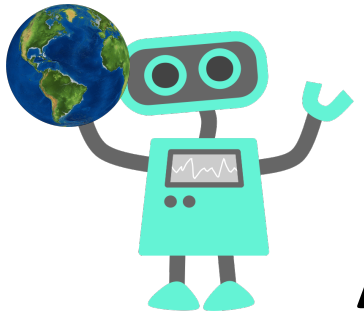


An introduction to Reinforcement Learning

First Nepal Winter School in AI

Kathmandu, Nepal

December 2018

Guillermo Garcia-Hernando

Imperial College London

Today’s overview

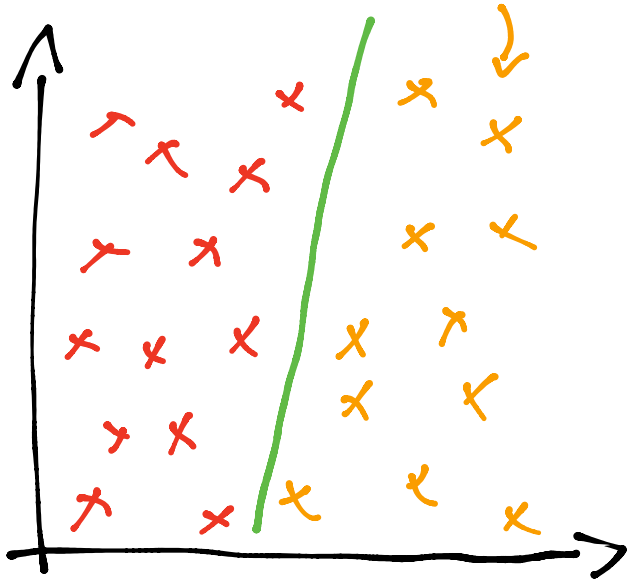
- What is reinforcement learning?
- Mathematical framework.
- Model-based reinforcement learning.
- Model free reinforcement learning.

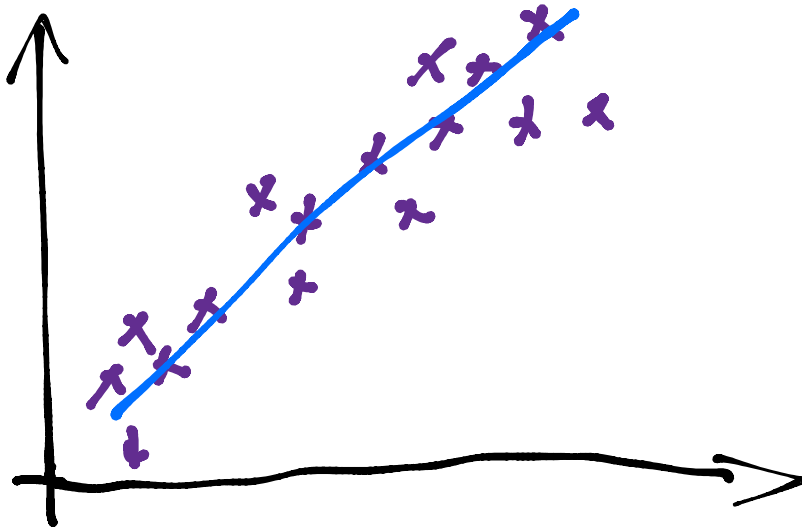
Today’s goals

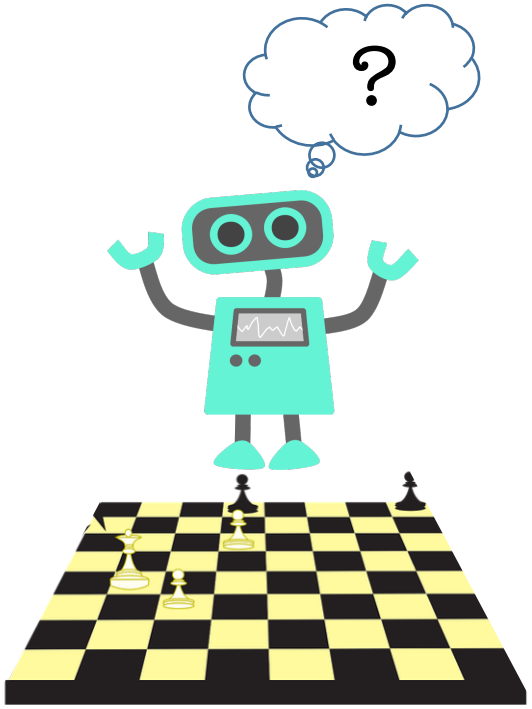
- Understand the underlying reinforcement learning key ideas.
- Understand definitions and notation.
- Get familiar with different reinforcement learning approaches.

What is Reinforcement Learning?

We've learnt how to solve many cool problems using supervised and unsupervised learning.







But, a major component of intelligence is decision making!

What is Reinforcement Learning?

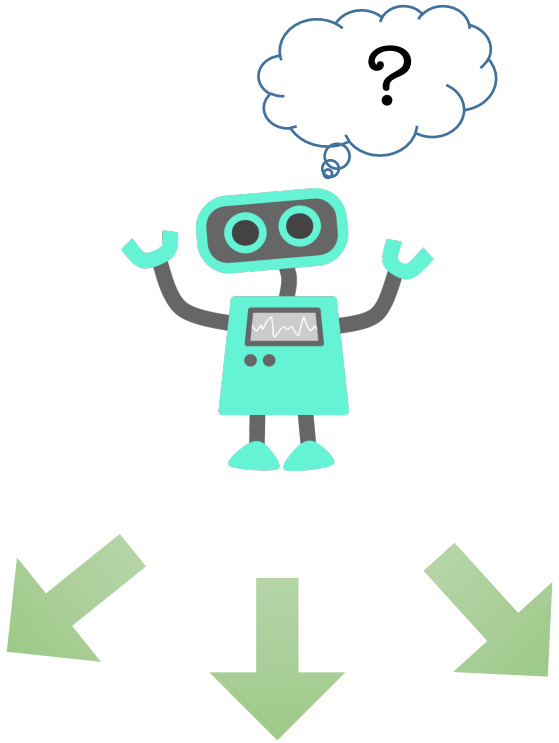
- A textbook definition…

  "RL is learning what to do so as to **maximize** a numerical **reward** signal. The learner (agent) is not told which **actions** to take, but instead must **discover** which actions yield the most reward by **trying them.**"

  [Sutton and Barto]

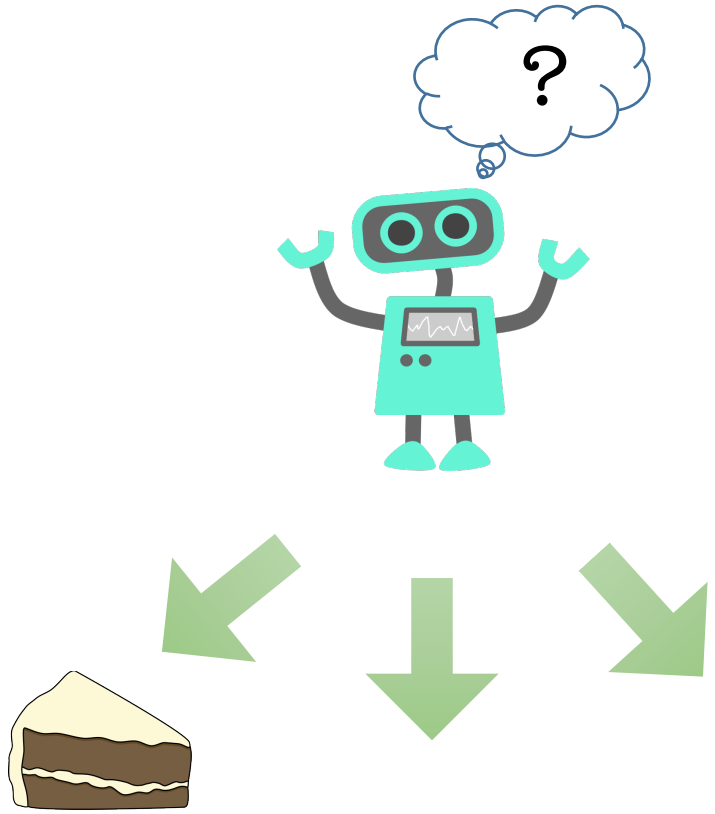
What is Reinforcement Learning?

- Reinforcement learning (RL) is the branch of machine learning relating to learning in **sequential decision making settings.**



What is Reinforcement Learning?

- Reinforcement learning (RL) is the branch of machine learning relating to learning in **sequential decision making settings.**

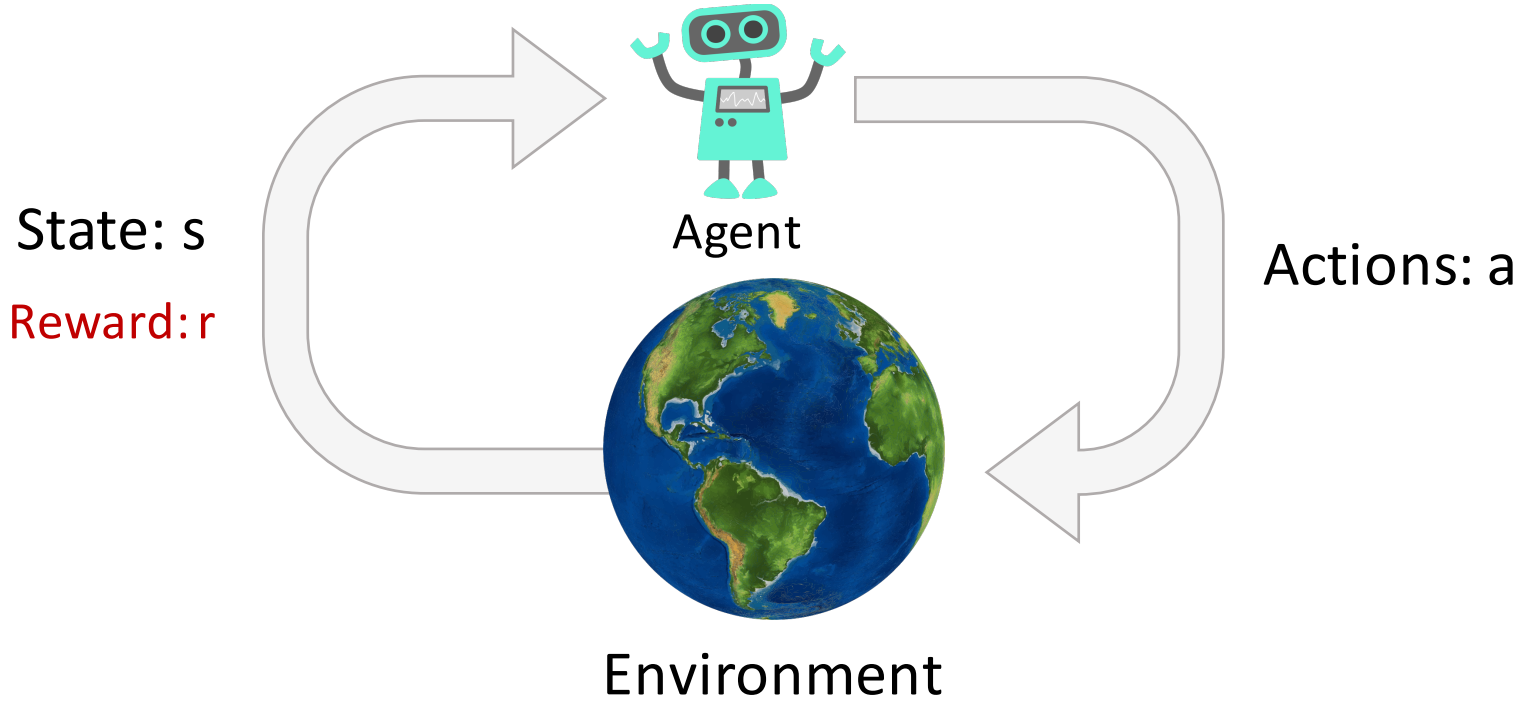


- RL is learning what to do so as to **maximize** a numerical **reward** signal.
- The agent (learner) is not told which **actions** to take.
- Instead, it must **discover** which actions yield the most reward by **trying them.**

Characteristics of Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal.
  - Feedback is delayed, not instantaneous.
  - Time really matters, sequential setting.
  - Agent's actions affect the subsequent data it receives.

RL Framework Overview



The decision maker (agent) lives within and interacts with environment:

- Agent takes actions based on the environment state.
- Environment state updates according to the agent´s action (transition).
- Agent receives a reward (feedback).

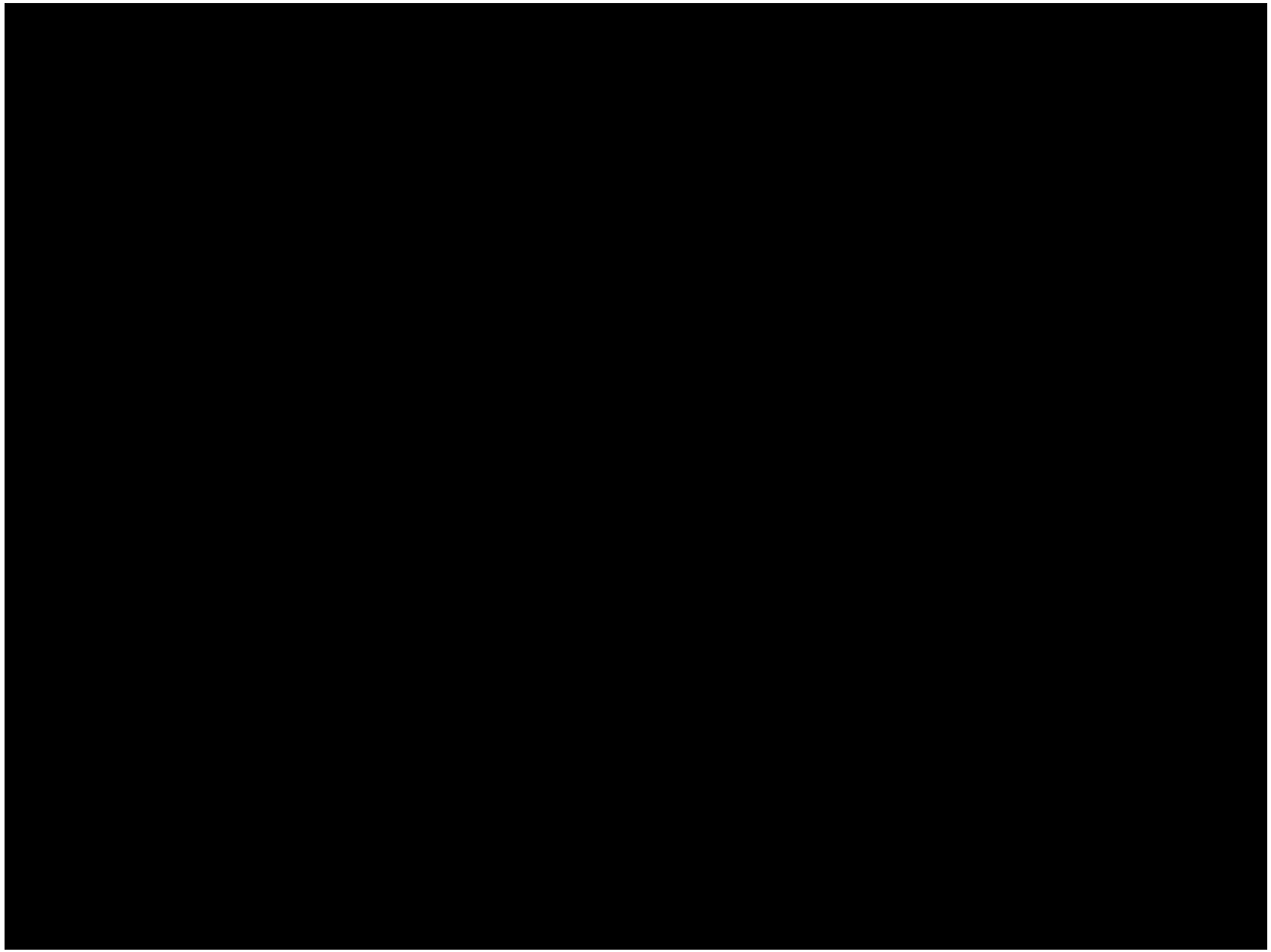
Examples of Reinforcement Learning

- Make a humanoid robot walk.
- Manage an investment portfolio.
- Play games (Chess, Go, Atari…).
- Robotic manipulation.
- Communications (previous talk by Prof. Sabita Maharjan!).

Learning locomotion with RL

[Schulman, Moritz, Levine, Jordan, Abbeel, ICLR 2016]

Mastering Atari games



Credit: DeepMind

RL Mathematical Framework

Markov Decision Process (MDP)

- Markov decision processes formally describe an environment for reinforcement learning.
- A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
  - $\mathcal{S}$ is a set of states "world configurations".
  - $\mathcal{A}$ is a set of actions "choices by the agent".
  - T is a state transition probability matrix:

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

  - R is a reward function:

$$R(s, a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$$

  - γ is a discount factor $\gamma \in [0, 1]$.

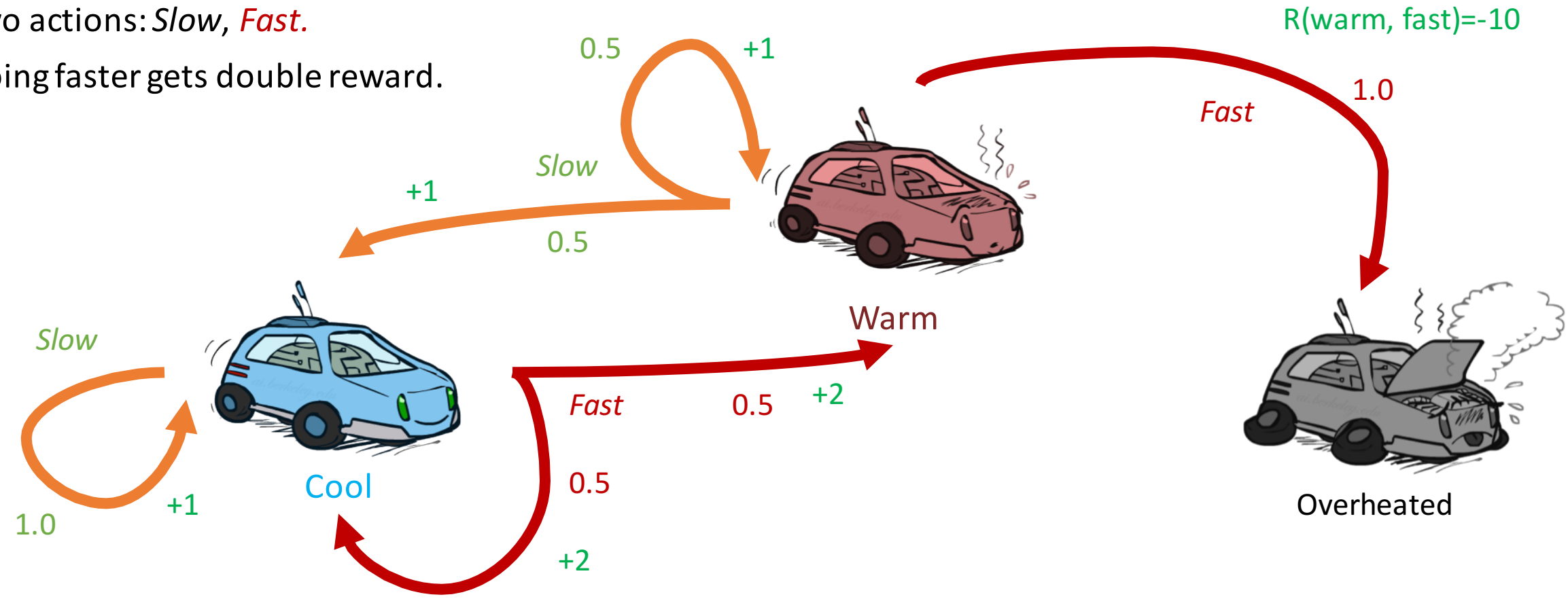
Andrey Markov
(1856-1922)

$$P(s_{t+1} | s_t) = P(s_{t+1} | s_1, \ldots, s_t)$$

Markov property "The future is independent of the past given the present".

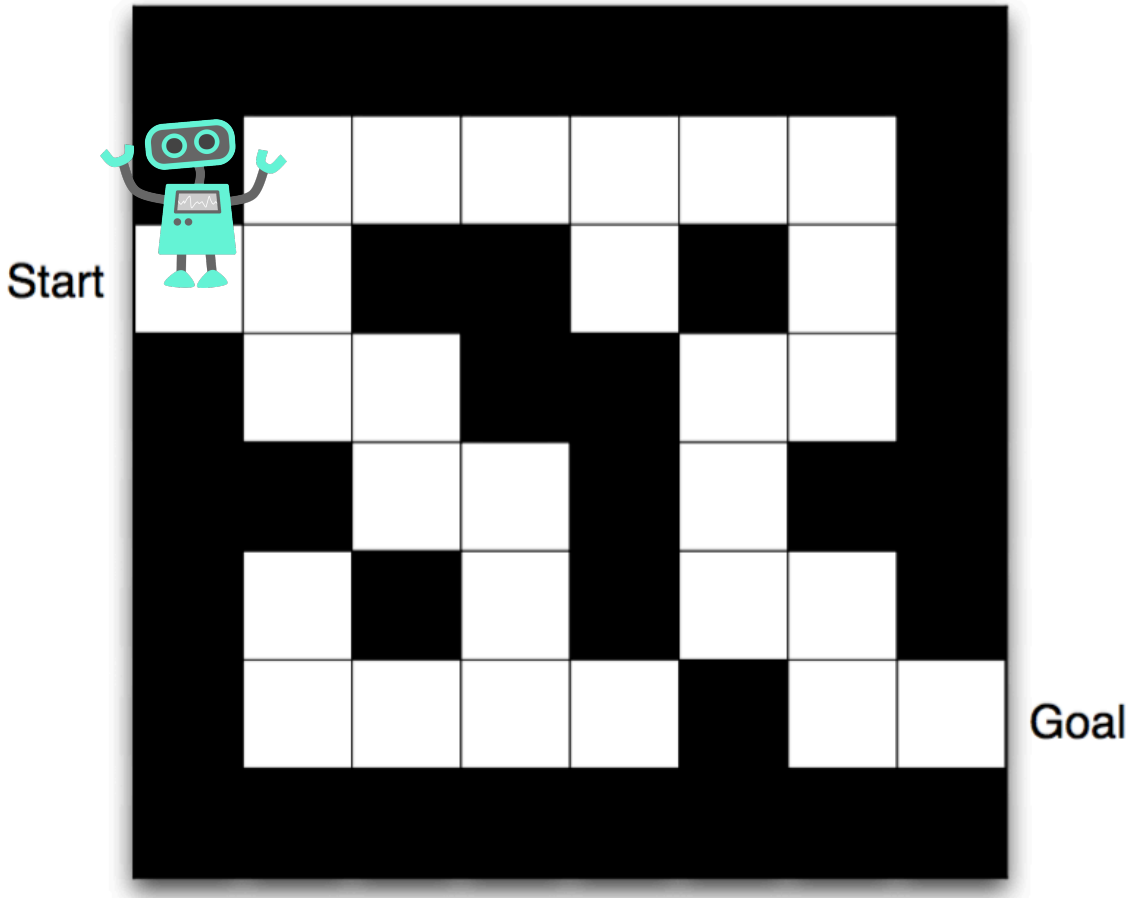
MDP Example: Racing

- A robot car wants to travel far, quickly.
- Three states: Cool, Warm, Overheated.
- Two actions: *Slow*, *Fast.*
- Going faster gets double reward.



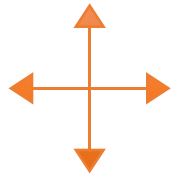
[Abbeel and Klein]

Maze Environment Example



Rewards: -1 per time step

Actions: N, E, S, W



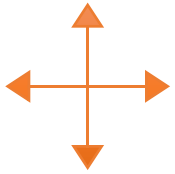
States: Agent's location in the maze

[Silver]

An example: Cleaning robot

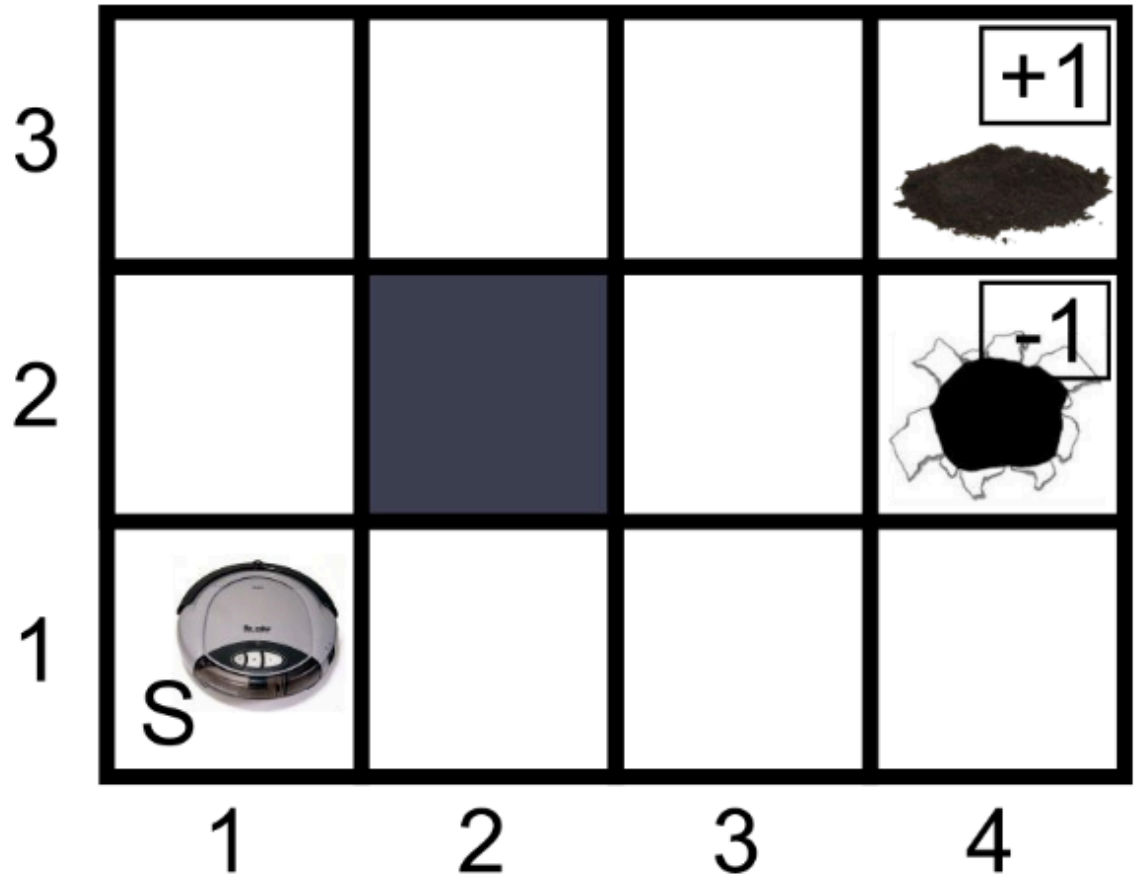
States: Position on grid e.g.
S (start) is (1,1), goal (4,3)

Actions:



Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move



[Marivate and Rosman]

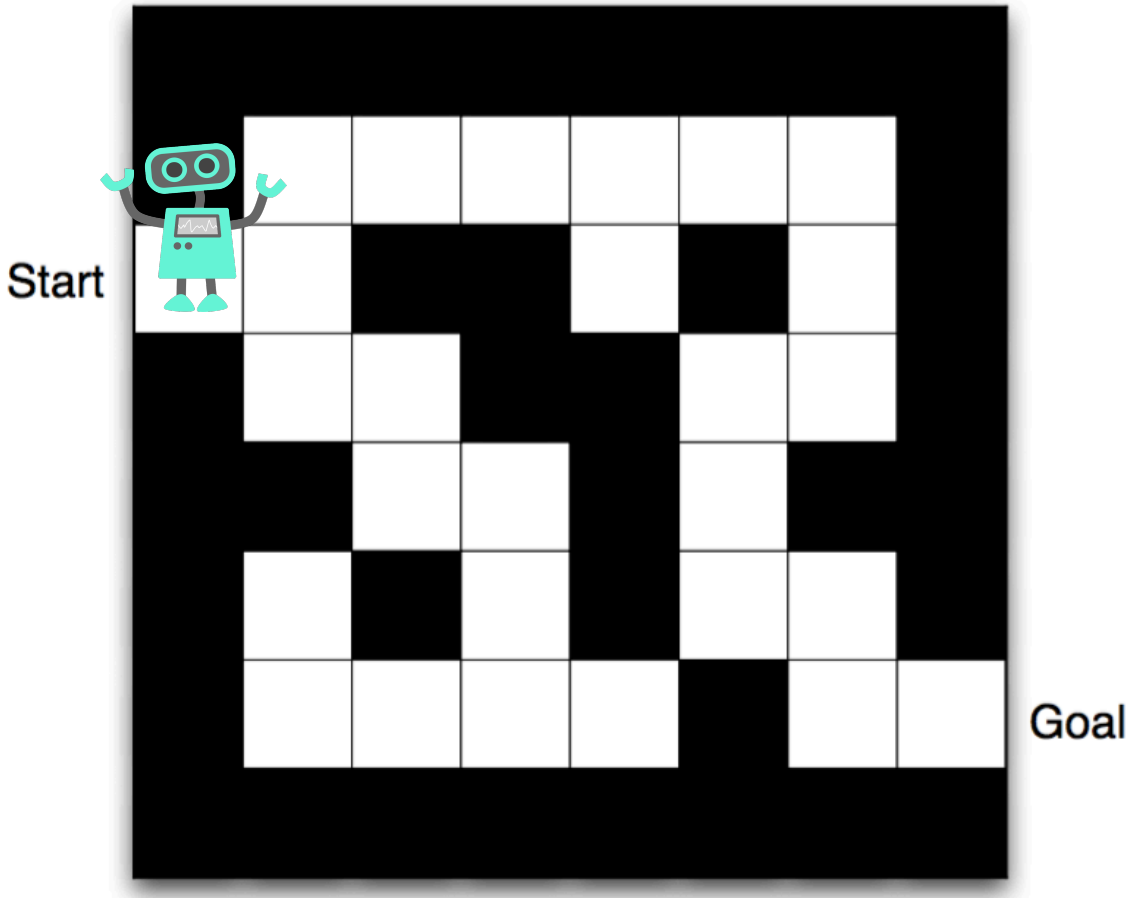
Evaluating behaviours

- An agent generates many trajectories (sequence of states-actions).
- We need to have some measure of “good” and “bad”!
- Definition: The return G_t is the total discounted reward from time-step t:

$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

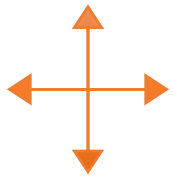
  - γ close to 0 leads to “myopic” evaluation
  - γ close to 1 leads to “far-sighted” evaluation
- Why the discount?
  - Mathematically convenient.
  - Animal/human behaviour shows preference for immediate reward.

Maze Environment Example



Rewards: -1 per time step

Actions: N, E, S, W



Assume $\gamma = 1$

Trajectory 1:

$G_t = -24$

Trajectory 2:

$G_t = -16$

About rewards

- Scalar feedback signal.
- In general, they are sparse, delayed and only have relative value.
- Some examples:
  - Make a humanoid robot walk
    - + reward for forward motion
    - – reward for falling over
  - Manage an investment portfolio
    - + reward for each ₹ in the bank
  - Play games (e.g. Atari)
    - +/- reward for increasing/decreasing score

Designing reward functions is a difficult problem and often requires domain knowledge!

Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function.
  - Value function: how good is each state and/or action.
  - Model: agent's representation of the environment.

Policy

- Definition: A policy π is a distribution over actions given states,

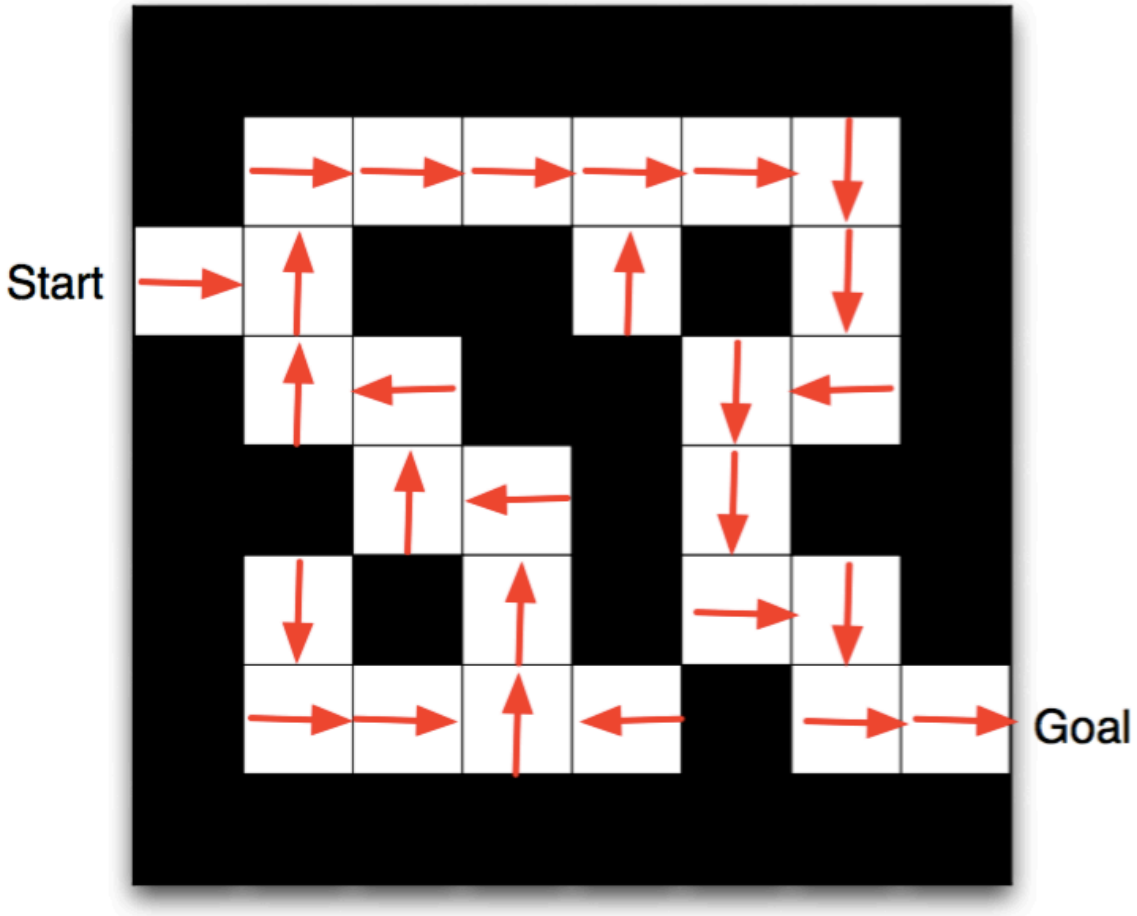
$$\pi(a|s) = P(a_t = a|s_t = s)$$

  - A policy fully defines the behaviour of an agent.
  - It can be deterministic or stochastic.
  - Policies depend on the current state.
  - Policies are stationary (time-independent).

- Optimal policy π^*: Accumulates maximal rewards over a trajectory.

This is what we want to learn!

Maze Example: Policy



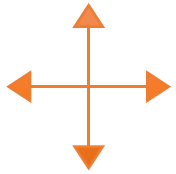
Arrows represent policy $\pi(s)$ for each state s

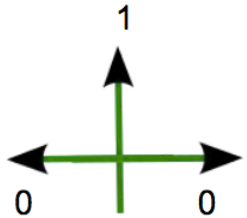
[Silver]

Cleaning Robot Example: Policy

States: Position on grid e.g.
S (start) is (1,1), goal (4,3)

Actions:

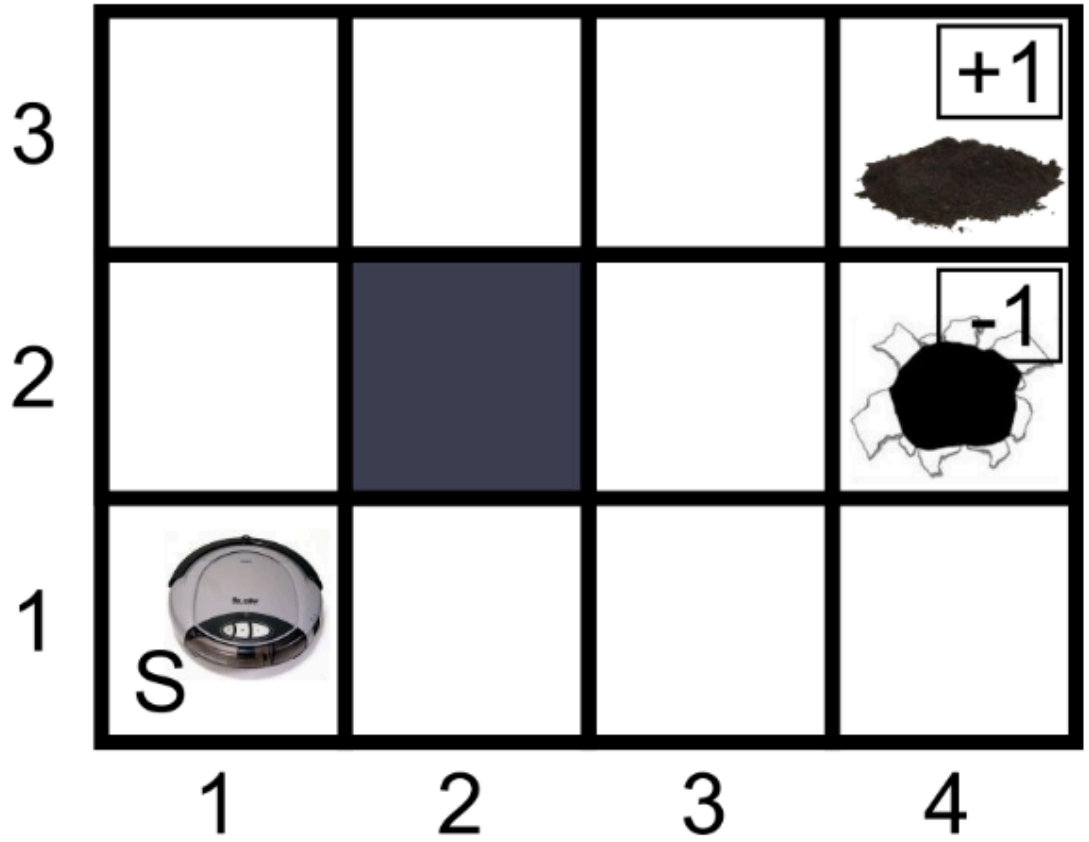




(deterministic)

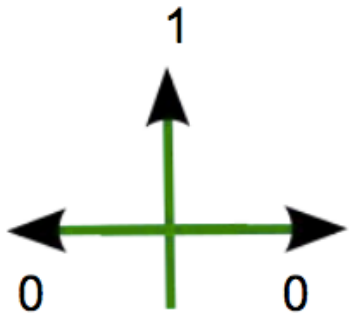
Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move

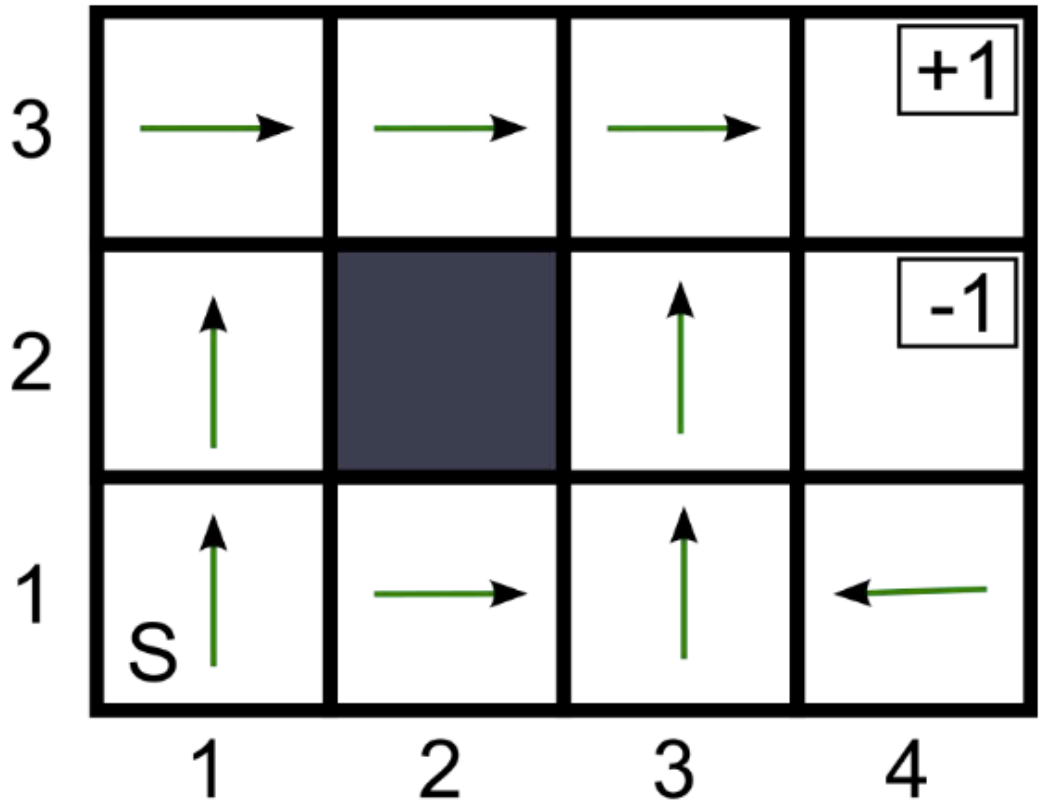


[Marivate and Rosman]

An example: Cleaning Robot

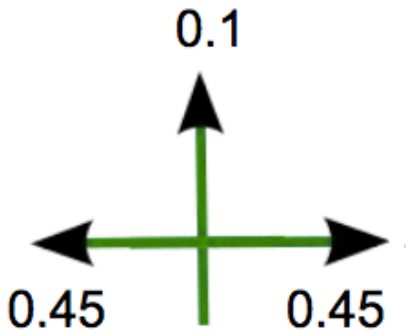


(deterministic)

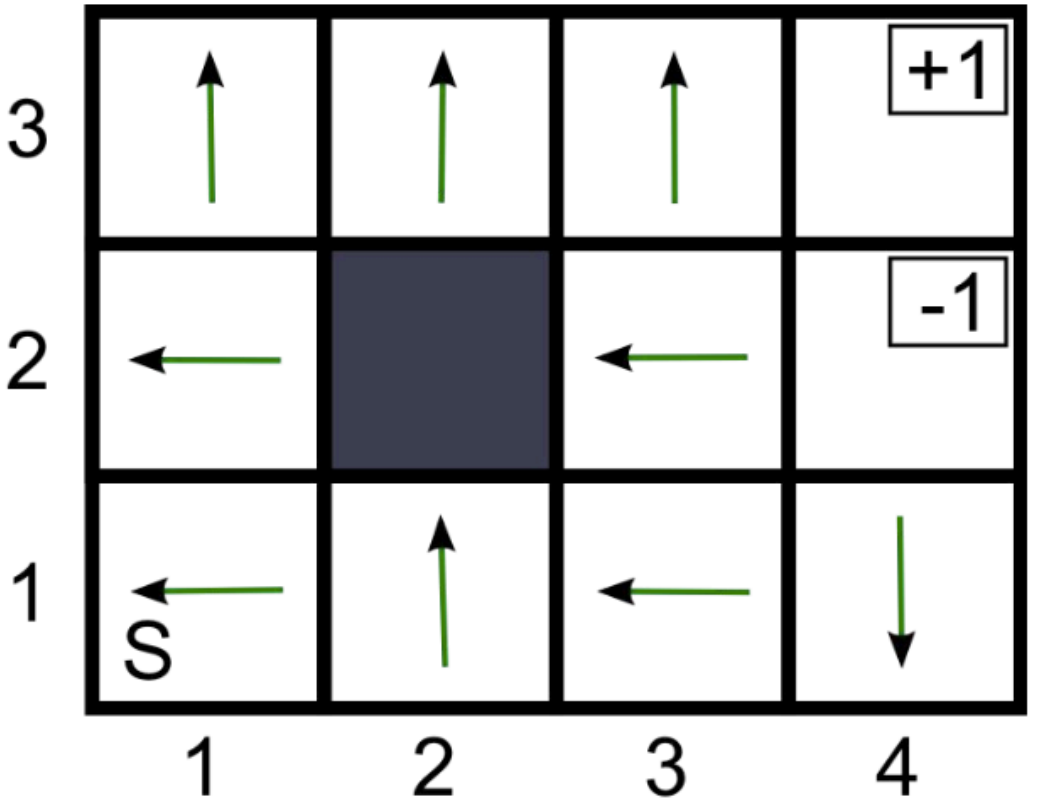


[Marivate and Rosman]

An example: Cleaning Robot



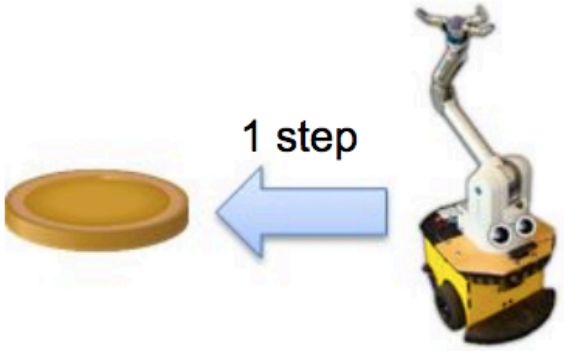
(stochastic)



[Marivate and Rosman]

Short-term vs Long-term Reward

- Cannot just rely on the instantaneous reward function:



- Notion of value to codify the goodness of a state, considering a policy running into the future:
  - Represented as a **value function**

[Marivate and Rosman]

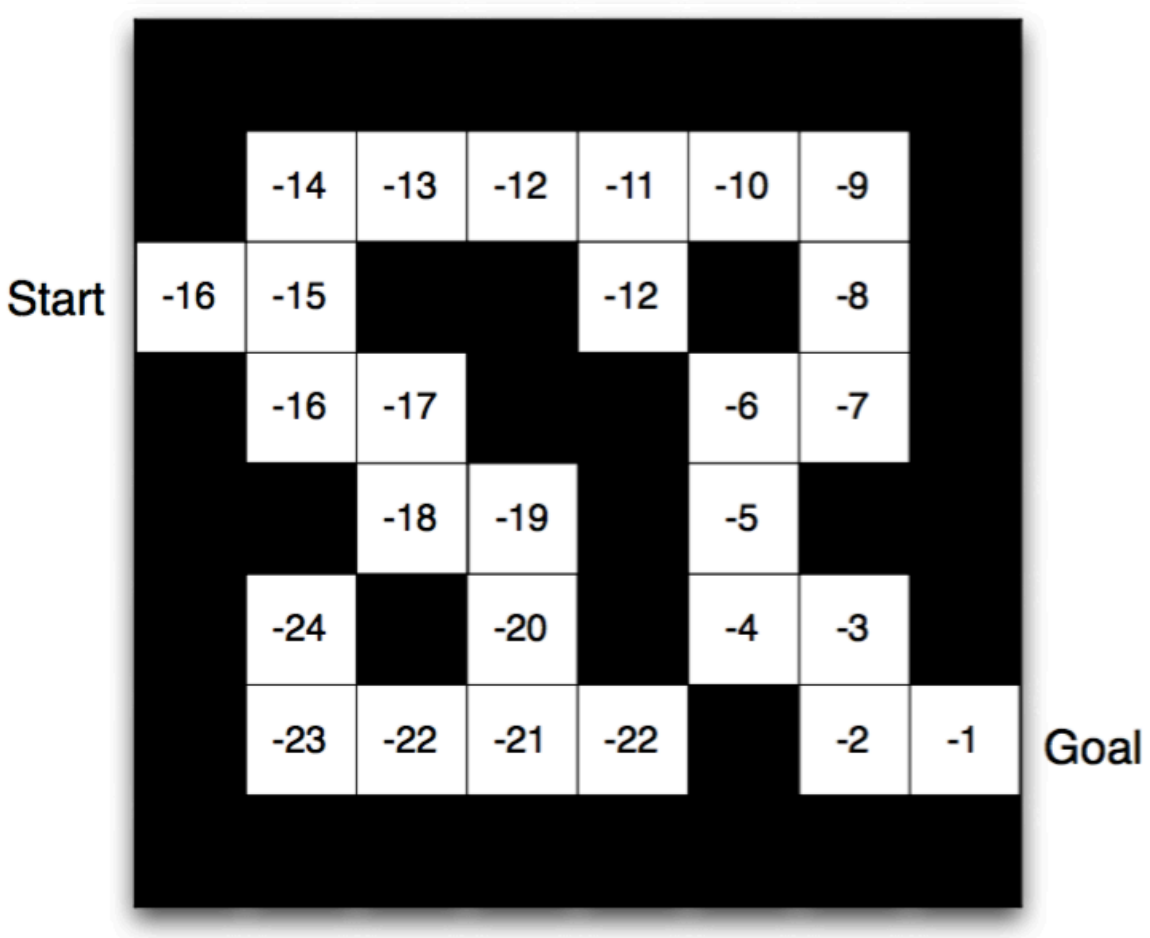
Value function

- Value function is a prediction of future reward.
- Used to evaluate the goodness/badness of states.
- Definition:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

"How good is state **s** when following policy $\boldsymbol{\pi}$?"

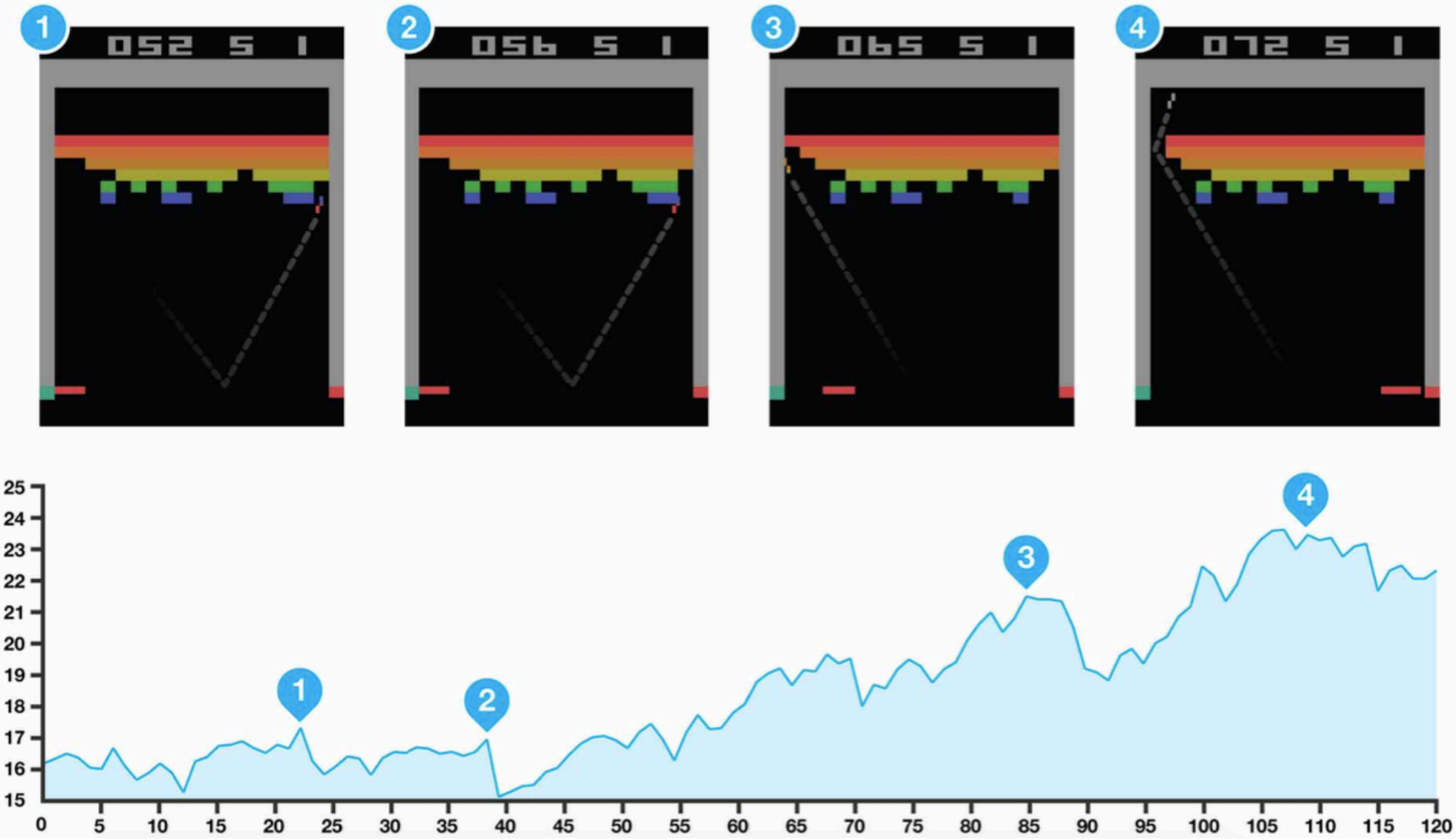
Example Value Function: Maze



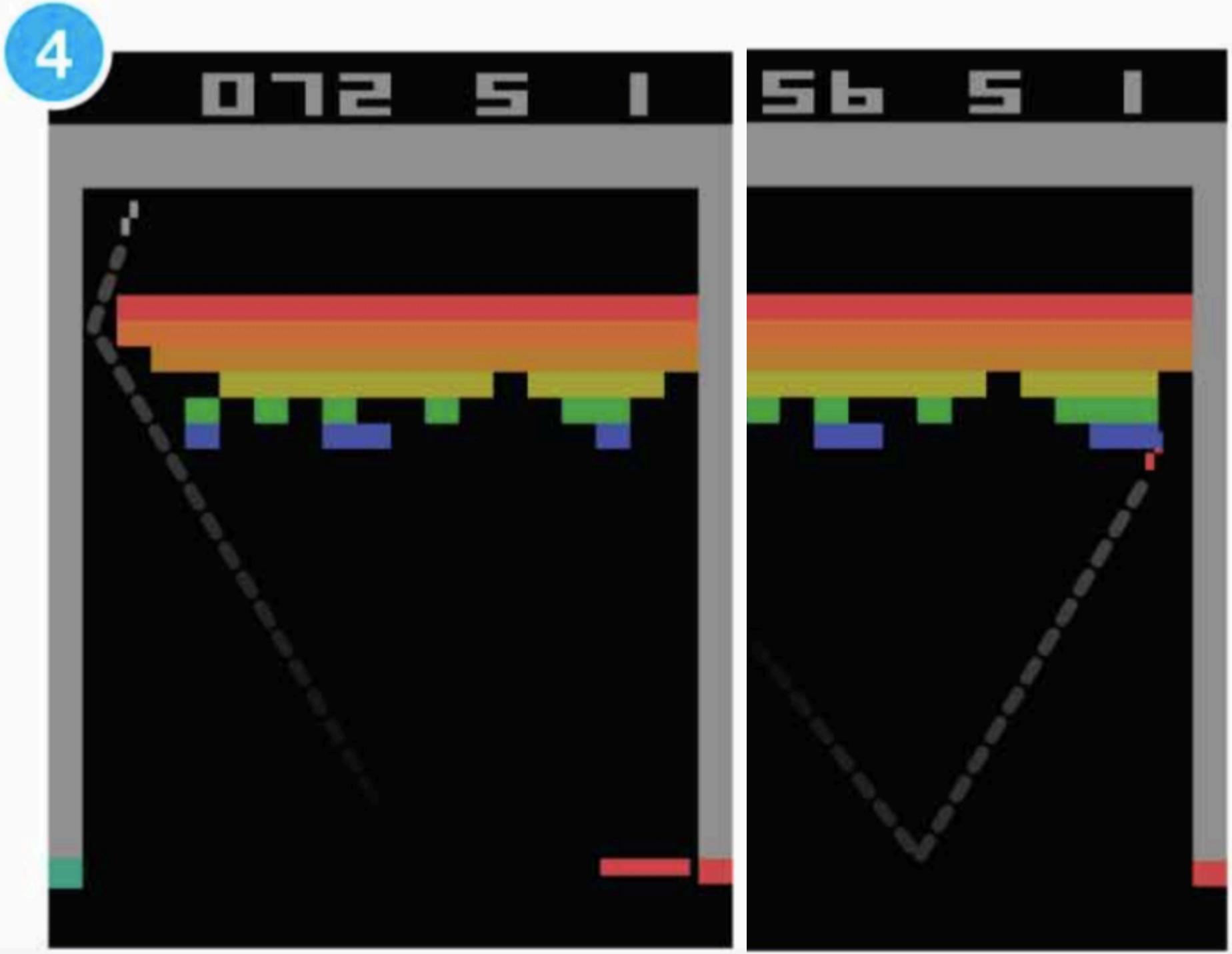
Numbers represent value $V^{\pi}(s)$ of each state s

[Silver]

Example Value Function: Atari



Human-level control through deep reinforcement learning, Mnih et al., Nature 2015


4
072 5 1
56 5 1

Model

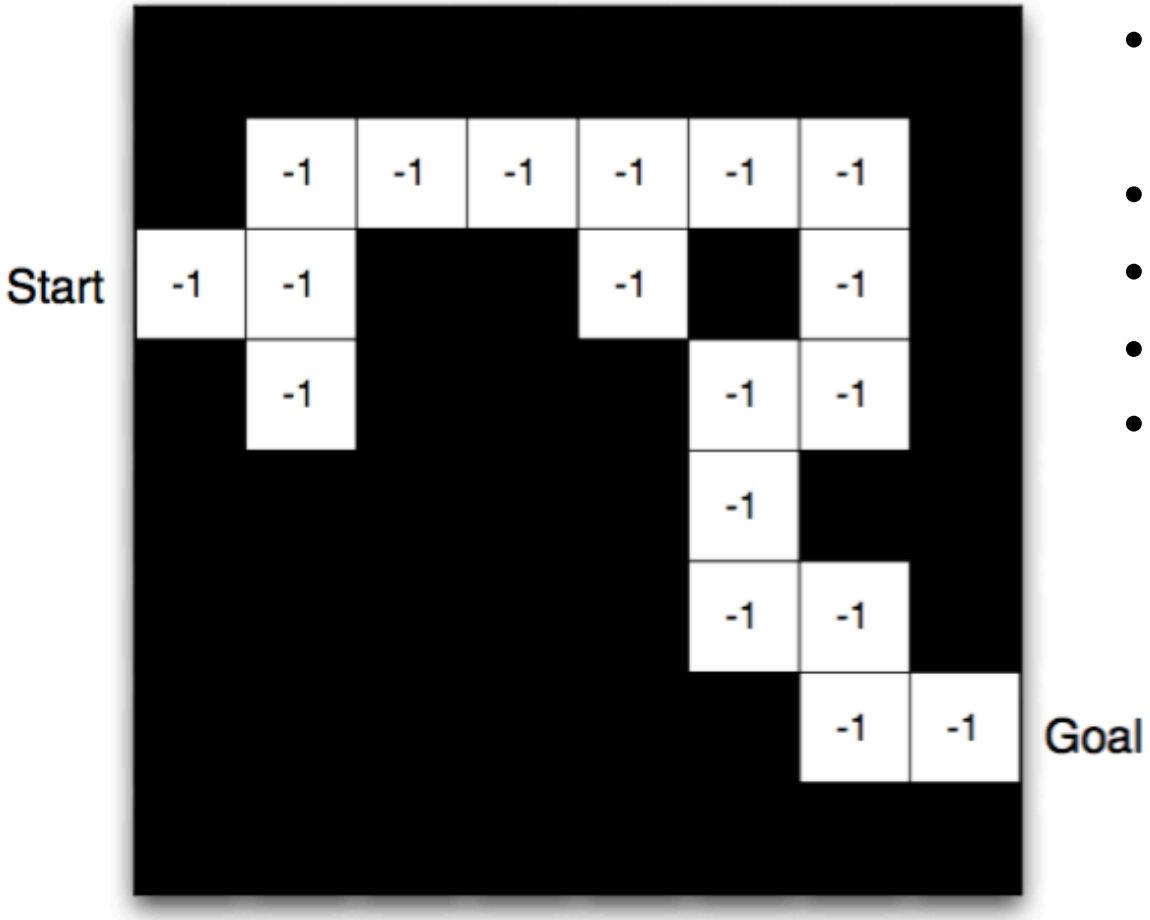
- A model predicts what the environment will do next.
- The model may be imperfect.
- Remember from the previous definition of a MDP:
  - T predicts the next state

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

  - R predicts the next (immediate) reward

$$R(s, a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$$

Maze Example: Model



- Agent may have an internal model of the environment
- Dynamics T: how actions change the state
- Rewards R: how much reward from each state
- Grid layout represents transition model
- Numbers represent immediate reward R(s,a) from each state s (same for all a)

[Silver]

Value Function Recursion: Bellman Equation

- Value function V(s): expected return starting at s and following π.
- Suggests dependence on V(s') from next state s'.

- Bellman equation:

For all states

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s')$$

Value of s

Immediate reward

Probability of reaching s' from s under π

Value of s'

Value Function Optimality

- For an optimal policy π^* with optimal value function V*:

- Bellman Optimality Equation:

$$V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s')]$$

Take the best possible action

Action-Value Function

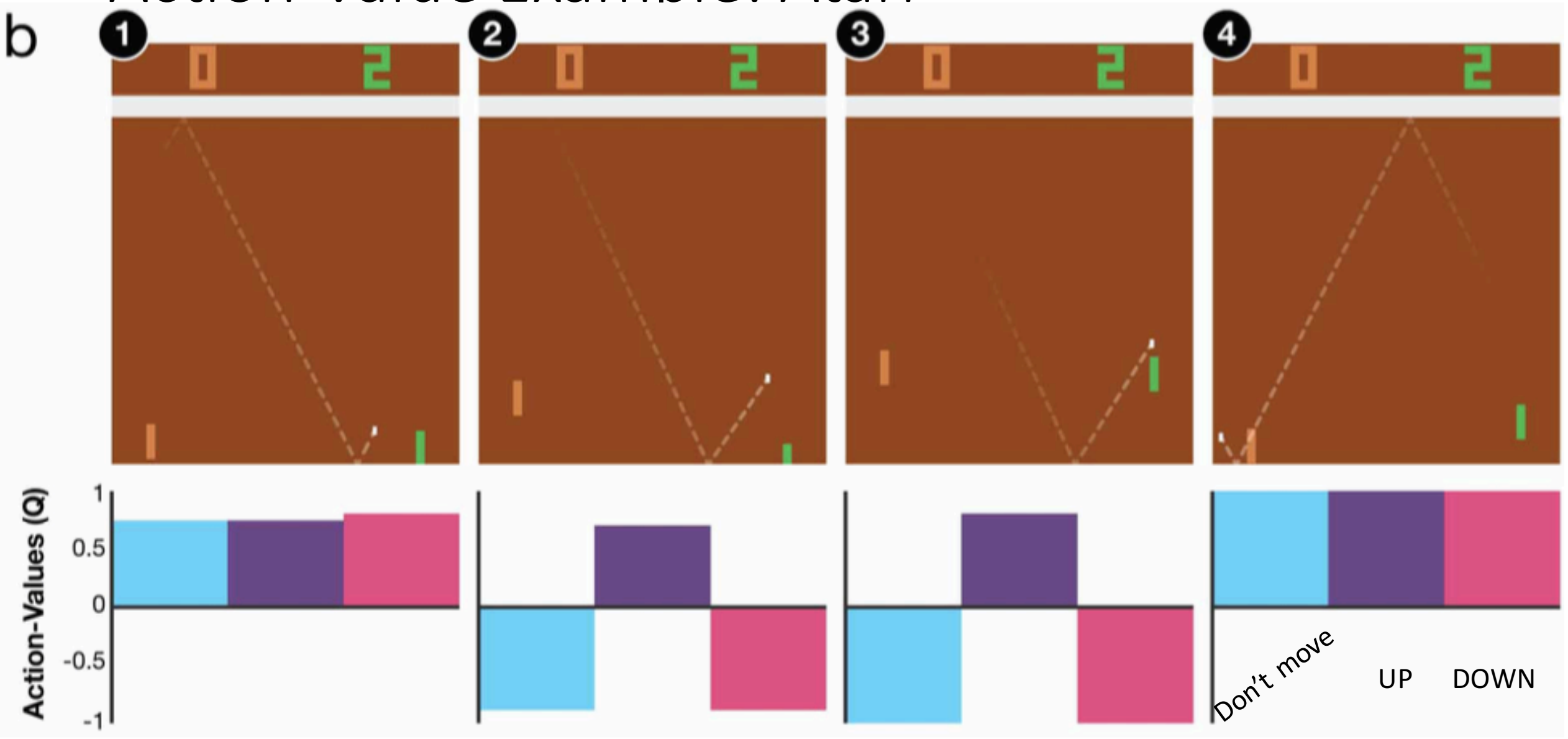
- Definition:

$$Q^{\pi}(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V^{\pi}(s'))$$

- The expected return starting at s and taking action a, and then following policy $\boldsymbol{\pi}$.

"How good is **a** in **s** under policy $\boldsymbol{\pi}$?

Action-Value Example: Atari



Human-level control through deep reinforcement learning, Mnih et al., Nature 2015

Optimal policies and value functions

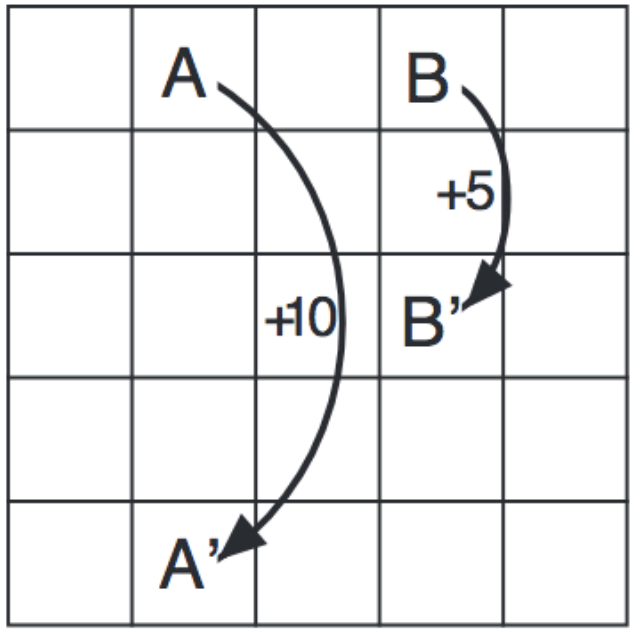
$$\pi^*(a|s) = \begin{cases} 1 & a = \arg\max Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

- Finding Q* (or V*) is equivalent to finding the optimal policy.
- Every MDP has an optimal policy π*.

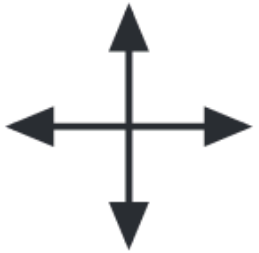
We want to move in direction of greatest value!

Example Value Function: Random Policy

- Reward: -1 at every move except for special states A and B (+10, +5)



(a)



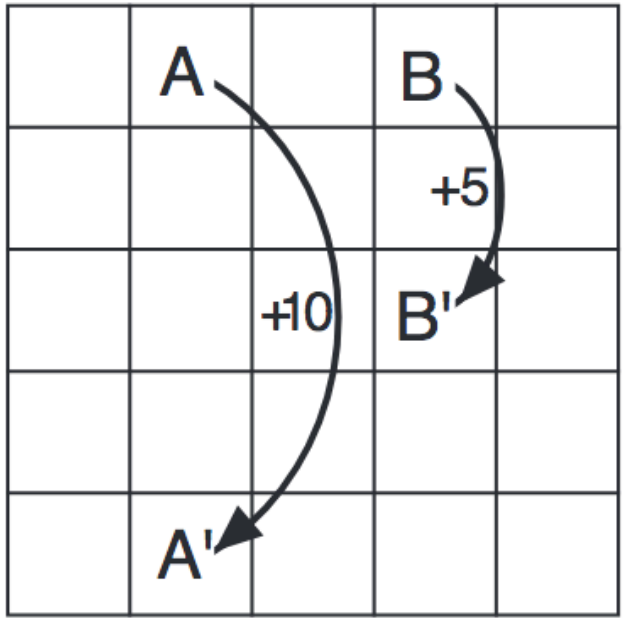
Actions

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

(b)

[Sutton and Barto]

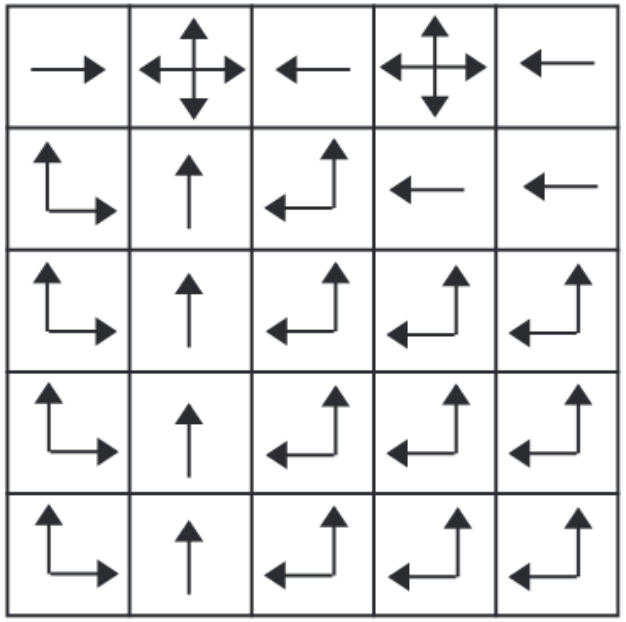
Example Value Function: Optimal Policy



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

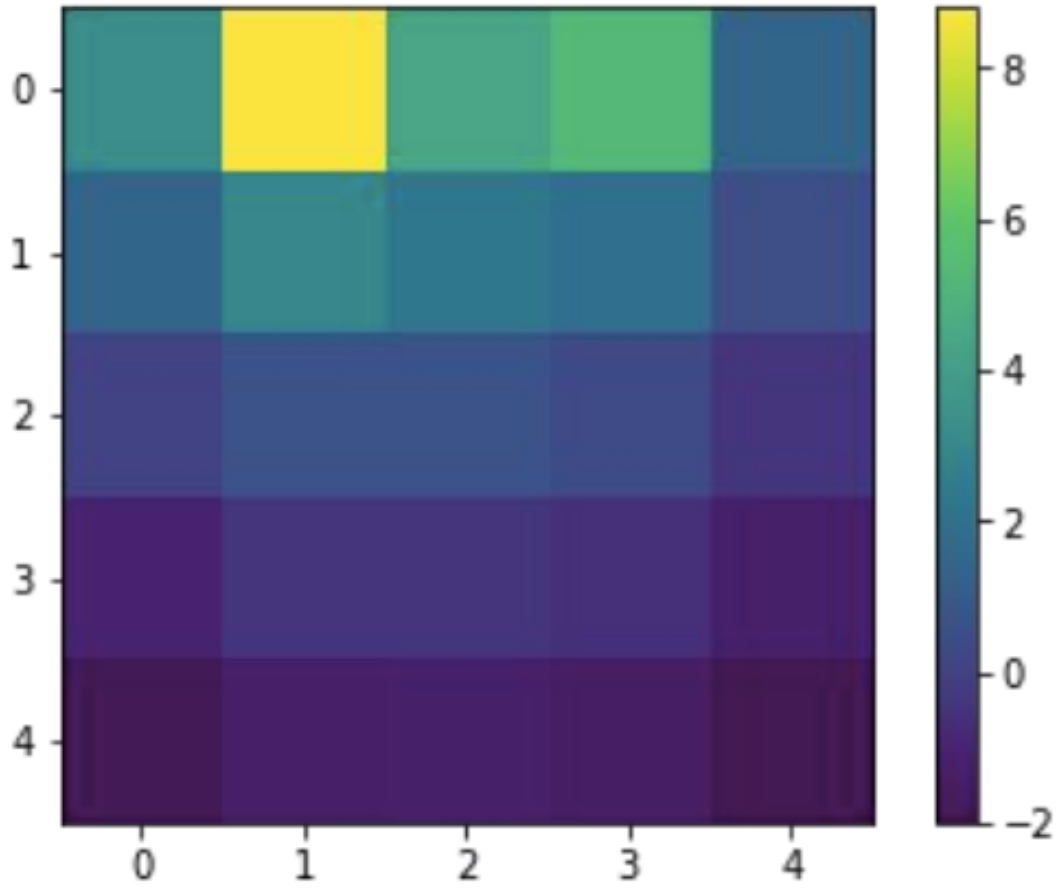
b) v_*



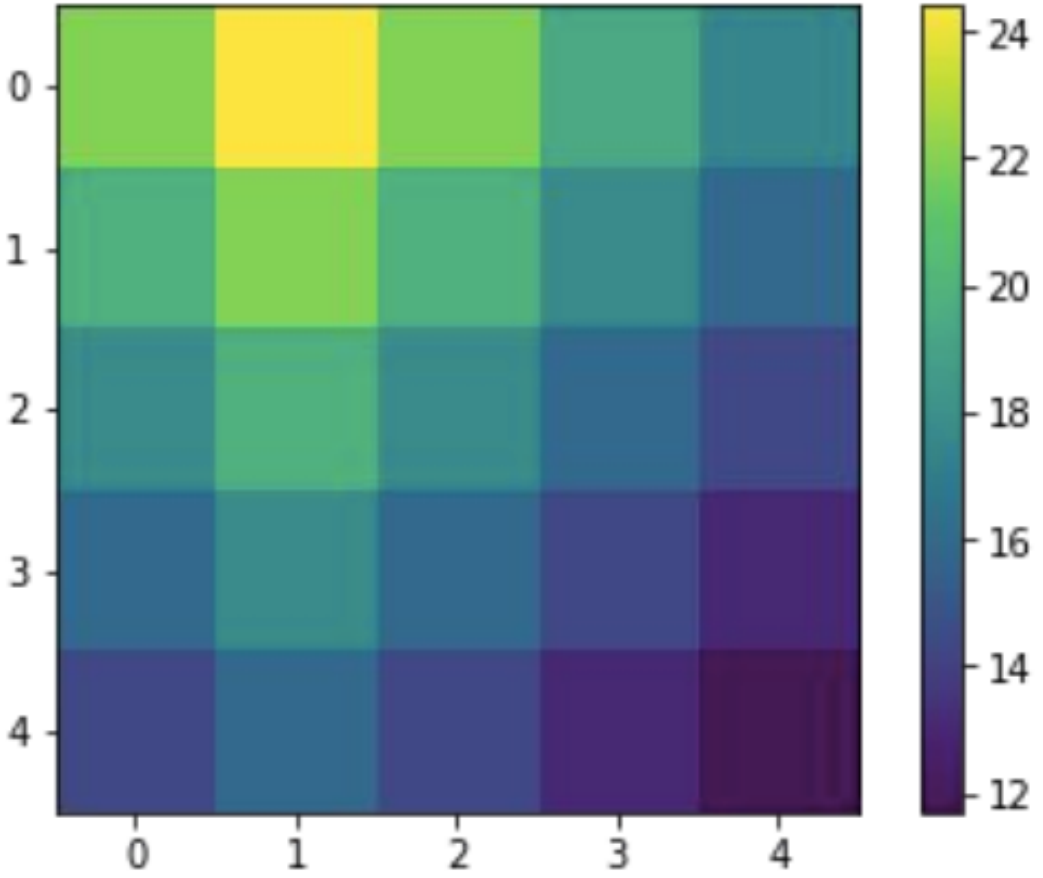
c) π_*

[Sutton and Barto]

Example Value Function: Random Policy vs Optimal



Value function: Random policy



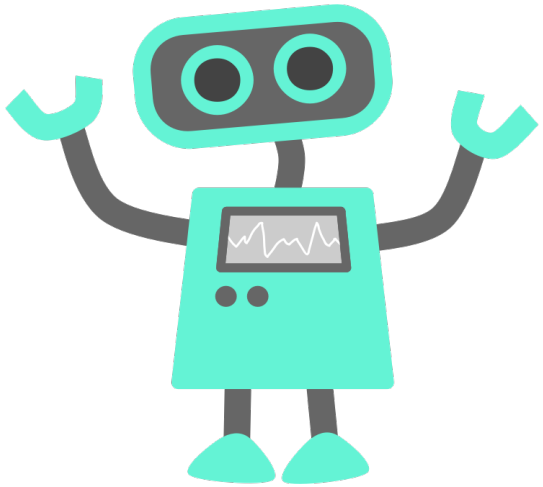
Value function: Optimal policy

Recall: Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function.
  - Value function: how good is each state and/or action.
  - Model: agent's representation of the environment.

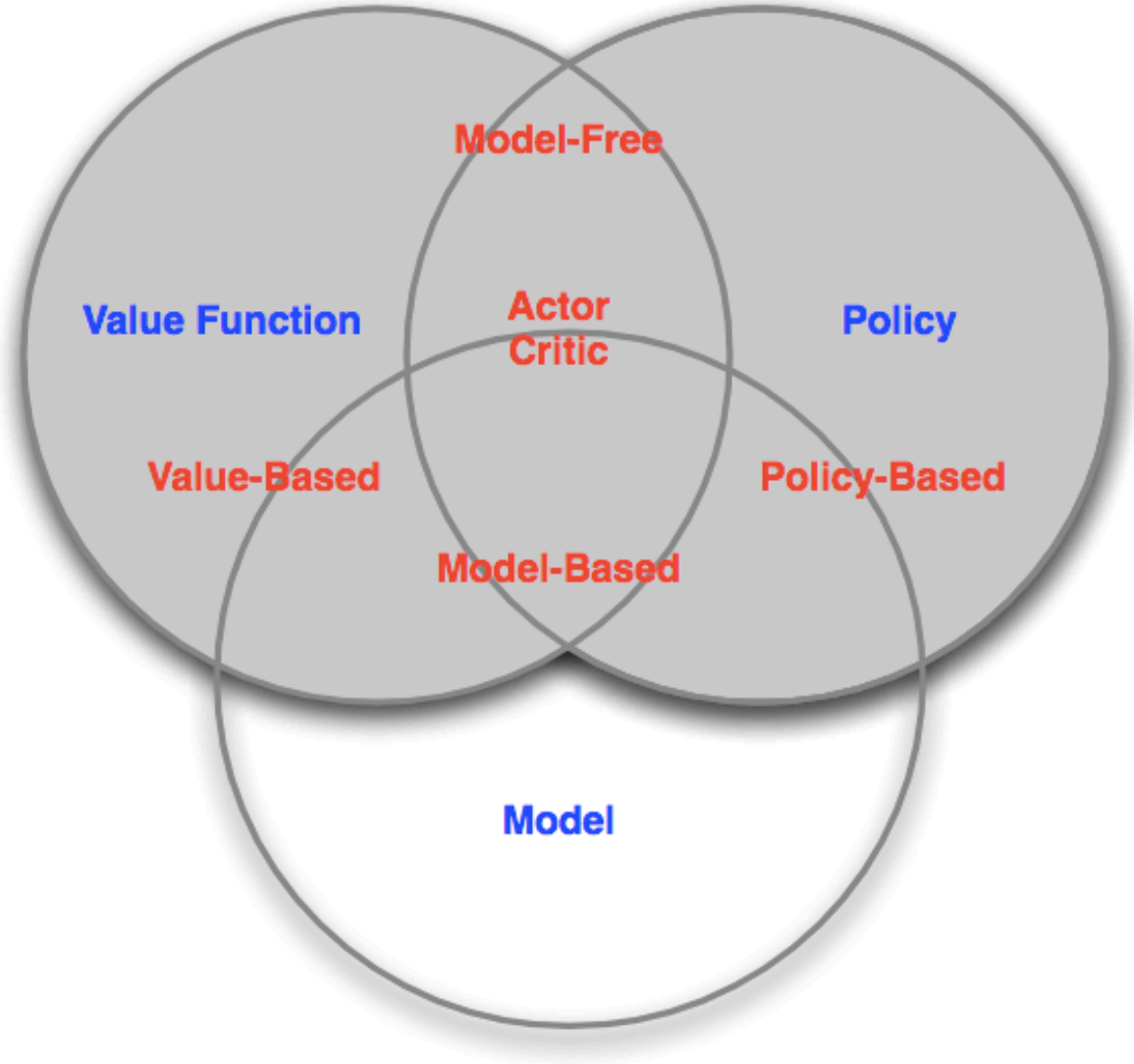
Categorising RL agents

- Value-Based
  - No Policy (Implicit)
  - Value Function
- Policy-Based
  - Policy (Explicit)
  - No Value Function
- Actor-Critic
  - Policy (Explicit)
  - Value Function



Categorising RL agents

- Model-Based
  - Policy and/or Value Function
  - Model
- Model Free
  - Policy and/or Value Function
  - No Model



[Silver]

Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model was correct

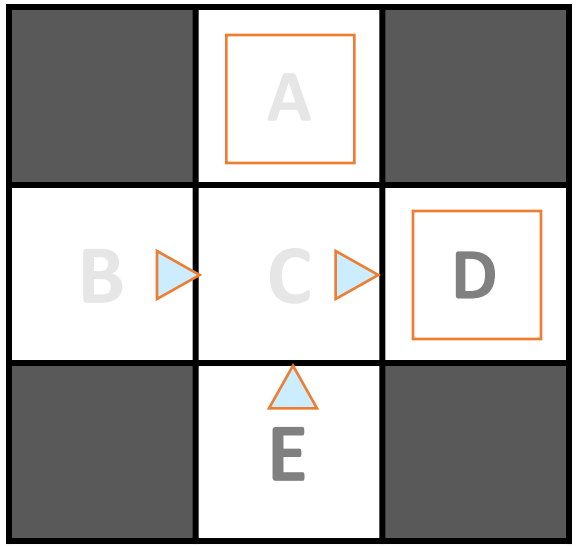
- Step 1: Learn empirical MDP model
  - Count outcomes s’ for each s, a
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s’)

- Step 2: Solve the learned MDP
  - For example using value iteration (we will see this later).

Example: Model-Based Learning

Input Policy

π



Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) =
T(C, east, D) =
T(C, east, A) =
…

$\hat{R}(s, a, s')$

R(B, east, C) =
R(C, east, D) =
R(D, exit, x) =
…

[Abbeel and Klein]

Solving the MDP

- We can try to solve the Bellman equation:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right]$$

- Solve this as a large system of value function equations
  - Problem: non linear (max operator)
  - An approach: solve iteratively with dynamic programming

Dynamic Programming: Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality equation
  - $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v^*$
  - Using synchronous backups
    - At each iteration k + 1
    - For all states s $\in$ S′
    - $V_{k+1}(s) = \max_a [\sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V_k(s'))]$

- Convergence to optimal policy can be proven [Sutton and Barto].

Advantages of Model-Based RL

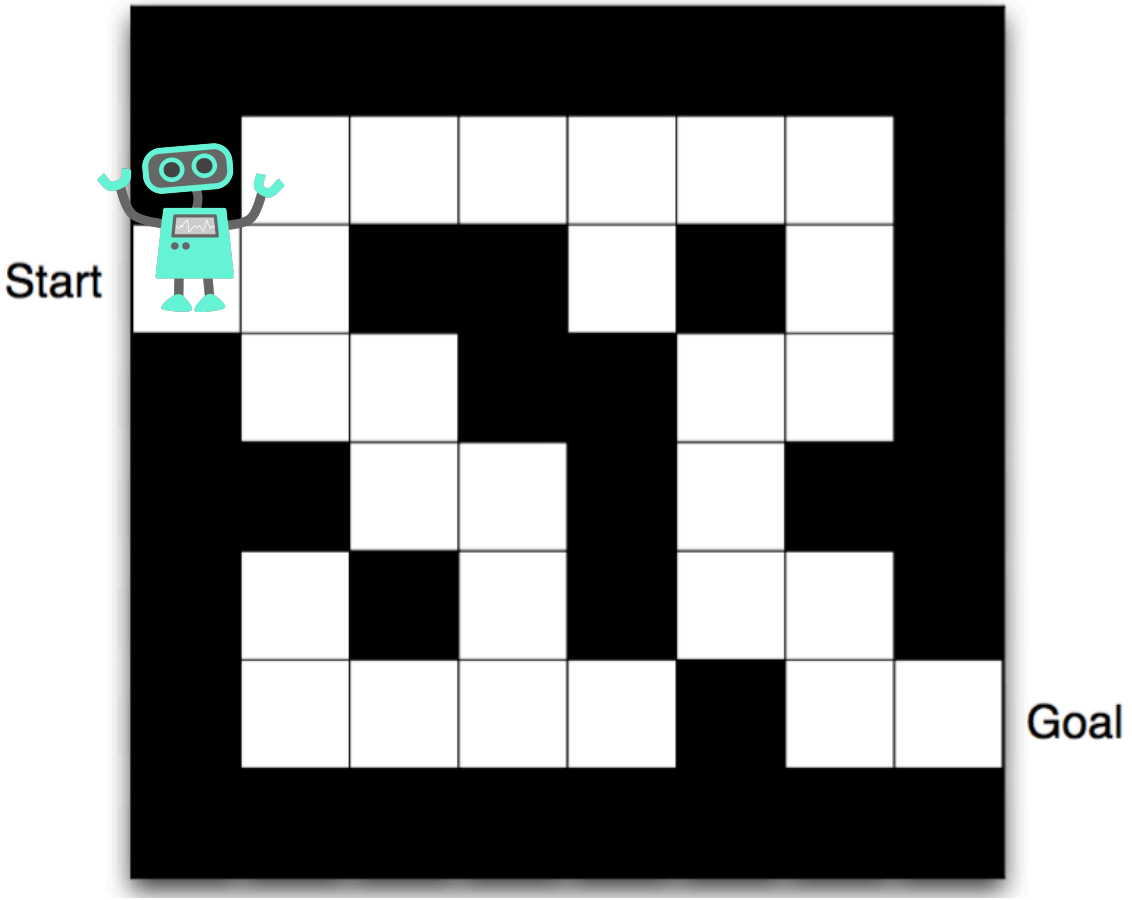
- Advantages:
  - Can efficiently learn model by supervised learning methods.
  - Can reason about model uncertainty.
- Disadvantages:
  - First learn a model, then construct a value function
    $\Rightarrow$ two sources of approximation error!

Model Free RL

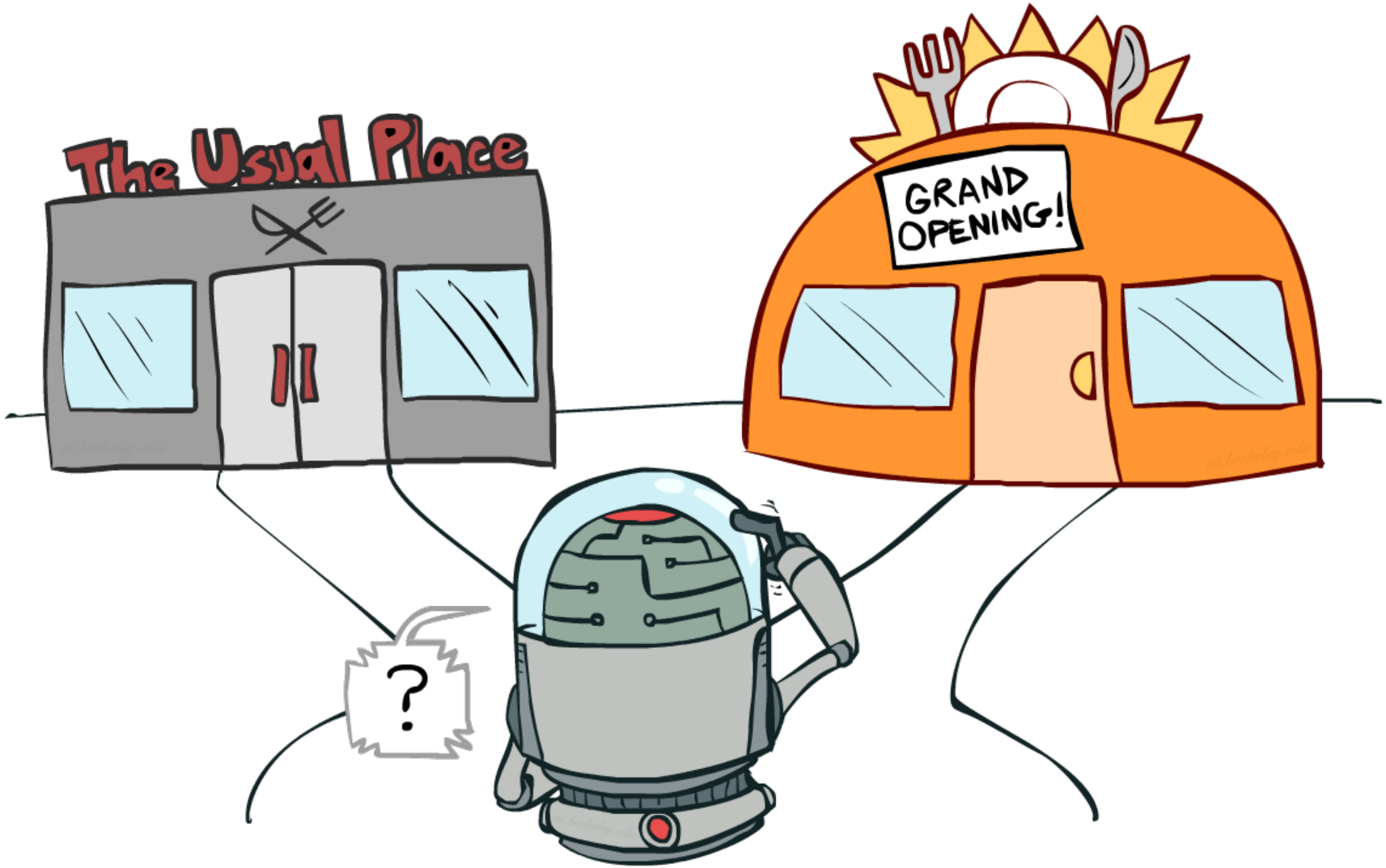
Model Free RL

- No Transition Model.
- No Reward Model.
- Access to the environment allowing the agent to interact with it or access to training data ***(s,a,r,s').***

- How does the agent choose actions?



Exploration vs. Exploitation



[Abbeel and Klein]

Exploration vs. Exploitation

- **Exploration** finds more information about the environment.
- **Exploitation** exploits known information to maximise reward.
- It is usually important to explore as well as exploit.
  - Tradeoff!

Exploration vs. Exploitation: Examples

- Restaurant Selection
  - **Exploitation:** Go to your favourite restaurant
  - **Exploration:** Try a new restaurant
- Online Banner Advertisements
  - **Exploitation:** Show the most successful advert
  - **Exploration:** Show a different advert
- Game Playing
  - **Exploitation:** Play the move you believe is best
  - **Exploration:** Play an experimental move

Action selection strategy

- Simplest: ϵ-greedy strategy
  - With probability 1-ϵ: Act on current policy (**Exploit**)
  - With probability ϵ: Act randomly (**Explore**)

- Problem with this strategy?
  - You do eventually explore the space, but keep thrashing around once learning is done.
  - One solution: Lower ϵ over time.
  - Other solution: Use more sophisticated approaches, e.g. exploration functions.

Value-Based Methods

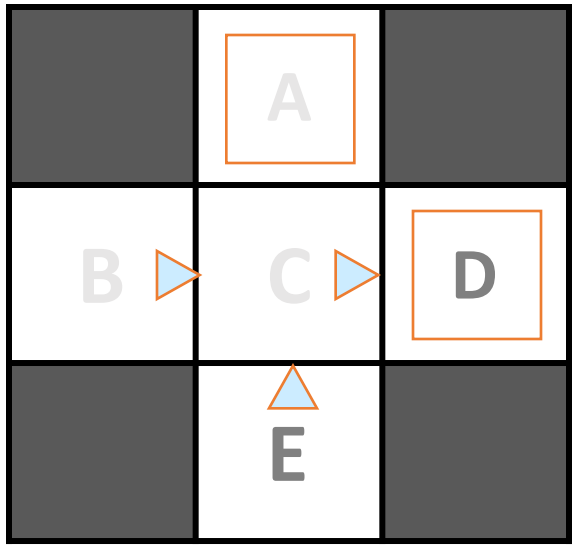
Value-Based RL

- No Transition Model.
- No Reward Model.
- Access to environment for experiment or access to training data ***(s,a,r,s').***

- **Goal:** Learn value of states, state-actions
  - Policy through learned values

Estimating Value Function: Direct Evaluation

Input Policy π



Assume: γ=1

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

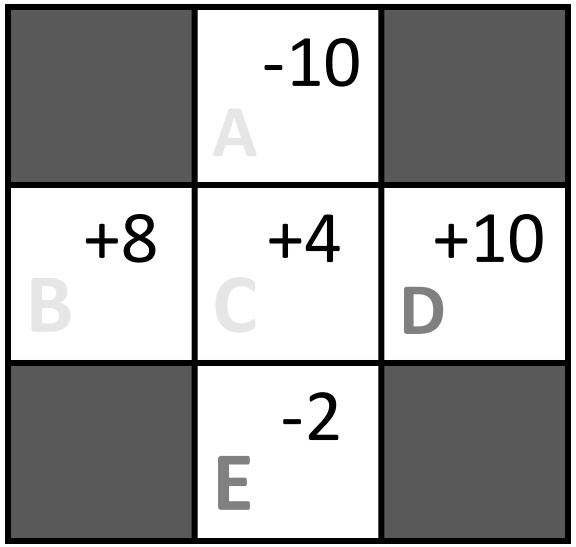
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



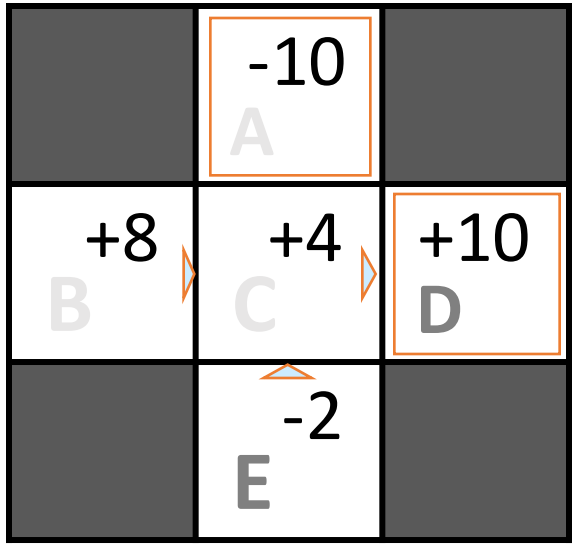
[Abbeel and Klein]

Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand.
  - It doesn't require any knowledge of T, R.
  - It eventually computes the correct average values, using just sample transitions.

- What's bad about it?
  - It wastes information about state connections.
  - Each state must be learned separately.
  - So, it takes a long time to learn.

Output Values



[Abbeel and Klein]

Sample-Based Policy Evaluation?

- Recall: We want to improve our estimate of V by computing these averages:

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')]$$

But we don't have T and R!

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\cdots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$

[Abbeel and Klein]

Temporal Difference Learning

- Idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
  - Move values toward value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

[Abbeel and Klein]

Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

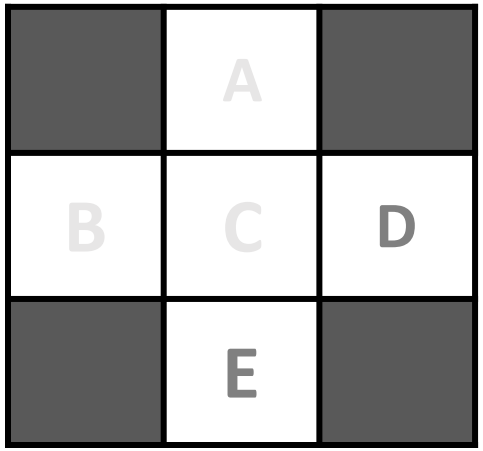
  - Forgets about the past (distant past values were wrong anyway).

- Decreasing learning rate (alpha) can give converging averages.

[Abbeel and Klein]

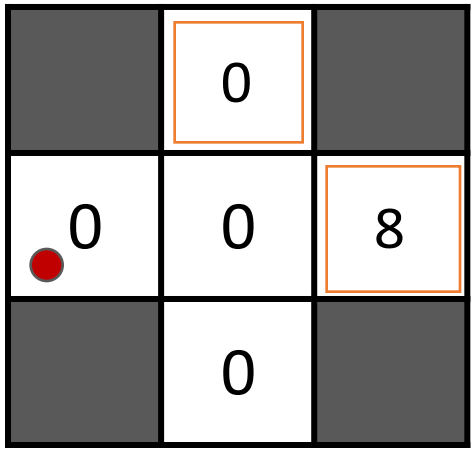
Example: Temporal Difference Learning

States

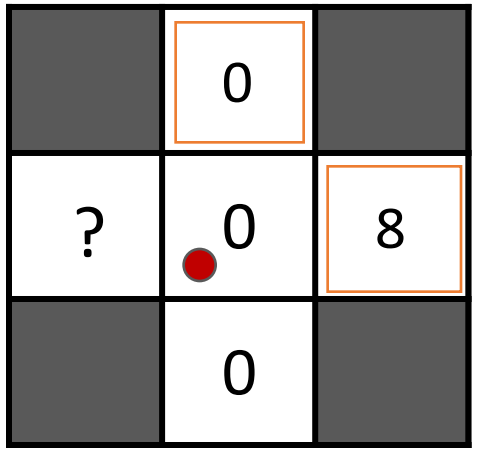


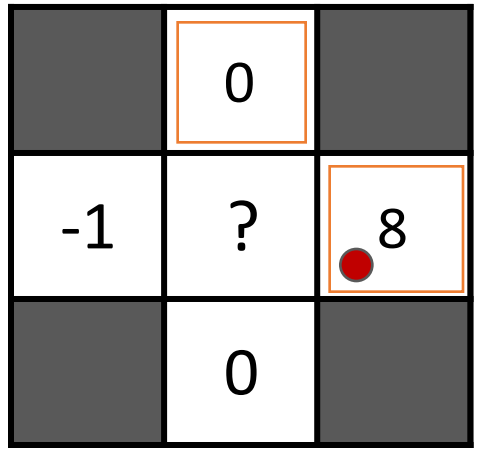
Observed Transitions

B, east, C, -2



C, east, D, -2





Assume: γ = 1, α = 1/2

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

[Abbeel and Klein]

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages.
- However, if we want to turn values into a (new) policy, we're sunk:

$$\pi(s) = \arg\max_a Q(s,a)$$

$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right]$$

- Idea: learn Q-values, not values
- Makes action selection model-free too!

[Abbeel and Klein]

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Slide credit: [Abbeel and Klein]

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

[Abbeel and Klein]

On Policy vs. Off Policy

- From the Q-Learning equation:

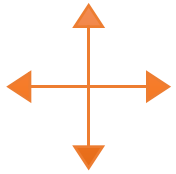
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- We are taking the best action. This action can come from:
  - Exploration.
  - External policy.
  - Demonstrations (expert).
- Q-Learning is an “off policy” approach.
- Instead, we could act following the current policy. This is called “on policy” learning, e.g., SARSA.

Cleaning Robot Example: Policy

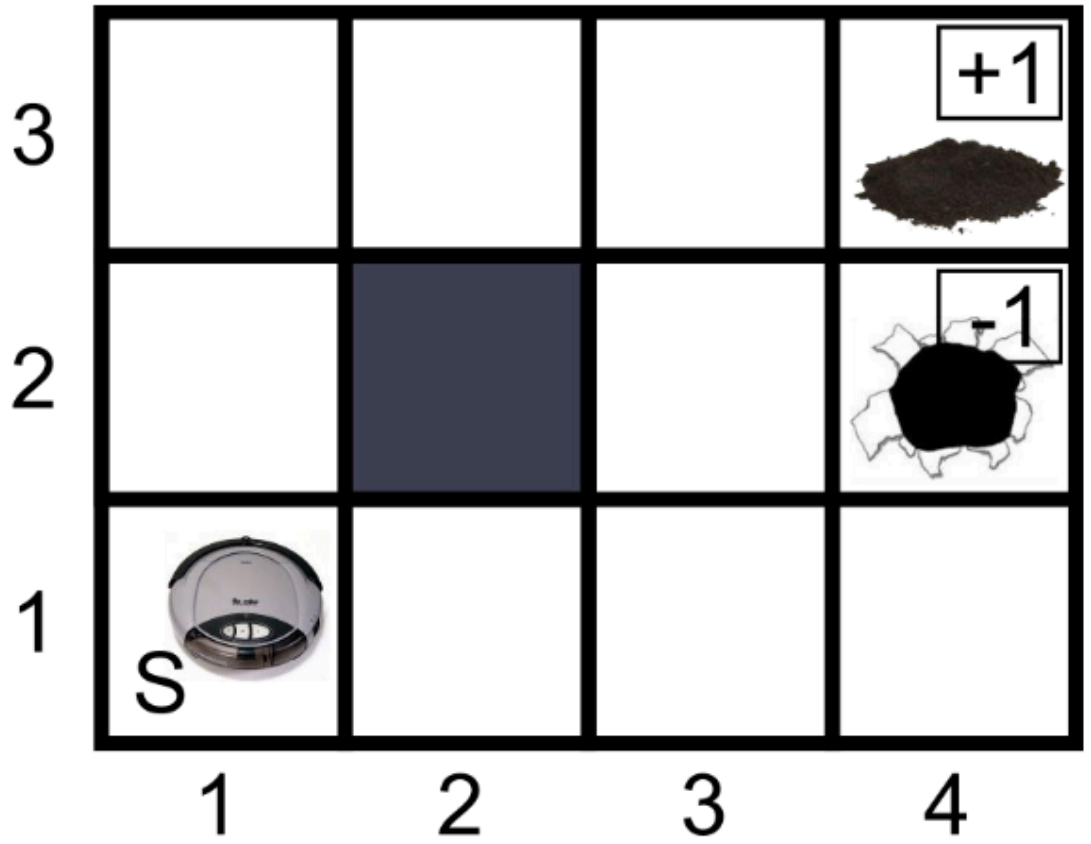
States: Position on grid e.g.
S (start) is (1,1), goal (4,3)

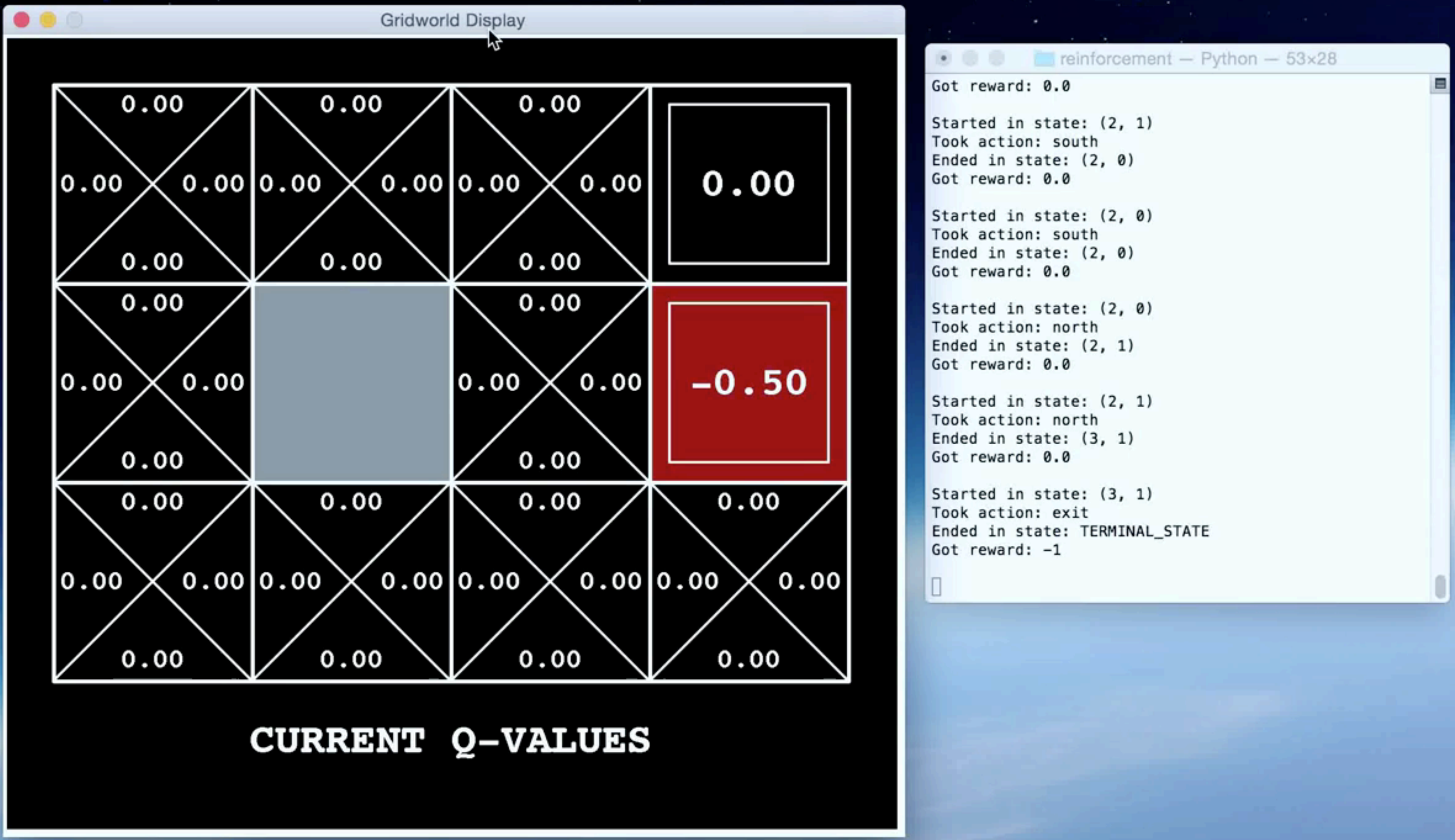
Actions:



Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move





[Abbeel and Klein]

Policy-Based RL

Policy-Based Reinforcement Learning

- In the previous studied methods a policy was generated from the value function, e.g. by using ϵ-greedy.

- Policy-Based methods directly parametrise the policy:

$$\pi_\theta(s, a) = P(a|s, \theta)$$

- e.g. using a neural network!

Policy Gradients

- Goal: given policy $\pi_\theta(s,a)$ with parameters θ, find best θ.
- Policy based reinforcement learning is an optimisation problem.
- An example: Policy Gradient Approach
  - Define a cost function $J(\theta)$
  - e.g. start value, average value…
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ.

$$\Delta\theta = \alpha\nabla_\theta J(\theta)$$

Policy Gradients II

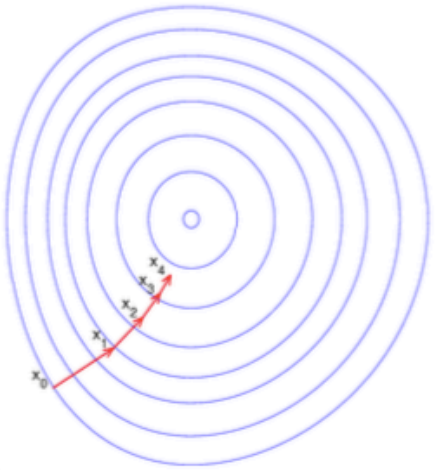
- Policy gradient algorithms search for a local maximum in J(θ) by ascending the gradient of the policy, w.r.t. parameters θ.

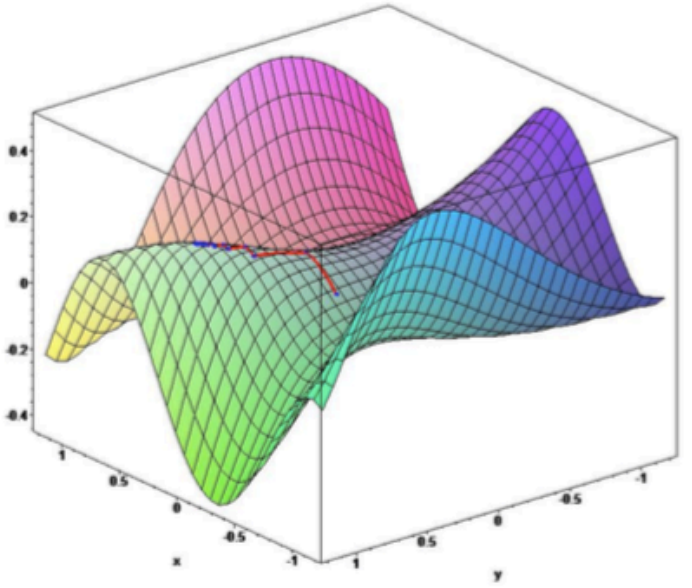
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- Where:

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- And α is a learning rate





[Silver]

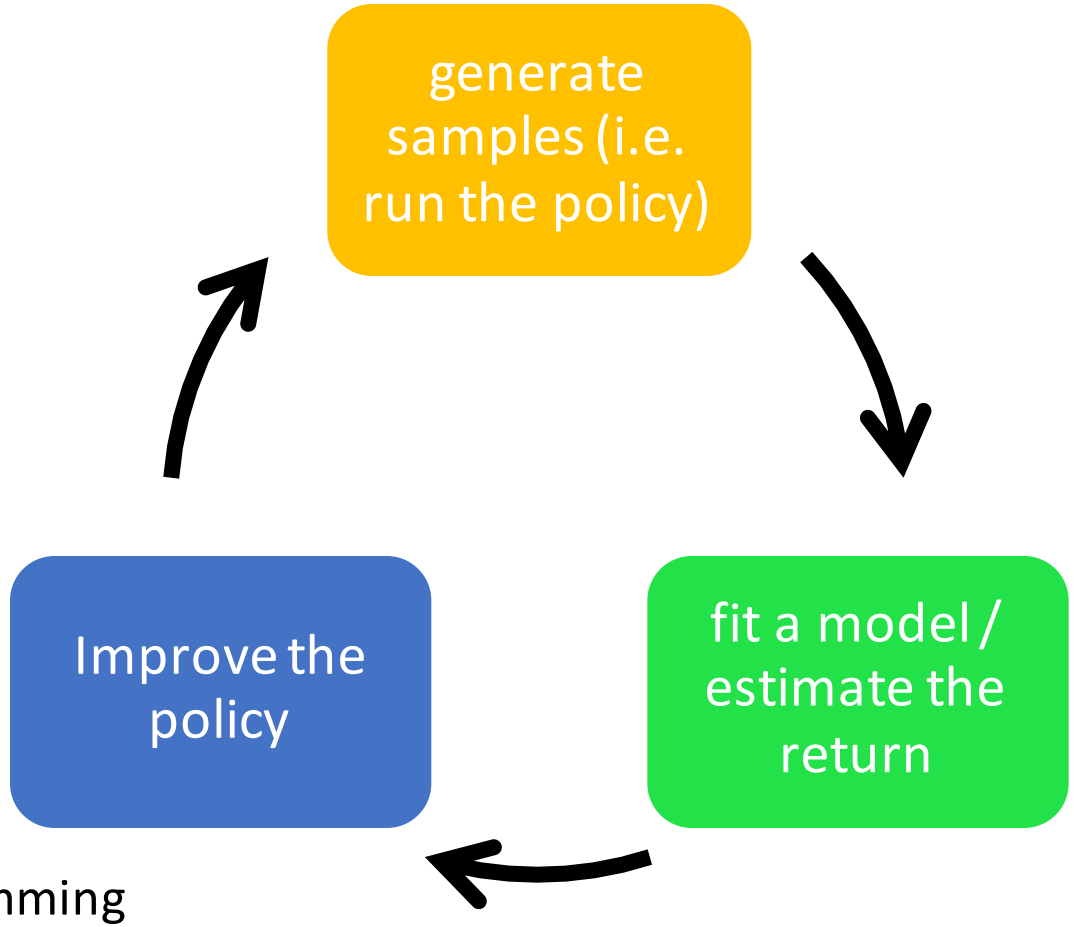
Advantages Policy-Based RL

- Advantages:
  - Better convergence properties.
  - Effective in high-dimensional or continuous action spaces.
  - Recent success in robotics and Go.
- Disadvantages:
  - Typically converge to a local rather than global optimum.
  - Evaluating a policy is typically inefficient and high variance.

A Brief Note on Actor-Critic Methods

- Problem: Policy gradient has high variance.
- One solution: We use a critic to estimate the action-value function.
- Actor-critic algorithms maintain two sets of parameters
  - **Critic:** Policy evaluation (e.g. direct evaluation, temporal difference).
  - **Actor:** Updates policy parameters θ, in direction suggested by critic.
- Actor-critic algorithms follow an approximate policy gradient.

Recap of today's methods:



Model-Based: Dynamic programming
Value-Based: $\pi(s) = \arg\max_a Q(s,a)$
Policy-Based and Actor-Critic:
$\Delta\theta = \alpha\nabla_\theta J(\theta)$

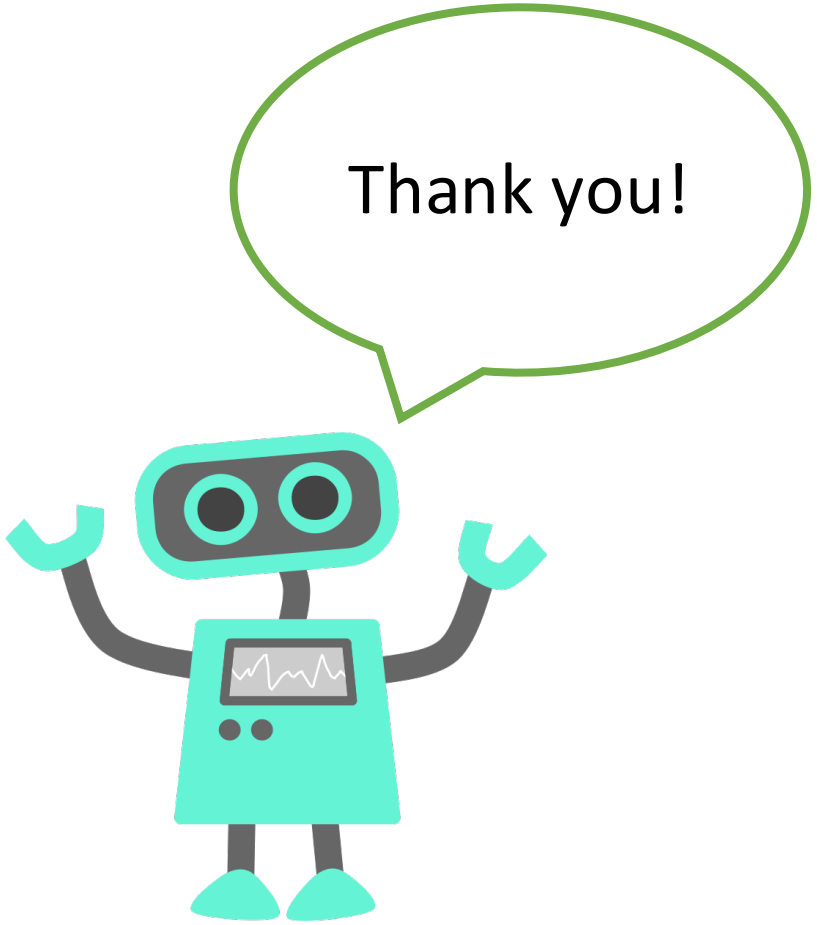
Model-Based: Fit a model
Value-Based: Fit V(s) or Q(a,s)
Policy-Based: Evaluate returns (J)
Actor-Critic: Fit V(s) or Q(a,s)

A Brief Note on Scaling up RL

- Reinforcement learning can be used to solve large problems, e.g.
  - Computer Go: 10^{170} states.
  - Navigation: continuous state space.
- Basic Q-Learning keeps a table of all q-values!!
- Instead, we want to generalise:
  - Learn about some small number of training states from experience.
  - Generalize that experience to new, similar situations.
- Solution: use function approximation to estimate Q!
  - Side note: state-of-the-art RL approaches use deep neural networks to approximate different components of the pipeline (Deep RL).

Bibliography and Resources

- Textbook:
  - [Sutton and Barto] Reinforcement Learning: An Introduction. Richard S. Sutton and Andrew G. Barto, 2nd Edition, MIT Press, 2018. Available free online!
- Reinforcement Learning courses :
  - [Abbeel and Klein] UC Berkeley Course "Introduction to Artificial Intelligence". Link
  - [Marivate and Rosman] Reinforcement Learning: An Introduction. Vukosi Marivate and Benjamin Rosman, Deep Learning Indaba, 2017. Link
  - [Silver] UCL Course on RL. David Silver, 2015. Link
- Advanced and state-of-the-art RL algorithms:
  - [Levine] UC Berkeley Course on Deep RL. Sergey Levine, 2018. Link


Thank you!