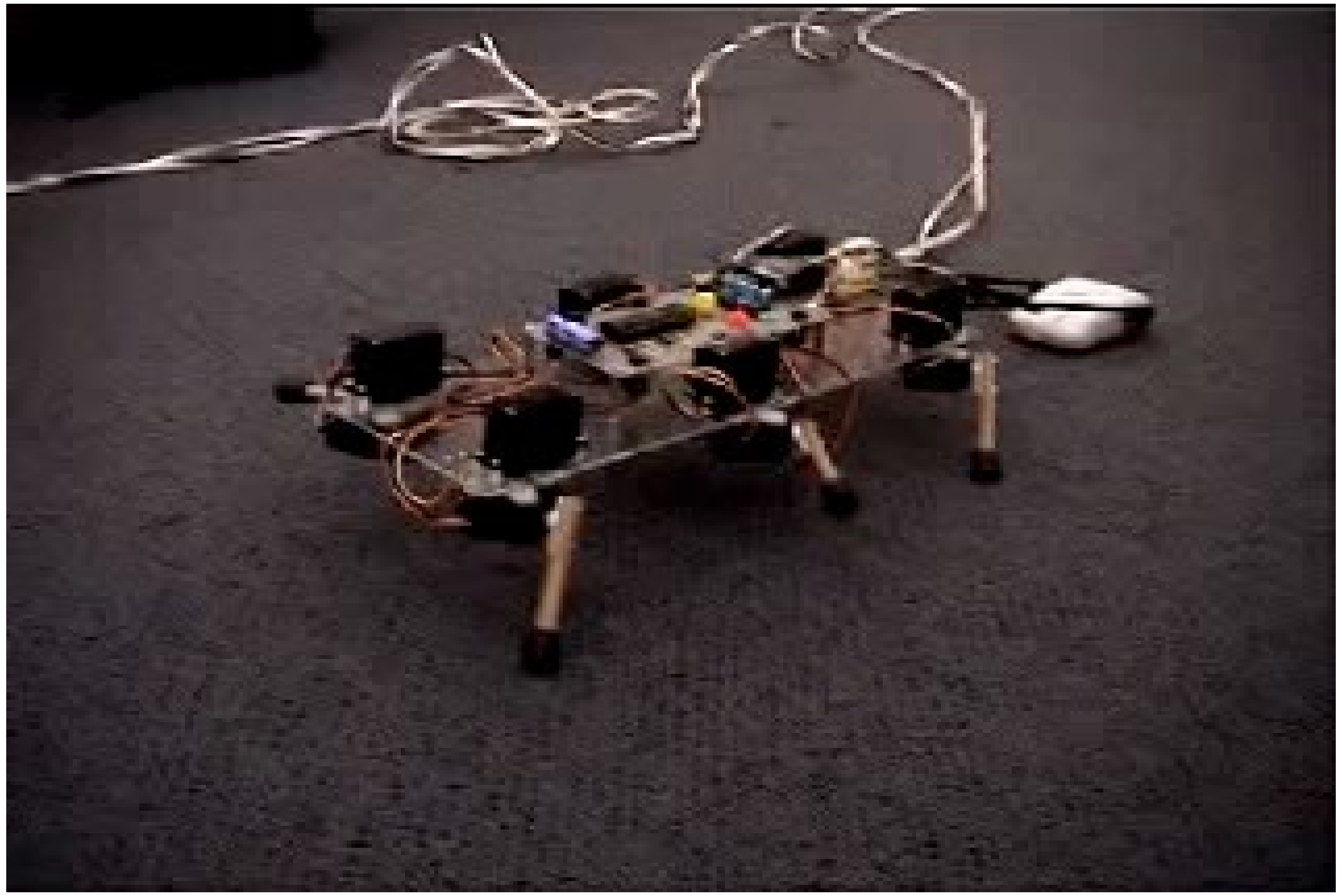
- Stumpy receives a *reward* after each action. Did it move forward or not?
- After each move, updates its policy.



Examples

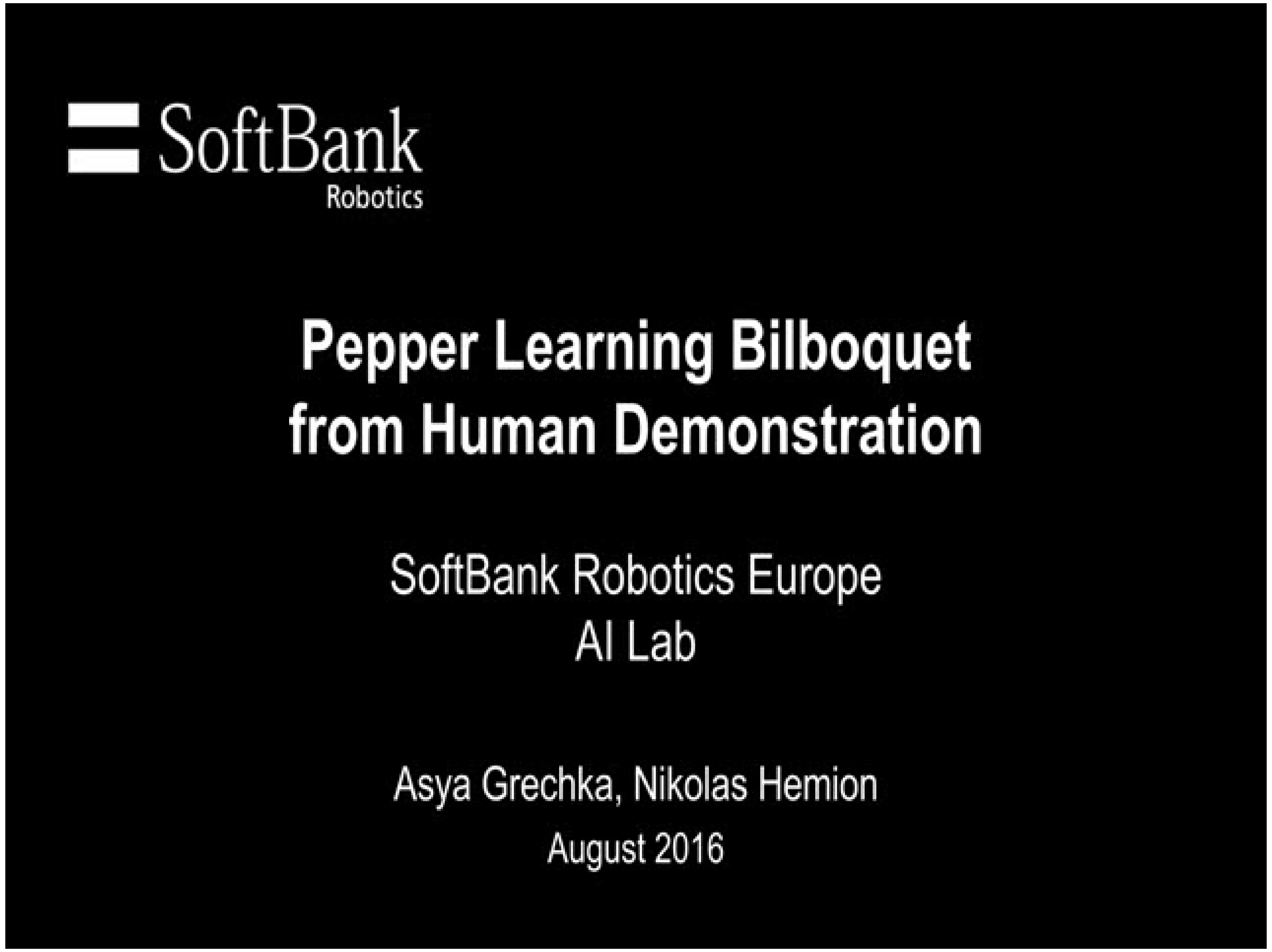
- **Stumpy - A simple learning robot.**

- Continues trying to maximise its reward.
- Stumpy after 30 minutes.



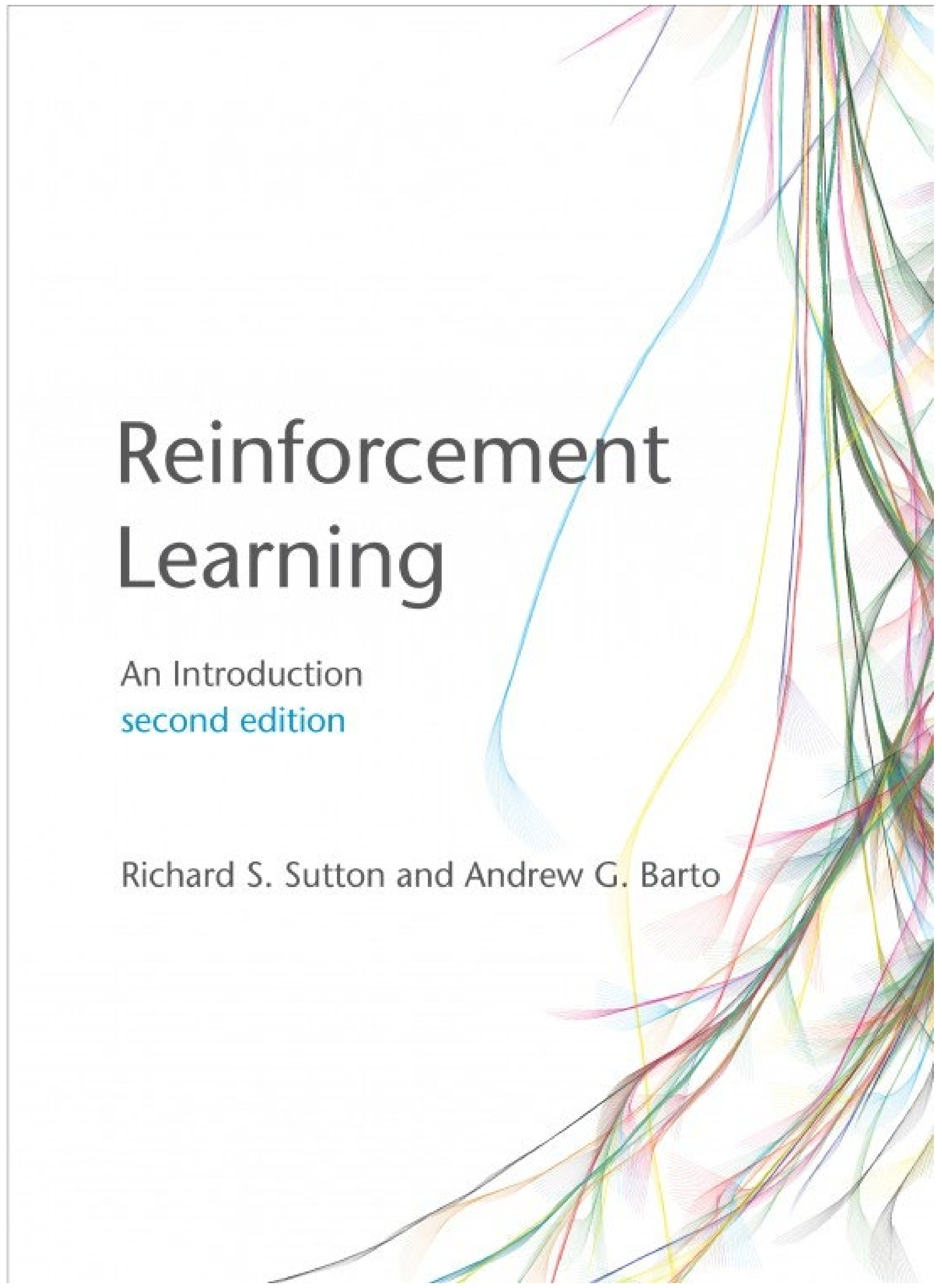
Examples

- Another example



Reference

- For a more comprehensive introduction, you should definitely have a look at:
  - Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
  - http://www.incompleteideas.net/book/the-book-2nd.html

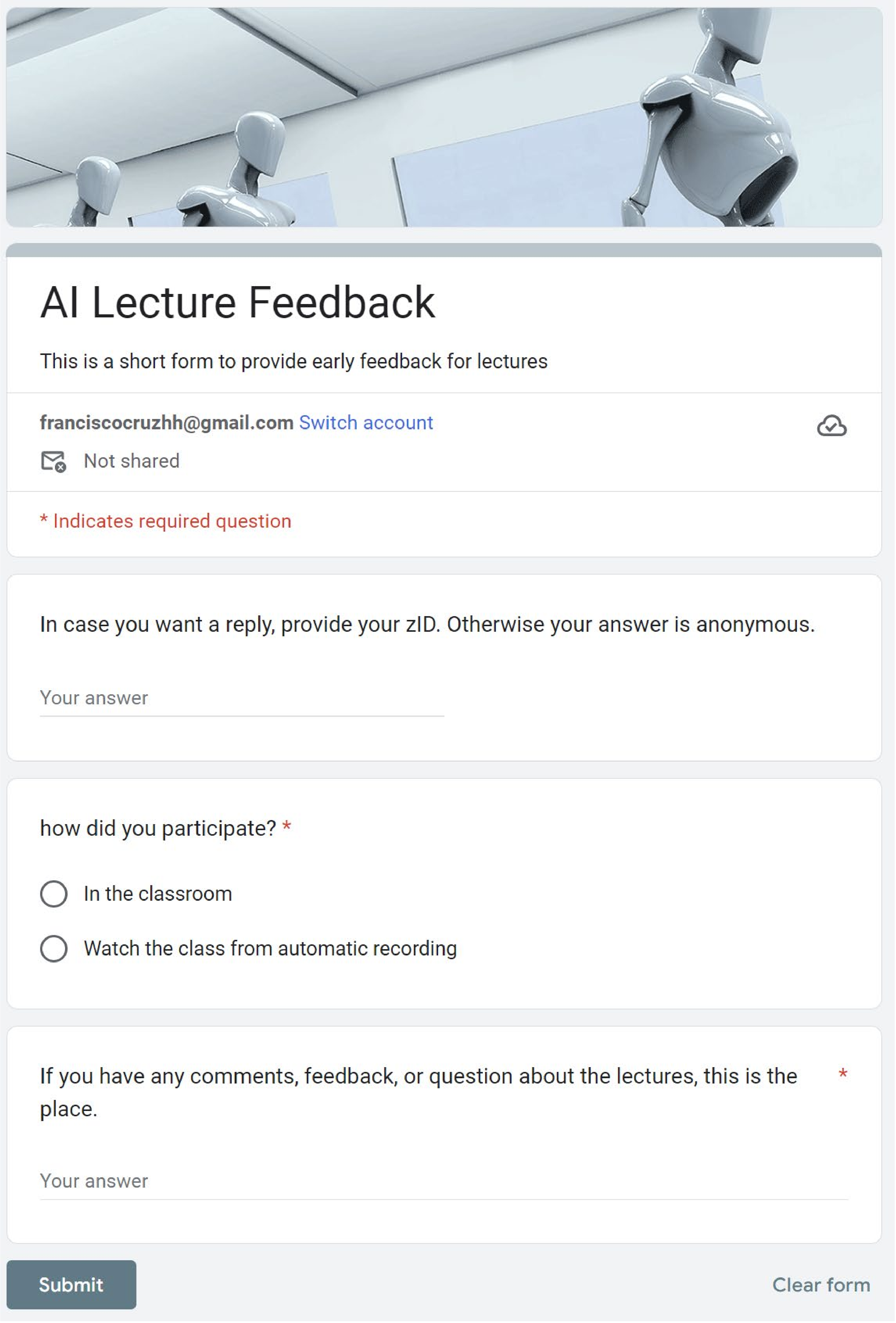


Feedback

- In case you want to provide anonymous feedback on these lectures, please visit:

- https://forms.gle/KBkN744QuffuAZLF8

Muchas gracias!


AI Lecture Feedback

This is a short form to provide early feedback for lectures

franciscocruzhh@gmail.com Switch account

Not shared

* Indicates required question

In case you want a reply, provide your zID. Otherwise your answer is anonymous.

Your answer

how did you participate? *

- In the classroom
- Watch the class from automatic recording

If you have any comments, feedback, or question about the lectures, this is the place. *

Your answer

Submit

Clear form