Machine Learning
CMPSCI 589

Introduction to Reinforcement Learning

UMassAmherst
College of Information & Computer Sciences

Reinforcement Learning

- Learn *how to act*
  - without a supervisor / training set
  - no previous knowledge about your environment
  - based only on rewards (and punishments)

- Positive rewards
  makes behavior more likely
- Negative rewards
  makes behavior less likely

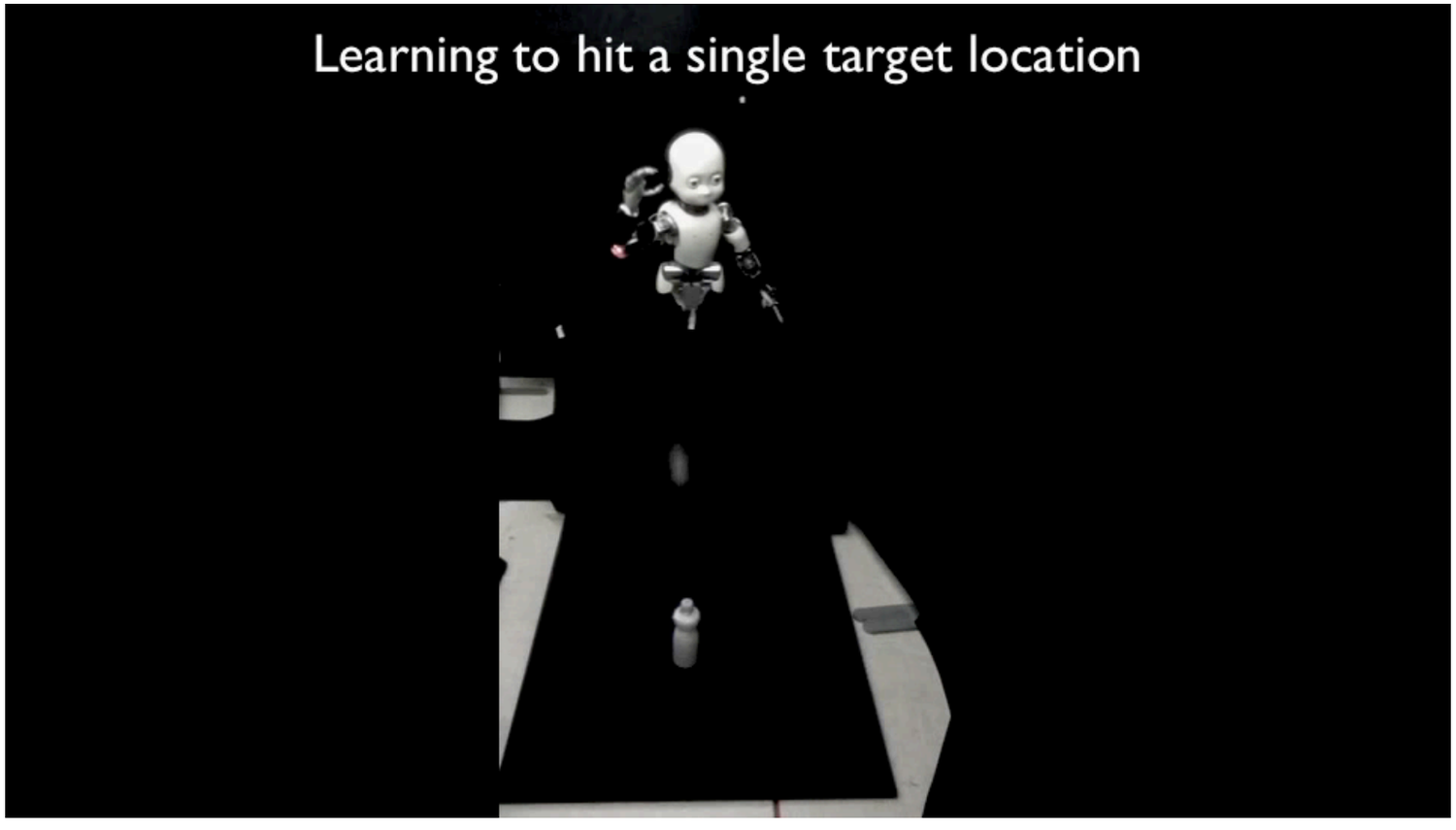
Reinforcement Learning

- Learn *how to act*
  - without a supervisor / training set
  - no previous knowledge about your environment
  - based only on rewards (and punishments)

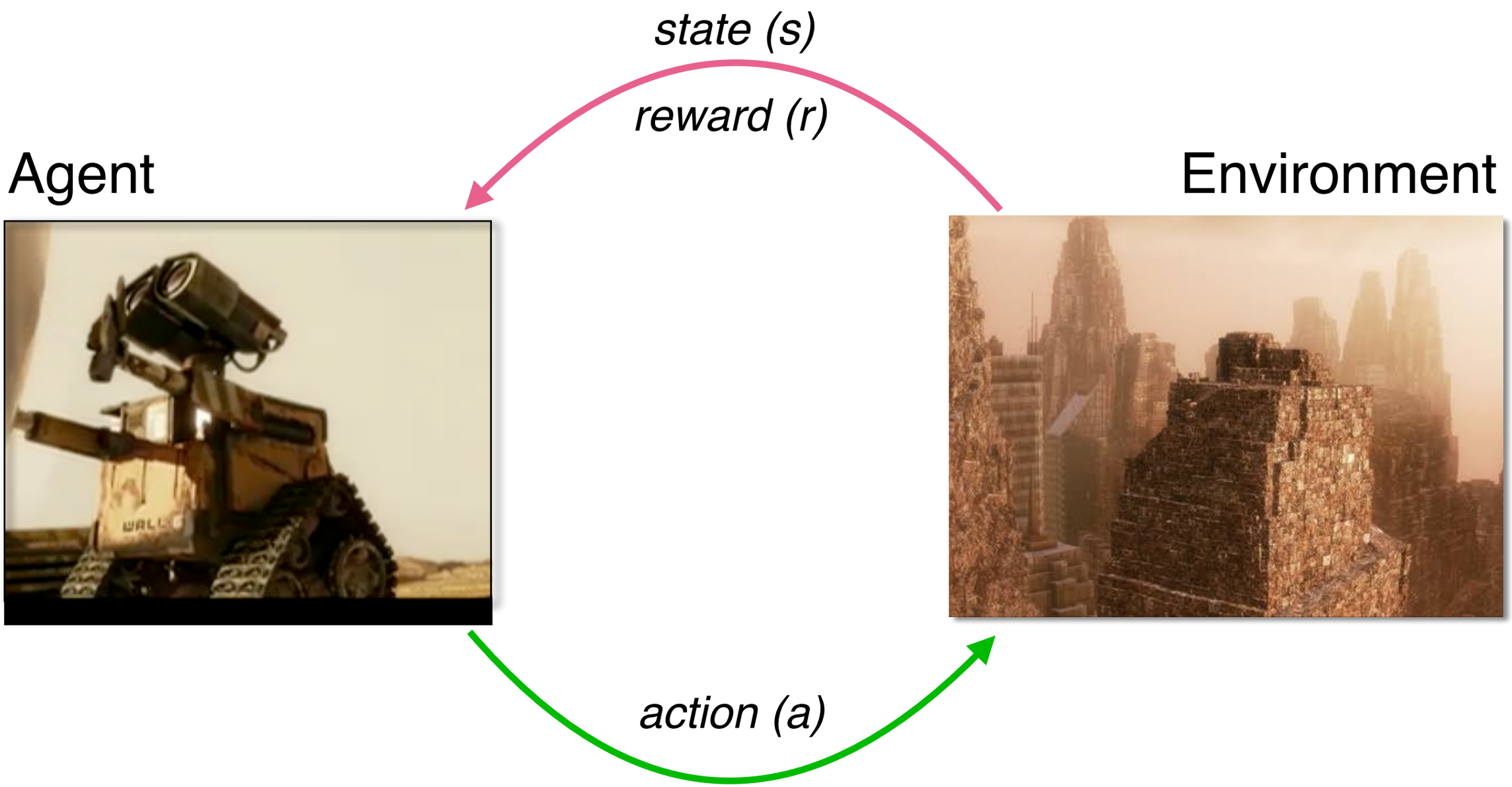
https://www.youtube.com/watch?v=mDntbGRPeEU

Reinforcement Learning

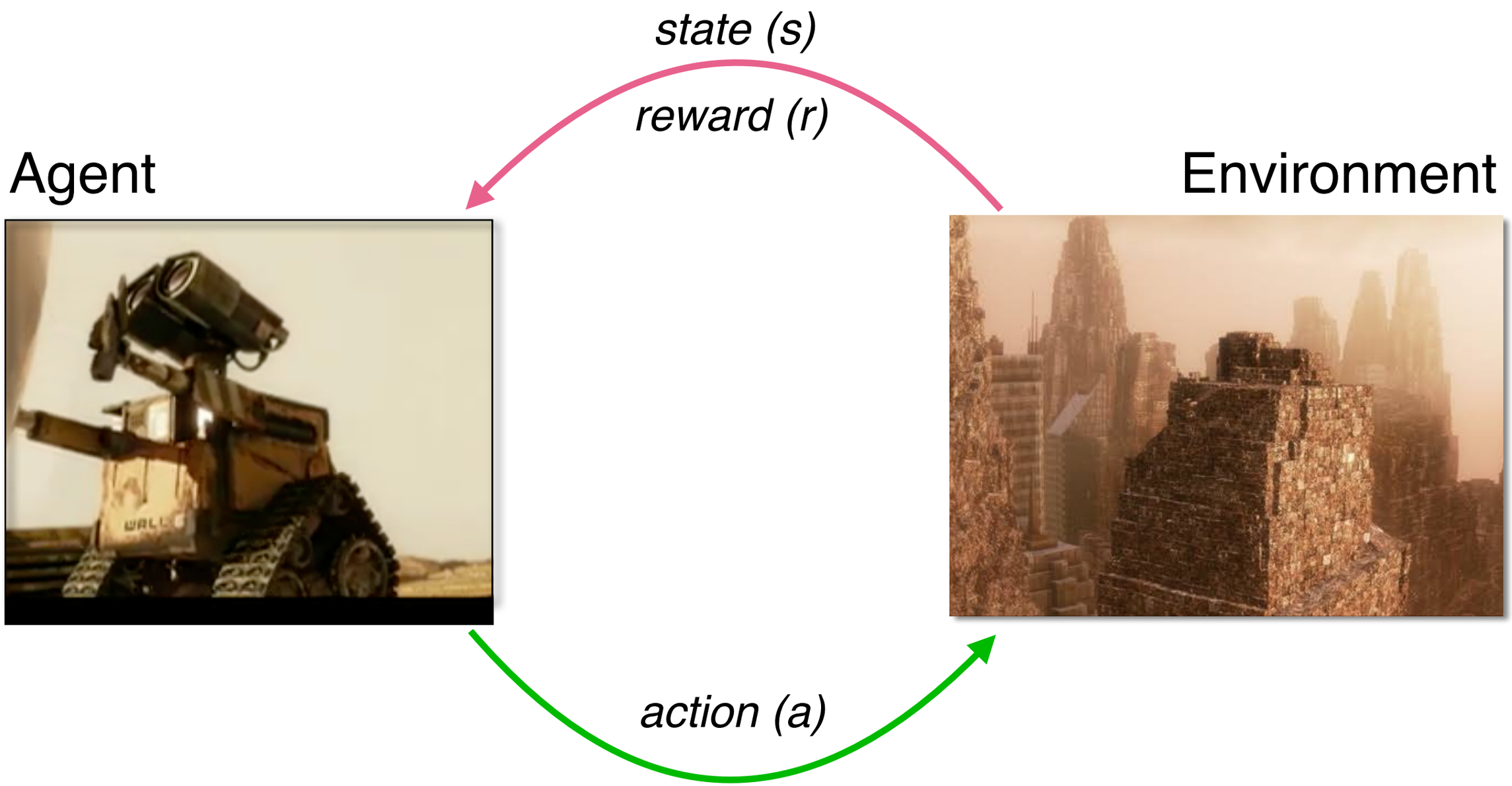
- Learn *how to act*
  - without a supervisor / training set
  - no previous knowledge about your environment
  - based only on rewards (and punishments)



Reinforcement Learning



Reinforcement Learning



Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Reinforcement Learning

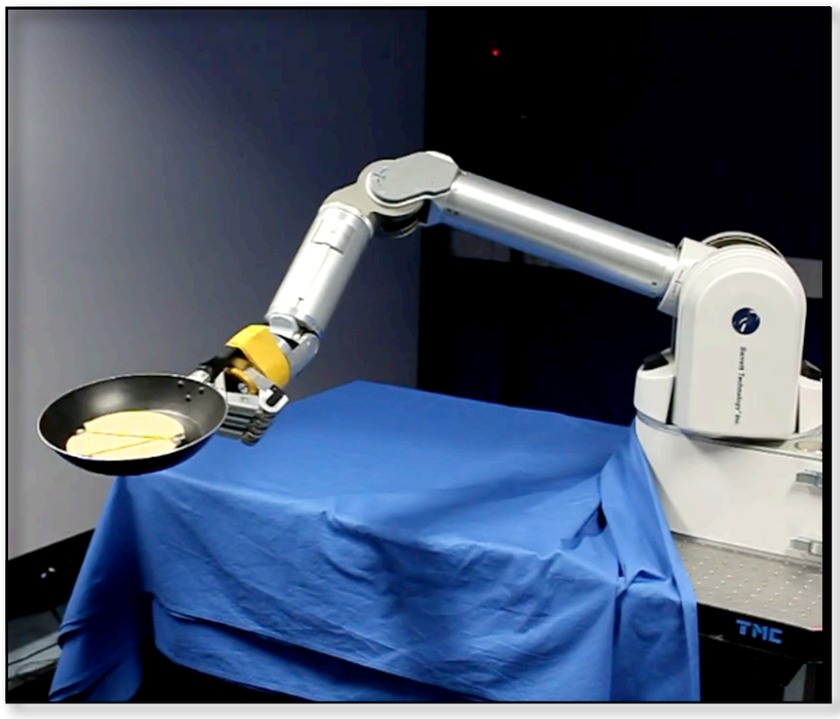
Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Robotics

State:

Action:

Reward:



(Kormushev et al., IROS 2010)

Reinforcement Learning

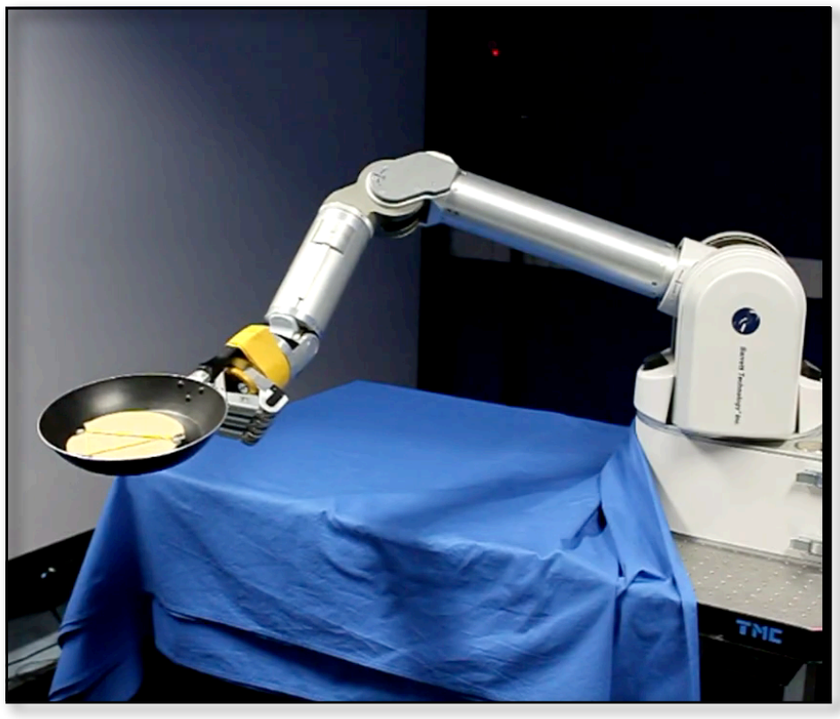
Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Robotics

State:

Action: how to move each joint

Reward:



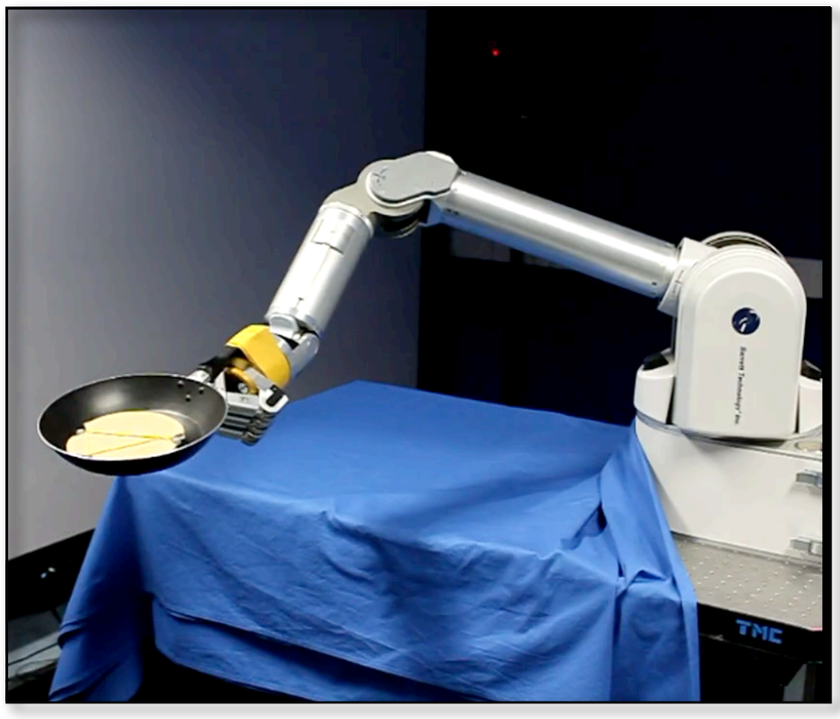
(Kormushev et al., IROS 2010)

Reinforcement Learning

Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Robotics

State: vector with information about the robot's current configuration
Action: how to move each joint
Reward:



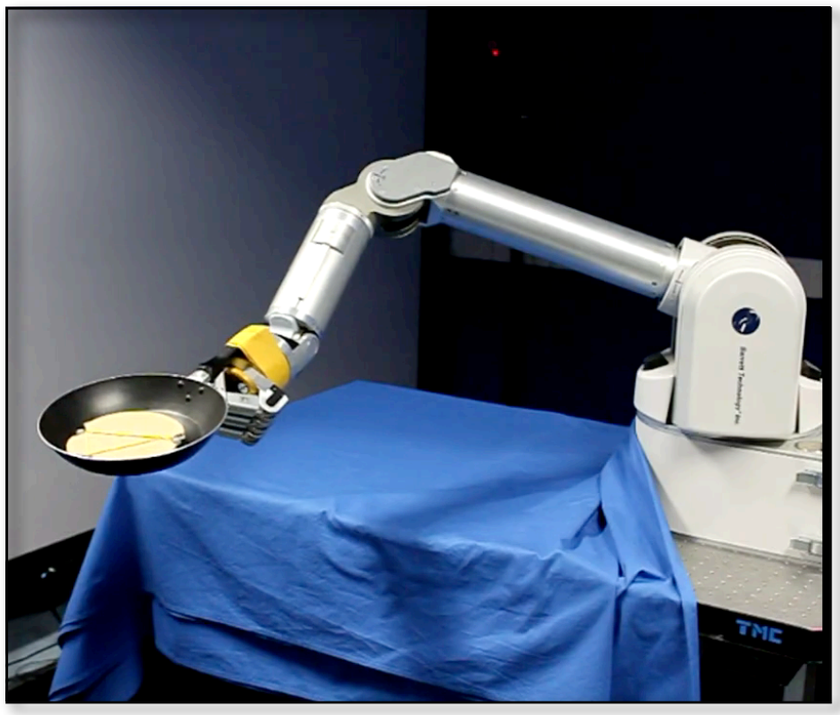
(Kormushev et al., IROS 2010)

Reinforcement Learning

Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Robotics

State: vector with information about the robot's current configuration
Action: how to move each joint
Reward: +1 if flip is successful; 0 if pancake falls off the pan



(Kormushev et al., IROS 2010)

Reinforcement Learning

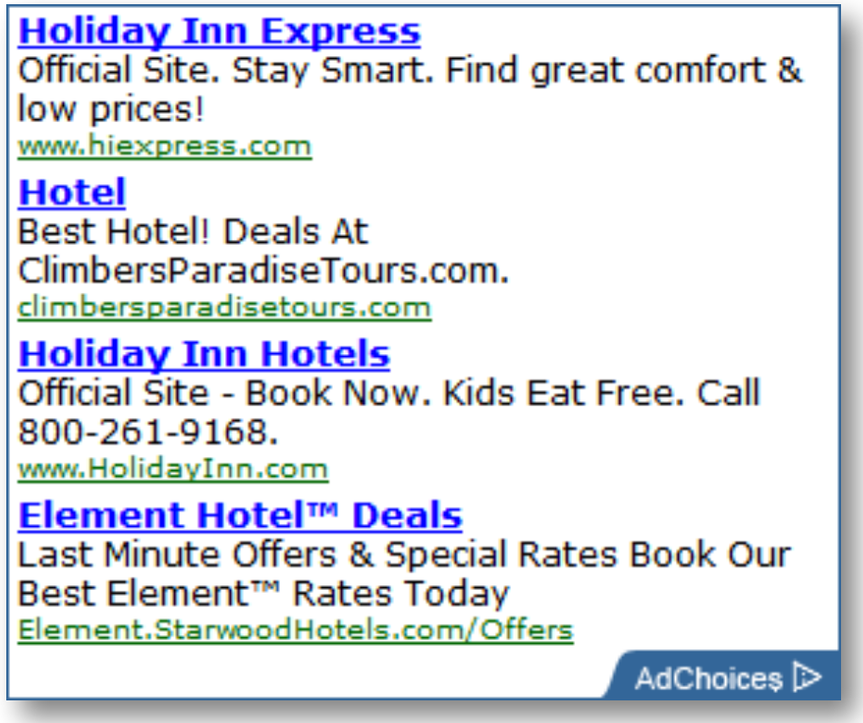
Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Digital Marketing

State:

Action:

Reward:



Reinforcement Learning

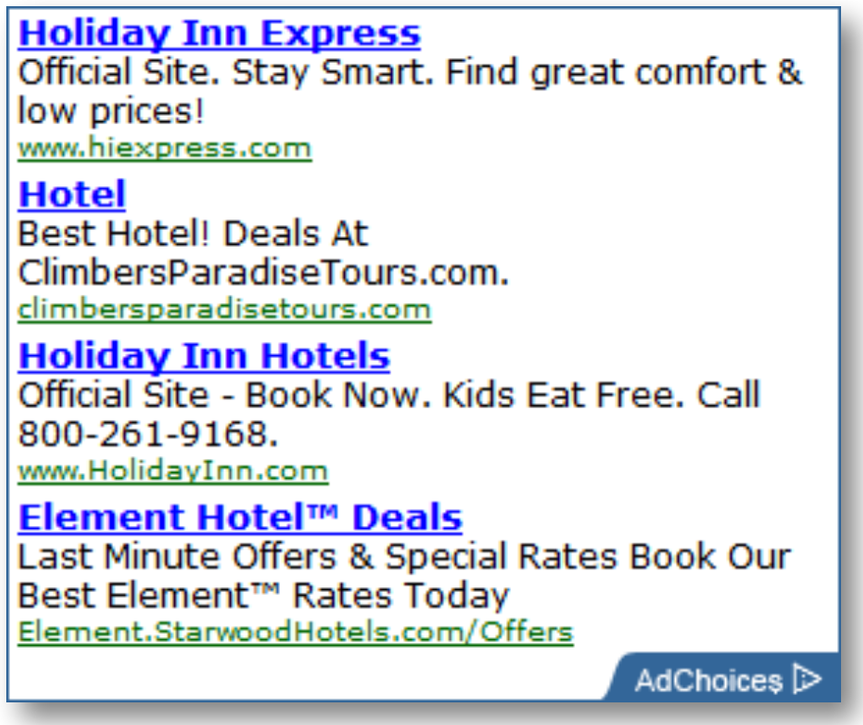
Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Digital Marketing

State:

Action: which advertisement to show

Reward:

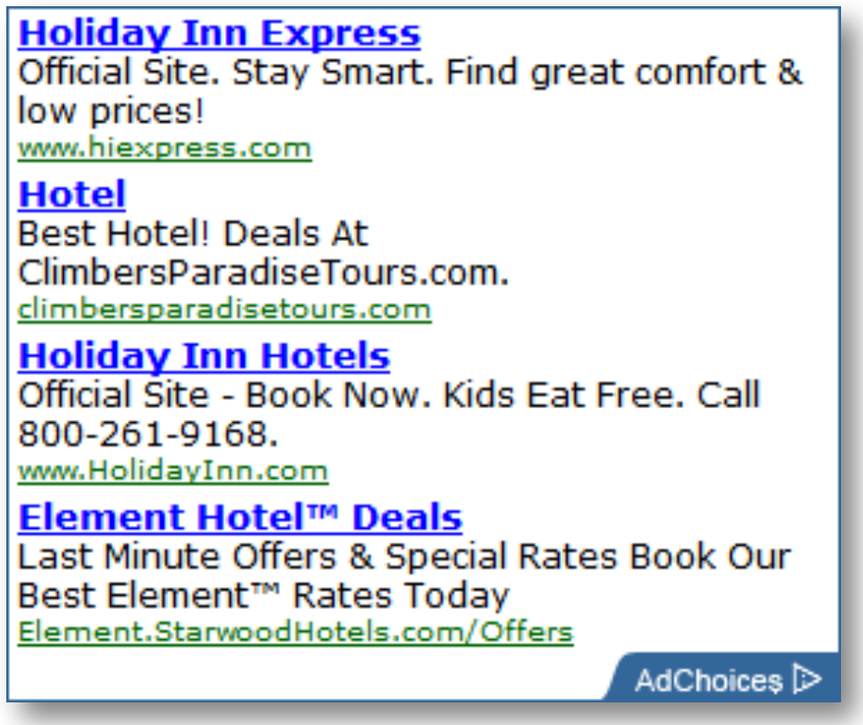


Reinforcement Learning

Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Digital Marketing

State: vector with all information known about the user
Action: which advertisement to show
Reward:



Reinforcement Learning

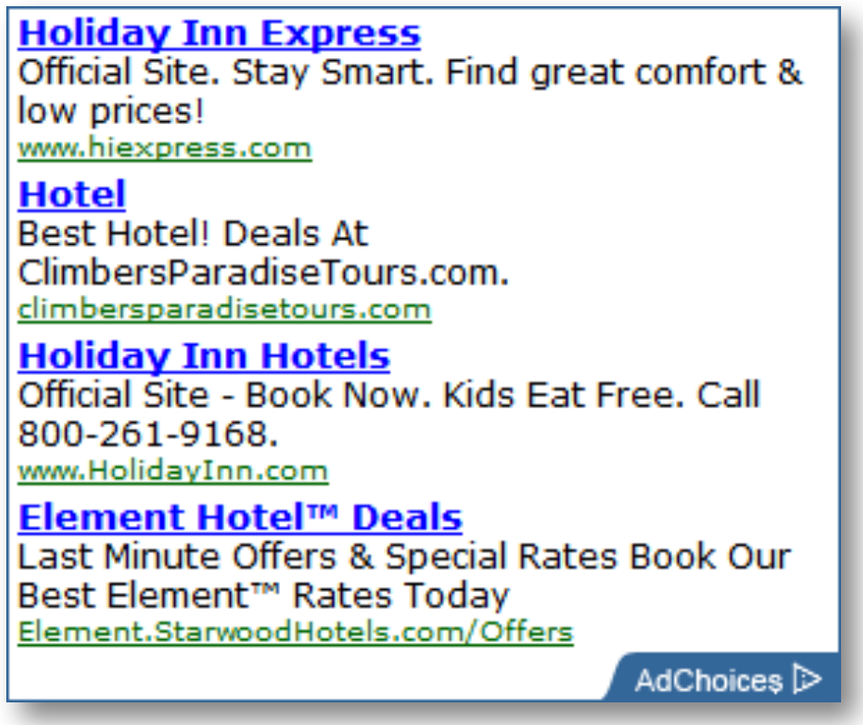
Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Digital Marketing

State: vector with all information known about the user

Action: which advertisement to show

Reward: +1 if the user clicks, 0 otherwise



Reinforcement Learning

Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Games

State:

Action: possible next plays

Reward:

Google DeepMind - AlphaGo

Reinforcement Learning

Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Games

State: current configuration of the board
Action: possible next plays
Reward:

Google DeepMind - AlphaGo

Reinforcement Learning

Behavior/**Policy**: Decision rule $\pi : S \rightarrow A$

Games

State: current configuration of the board
Action: possible next plays
Reward: +1 if win, 0 otherwise

Google DeepMind - AlphaGo

Reinforcement Learning

- **Learning what to do → map situations to actions**
  - Maximize the total amount of (possibly delayed) reward received
  - Agent is not told what actions to take

→ Why not?
→ Can't we train it by telling it how to act in a few situations?

Reinforcement Learning

- **Learning what to do → map situations to actions**
  - Maximize the total amount of (possibly delayed) reward received
  - Agent is not told what actions to take

State: current configuration of the robot joints, obstacles, orientation

Actions: how to move each joint

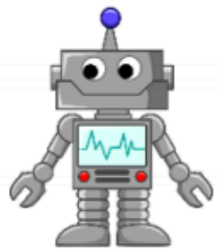
(Hess et al., arXiv 2017)

Reinforcement Learning

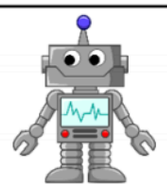
- **Learning what to do → map situations to actions**
  - Maximize the total amount of (possibly delayed) reward received
  - Agent is not told what actions to take
    - Impractical to tell the agent how to act in each possible situation
    - Discovers them by trial-and-error

Markov Decision Processes

687-Gridworld: A Simple Environment

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

687-Gridworld: A Simple Environment

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

- Agent starts in State 1
- Process ends when agent reaches State 23

- Number of states: $|\mathcal{S}| = 23$

23 “regular states"

State: Position of robot (but not the direction it is facing)

Actions: Attempt Up (AU)
Attempt Down (AD)
Attempt Left (AL)
Attempt Right (AR)

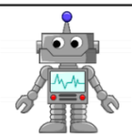
Environment Dynamics:
80% - moves in specified direction
5% - gets confused, veers right of intended direction
5% - gets confused, veers left of intended direction
10% - temporarily breaks and doesn’t move at all

If colliding with walls or obstacles, doesn’t move

Rewards:
-10 - entering the state with water
+10 - entering the goal state (and process terminates)
0 - entering any other state

Reward discount parameter: $\gamma = 0.9$
(will be discussed later)

Markov Decision Processes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

- Let $\mathcal{S}$ be the set of all possible states of the environment ("state set" or "state space")

  E.g., $\mathcal{S} = \{\text{State1}, \ldots, \text{State23}, s_\infty\}$

- Let $\mathcal{A}$ be the set of all possible actions the agent can take ("action set" or "action space")

  E.g., $\mathcal{A} = \{\text{AU}, \text{AD}, \text{AL}, \text{AR}\}$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = \textbf{?}$$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.8$$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.8$$

$$p(\text{State9},\text{AU},\text{State8}) = \Pr(S_{t+1} = \text{State8} \mid S_t = \text{State9}, A_t = \text{AU}) = \ ?$$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.8$$

$$p(\text{State9},\text{AU},\text{State8}) = \Pr(S_{t+1} = \text{State8} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.05$$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.8$$

$$p(\text{State9},\text{AU},\text{State8}) = \Pr(S_{t+1} = \text{State8} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.05$$

$$p(\text{State9},\text{AU},\text{State10}) = \Pr(S_{t+1} = \text{State10} \mid S_t = \text{State9}, A_t = \text{AU}) = \ ?$$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.8$$

$$p(\text{State9},\text{AU},\text{State8}) = \Pr(S_{t+1} = \text{State8} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.05$$

$$p(\text{State9},\text{AU},\text{State10}) = \Pr(S_{t+1} = \text{State10} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.05$$

Markov Decision Processes

- Let p be the transition function

  Describes how the state of the environment changes

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

$$p(\text{State9},\text{AU},\text{State4}) = \Pr(S_{t+1} = \text{State4} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.8$$

$$p(\text{State9},\text{AU},\text{State8}) = \Pr(S_{t+1} = \text{State8} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.05$$

$$p(\text{State9},\text{AU},\text{State10}) = \Pr(S_{t+1} = \text{State10} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.05$$

$$p(\text{State9},\text{AU},\text{State9}) = \Pr(S_{t+1} = \text{State9} \mid S_t = \text{State9}, A_t = \text{AU}) = 0.1$$

Markov Decision Processes

- Let R be the reward function:

Rewards:
-10 - entering the state with water
+10 - entering the goal state (and process terminates)
0 - entering any other state

Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**End** State 23

Markov Decision Processes

- Let γ be the reward discount parameter

  $\gamma \in [0,1]$

  (will be discussed later)

Markov Decision Processes

- So these are the elements of an MDP

Now, how to we encode the *behavior* that the agent could/should execute?

Policies

- Policy: decision rule - how agent selects actions

  maps states to actions (determines behavior of agent)

Current state	Action to take
State1	AR
State2	AU
State3	AL
State4	AL
...	

This is a deterministic policy
More generally, policies can be stochastic

Policies

- Policy: decision rule - how agent selects actions

  maps states to actions (determines behavior of agent)

Current state	Probability of taking action AU	Probability of taking action AD	Probability of taking action AL	Probability of taking action AR
State1	0	0.1	0.3	0.6
State2	0.8	0	0	0.2
State3	0.1	0.1	0.5	0.3
State4	0.25	0.25	0.25	0.25
…				

- Let the function π be a policy

What is, e.g., π(State3,AD) ?

Policies

- Policy: decision rule - how agent selects actions

  maps states to actions (determines behavior of agent)

Current state	Probability of taking action AU	Probability of taking action AD	Probability of taking action AL	Probability of taking action AR
State1	0	0.1	0.3	0.6
State2	0.8	0	0	0.2
State3	0.1	0.1	0.5	0.3
State4	0.25	0.25	0.25	0.25
…				

- Let the function π be a policy

What is, e.g., $\pi(\text{State3},\text{AD}) = 0.1$

Policies

- Policy: decision rule - how agent selects actions

  maps states to actions (determines behavior of agent)

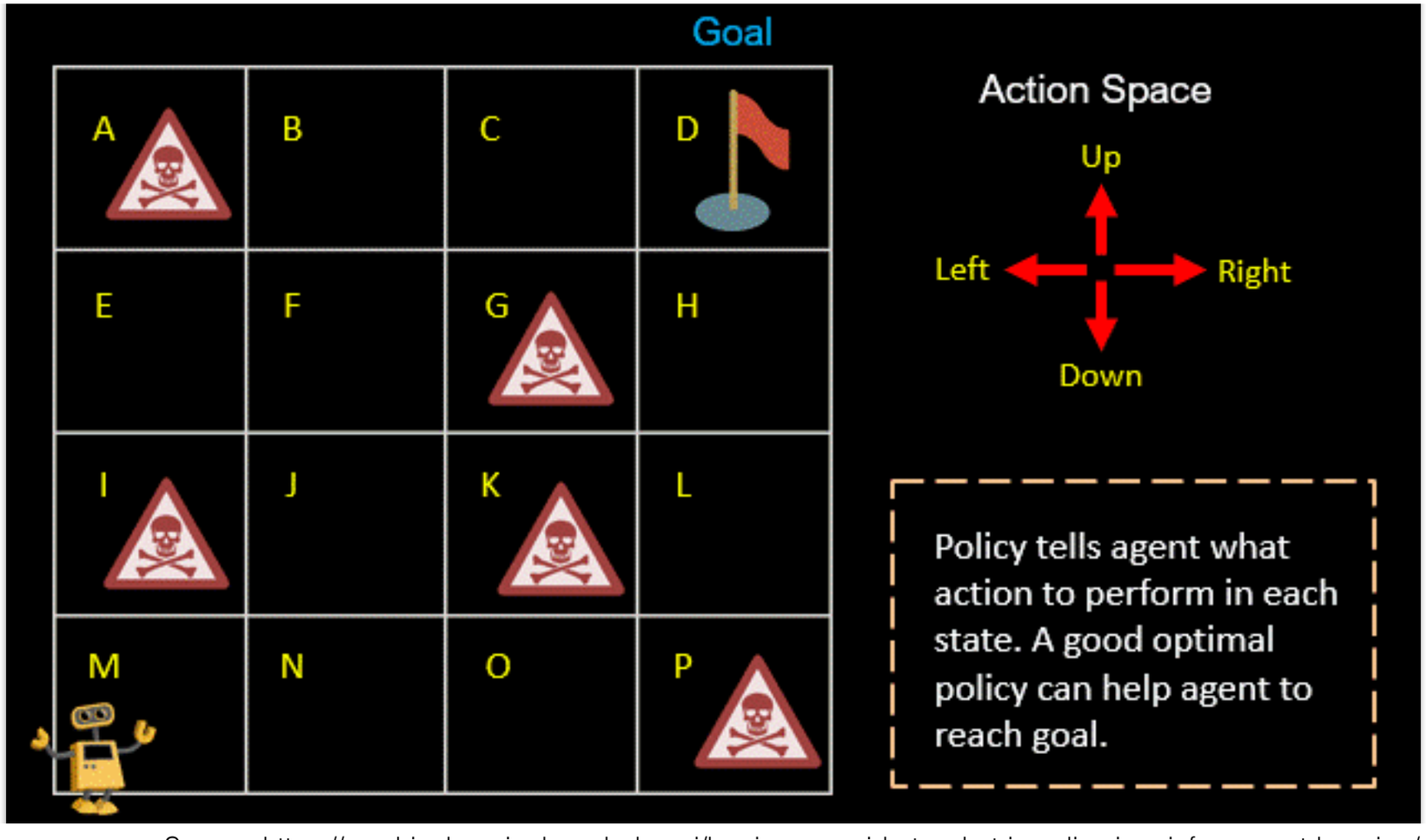
Current state	Probability of taking action AU	Probability of taking action AD	Probability of taking action AL	Probability of taking action AR
State1	0	0.1	0.3	0.6
State2	0.8	0	0	0.2
State3	0.1	0.1	0.5	0.3
State4	0.25	0.25	0.25	0.25
…				

- Let the function π be a policy

Learning: corresponds to agent changing its policy

Agent's Goal

- Find a policy, π^*, called an optimal policy
- Intuitively, an agent that follows π^*
  maximizes the expected total amount of reward
  that it will obtain



Source: https://machinelearningknowledge.ai/beginners-guide-to-what-is-policy-in-reinforcement-learning/

Agent-Environment Interaction

Algorithm 2: Pseudocode for an agent interacting with an environment.

```
for episode = 0, 1, 2, ... do
    s ~ d_0;                                  // Samples initial state
    for t = 0 to ∞ do
        a = agent.getAction(s);               // Samples action from policy
        s' ~ p(s, a, ·);
        r = R(s, a);
        agent.train(s, a, r, s');             // May update π based on experience
        if s' == s_∞ then
            break;                            // Exits out of loop if episode ended
        s = s';
    agent.newEpisode();                       // Alerts agent that episode ended
```

Discounting

- Let γ be the reward discount parameter

$$\gamma \in [0,1]$$

"Will be discussed ~~later~~ **now**"

Discounting

The Marshmallow Test

"Here's a marshmallow.

*You can either **wait**, and I'll give you another one, so **you'll have two**,*

*or you can **eat it now**"*

Igniter Media, 2009 - https://www.youtube.com/watch?v=QX_oy9614HQ

Discounting

The Marshmallow Test

"Here's a marshmallow.

*You can either **wait**, and I'll give you another one, so **you'll have two**,*

*or you can **eat it now***"

- Many people pick one marshmallow *now* when presented with these options
- Suggests that rewards obtained in the distant future are worth less to us than rewards in the near future
- Reward discount parameter, γ
  - encodes, in the objective function, discounting of rewards based on how distant in the future they may occur

Discounting

- Recall that an agent that follows an optimal policy, π^*, maximizes the expected total amount of reward

  Intuitively, what is a "good" action to take when in state S_t ?

  1) possibly good "immediate" reward (in the current state)

  2) but also that takes agent to future states from which it can collect lots of rewards

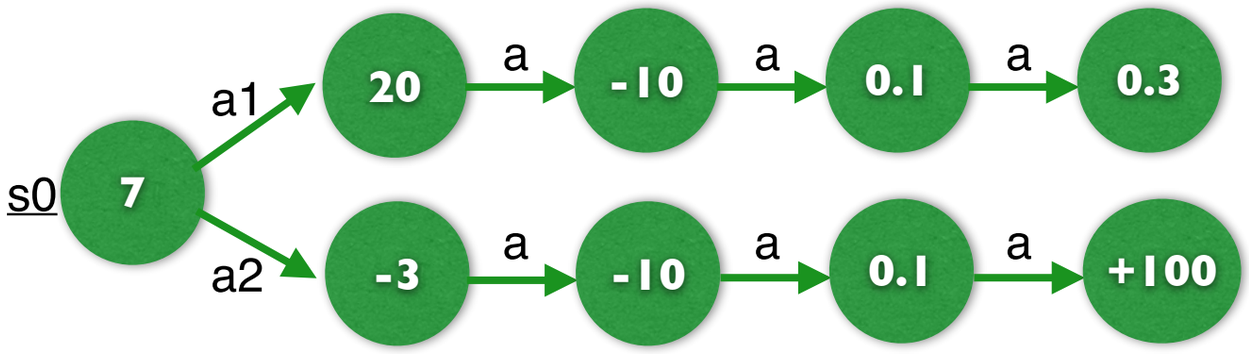
Discounting

- Recall that an agent that follows an optimal policy, π^*, maximizes the expected total amount of reward

Intuitively, what is a "good" action to take when in state S_t ?

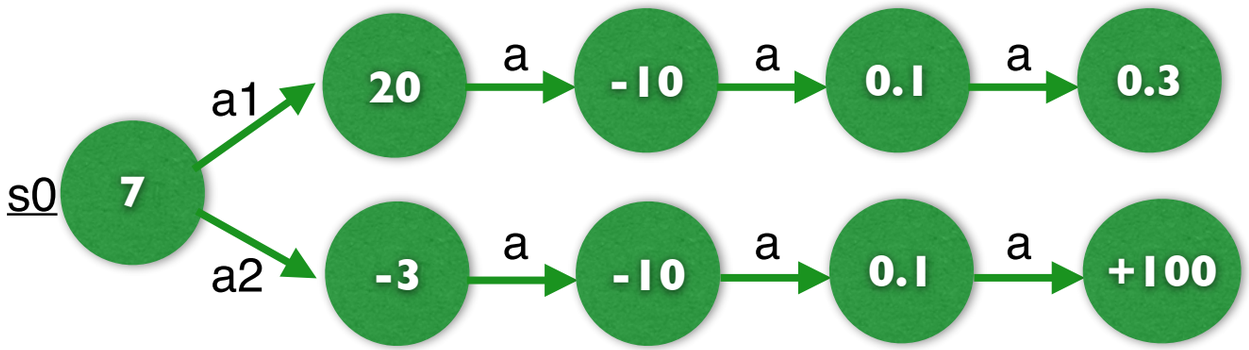
1) possibly good "immediate" reward (in the current state)

2) but also that takes agent to future states from which it can collect lots of rewards



which action to take in s_0 to obtain the largest sum of rewards?

Discounting



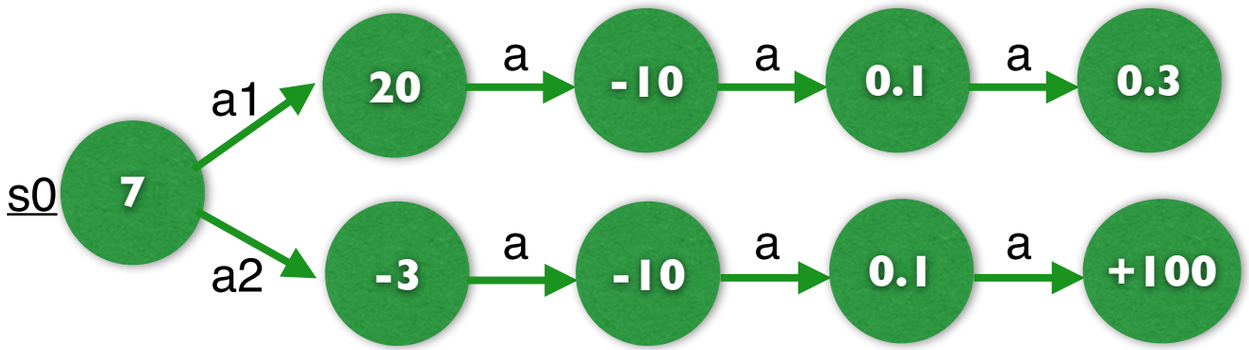
Should policy π take action a_1 or a_2, when in state s_0 ?

$$J(\pi) = 7 + \begin{cases} a_1 : 20 + (-10) + 0.1 + 0.3 \\ a_2 : -3 + (-10) + 0.1 + 100 \end{cases}$$

= 7 + (-3) + (-10) + 0.1 + 100 = 94.1

Are rewards a few steps ahead worth the same to us?

Discounting



Should policy π take action a_1 or a_2, when in state s_0 ?

$$J(\pi) = 7 + \begin{cases} a_1 : 20 + (-10) + 0.1 + 0.3 \\ a_2 : -3 + (-10) + 0.1 + 100 \end{cases}$$

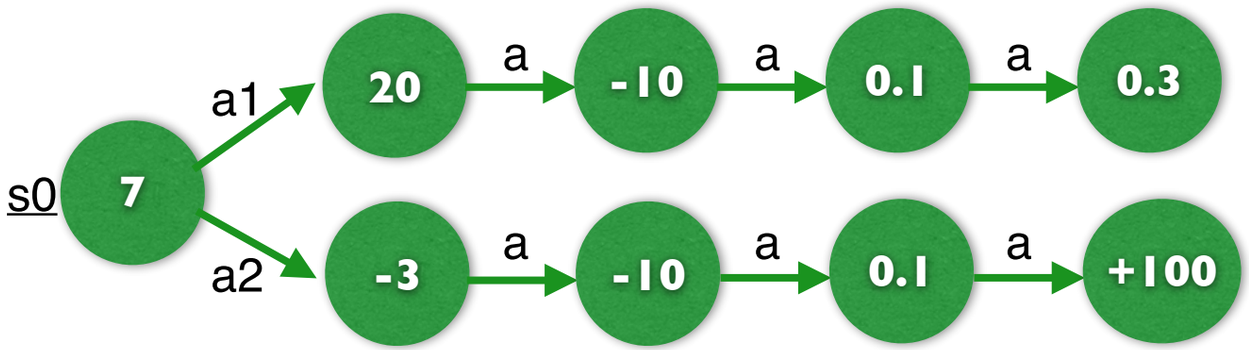
= 7 + (-3) + (-10) + 0.1 + 100 = 94.1

Are rewards a few steps ahead worth the same to us?

Agent could:

- execute a_1, get a good immediate reward (20) but then keep getting low rewards
- execute a_2, get low rewards for a while, but then get a really large reward (100)

Discounting



Should policy π take action a_1 or a_2, when in state s_0 ?

$$J(\pi) = 7 + \begin{cases} a_1 : 20 + (-10) + 0.1 + 0.3 \\ a_2 : -3 + (-10) + 0.1 + 100 \end{cases}$$

= 7 + (-3) + (-10) + 0.1 + 100 = 94.1

Are rewards a few steps ahead worth the same to us?

$$a_1 : 20 + \gamma(-10) + \gamma^2 0.1 + \gamma^3 0.3$$

$$a_2 : -3 + \gamma(-10) + \gamma^2 0.1 + \gamma^3 100$$

γ = 0?

γ = 1?

γ = [0,1)?

Q-Learning

1. Agent is in s, picks action a.

2. Receives reward r and observes next state s'

3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning

$$Q_k(s, a)$$

current estimate of how much return it can get if executing action a in state s

1. Agent is in s, picks action a.

2. Receives reward r and observes next state s'

3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow \qquad Q_k(s,a)$$

new estimate of how much return it can get if executing action a in state s

current estimate of how much return it can get if executing action a in state s

1. Agent is in s, picks action a.

2. Receives reward r and observes next state s'

3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.

2. Receives reward r and observes next state s'

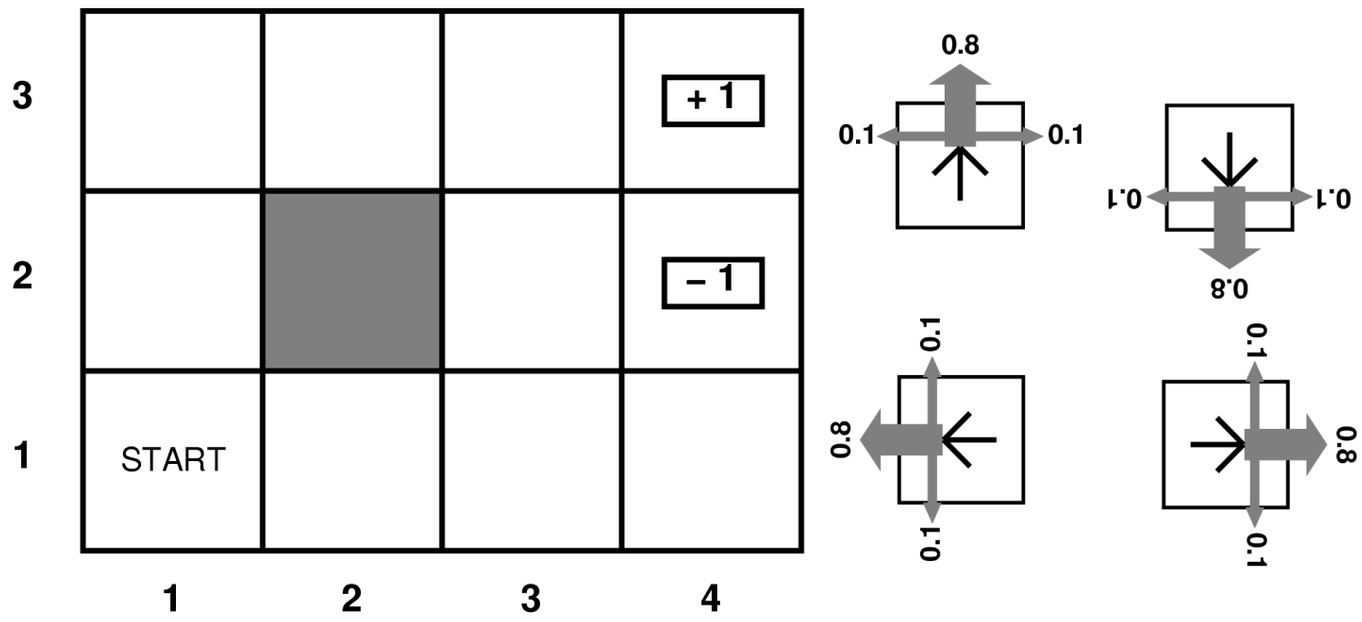
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



reward -0.04 after every action
except when entering the rightmost states

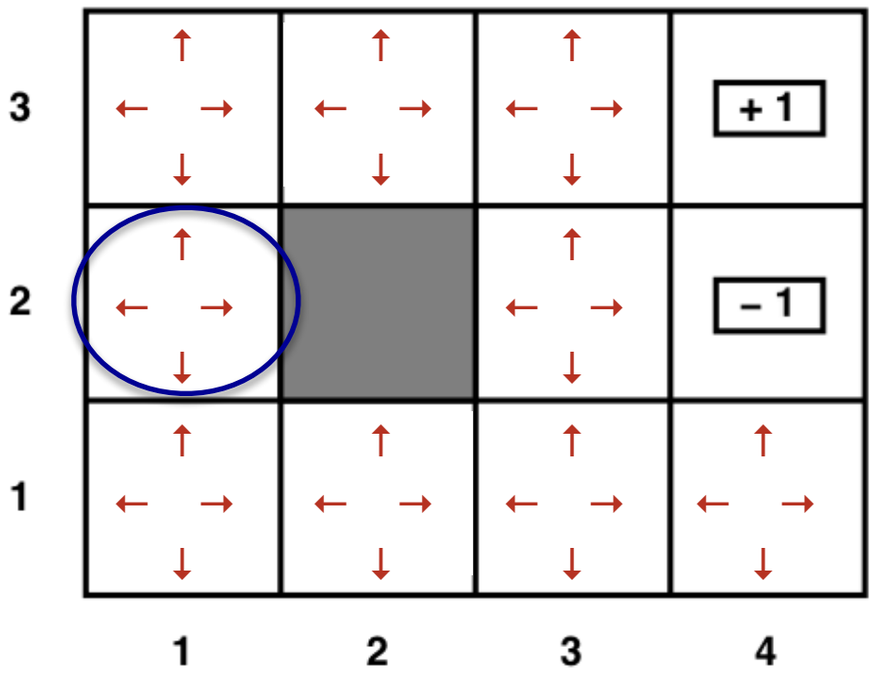
Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm

- Stores for each state the preference of the agent for taking each one of the actions: Q(s,a)
- E.g., in state (2,1), we'd have
  - Q((2,1), ↑) = preference for action UP when in state (2,1)
  - Q((2,1), ←) = preference for action LEFT when in state (2,1)
  - Q((2,1), →) = preference for action RIGHT when in state (2,1)
  - Q((2,1), ↓) = preference for action DOWN when in state (2,1)

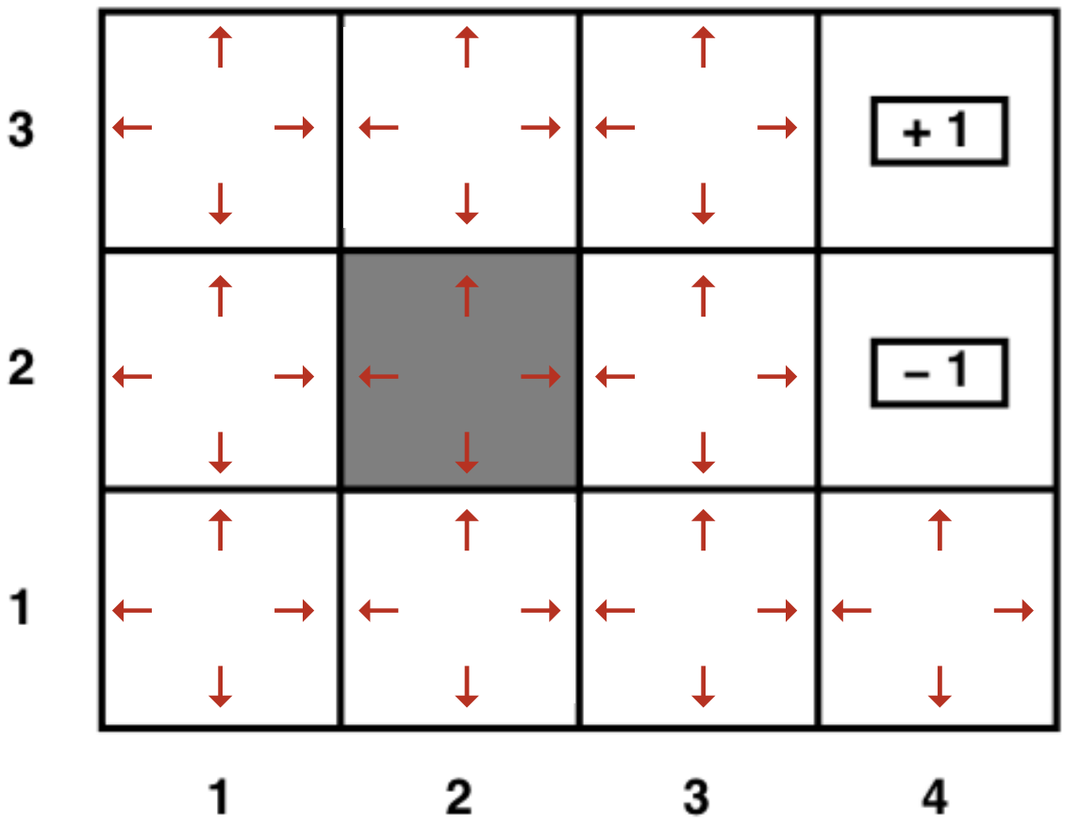


Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



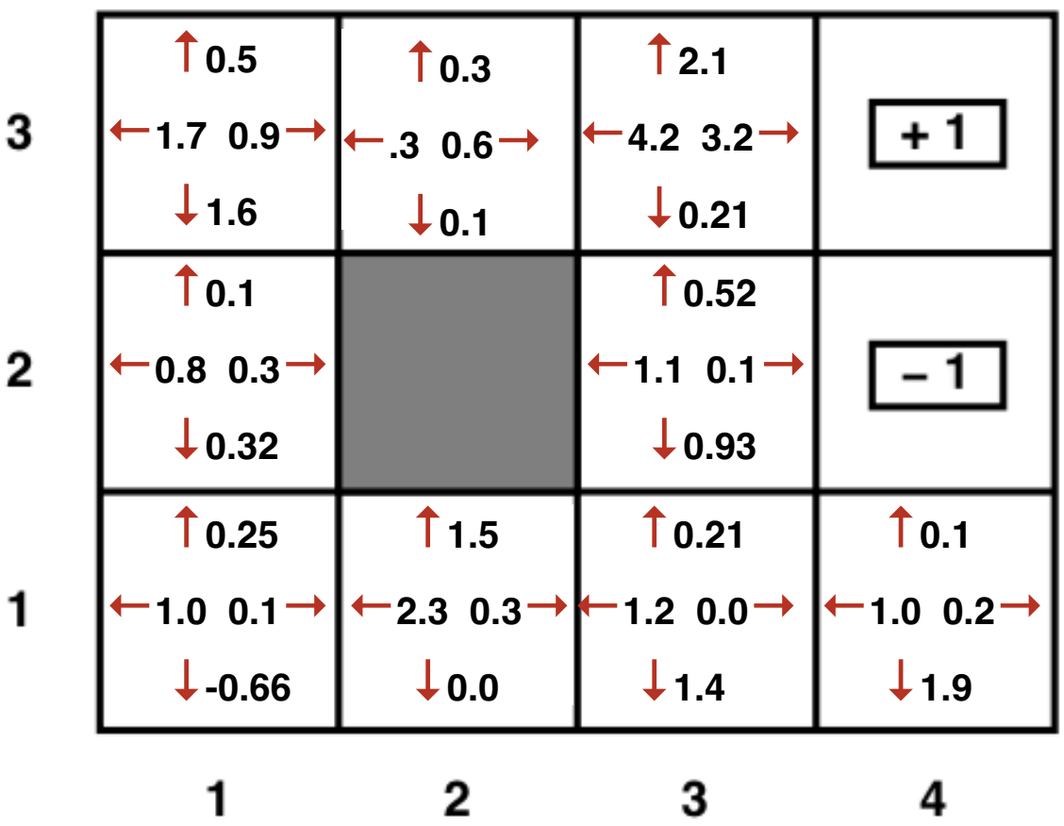
- Initialize the Q table
  - Q(s,a) ← some number (zero or random), ∀ **s,a**

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



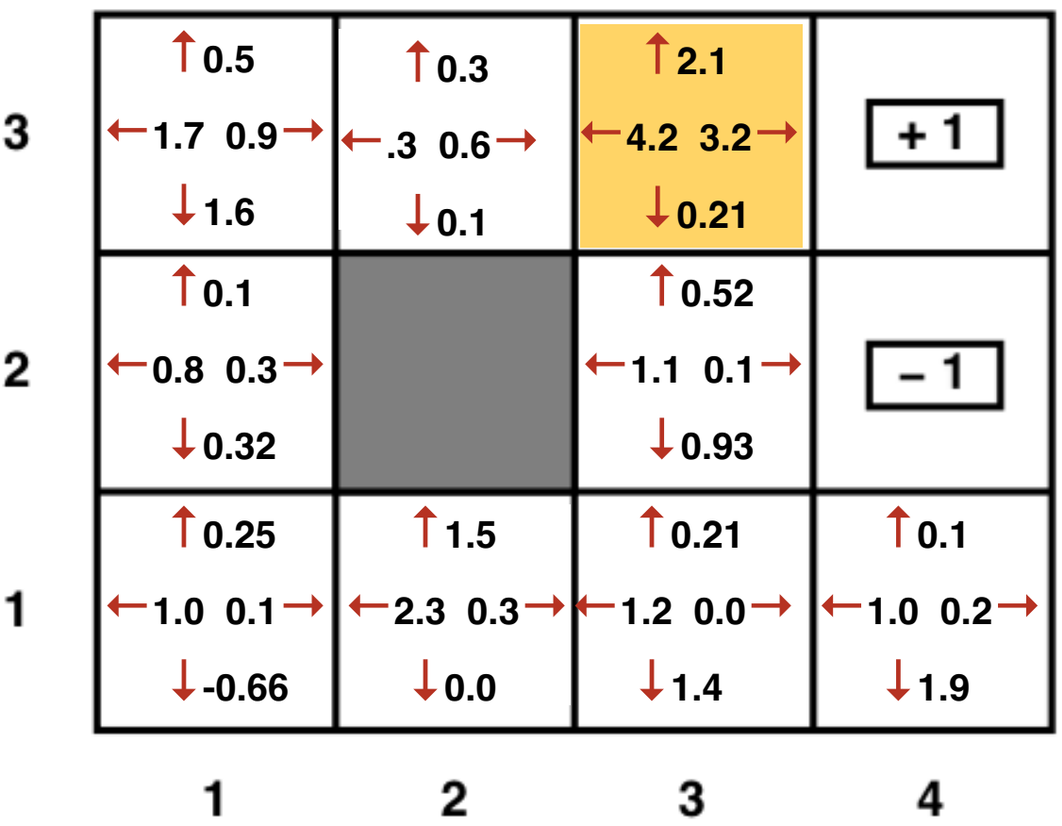
- Initialize the Q table
  - Q(s,a) ← some number (zero or random), ∀ **s,a**

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



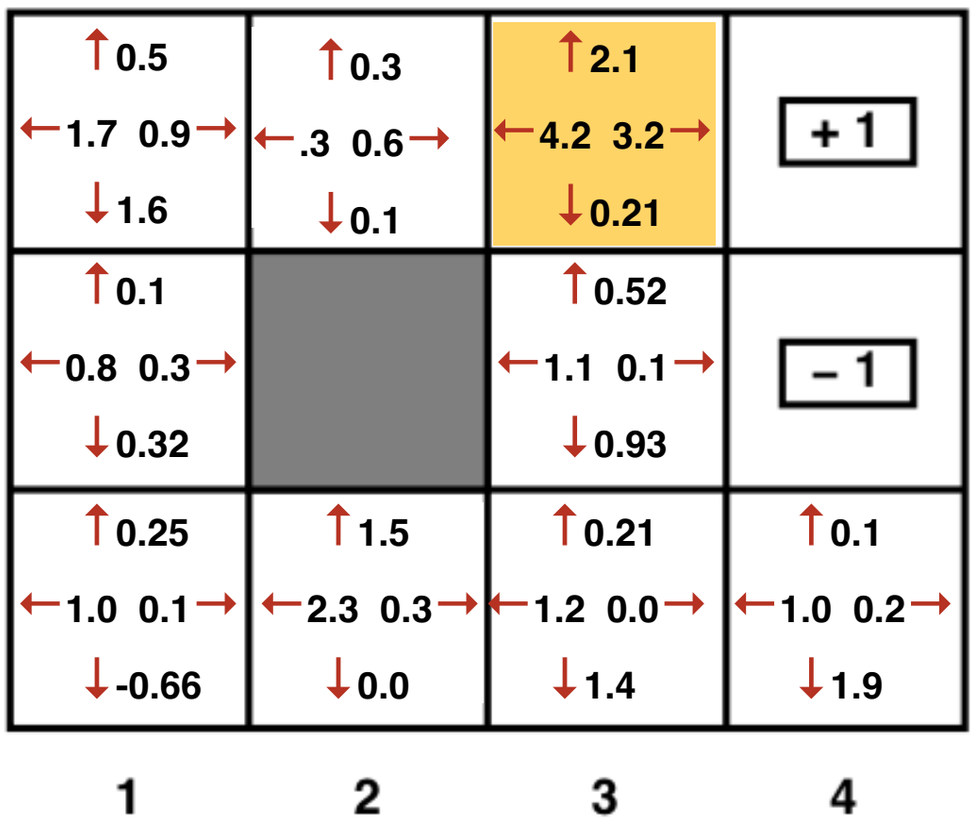
- Assume the agent was in state (3,3)

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Q((3,3), ↑)=2.1
  - Q((3,3), ←)=4.2
  - Q((3,3), →)=3.2
  - Q((3,3), ↓)=0.21

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

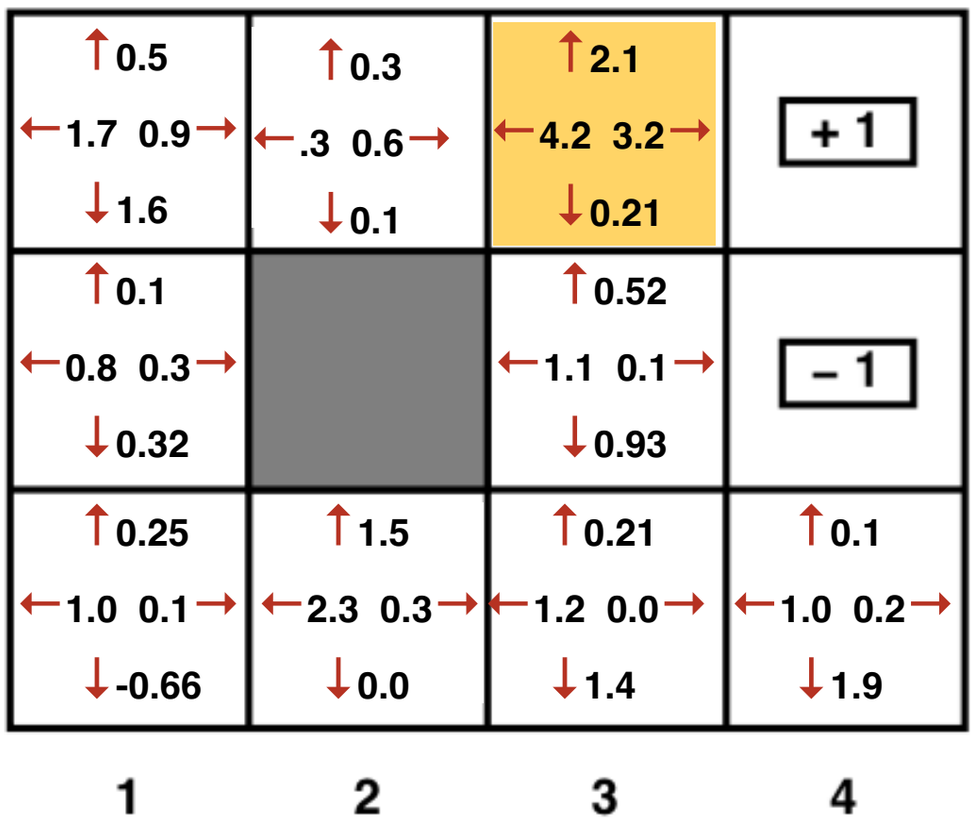
$$\pi(s) = \leftarrow$$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



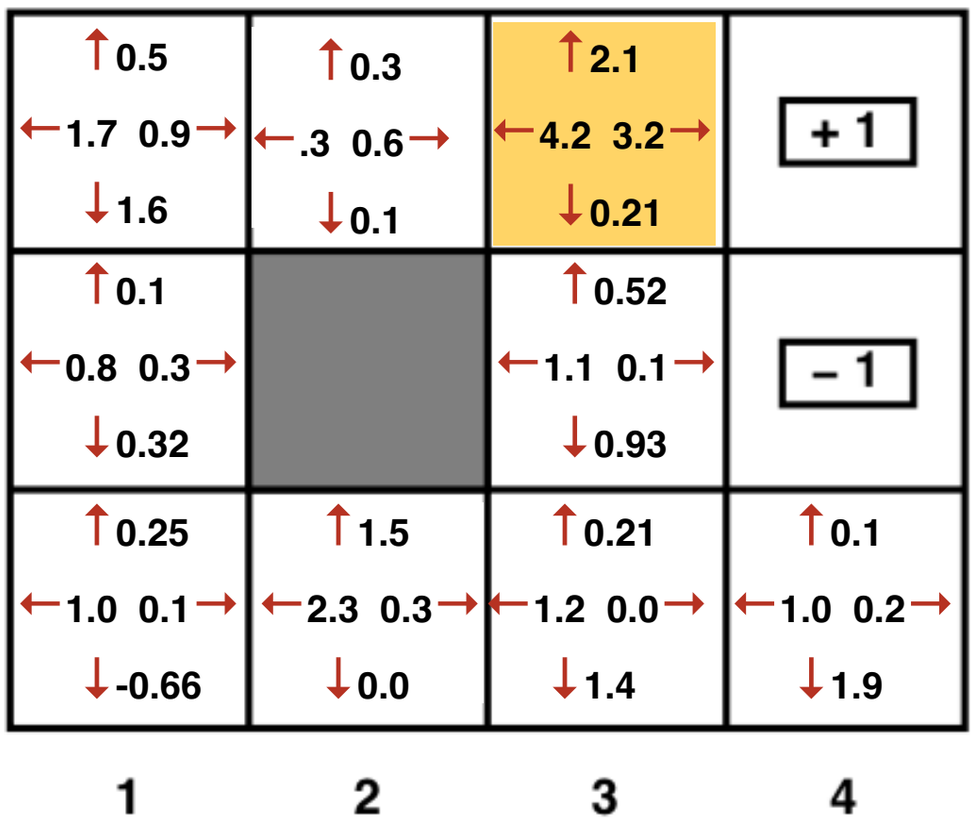
- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



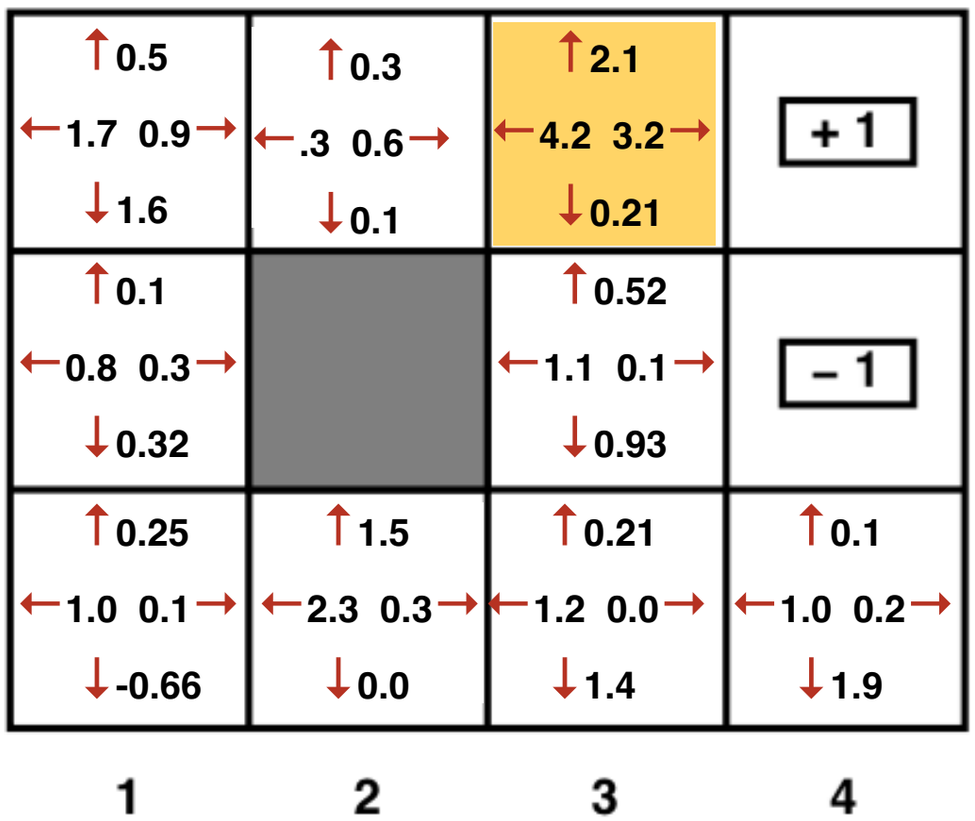
- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = ?
    - s' = ?

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



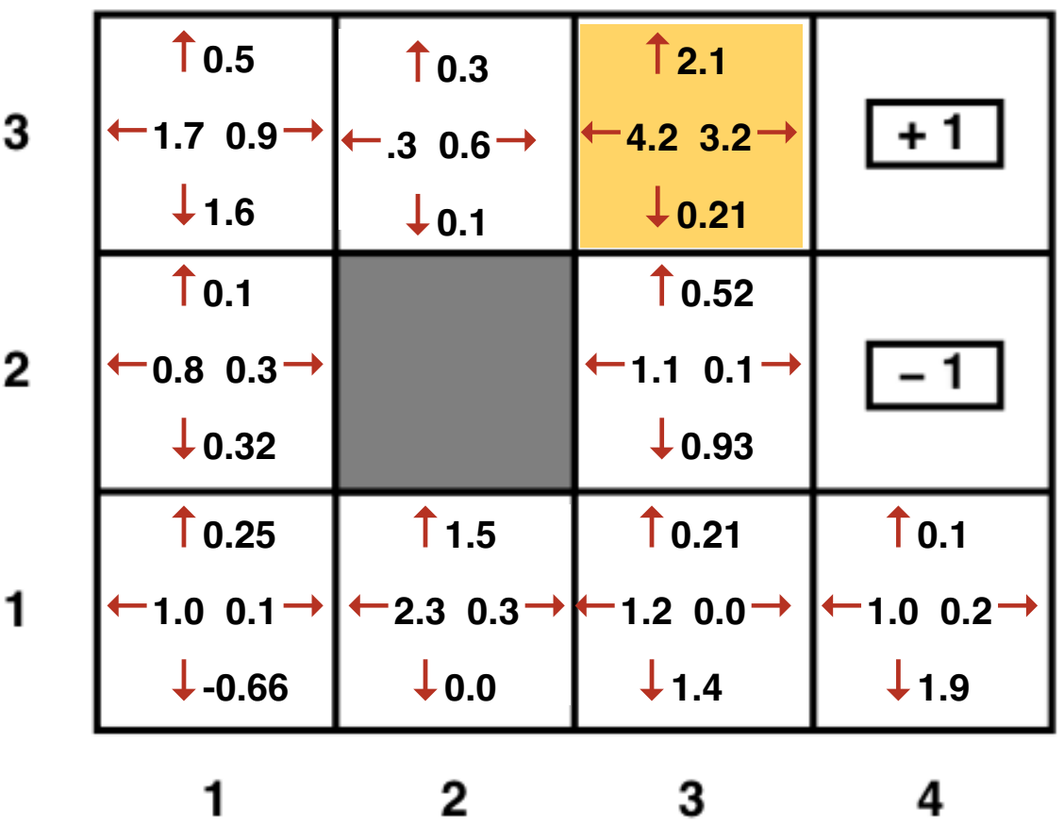
- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
- Executed action ←
  - s = **(3,3)**
  - a = ←
  - r = **-0.04**
  - s' = **(3,2)**

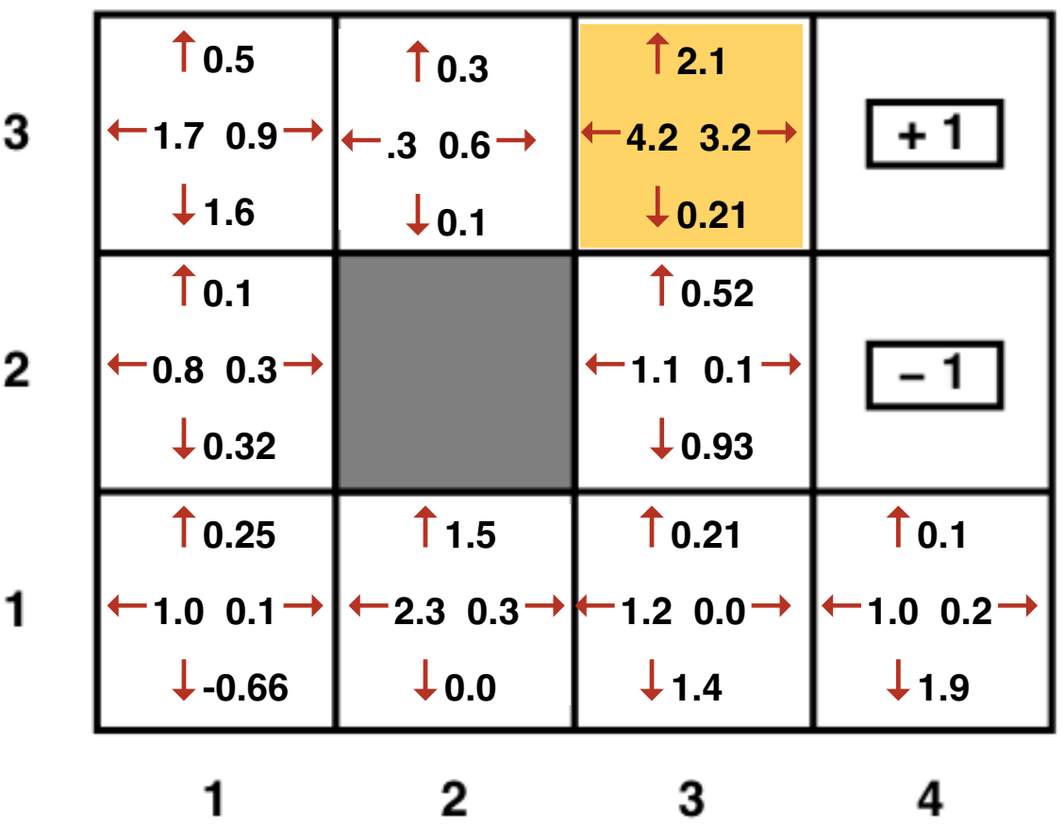
Expected to get 4.2 total reward by executing LEFT

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
- Executed action ←
  - s = **(3,3)**
  - a = ←
  - r = **-0.04**
  - s' = **(3,2)**

Expected to get 4.2 total reward by executing LEFT

But in reality…

it received a first reward of -0.04
and went to a state from which it thinks it can get 0.6

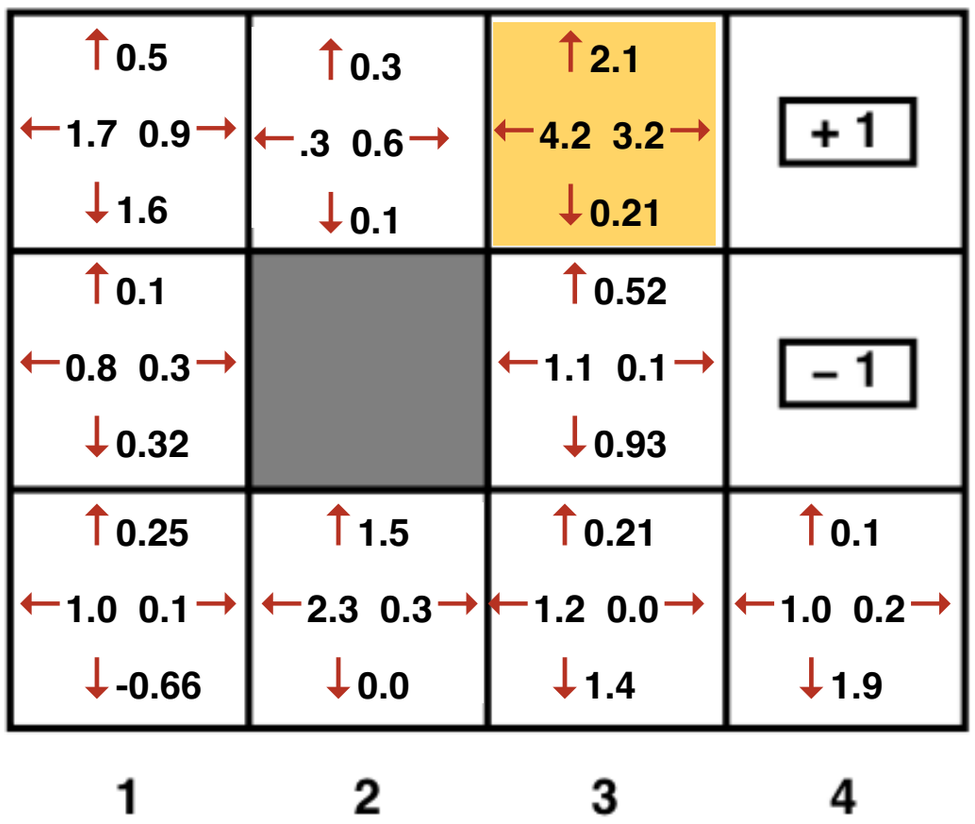
Based on this last experience, it now thinks it can get
-0.04 + 0.6 = 0.56 ***(way less than 4.2!)***

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



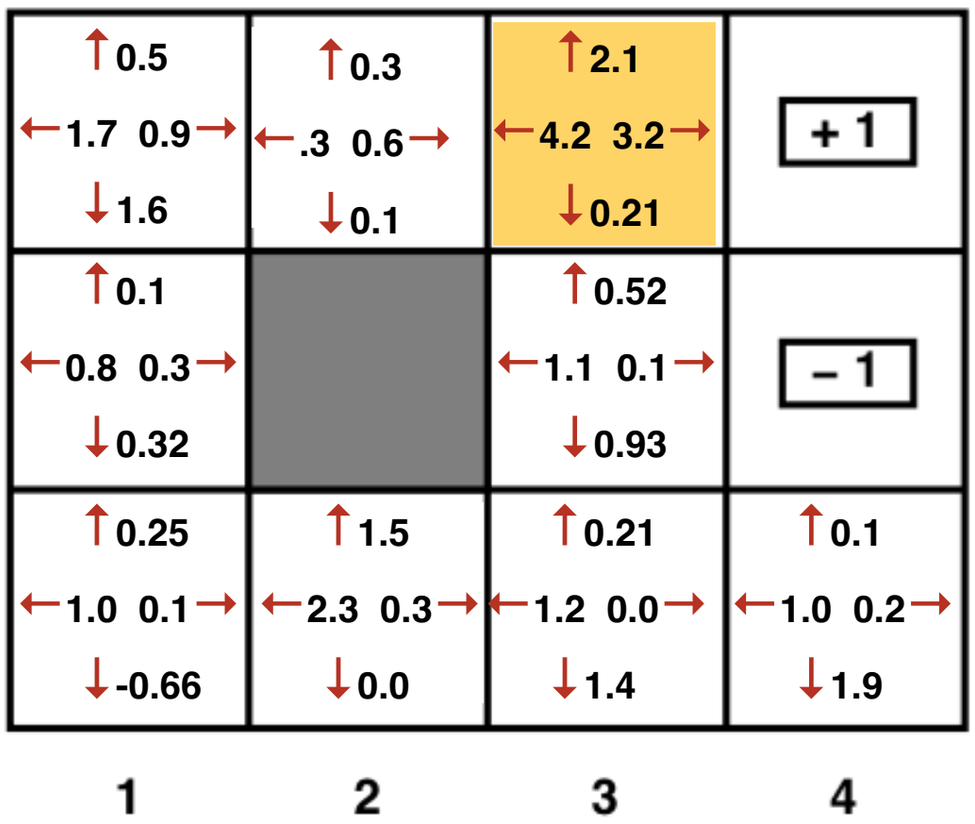
- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' =**(3,2)**

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' =**(3,2)**

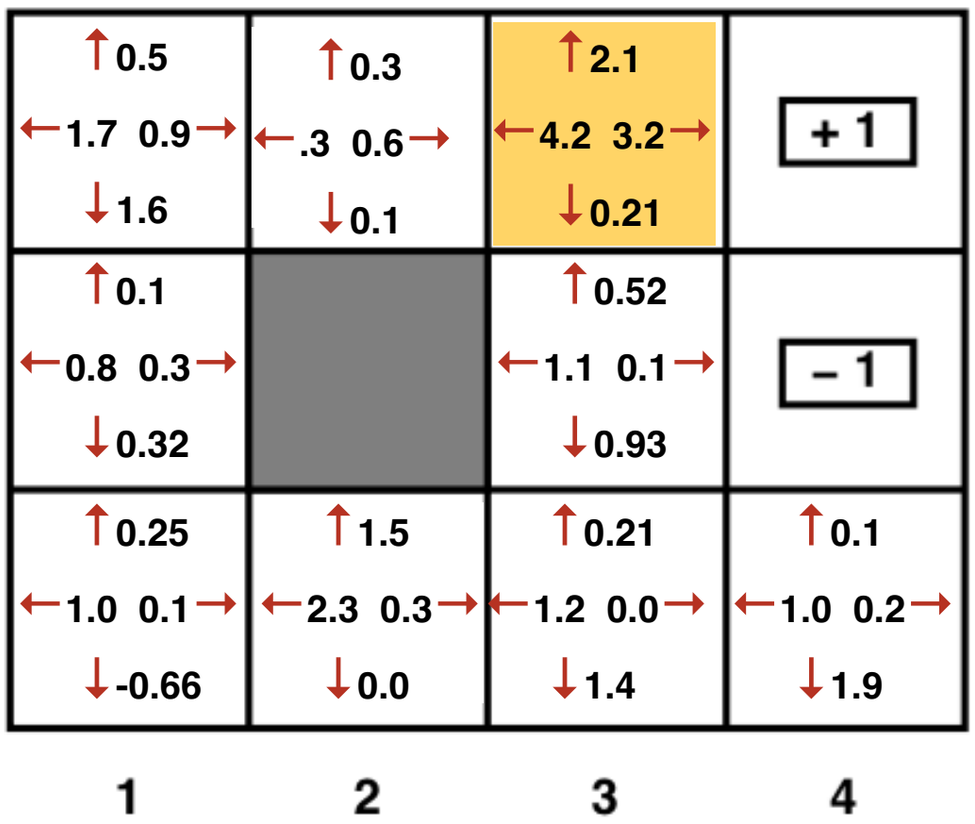
$$\left(r + \max_{a'} Q_k(s',a')\right)$$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

$$\left(r + \max_{a'} Q_k(s',a')\right)$$

-0.04

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm

	1	2	3	4
3	↑0.5 ←1.7 0.9→ ↓1.6	↑0.3 ←.3 0.6→ ↓0.1	↑2.1 ←4.2 3.2→ ↓0.21	+1
2	↑0.1 ←0.8 0.3→ ↓0.32		↑0.52 ←1.1 0.1→ ↓0.93	−1
1	↑0.25 ←1.0 0.1→ ↓-0.66	↑1.5 ←2.3 0.3→ ↓0.0	↑0.21 ←1.2 0.0→ ↓1.4	↑0.1 ←1.0 0.2→ ↓1.9

- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' =**(3,2)**

$$\left(r + \max_{a'} Q_k(s',a')\right)$$

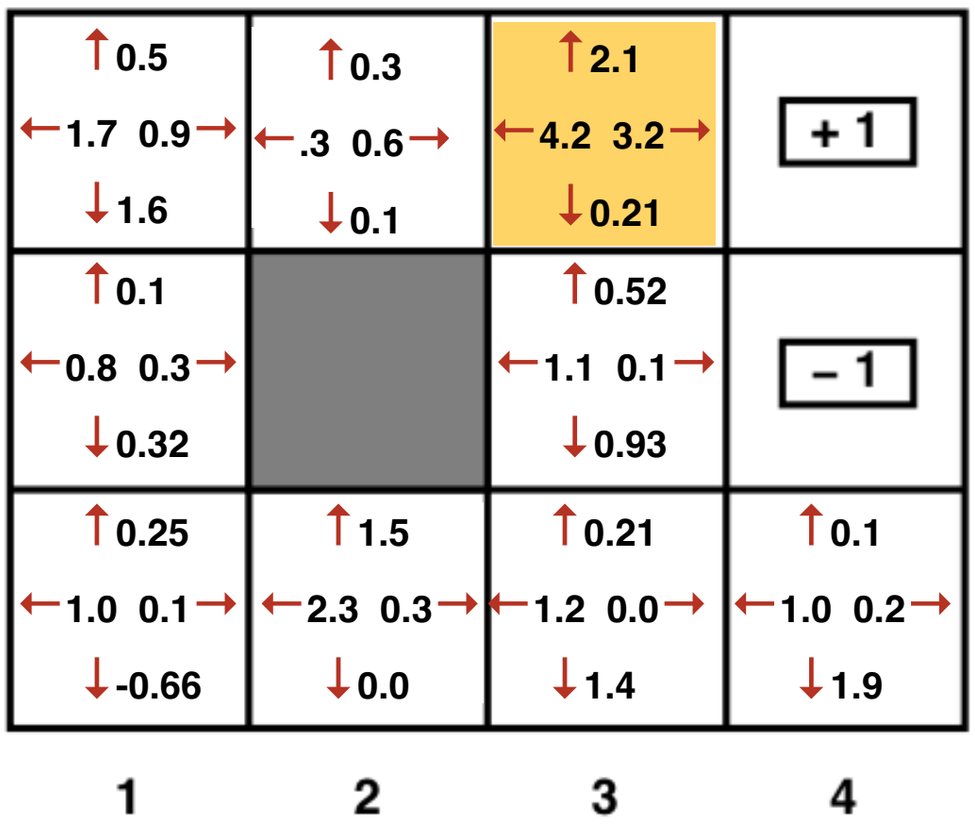
-0.04 **0.6**

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

$$Q_{k+1}(s,a) \leftarrow \left(r + \max_{a'} Q_k(s',a')\right)$$

Should be updated towards
-0.04 + 0.6 = 0.56

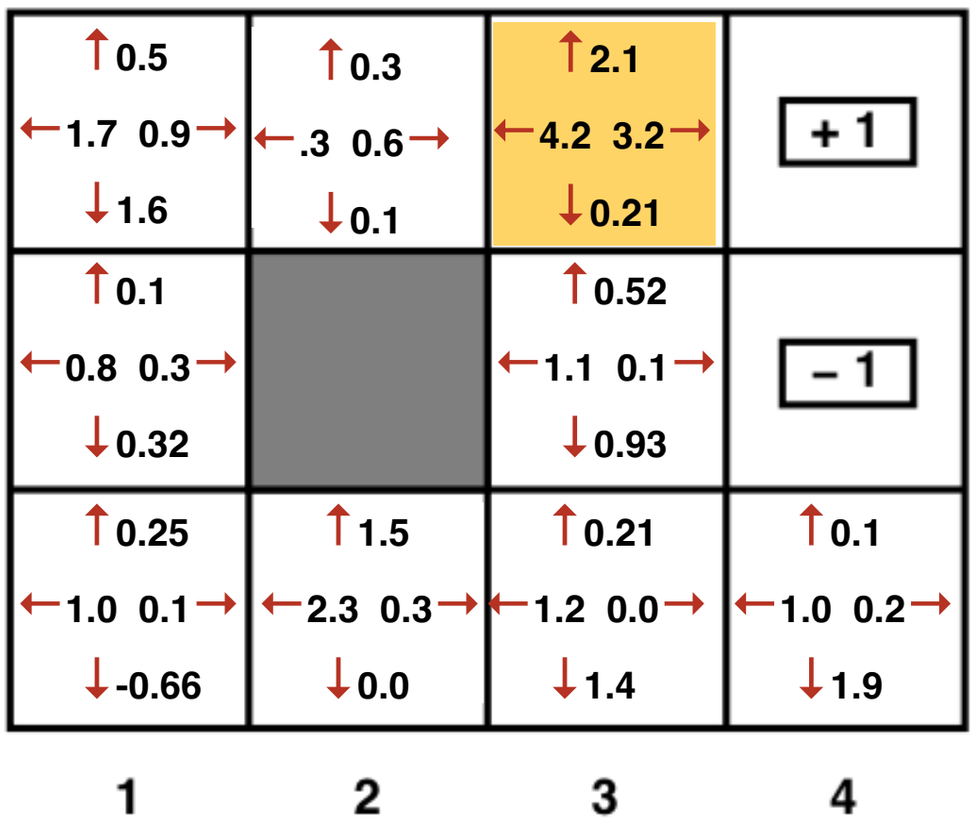
-0.04 0.6

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

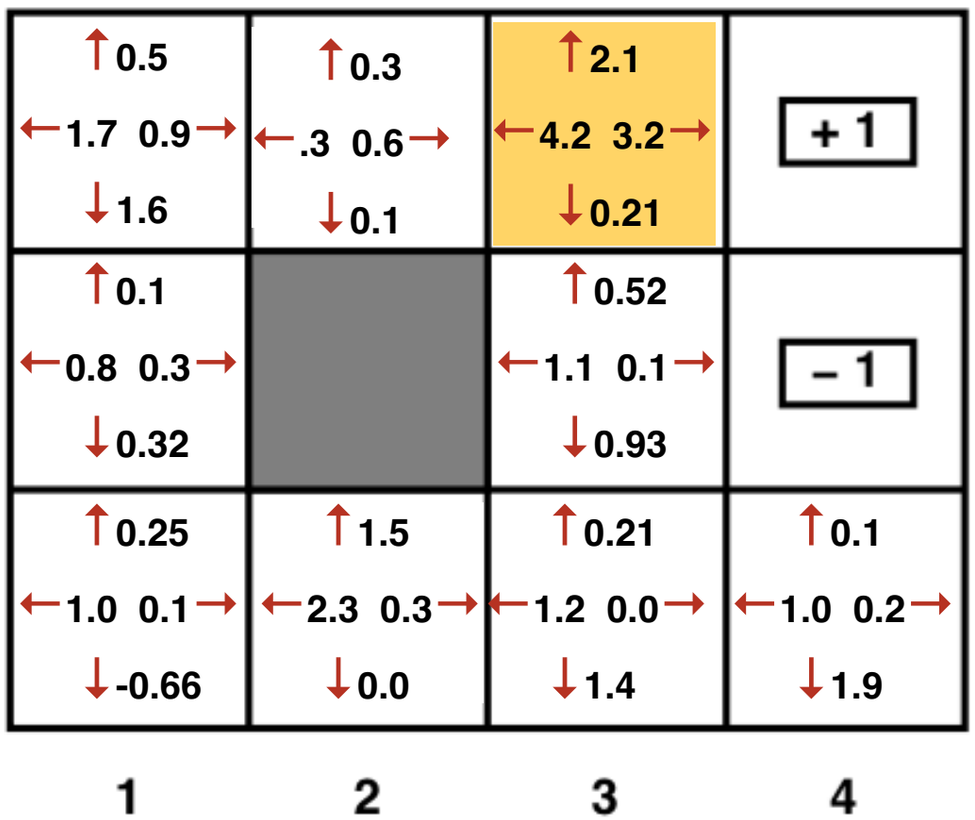
$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' =**(3,2)**

assume
α = 0.5
γ = 1.0

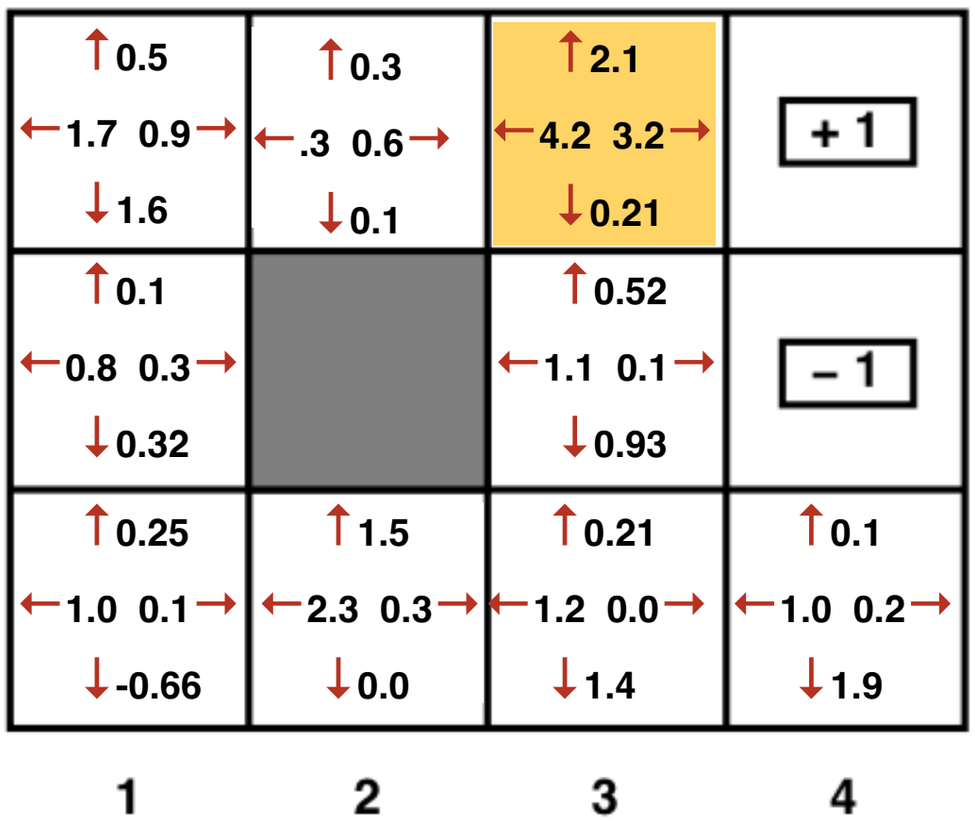
$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

assume
α = 0.5
γ = 1.0

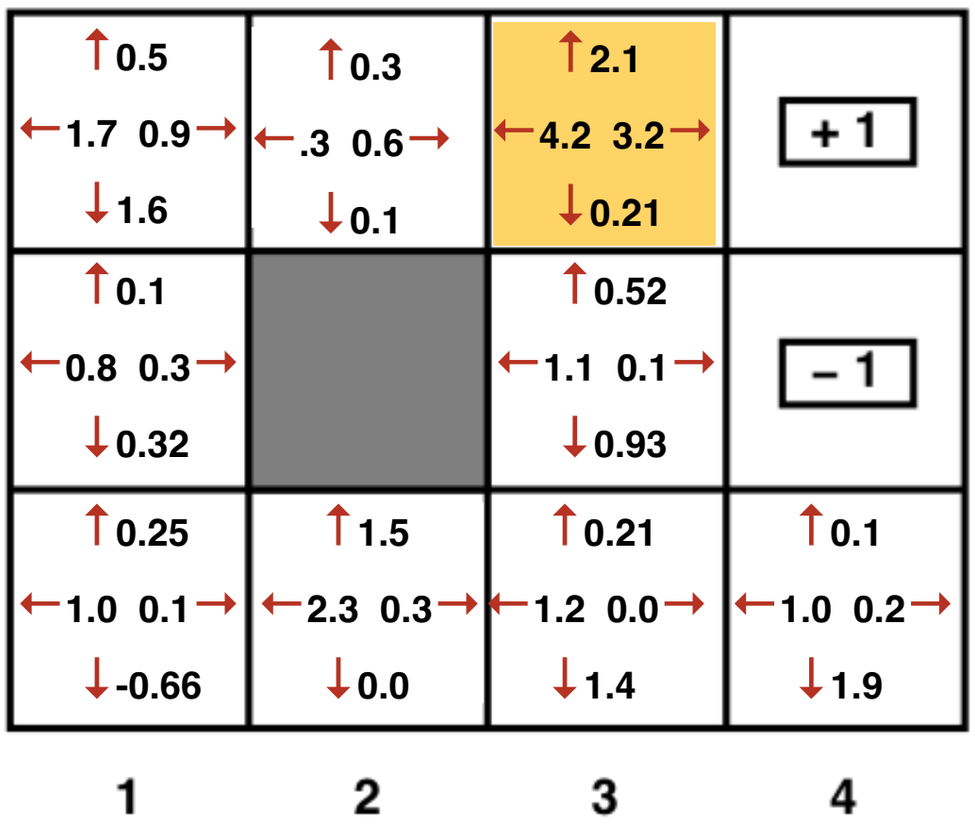
Q((3,3),←) ← (1-α)Q((3,3),←) + α(-0.04 + γ max Q((3,2),a'))

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

assume
α = 0.5
γ = 1.0

Q((3,3),←) ← (1-α)Q((3,3),←) + α(-0.04 + γ max Q((3,2),a'))

← (1-α) 4.2 + α(-0.04 + γ * 0.6)

← 0.5 * 4.2 + 0.5(-0.04 + 1.0 * 0.6) = 2.38

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm

	1	2	3	4
3	↑0.5 ←1.7 0.9→ ↓1.6	↑0.3 ←.3 0.6→ ↓0.1	↑2.1 ←2.38 3.2→ ↓0.21	+1
2	↑0.1 ←0.8 0.3→ ↓0.32		↑0.52 ←1.1 0.1→ ↓0.93	−1
1	↑0.25 ←1.0 0.1→ ↓-0.66	↑1.5 ←2.3 0.3→ ↓0.0	↑0.21 ←1.2 0.0→ ↓1.4	↑0.1 ←1.0 0.2→ ↓1.9

- Assume the agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
    - r = **-0.04**
    - s' = **(3,2)**

assume
α = 0.5
γ = 1.0

Q((3,3),←) ← (1-α)Q((3,3),←) + α(-0.04 + γ max Q((3,2),a'))

← (1-α) 4.2 + α(-0.04 + γ * 0.6)

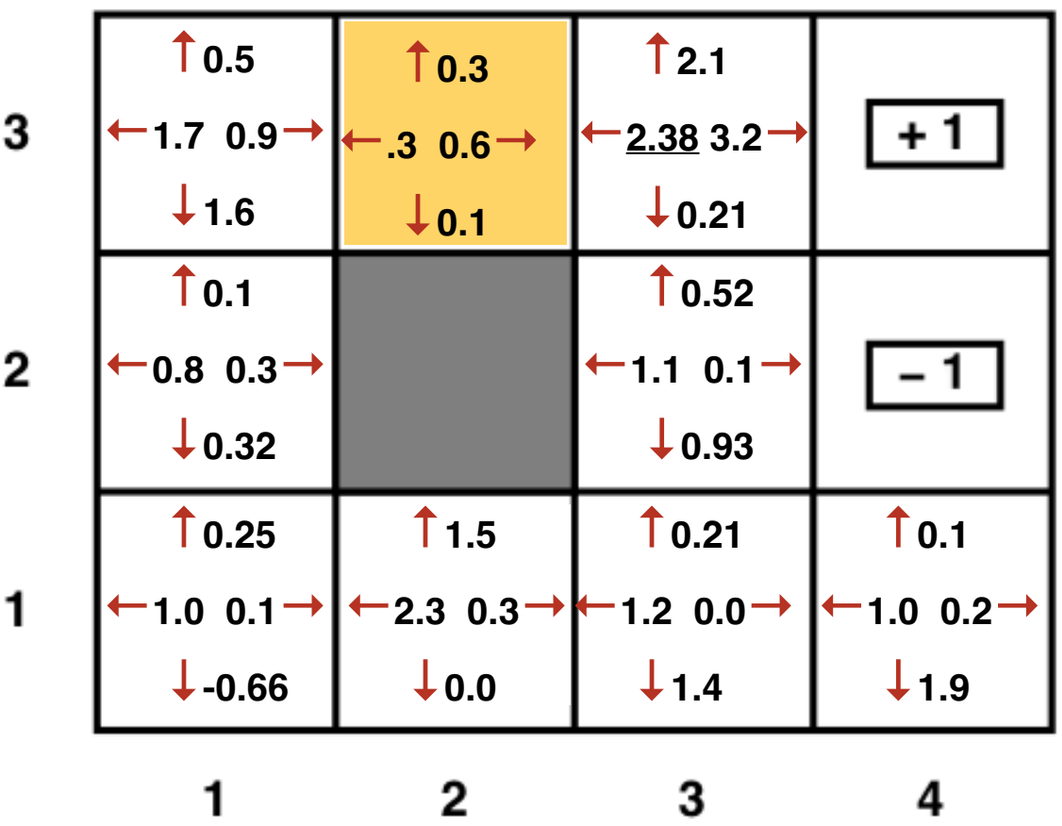
← 0.5 * 4.2 + 0.5(-0.04 + 1.0 * 0.6) = 2.38

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



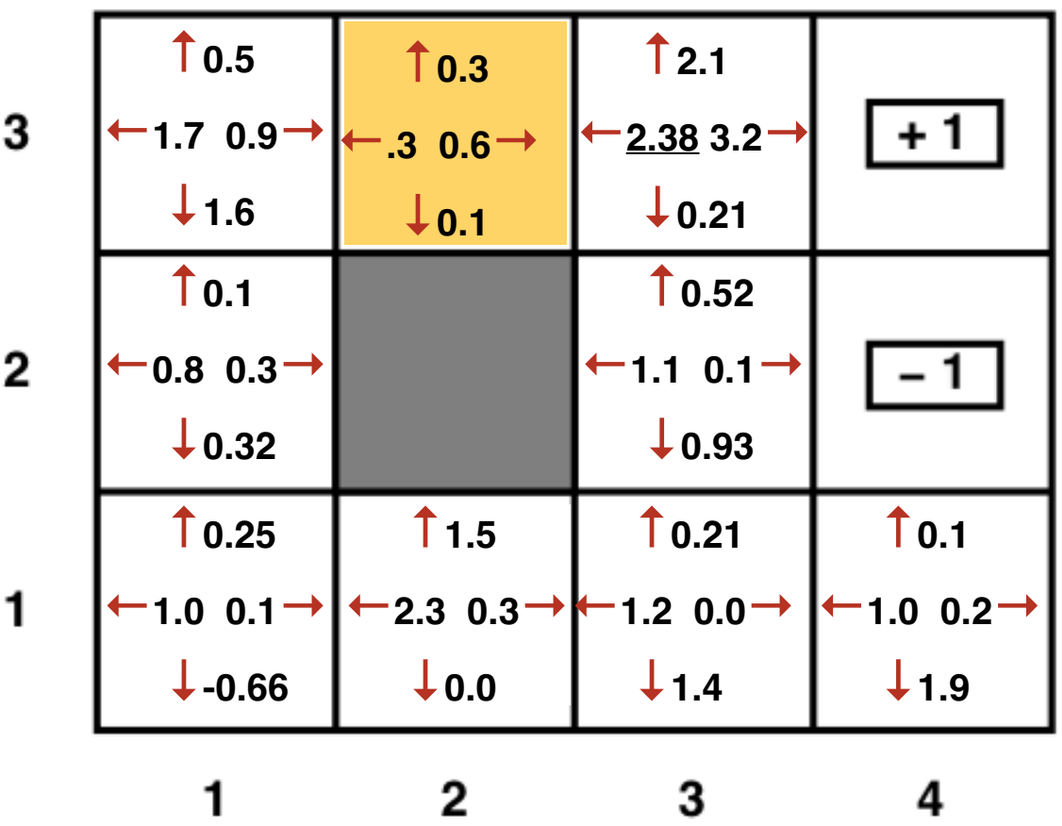
- Agent is now in state (2,3)

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Agent is now in state (2,3)

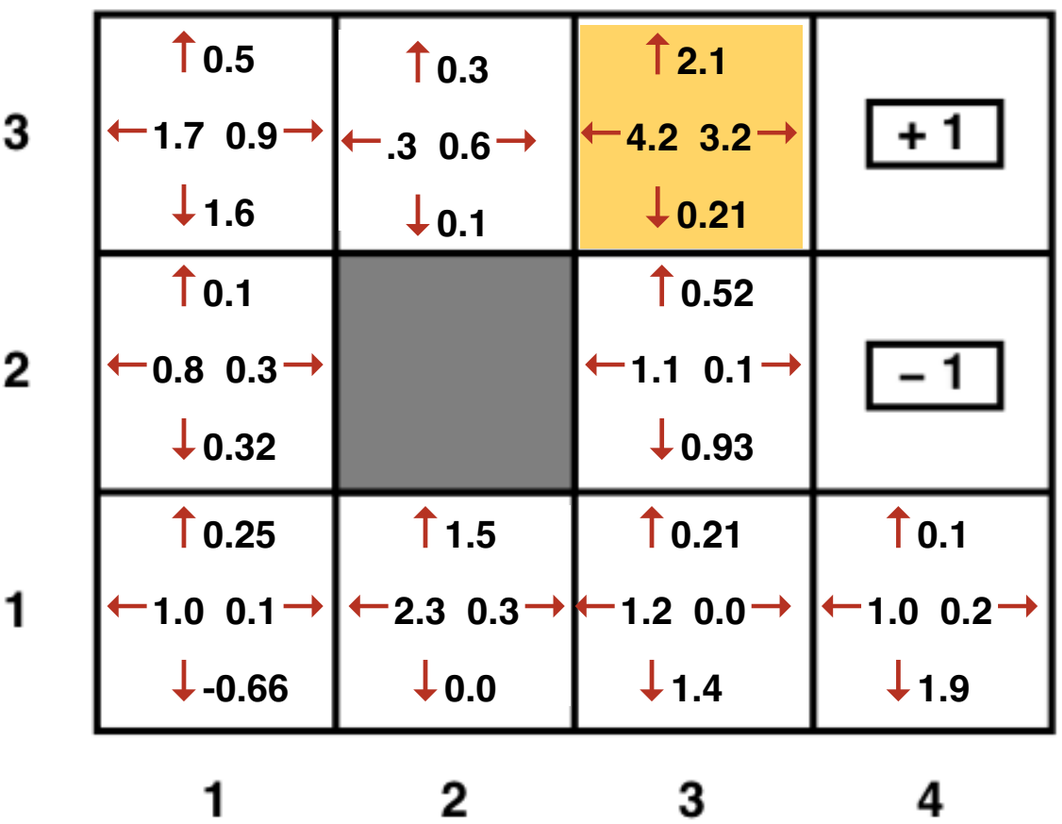
(…)
process repeats

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



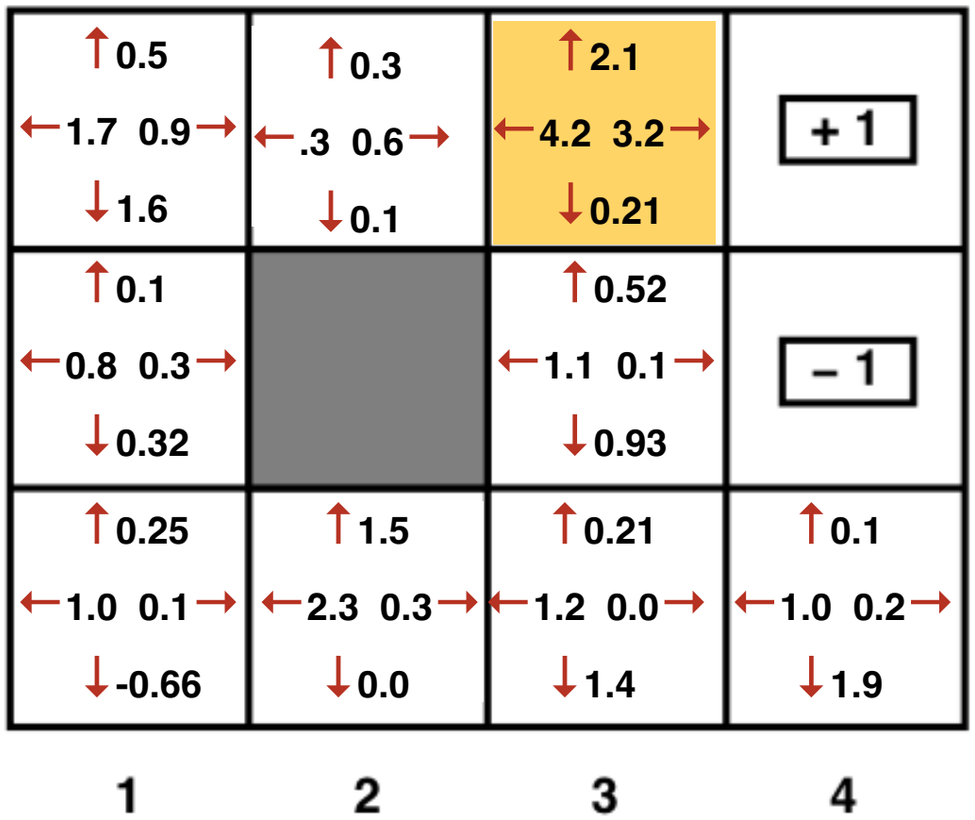
- Problem: when agent was in state (3,3)

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



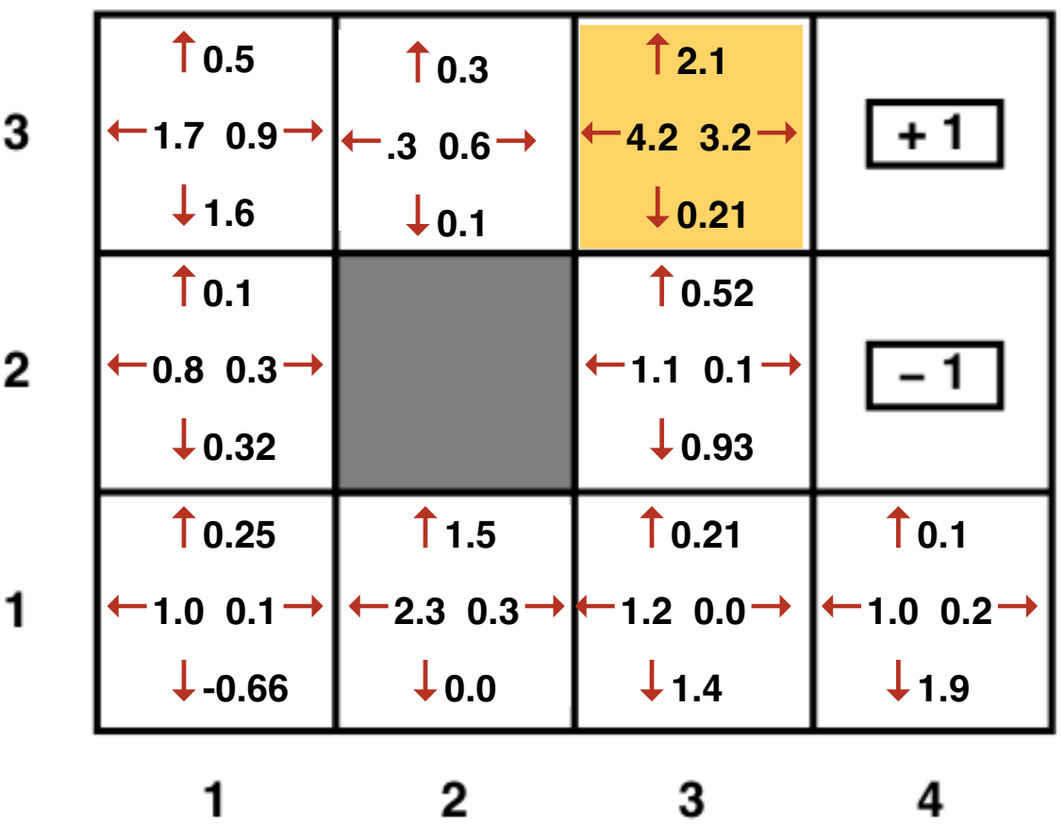
- Problem: when agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm



- Problem: when agent was in state (3,3)
  - Executed action ←
    - s = **(3,3)**
    - a = ←
- Selected action that *seemed* the best (highest Q)
- But during learning, Q's might be wrong!
- Agent doesn't know yet which actions are best
- Should it always execute the one that *seems* the best?

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

Q-Learning algorithm

- How to select an action
  - **Usually pick the one that seems the best, but also try out new actions!**
  - e.g.
    - execute the action with highest Q-value 90% of the time
    - execute a random action 10% of the time

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$

1. Agent is in s, picks action a.
2. Receives reward r and observes next state s'
3. Updates $Q_k \rightarrow Q_{k+1}$

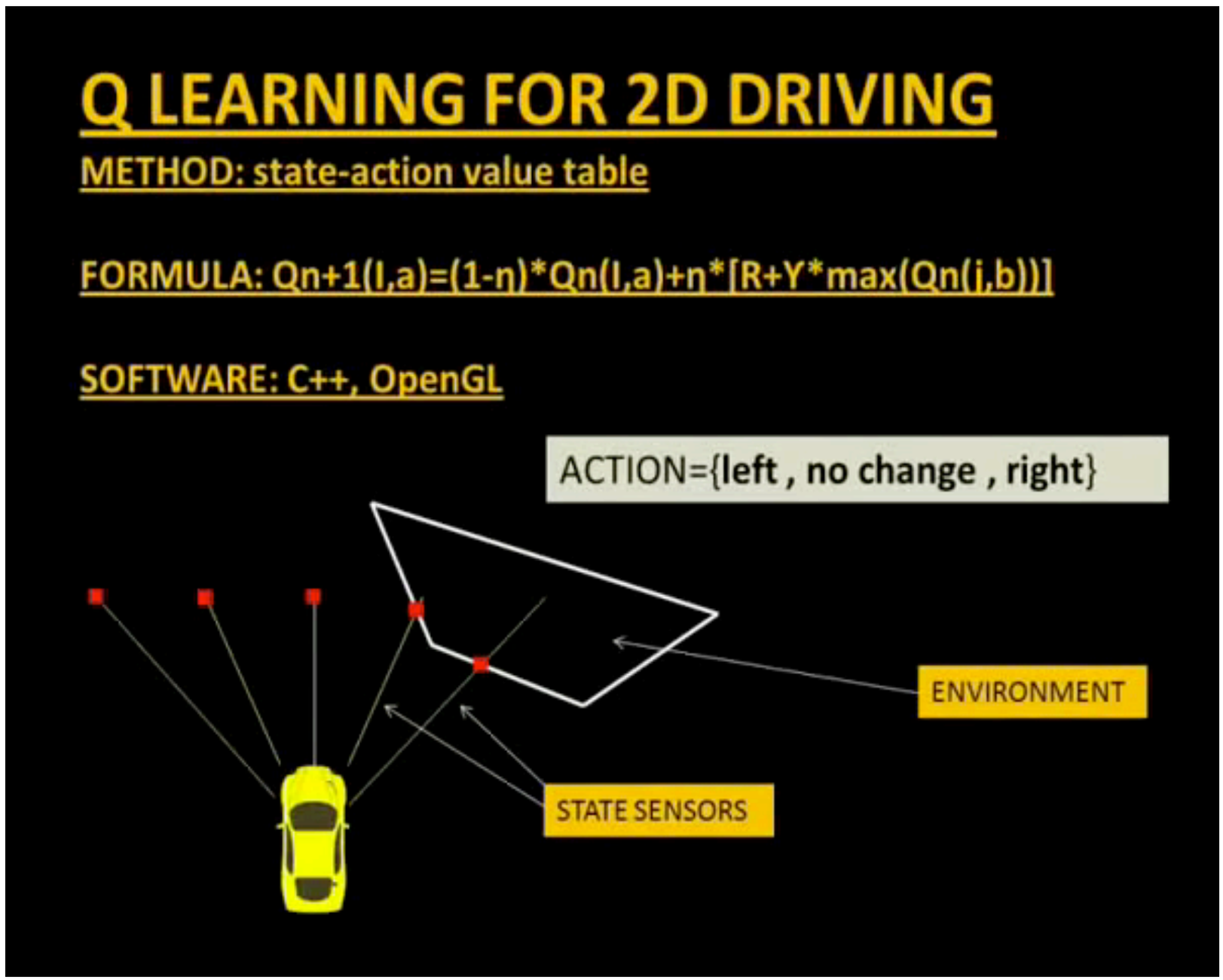
Q-Learning algorithm

- How to select an action
  - **Usually pick the one that seems the best, but also try out new actions!**
  - e.g.
    - execute the action with highest Q-value 90% of the time
    - execute a random action 10% of the time

Initially agent doesn't know environment/outcome of actions → Exploration

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$



https://www.youtube.com/watch?v=zOgSC---rgM

Q-Learning

Q-Learning Algorithm for q^* ***(tabular)***

Initialization: initialize $q(s,a)$ arbitrarily *(except $q(s_\infty, \cdot) = 0$, by definition)*

Algorithm parameter: step size $\alpha \in (0,1]$ and small $\epsilon > 0$

Loop for each episode:

Initialize initial state: $s \sim d_0$

Loop for each step of the episode:

Choose a from s using a policy derived from q // e.g., ϵ-greedy or softmax

Take action a, observe reward r and next state s'

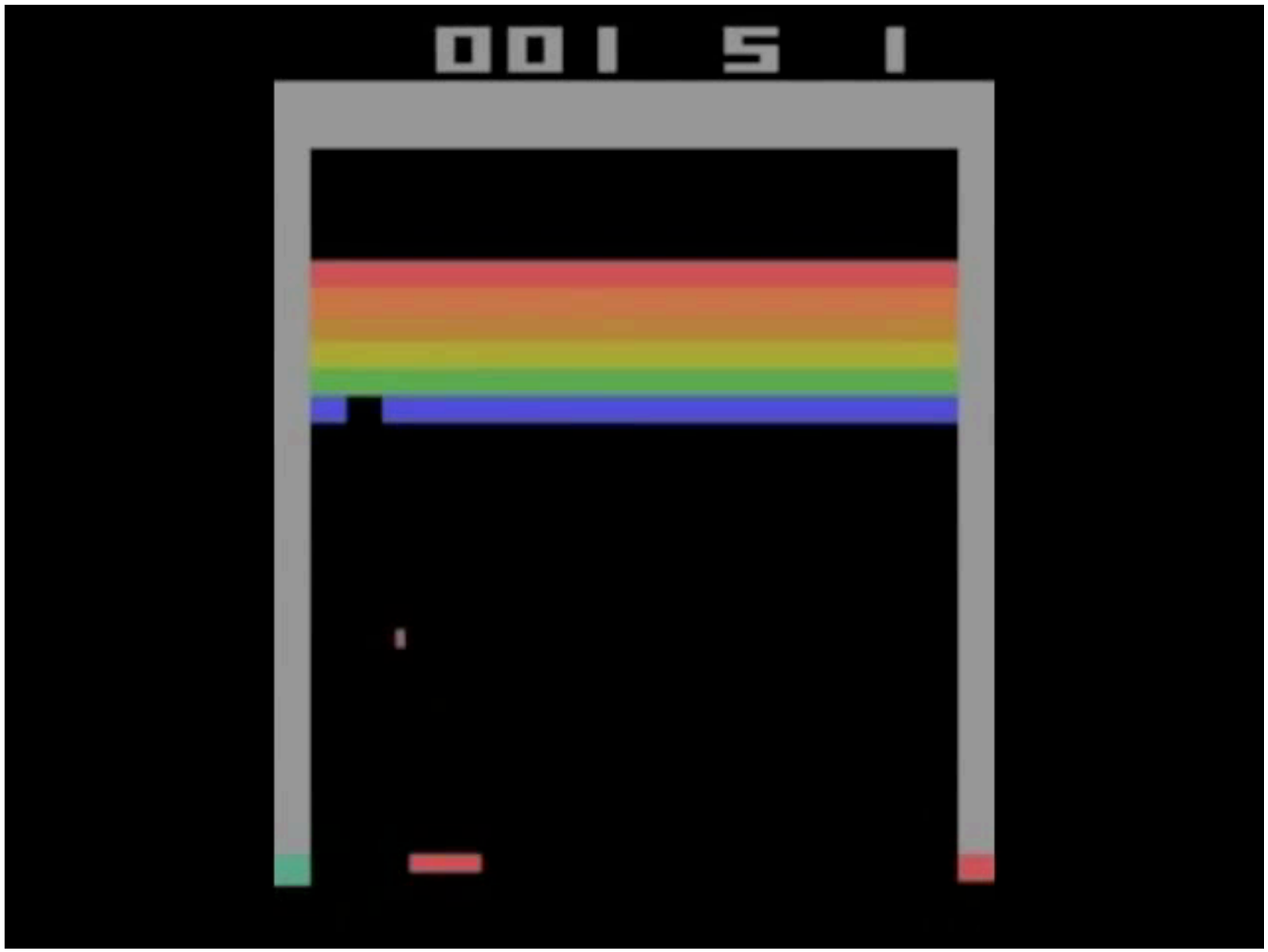
$$q(s,a) \leftarrow q(s,a) + \alpha\left(r + \gamma \max_{a'} q(s',a') - q(s,a)\right)$$

$$s \leftarrow s'$$

until s is the terminal absorbing state

Q-Learning

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left(r + \gamma \max_{a'} Q_k(s',a')\right)$$



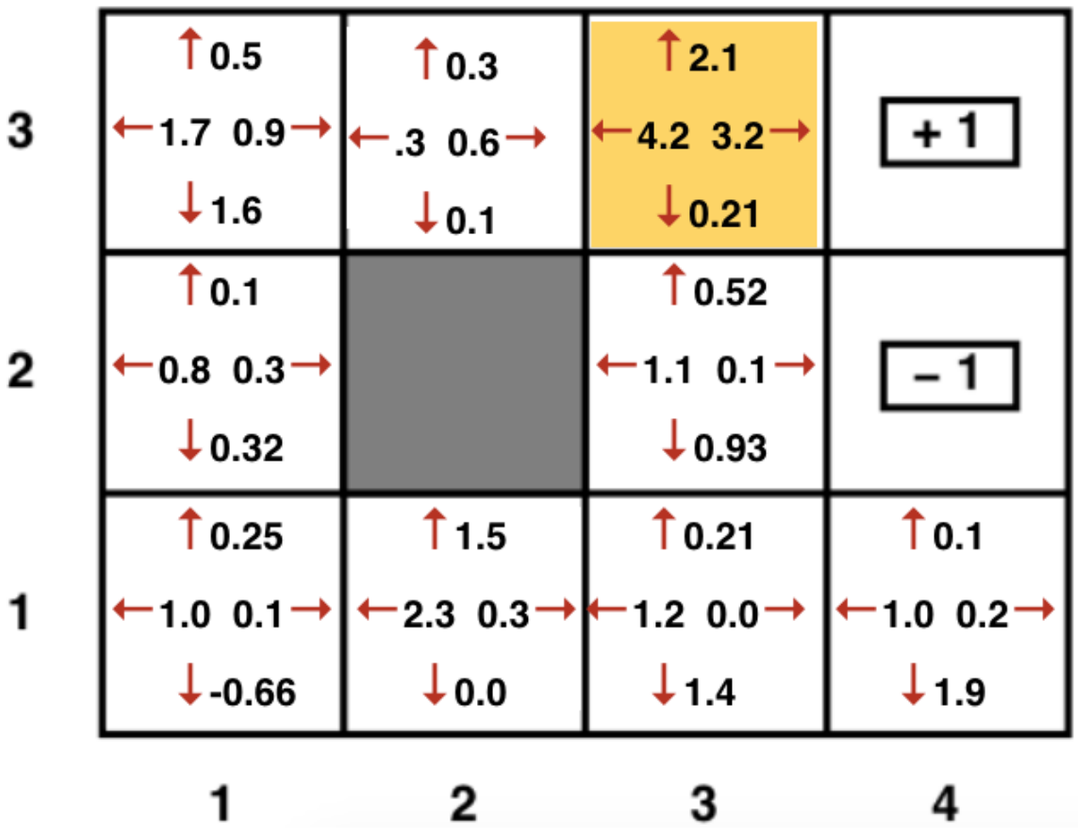
https://www.youtube.com/watch?t=26&v=cjpEIotvwFY

Reinforcement Learning

https://www.youtube.com/watch?v=SH3bADiB7uQ

Reinforcement Learning with (Q-)Value Function Approximation

Representing Q (in the case of discrete states)



Q	↑	←	↓	→
s1=[1,1]	0.25	1.0	-0.66	0.1
s2=[1,2]	1.5	2.3	0.0	0.3
s3=[1,3]	0.21	1.2	1.4	0.0
s4=[1,4]	0.1	1.0	1.9	0.2
s5=[2,1]	0.1	0.8	0.32	0.3
s6=[2,3]	0.52	1.1	0.1	0.93
s7=[3,1]	0.5	1.7	1.6	0.9
s8=[3,2]	0.3	0.3	0.1	0.6
s9=[3,3]	2.1	4.2	0.21	3.2

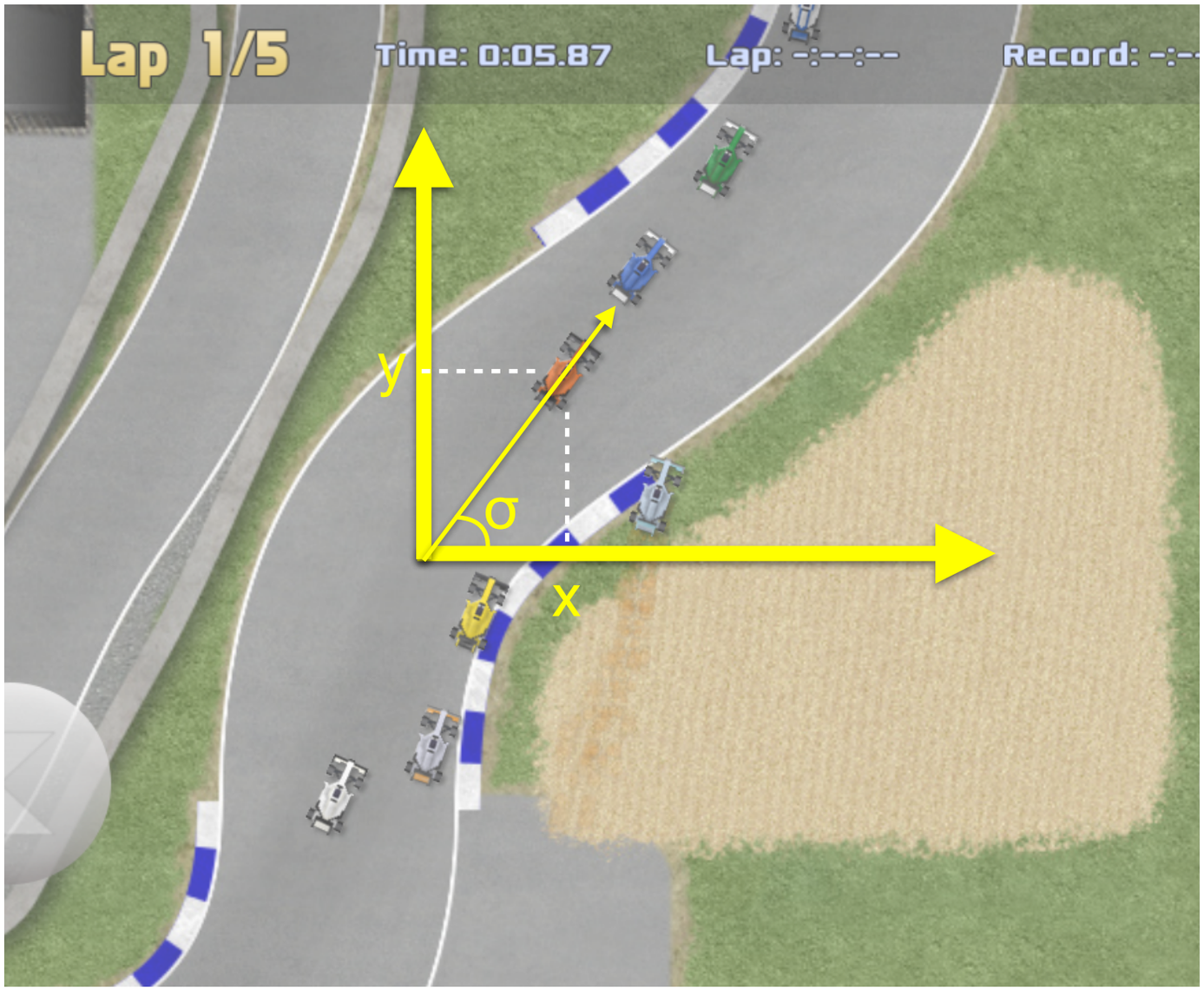
Reinforcement Learning with (Q-)Value Function Approximation

Representing Q (in the case of continuous states)

State?

Reinforcement Learning with (Q-)Value Function Approximation

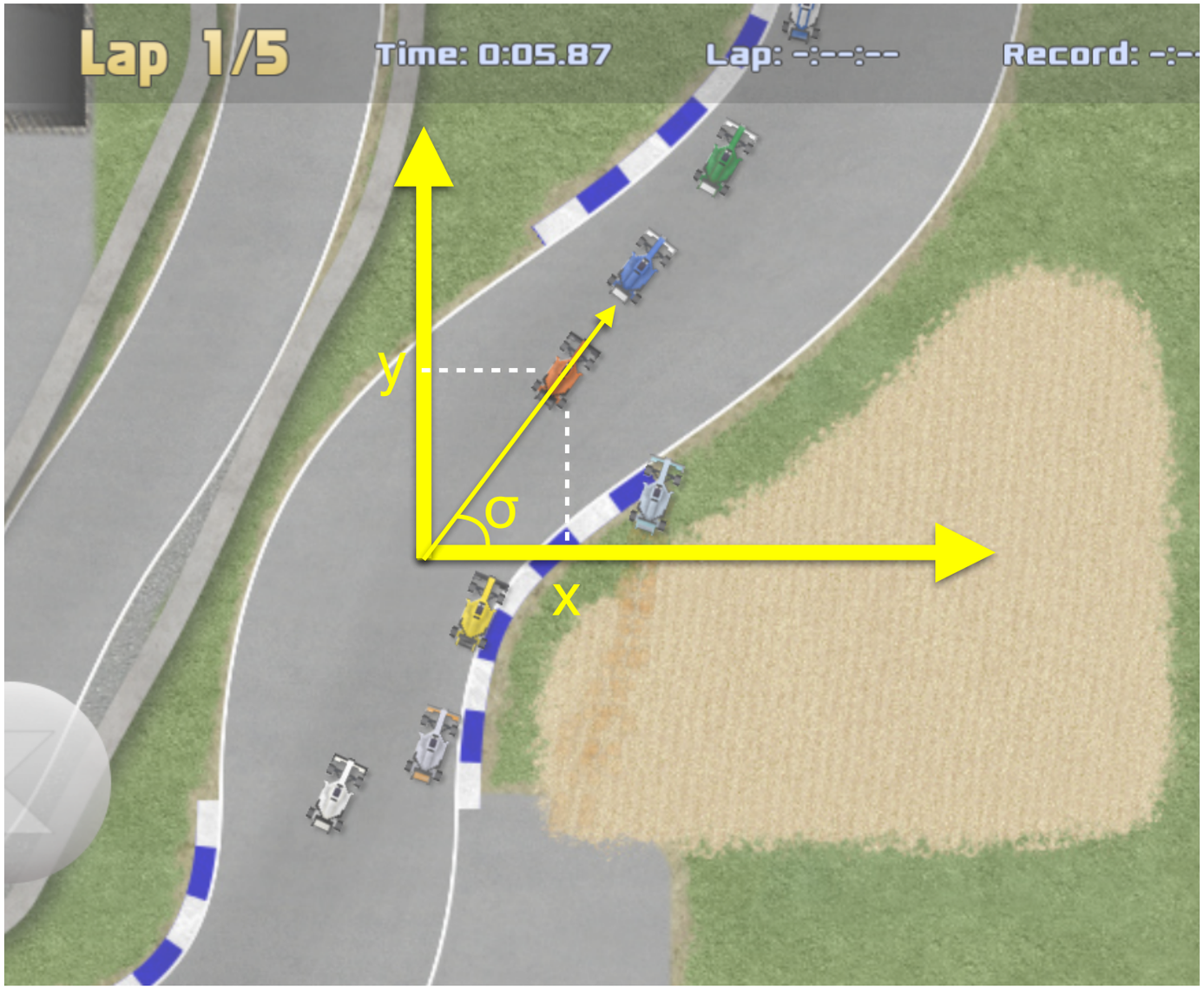
Representing Q (in the case of continuous states)



State: **s=[x,y,σ]**

Reinforcement Learning with (Q-)Value Function Approximation

Representing Q (in the case of continuous states)



State: $s=[x,y,\sigma]$

How many possible values/rows?

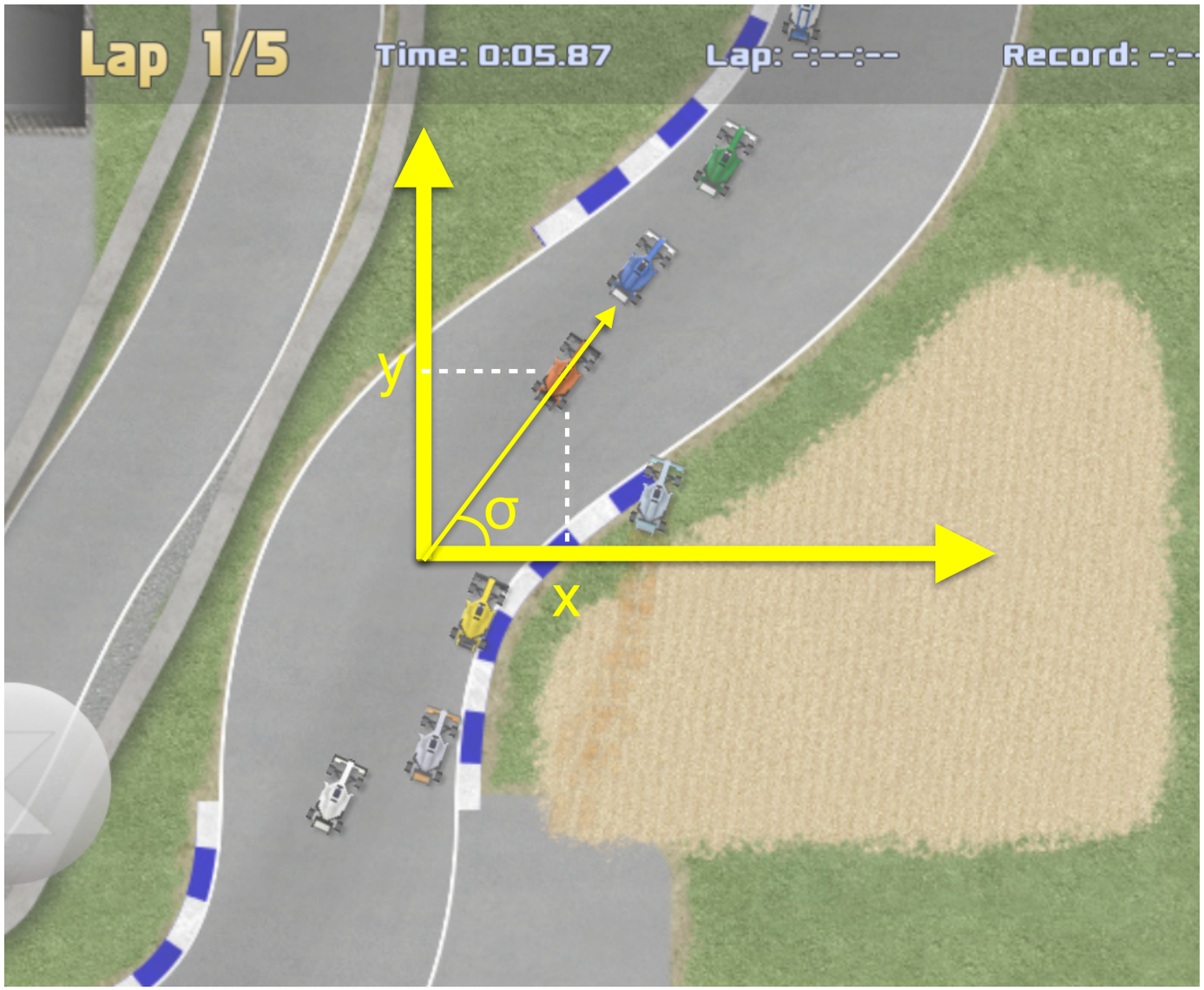
Q	a_1	a_2	…	a_N
$s_1=[0,0,0]$				
$s_2=[0,0,0.5]$				
$s_3=[0,0,1]$				
…				
$s_{1000}=[0,1,0]$				
$s_{1001}=[0,1,0.5]$				
$s_{1002}=[0,1,1]$				
…				

If we discretize x into 100 bins, y into 100 bins, and angle σ in 1 degree increments:

100*100*360 = 3.6 million states

Reinforcement Learning with (Q-)Value Function Approximation

Representing Q (in the case of continuous states)



State: s=[x,y,σ]

How many possible values/rows?

Cannot store Q in a table!

Reinforcement Learning with (Q-)Value Function Approximation

Representing Q (in the case of continuous states)

s=[x,y,σ]

- **Instead of storing Q-values in a table**
- **Use an equation (with learnable weights)**

→ to compute/predict the Q-value of some state/action

$$Q_{\uparrow}(x, y, \sigma) \quad = \quad \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sigma$$

Reinforcement Learning with (Q-)Value Function Approximation

Representing Q (in the case of continuous states)

s=[x,y,σ]

- **Instead of storing Q-values in a table**
- **Use an equation (with learnable weights)**

→ to compute/predict the Q-value of some state/action

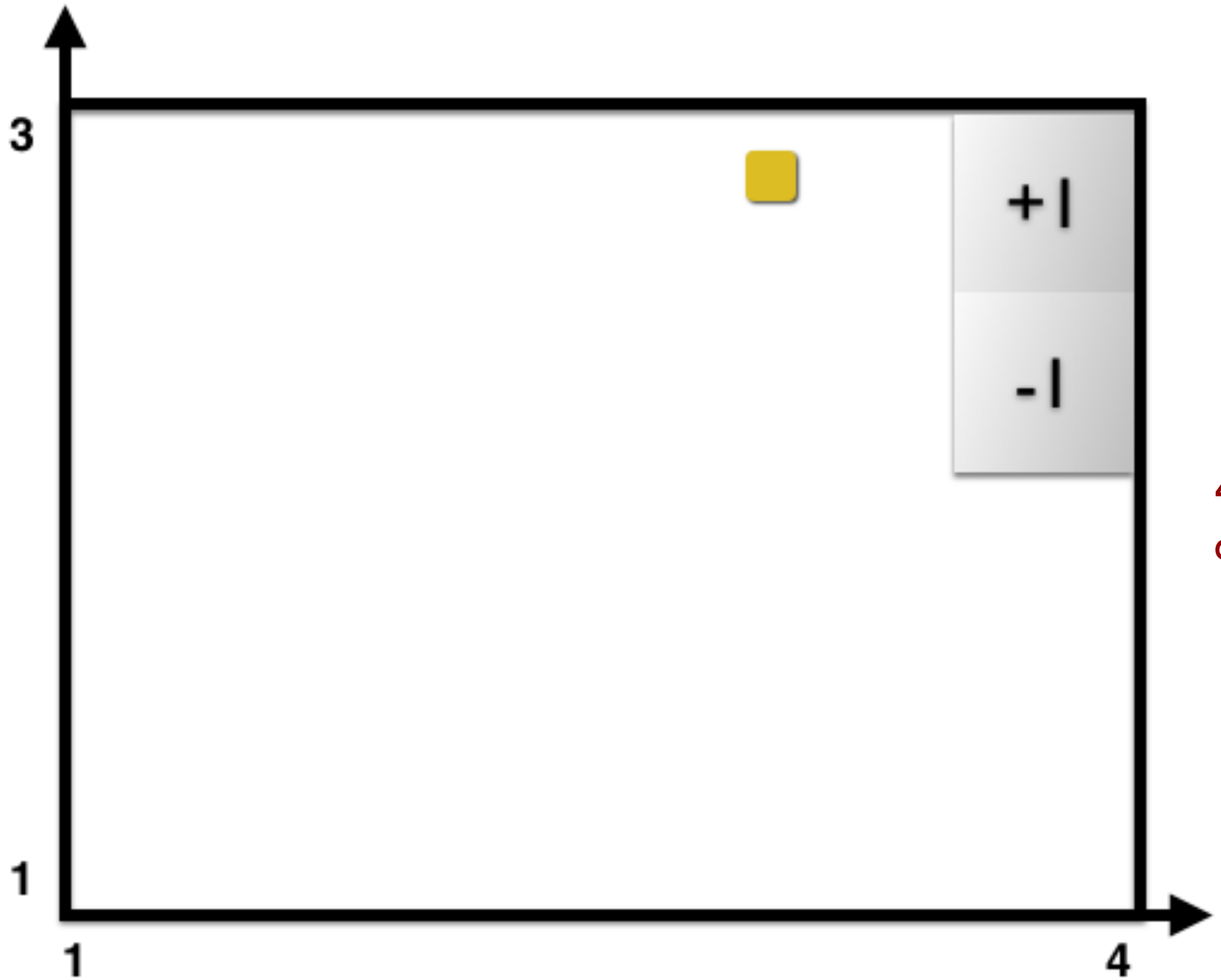
$$Q_{\uparrow}(x, y, \sigma) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sigma$$

some number

estimate of how good it is to
execute action ↑ when in state (x, y, σ)

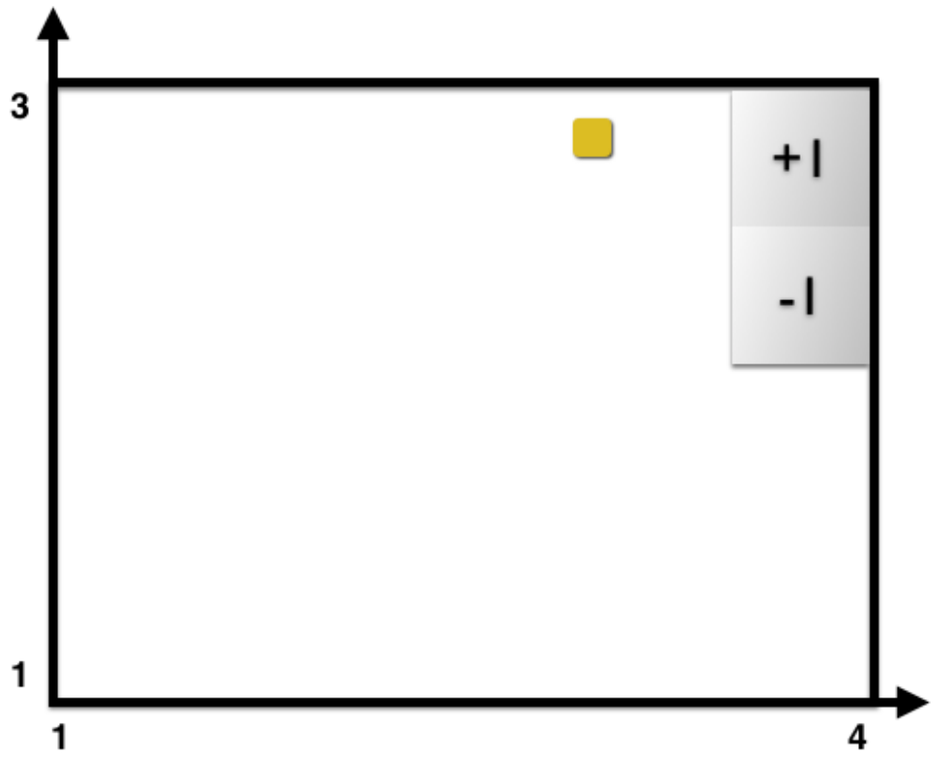
As we adjust weights θ, we obtain different functions

Reinforcement Learning with (Q-)Value Function Approximation



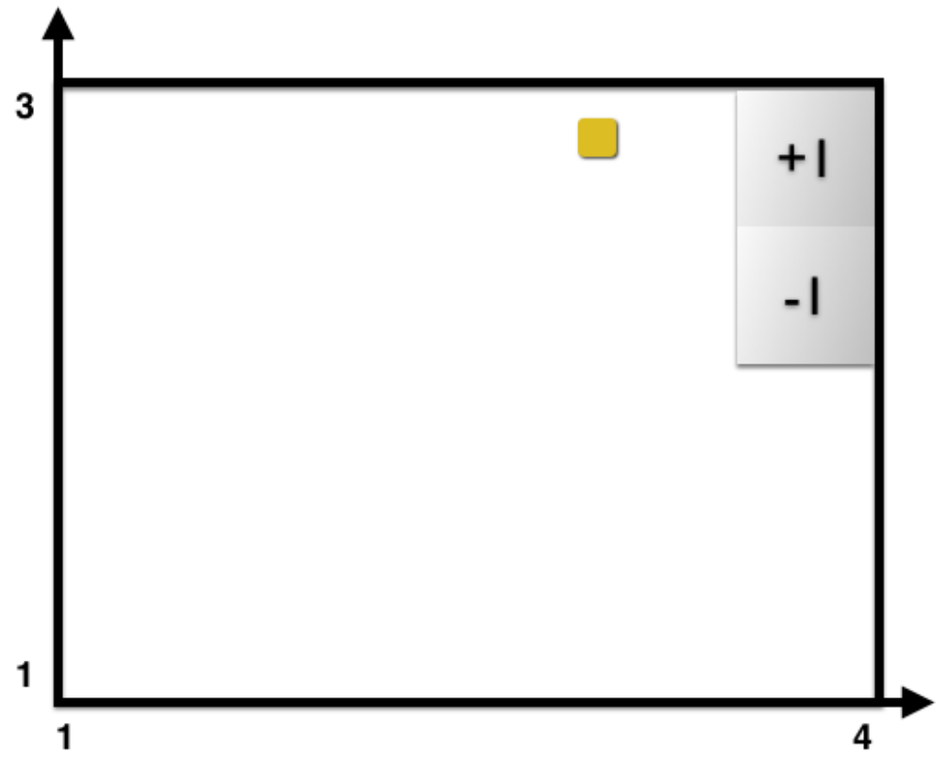
4 actions (↑←↓→)
∞ continuous states

Reinforcement Learning with (Q-)Value Function Approximation



$$Q_{\uparrow}(x, y, \sigma) \quad = \quad \theta_0^{\uparrow} + \theta_1^{\uparrow} x + \theta_2^{\uparrow} y + \theta_3^{\uparrow} \sigma$$

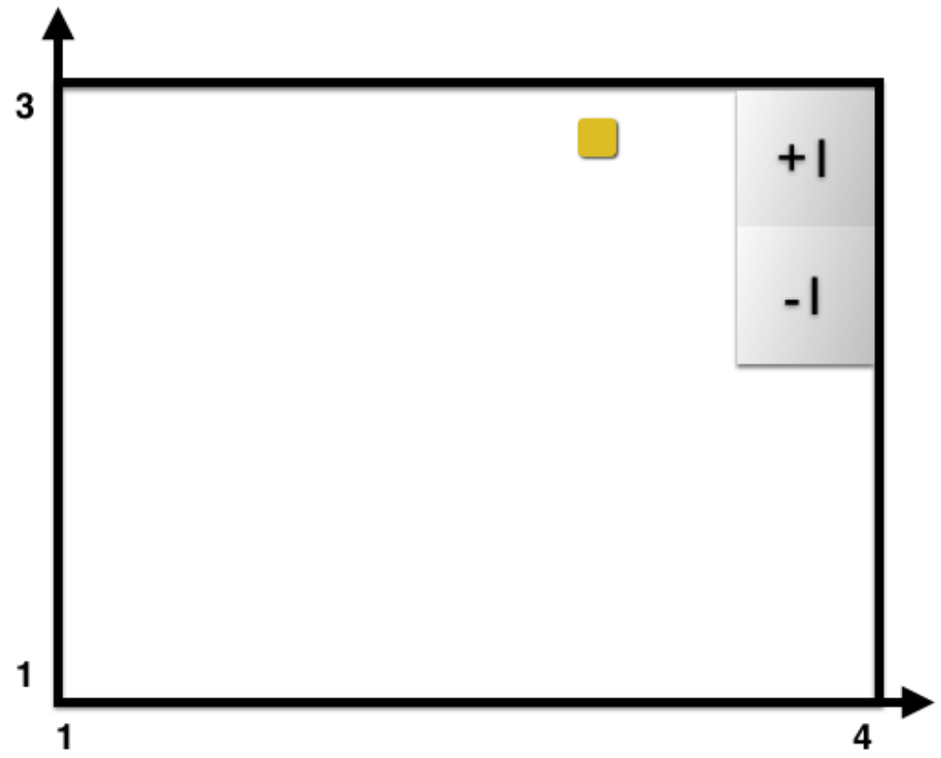
Reinforcement Learning with (Q-)Value Function Approximation



$$Q_{\uparrow}(x, y, \sigma) = \theta_0^{\uparrow} + \theta_1^{\uparrow} x + \theta_2^{\uparrow} y + \theta_3^{\uparrow} \sigma$$

$$Q_{\downarrow}(x, y, \sigma) = \theta_0^{\downarrow} + \theta_1^{\downarrow} x + \theta_2^{\downarrow} y + \theta_3^{\downarrow} \sigma$$

Reinforcement Learning with (Q-)Value Function Approximation



$$
\begin{aligned}
Q_{\uparrow}(x, y, \sigma) &= \theta_0^{\uparrow} + \theta_1^{\uparrow} x + \theta_2^{\uparrow} y + \theta_3^{\uparrow} \sigma \\
Q_{\downarrow}(x, y, \sigma) &= \theta_0^{\downarrow} + \theta_1^{\downarrow} x + \theta_2^{\downarrow} y + \theta_3^{\downarrow} \sigma \\
Q_{\leftarrow}(x, y, \sigma) &= \theta_0^{\leftarrow} + \theta_1^{\leftarrow} x + \theta_2^{\leftarrow} y + \theta_3^{\leftarrow} \sigma \\
Q_{\rightarrow}(x, y, \sigma) &= \theta_0^{\rightarrow} + \theta_1^{\rightarrow} x + \theta_2^{\rightarrow} y + \theta_3^{\rightarrow} \sigma
\end{aligned}
$$

Reinforcement Learning with (Q-)Value Function Approximation

$$
\begin{aligned}
Q_{\uparrow}(x,y,\sigma) &= \theta_0^{\uparrow} + \theta_1^{\uparrow} x + \theta_2^{\uparrow} y + \theta_3^{\uparrow} \sigma \\
Q_{\downarrow}(x,y,\sigma) &= \theta_0^{\downarrow} + \theta_1^{\downarrow} x + \theta_2^{\downarrow} y + \theta_3^{\downarrow} \sigma \\
Q_{\leftarrow}(x,y,\sigma) &= \theta_0^{\leftarrow} + \theta_1^{\leftarrow} x + \theta_2^{\leftarrow} y + \theta_3^{\leftarrow} \sigma \\
Q_{\rightarrow}(x,y,\sigma) &= \theta_0^{\rightarrow} + \theta_1^{\rightarrow} x + \theta_2^{\rightarrow} y + \theta_3^{\rightarrow} \sigma
\end{aligned}
$$

As we adjust weights θ, we obtain different functions

- now we need to store/adjust only 4 numbers θ for each action:

$\theta_0^{\uparrow}, \theta_1^{\uparrow}, \theta_2^{\uparrow}, \theta_3^{\uparrow}$ **to represent the preference Q↑ for action ↑**

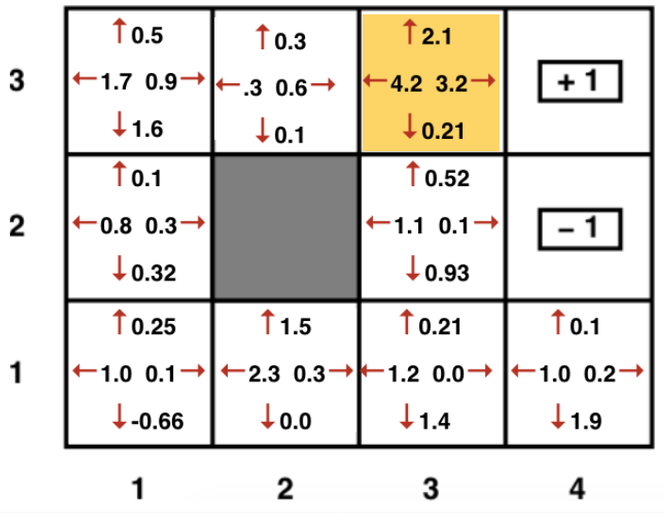
$\theta_0^{\downarrow}, \theta_1^{\downarrow}, \theta_2^{\downarrow}, \theta_3^{\downarrow}$ **to represent the preference Q↓ for action ↓**

$\theta_0^{\leftarrow}, \theta_1^{\leftarrow}, \theta_2^{\leftarrow}, \theta_3^{\leftarrow}$ **to represent the preference Q← for action ←**

$\theta_0^{\rightarrow}, \theta_1^{\rightarrow}, \theta_2^{\rightarrow}, \theta_3^{\rightarrow}$ **to represent the preference Q→ for action →**

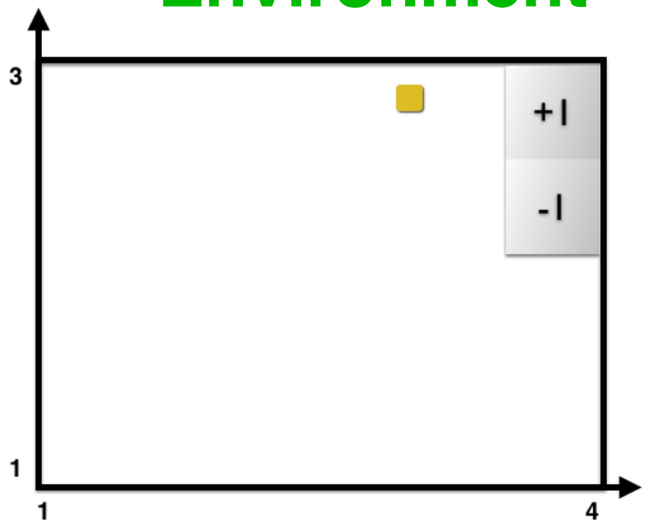
Before (discrete states)

Environment



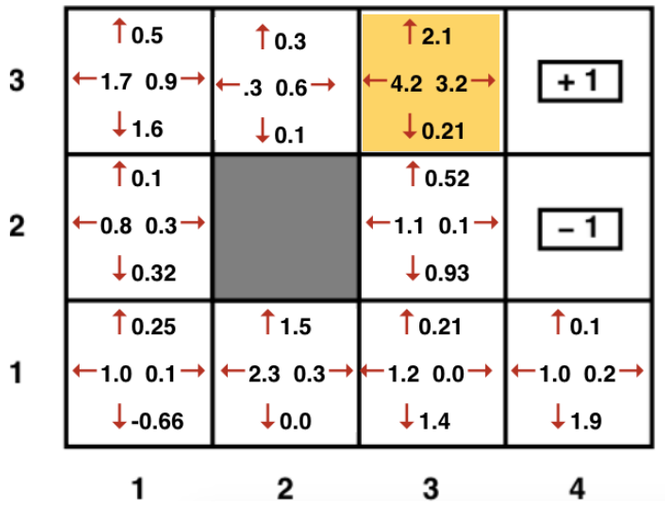
Now (continuous states)

Environment



Before (discrete states)

Environment

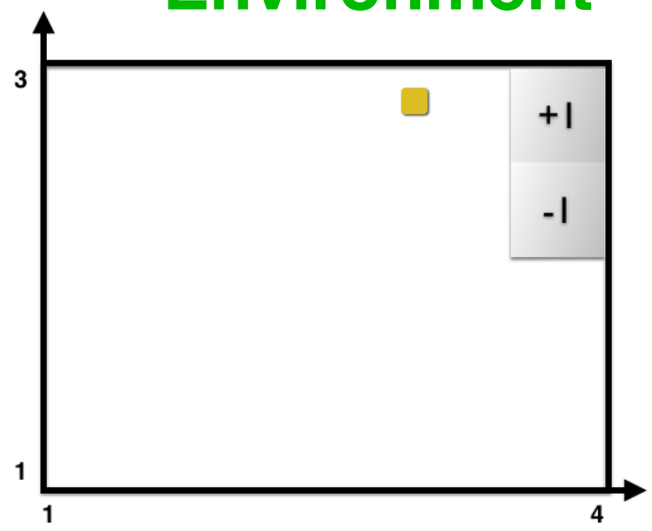


Representing Q (table)

Q	↑	←	↓	→
s1=[1,1]	0.25	1.0	-0.66	0.1
s2=[1,2]	1.5	2.3	0.0	0.3
s3=[1,3]	0.21	1.2	1.4	0.0
s4=[1,4]	0.1	1.0	1.9	0.2
s5=[2,1]	0.1	0.8	0.32	0.3
s6=[2,3]	0.52	1.1	0.1	0.93
s7=[3,1]	0.5	1.7	1.6	0.9
s8=[3,2]	0.3	0.3	0.1	0.6
s9=[3,3]	2.1	4.2	0.21	3.2

Now (continuous states)

Environment



Representing Q (weights)

$$Q_{\uparrow}(x,y,\sigma) = \theta_0^{\uparrow} + \theta_1^{\uparrow} x + \theta_2^{\uparrow} y + \theta_3^{\uparrow} \sigma$$

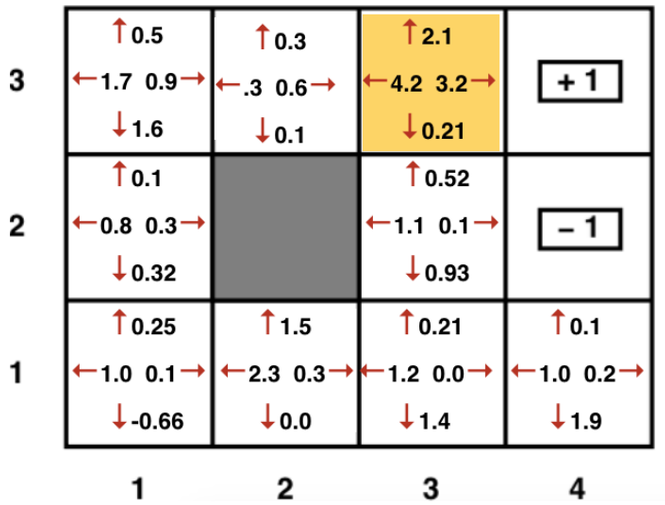
$$Q_{\downarrow}(x,y,\sigma) = \theta_0^{\downarrow} + \theta_1^{\downarrow} x + \theta_2^{\downarrow} y + \theta_3^{\downarrow} \sigma$$

$$Q_{\leftarrow}(x,y,\sigma) = \theta_0^{\leftarrow} + \theta_1^{\leftarrow} x + \theta_2^{\leftarrow} y + \theta_3^{\leftarrow} \sigma$$

$$Q_{\rightarrow}(x,y,\sigma) = \theta_0^{\rightarrow} + \theta_1^{\rightarrow} x + \theta_2^{\rightarrow} y + \theta_3^{\rightarrow} \sigma$$

Before (discrete states)

Environment



Representing Q (table)

Q	↑	←	↓	→
s1=[1,1]	0.25	1.0	-0.66	0.1
s2=[1,2]	1.5	2.3	0.0	0.3
s3=[1,3]	0.21	1.2	1.4	0.0
s4=[1,4]	0.1	1.0	1.9	0.2
s5=[2,1]	0.1	0.8	0.32	0.3
s6=[2,3]	0.52	1.1	0.1	0.93
s7=[3,1]	0.5	1.7	1.6	0.9
s8=[3,2]	0.3	0.3	0.1	0.6
s9=[3,3]	2.1	4.2	0.21	3.2

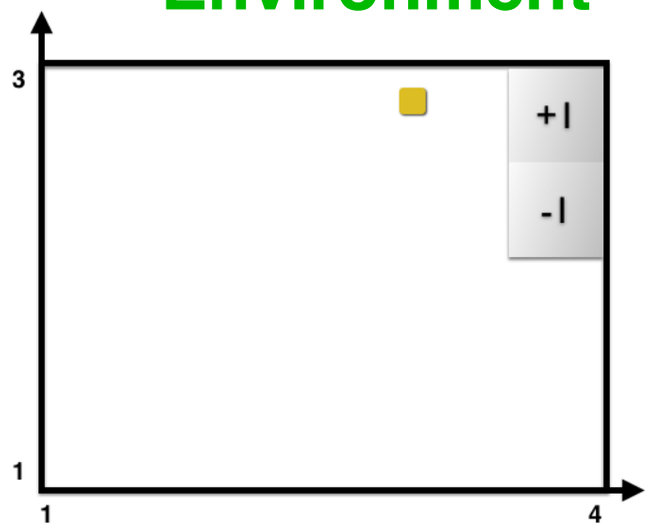
Updating Q (writing in the table)

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\delta_k$$

where $\delta_k = r + \gamma Q_k(s',a') - Q_k(s,a)$

Now (continuous states)

Environment



Representing Q (weights)

$$Q_{\uparrow}(x,y,\sigma) = \theta_0^{\uparrow} + \theta_1^{\uparrow}x + \theta_2^{\uparrow}y + \theta_3^{\uparrow}\sigma$$

$$Q_{\downarrow}(x,y,\sigma) = \theta_0^{\downarrow} + \theta_1^{\downarrow}x + \theta_2^{\downarrow}y + \theta_3^{\downarrow}\sigma$$

$$Q_{\leftarrow}(x,y,\sigma) = \theta_0^{\leftarrow} + \theta_1^{\leftarrow}x + \theta_2^{\leftarrow}y + \theta_3^{\leftarrow}\sigma$$

$$Q_{\rightarrow}(x,y,\sigma) = \theta_0^{\rightarrow} + \theta_1^{\rightarrow}x + \theta_2^{\rightarrow}y + \theta_3^{\rightarrow}\sigma$$

Updating Q (adjusting weights)

$$\theta_i^a \leftarrow \theta_i^a + \alpha\delta f_i(s)$$

where $\delta = r + \gamma Q_{a'}(s') - Q_a(s)$