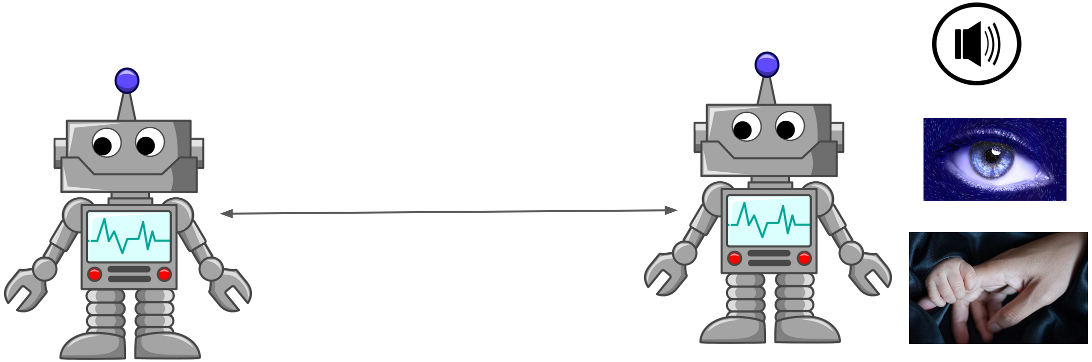
Reinforcement Learning(2)

(How agents can learn to decide?)

04/21/2021

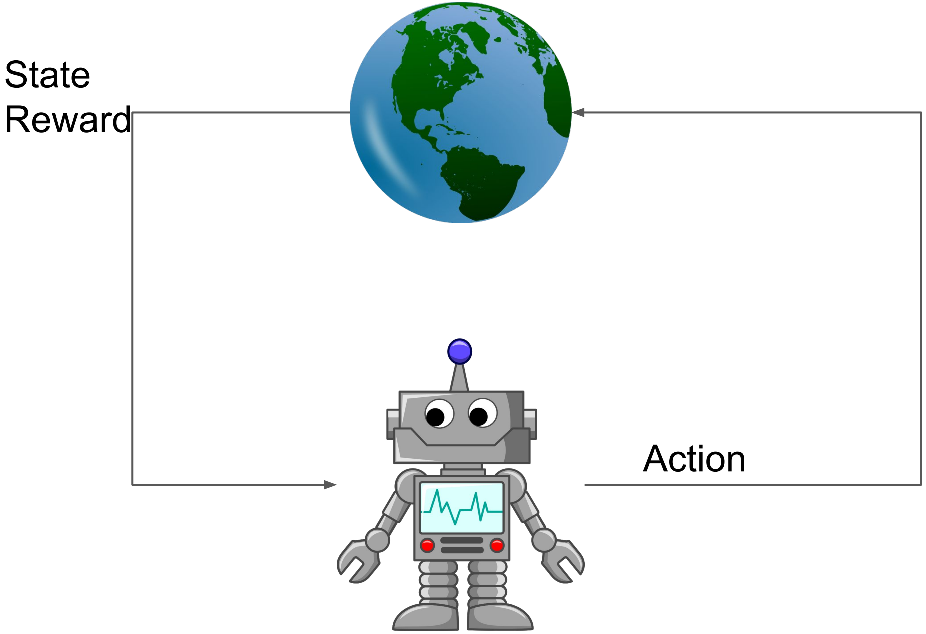
Announcement

- There will be a research seminar in CS department about multi-modal perception and transfer learning in robotics.
- Time: Friday (4/23) 12 pm
- Zoom link: https://binghamton.zoom.us/j/99641908614
- Presenter: Jivko Sinapov



Recap: Reinforcement Learning

A computational approach to learning from interaction



Maximize
long-term
reward

Recap: Important concepts

Reward (R): The signal the environment sends to the agent

- a single number (could be negative).
- The objective is to maximize the total reward it receives over the long run.
- It defines what are the good and bad events for the agent.
- In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain.

Recap: Important concepts

Value V(s): Total amount of reward an agent can expect to accumulate over the future, starting from state **s**.

Policy (π): a policy is a mapping from perceived states of the environment to actions to be taken

Policy could be optimal, or suboptimal

Since we care about long-term utility, we care more about values than a single reward

One more concept

Episodic task vs. continuing task:

If we can break down the agent interaction into finite sequences, then it’s an **episode**.

Each episode terminates in a **terminal** state.

On the other hand, continuing task could be infinite, such as some games.

Let’s come up with an algorithm

We understood the basic concepts in RL.

Now, it’s time to learn about one RL algorithm.

Q-Learning is one of the fundamental RL algorithms.

To understand it,

Let’s see what **q-value** is?

What is q-value (i.e. state-action value)?

Example:

States: s_1 s_2 s_3 s_4

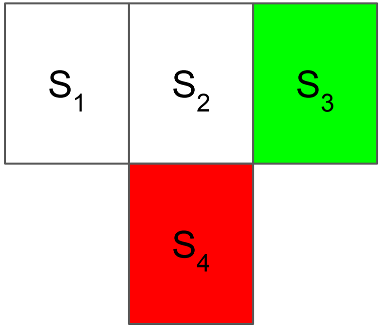
Actions: right, left, up, down

Transitions: 0.1 probability that actions fail

Rewards: Green state has +10 reward

Red state has -10 reward, 0 in other states

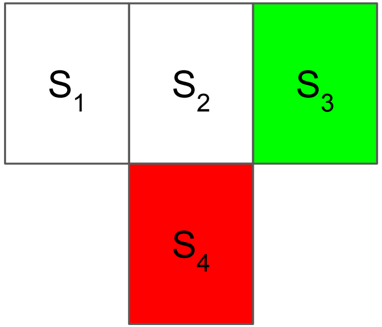
Discount factor: 0.9



	A_1 =right	a_2=left	a_3=up	a_4= down
s_1				
s_2				
s_3				
s_4				

Q-values

$Q(s_1,a_1)$ means that if the robot starts at state s_1 and takes action a_1 and then follows the optimal, Policy, how much reward it can collect

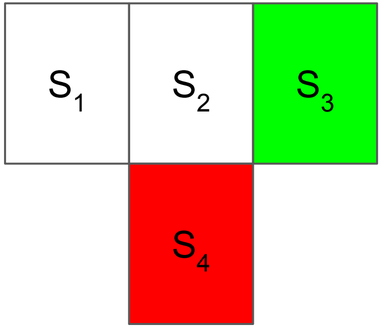


We don't know Q-values, we can find it by interacting with the environment **many** times.

	A_1 =right	a_2=left	a_3=up	a_4= down
s_1				
s_2				
s_3				
s_4				

Q-values

- Q-values are defined in a recursive manner.
- **$Q(s_1,a_1)$** can be determined based on one of the following:
  - **$Q(s_2,a_1)$, $Q(s_2,a_2)$, $Q(s_2,a_3)$, $Q(s_2,a_4)$**



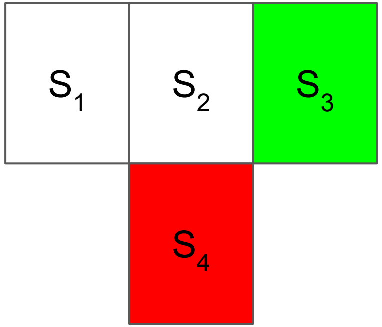
Keep interacting with the environment,

And fill in the table using the equation

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

	A_1 =right	a_2=left	a_3=up	a_4= down
s_1				
s_2				
s_3				
s_4				

Q-values



- Question:
  - If I try many times, what if I get different values for Q(s1,a1) at different times?
- Answer:
  - Why not average them? Or in general why not do a weighted average?

	A_1 =right	a_2=left	a_3=up	a_4= down
s_4				

Source: https://en.wikipedia.org/wiki/Q-learning

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Updating Q-values

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

Source: https://en.wikipedia.org/wiki/Q-learning

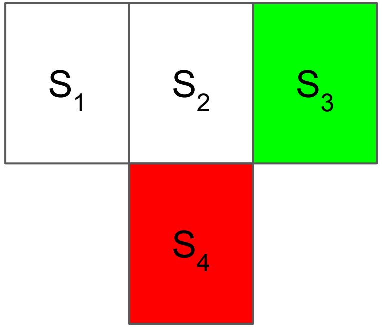
$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}\right)}^{\text{learned value}}$$

old value

New value

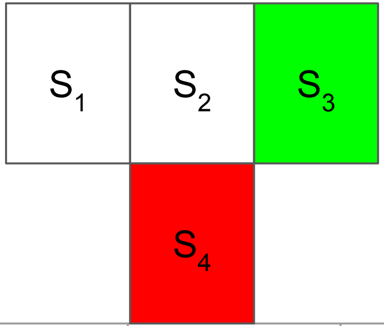
Demo (t=0)

- Initialize the table with 0s



	A_1 =right	a_2=left	a_3=up	a_4= down
s_1	0	0	0	0
s_2	0	0	0	0
s_3	0			
s_4	0			

Demo (t=1)



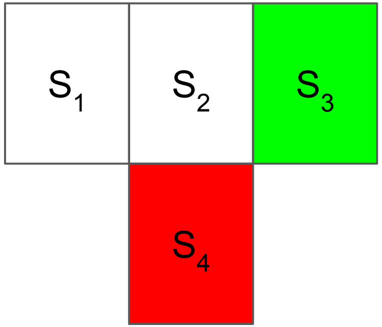
- **Episode:** S_1,right, s_2,right, s_3

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

	A_1 =right	a_2=left	a_3=up	a_4= down
s_1	(8.1 +0)/2	0	0	0
s_2	(0+9)/2	0	0	0
s_3	(+10+0)/2=5			
s_4	0			

Demo (t=2)

- **Episode:** s_1,right, s_2,down, s_4

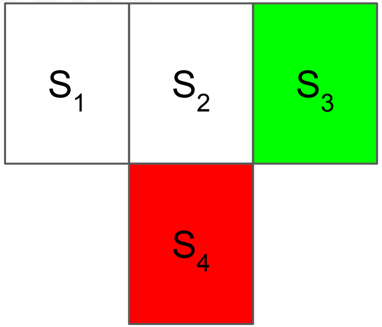


$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

	A_1 =right	a_2=left	a_3=up	a_4= down
s_1	(4.05+0.9*4.5)/2	0	0	0
s_2	4.5	0	0	-4.5
s_3	+5			
s_4	(0-10)=-5			

Demo (t=3) table to be updated

- **Episode:** s_1,right, s_2,left, s_1,right, $s_{2,}$right, s_3

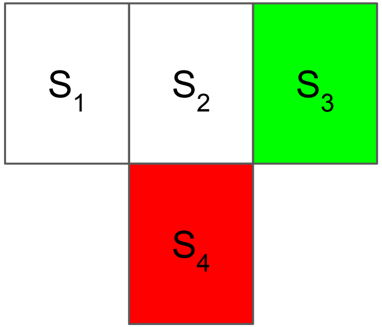


$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

	A_1 =right	a_2=left	a_3=up	a_4= down
s_1	4.05	0	0	-4.05
s_2	4.5	0.9*4.05	0	-4.5
s_3	(5 +10)= 7.5			
s_4	-5			

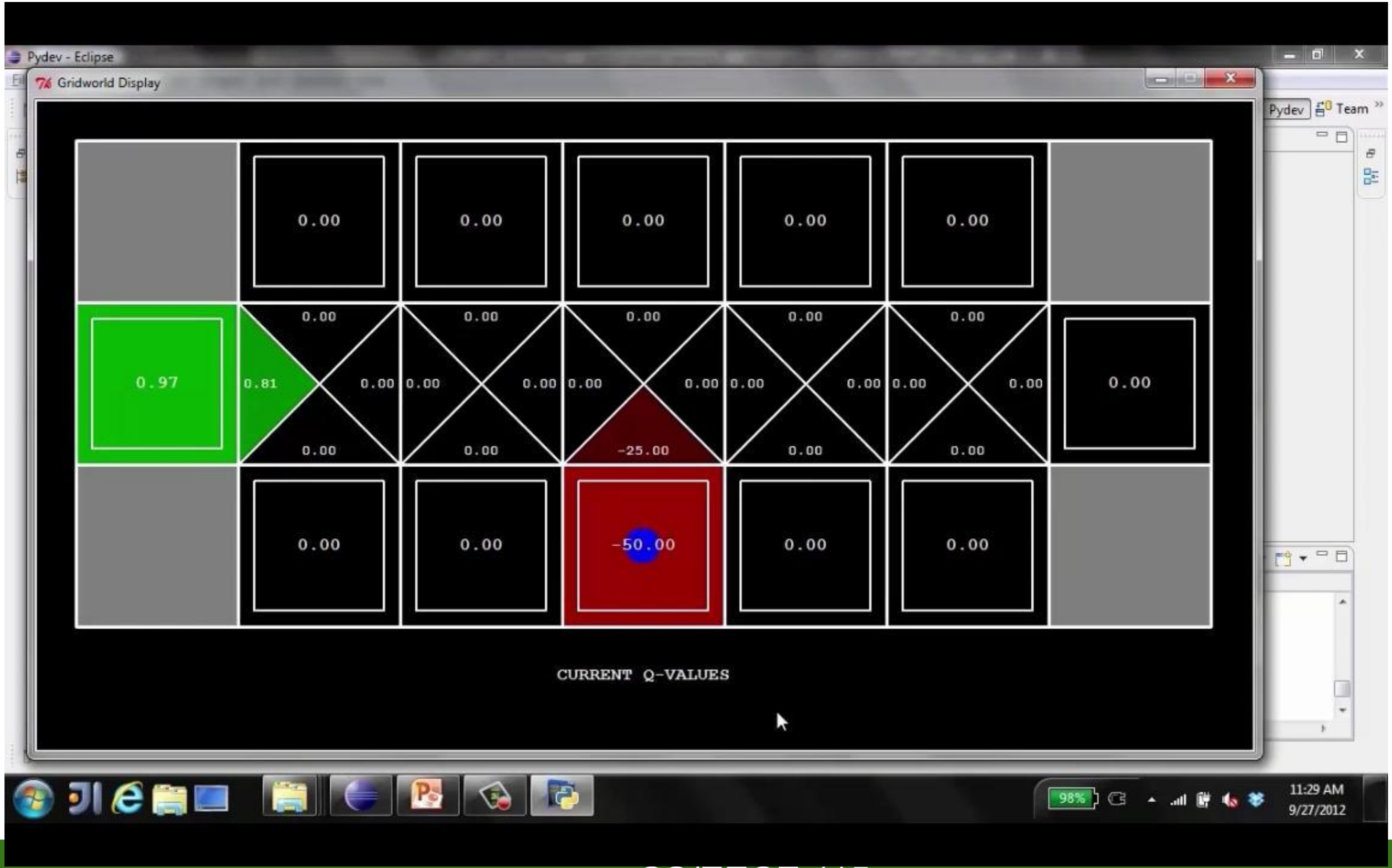
Demo (t=1000)

- After many (say 1000) interactions,
- The table is not being updated anymore, therefore it's converged
- Maximum value in each row, specifies the policy



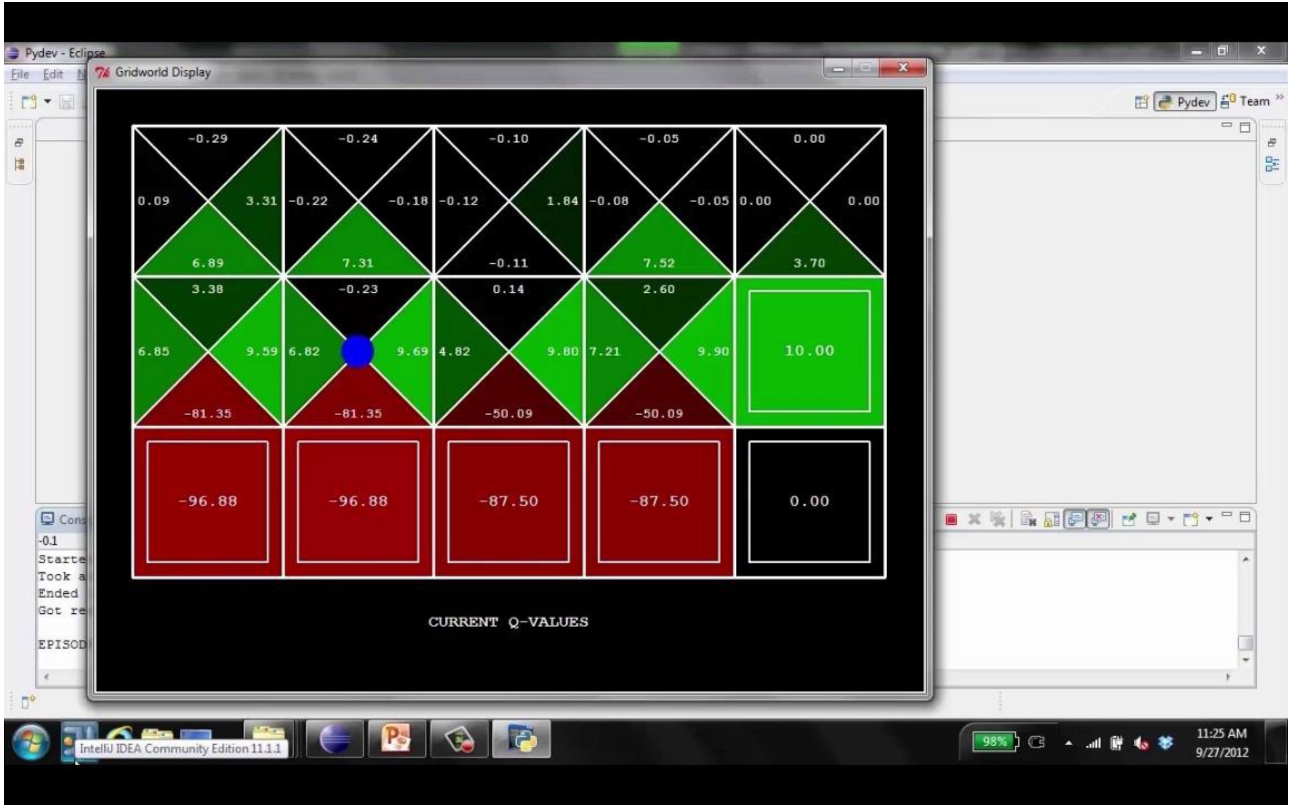
	A_1 =right	a_2=left	a_3=up	a_4= down
s_1	**8.1**	0.21	5.4	-8.1
s_2	**9**	0.73	-0.2	-9
s_3	+10			
s_4	-10			

Demo video



Pydev - Eclipse
Gridworld Display
0.00
0.00
0.00
0.00
0.00
0.97
0.81
-25.00
-50.00
CURRENT Q-VALUES
11:29 AM
9/27/2012

Demo video


Gridworld Display
-0.29
-0.24
-0.10
-0.05
0.00
0.09
3.31
-0.22
-0.18
-0.12
1.84
-0.08
-0.05
0.00
0.00
6.89
7.31
-0.11
7.52
3.70
3.38
-0.23
0.14
2.60
6.85
9.59
6.82
9.69
4.82
9.80
7.21
9.90
10.00
-81.35
-81.35
-50.09
-50.09
-96.88
-96.88
-87.50
-87.50
0.00
CURRENT Q-VALUES

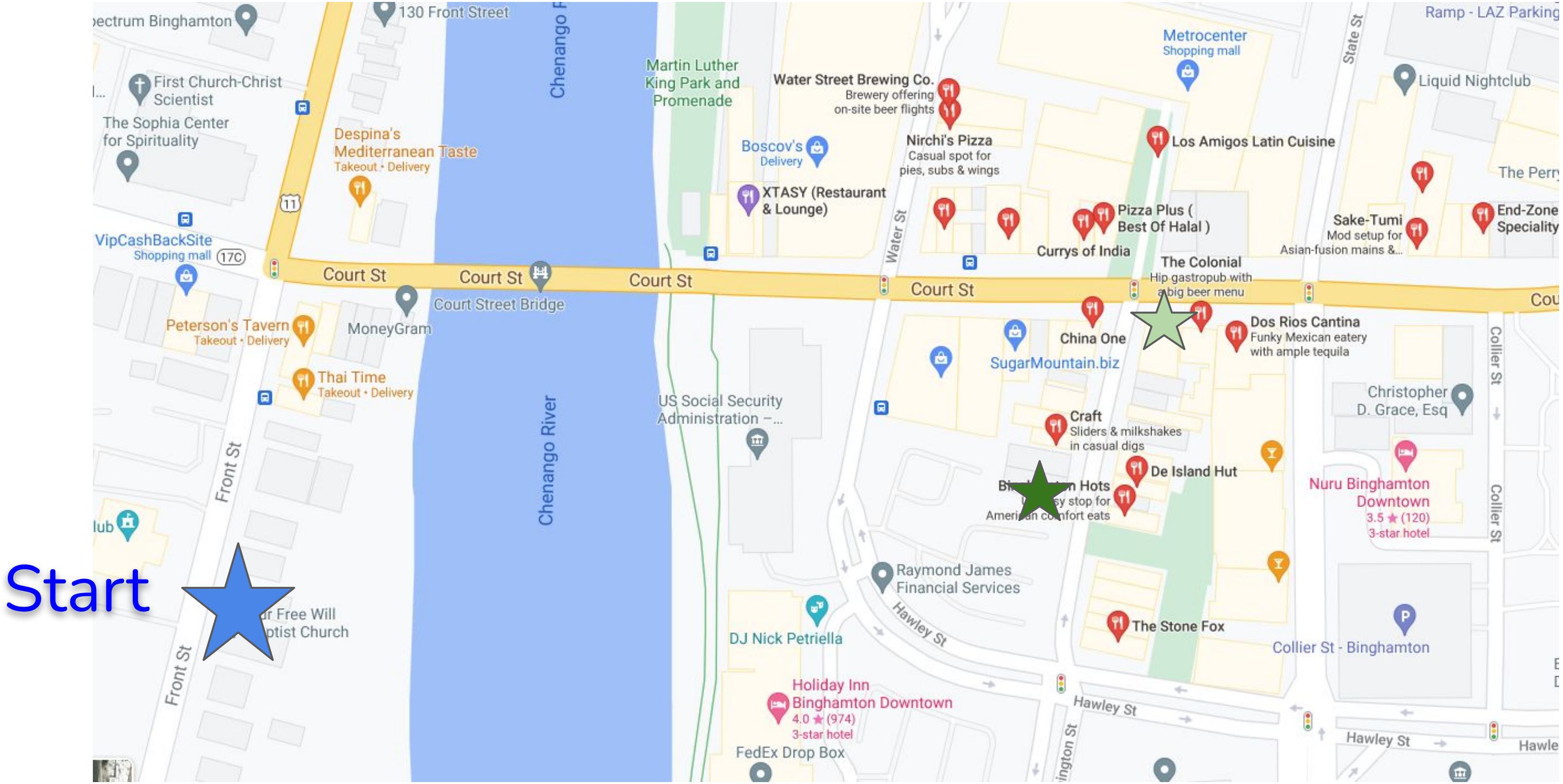
q-learning

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

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

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

Exploration vs. exploitation



Exploration vs. exploitation

One idea:
Initially **explore**
Towards the end, **exploit**

Other RL types

There are various RL methods:

Model-free:

- **Q-learning**
- SARSA

Model-based:

- Dyna-Q

Also, RL methods are categorized into:

- Policy-based methods
- Value based methods
- Actor-Critic

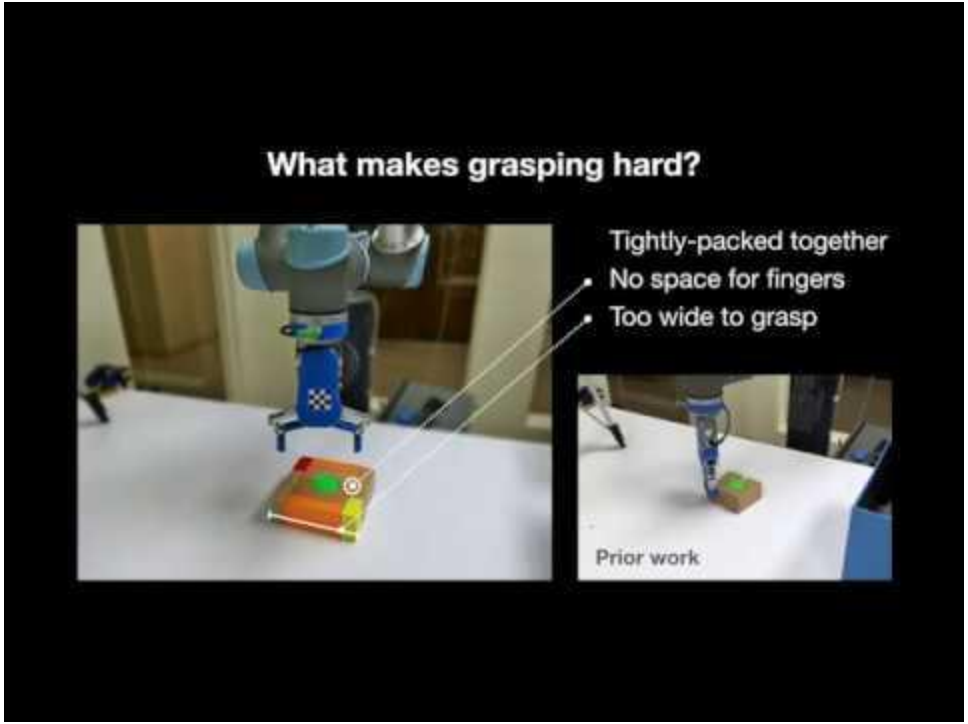
Going deep

- Q(s,a) table can become huge.
- In that case, building a table is not feasible
- Instead, let’s use a neural network

DRL in Games

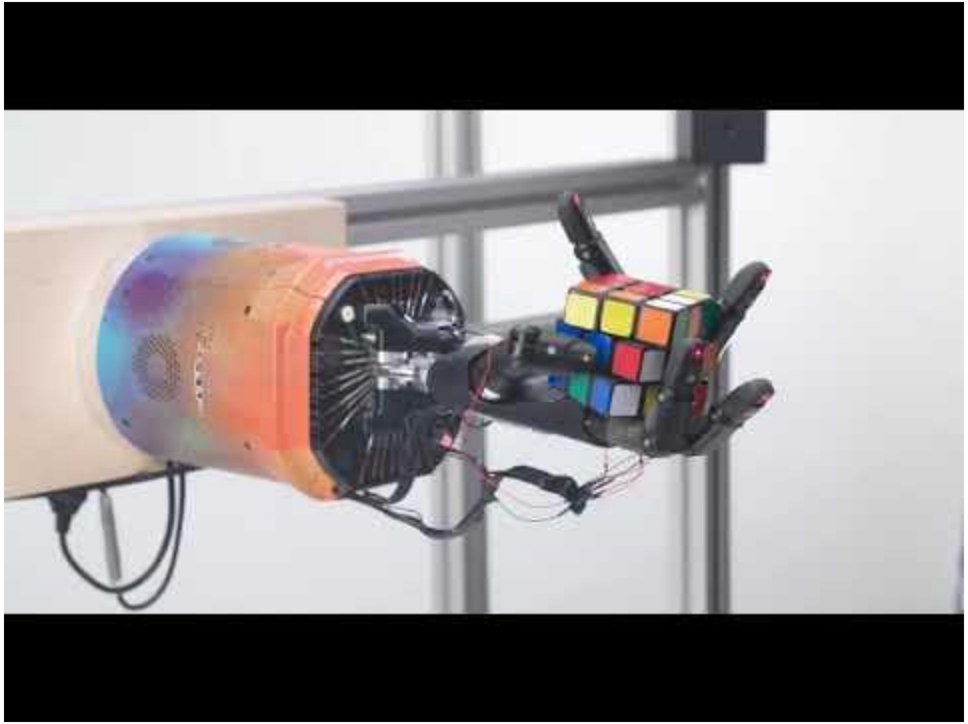
[Mnih et al.,2018]

DRL in Robotics



[Zeng et al.,2018]

DRL in Robotics



[Akkaya et al.,2019]

Take home message

- A lot of complex tasks can be solved using RL when we design good reward functions
- The dilemma of exploration vs. exploitation

References

Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

Zeng, Andy, et al. "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning." 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018.

Akkaya, Ilge, et al. "Solving rubik's cube with a robot hand." arXiv preprint arXiv:1910.07113 (2019).

Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).