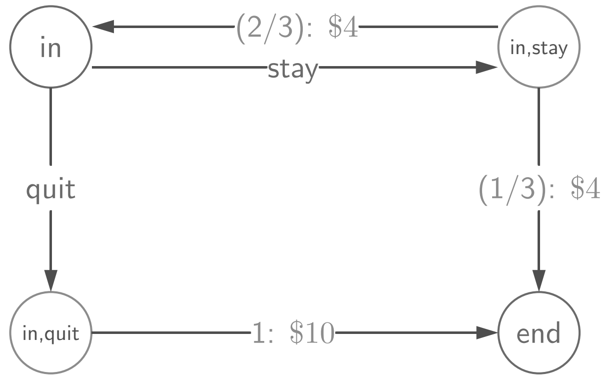
MDPs 2: Reinforcement Learning

Review: MDPs



Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in$ States: starting state

Actions(s): possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

Reward(s, a, s'): reward for the transition (s, a, s')

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Review: MDPs

- Following a **policy** π produces a path (**episode**)

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

- **Value** function $V_\pi(s)$: expected utility if follow π from state s

$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

- **Q-value** function $Q_\pi(s, a)$: expected utility if first take action a from state s and then follow π

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

Unknown transitions and rewards

Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in$ States: starting state

Actions(s): possible actions from state s

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

reinforcement learning!

CS234 course: https://web.stanford.edu/class/cs234/

Mystery game

Example: mystery buttons

For each round $r = 1, 2, \ldots$

- You choose A or B.
- You move to a new state and get some rewards.

Start A B

State: 5,0 Rewards: 0

From MDPs to reinforcement learning

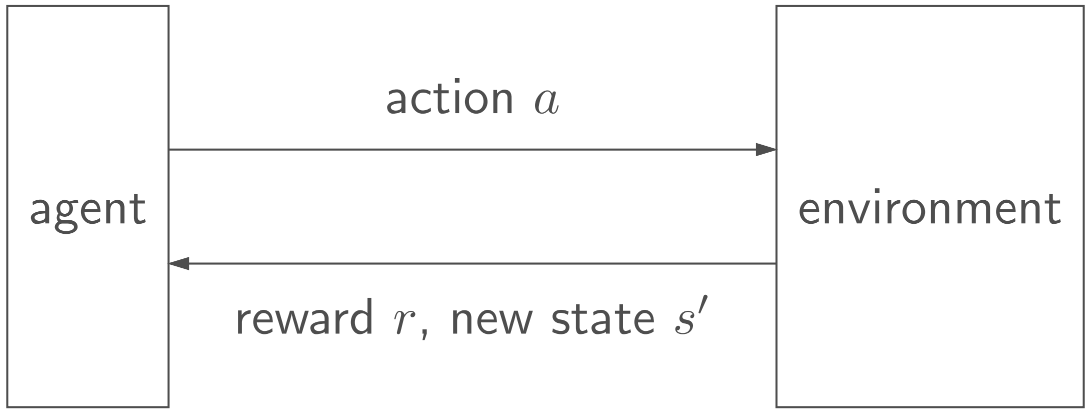
Markov decision process (offline)

- Have mental model of how the world works.
- Find policy to collect maximum rewards.

Reinforcement learning (online)

- Don't know how the world works.
- Perform actions in the world to find out and collect rewards.

Reinforcement learning framework



Algorithm: reinforcement learning template

For $t = 1, 2, 3, \dots$

Choose action $a_t = \pi_{\text{act}}(s_{t-1})$ (**how?**)

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

Volcano crossing

Run (or press ctrl-enter)

		-50	20
		-50	
2			

Utility: 2

a	r	s
		(2,1)
W	0	(2,1)
W	0	(2,1)
N	0	(1,1)
W	0	(1,1)
N	0	(1,1)
E	0	(1,2)
S	0	(2,2)
W	0	(2,1)
N	0	(2,2)
N	0	(3,2)
S	0	(3,2)
W	2	(3,1)

Outline

MDPs: model-based methods

MDPs: model-free methods

MDPs: SARSA

MDPs: Q-learning

MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions

Model-Based Value Iteration

Data: $s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$

Key idea: model-based learning

Estimate the MDP: $T(s, a, s')$ and $\text{Reward}(s, a, s')$

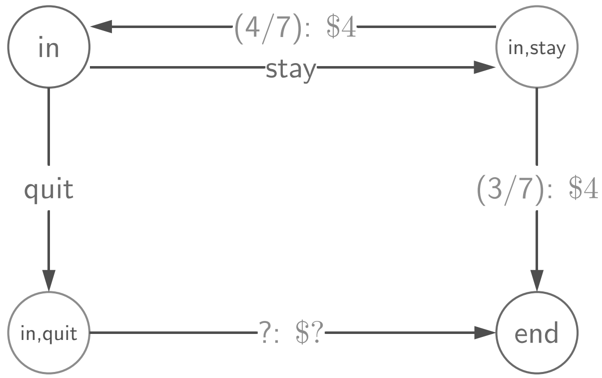
Transitions:

$$\hat{T}(s, a, s') = \frac{\#\text{ times } (s, a, s') \text{ occurs}}{\#\text{ times } (s, a) \text{ occurs}}$$

Rewards:

$$\widehat{\text{Reward}}(s, a, s') = r \text{ in } (s, a, r, s')$$

Model-Based Value Iteration

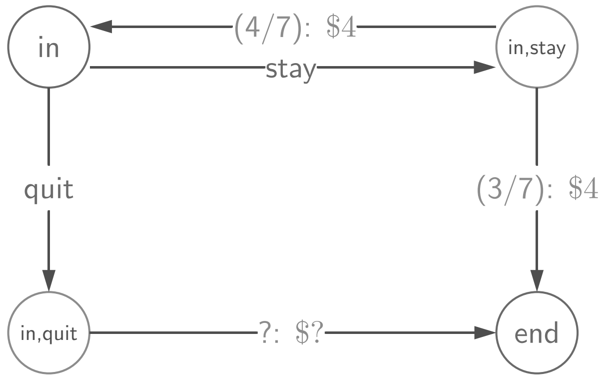


Data (following policy $\pi(s) =$ stay):

[in; stay, 4, end]

- Estimates converge to true values (under certain conditions)
- With estimated MDP $(\hat{T}, \widehat{\text{Reward}})$, compute policy using value iteration

Problem



Problem: won't even see (s, a) if $a \neq \pi(s)$ ($a =$ quit)

Key idea: exploration

To do reinforcement learning, need to explore the state space.

Solution: need π to **explore** explicitly (more on this later)

Outline

MDPs: model-based methods

MDPs: model-free methods

MDPs: SARSA

MDPs: Q-learning

MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions

From model-based to model-free

$$\hat{Q}_{\text{opt}}(s,a) = \sum_{s'} \hat{T}(s,a,s')[\widehat{\text{Reward}}(s,a,s') + \gamma \hat{V}_{\text{opt}}(s')]$$

All that matters for prediction is (estimate of) $Q_{\text{opt}}(s,a)$.

Key idea: model-free learning

Try to estimate $Q_{\text{opt}}(s,a)$ directly.

Model-free Monte Carlo

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

Recall:

$Q_\pi(s, a)$ is expected utility starting at s, first taking action a, and then following policy π

Utility:

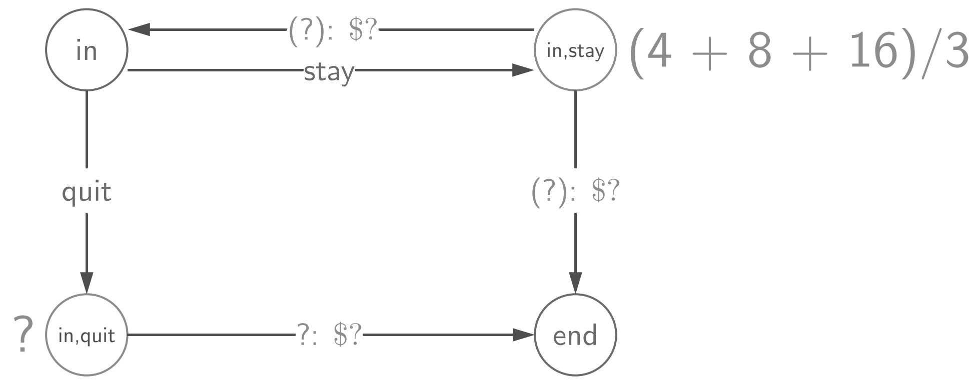
$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$$

Estimate:

$\hat{Q}_\pi(s, a) =$ average of u_t where $s_{t-1} = s, a_t = a$

(and s, a doesn't occur in $s_0, \cdots, s_{t-2}$)

Model-free Monte Carlo



Data (following policy $\pi(s) = \text{stay}$):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Note: we are estimating Q_π now, not Q_{opt}

Definition: on-policy versus off-policy

On-policy: estimate the value of data-generating policy

Off-policy: estimate the value of another policy

Model-free Monte Carlo (equivalences)

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

Original formulation

$$\hat{Q}_\pi(s,a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a$$

Equivalent formulation (convex combination)

On each (s,a,u):

$$\eta = \frac{1}{1 + (\#\text{ updates to } (s,a))}$$

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

Model-free Monte Carlo (equivalences)

Equivalent formulation (convex combination)

On each (s, a, u):

$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta u$$

Equivalent formulation (stochastic gradient)

On each (s, a, u):

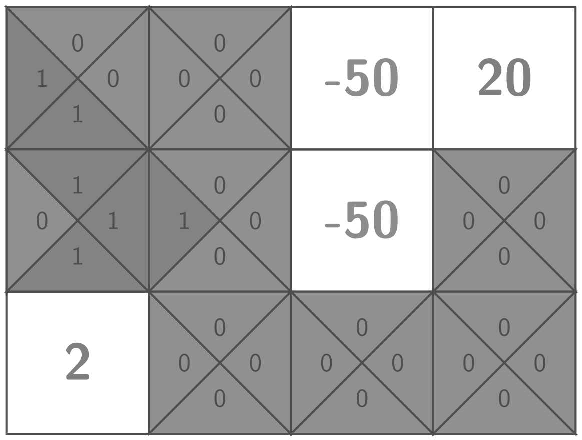
$$\hat{Q}_\pi(s, a) \leftarrow \hat{Q}_\pi(s, a) - \eta[\underbrace{\hat{Q}_\pi(s, a)}_{\text{prediction}} - \underbrace{u}_{\text{target}}]$$

Implied objective: least squares regression

$$(\hat{Q}_\pi(s, a) - u)^2$$

Volcanic model-free Monte Carlo

Run (or press ctrl-enter)



a	r	s
		(2,1)
N	0	(1,1)
W	0	(1,1)
S	0	(2,1)
E	0	(2,2)
W	0	(2,1)
S	2	(3,1)

Utility: 2

Outline

MDPs: model-based methods

MDPs: model-free methods

MDPs: SARSA

MDPs: Q-learning

MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions

Using the utility

Data (following policy $\pi(s) =$ stay):

[in; stay, 4, end]	$u = 4$
[in; stay, 4, in; stay, 4, end]	$u = 8$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 12$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 16$

Algorithm: model-free Monte Carlo

On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta \underbrace{u}_{\text{data}}$$

Using the reward + Q-value

Current estimate: $\hat{Q}_\pi(s, \text{stay}) = 11$

Data (following policy $\pi(s) = \text{stay}$):

[in; stay, 4, end]	$4 + 0$
[in; stay, 4, in; stay, 4, end]	$4 + 11$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$4 + 11$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$4 + 11$

Algorithm: SARSA

On each (s, a, r, s', a'):

$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta[\underbrace{r}_{\text{data}} + \gamma \underbrace{\hat{Q}_\pi(s', a')}_{\text{estimate}}]$$

Model-free Monte Carlo versus SARSA

Key idea: bootstrapping

SARSA uses estimate $\hat{Q}_\pi(s, a)$ instead of just raw data u.

u	$r + \hat{Q}_\pi(s', a')$
based on one path	based on estimate
unbiased	biased
large variance	small variance
wait until end to update	can update immediately

think and share

Question

Which of the following algorithms allows you to estimate $Q^*(s, a)$ (select all that apply)?

(a) model-based value iteration

(b) model-free Monte Carlo

(c) SARSA

Outline

MDPs: model-based methods

MDPs: model-free methods

MDPs: SARSA

MDPs: Q-learning

MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions

Q-learning

Problem: model-free Monte Carlo and SARSA only estimate Q_π, but want Q_{opt} to act optimally

Output	**MDP**	**reinforcement learning**
Q_π	policy evaluation	model-free Monte Carlo, SARSA
Q_{opt}	value iteration	**Q-learning**

Q-learning

Bellman optimality equation:

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')]$$

Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s,a,r,s'):

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} + \eta\underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}$$

Recall: $\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s',a')$

SARSA versus Q-learning

Algorithm: SARSA

On each (s, a, r, s', a'):

$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta(r + \gamma\hat{Q}_\pi(s', a'))$$

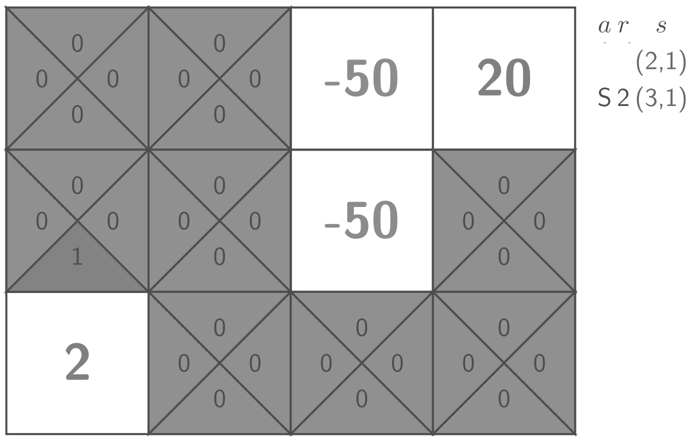
Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s, a, r, s'):

$$\hat{Q}_{\text{opt}}(s, a) \leftarrow (1 - \eta)\hat{Q}_{\text{opt}}(s, a) + \eta(r + \gamma \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s', a'))]$$

Volcanic SARSA and Q-learning

Run (or press ctrl-enter)



Utility: 2

Off-Policy versus On-Policy

Definition: on-policy versus off-policy

On-policy: evaluate or improve the data-generating policy

Off-policy: evaluate or learn using data from another policy

	on-policy	**off-policy**
policy evaluation	Monte Carlo SARSA	
policy optimization		Q-learning

Reinforcement Learning Algorithms

Algorithm	**Estimating**	**Based on**
Model-Based Monte Carlo	$\hat{T}, \hat{R}$	$s_0, a_1, r_1, s_1, \ldots$
Model-Free Monte Carlo	$\hat{Q}_\pi$	u
SARSA	$\hat{Q}_\pi$	$r + \hat{Q}_\pi$
Q-Learning	$\hat{Q}_{\text{opt}}$	$r + \hat{Q}_{\text{opt}}$

Outline

MDPs: model-based methods

MDPs: model-free methods

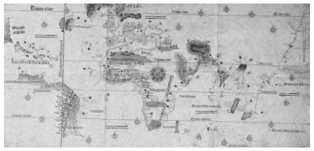
MDPs: SARSA

MDPs: Q-learning

MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions



Exploration

Algorithm: reinforcement learning template

For $t = 1, 2, 3, \dots$

  Choose action $a_t = \pi_{\text{act}}(s_{t-1})$ (**how?**)

  Receive reward r_t and observe new state s_t

  Update parameters (**how?**)

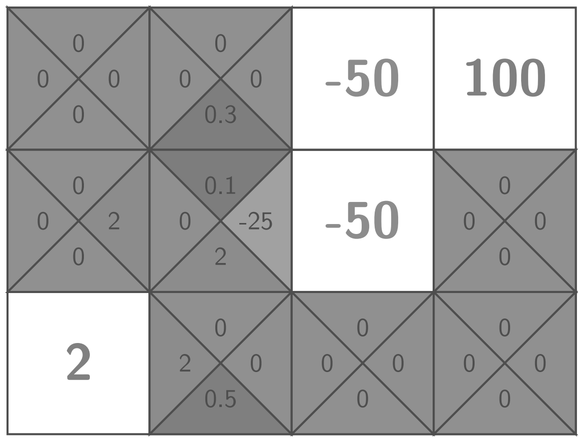
$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

Which **exploration policy** π_{act} to use?

No exploration, all exploitation

Attempt 1: Set $\pi_{\text{act}}(s) = \arg\max_{a \in \text{Actions}(s)} \hat{Q}_{\text{opt}}(s,a)$

Run (or press ctrl-enter)



a	r	s
		(2,1)
E	0	(2,2)
S	0	(3,2)
W	2	(3,1)

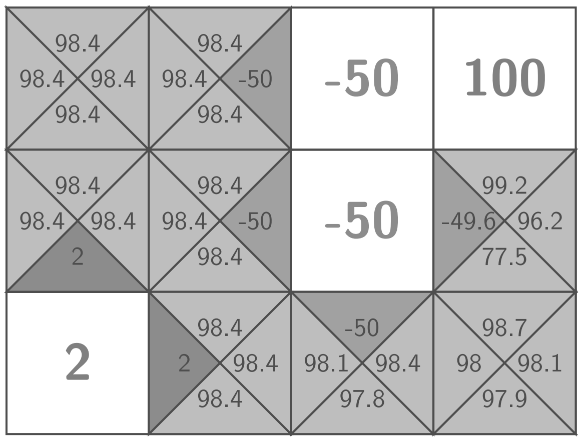
Average (lifetime) utility: 1.95

Problem: $\hat{Q}_{\text{opt}}(s,a)$ estimates are inaccurate, **too greedy**!

No exploitation, all exploration

Attempt 2: Set $\pi_{\text{act}}(s) = \text{random from Actions}(s)$

Run (or press ctrl-enter)



$a\ r\ \ s$
(2,1)
S 2 (3,1)

Average (lifetime) utility: -19.15

Problem: average utility is low because exploration is **not guided**

Exploration/exploitation tradeoff

Key idea: balance

Need to balance **exploration** and **exploitation**.

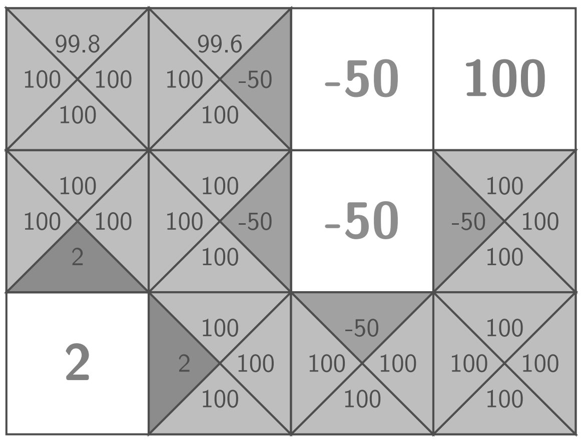
Examples from life: restaurants, routes, research

Epsilon-greedy

Algorithm: epsilon-greedy policy

$$\pi_{\text{act}}(s) = \begin{cases} \arg\max_{a \in \text{Actions}} \hat{Q}_{\text{opt}}(s, a) & \text{probability } 1 - \epsilon, \\ \text{random from Actions}(s) & \text{probability } \epsilon. \end{cases}$$

Run (or press ctrl-enter)



a	r	s
		(2,1)
W	0	(2,1)
N	0	(1,1)
S	0	(2,1)
W	0	(2,1)
W	0	(2,1)
E	0	(2,2)
S	0	(3,2)
E	0	(3,3)
E	0	(3,4)
N	0	(2,4)
N	100	(1,4)

Average (lifetime) utility: 30.71

Outline

MDPs: model-based methods

MDPs: model-free methods

MDPs: SARSA

MDPs: Q-learning

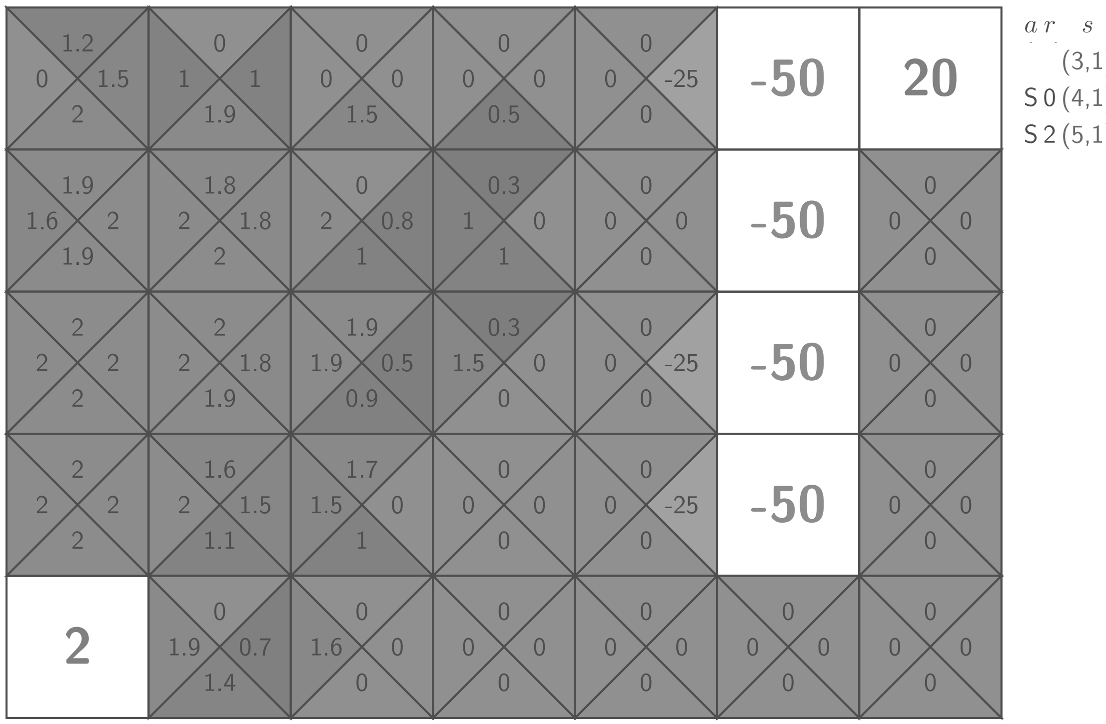
MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions

Generalization

Problem: large state spaces, hard to explore



Average (lifetime) utility: 0.44

Q-learning

Stochastic gradient update:

$$\hat{Q}_{\textsf{opt}}(s,a) \leftarrow \hat{Q}_{\textsf{opt}}(s,a) - \eta[\underbrace{\hat{Q}_{\textsf{opt}}(s,a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\textsf{opt}}(s'))}_{\text{target}}]$$

This is **rote learning**: every $\hat{Q}_{\textsf{opt}}(s,a)$ has a different value

Problem: doesn't generalize to unseen states/actions

Function approximation

Key idea: linear regression model

Define **features** $\phi(s,a)$ and **weights** $\mathbf{w}$:

$$\hat{Q}_{\text{opt}}(s,a;\mathbf{w}) = \mathbf{w} \cdot \phi(s,a)$$

Example: features for volcano crossing

$$\phi_1(s,a) = \mathbf{1}[a = \text{W}] \qquad \phi_7(s,a) = \mathbf{1}[s = (5,*)]$$

$$\phi_2(s,a) = \mathbf{1}[a = \text{E}] \qquad \phi_8(s,a) = \mathbf{1}[s = (*,6)]$$

... ...

Function approximation

Algorithm: Q-learning with function approximation

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s, a)$$

Implied objective function:

$$(\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}})^2$$

Outline

MDPs: model-based methods

MDPs: model-free methods

MDPs: SARSA

MDPs: Q-learning

MDPs: epsilon-greedy

MDPs: function approximation

MDPs: recap and extensions

Summary of MDPs

- **Markov decision processes** (MDPs) cope with uncertainty

- Solutions are **policies** rather than paths

- **Policy evaluation** computes policy value (expected utility)

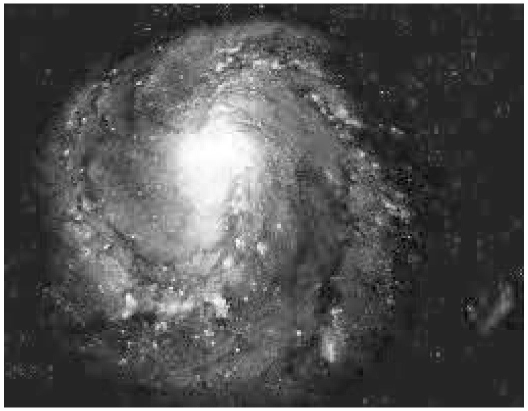
- **Value iteration** computes optimal value (maximum expected utility) and optimal policy

- Main technique: write recurrences $\rightarrow$ algorithm

Summary of Reinforcement Learning

- Online setting: learn and take actions in the real world!
- Monte Carlo: estimate transitions, rewards, Q-values from data
- Bootstrapping: update towards target that depends on estimate rather than just raw data

Covering the unknown



Epsilon-greedy: balance the exploration/exploitation tradeoff

Function approximation: can generalize to unseen states

Challenges in reinforcement learning

Binary classification (sentiment classification, SVMs):

- Stateless, full supervision

Reinforcement learning (flying helicopters, Q-learning):

- Stateful, partial supervision

Key idea: partial supervision

Reward feedback, but not given the solution directly.

Key idea: state

Rewards depend on previous actions $\Rightarrow$ can have delayed rewards.

States and information

	stateless	**state**
full supervision	supervised learning (binary classification)	supervised imitation learning (structured prediction)
partial supervision	multi-armed bandits	reinforcement learning

Deep reinforcement learning

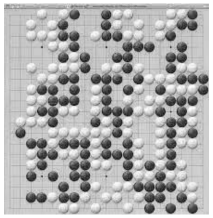
just use a neural network for $\hat{Q}_{\text{opt}}(s, a)$, π_{opt}, T, etc

Playing Atari [Google DeepMind, 2013]:

- last 4 frames (images) $\Rightarrow$ 3-layer NN $\Rightarrow$ keystroke
- ϵ-greedy, train over 10M frames with 1M replay memory
- Human-level performance on some games (breakout), less good on others (space invaders)

Deep reinforcement learning

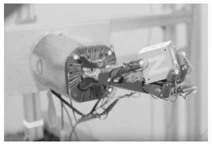
- Policy gradient: train a policy $\pi(a \mid s)$ (say, a neural network) to directly maximize expected reward
- Google DeepMind's AlphaGo (2016), AlphaZero (2017)



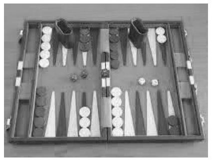
- Stanford CS224R course:

https://cs224r.stanford.edu/

Applications



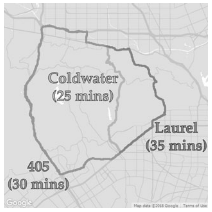
Robotics Applications: learning dexterous manipulation, control helicopter to do maneuvers in the air



Backgammon: TD-Gammon plays 1-2 million games against itself, human-level performance

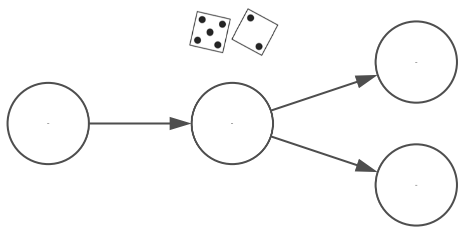
Games: DQN solving Atari Games, openAI Five playing Dota.

Managing datacenters; actions: bring up and shut down machine to minimize time/cost



Routing Autonomous Cars: bring down the total latency of vehicles on the road

Markov decision processes: against nature (e.g., Blackjack)



Next time...

Adversarial games: against opponent (e.g., chess)