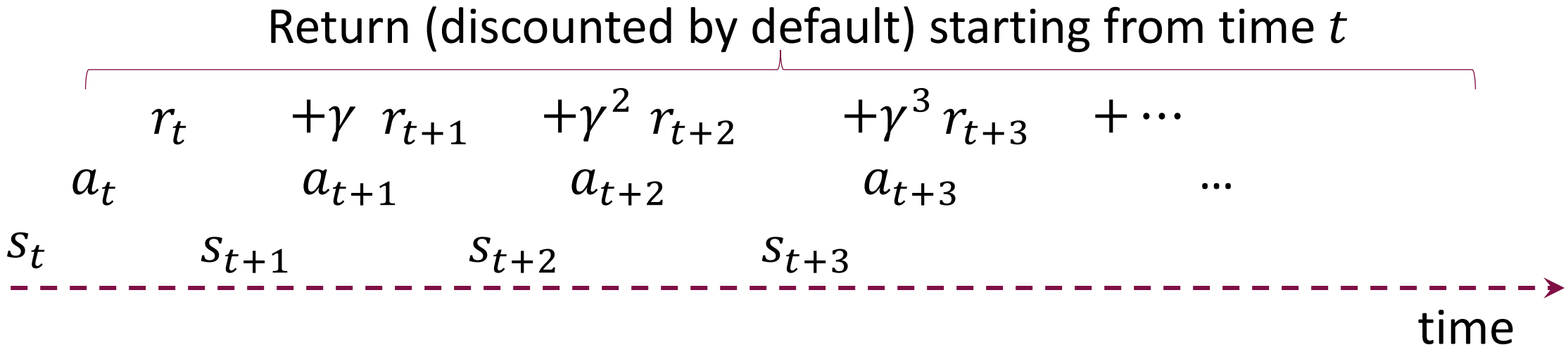
CIS 4190/5190: Lec 18 Wed Nov 06, 2024

Reinforcement Learning Part 2

(Discounted) Return



Suppose that:

- this trajectory was produced by following a policy π
- π and the environment transitions $P(s'|s,a)$ are both deterministic.

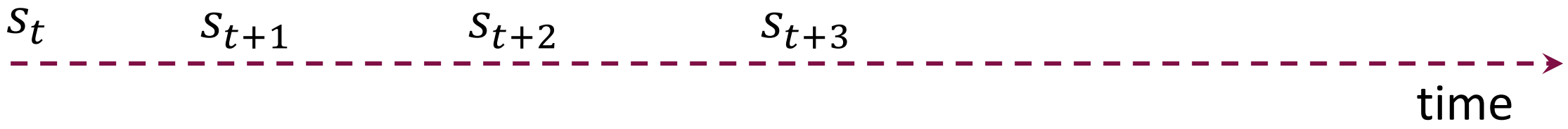
Value Function

Suppose that:

- this trajectory was produced by following a policy π
- π and the environment transitions $P(s'|s,a)$ are both deterministic.

Then this return is also the *"value"* of the first state according to the policy (otherwise value is the *expected* return, i.e. averaged over many such trajectories)

$$V^{\pi}(s_t) = \underset{a_t}{r_t} + \underset{a_{t+1}}{\gamma\, r_{t+1}} + \underset{a_{t+2}}{\gamma^2\, r_{t+2}} + \underset{a_{t+3}}{\gamma^3\, r_{t+3}} + \underset{\dots}{\cdots}$$



Demoing Bellman Equation for Deterministic System

(Policy and transitions are both deterministic)

$$V^{\pi}(s_t) = \quad r_t \quad +\gamma\ r_{t+1} \quad +\gamma^2\ r_{t+2} \quad +\gamma^3\, r_{t+3} \quad +\cdots$$

$$V^{\pi}(s_{t+1}) = \quad r_{t+1} \quad +\gamma\ r_{t+2} \quad +\gamma^2\ r_{t+3} \quad +\gamma^3\ r_{t+4}$$

$$V^{\pi}(s_t) \ = \ r_t \quad +\gamma\, V^{\pi}(s_{t+1})$$

Bellman equation is “just” good bookkeeping.
Value is neither created nor destroyed.

Demoing Bellman Equation for Deterministic System

$$V^{\pi}(s_t) = r_t + \gamma V^{\pi}(s_{t+1})$$

When policy is optimal, this produces the special case:

$$V^{\pi^*}(s_t) = r_t + \gamma V^{\pi^*}(s_{t+1})$$

$$V^*(s_t) = \max_{a_t} [r_t + \gamma V^*(s_{t+1})]$$

(since π^* is optimal, it must maximize return starting from any s_t)

When non-deterministic, Bellman Equation for V^{π} simply becomes:

$$V^{\pi}(s_t) = \mathbb{E}_{s_{t+1} \sim P(s'|s,a), a_t \sim \pi(a|s_t)} [r_t + \gamma V^{\pi}(s_{t+1})]$$

Exercise: work through this same reasoning for Q value functions $Q(s_t, a_t)$.

Coming up next

- How to compute Q^* if we knew the full MDP (S, A, P, R, γ)?
  - "Q-policy and Q-value iteration". (Also briefly V-value iteration)

} Not RL!

- How to *learn* Q^* from experience if we only had (S, A, γ) and didn't know P, R?
  - "Q learning",
    - One of the first RL algorithms that got successfully implemented in deep neural networks
    - Closely connected to the Bellman Equation for Q^*
    - Still widely used in many domains

Finding Q^* & π^* in "known environments":

Q Policy Iteration & Q Value Iteration

Q-Policy Iteration (Known $\boldsymbol{P}$ and $\boldsymbol{R}$)

Key Idea: To find Q^*, solve iteratively via dynamic programming

- Start with a random guess, e.g.,$Q_0^*(s,a) \leftarrow 0$ for all states s and actions a
- Iterate (incrementing i, till convergence):
  1. **Policy Improvement:**
     - For all s: Update the guess for π^* to be compatible with Q_i
     - $\pi_i(s) \leftarrow \underset{a}{\operatorname{argmax}}\, Q_i(s,a)$
  2. **Policy Evaluation:**
     - For all s, a: Update your guess for Q^* to be compatible with π_i:
       - $Q_{i+1}(s,a) \leftarrow Q^{\pi_i}(s,a)$

usually easy to do

how to compute?

Dealing with Step 2: Q-Policy Evaluation

How do we calculate the $Q^{\pi}(s,a)$ for some policy $\pi(s)$?

- Recall, Bellman Equation gives us a rule that all Q functions must obey:

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q^{\pi}(s', \pi(s'))]$$

- Idea: **convert the Bellman equation for $Q^{\pi}(s,a)$ into an update rule**

For all (s,a)

$$Q_0^{\pi}(s,a) \leftarrow 0 \text{ (could also be initialized differently)}$$

$$Q_{j+1}^{\pi}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q_j^{\pi}(s', \pi(s'))]$$

Lots of theory that explains why the "fixed point" that this recursive update procedure converges to is indeed the correct Q^{π}. We will skip this.

Putting it together: Q-Policy Iteration & Q-Value Iteration

- Start with a random guess, e.g., $Q_0^*(s,a) \leftarrow 0$ for all states s and actions a
- Iterate (incrementing i, till convergence):
  1. **Policy Improvement:** (For all s)
     - Compute the corresponding policy $\pi_i^*(s) \leftarrow \underset{a}{\text{argmax}}\, Q_i^*(s,a)$
  2. **Policy Evaluation:** (For all s, a)
     - Iterate (incrementing j, till convergence?):
       - $Q_j^{\pi_i^*}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q_{j-1}^{\pi_i}(s', \pi_i(s'))]$

Can also run only a single update: "Q value iteration", or just "Q iteration"

More concise expression for Q-value iteration:

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

$$\updownarrow$$

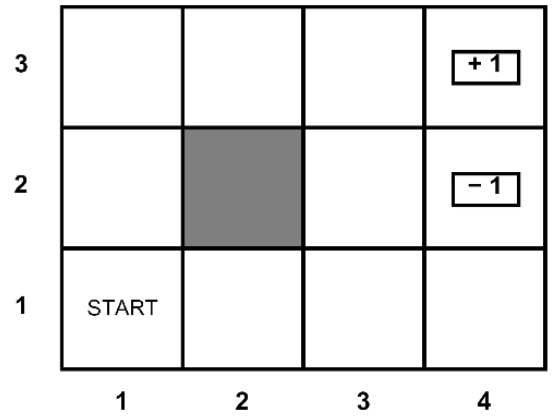
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

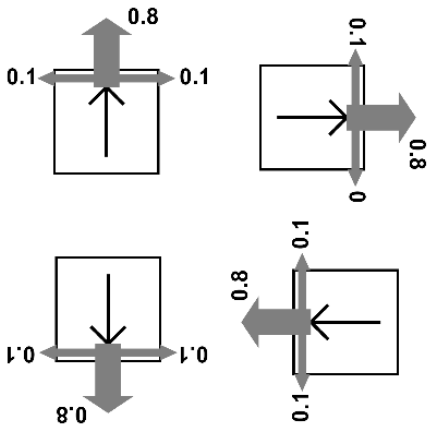
Bellman Equation for $Q^*(s,a)$ converted to an update rule!

Example of Q-Iteration (by default Q-*Value* Iteration)

Behind The Scenes: The Full Environment

- A grid map with solid / open cells. Agent('s dot) moves between open cells.
- From terminal states $(4,3)$ and $(4,2)$, any action ends the episode, and results in a +1/-1 reward respectively.
- For each timestep outside terminal states , the agent pays a small "living" cost (negative reward): -0.03
- The agent actions N, E, S, W correspond to North, East, South, West
  - **But the outcomes of actions are not deterministic!**
    - The dot obeys the commanded motion direction 80% of the time
    - 10% of the time, the dot instead executes a different direction 90° off from the agent command. Another 10% of the time, -90° off.
    - E.g. if dot surrounded by open cells and executing action N, will end up in the northern cell 80% of the time, in the eastern cell 10% of the time, and in the western cell 10% of the time.
  - **The dot stays put if it attempts to move into a solid cell** or outside the world. (Imagine the map is surrounded by solid cells)
- Goal: As always, maximize the sum of discounted future rewards within an episode

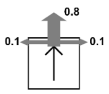




Based on slide by Dan Klein

Q-Iteration example

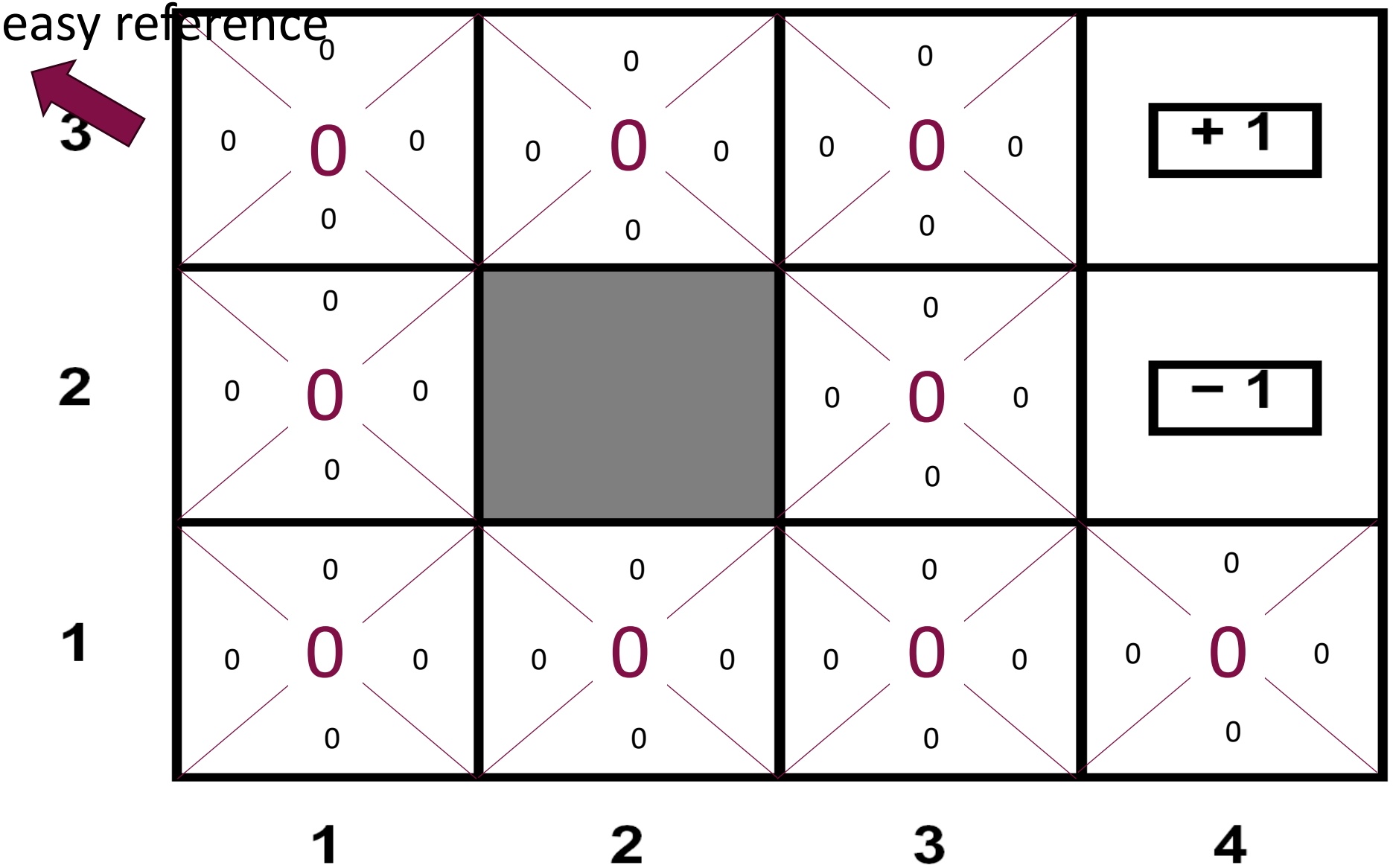
Store maxQ in each cell for easy reference



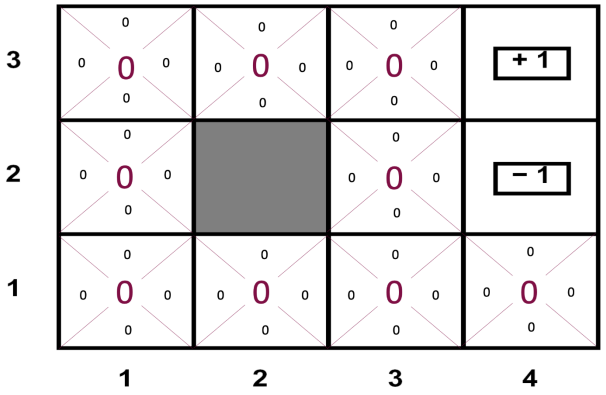
Living cost 0

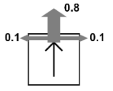
0.9

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$



Q-Iteration example





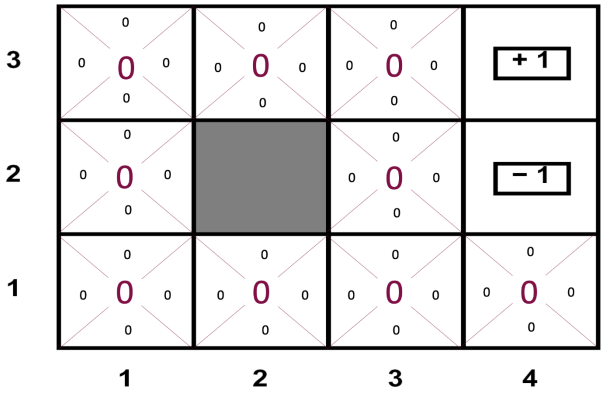
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

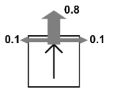
0

0.9

0.8x[0+0.9x1]
+ 0.1x[0 + 0]
+0.1x[0+0]
=0.72

Q-Iteration example

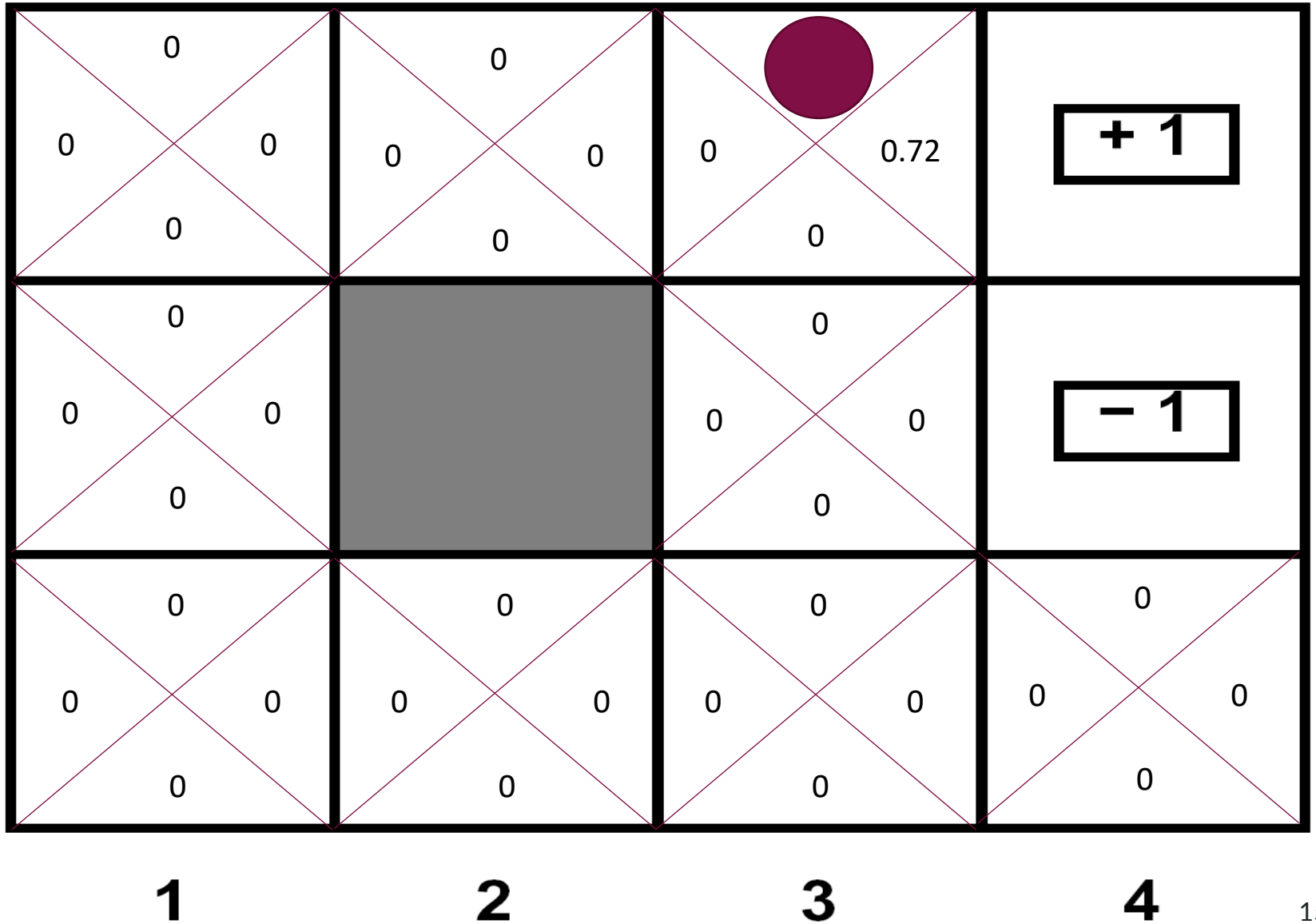




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

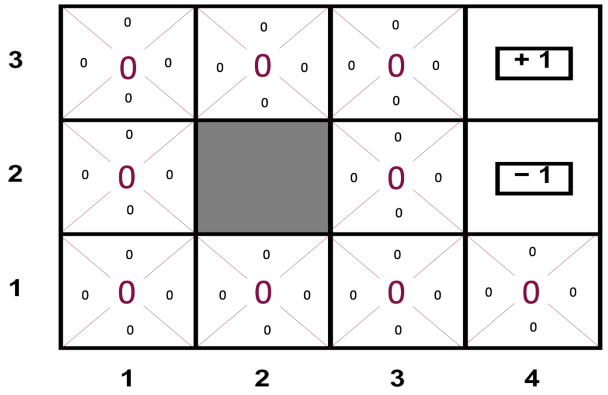
0

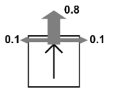
0.9



0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09

Q-Iteration example

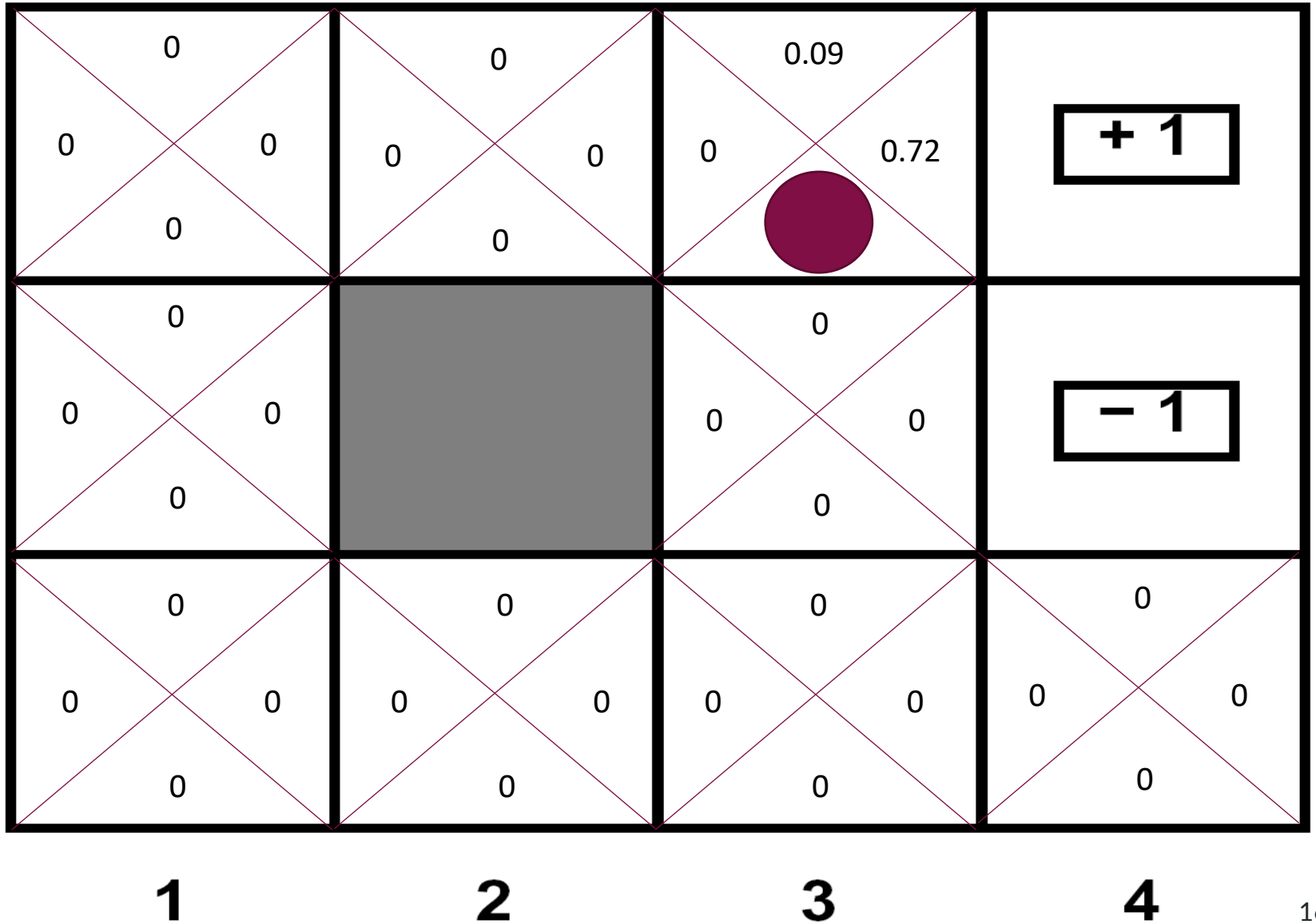




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

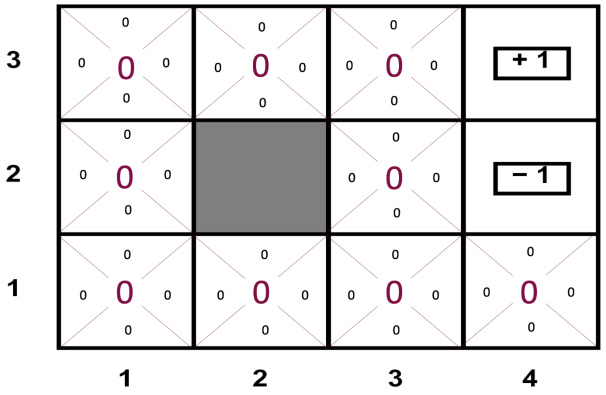
0

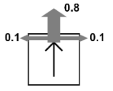
0.9



0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09

Q-Iteration example

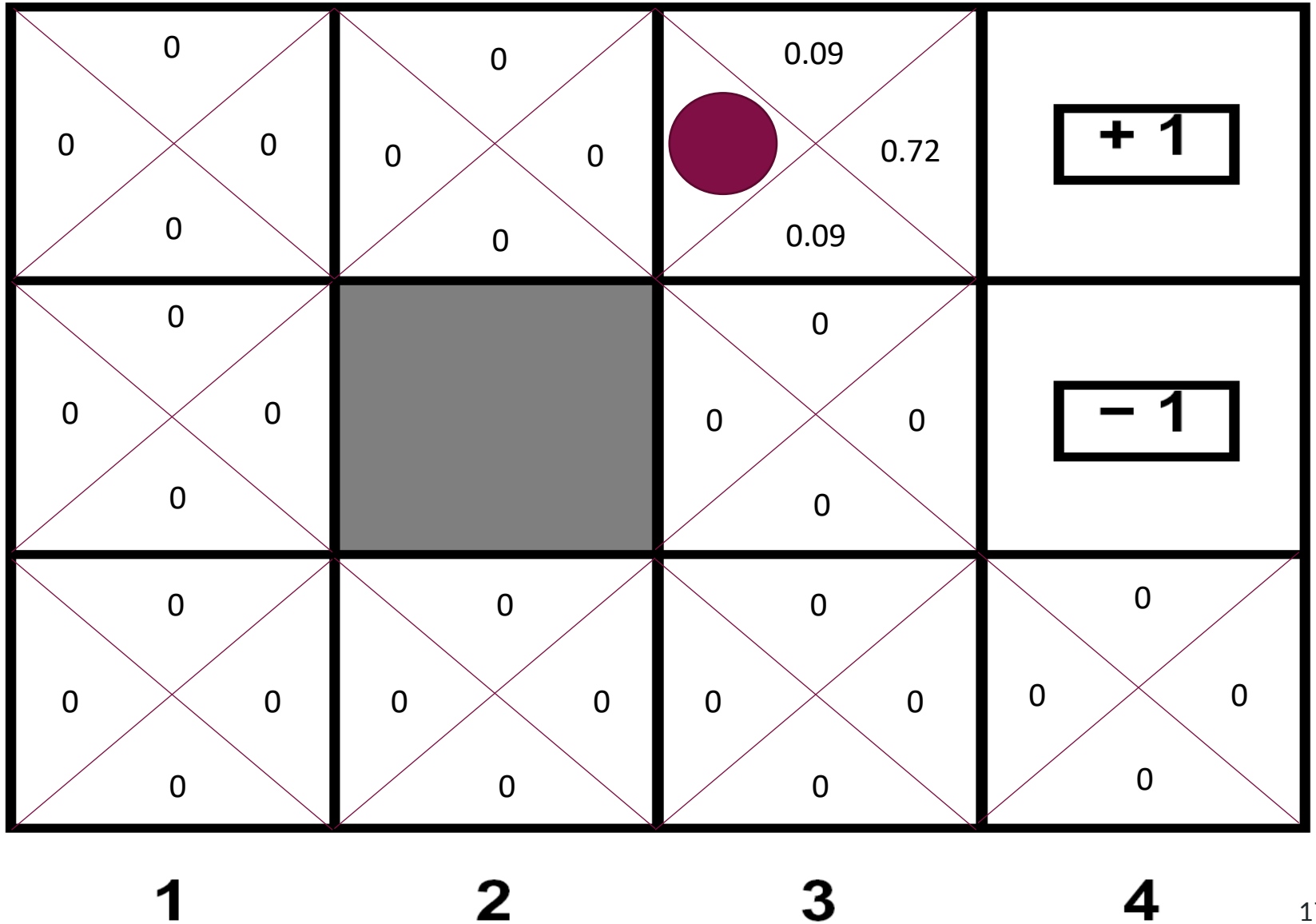




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

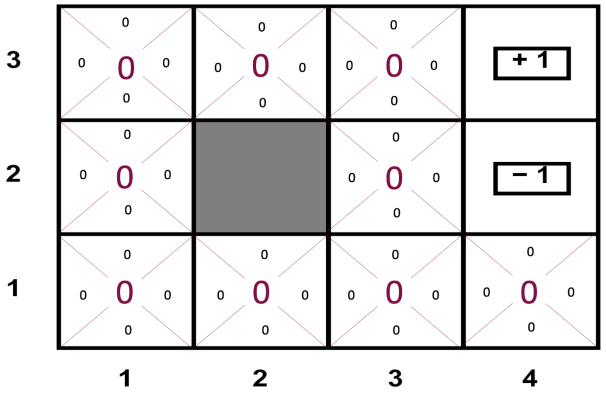
0

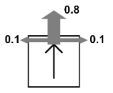
0.9



0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0]
=0

Q-Iteration example

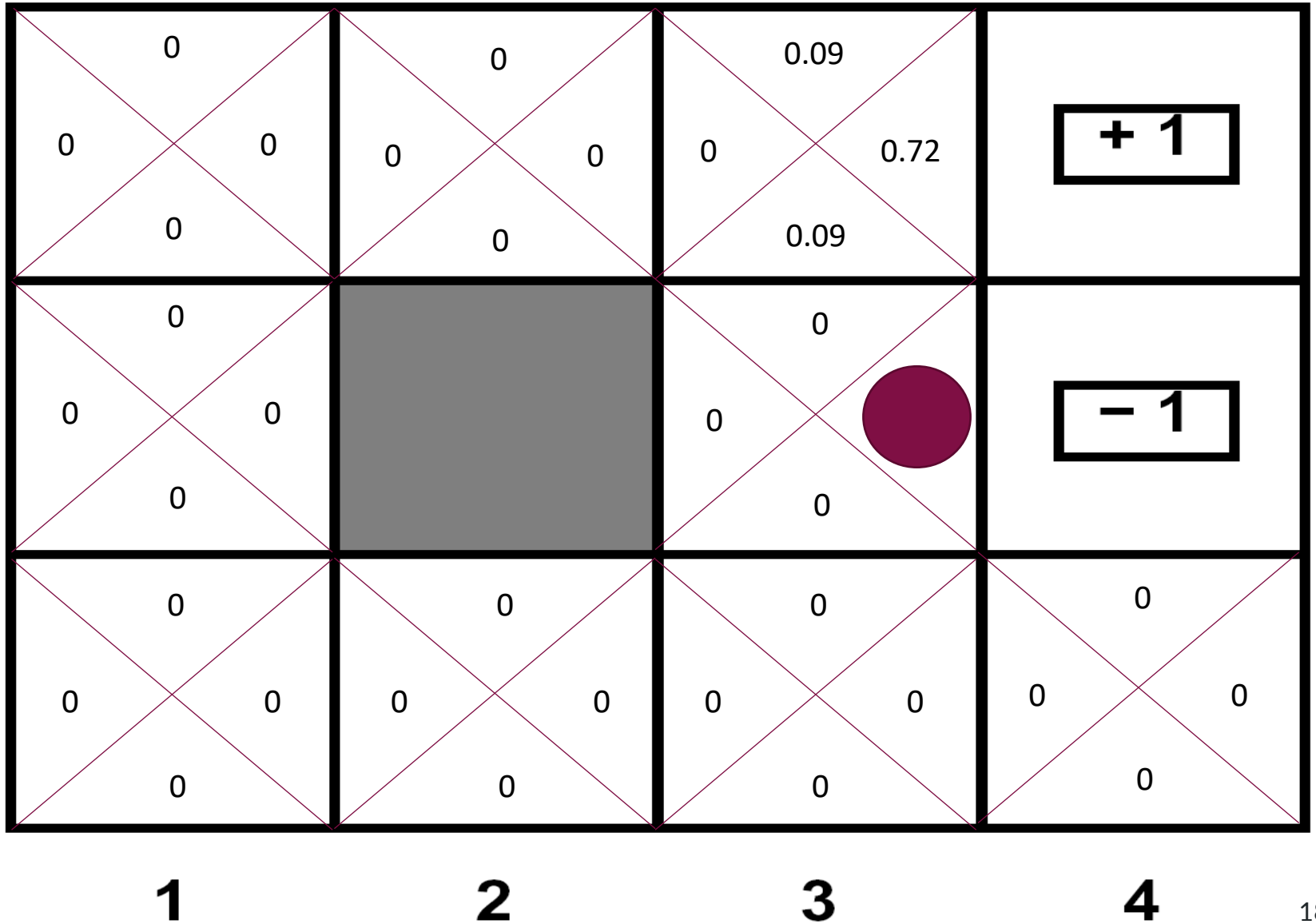




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

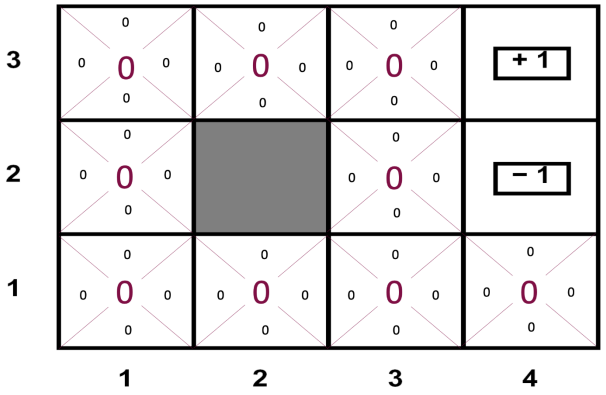
0

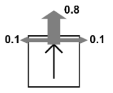
0.9



0.8x[0+0.9x-1]
+ 0.1x[0+0]
+0.1x[0+0]
=-0.72

Q-Iteration example

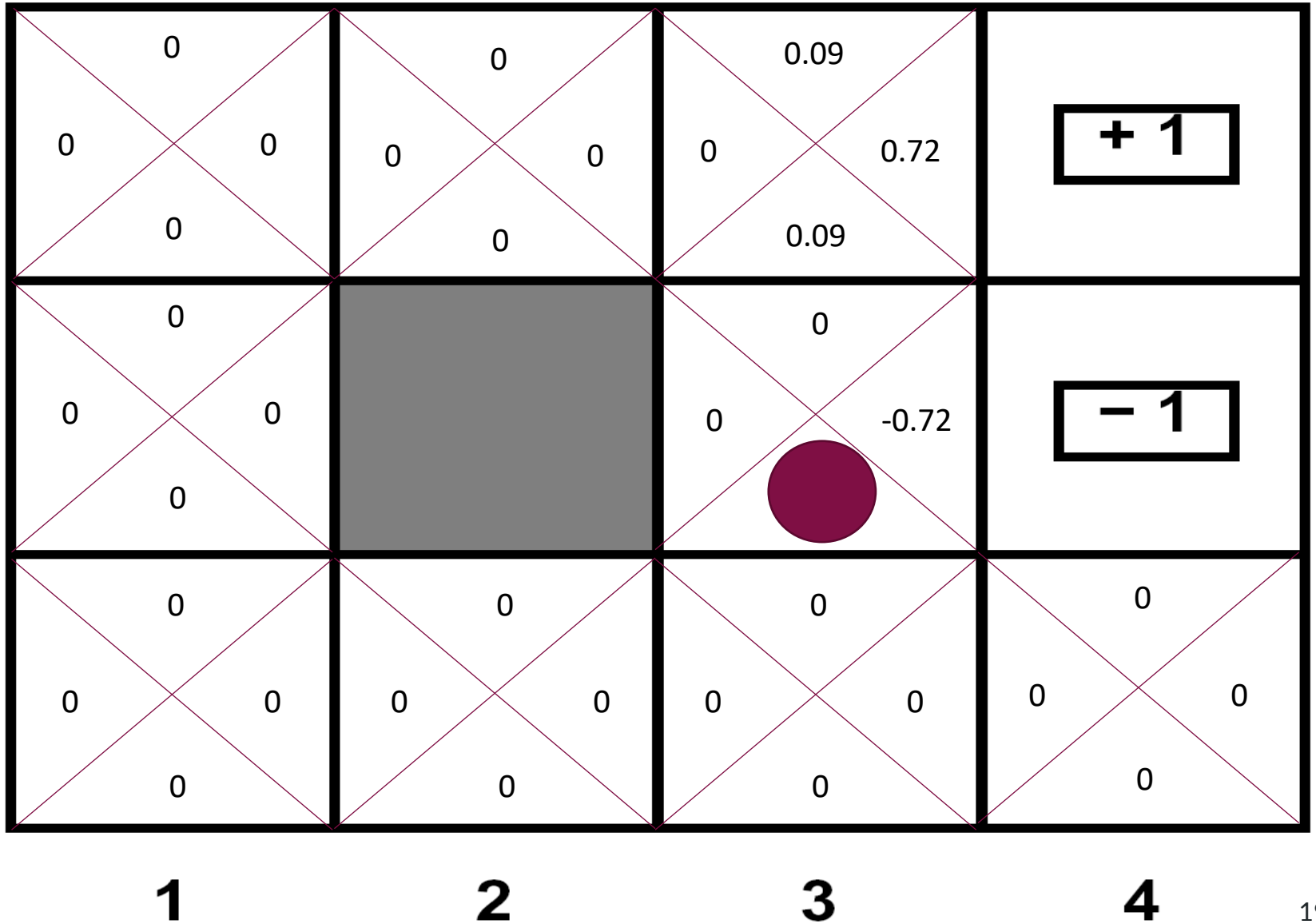




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

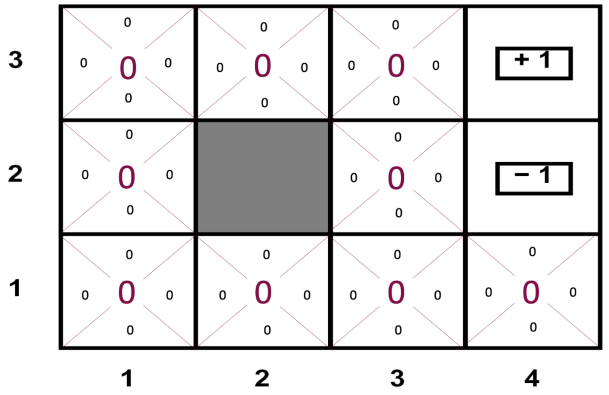
0

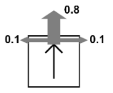
0.9



0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0.9x-1]
=-0.09

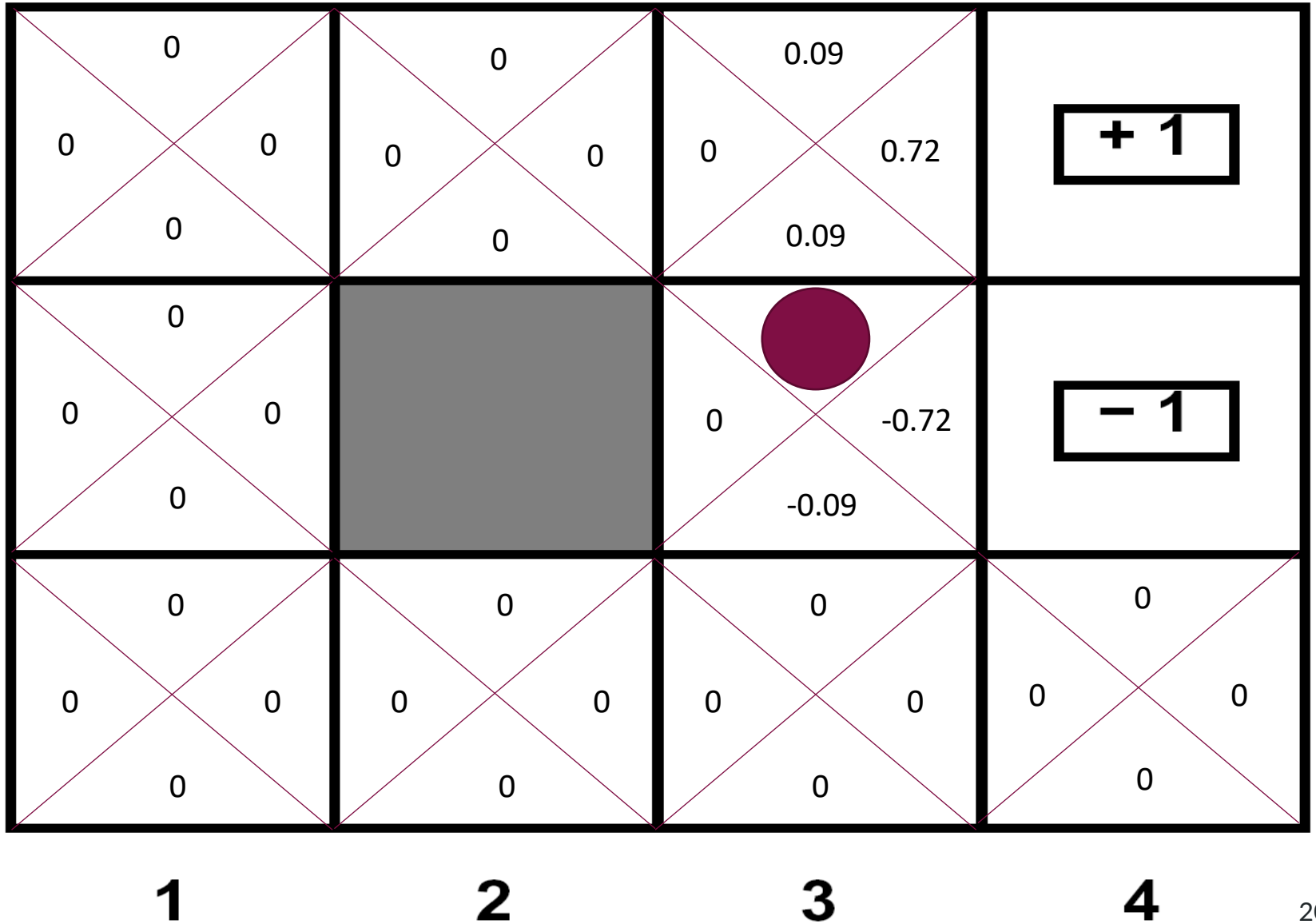
Q-Iteration example





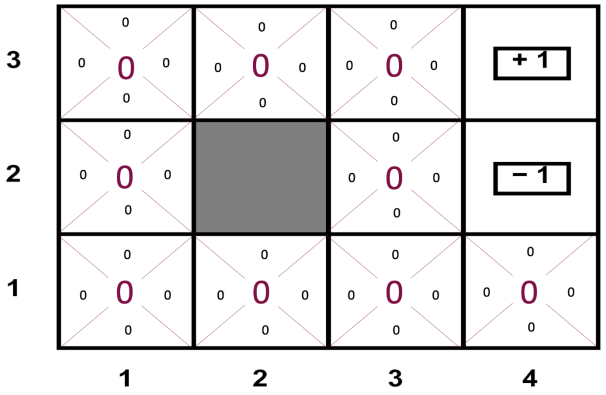
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

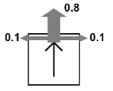
0 0.9



0.8x[0+0]
+ 0.1x[0+0.9x-1]
+0.1x[0+0]
=-0.09

Q-Iteration example

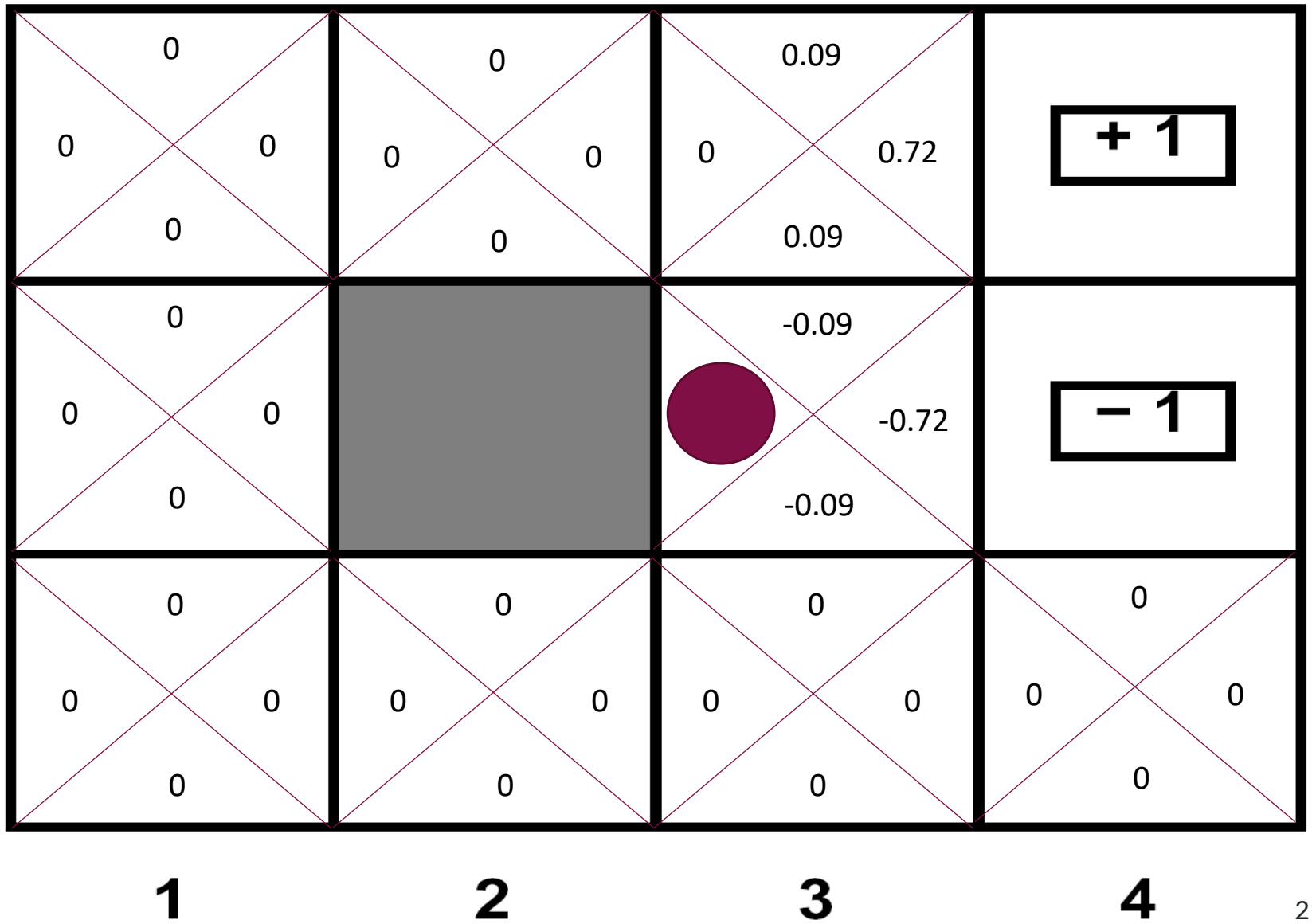




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

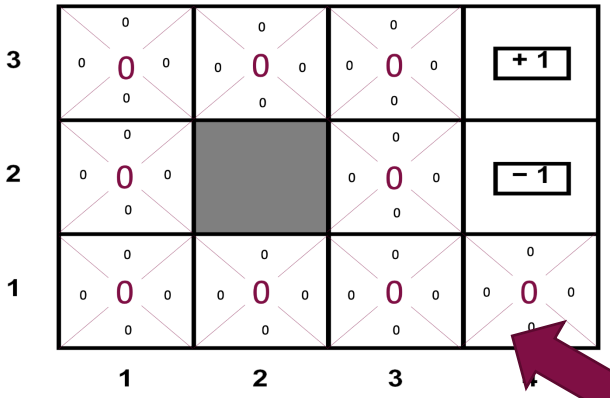
0

0.9



0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0]
=0

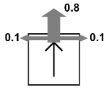
Q-Iteration example



Store maxQ in each cell for easy reference

Now we have $Q_1(s,a)$ for all (s,a)

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$



0

0.9

0 0 0 0 0	0 0 0 0 0	0 0.09 0 0.72 .72 0.09	+ 1
0 0 0 0 0		0 -0.09 0 0 -0.72 -0.09	– 1
0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	-0.72 -0.09 0 -0.09 0

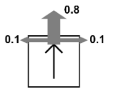
3

2

1

1 2 3 4

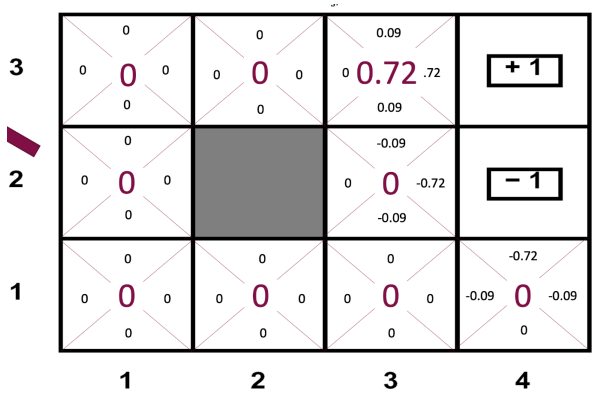
Q-Iteration example



$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

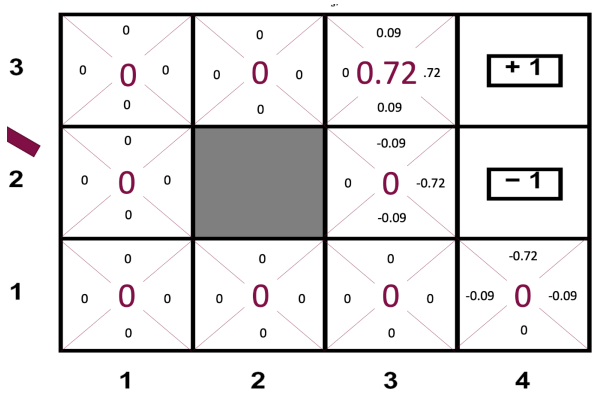
0 (arrow to $R(s,a,s')$)

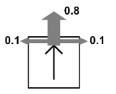
0.9 (arrow to γ)



0.8x[0+0.9x1]
+ 0.1x[0+0.9x0.72]
+0.1x[0+0]
=0.7848

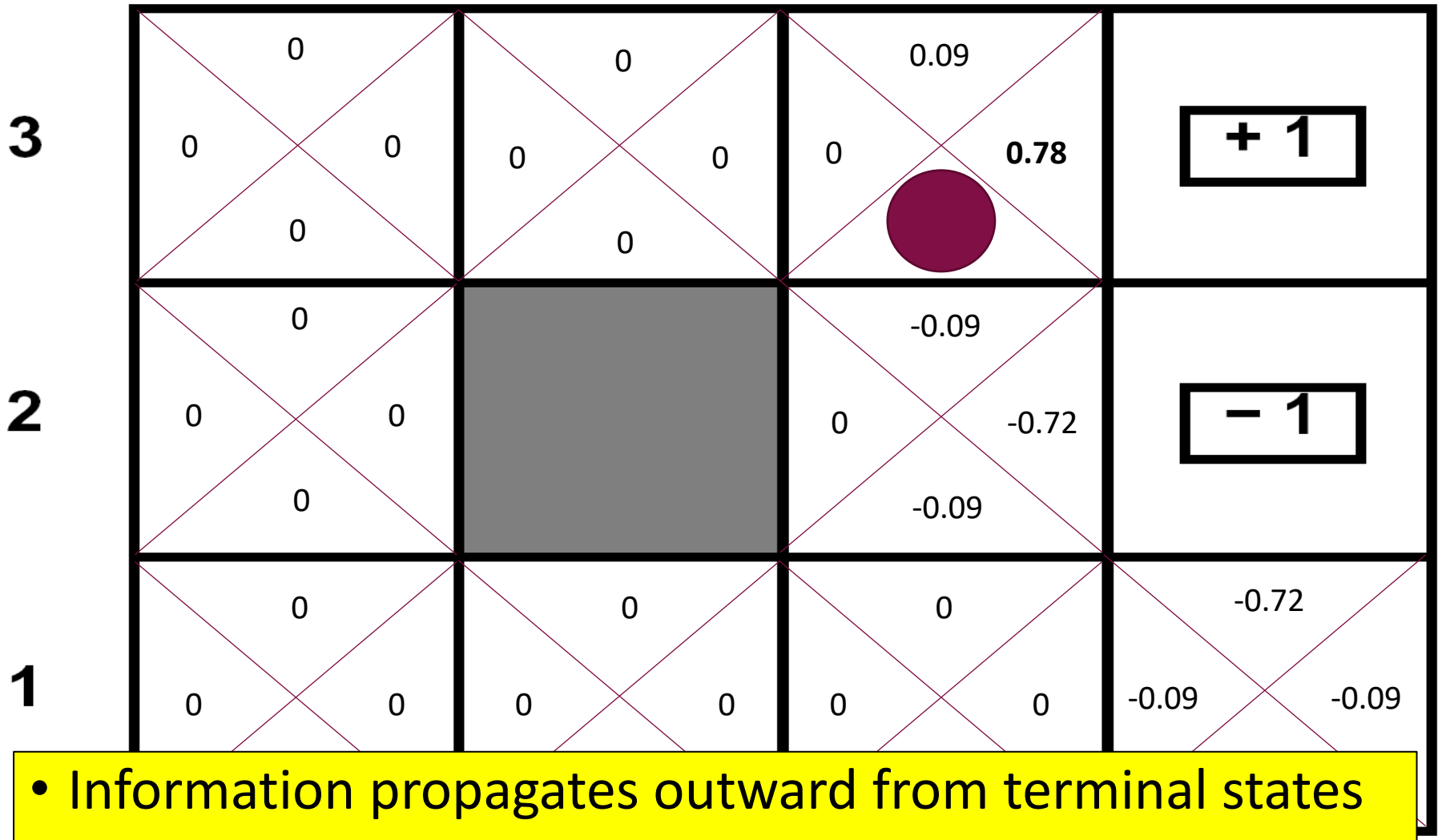
Q-Iteration example





$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

0 (reward) 0.9 (discount)



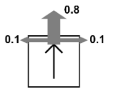
0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09

And so on till convergence…

- Information propagates outward from terminal states
- Eventually all states have correct value estimates

Q-Iteration example

(after 1000 sweeps over (s,a))



$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

0 (reward) 0.9 (discount)

3	0.59 0.57 → **0.64** 0.53	0.67 0.60 → **0.74** 0.67	0.77 0.66 → **0.85** 0.57	**+ 1**
2	**0.57** 0.51 ↑ 0.51 0.46		**0.57** 0.53 ↑ -0.60 0.30	**− 1**
1	**0.49** 0.45 ↑ 0.41 0.44	0.40 **0.43** ← 0.42 0.40	**0.48** 0.40 ↑ 0.29 0.41	-0.65 **0.28** ← 0.13 0.27
	1	2	3	4

Recap: Q-Value iteration

To find optimal Q value, given known P, R:

- Start with a random guess, e.g.,$Q_0(s, a) \leftarrow 0$ for all states s and actions a
- Then repeat till convergence, for all s, a:

$$Q_{i+1}(s, a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]$$

(Bellman Eq for optimal Q, converted to update rule)

Two extensions:

1. Can also do this to find the optimal V value: called V value iteration, or simply value iteration.

V Value Iteration

Finding V^* in “known environments”

V-Value Iteration

Fully analogous to the Q value iteration we have already seen.

Bellman says optimal value functions $V^*(s)$ should obey:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Convert to an update rule!

Algorithm:

- Start with $V_0(s) = 0$ for all states s
- Iterate the Bellman update until convergence, sweeping over all states s:

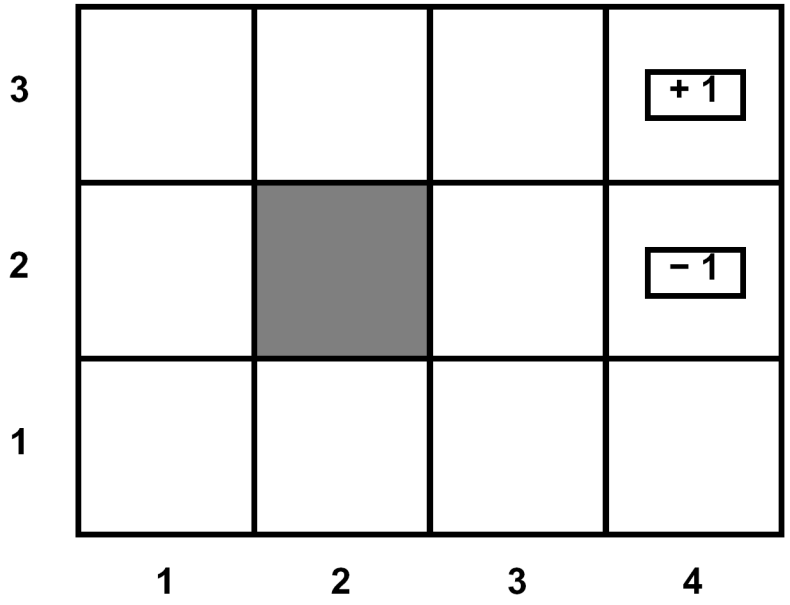
$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

Example: V- Value Iteration

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

γ=0.9, living cost=0

Example MDP



V_0

3	0	0	0	0
2	0		0	0
1	0	0	0	0
	1	2	3	4

Start with $V_0(s) = 0$

V_1

3	0	0	0	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

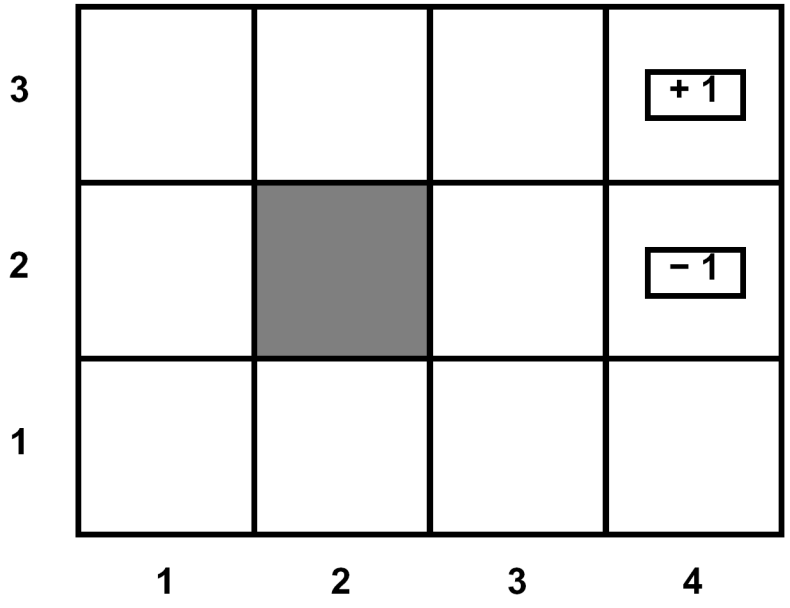
Based on example by Dan Klein

Example: V- Value Iteration

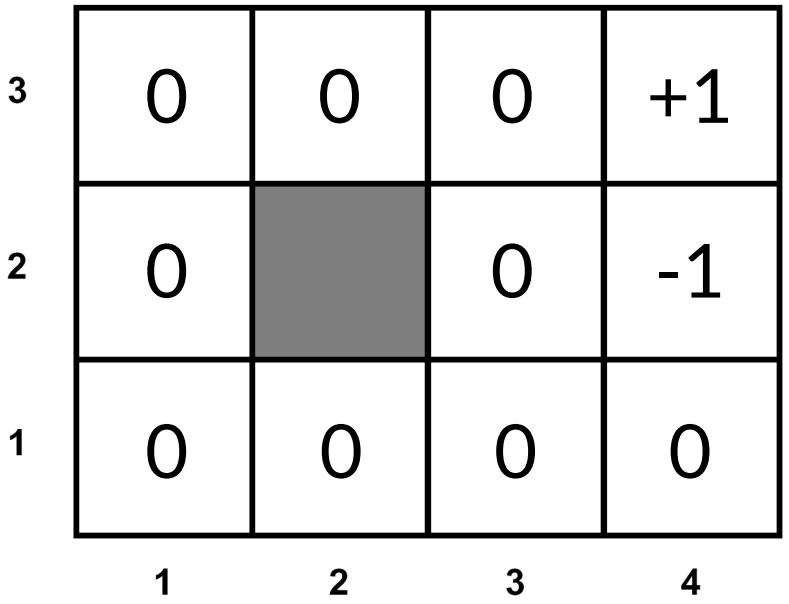
$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

γ=0.9, living cost=0

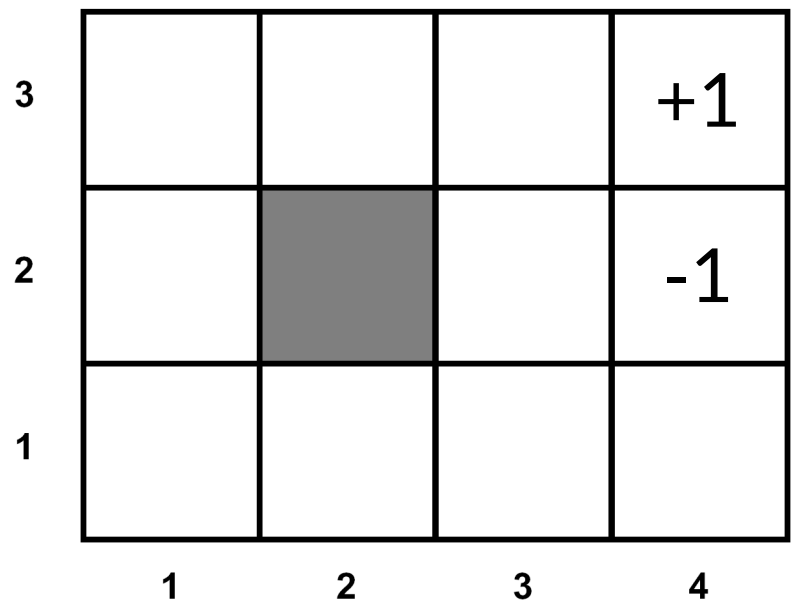
Example MDP



V_1



V_2



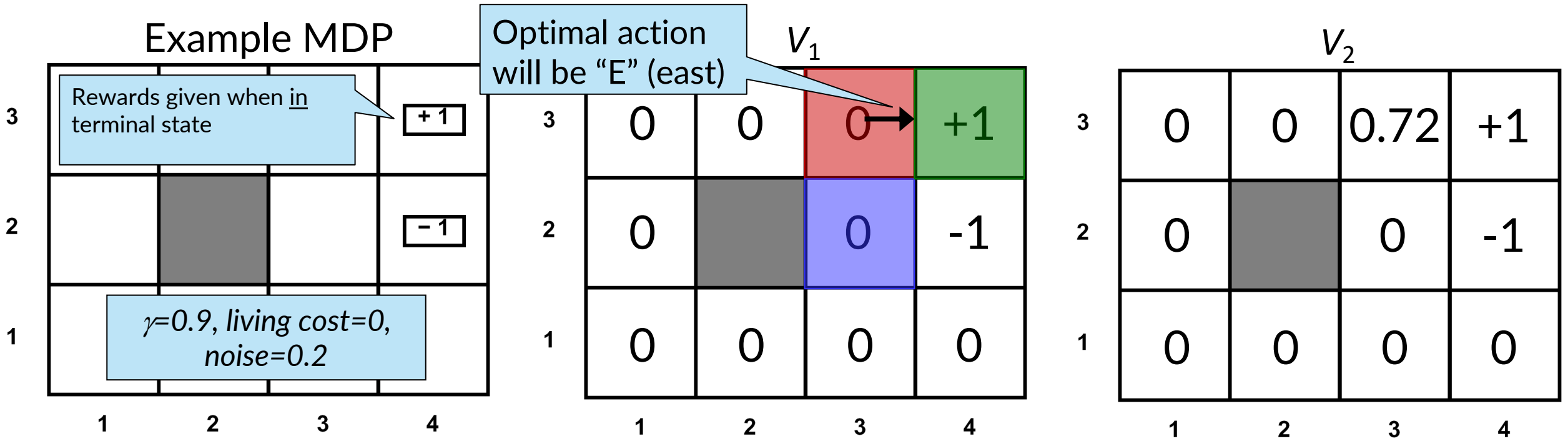
$V_2(\langle 4,3 \rangle) \leftarrow 1$

$V_2(\langle 4,2 \rangle) \leftarrow -1$

Based on example by Dan Klein

Example: V- Value Iteration

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$



$$V_2(\langle 3,3 \rangle) \leftarrow \sum_{s' \in S} P(s'|\langle 3,3 \rangle, E)[r(\langle 3,3 \rangle, E, s') + 0.9V_i(s')]$$

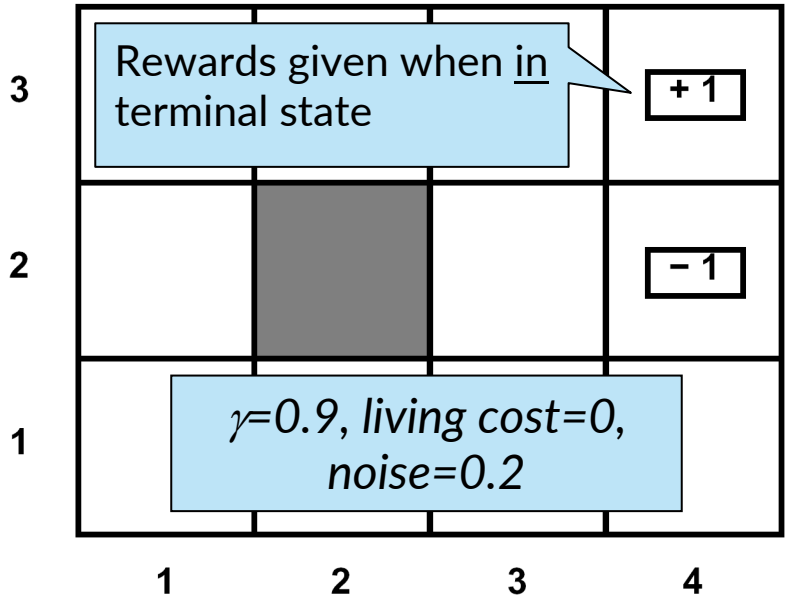
$$\leftarrow 0.8[0 + 0.9\times1] + 0.1[0 + 0.9\times0] + 0.1[0 + 0.9\times0] = 0.72$$

Based on example by Dan Klein

Example: V- Value Iteration

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

Example MDP



V_2

3	0	0	0.72	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

V_3

3	0	0.52	0.78	+1
2	0		0.43	-1
1	0	0	0	0
	1	2	3	4

Eventually all states have correct V^* estimates.

What is the relationship between $V^*(s)$ and $Q^*(s,a)$?

Answer: $V^*(s) = \max_a Q^*(s,a)$

Based on example by Dan Klein

Your Very First RL Algorithm: Q Learning

Replacing known transition function P and known reward function R with samples from experience

Recap: $\boldsymbol{Q}$ Value Iteration

To find optimal Q value, given known P, R:

- Start with a random guess, e.g.,$Q_0(s, a) \leftarrow 0$ for all states s and actions a
- Then repeat till convergence, for all s, a:

$$Q_{i+1}(s, a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]$$

(Bellman Eq for optimal Q, converted to update rule)

- Two extensions:
  1. Can also do this to find the optimal V value: called V value iteration, or simply value iteration.
  2. How about in the absence of known P, R ? (i.e., the reinforcement learning setting)

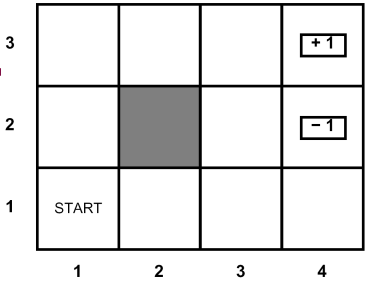
Q Learning

Q-value iteration: $Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$

How to extend this to when the functions $P(s'|s,a)$ and $R(s,a,s')$ are unknown and only revealed gradually through experience?

Note: Every time you take action a from state s, you get one sample from the unknown $P(s'|s,a)$ and the corresponding reward $R(s,a,s')$

Samples of the full Transition / Reward functior



For example, the full transition function $P(s'|s,a)$ might look like:

	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	(4,1)	(4,2)	(4,3)
(1,1), N	0.1	0.8	0	0.1	0	0	0	0	0	0	0	0
(1,1), E	0.1	0.1	0	0.8	0	0	0	0	0	0	0	0
(1,1), S	0.9	0	0	0.1	0	0	0	0	0	0	0	0
…												

In RL, if you perform action "N" from state (1,1), you might find out that you have ended up back at (1,1).

- Until you try it, you have no knowledge at all about $P(s'|s=(1,1),a=N)$
- Even after you try it, you only get a single sample.

Can't really do $Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$ as in Q value iteration, because we don't know $P(s'|s,a)$ and $R(s,a,s')$ for all s'!

Solution: "Q Learning"

Q-value iteration: $Q_{i+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$

Idea: Treat the single sample you get as a rough estimate of the expectation, and apply an *incremental* update to reduce the "Bellman error":

- Execute a single action a from state s and observe s' and R:

$$sample = R + \gamma \max_{a'} Q_{old}(s',a')$$

- Now, compare this sample to the LHS, and apply the *incremental* update:

$$Q(s,a) \leftarrow Q_{old}(s,a) + \alpha \underbrace{\left(R + \gamma \max_{a'} Q_{old}(s',a') - Q_{old}(s,a)\right)}_{\text{Bellman error}}$$

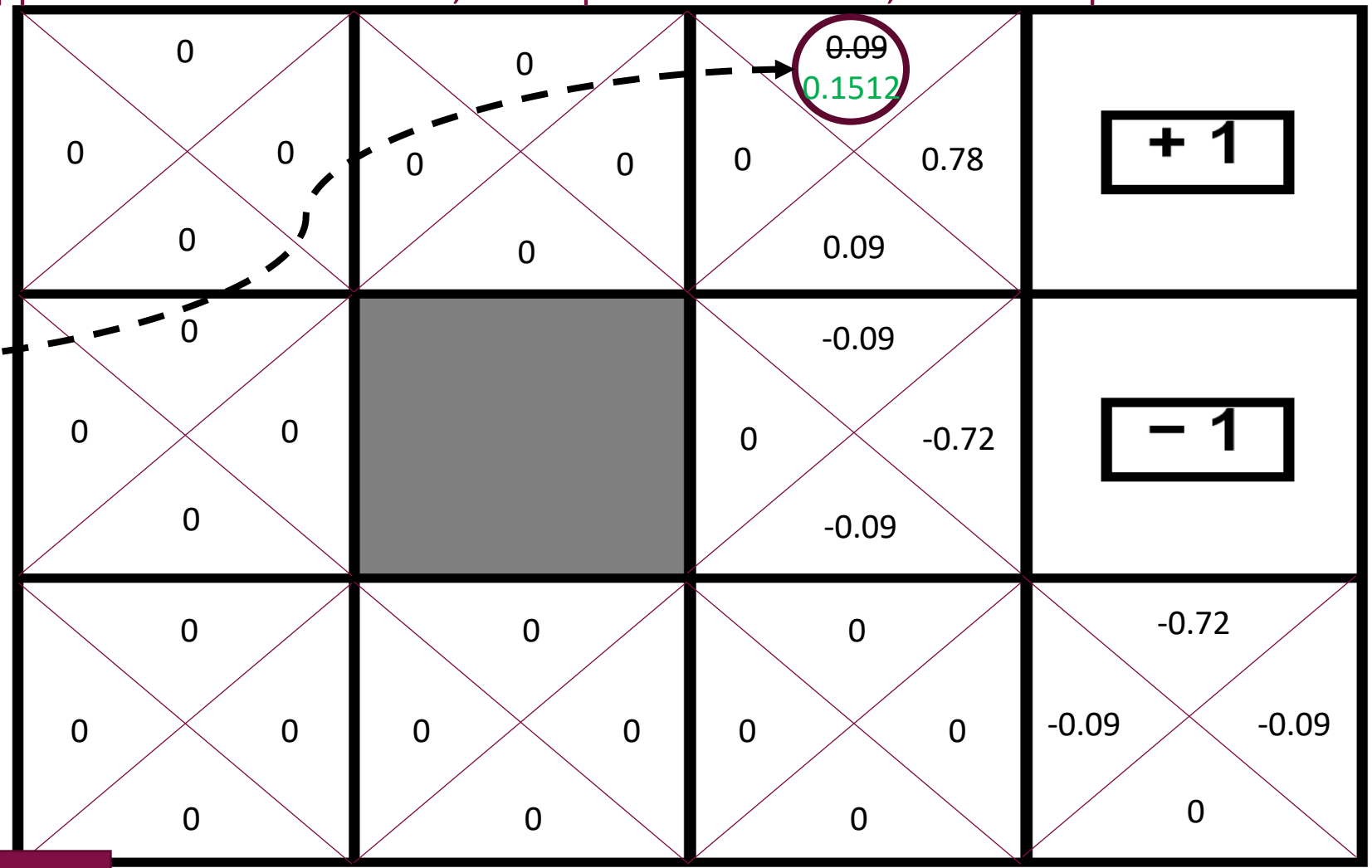
Thus, we can now get the optimal Q from the agent's trial-and-error experience. **This is called "Q-Learning".** Q iteration + 1-sample-based incremental update.

Q-Learning example step

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

0.1 0.9

Suppose we start at this cell, attempt to move north, and end up at same cell



Current Q=0.09

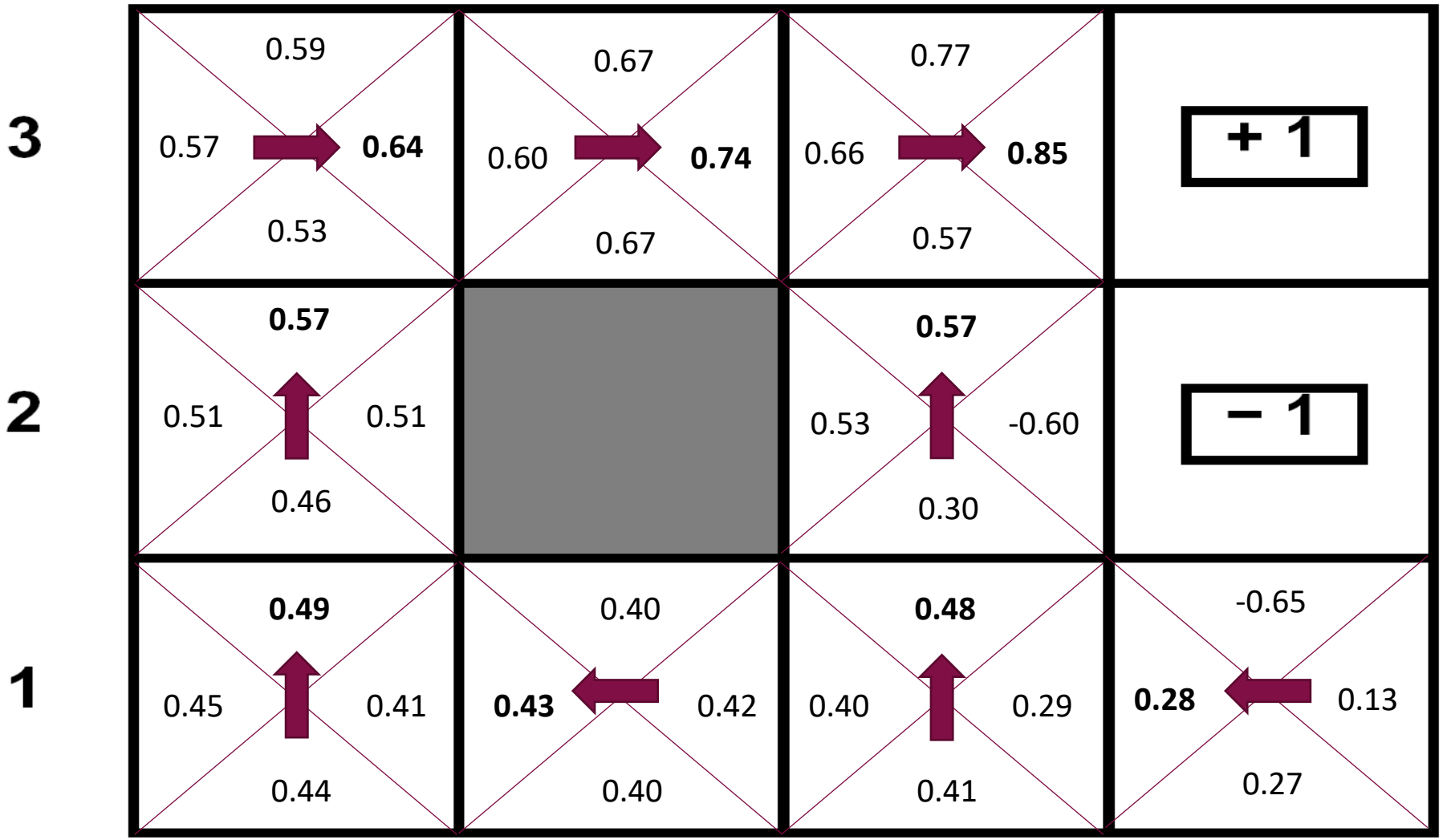
Sample $R + \gamma \max Q$ =
0+0.9x0.78 = 0.702

New Q =
0.09+0.1X(0.702-0.09)
= 0.1512

And so on till convergence…

Q-Learning example
(after 100,000 actions)

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



Q Learning works! Recovers the true Q*, even with unknown P, R.

Exploration and Exploitation

How to Act During Q-Learning: "Off-policy" vs "On-Policy"

- Execute a single action a from state s and observe s' and R:

$$sample = R(s,a,s') + \gamma \max_{a'} Q^*(s',a')$$

- So, the incremental TD update is:

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

Bellman error

This is called "Q-Learning".

Note that we have said nothing about *which* actions to sample.

- You can act any way you want, and as long as you "explore" the environment well, Q-Learning will eventually converge to the optimal $Q^*(s,a)$.
- This is not true for all RL approaches: many rely on executing actions from the current best policy. Those are "on-policy" approaches.

Exploration vs Exploitation

- What happens if you execute only your current-best policy all the time?
  - Might not explore enough to discover other solutions.
  - For example, you might never discover a shortcut if you only stick to a known route to a target.
- What happens if you only execute random actions all the time?
  - Wasteful. You mainly care about states and actions encountered by the optimal policy.
  - For example, if you keep exploring the city randomly, it will take a really long time for you to learn any meaningful route to your target.

Exploration vs. Exploitation Tradeoff

Based on slide by Dan Klein

Some Simple Schemes for Balancing Explore-Exploit

- ϵ −greedy:
  - At every state,
    - With small probability ϵ, perform a random action
    - Otherwise, follow current best $a^* = \text{argmax}_a[Q(s,a)]$
  - Can anneal ϵ over time
    - Intuition: should explore more when you know very little about the city. After having lots of experience navigating it, there isn't much value to exploration any more.
- Track Visitation Counts:
  - Maintain a running count of the number of times $N(s,a)$ that you have tried executing a from state s.
  - Select $a^* = \text{argmax}_a[Q(s,a) + 1/N(s,a)]$, inflating the return of states that you have not visited.

Q-Learning and More in HW4

- In HW4, you will implement Q-Learning.
- Also, you will see an algorithm called “behavior cloning” through which sequential decision making problems can be made to look like supervised learning problems.
  - Suppose that you had an “expert” who could label each state with the optimal action …
  - Not always possible, and has many issues.
  - When possible, often works surprisingly well.

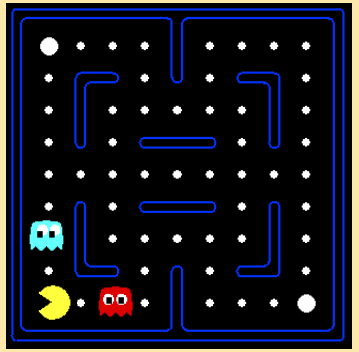
From here on out: not covered in class

- This material is provided only for your reference, and we will not actively test it (unless it is taught through homework or in a future lecture).
- Deep Q Learning
- Real-World Applications Of RL

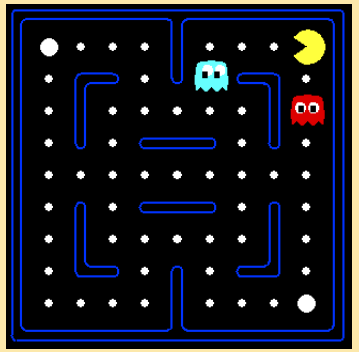
From Tabular to Deep Q Learning

High-dimensional states example: Pacman

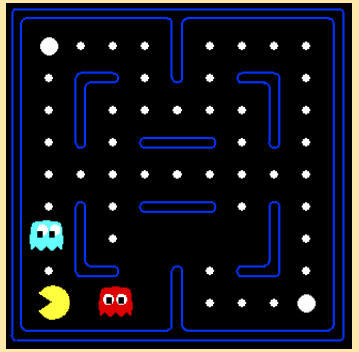
- Let's say we discover through experience that this state is bad:



- In naïve Q-learning, we know nothing about this state or its Q states:



- Or even about this one!



Slide by Dan Klein

Q-Learning

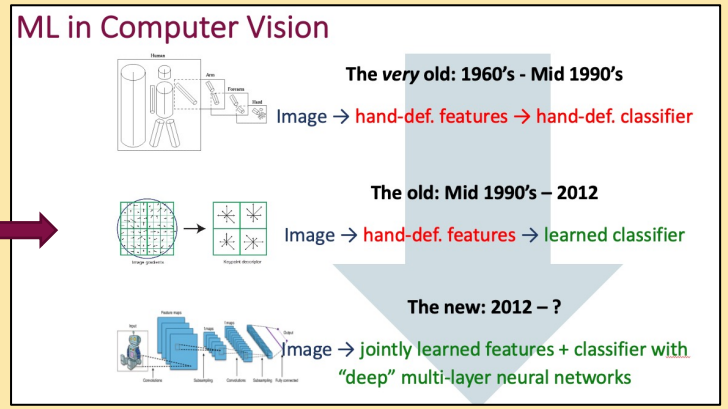
- In many real situations, we cannot possibly learn about every single state+action!
  - Too many state-action pairs to visit them all in training
  - Too many state-action pairs to hold the Q-tables in memory

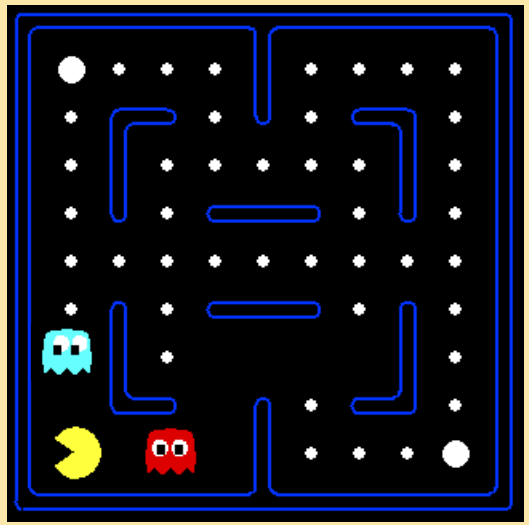
- Instead, we want to generalize:
  - Learn about some small number of training Q-states from experience
  - Generalize that experience to new, similar Q-states
  - This is a fundamental idea in machine learning, and we see it over and over again

Based on slide by Dan Klein

Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features
    - e.g. action moves closer to food



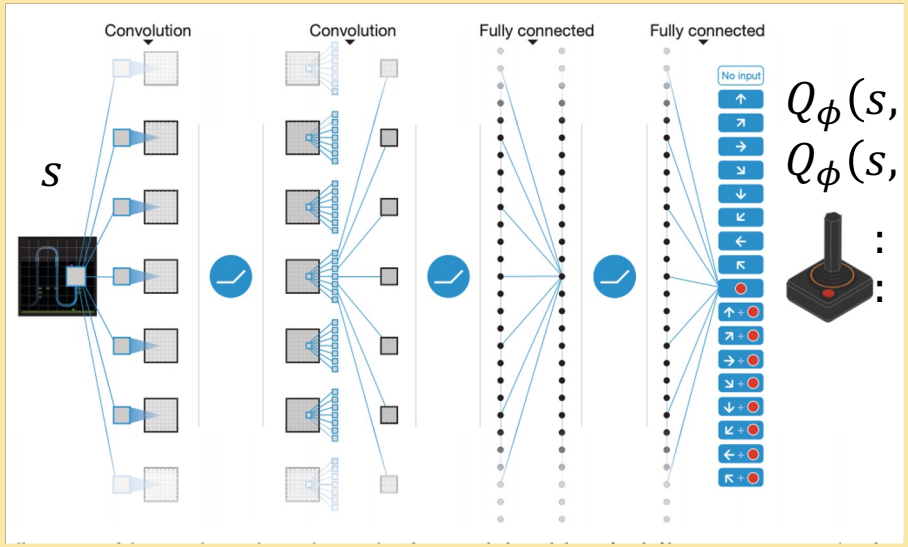


As we now do in computer vision/NLP, can we avoid engineering these features?

Based on slide by Dan Klein

A Neural Network to Predict Q from "Raw" State Input

Predict Q-values with a deep neural network



- **Input:** the state, e.g. an image
- **Output:** Q-values of various actions

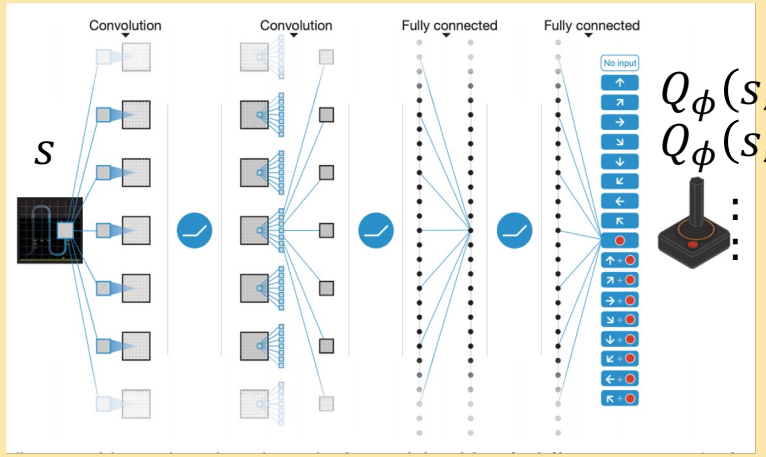
- **Learning:**

gradient descent* with the squared Bellman error loss:

$$\left(\underbrace{\left(R + \gamma \max_{a'} Q_\phi(s', a')\right)}_{= y_i} - Q_\phi(s, a)\right)^2$$

As always, the policy action is the one with the highest predicted Q-value

Deep Q-Learning v1



1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\right)$

$$= \frac{d}{d\phi}\left(Q_\phi - y_i\right)^2$$

Note: we pretend that y_i is a constant while computing the gradient, to resemble regression

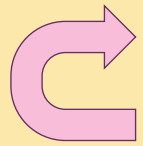
Incremental update step → gradient descent* on the squared Bellman error loss!

Closely connected to the tabular Q learning update. Hint: if you replace the neural network with a Q table, its parameters ϕ are just Q value entries?

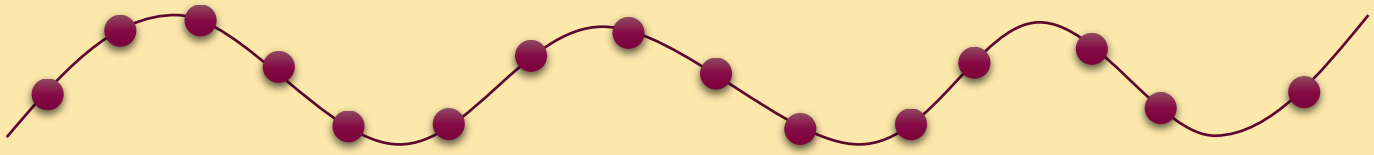
- Execute a single action a from state s and observe s' and R:
$$sample = R + \gamma \max_{a'} Q_{old}(s', a')$$
- Now, compare this sample to the LHS, and apply the *incremental* update:
$$Q(s, a) \leftarrow Q_{old}(s, a) + \alpha \underbrace{\left(R + \gamma \max_{a'} Q_{old}(s', a') - Q_{old}(s, a)\right)}_{\text{Bellman error}}$$

Based on slide by Sergey Levine

Problems with Deep Q-Learning v1

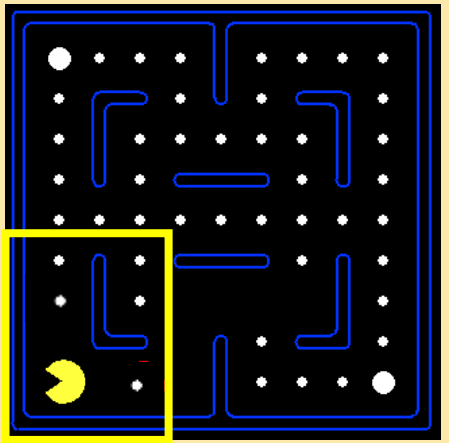


1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$



Problems:

1. sequential states are strongly correlated (not i.i.d.)



So consecutive Q updates drive the network to overfit to recently encountered states and forget previous experiences

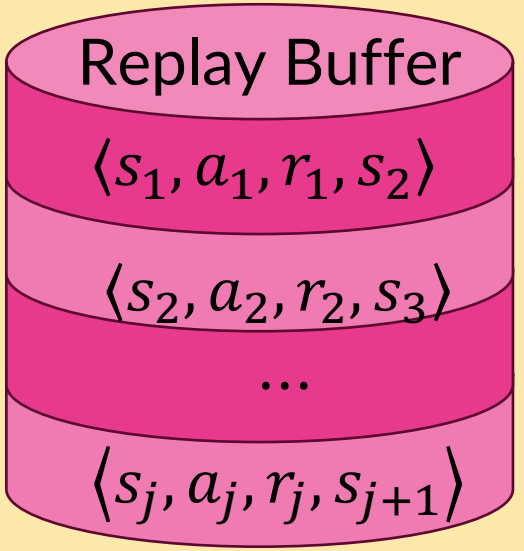
Based on slide by Sergey Levine

Addressing Correlations: Experience Replay

- Q-Learning is “off-policy”: we don’t say anything about the specific actions that need to be executed, and we don’t need the transitions to be in sequence.

- Maintain a “replay buffer” of previous experiences

- Perform Q-updates based on a sample from the replay buffer



FIFO or Priority Queue

- Advantages:
  - Breaks correlations between consecutive samples
  - Each experience step may influence multiple gradient updates

Based on slide by Sergey Levine

Deep Q Learning v2 (with replay buffer $\mathcal{D}$)

Deep Q Learning v1

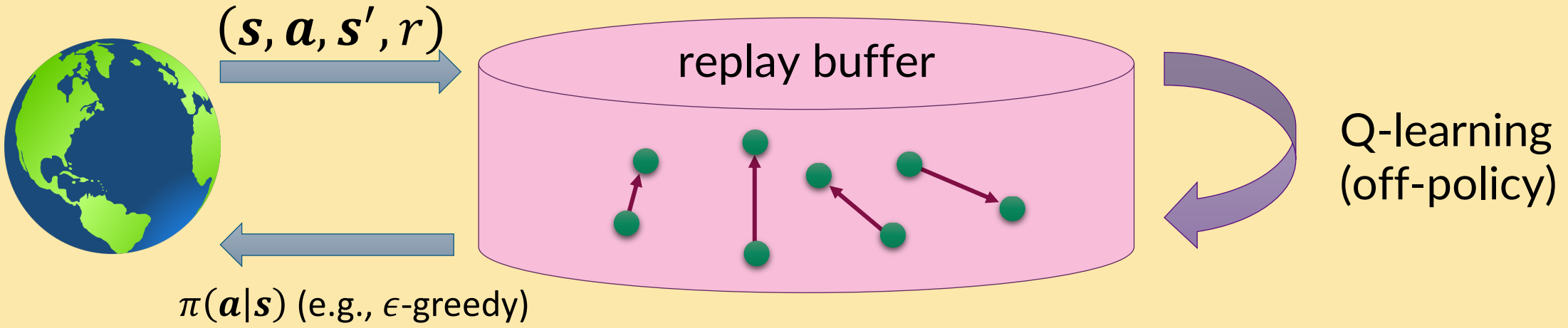
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

Deep Q Learning v2

$K\times$

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{D}$
2. Loop K times, do:
3. sample a batch of $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$'s from $\mathcal{D}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

$K = 1$ is common, though larger K may sometimes be more efficient



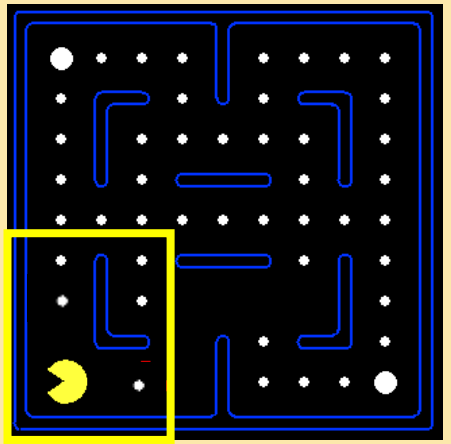
Based on slide by Sergey Levine

Problems with Deep Q-Learning v1

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)] \right)$

Problems:

1. sequential states are strongly correlated (not i.i.d.)
2. Target value is always changing!



Based on slide by Sergey Levine

Problem: Moving Target for Q-regression

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \underbrace{[r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)]}_{\text{no gradient through target value}} \right)$

Problem: Instability (e.g., rapid changes) in Q(·) can cause it to diverge

- Q-learning is *not* gradient descent on any fixed objective!

Solution: use two nets to provide stability

- The Q-network is updated regularly
- The target network is an older version of the Q-network, updated occasionally

$$\Big(\underbrace{Q_\phi(s,a)}_{\text{computed via Q-network}} - \big(r_i + \gamma \max_{a'} \underbrace{Q_{\phi'}(s',a')}_{\text{computed via target network}} \big) \Big)^2$$

Based on slide by Sergey Levine

Deep Q Learning v3

Deep Q Learning v2

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{D}$
2. Loop K times, do:
3. sample a batch of $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$'s from $\mathcal{D}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)]\right)$

($K\times$)

Deep Q Learning v3

($N\times$)

1. save target network parameters: $\phi' \leftarrow \phi$

($K\times$)

2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{D}$
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{D}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r_i + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

supervised regression

targets don't change in inner loop!

This is the "classic" deep Q Learning algorithm from 2015!*

*(usually K=1)

Based on slide by Sergey Levine

DQN on Atari Games

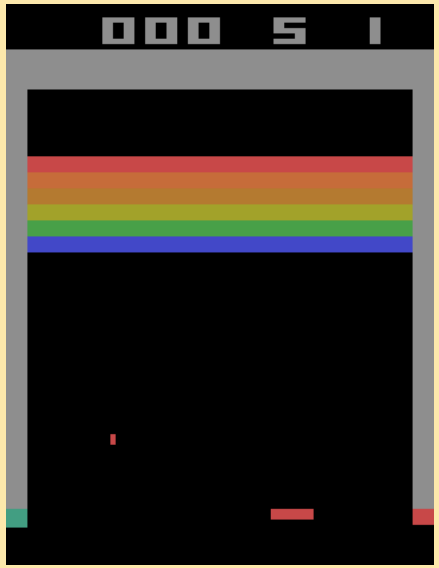


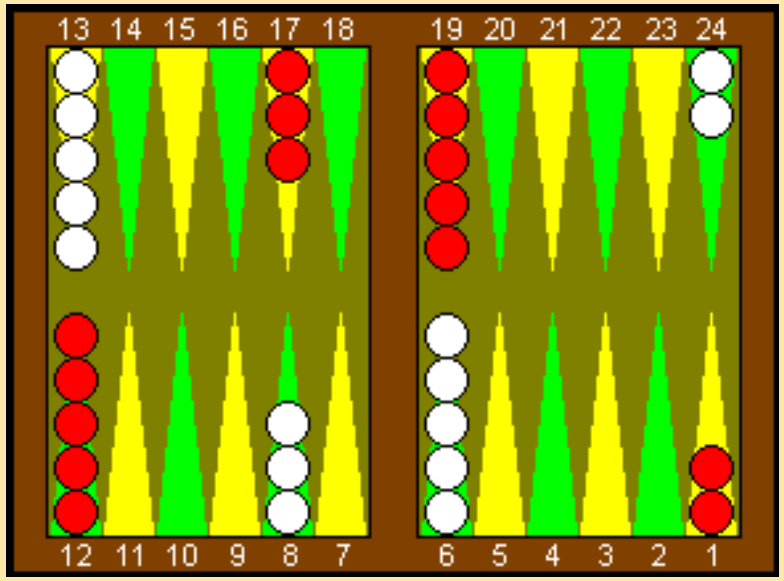
Image Sources:
https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756
https://deepmind.com/blog/going-beyond-average-reinforcement-learning/
https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/

DQN on Atari Games

https://www.youtube.com/watch?v=rQIShnTz1kU

DQN for Continuous Control (DDPG)

Test Bed: Video Games and Board Games



TD-Gammon (1992).

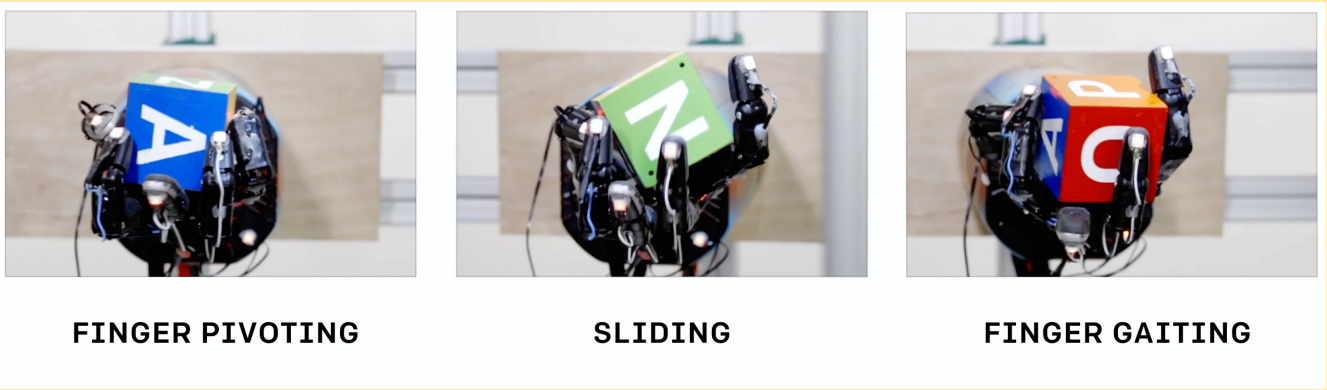
Deep Q Learning (2013)

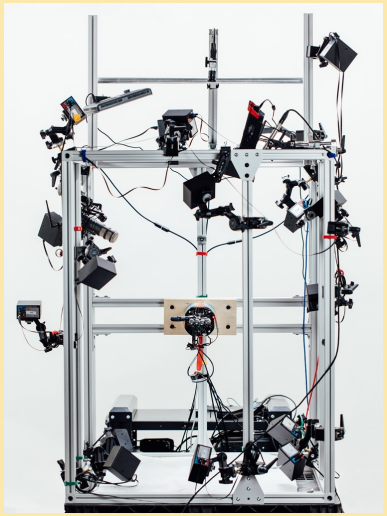
AlphaGo (2016).

GT Sophy (2023)

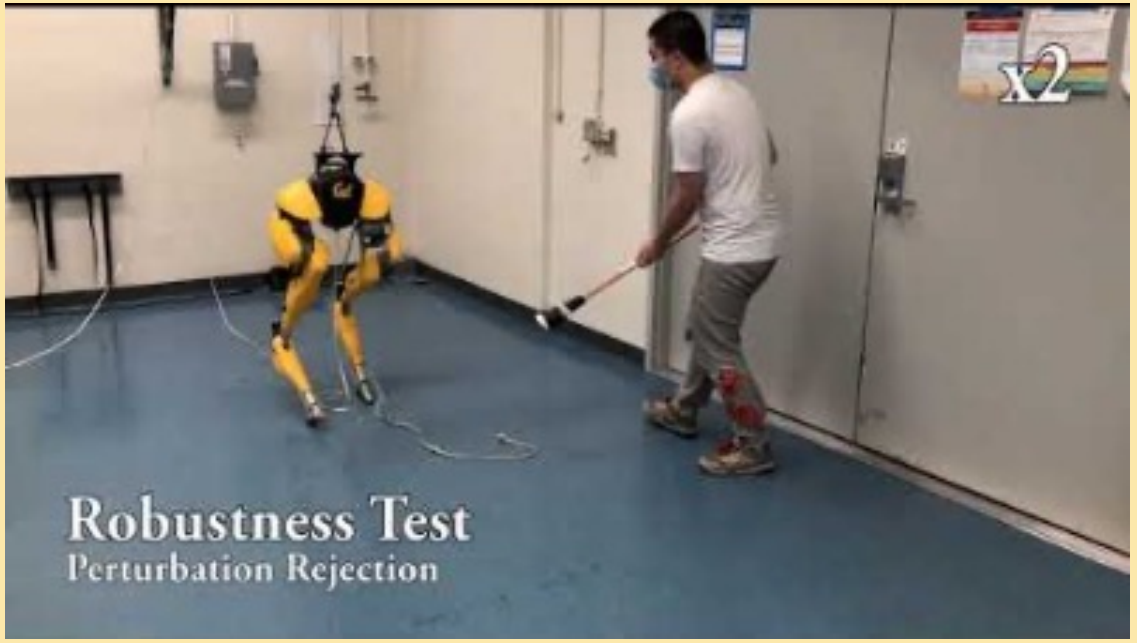
Robotics

Robotics





Open AI Dactyl (2018)



Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots, 2022

More RL for Robotics

- Guiness world record in 100 meters by biped robots (Oregon State University)
- Learned quadrupedal locomotion in challenging environments (ETH Zurich)
- Autonomous Navigation of Stratospheric Balloons (Google AI), blog (was real, just Google canceled the whole project.. sadly..)
- Not yet perching (article), but soon? Just for inspiration..
- Video games; car racing in video games, competing with humans

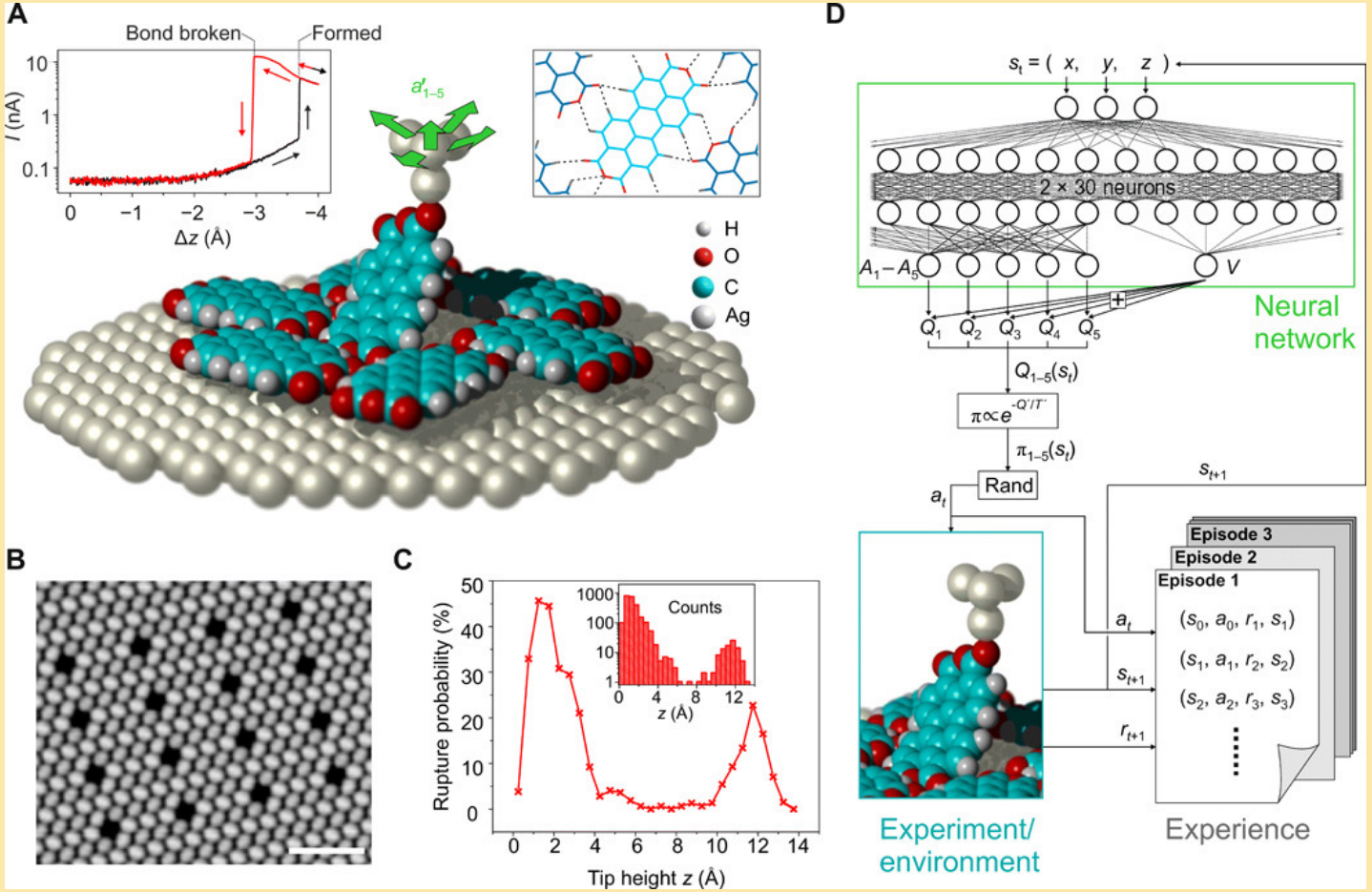
- Vision-based autonomous drone racing (video, UZH RPG)
- Commanding robots using natural language to perform tasks (SayCan project, Google)
- behavioral cloning/imitation learning (not RL) is doing well with transformers in the kitchen (Google)
- Yet it is not enough to learn to drive well
- Quadruped learns to walk in the park in 20 minutes, model-free (UC Berkeley)
    - More of this
- Still, dexterous manipulation is not easy.. (Berkeley, Meta, UW)
- Visual Navigation (Berkeley)
- In the need for resets (Berkeley)

Credit: Csaba Szepasvari

Subtractive Manufacturing

Microscopy

(steering a microscope to separate molecules)



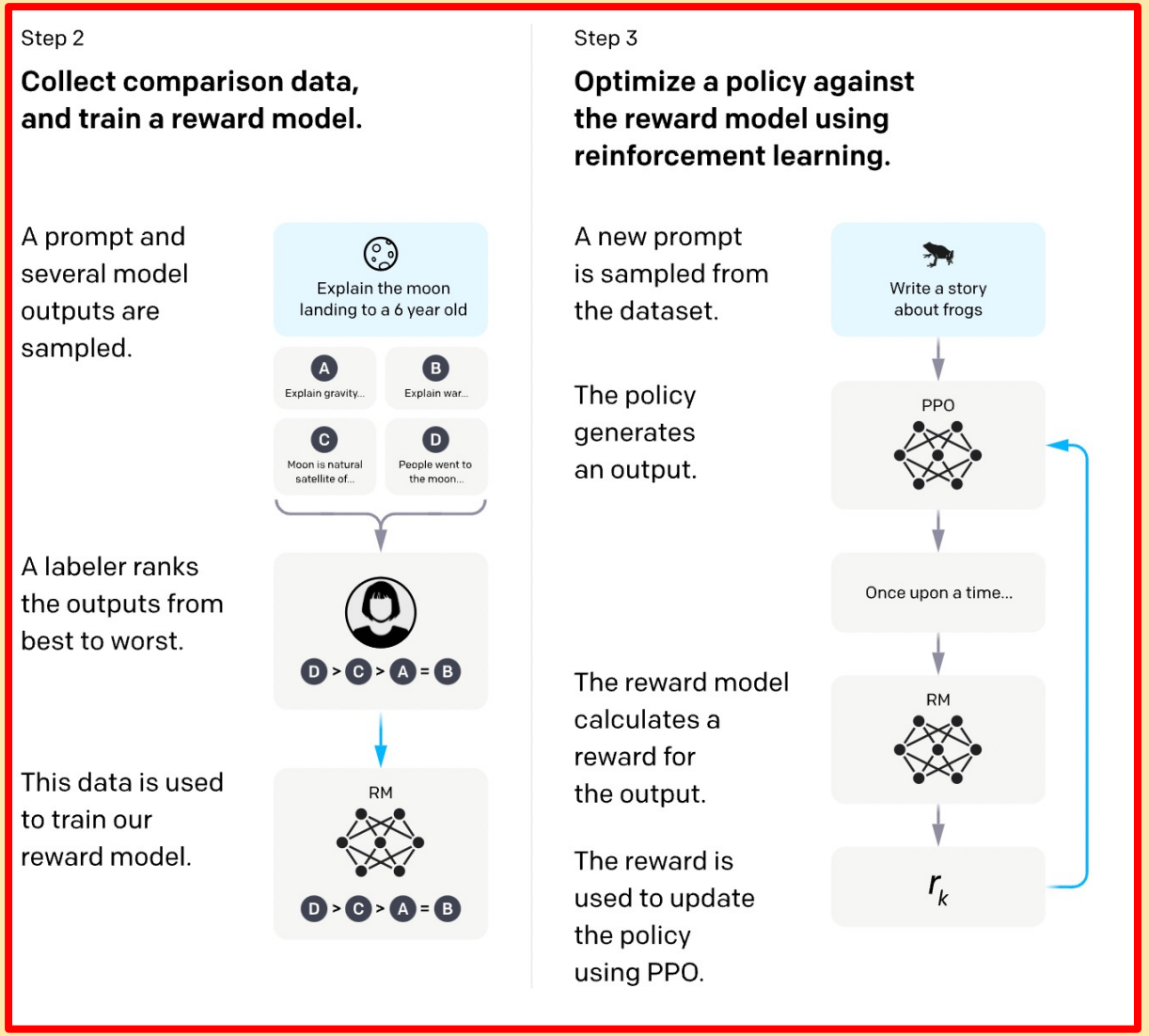
https://www.science.org/doi/10.1126/sciadv.abb6987

Finetuning Conversational Agents

ES

Briefly explain chatGPT using a metaphor that would resonate with an audience of stock-market investors and avoiding AI jargon.

One way to explain chatGPT to stock-market investors is to compare it to a financial advisor who uses past performance data and market trends to make predictions about future investments. Just like a skilled financial advisor, chatGPT uses large amounts of training data to generate intelligent and informed responses to questions, providing valuable insights and suggestions to users.

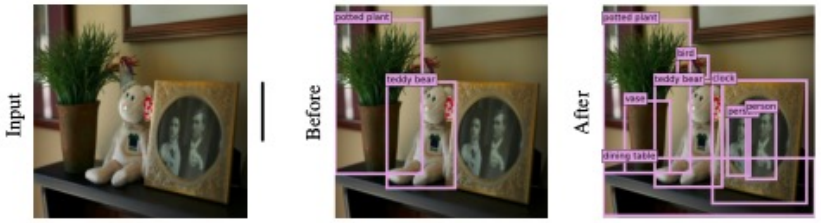


https://openai.com/research/instruction-following

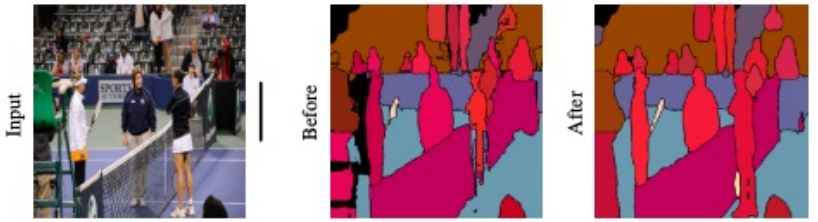
Finetuning Other Pretrained ML Models

- Like ChatGPT, can also use RL to finetune other models to maximize some performance score.

Aside: Why would these models not have been trained directly to maximize the performance scores in the first place?



(a) Optimize mAP: 39 → 54, results in a much high recall and learns box prediction confidences.



(b) Optimize PQ: 43.1 → 46.1, removes many incoherent predictions, especially for small-scale objects.

(c) Optimize "colorfulness" score: 0.41 → 1.79, improves color diversity and saturation.

Figure 1. By tuning a strong, pretrained model with a reward that relates to the task, we can significantly improve the model's alignment with the intended usage.

https://arxiv.org/pdf/2302.08242.pdf

Web Assistants

Web navigation

(e.g. navigating a flight-booking website to make a purchase)

Gur et al 2021, Environment Generation for Zero-Shot Compositional Reinforcement Learning

Real-world applications using RL: Already working

- Applications to algorithms:
  - Video compression on Youtube using nuzero (DeepMind)
  - Faster matrix multiplication (blog, article) (DeepMind)
  - Faster std::sort in the LLVM compiler toolchain, background:, the new part (DeepMind)
  - Chip design applied to Google TPUs (Google AI)
- Industrial automation:
  - Cooling the interior of large commercial buildings (DeepMind), (Vector&Telus, prelim)
  - Amazon “deep” inventory management

Credit: Csaba Szepasvari

Real-world applications using RL: In the works

- Control: Nuclear fusion (DeepMind)
- Education (2021 paper)
- Healthcare (2020 survey)
- Power grids: Reinforcement learning for demand response: A review of algorithms and modeling techniques José R. Vázquez-Canteli, Z. Nagy
- Recommender systems: https://github.com/google-research/recsim
- Automated stock trading: https://github.com/AI4Finance-LLC/FinRL-Library
- And many other "real-world" applications
  - https://arxiv.org/abs/1904.12901
  - https://arxiv.org/abs/2202.11296

RL is not yet "just working", but there is hope. Several important open problems with potential for impact in large numbers of applications!

Credit: Csaba Szepasvari

Initial applications of reinforcement learning span most, if not all, industries.

● Optimizing product development cycles (AI-assisted design) ● Optimizing complex operations ● Informing next best action for each customer

Industry	Sample reinforcement learning applications
Advanced electronics and semiconductors	● Optimize silicon and chip design to increase performance and reduce manufacturing costs ● Optimize fabrication manufacturing process for improved yield and throughput
Agriculture	● Solve scheduling and production allocation challenges to increase yield ● Optimize network and warehouse logistics for reduced waste and costs ● Apply advanced pricing and promotion to improve product margins
Aerospace and defense	● Optimize engineering design processes to reduce time to market for new systems and improve quality
Automotive	● Optimize design processes to shorten development cycle for new cars and features and improve quality ● Deploy advanced predictive maintenance to prevent rare failures and unplanned outages ● Deliver real-time production monitoring and controls to increase manufacturing yield
Financial services	● Apply real-time trading and pricing strategies for greater agility and revenue ● Optimize ATM replenishment and allocation strategies to reduce costs and improve the customer experience ● Deliver advanced personalization capabilities that adapt promotions, offers, and recommendations daily for increased customer satisfaction and sales
Mining	● Optimize design process so teams can explore a greater range of mine designs for improving mine yield ● Use intelligent process controls for managing power generation and bore milling to increase yield and reduce costs ● Apply holistic logistics scheduling to optimize mine-to-shipping operations and reduce costs
Oil and gas	● Enable real-time well monitoring and precision drilling for increased yield ● Optimize tanker routing to reduce costs and ensure on-time delivery ● Enable advanced predictive maintenance to prevent rare equipment failures and unplanned outages
Pharmaceuticals	● Optimize drug discovery, identifying molecules of interest faster to reduce the time and cost of research and bring new therapies to market faster ● Automate chemistry, manufacturing, and controls (CMC) to maximize batch yield and quality ● Optimize biological methods to reach peak production output
Retail	● Optimize routing, logistics network planning, and warehouse operations to reduce costs and keep shelves stocked ● Implement advanced inventory modeling and digitize supply-chain planning to prevent out-of-stocks and waste ● Deliver advanced personalization capabilities that adapt promotions, offers, and recommendations daily for increased customer satisfaction and sales
Telecom	● Optimize network layout to maximize coverage and minimize power consumption ● Manage networks in real time to optimize service quality and reduce downtime ● Apply advanced personalization to increase cross-sell and upsell revenue
Transport and logistics	● Optimize routing, logistics network planning, and warehouse op costs and improve customer satisfaction ● Optimize inbound and outbound delivery networks to minimize s associated costs

McKinsey

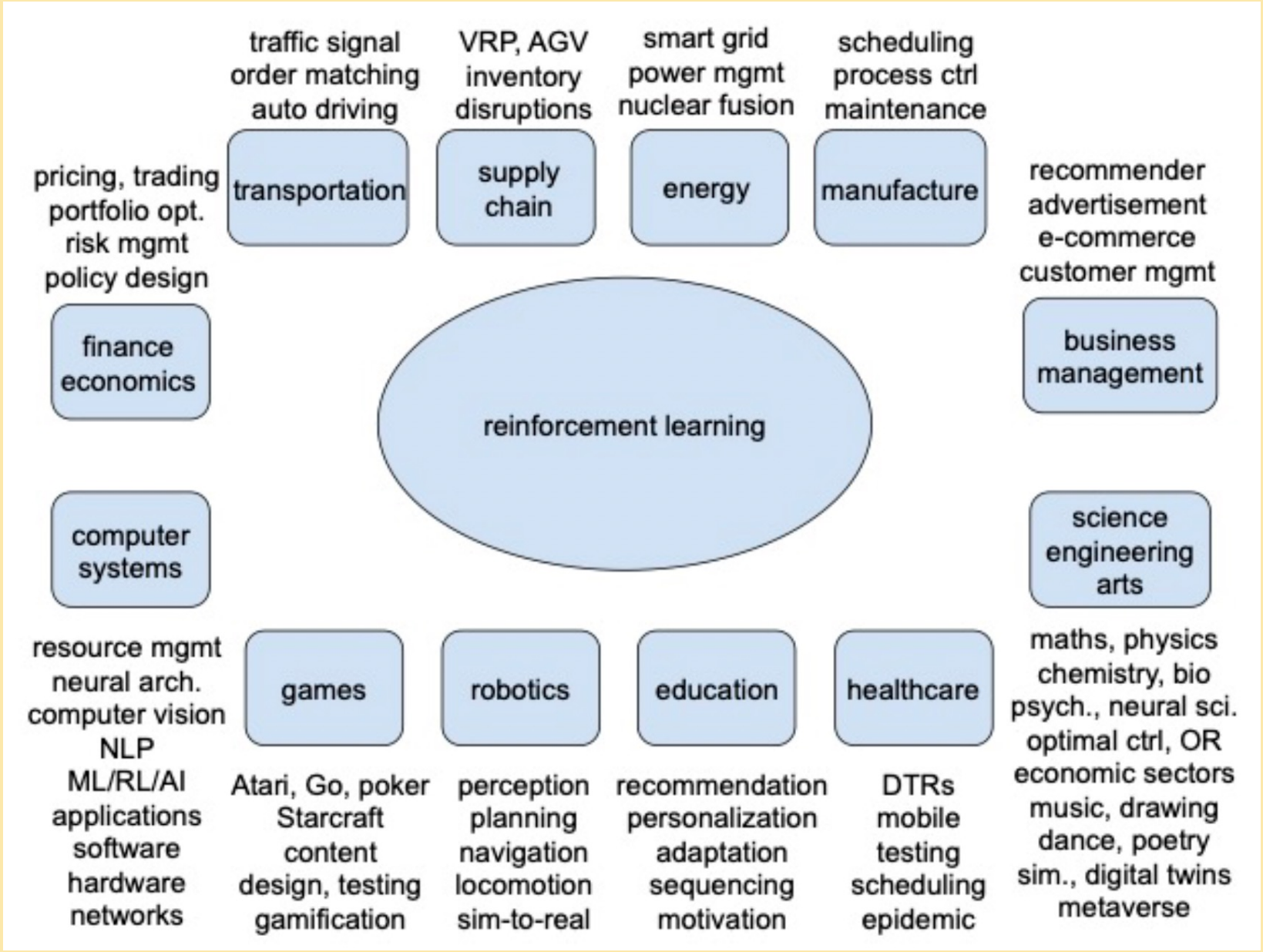
Reinforcement Learning at Work

How top companies are using this breed of AI to solve tough problems.

Company	Application	Sector	Inputs	Actions	Objective
Royal Bank of Canada	Trade execution platform for multiple strategies	Financial services	200-plus market-related data inputs	Sell, buy, hold stocks	To trade as close as possible to VWAP, a common price metric
Netflix	Test schedules for business partner devices	Technology	Historical test and device performance information	Which test to do next	Minimize device failure
Spotify	Recommendation engine	Entertainment	Previous songs liked/disliked/not played	Which songs to put in your playlist	Maximize user listening time
JPMorgan Chase	Financial derivatives risk and pricing calculations	Financial services	Historical market data	Price and sell a financial product	Maximize future cash flows of an investment portfolio
Google	Data center cooling	Technology	Temperature/air pressure	Turn on fan; add water to air unit	Control temperature and reduce energy usage
DiDi	Order dispatching	Ride hailing	Number of idle vehicles, number of orders, location, destination	Match driver to passenger	Minimize pickup time and maximize revenue

Note: Details for use cases can be found in published papers, but we could not verify if they are used in production applications.

HBR



Credit: Yuxi Li