

<https://introml.mit.edu/>

6.390 Intro to Machine Learning

Lecture 11: Reinforcement Learning

Shen Shen

April 25, 2025

11am, Room 10-250

Outline

- Recap: Markov decision processes
- Reinforcement learning setup
- Model-based methods
- Model-free methods
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again

Markov Decision Processes - Definition and terminologies

- \mathcal{S} : state space, contains all possible states s .
- \mathcal{A} : action space, contains all possible actions a .
- $T(s, a, s')$: the probability of transition from state s to s' when action a is taken.
- $R(s, a)$: reward, takes in a (state, action) pair and returns a reward.
- $\gamma \in [0, 1]$: discount factor, a scalar.

- $\pi(s)$: policy, takes in a state and returns an action.

The goal of an MDP is to find a "good" policy.

In 6.390,

- \mathcal{S} and \mathcal{A} are small discrete sets, unless otherwise specified.
- s' and a' are short-hand for the next-timestep
- $R(s, a)$ is deterministic and bounded.
- $\pi(s)$ is deterministic.

horizon- h value in state s : the expected sum of discounted rewards, starting in state s and following policy π for h steps.

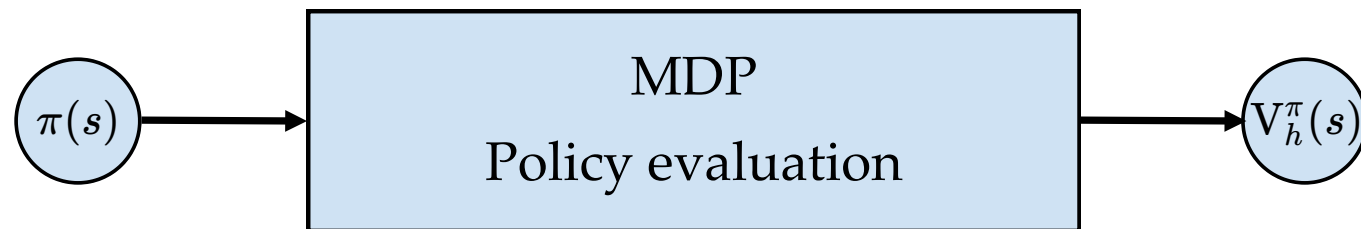
$$V_h^\pi(s) = \mathbf{R}(s, \pi(s)) + \gamma \sum_{s'} \mathbf{T}(s, \pi(s), s') V_{h-1}^\pi(s')$$

the immediate reward for taking the policy-prescribed action $\pi(s)$ in state s .

$(h - 1)$ horizon future values at a next state s'

sum up future values weighted by the probability of getting to that next state s'

discounted by γ



1. By summing h terms:

Recall: For a *given* policy $\pi(s)$, the (state) **value functions**

$$V_h^\pi(s) := \mathbb{E} \left[\sum_{t=0}^{h-1} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s, \pi \right], \forall s, h$$

2. By leveraging structure:

finite-horizon Bellman recursions

$$V_h^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_{h-1}^\pi(s'), \forall s$$

infinite-horizon Bellman equations

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

Given the recursion $Q_h^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{h-1}^*(s', a')$

we can have an infinite-horizon equation

$$Q_\infty^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_\infty^*(s', a')$$

Value Iteration

1. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:

2. $Q_{\text{old}}(s, a) = 0$

3. **while** True:

4. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:

5. $Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$

6. **if** $\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon$:

7. **return** Q_{new}

8. $Q_{\text{old}} \leftarrow Q_{\text{new}}$

if run this block h times
and break, then the
returns are exactly Q_h^*

$Q_\infty^*(s, a)$

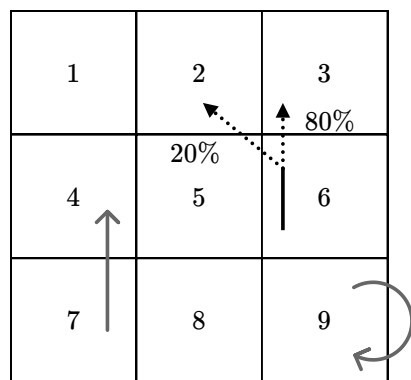


Outline

- Recap: Markov decision processes
- Reinforcement learning setup
- Model-based methods
- Model-free methods
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again






Recall Running example: Mario in a grid-world

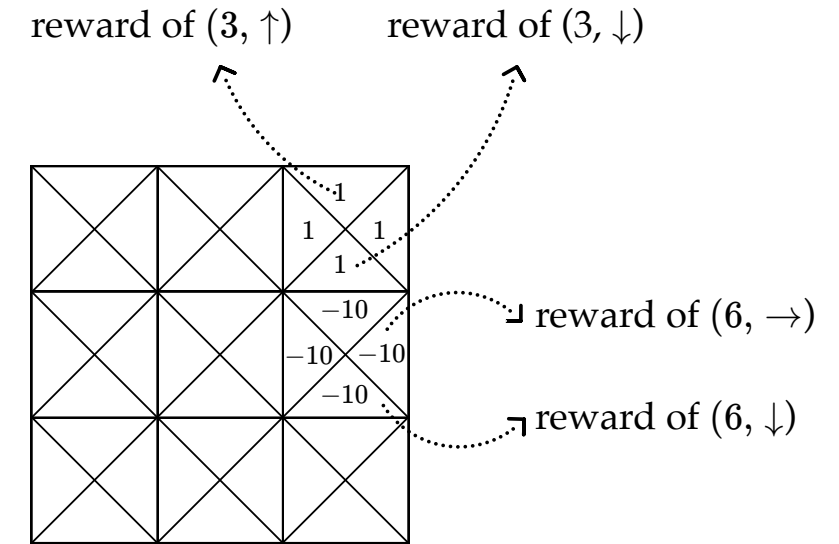


- 9 possible **states**
- 4 possible **actions**: {Up \uparrow , Down \downarrow , Left \leftarrow , Right \rightarrow }
- (state, action) results in a **transition** into a next state:
 - Normally, we get to the “intended” state;
 - E.g., in state (7), action “ \uparrow ” gets to state (4)
 - If an action would take Mario out of the grid world, stay put;
 - E.g., in state (9), “ \rightarrow ” gets back to state (9)
 - In state (6), action “ \uparrow ” leads to two possibilities:
 - 20% chance to (2)
 - 80% chance to (3)



Mario in a grid-world, cont'd

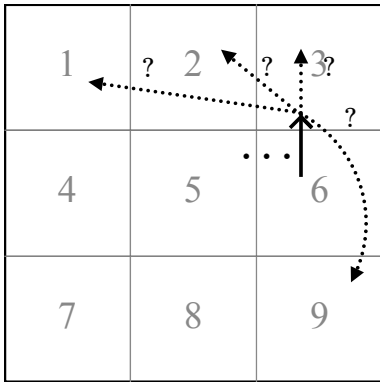
- (state, action) pairs give out **rewards**:
 -  in state 3, any action gives reward 1
 -  in state 6, any action gives reward -10
 -  any other (state, action) pair gives reward 0



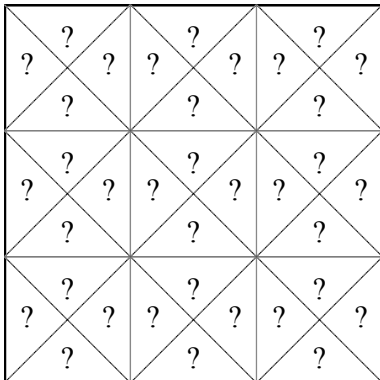
- **discount factor**: a scalar of 0.9 that reduces the "worth" of rewards, depending on the timing we receive them.
 - e.g., for $(3, \leftarrow)$ pair, we receive a reward of 1 at the start of the game; at the 2nd time step, we receive a discounted reward of 0.9; at the 3rd time step, it is further discounted to $(0.9)^2$, and so on.



Now Running example: Mario in a grid-world Reinforcement learning setup



- 9 possible *states*
- 4 possible *actions*: {Up \uparrow , Down \downarrow , Left \leftarrow , Right \rightarrow }
- *transition* probabilities are **unknown**



- *rewards* **unknown**
- *discount factor* $\gamma = 0.9$

~~Markov Decision Processes~~ - Definition and terminologies

Reinforcement Learning

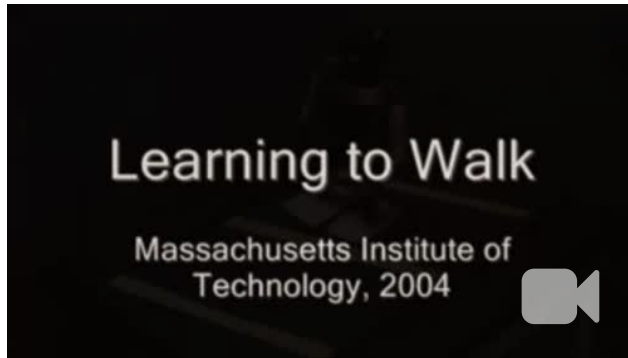
- \mathcal{S} : state space, contains all possible states s .
- \mathcal{A} : action space, contains all possible actions a .
- $T(s, a, s')$: the probability of transition from state s to s' when action a is taken
- $R(s, a)$: reward, takes in a (state, action) pair and returns a reward.
- $\gamma \in [0, 1]$: discount factor, a scalar.
- $\pi(s)$: policy, takes in a state and returns an action.

The goal of an ~~MDP~~ problem is to find a "good" policy.

RL

Reinforcement learning is very *general*:

robotics

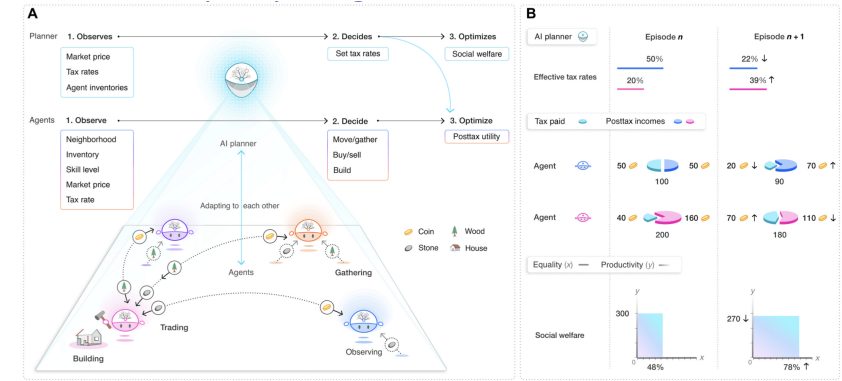


[Mastering the game of Go with deep neural networks and tree search. Silver et al. Nature 2017]

games



social sciences



[The AI Economist: Taxation policy design via two-level deep multiagent RL. Zheng et al. Science 2022]

chatbot (RLHF)



[Aligning language models to follow instructions. Ouyang et al. 2022]

health care

nature medicine

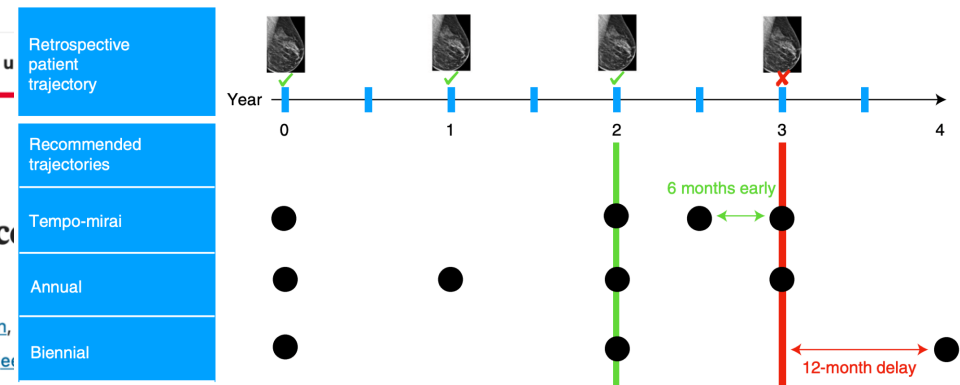
Explore content ▾ About the journal ▾ Publish with us ▾

[nature](#) > [nature medicine](#) > [articles](#) > article

Article | [Published: 13 January 2022](#)

Optimizing risk-based breast cancer treatment with reinforcement learning

[Adam Yala](#) , [Peter G. Mikhael](#), [Constance Lehman](#), [Gigin Lin](#), [Kevin Hughes](#), [Siddharth Satuluru](#), [Thomas Kim](#), [Imon Banerjee](#), [Barzilay](#)



Outline

- Recap: Markov decision processes
- Reinforcement learning setup
- **Model-based methods**
- Model-free methods
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again

MDP-Model-Based Methods (for solving RL)

Keep playing the game to approximate the unknown rewards and transitions.

- Rewards are particularly easy:

e.g. observe what reward r is received from taking the $(6, \uparrow)$ pair, we get $R(6, \uparrow)$

- Transitions are a bit more involved but still simple:

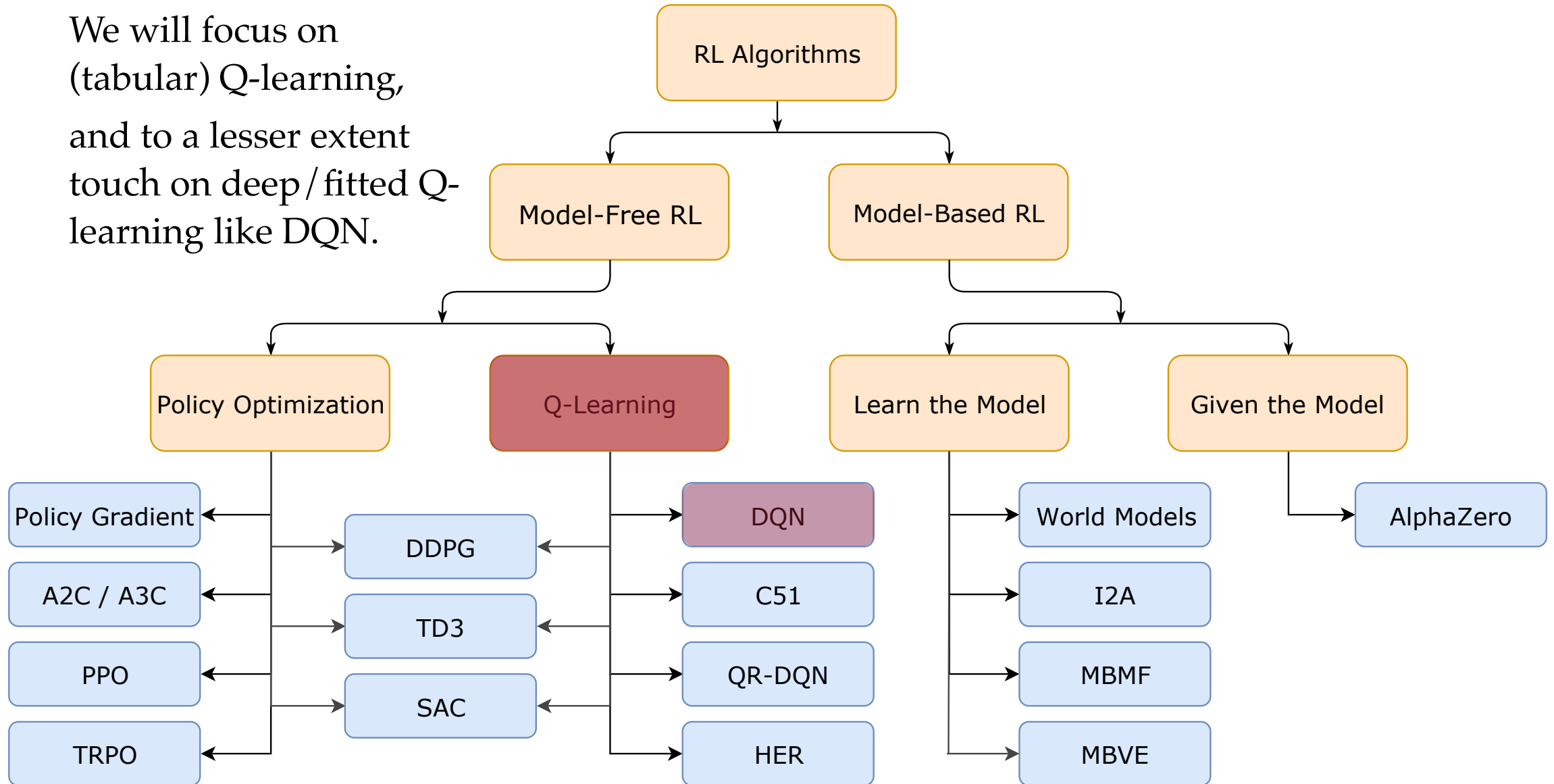
e.g. play the game 1000 times, count the # of times that (start in state 6, take \uparrow action, end in state 2), then, roughly, $T(6, \uparrow, 2) = (\text{that count}/1000)$

Now, with R and T estimated, we're back in MDP setting.

In Reinforcement Learning:

- *Model* typically means the MDP tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
- What's being learned is not usually called a *hypothesis*—we simply refer to it as the *value* or *policy*.

We will focus on
(tabular) Q-learning,
and to a lesser extent
touch on deep / fitted Q-
learning like DQN.



[A non-exhaustive, but useful taxonomy of algorithms in modern RL. [Source](#)]

Outline

- Recap: Markov decision processes
- Reinforcement learning setup
- Model-based methods
- **Model-free methods**
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again

Is it possible to get an optimal policy **without** learning transition or rewards explicitly?

We kinda know a way already:

We switch to an infinite-horizon scenario. For our stochastic machine, here is the infinite-horizon Q_{∞}^* function (computed via value iteration) for γ near 1.

	wash	paint	eject
dirty	[2.32274541	-0.70048204	0.]
clean	[2.32274541	5.71581775	0.]
painted	[2.32274541	6.9	10.]
ejected	[0.	0.	0.]]

What is the optimal thing to do with a **clean** object?

What will you do if it becomes **dirty**?

Does this optimal policy make intuitive sense?

(Recall, from the MDP lab)

With Q^* , we can back out π^* easily (greedily $\arg \max Q^*$, no need of transition or rewards)

But...

didn't we arrive at Q^* by value iteration;

We switch to an infinite-horizon scenario. For our stochastic machine, here is the infinite-horizon Q_∞^* function (computed via value iteration) for γ near 1.

```
           wash      paint      eject
dirty  [[ 2.32274541 -0.70048204  0.      ]
clean  [ 2.32274541  5.71581775  0.      ]
painted [ 2.32274541  6.9          10.     ]
ejected [ 0.          0.          0.      ]]
```

What is the optimal thing to do with a **clean** object?

What will you do if it becomes **dirty**?

Does this optimal policy make intuitive sense?

and didn't value iteration rely on transition and rewards explicitly?

Value Iteration

1. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2. $Q_{\text{old}}(s, a) = 0$
3. **while** True:
4. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
5. $Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$
6. **if** $\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon$:
7. **return** Q_{new}
8. $Q_{\text{old}} \leftarrow Q_{\text{new}}$

- Indeed, value iteration relied on having full access to R and T

$$Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$$

- Without R and T, how about we approximate like so:
 - pick an (s, a) pair
 - execute (s, a)
 - observe r and s'
 - update:

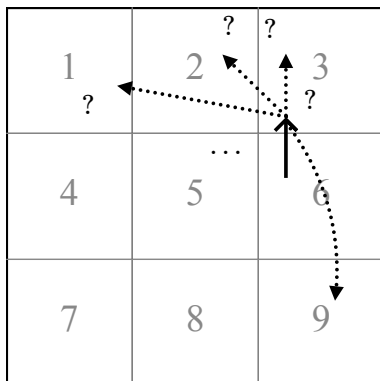
$$Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$$

target

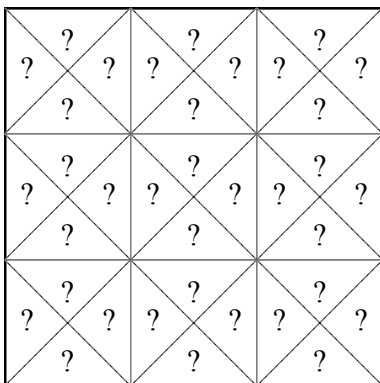
(we will see this idea has issues)



States & **unknown** transition:



unknown rewards:



$\gamma = 0.9$

Let's try $Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$

$Q_{\text{old}}(s, a)$

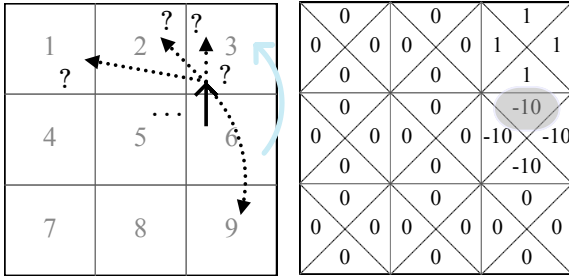
	0		0		0		0
0		0		0		0	
	0		0		0		0
0		0		0		0	
	0		0		0		0
0		0		0		0	
	0		0		0		0
0		0		0		0	

$Q_{\text{new}}(s, a)$

	0		0		1		1
0		0		0	1		1
	0		0		1		1
0		0		0	-10		-10
	0		0		-10		-10
0		0		0	-10		-10
	0		0		0		0
0		0		0	0		0

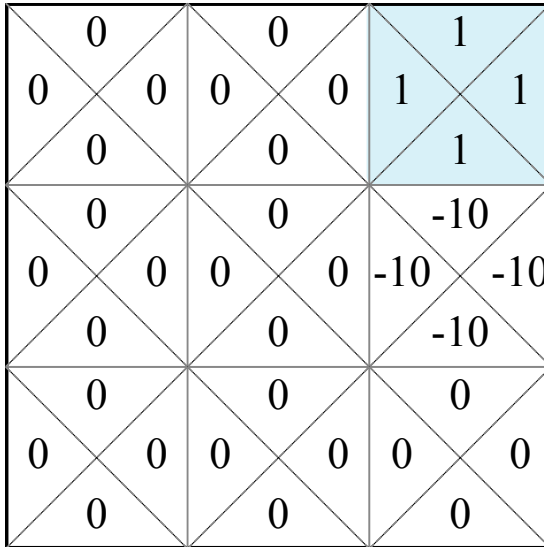
execute (3, \uparrow),
observe a reward
 $r = 1$

$\gamma = 0.9$ rewards now known

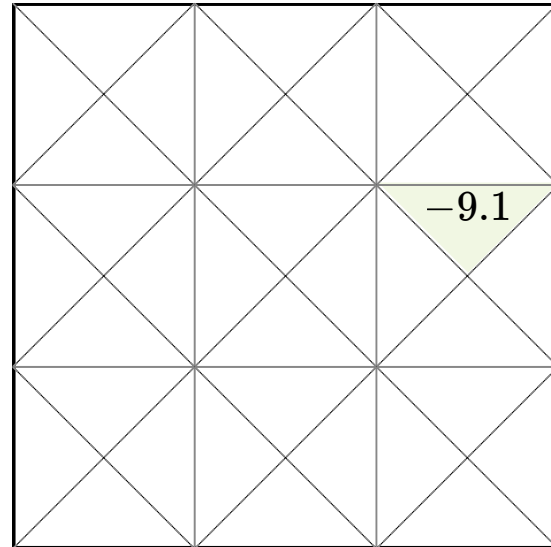


Let's try $Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$

$Q_{\text{old}}(s, a)$



$Q_{\text{new}}(s, a)$



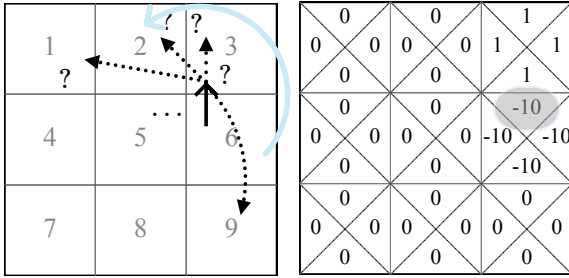
To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 3$
- update $Q(6, \uparrow)$ as:

$$-10 + 0.9 \max_{a'} Q_{\text{old}}(3, a')$$

$$= -10 + 0.9 = -9.1$$

$\gamma = 0.9$ rewards now known



Let's try $Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$

$Q_{\text{old}}(s, a)$

	0	0	0	1	1
0	0	0	0	1	1
	0	0	0	1	
0	0	0	0	-9.1	-10
	0	0	0	-10	
0	0	0	0	0	0
	0	0	0	0	
0	0	0	0	0	0
	0	0	0	0	

$Q_{\text{new}}(s, a)$

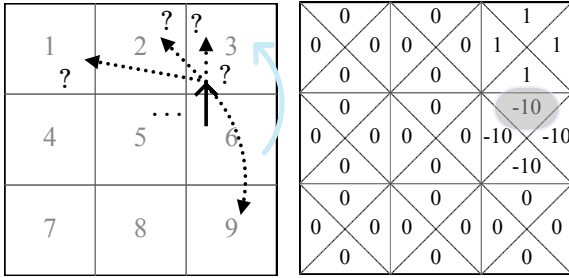
				-10	

To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 2$
- update $Q(6, \uparrow)$ as:

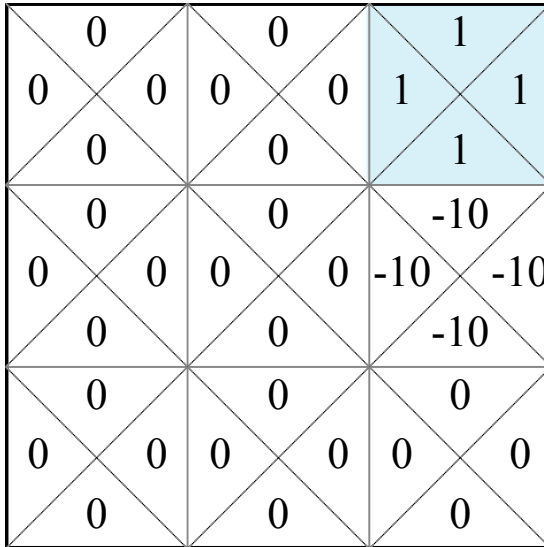
$$-10 + 0.9 \max_{a'} Q_{\text{old}}(2, a') \\ = -10 + 0 = -10$$

$\gamma = 0.9$ rewards now known

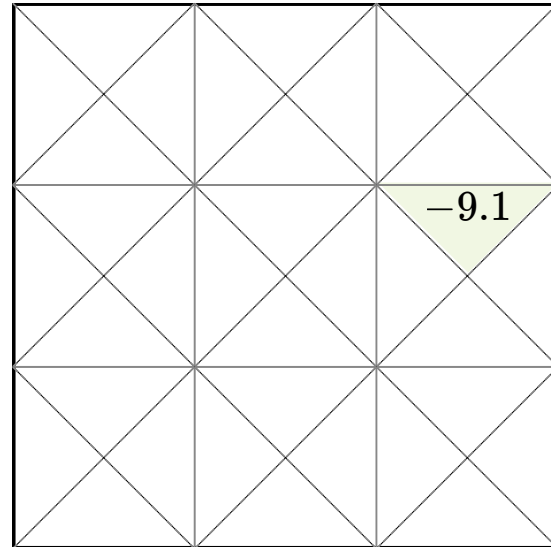


Let's try $Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$

$Q_{\text{old}}(s, a)$



$Q_{\text{new}}(s, a)$

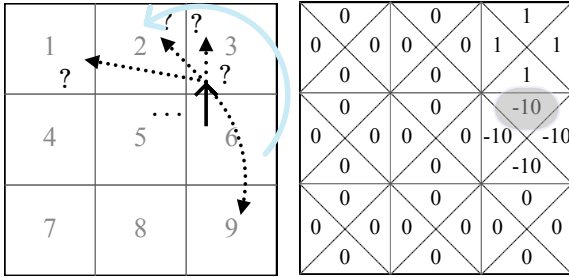


To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 3$
- update $Q(6, \uparrow)$ as:

$$-10 + 0.9 \max_{a'} Q_{\text{old}}(3, a') \\ = -10 + 0.9 = -9.1$$

$\gamma = 0.9$ rewards now known



Let's try $Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$

$Q_{\text{old}}(s, a)$

	0	0	0	1	1
0	0	0	0	1	1
	0	0	0	1	
0	0	0	0	-9.1	-10
	0	0	0	-10	
0	0	0	0	0	0
	0	0	0	0	
0	0	0	0	0	0
	0	0	0	0	

$Q_{\text{new}}(s, a)$

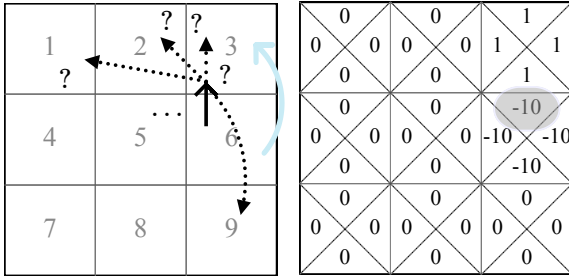
				-10	

To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 2$
- update $Q(6, \uparrow)$ as:

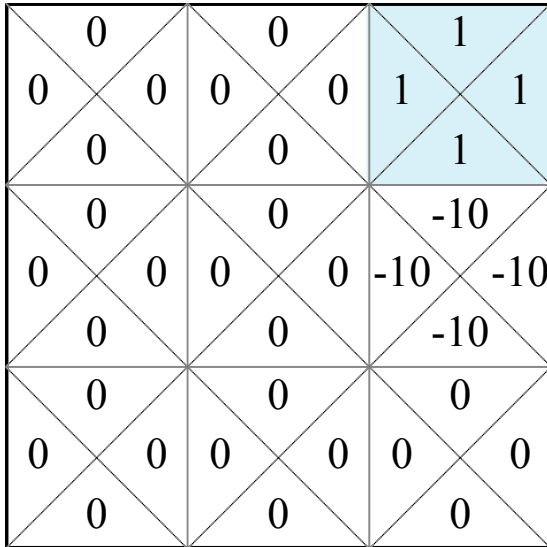
$$-10 + 0.9 \max_{a'} Q_{\text{old}}(2, a') \\ = -10 + 0 = -10$$

$\gamma = 0.9$ rewards now known

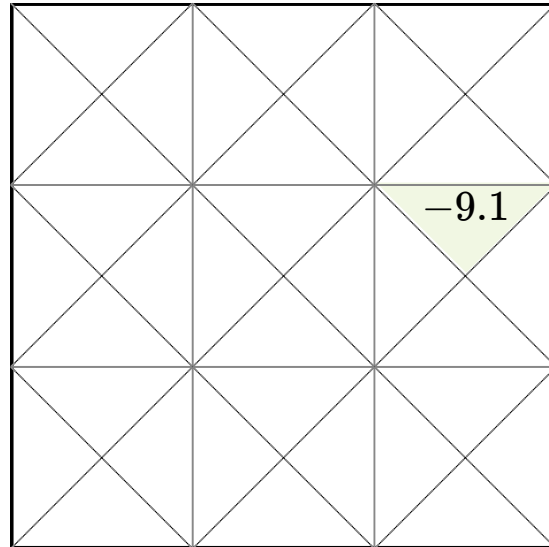


Let's try $Q_{\text{new}}(s, a) \leftarrow r + \gamma \max_{a'} Q_{\text{old}}(s', a')$

$Q_{\text{old}}(s, a)$



$Q_{\text{new}}(s, a)$



To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 3$
- update $Q(6, \uparrow)$ as:

$$-10 + 0.9 \max_{a'} Q_{\text{old}}(3, a')$$

$$= -10 + 0.9 = -9.1$$

- Value iteration relies on having full access to R and T

$$Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$$

- Without R and T , perhaps we could execute (s, a) , observe r and s'

$$Q_{\text{new}}(s, a) \leftarrow \underbrace{r}_{\text{target}} + \gamma \max_{a'} Q_{\text{old}}(s', a')$$

but the target keeps "washing away" the old progress. 🙄

- *Value iteration* relies on having full access to R and T

$$Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$$

- Without R and T , perhaps we could execute (s, a) , observe r and s'

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

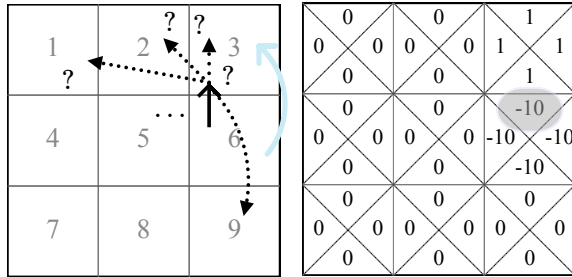
old belief
learning rate
target

core update rule of *Q-learning* 🥰

Outline

- Recap: Markov decision processes
- Reinforcement learning setup
- Model-based methods
- Model-free methods
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again

$\gamma = 0.9$ rewards now known

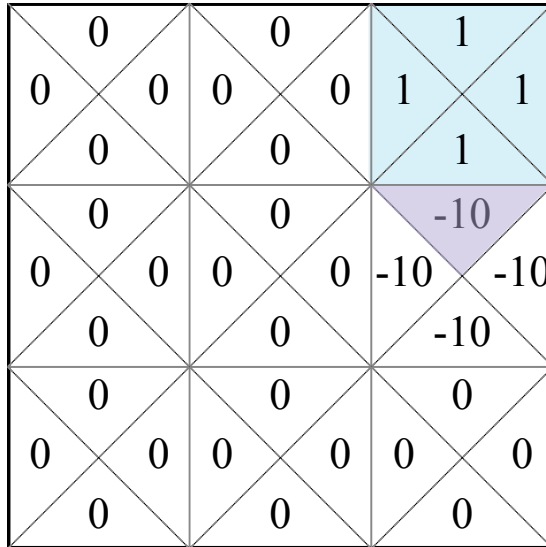


Q-learning update

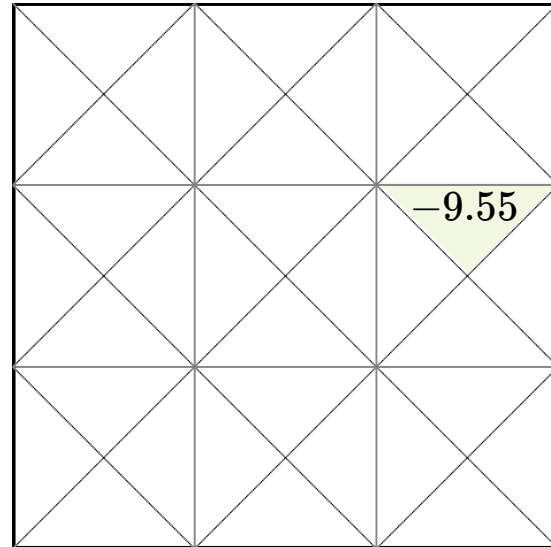
e.g. pick $\alpha = 0.5$

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

$Q_{\text{old}}(s, a)$



$Q_{\text{new}}(s, a)$



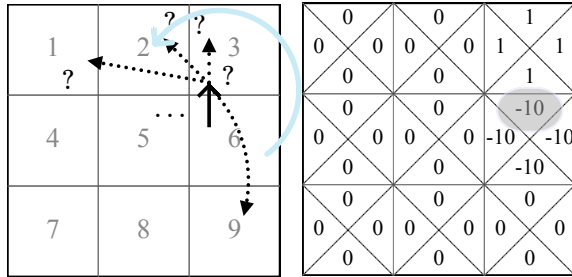
To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 3$
- update $Q(6, \uparrow)$ as:

$$(1 - 0.5) * -10 + 0.5(-10 + 0.9 \max_{a'} Q_{\text{old}}(3, a'))$$

$$= -5 + 0.5(-10 + 0.9) = -9.55$$

$\gamma = 0.9$ rewards now known



Q-learning update

e.g. pick $\alpha = 0.5$

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

$Q_{\text{old}}(s, a)$

0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	-9.55	-10
0	0	0	0	-10	-10
0	0	0	0	0	0
0	0	0	0	0	0

$Q_{\text{new}}(s, a)$

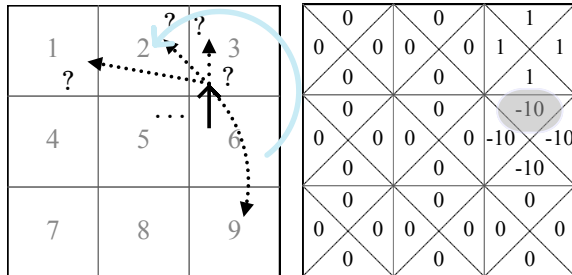
To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 2$
- update $Q(6, \uparrow)$ as:

$$(1 - 0.5) * -9.55 + 0.5(-10 + 0.9 \max_{a'} Q_{\text{old}}(2, a'))$$

$$= -4.775 + 0.5(-10 + 0) = -9.775$$

$\gamma = 0.9$ rewards now known

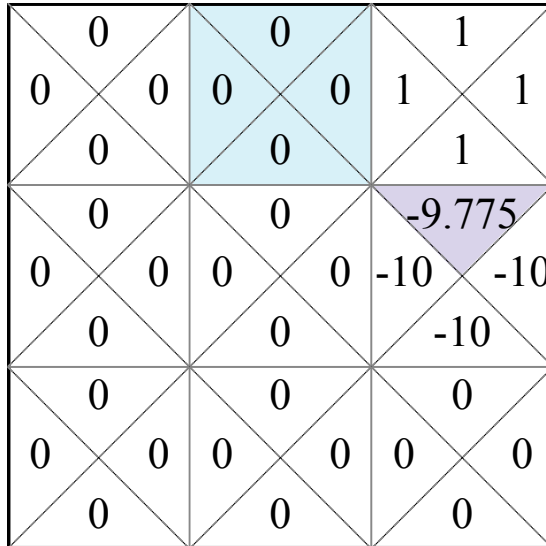


Q-learning update

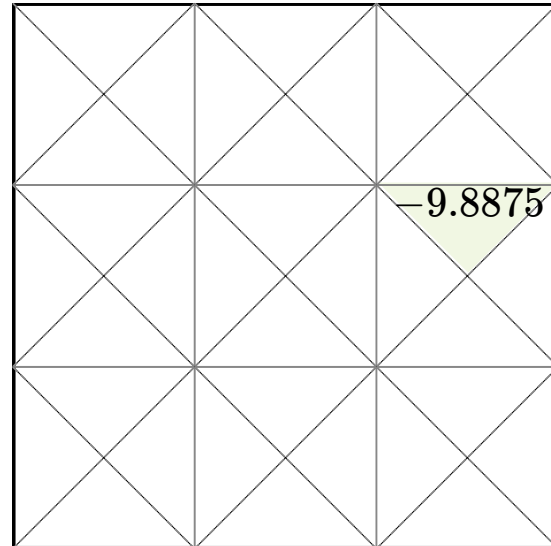
e.g. pick $\alpha = 0.5$

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

$Q_{\text{old}}(s, a)$



$Q_{\text{new}}(s, a)$



To update the estimate of $Q(6, \uparrow)$:

- execute $(6, \uparrow)$
- observe reward $r = -10$, next state $s' = 2$
- update $Q(6, \uparrow)$ as:

$$(1 - 0.5) * -9.775 + 0.5(-10 + 0.9 \max_{a'} Q_{\text{old}}(2, a'))$$

$$= -4.8875 + 0.5(-10 + 0) = -9.8875$$

Value Iteration($\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon$)

1. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2. $Q_{\text{old}}(s, a) = 0$
3. **while** True:
4. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
5. $Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$
6. **if** $\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon$:
7. **return** Q_{new}
8. $Q_{\text{old}} \leftarrow Q_{\text{new}}$

"calculating"

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$ max-iter)

1. $i = 0$
2. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
3. $Q_{\text{old}}(s, a) = 0$
4. $s \leftarrow s_0$
5. **while** $i < \text{max-iter}$:
6. $a \leftarrow \text{select_action}(s, Q_{\text{old}}(s, a))$
7. $r, s' \leftarrow \text{execute}(a)$
8. $Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s', a'))$
9. $s \leftarrow s'$
10. $i \leftarrow (i + 1)$
11. $Q_{\text{old}} \leftarrow Q_{\text{new}}$
12. **return** Q_{new}

"learning" (estimating)

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$ max-iter)

```
1.  $i = 0$ 
2. for  $s \in \mathcal{S}, a \in \mathcal{A}$  :
3.    $Q_{\text{old}}(s, a) = 0$ 
4.  $s \leftarrow s_0$ 
5. while  $i < \text{max-iter}$  :
6.    $a \leftarrow \text{select\_action}(s, Q_{\text{old}}(s, a))$ 
7.    $r, s' \leftarrow \text{execute}(a)$ 
8.    $Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s', a'))$ 
9.    $s \leftarrow s'$ 
10.   $i \leftarrow (i + 1)$ 
11.   $Q_{\text{old}} \leftarrow Q_{\text{new}}$ 
12. return  $Q_{\text{new}}$ 
```

"learning"

- Remarkably, 🖐️ can converge to the true infinite-horizon Q^* -values¹.
- Once converged, act greedily w.r.t Q^* again.
- But convergence can be extremely slow;
- We also only ever update an (s, a) entry by "playing" it.
- What if we are impatient, or resource constrained?

¹ given we visit all s, a infinitely often, and satisfy a decaying condition on the learning rate α .

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$ max-iter)

```
1.  $i = 0$ 
2. for  $s \in \mathcal{S}, a \in \mathcal{A}$  :
3.    $Q_{\text{old}}(s, a) = 0$ 
4.  $s \leftarrow s_0$ 
5. while  $i < \text{max-iter}$  :
6.    $a \leftarrow \text{select\_action}(s, Q_{\text{old}}(s, a))$ 
7.    $r, s' \leftarrow \text{execute}(a)$ 
8.    $Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s', a'))$ 
9.    $s \leftarrow s'$ 
10.   $i \leftarrow (i + 1)$ 
11.   $Q_{\text{old}} \leftarrow Q_{\text{new}}$ 
12. return  $Q_{\text{new}}$ 
```

"learning"

- During learning, especially in early stages, we'd like to explore, and observe diverse (s, a) consequences.
- During later stages, can act more greedily w.r.t. the estimated Q values

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$ max-iter)

```
1.  $i = 0$ 
2. for  $s \in \mathcal{S}, a \in \mathcal{A}$  :
3.    $Q_{\text{old}}(s, a) = 0$ 
4.  $s \leftarrow s_0$ 
5. while  $i < \text{max-iter}$  :
6.    $a \leftarrow \text{select\_action}(s, Q_{\text{old}}(s, a))$ 
7.    $r, s' \leftarrow \text{execute}(a)$ 
8.    $Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s', a'))$ 
9.    $s \leftarrow s'$ 
10.   $i \leftarrow (i + 1)$ 
11.   $Q_{\text{old}} \leftarrow Q_{\text{new}}$ 
12. return  $Q_{\text{new}}$ 
```

"learning"

- ϵ -greedy action selection strategy:
 - with probability ϵ , choose an action $a \in \mathcal{A}$ uniformly at random
 - with probability $1 - \epsilon$, choose $\arg \max_a Q_{\text{old}}(s, a)$

the current estimate of Q values

- ϵ controls the trade-off between *exploration vs. exploitation*.

Outline

- Recap: Markov decision processes
- Reinforcement learning setup
- Model-based methods
- Model-free methods
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again

- So far, Q-learning is only kinda sensible for (small) tabular setting.
- What do we do if \mathcal{S} and/or \mathcal{A} are large, or even continuous?
- Notice that the key update line in Q-learning algorithm:

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

is equivalently:

$$Q_{\text{new}}(s, a) \leftarrow Q_{\text{old}}(s, a) + \alpha ([r + \gamma \max_{a'} Q_{\text{old}}(s', a')] - Q_{\text{old}}(s, a))$$

$$\text{new belief} \leftarrow \text{old belief} + \text{learning rate} (\text{target} - \text{old belief})$$

$$Q_{\text{new}}(s, a) \leftarrow Q_{\text{old}}(s, a) + \alpha ([r + \gamma \max_{a'} Q_{\text{old}}(s', a')] - Q_{\text{old}}(s, a))$$

$$\text{new belief} \leftarrow \text{old belief} + \text{learning rate} (\text{target} - \text{old belief})$$

- Reminds us of: when minimizing $(\text{target} - \text{guess}_{\theta})^2$

$$\text{Gradient descent does: } \theta_{\text{new}} \leftarrow \theta_{\text{old}} + \eta (\text{target} - \text{guess}_{\theta}) \frac{d(\text{guess})}{d\theta}$$

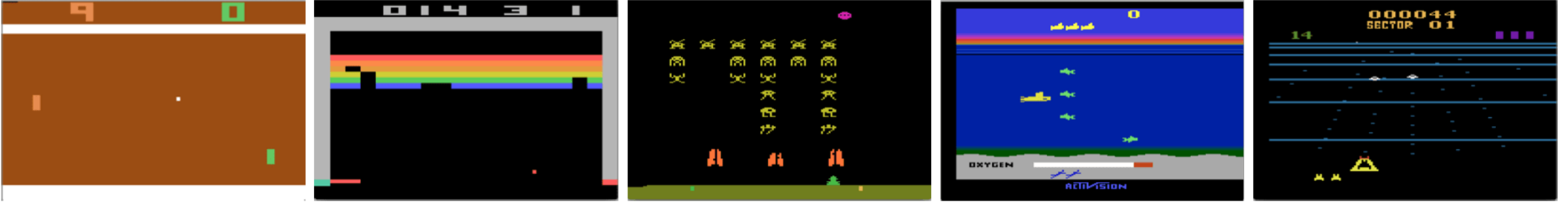
- Generalize tabular Q-learning for continuous state / action space:

1. parameterize $Q_{\theta}(s, a)$

2. execute (s, a) , observe (r, s') , construct the target $r + \gamma \max_{a'} Q_{\theta}(s', a')$

3. regress $Q_{\theta}(s, a)$ against the target, i.e. update θ via

gradient-descent methods to minimize $(\text{target} - Q_{\theta}(s, a))^2$



Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N

Initialize action-value function Q with random weights

for episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

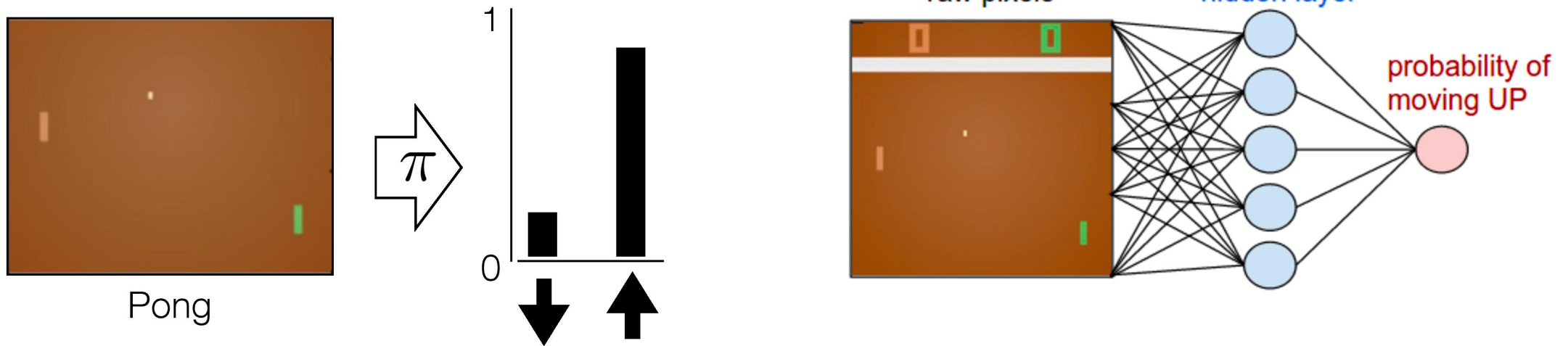
end for

end for

Outline

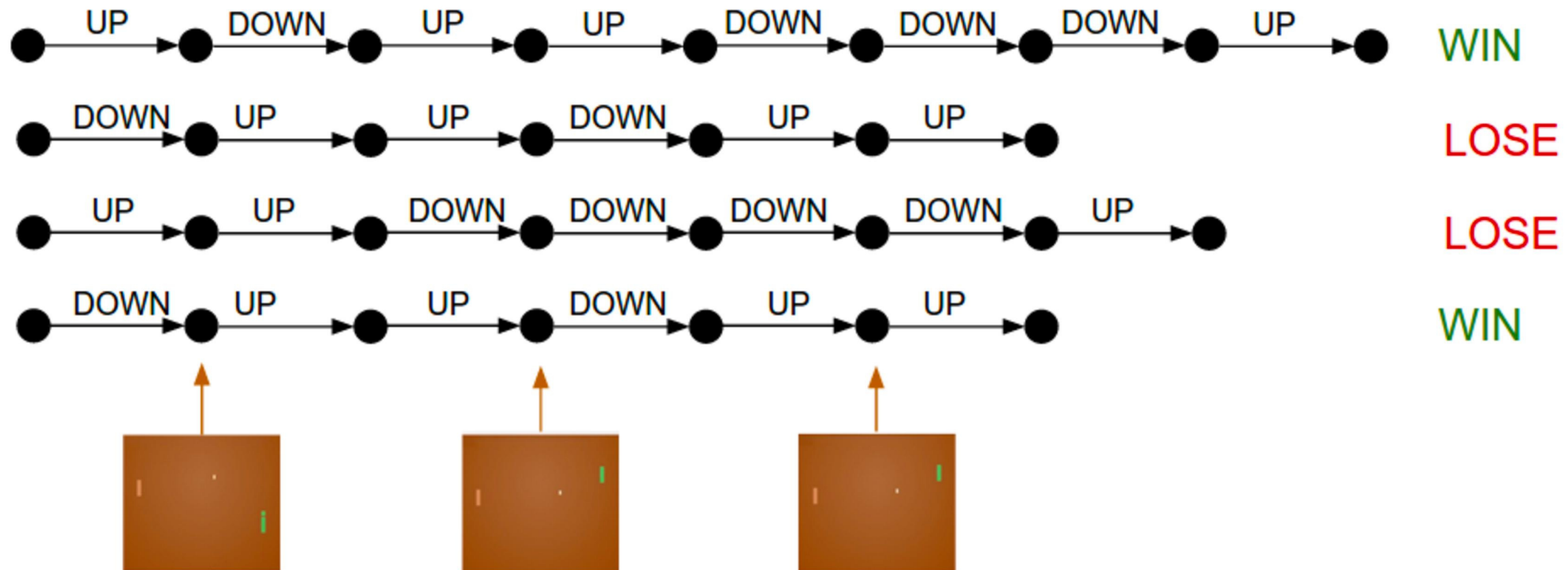
- Recap: Markov decision processes
- Reinforcement learning setup
- Model-based methods
- Model-free methods
 - (tabular) Q-learning
 - ϵ -greedy action selection
 - exploration vs. exploitation
 - (neural network) Q-learning
- Reinforcement learning setup again

- If explicit “good” state-action pair is given, also supervised learning.
- Behavior cloning or imitation learning.
- But what if no explicit guide?



[Adapted from Andrej Karpathy: <http://karpathy.github.io/2016/05/31/rl/>]

- If no direct supervision is available?
- Strictly RL setting. Interact, observe, get data, use rewards as "coy" supervision signal.



[Adapted from Andrej Karpathy: <http://karpathy.github.io/2016/05/31/rl/>]

How Much Information is the Machine Given during Learning?

► “Pure” Reinforcement Learning (**cherry**)

- The machine predicts a scalar reward given once in a while.

► **A few bits for some samples**

► Supervised Learning (**icing**)

- The machine predicts a category or a few numbers for each input
- Predicting human-supplied data
- **10→10,000 bits per sample**

► Self-Supervised Learning (**cake génoise**)

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- **Millions of bits per sample**



Reinforcement learning has a lot of challenges:

- Data can be very expensive / tricky to get
 - sim-to-real gap
 - sparse rewards
 - exploration-exploitation trade-off
 - catastrophic forgetting
- Learning can be very inefficient
 - temporal process, compound error
 - super high variance
 - learning process hard to stabilize

...

Summary

- We saw, last week, how to find good in a known MDP: these are policies with high cumulative expected reward.
- In reinforcement learning, we assume we are interacting with an *unknown* MDP, but we still want to find a good policy. We will do so via estimating the Q value function.
- One problem is how to select actions to gain good reward while learning. This “exploration vs exploitation” problem is important.
- Q-learning, for discrete-state problems, will converge to the optimal value function (with enough exploration).
- “Deep Q learning” can be applied to continuous-state or large discrete-state problems by using a parameterized function to represent the Q-values.

<https://forms.gle/6snt5oZgS1N9nZY78>

We'd love to hear
your thoughts.

Thanks!