

REINFORCEMENT LEARNING 2

Lara J. Martin (she/they)

TA: Aydin Ayanzadeh (he)

10/26/2023

CMSC 671

By the end of class today, you will be able to:

- Compare and contrast the effects of exploration and exploitation
- Access a basic understanding of the RL algorithm
- Extend Q-tables to real-world scenarios

IMPORTANT CLASS STUFF

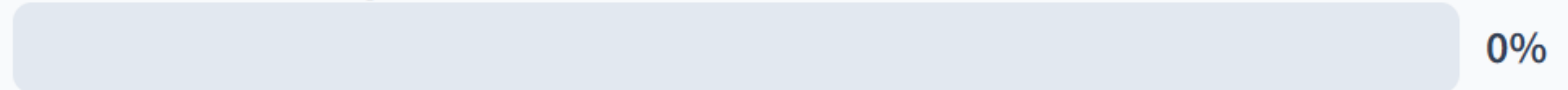
- Midterm is 11/2
 - Next lecture will be review (+ maybe start of probability)
 - Paper & pencil or on laptop – Respondus
 - “Possible Topics for Midterm” list on Blackboard
- Expect an **email from CATME** to form teams for the final project
 - Should take you 5 minutes
 - Due 11/3
- HW 3 is released
 - Similar to HW 1 (so should be easier)
 - Due 11/14



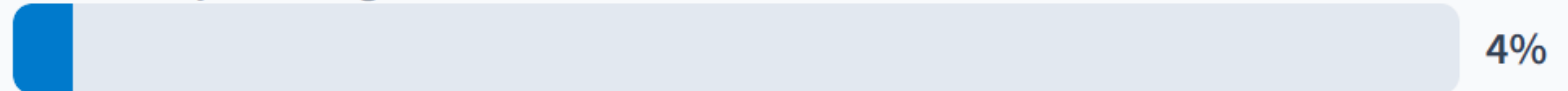
RECAP

What is a Q-value?

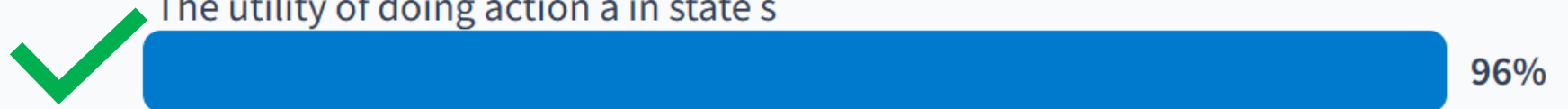
The utility of doing action a



The utility of being in state s



The utility of doing action a in state s



RECAP

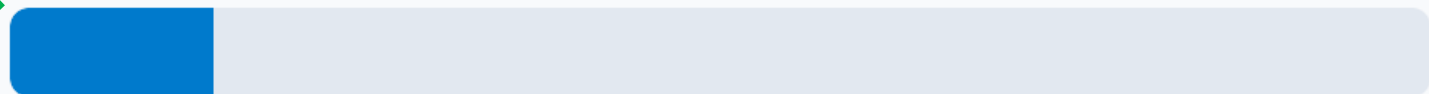
True or False: An RL agent uses its reward to determine what action to take next.

True



88%

False



13%

RECAP

True or False: Once a policy is created, an agent can just follow it to reach its goal.


✓ True



False



THE Q-LEARNING ALGORITHM

1. Start in initial state s_t ($t =$ current time)
2. Observe the reward r
3. Pick an action a_t to execute 
4. Executing a_t will cause the agent to go to state s_{t+1}
5. Update Q-value for $Q(s_t, a_t)$
6. Repeat steps 1-5, until we have a lot of data
7. Compute $\pi(s)$ from $Q(s, a)$

$$\pi^*(s) = \arg \max_a Q^{\pi^*}(s, a)$$

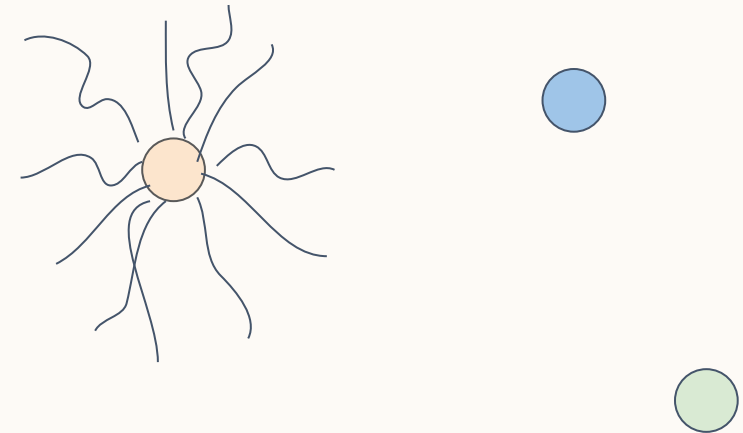
SELECTING ACTIONS DURING TRIALS

Option 1: Random Actions

Always pick an action at random

This provides the extensive **exploration** of the state space

Start
Medium reward
High reward



SELECTING ACTIONS DURING TRIALS - EXAMPLE

Scenario: We just moved to a new city for college. We need to learn a policy for ourselves to get from our apartment to campus.

Option 1: Random Actions

Always pick an action at random

Every day, we'll take a random turn at every intersection.

What's going to happen in this case?

SELECTING ACTIONS DURING TRIALS

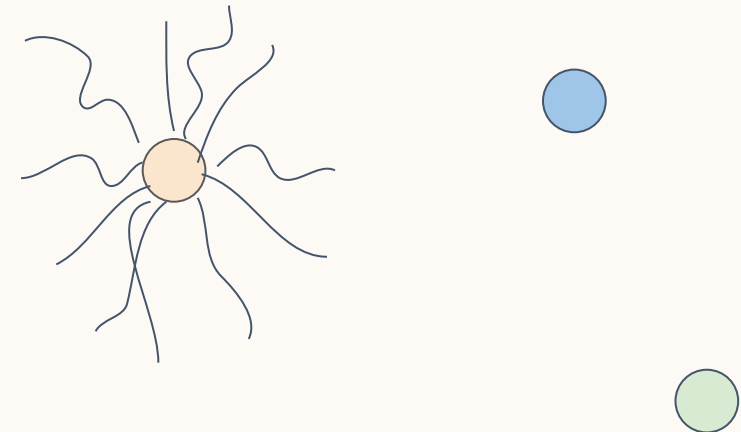
Option 1: Random Actions

Always pick an action at random

This provides the extensive **exploration** of the state space

- Eventually cover the state-action space
- May not see the same states multiple times (problem if we have stochastic actions)
- Biased towards seeing states near start state more often

Start
Medium reward
High reward



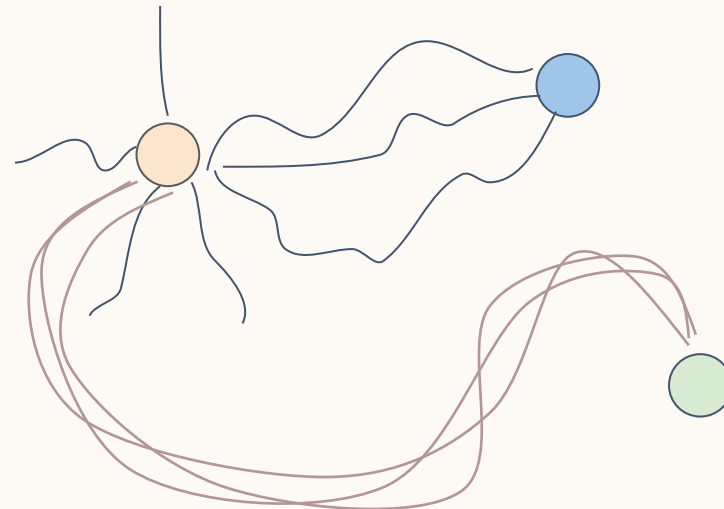
SELECTING ACTIONS DURING TRIALS

Option 2: Greedy Actions

Always pick action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$

This provides **exploitation** of the Q-values and rewards we have seen in past trials

Start
Medium reward
High reward



SELECTING ACTIONS DURING TRIALS - EXAMPLE

Scenario: We just moved to a new city for college. We need to learn a policy for ourselves to get from our apartment to campus.

Option 2: Greedy Actions

Always pick action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$

As soon as we find a route through random wandering, then every day we'll take the turn that got us the best result in the past

What's going to happen in this case?

SELECTING ACTIONS DURING TRIALS

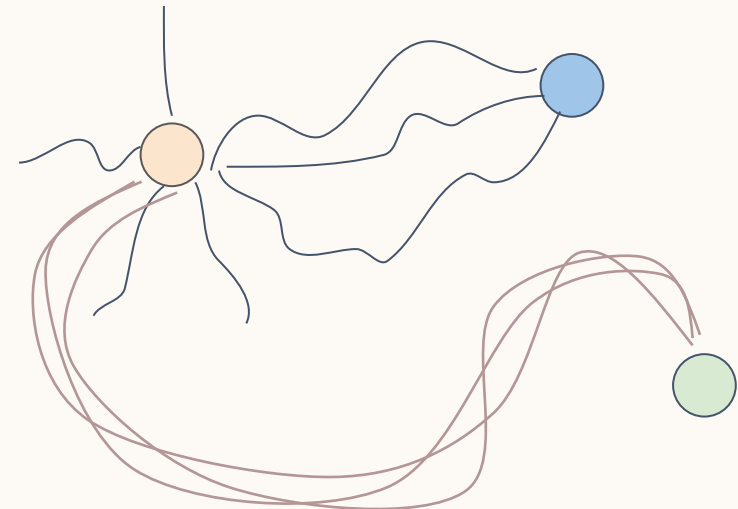
Option 2: Greedy Actions

Always pick action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$

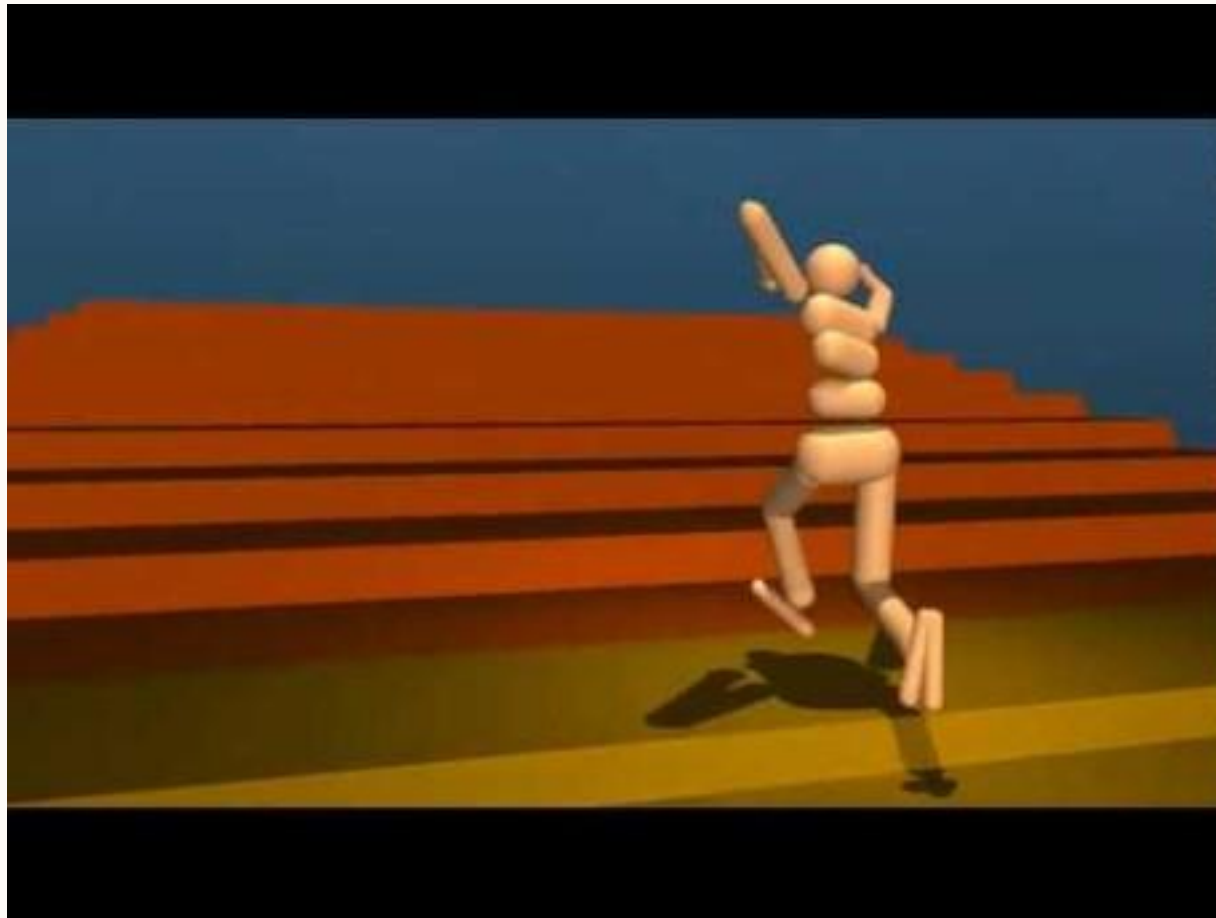
This provides **exploitation** of the Q-values and rewards we have seen in past trials

- Starts by exploring randomly
- Over time, bias towards states with reward
- May focus in on sub-optimal reward states
- May focus on repeating sub-optimal policies

Start
Medium reward
High reward



SUBOPTIMAL PATHS



https://www.youtube.com/watch?v=hx_bgoTF7bs&t=90s

SELECTING ACTIONS DURING TRIALS

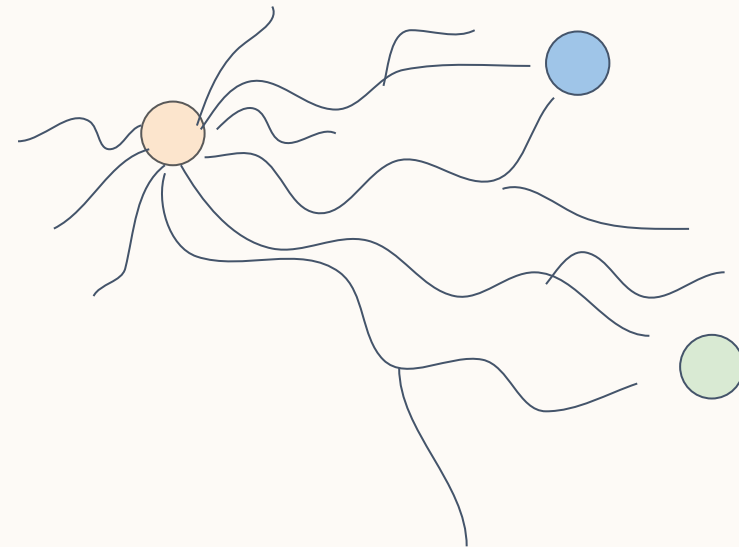
Option 3: ϵ -Greedy Action Selection

With probability ϵ , pick a random action

Otherwise, pick action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$

Simple way to trade off **exploration** and **exploitation**

Start
Medium reward
High reward



SELECTING ACTIONS DURING TRIALS - EXAMPLE

Scenario: We just moved to a new city for college. We need to learn a policy for ourselves to get from our apartment to campus.

Option 3: ϵ -Greedy Action Selection

With probability ϵ , pick a random action

Otherwise, pick action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$

Every day, we'll usually take turns that gave us the best results in the past, but sometimes we'll try something new or retry something less efficient

What's going to happen in this case?

SELECTING ACTIONS DURING TRIALS

Option 3: ϵ -Greedy Action Selection

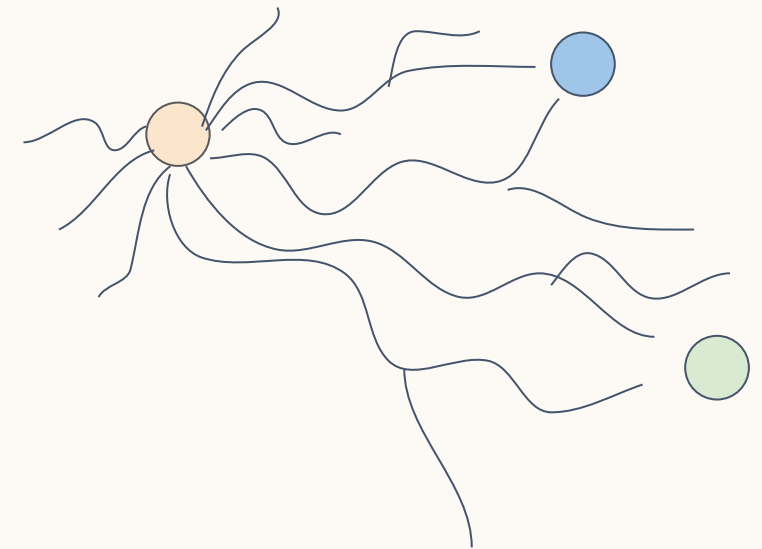
With probability ϵ , pick a random action

Otherwise, pick action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$ Simple

way to trade off **exploration** and **exploitation**

- Can explore at any time
- Tends to move towards rewards (i.e. spend time exploring relevant areas of state space)
- **GLIE**: Greedy in the Limit of Infinite Exploration
 - Guaranteed to converge to optimal $\pi(s)$!

Start
Medium reward
High reward



UPDATING Q-VALUES

The Q-value update equation:

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

The diagram illustrates the components of the Q-value update equation with annotations:

- $Q(s_t, a_t)$ (left): Update the state we came from
- $Q(s_t, a_t)$ (middle): Update the state we came from
- α : Learning rate $0 < \alpha \leq 1$
- r : Observed reward
- $\max_{a'}(Q(s_{t+1}, a'))$: The best value of doing a' in s_{t+1} . All actions available from s_{t+1} .
- $Q(s_t, a_t)$ (right): Temporal difference

UPDATING Q-VALUES

The Q-value update equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(\underbrace{r + \gamma \max_{a'} Q(s_{t+1}, a')}_{\substack{\text{New measurement} \\ \text{of } Q(s_t, a_t)}} - Q(s_t, a_t) \right)$$

$U(s_{t+1})$

Recall: $U(s) = \max_a Q(s, a)$

$$Q(s, a) = R(s) + \gamma \sum_{s'} p(s'|s, a) U(s')$$

We're updating the Q-Value based on its observed change in value

UPDATING Q-VALUES

The Q-value update equation (for terminal states):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r + \gamma \max_{a'} \cancel{Q(s_{t+1}, a')}) - Q(s_t, a_t)$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r - Q(s_t, a_t))$$

Why not set this directly to the observed reward?

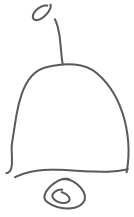
- We can set $Q(s_t, a_t) = r$ if our rewards are deterministic
- We should use the update equation if our rewards are stochastic!

FULL Q-LEARNING ALGORITHM

1. Start in initial state s_t ($t =$ current time)
2. Pick an action a_t to execute, **balancing exploration and exploitation**
3. Executing a_t will cause the agent to go to state s_{t+1}
4. Get the reward r
5. Update Q-value for $Q(s_t, a_t)$:
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r + \gamma \max_{a'}(Q(s_{t+1}, a')) - Q(s_t, a_t))$$
6. Repeat steps 1-5, until we have a lot of data
7. Compute $\pi(s)$ from $Q(s, a)$: $\pi^*(s) = \arg \max_a Q^{\pi^*}(s, a)$

TAKING RL FURTHER

How do we go from this...

3	R=0	R=0	R=0	R=+1
2	R=0	Wall	R=0	R=-1
1		R=0	R=0	R=0
	1	2	3	4

TAKING RL FURTHER

To this...?



https://www.youtube.com/watch?v=eHipy_j29Xw

OpenAI 2018

WHAT ARE WE “LEARNING” IN Q-LEARNING?

Instead of specifying a complex transition function and reward function, we are learning a table of Q-values:

	Use Sword	Move forward	Move back	Heal
S1 <i>Have sword</i> <i>Monster alive</i>	5	2	0	0
S2 <i>No sword</i> <i>Monster alive</i>	0	0	10	11
S3 <i>Have sword</i> <i>Monster dead</i>	0.001	2	1	5
S4 <i>No Sword</i> <i>Monster dead</i>	0	1	0	3

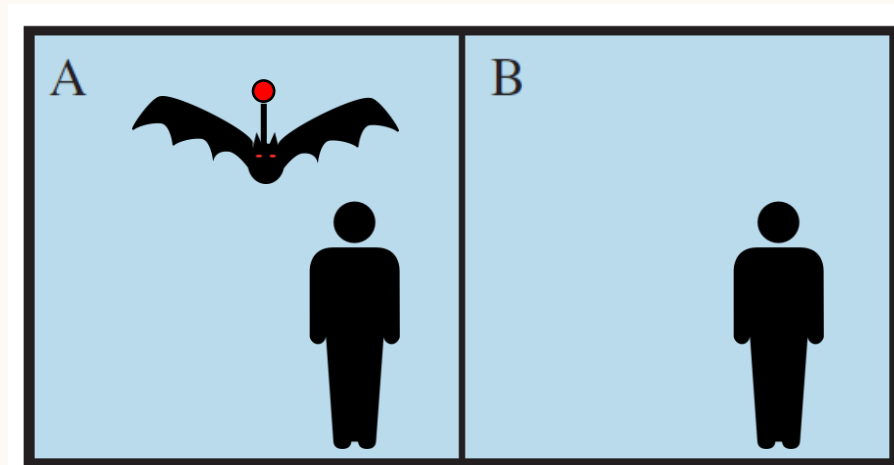
What does this mean for our *Agent Program*? Does this remind you of anything from a previous section?

WHAT ARE WE “LEARNING” IN Q-LEARNING?

We’re essentially learning a **lookup table** for states and actions.

Agent program:

Percept sequence	Action
[A, Empty]	Right
[A, Human]	Suck
[B, Empty]	Left
[B, Human]	Suck
...	...
[A, Empty], [B, Human]	Suck
[A, Empty], [B, Empty]	Left
...	...



WHY A LOOKUP TABLE?

- Learning a table is better than specifying it by hand
- Easy to interpret

But can we learn a more concise representation?

We can use **function approximation** to approximate the values in the Q-value table.

FUNCTION APPROXIMATION FOR Q-LEARNING

Function approximation: using a parameterized representation to generate approximate values instead of using a table of exact values

Example: use a weighted linear function to approximate Q-values

$$Q_{\theta}(s, a) = \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \theta_3 f_3(s, a) + \dots + \theta_n f_n(s, a)$$

Approximate Q-value

Weight parameter, which we can learn using linear regression or online learning (ML Module!)

Feature calculated from state and action

FUNCTION APPROXIMATION FOR Q-LEARNING

$$Q_{\theta}(s, a) = \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \theta_3 f_3(s, a) + \dots + \theta_n f_n(s, a)$$

What does this do for us?

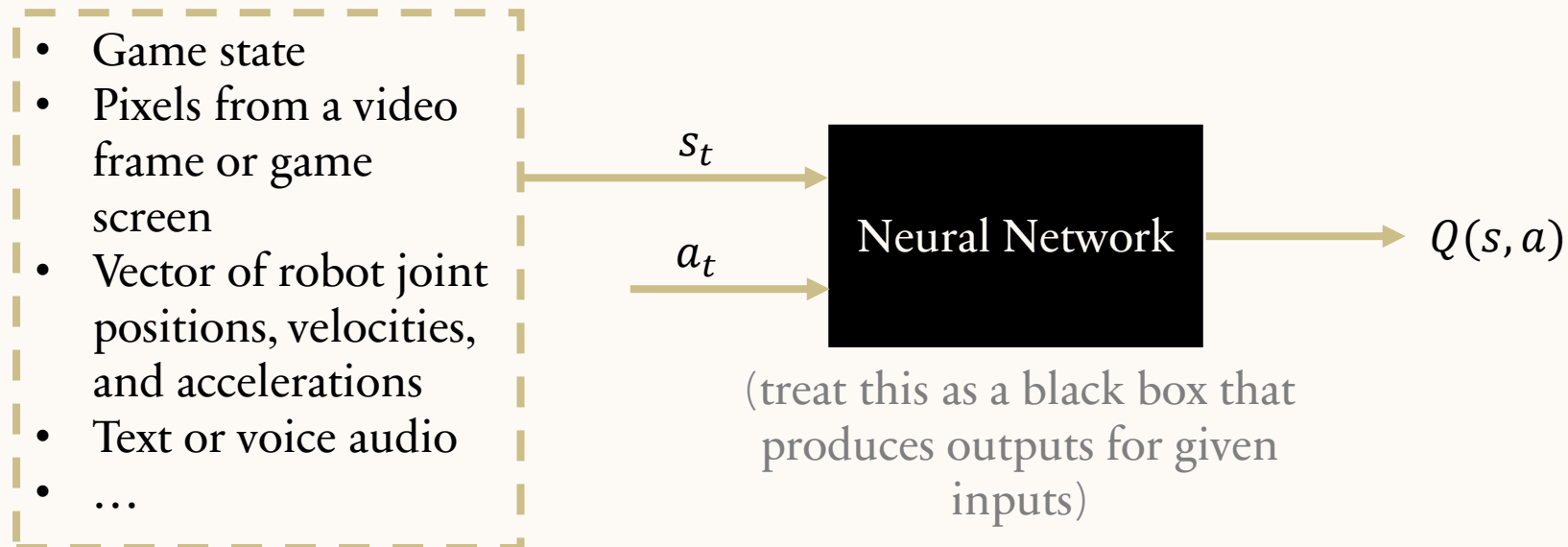
- We don't have to store a table of size $|S| \times |A|$
- We can calculate Q-functions for states and actions that we **haven't seen before!**

Function approximation **generalizes** to unexplored states and actions.

GETTING DEEP: MODERN FUNCTION APPROXIMATION

Instead of a linear function, we can use a deep neural network as a function approximator for the Q-values.

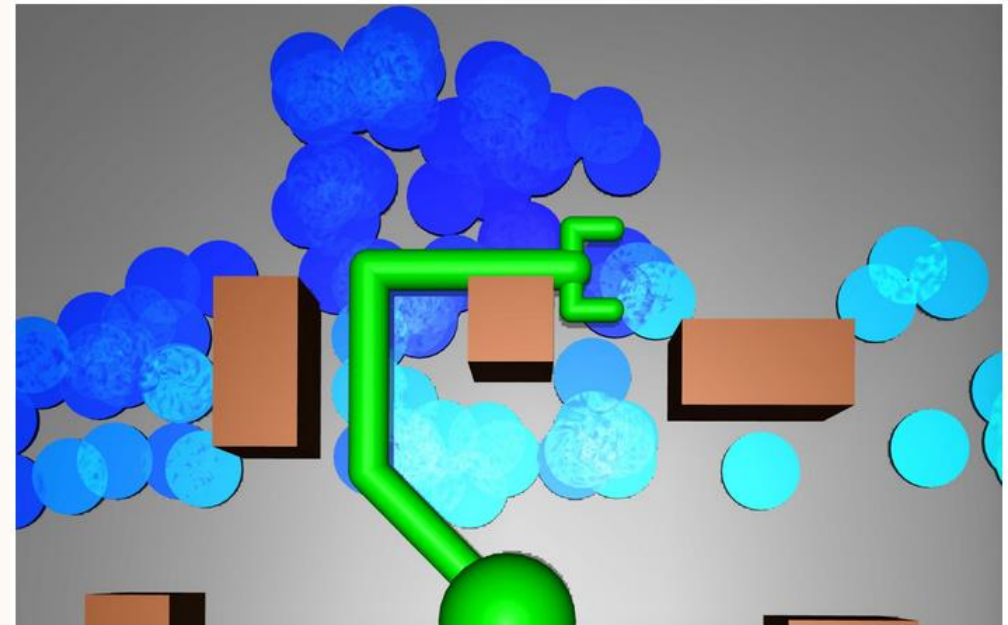
Goal: learn a neural network to replace the Q-value table:



WHY IS DEEP LEARNING EFFECTIVE FOR RL?

Advantage of neural networks for function approximation:

- Can be designed *without* the assumption that nearby states should generalize to similar values
- Models are large enough to encode complex relationships between agent and environment
- Example on right: similar colored positions are states that are “close” to each other



A distance metric learned by a neural network [3]. **Lighter blue** → **more distant**. The agent, which was trained to grasp objects using the robotic arm, takes into account obstacles and arm length when it measures the distance between two states.