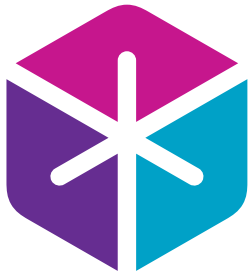
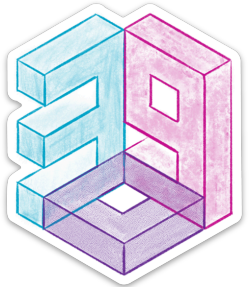




https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 11: Reinforcement Learning

Shen Shen

Apr 27, 2026

3pm, Room 10-250

Slides and Lecture Recording

Recall

MDP: Definition and terminologies

- $\mathcal{S}$: state space, contains all possible states s.
- $\mathcal{A}$: action space, contains all possible actions a.
- $\mathrm{T}(s, a, s')$: the probability of transition from state s to s' when action a is taken.
- $\mathrm{R}(s, a)$: 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 state and action.
- $\mathrm{R}(s, a)$ is deterministic and bounded.
- $\pi(s)$ is deterministic.

`Recall`

Use the definition and sum up expected rewards:

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

Or, leverage the recursive structure:

2. **finite-**horizon Bellman recursions

$$\mathrm{V}_h^\pi(s) = \mathrm{R}(s, \pi(s)) + \gamma \sum_{s'} \mathrm{T}\left(s, \pi(s), s'\right) \mathrm{V}_{h-1}^\pi\left(s'\right)$$

3. **infinite-**horizon Bellman equations

$$\mathrm{V}_\infty^\pi(s) = \mathrm{R}(s, \pi(s)) + \gamma \sum_{s'} \mathrm{T}\left(s, \pi(s), s'\right) \mathrm{V}_\infty^\pi\left(s'\right)$$

`Recall`

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

2️⃣ $$\mathrm{V}_h^\pi(s) = \mathrm{R}(s, \pi(s)) + \gamma \sum_{s'} \mathrm{T}(s, \pi(s), s') \mathrm{V}_{h-1}^\pi(s')$$

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

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

sum of future values weighted by the probability of reaching that next state s'

discounted by γ

`Recall`

the optimal state-action value function $\mathrm{Q}_h^*(s,a)$:

the expected sum of discounted rewards, obtained by

- starting in state s
- taking action a, for one step
- acting **optimally** thereafter for the remaining $(h-1)$ steps

4 $\mathrm{V}_h^*(s) = \max_a \left[\mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s,a,s') \mathrm{V}_{h-1}^*(s')\right] = \max_a \left[\mathrm{Q}_h^*(s,a)\right]$

Q^* satisfies the Bellman recursion:

5 $$\mathrm{Q}_h^*(s,a) = \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}\left(s,a,s'\right) \max_{a'} \mathrm{Q}_{h-1}^*\left(s',a'\right)$$

Recall

Value iteration: iteratively invoke

5️⃣ $$\mathrm{Q}_h^*(s,a) = \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s,a,s') \max_{a'} \mathrm{Q}_{h-1}^*(s',a')$$

Value Iteration

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

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

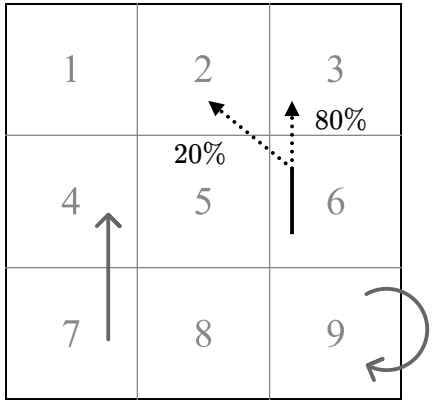
returns are Q_∞^*

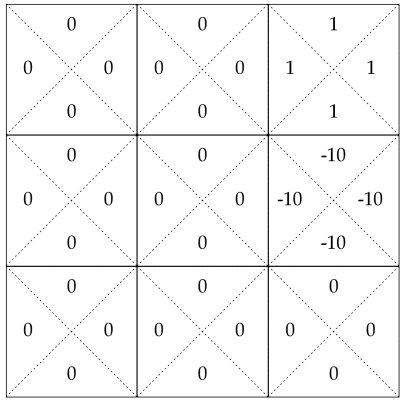
Outline

- Reinforcement learning setup
- Tabular Q-learning
  - exploration vs. exploitation
  - ϵ-greedy action selection
- Fitted Q-learning
- (Policy gradient)

Mario in a grid-world *v1.0*

(Markov-decision-process version)





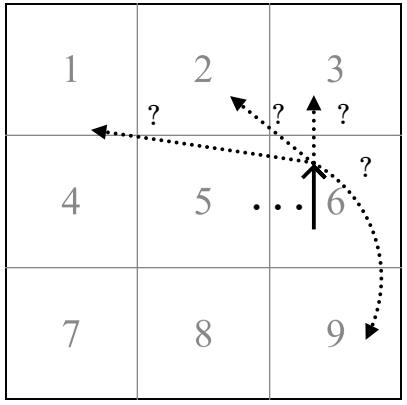
- 9 possible *states*
- 4 possible *actions*: {Up ↑, Down ↓, Left ←, Right →}
- *transition* probabilities are **known**

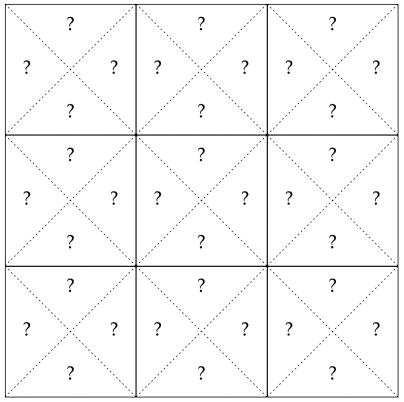
$$\text{e.g., } \mathrm{T}(7, \uparrow, 4) = 1 \quad \mathrm{T}(9, \rightarrow, 9) = 1 \quad \mathrm{T}(6, \uparrow, 3) = 0.8 \quad \mathrm{T}(6, \uparrow, 2) = 0.2$$

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

Mario in a grid-world *v2.0*

(reinforcement learning version)





- 9 possible *states*
- 4 possible *actions*: {Up ↑, Down ↓, Left ←, Right →}
- *transition* probabilities are **unknown**

e.g., $\mathrm{T}(7, \uparrow, 4) = ?$ $\mathrm{T}(9, \rightarrow, 9) = ?$ $\mathrm{T}(6, \uparrow, 3) = ?$ $\mathrm{T}(6, \uparrow, 2) = ?$

- *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.
- ~~$\mathrm{T}(s, a, s')$: the probability of transition from state s to s' when action a is taken~~
- ~~$\mathrm{R}(s, a)$: reward, takes in a (state, action) pair and returns a reward~~.
- $\gamma \in [0, 1]$: discount factor, a scalar.

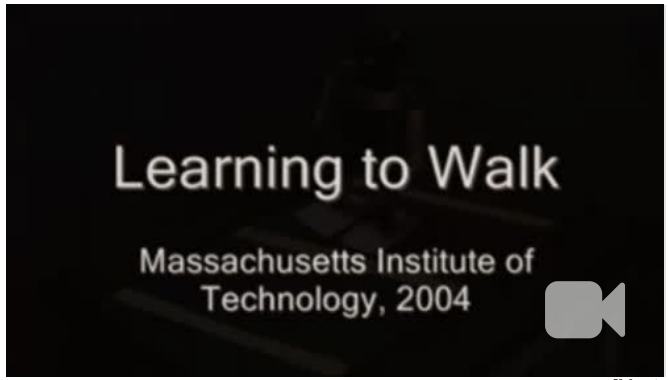
- $\pi(s)$: policy, takes in a state and returns an action.

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

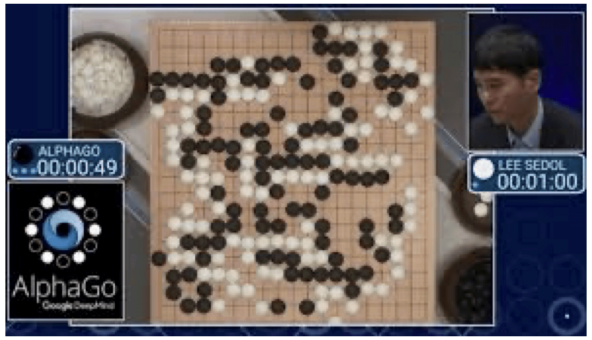
  RL

Reinforcement learning is very *general*:

robotics

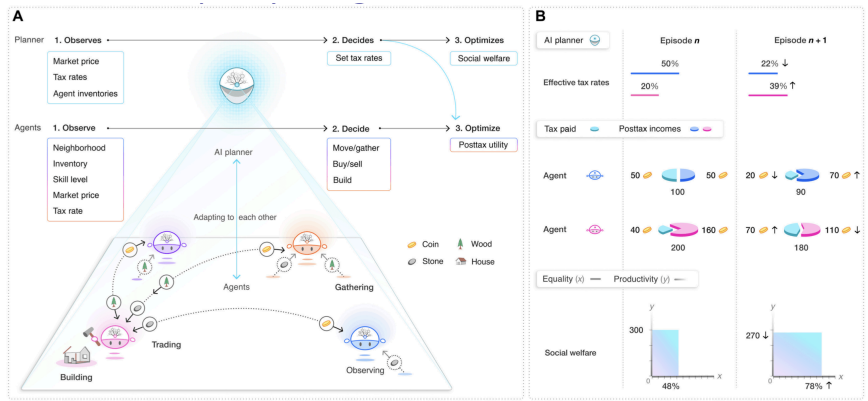


games



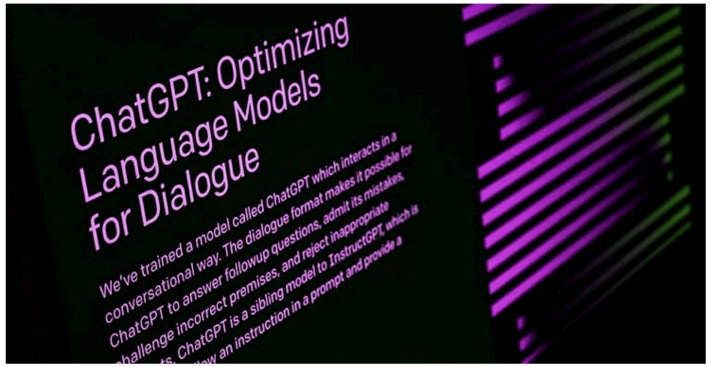
[Mastering the game of Go with deep neural networks and tree search. Silver et al. Nature 20

social sciences



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

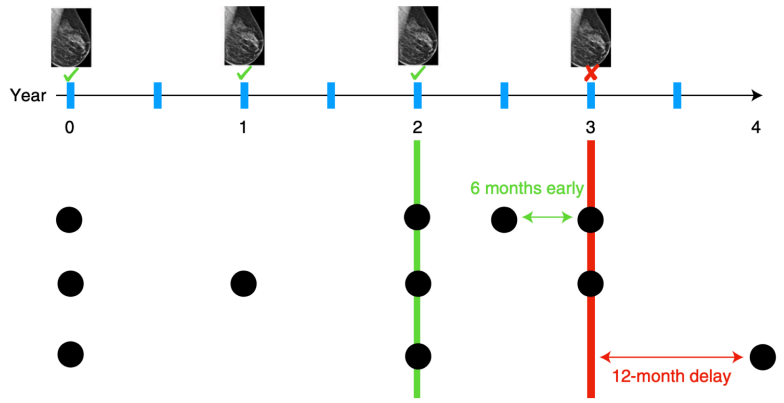
chatbots (RLHF)



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

health care

…



Model-based RL: Learn the MDP tuple

1. Collect experiences (s, a, r, s')

2. Estimate $\hat{\mathrm{T}}$, $\hat{\mathrm{R}}$

e.g. $\hat{\mathrm{T}}(6, \uparrow, 2) \approx$

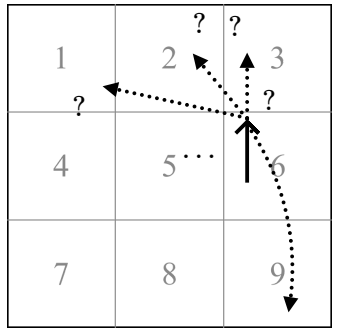
$$\frac{\text{observed } (6,\uparrow,2)}{\text{total}(6,\uparrow) \text{ play count}}$$

e.g. $\hat{\mathrm{R}}(6, \uparrow) =$

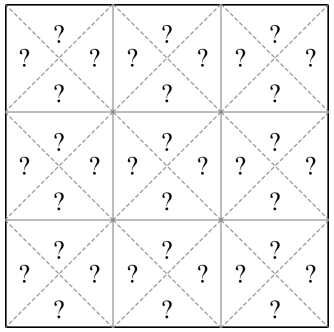
observed reward
received from $(6, \uparrow)$

3. Solve $\langle \mathcal{S}, \mathcal{A}, \hat{\mathrm{T}}, \hat{\mathrm{R}}, \gamma \rangle$ via e.g. *Value Iteration*

Unknown transition:

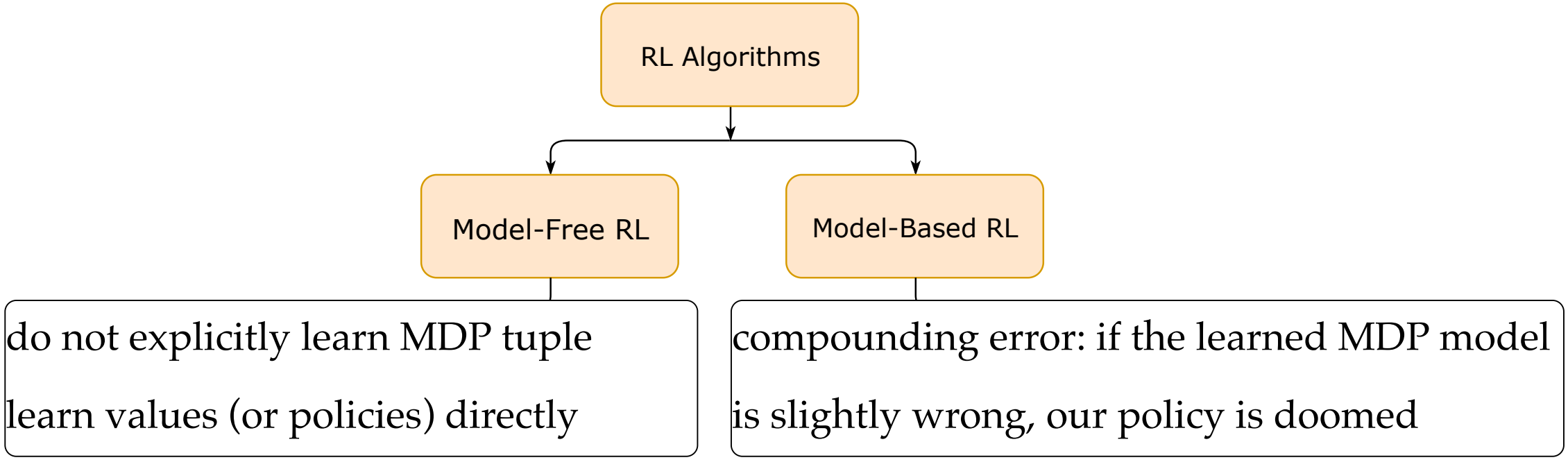


Unknown rewards



$\gamma = 0.9$

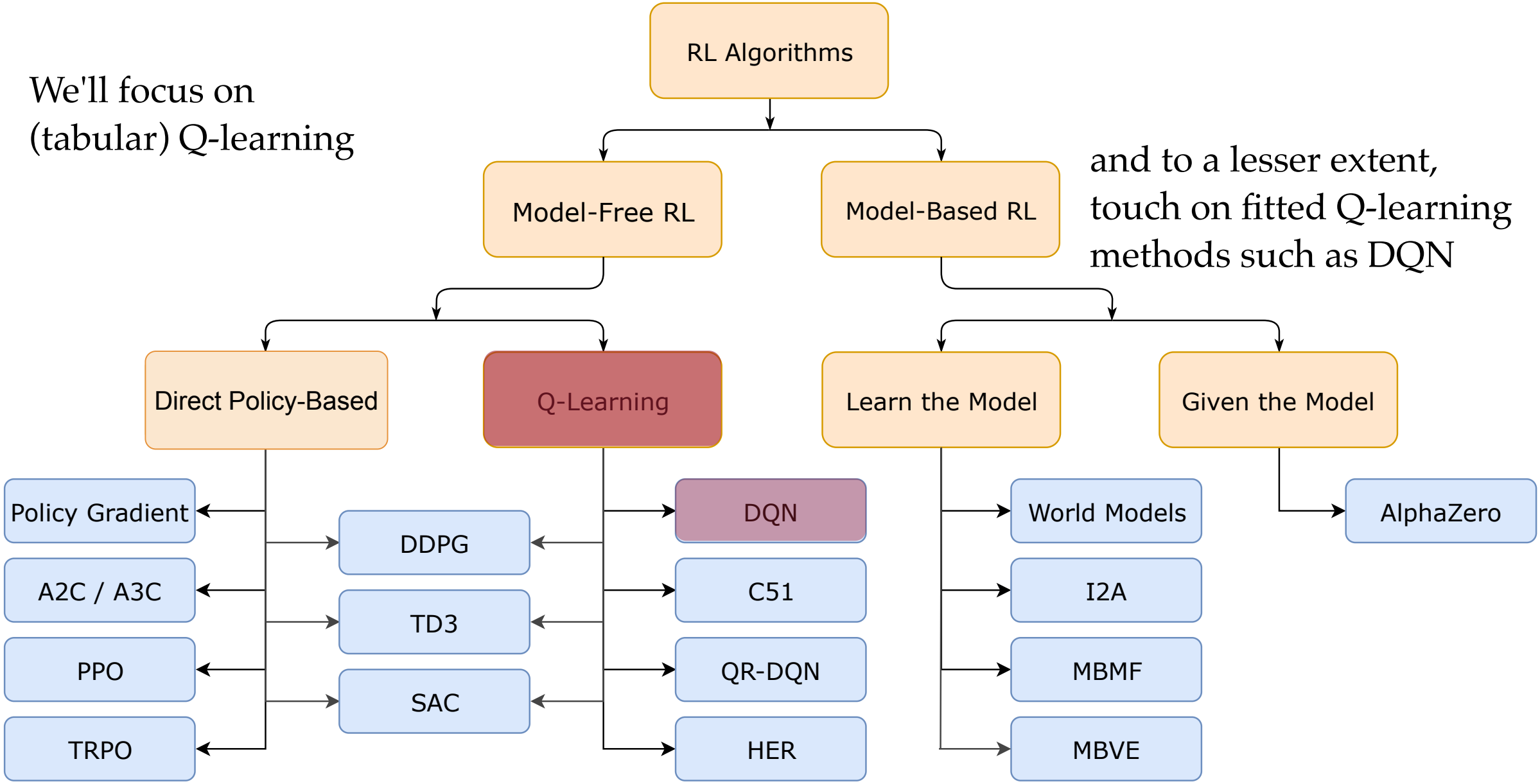
(s, a)	r	s'
$(1, \uparrow)$	0	1
$(1, \downarrow)$	0	4
	...	
$(3, \uparrow)$	1	3
$(3, \downarrow)$	1	6
	...	
$(6, \uparrow)$	-10	3
$(6, \uparrow)$	-10	2
	...	



- In RL we don't say hypothesis; we say model, value, or policy
  Model in MDP/RL typically means the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}, \gamma \rangle$

We'll focus on
(tabular) Q-learning

and to a lesser extent,
touch on fitted Q-learning
methods such as DQN



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

Outline

- Reinforcement learning setup
- Tabular Q-learning
  - exploration vs. exploitation
  - ϵ-greedy action selection
- Fitted Q-learning
- (Policy gradient)

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

Yes! We know one 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)

Optimal policy π^* easily extracted from Q^*:

6 $$\pi^*_h(s) = \arg\max_a Q^*_h(s,a)$$

But... didn't we arrive at Q^* by value iteration,

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

Value Iteration

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

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

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

immediate reward

future value, starting in state s' and acting ***optimally*** for $(h-1)$ steps

expected future value, weighted by the chance of landing in that particular future state s'

- Without R and T, how about: execute (s,a), observe r and s', and update:

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

target

(we'll see why this fails)

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

$\mathrm{Q}_{\text{old}}(s,a)$

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

$\mathrm{Q}_{\text{new}}(s,a)$

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

$\gamma = 0.9$

(s,a)	r	s'
$(1,\uparrow)$	0	1
$(1,\downarrow)$	0	1
	...	
$(3,\uparrow)$	1	3
$(3,\downarrow)$	1	6
	...	
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
	...	

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

$Q_{old}(s,a)$

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

$Q_{new}(s,a)$

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

$$Q_{new}(6,\uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{old}(3,a')$$

$$= -10 + 0.9 * 1 = -9.1$$

$\gamma = 0.9$

(s,a)	r	s'
	...	
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
	...	

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

$Q_{old}(s,a)$

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

$Q_{new}(s,a)$

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

$$Q_{new}(6,\uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{old}(2,a')$$

$$= -10 + 0.9 * 0 = -10$$

$\gamma = 0.9$

(s,a)	r	s'
	...	
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
	...	

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

$Q_{old}(s,a)$

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

$Q_{new}(s,a)$

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

$$Q_{new}(6,\uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{old}(3,a')$$

$$= -10 + 0.9 * 1 = -9.1$$

$\gamma = 0.9$

(s,a)	r	s'
	...	
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
	...	

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

$Q_{old}(s,a)$

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

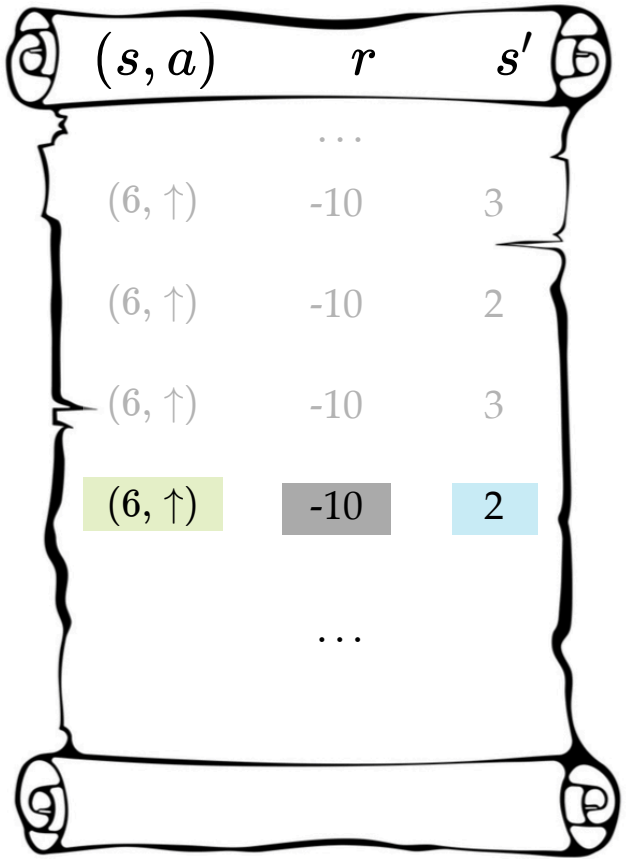
$Q_{new}(s,a)$

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

$$Q_{new}(6,\uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{old}(2,a')$$

$$= -10 + 0.9 * 0 = -10$$

$\gamma = 0.9$



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

target

Whenever observe $(6, \uparrow)$, -10, 3:

$$Q_{new}(6,\uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{old}(3,a')$$

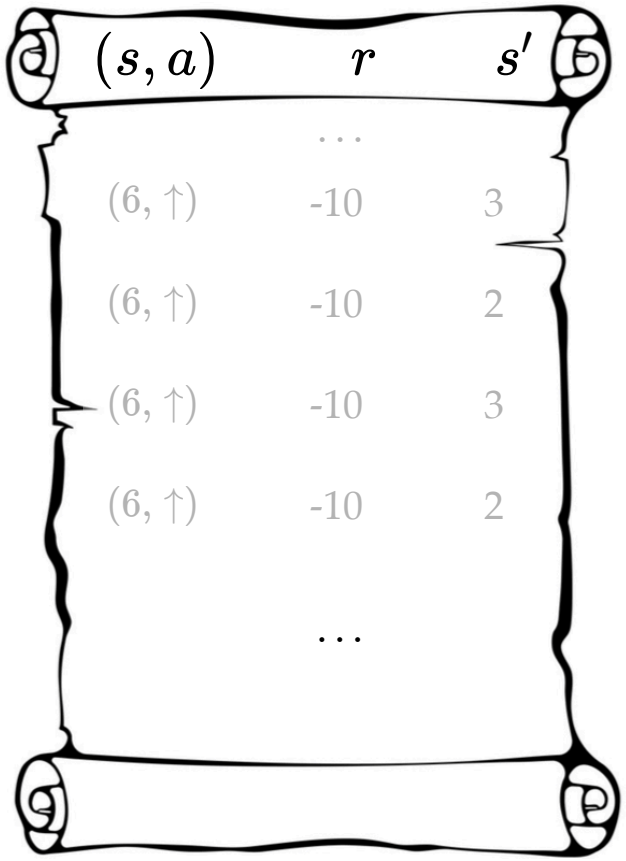
$$= -10 + 0.9 * 1 = -9.1$$

Whenever observe $(6, \uparrow)$, -10, 2:

$$Q_{new}(6,\uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{old}(2,a')$$

$$= -10 + 0.9 * 0 = -10$$

$\gamma = 0.9$



🥺 Simply committing to the new target keeps "washing away" the old belief

😍 merge *old belief* and *target*

$$(1-\text{learning rate})\ \text{old belief} \quad + \quad \text{learning rate}\ \ \text{target}$$

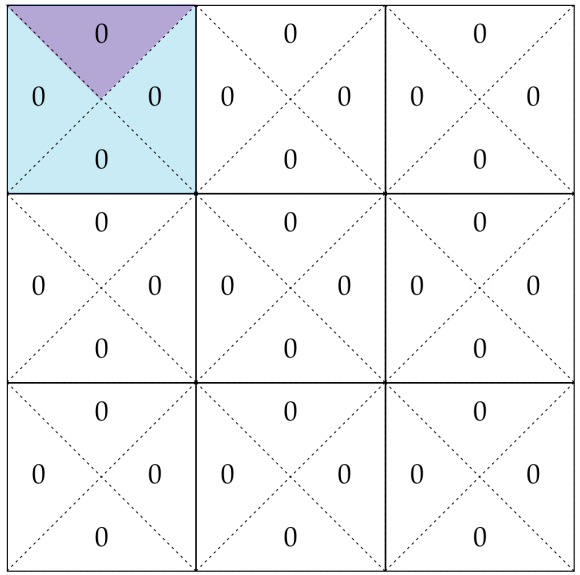
$$Q_{\text{new}}(s,a) \leftarrow (1-\alpha)\, Q_{\text{old}}(s,a) + \alpha \left(r + \gamma \max_{a'} Q_{\text{old}}(s',a') \right)$$

- Core update rule of *Q-learning*
- $\alpha \in [0,1]$: hyperparameter, trades off how much we trust new evidence versus old belief
- What we saw was $\alpha = 1$: trust new experience entirely
- Let's see an example using $\alpha = 0.7$

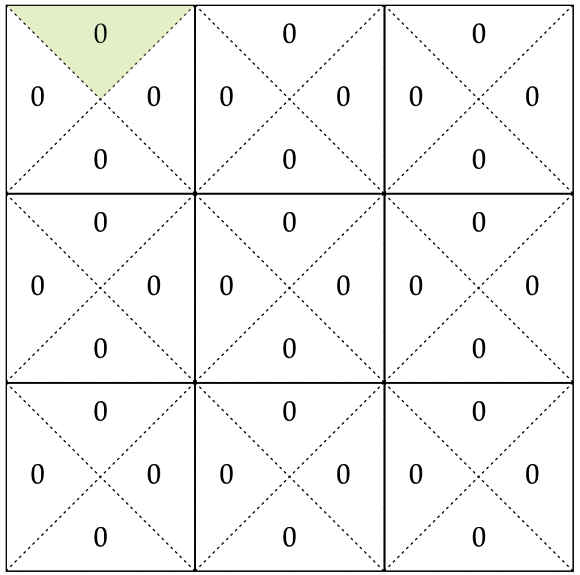
$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$\gamma = 0.9$ $\alpha = 0.7$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



(s,a)	r	s'
$(1,\uparrow)$	0	1
$(3,\uparrow)$	1	3
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2

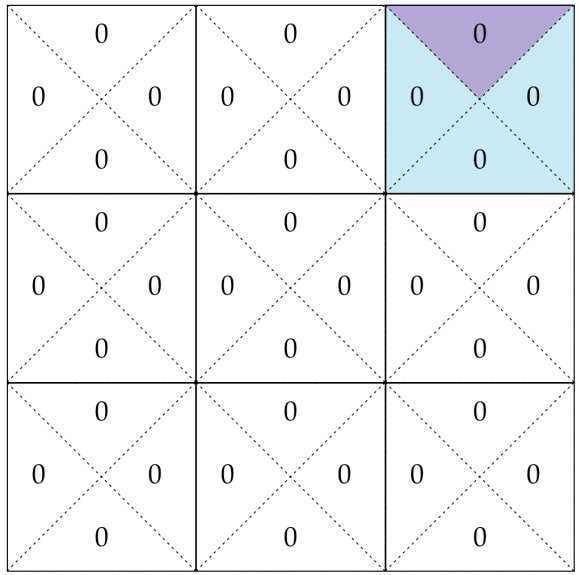
$$Q_{new}(1,\uparrow) \leftarrow (1-0.7)Q_{old}(1,\uparrow) + 0.7\left(0 + 0.9\max_{a'} Q_{old}(1,a')\right)$$

$$= (1-0.7) * 0 + 0.7 * (0 + 0.9 * 0) = 0$$

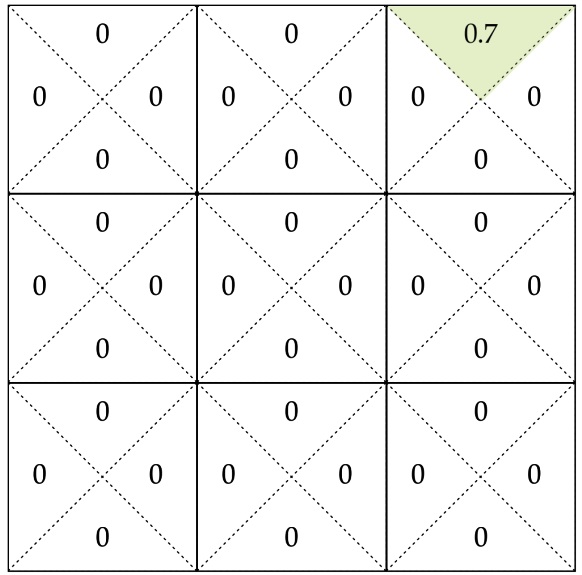
$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$\gamma = 0.9$ $\alpha = 0.7$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



(s,a)	r	s'
$(1,\uparrow)$	0	1
$(3,\uparrow)$	1	3
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2

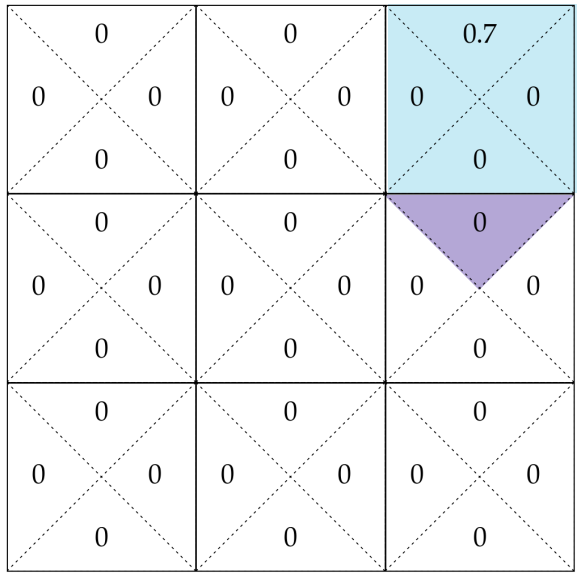
$$Q_{new}(3,\uparrow) \leftarrow (1-0.7)Q_{old}(3,\uparrow) + 0.7\left(1 + 0.9 \max_{a'} Q_{old}(3,a')\right)$$

$$= (1-0.7) * 0 + 0.7 * (1 + 0.9 * 0) = 0.7$$

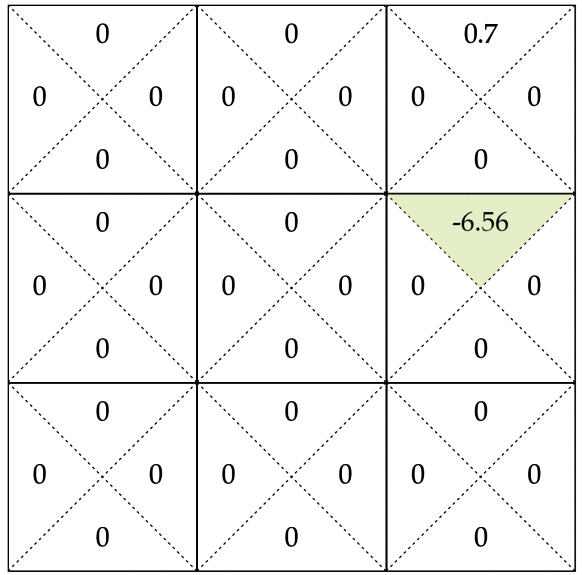
$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$\gamma = 0.9$ $\alpha = 0.7$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



(s,a)	r	s'
$(1,\uparrow)$	0	1
$(3,\uparrow)$	1	3
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2

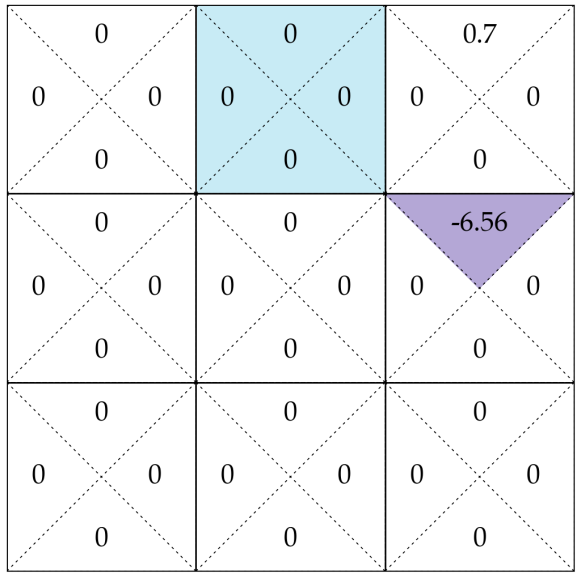
$$Q_{new}(6,\uparrow) \leftarrow (1-0.7)Q_{old}(6,\uparrow) + 0.7\left(-10 + 0.9 \max_{a'} Q_{old}(3,a')\right)$$

$$= (1-0.7) * 0 + 0.7 * (-10 + 0.9 * 0.7) = -6.56$$

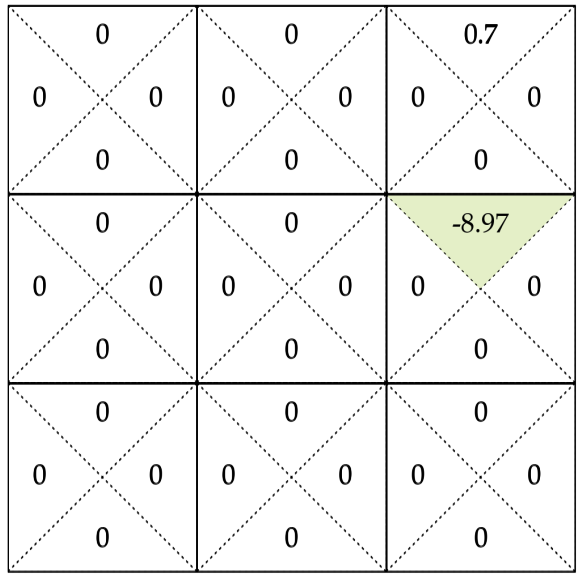
$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$\gamma = 0.9$ $\alpha = 0.7$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



(s,a)	r	s'
$(1,\uparrow)$	0	1
$(3,\uparrow)$	1	3
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2

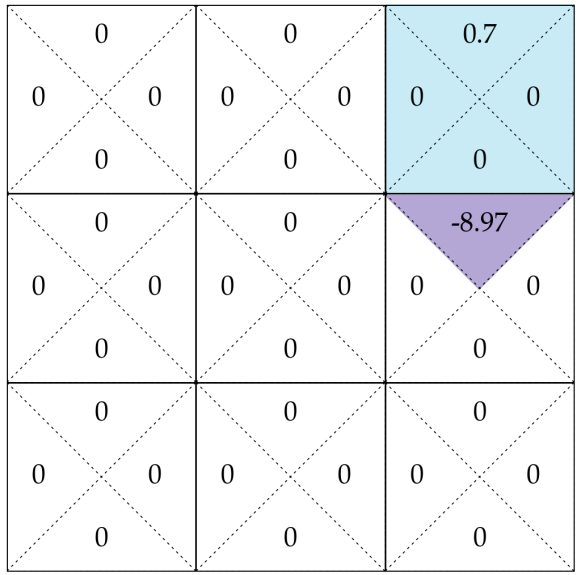
$$Q_{new}(6,\uparrow) \leftarrow (1-0.7)Q_{old}(6,\uparrow) + 0.7\left(-10 + 0.9 \max_{a'} Q_{old}(2,a')\right)$$

$$= (1-0.7) * -6.56 + 0.7 * (-10 + 0.9 * 0) = -8.97$$

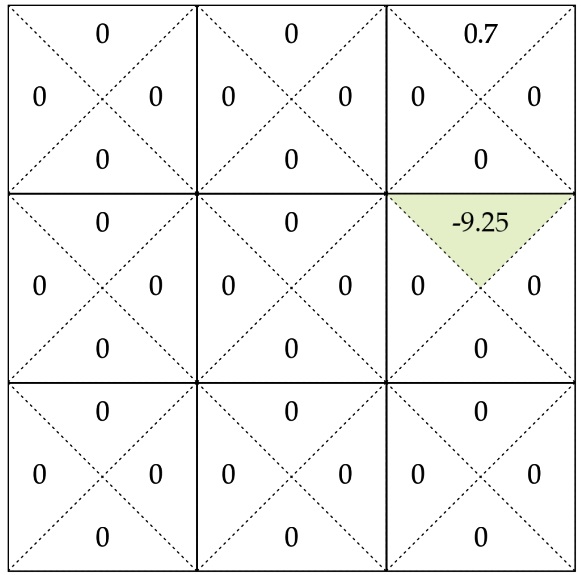
$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$\gamma = 0.9$ $\alpha = 0.7$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



(s,a)	r	s'
$(1,\uparrow)$	0	1
$(3,\uparrow)$	1	3
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2

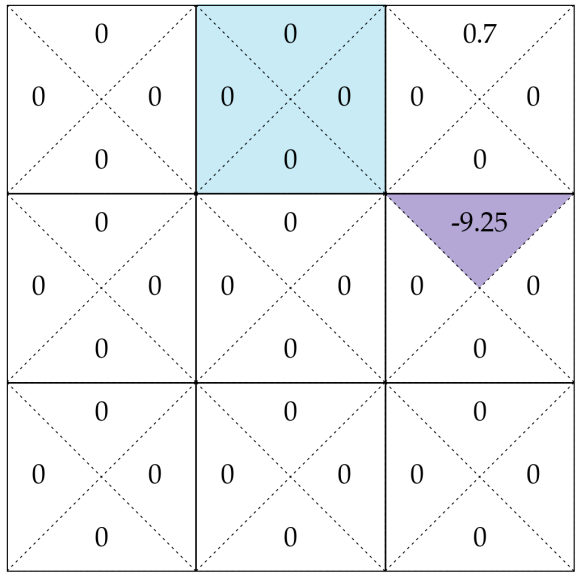
$$Q_{new}(6,\uparrow) \leftarrow (1-0.7)Q_{old}(6,\uparrow) + 0.7\left(-10 + 0.9 \max_{a'} Q_{old}(3,a')\right)$$

$$= (1-0.7) * -8.97 + 0.7 * (-10 + 0.9 * 0.7) = -9.25$$

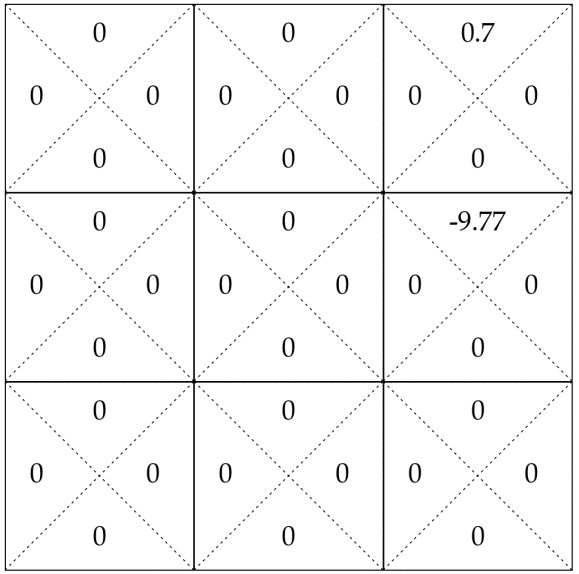
$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$\gamma = 0.9$ $\alpha = 0.7$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



(s,a)	r	s'
$(1,\uparrow)$	0	1
$(3,\uparrow)$	1	3
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2
$(6,\uparrow)$	-10	3
$(6,\uparrow)$	-10	2

$$Q_{new}(6,\uparrow) \leftarrow (1-0.7)Q_{old}(6,\uparrow) + 0.7\left(-10 + 0.9 \max_{a'} Q_{old}(2,a')\right)$$

$$= (1-0.7) * -9.25 + 0.7 * (-10 + 0.9 * 0) = -9.77$$

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

- To update a particular Q(s, a), we need experience from actually taking that a in that s.
- If we never try a move, its Q-value never changes.
- So if we have one play left, which (s,a) should we try?

Depends on

- are we trying to "win" the game, or
- are we trying to get more accurate Q estimates

This is the fundamental dilemma in reinforcement learning... and in life!

Whether to *exploit* what we've already learned or *explore* to discover something better.

FEYNMAN'S ORIGINAL RESTAURANT PROBLEM

Richard Feynman and Ralph Leighton had a hard time at restaurants deciding whether to order the best dish they had tried so far or something new, which - who knows - might be even better. So, naturally, they decided to formalize this as a mathematical problem:

Assume that a restaurant has N dishes on its menu that are rated from worst to best, 1 to
personal preferences). You, however, don't know the ratings of the dishes, and when you
learn is whether it is the best (highest rated) dish you have tried so far, or not. Each time
restaurant you either order a new dish or you order the best dish you have tried so far. Yo
the average total ratings of the dishes you eat in M meals (where M is less than or equal

Seinfeld, S06E09

The average total ratings in a sequence of meals that includes *n* "new" dishes and *b* "best
higher than the average total ratings in the sequence having all *n* "new" dishes followed
dishes. Thus a successful strategy requires you to order some number of new dishes and
best dish so far. The problem then reduces to the following:

Given N (dishes on the menu) and M <= N (meals to be eaten at the restaurant), how many new dishes D should you try before switching to ordering the best of them for all the remaining (M–D) meals, in order to maximize the average total ratings of the dishes consumed?

[feynmanlectures.caltech.edu]

Exploration has real costs: A/B tests give real users bad experiences, medical trials risk patient safety, self-driving cars risk lives.

ϵ-greedy action selection strategy

- with probability ϵ, choose an action $a \in \mathcal{A}$ uniformly at random

  During learning, especially in early stages, we'd like to explore and observe diverse (s, a) consequences.

- with probability $(1 - \epsilon)$, choose $\arg\max_{\mathrm{a}} \mathrm{Q}_{\mathrm{old}}(s, \mathrm{a})$

  During later stages, we can act more greedily w.r.t. the current estimated Q values

ϵ trades off *exploration* versus *exploitation*.

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

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

"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. $\quad \mathrm{Q}_{\text{old}}(s, a) = 0$
4. $s \leftarrow s_0$
5. **while** $i <$ max-iter :
6. $\quad a \leftarrow \text{select\_action}(s, \mathrm{Q}_{\text{old}}(s, a))$
7. $\quad r, s' \leftarrow \text{execute}(a)$
8. $\quad \mathrm{Q}_{\text{new}}(s, a) \leftarrow (1 - \alpha)\mathrm{Q}_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} \mathrm{Q}_{\text{old}}(s', a'))$
9. $\quad s \leftarrow s'$
10. $\quad i \leftarrow (i + 1)$
11. $\quad \mathrm{Q}_{\text{old}} \leftarrow \mathrm{Q}_{\text{new}}$
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. $\quad \mathrm{Q}_{\text{old}}(s, a) = 0$
4. $s \leftarrow s_0$
5. **while** $i <$ max-iter :
6. $\quad a \leftarrow \text{select\_action}(s, \mathrm{Q}_{\text{old}}(s, a))$
7. $\quad r, s' \leftarrow \text{execute}(a)$
8. $\quad \mathrm{Q}_{\text{new}}(s, a) \leftarrow (1-\alpha)\mathrm{Q}_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} \mathrm{Q}_{\text{old}}(s', a'))$
9. $\quad s \leftarrow s'$
10. $\quad i \leftarrow (i+1)$
11. $\quad \mathrm{Q}_{\text{old}} \leftarrow \mathrm{Q}_{\text{new}}$
12. **return** Q_{new}

"learning"

- Remarkably, 👈 *can* converge to the true Q^*_∞
- Once converged, act greedily w.r.t Q^* again.
- But convergence can be extremely slow,

  and often not practical.
- Three hyperparameters :
  - discount factor γ
  - learning rate α
  - exploration rate ϵ

all between 0 and 1

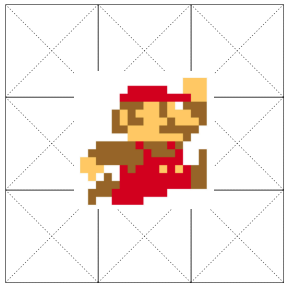
each controls some trade-off

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

Outline

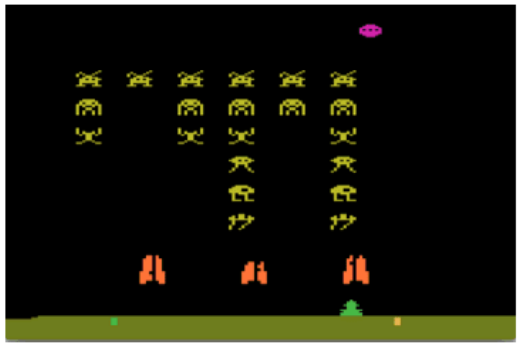
- Reinforcement learning setup
- Tabular Q-learning
  - exploration vs. exploitation
  - ϵ-greedy action selection
- Fitted Q-learning
- (Policy gradient)

Fitted Q-learning: from table to functions

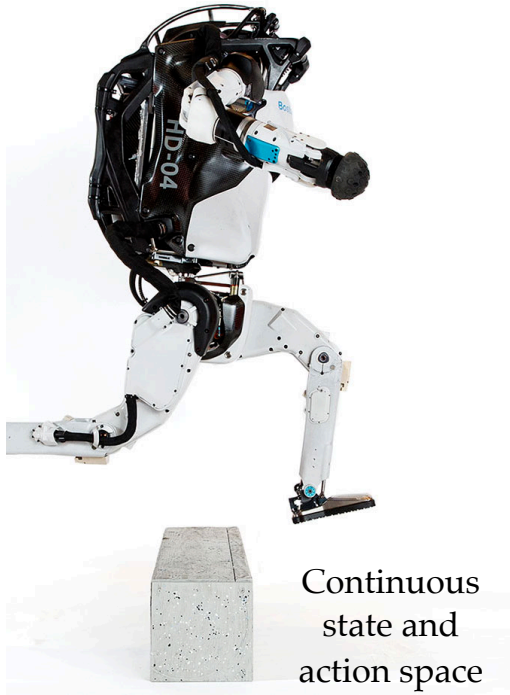


- So far, Q-learning is sensible for the (small) tabular setting.
- What if $\mathcal{S}$ and/or $\mathcal{A}$ are large, or even continuous?

Game	Board size	State space
Go	19 x 19	10^{172}
Chess	8 x 8	10^{50}



10^{16992} (pixels) states



Continuous state and action space

- We can't keep a table; we must approximate $Q(s, a)$ with a function $Q_\theta(s, a)$.

- Notice that the core update rule of Q-learning:

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

  is equivalent to:

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

- Does this

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

  remind you of something?

- Yes! Recall:

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} + \eta(\text{target} - \text{guess}_\theta)\nabla_\theta \text{guess}$$

Gradient update rule when minimizing $(\text{target} - \text{guess}_\theta)^2$

- Same idea as before — we adjust our belief toward a target.

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

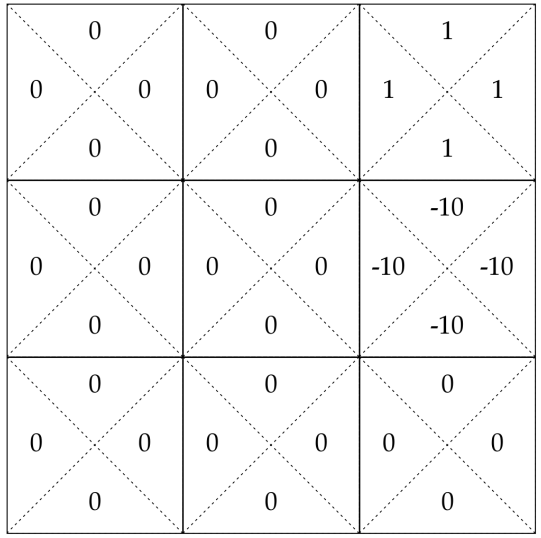
$$\text{new belief} \leftarrow \text{old belief} + \text{learning rate}\,(\text{target} - \text{old belief})$$

- Now Q is a function (e.g. a neural network) and θ are its weights.
- Generalize tabular Q-learning for continuous state/action space:

1. parameterize $\mathrm{Q}_\theta(s,a)$
2. execute (s,a), observe(r,s'), construct the target $r + \gamma \max_{a'} \mathrm{Q}_\theta(s',a')$
3. regress $\mathrm{Q}_\theta(s,a)$ against the target, i.e. update θ via gradient-descent to minimize

$$(\text{target} - \mathrm{Q}_\theta(s,a))^2$$

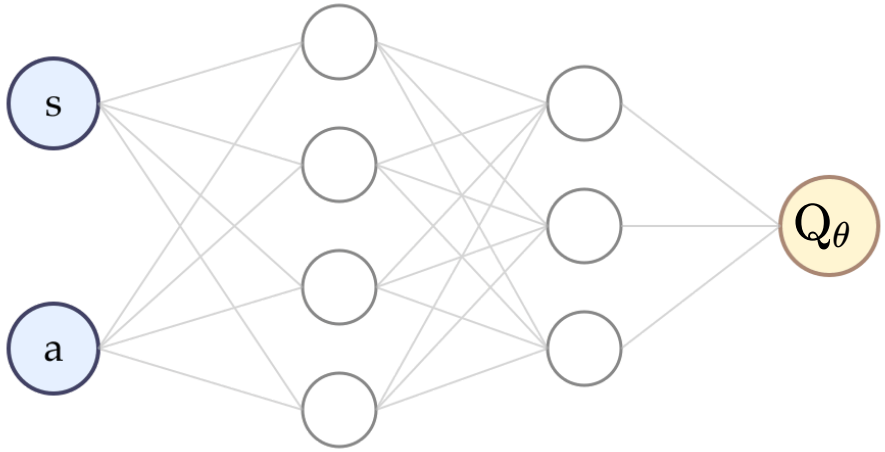
From table to functions



tabular: a lookup, one entry per (s, a)

$\Longrightarrow$

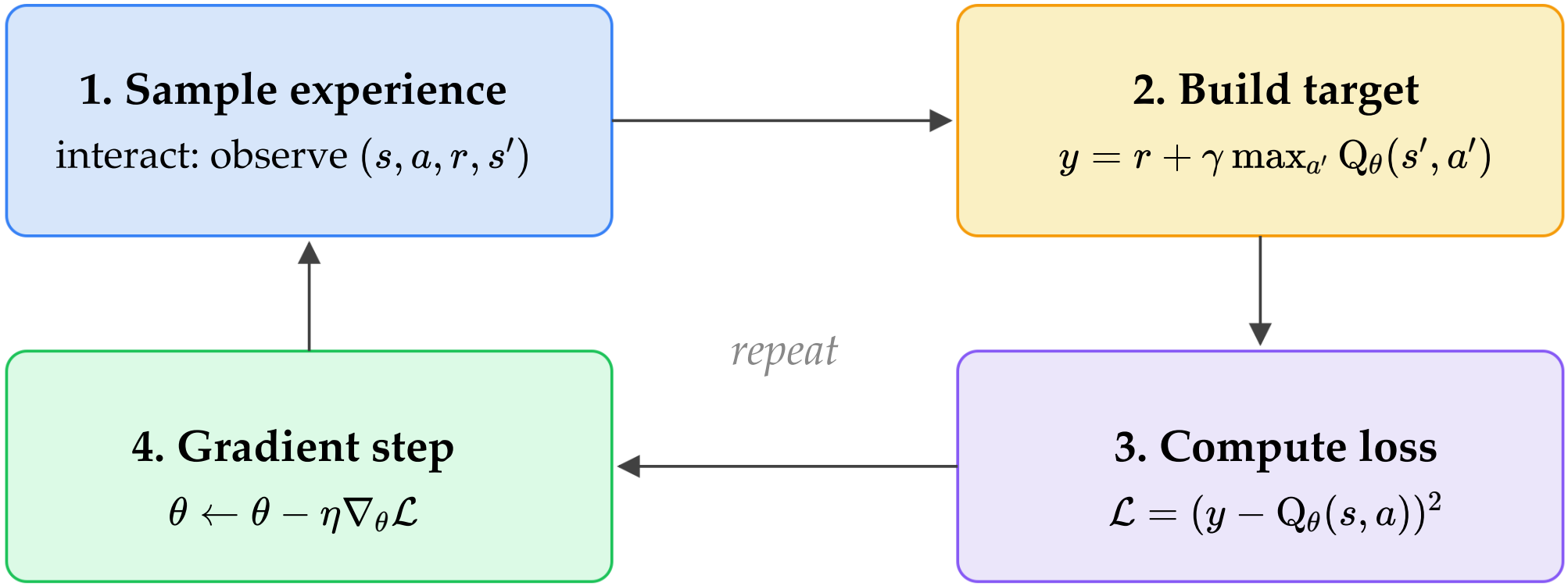
scale up:
large / continuous $\mathcal{S}$, $\mathcal{A}$



fitted: any (s, a) goes through one shared θ

Updating one (s, a) now nudges θ, which changes Q_θ at *nearby* (s, a) too.
The function generalizes; the table doesn't.

Fitted Q-learning: the training loop



Same shape as supervised learning; except the "label" y is built from our own current estimate. (Bootstrapping.)

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Rie**

DeepMind Technologie

{vlad,koray,david,alex.graves,ioannis,daan,mart

Abstract

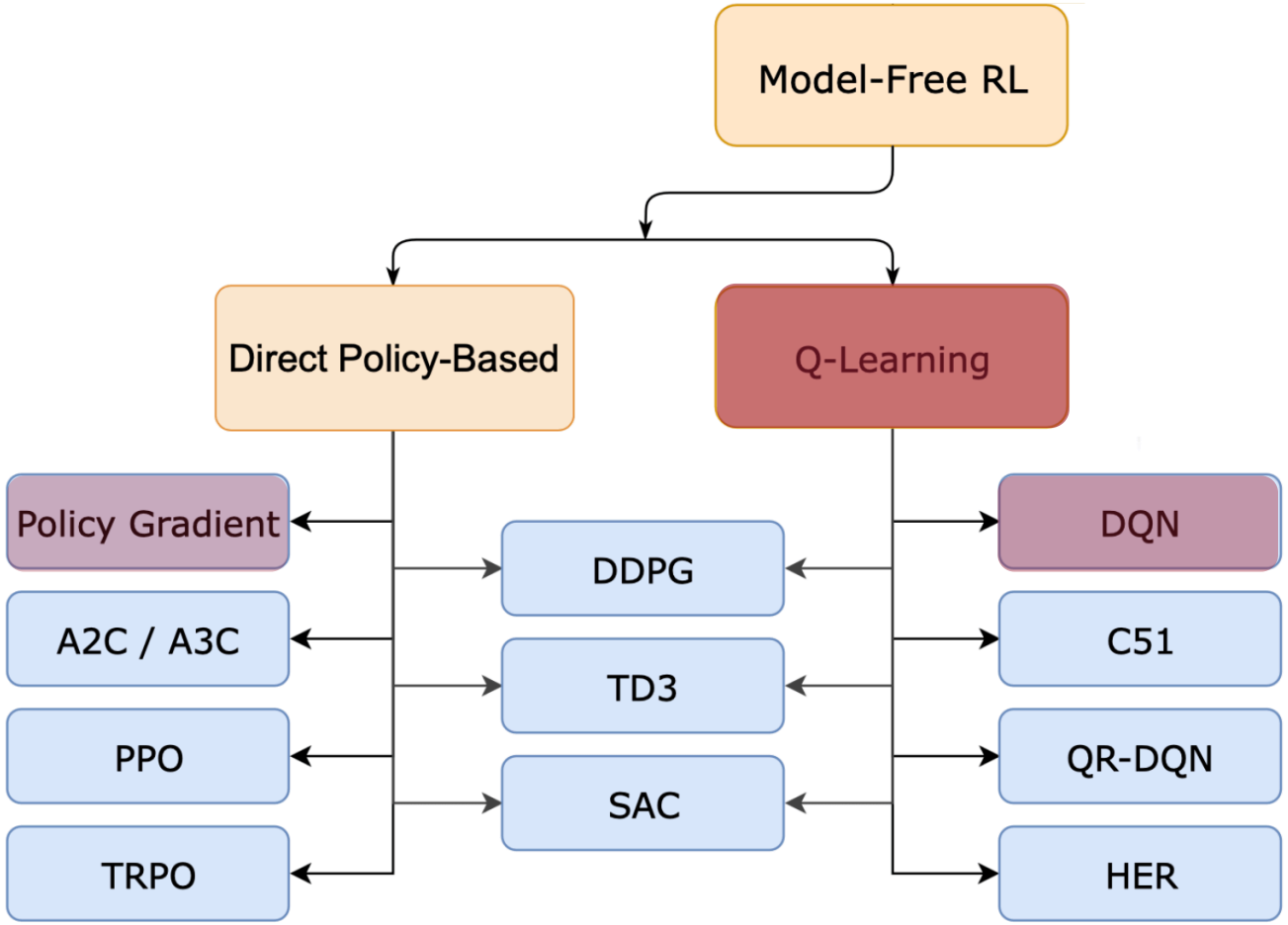
We present the first deep learning model to successfully
rectly from high-dimensional sensory input using reinf
model is a convolutional neural network, trained with
whose input is raw pixels and whose output is a value fu
rewards. We apply our method to seven Atari 2600 game
ing Environment, with no adjustment of the architecture c
find that it outperforms all previous approaches on six of
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**
  Initialise 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

- Reinforcement learning setup
- Tabular Q-learning
  - exploration vs. exploitation
  - ϵ-greedy action selection
- Fitted Q-learning
- (Policy gradient)



Recall:

current prediction
$g = \text{softmax}(\cdot)$

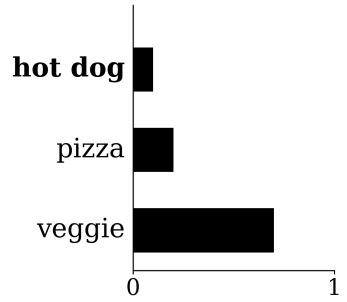
$\xrightarrow{-\log}$

$-\log(g)$

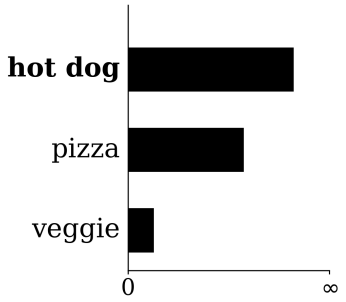
y (true label)

$$\mathcal{L}_{\text{nllm}} = -\sum_k y_k \cdot \log(g_k)$$

$\xrightarrow{h_\theta}$

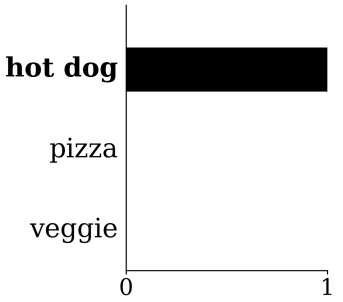


$= [0.1,\ 0.2,\ 0.7]$

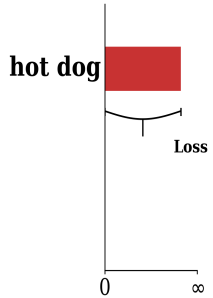


$\approx [2.30,\ 1.61,\ 0.36]$

$\odot$



$= [1,\ 0,\ 0]$



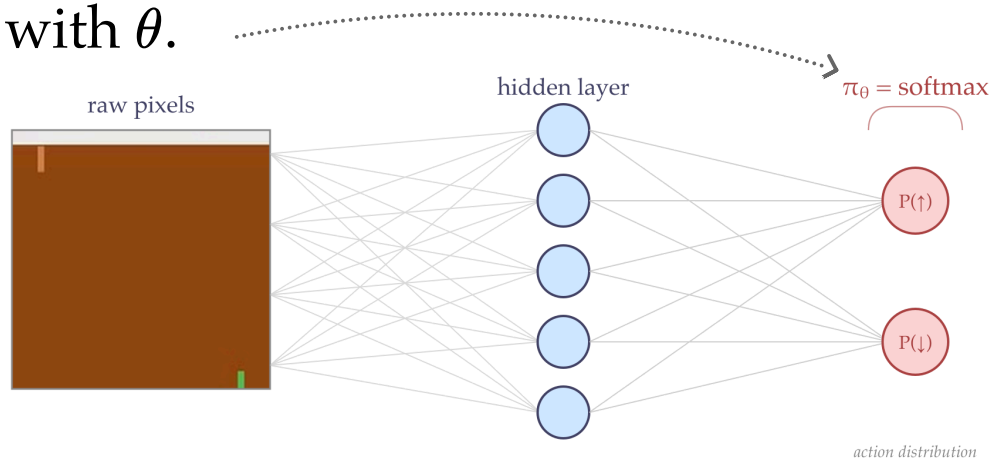
Loss ≈ 2.30

To reduce the loss, θ needs to change so g_{hotdog} can go up.

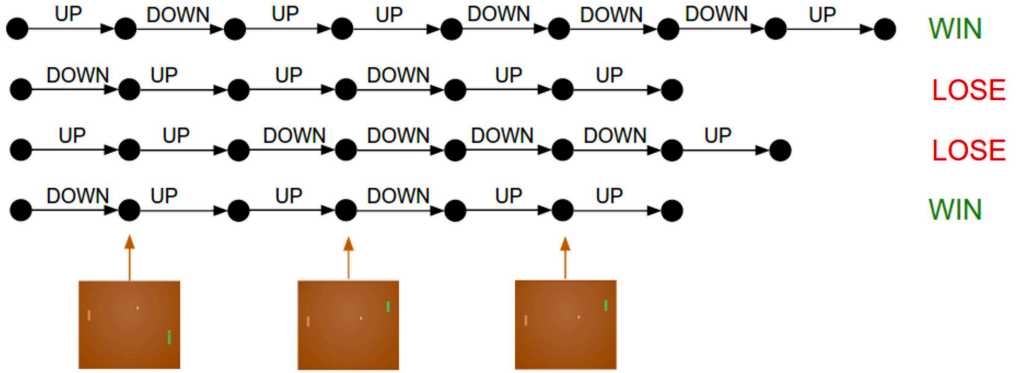
This signal (from the true label y) backprop to θ through $-\log$ and softmax, so we can optimize via gradient descent.

Policy gradients:

1. Parametrize the policy π with θ.



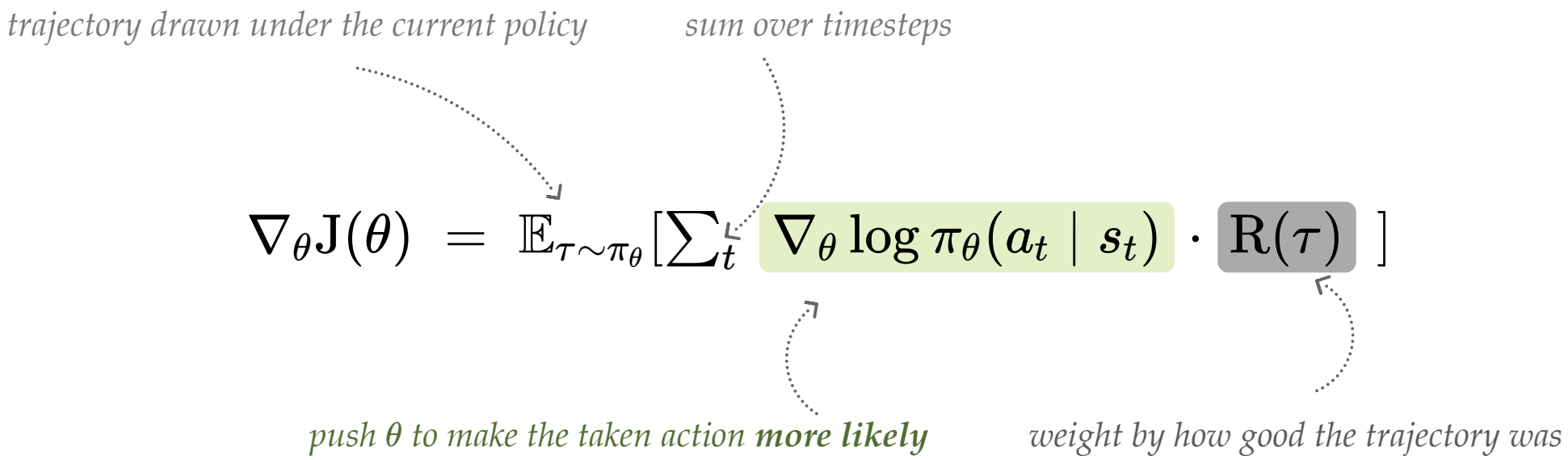
2. Run π_θ for a while. Keep track of the return.



3. Update θ so "good" trajectories more likely to happen.

[image edited from: Andrej Karpathy]

Made rigorous by the *log-derivative trick* (REINFORCE). We will skip the derivation.



Sample trajectories, observe returns, nudge θ to make actions in *good* trajectories more likely.

Y. LeCun

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**

[Slide Credit: Yann LeCun]

Summary

- Last week: find good policies in a *known* MDP (high cumulative expected reward).
- In RL, the transitions and rewards are *unknown*. We still want a good policy; we get one by estimating Q.
- Exploration vs. exploitation: choose actions to gain reward while still learning.
- Q-learning converges to Q^* (with enough exploration).
- Deep Q-learning extends to large or continuous state spaces by parameterizing Q.