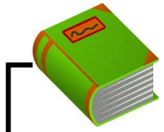
MDPs: reinforcement learning

Markov decision process



Definition: Markov decision process

States: the set of states

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

Actions(s): possible actions from state s

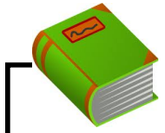
$T(s, a, s')$: probability of s' if take action a in state s

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)

Unknown transitions and rewards



Definition: Markov decision process

States: the set of states

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

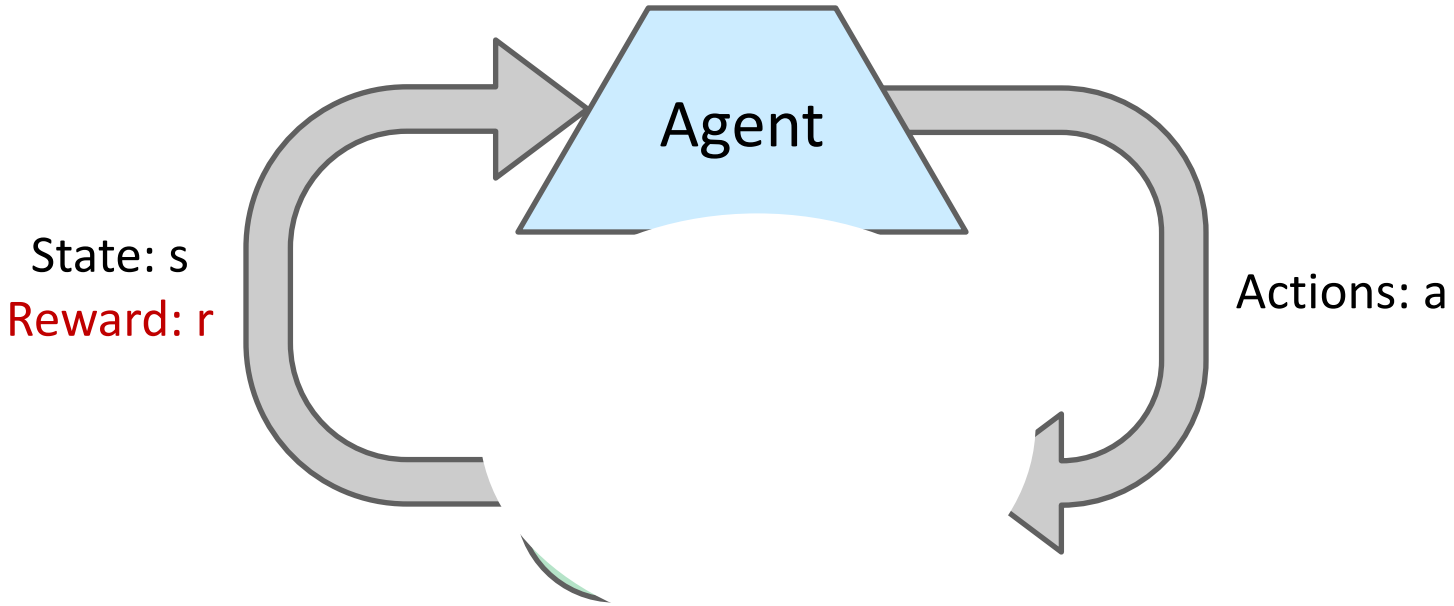
Actions(s): possible actions from state s

IsEnd(s): whether at end of game

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

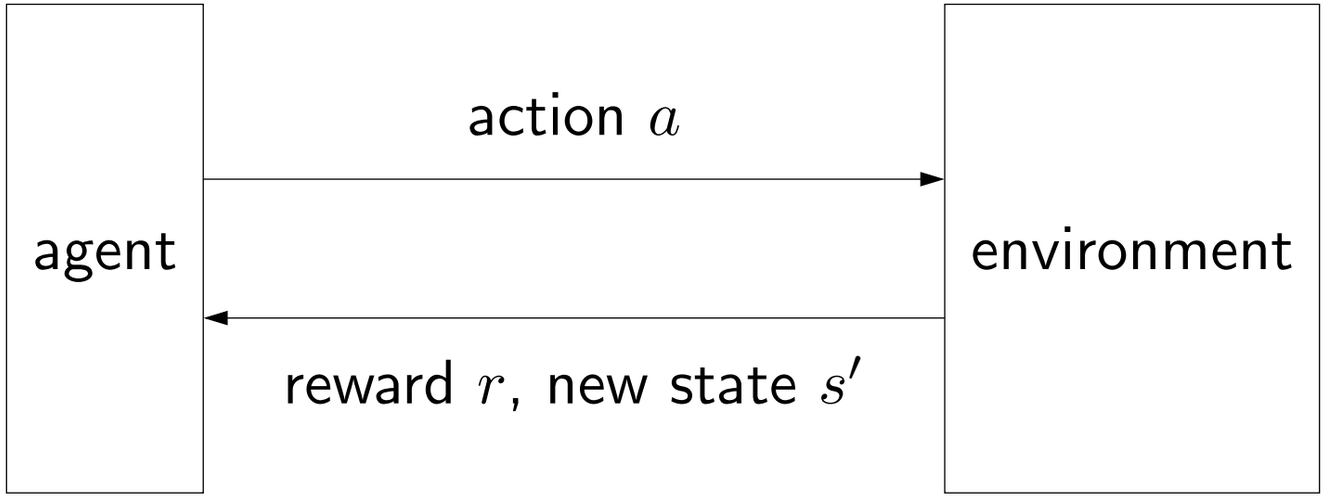
reinforcement learning!

Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Reinforcement learning framework



Algorithm: reinforcement learning template

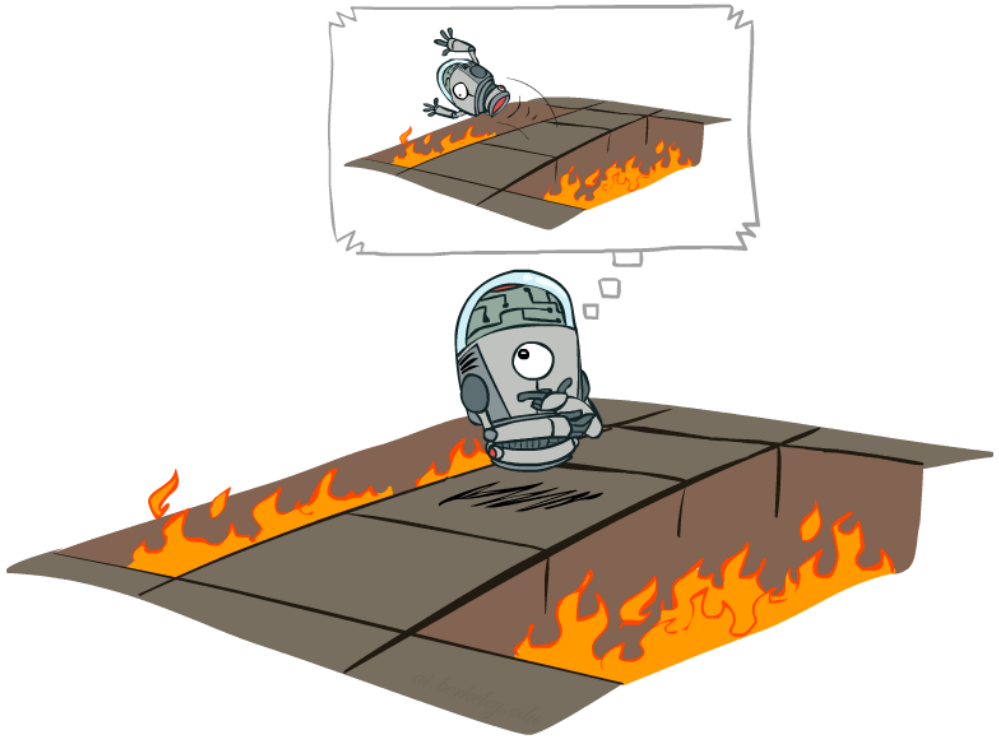
For $t = 1, 2, 3, \ldots$

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

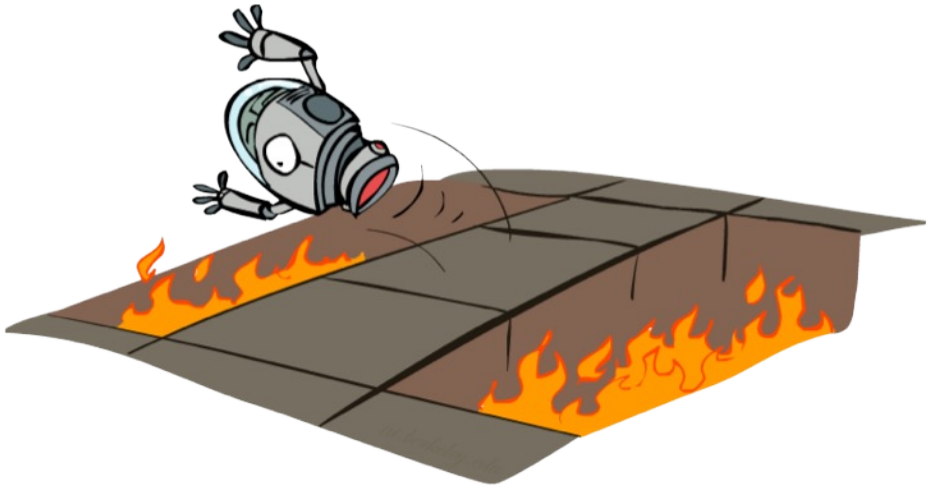
Receive reward r_t and observe new state s_t

Update parameters (**how?**)

Offline (MDPs) vs. Online (RL)



Offline Solution



Online Learning

MDPs: model-based methods

Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s’ for each s, a
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s’)

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

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 Reward(s, a, s')

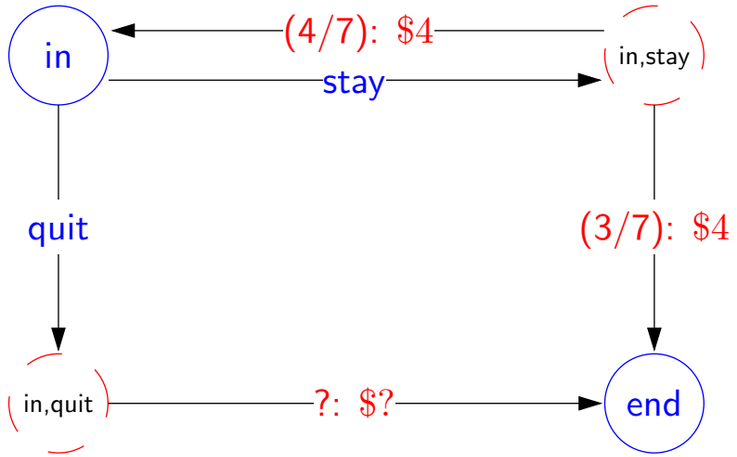
Transitions:

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

Rewards:

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

Model-Based Value Iteration

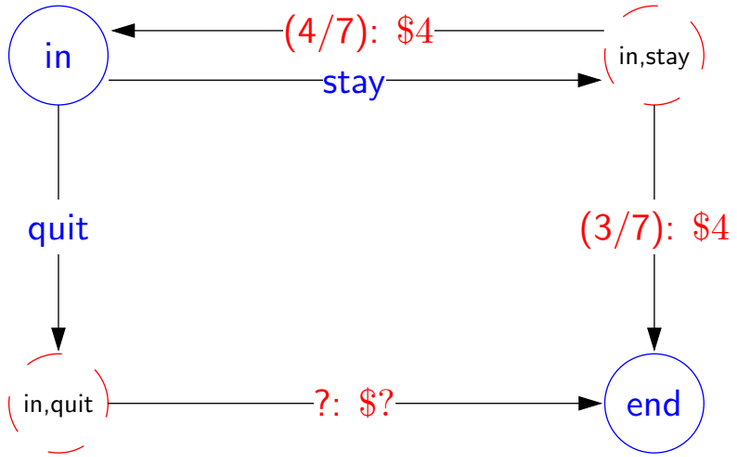


Data (following policy $\pi(s) = \text{stay}$):

[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)

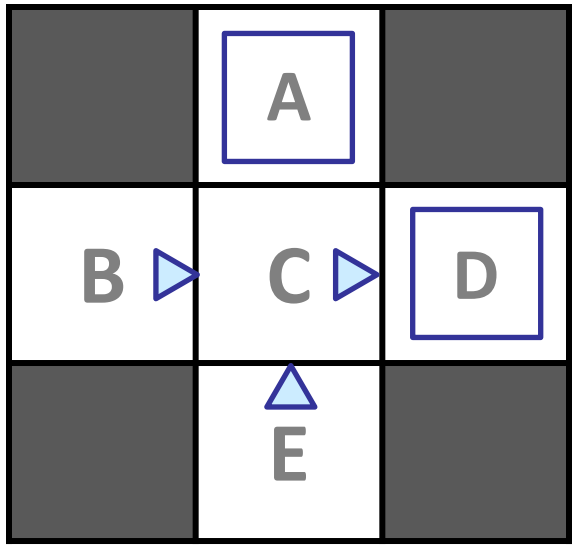
Key idea: exploration

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

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

Example: Model-Based Learning

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

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.

Example: Expected Age

Goal: Compute expected age of cs188 students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

MDPs: model-free methods

Model-free Monte Carlo

Data (following policy π):

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

Recall:

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

Utility:

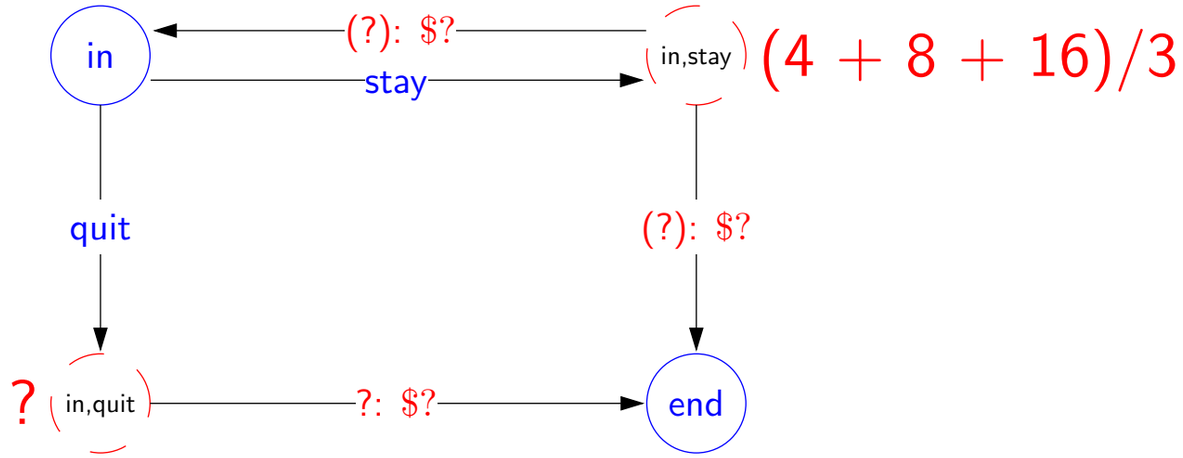
$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$$

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



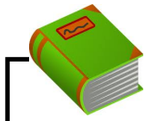
Data (following policy $\pi(s) = \text{stay}$):

[in;stay, 4, end]

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

[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):

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

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):

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

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

[whiteboard: u_1, u_2, u_3]

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

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

Sample-Based Policy Evaluation?

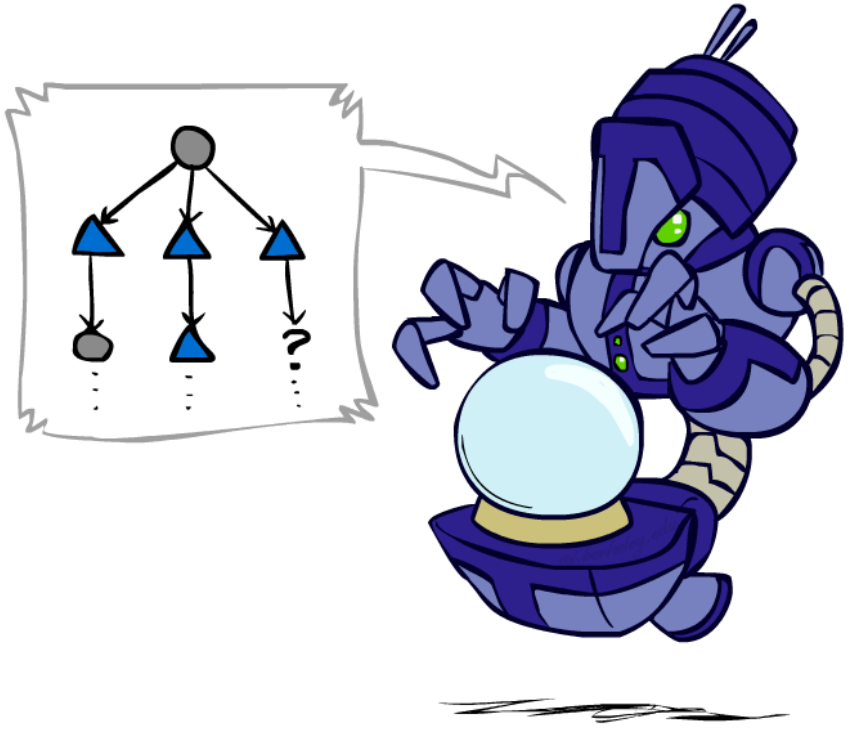
- We want to improve our estimate of V by computing these averages:

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')]$$

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\ldots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$



MDPs: SARSA

Using the utility

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

[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] $u = 16$

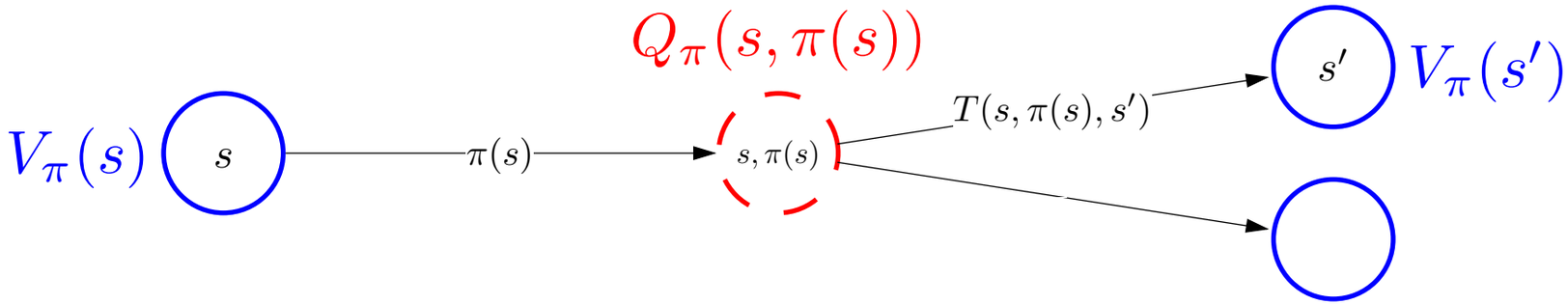
Algorithm: model-free Monte Carlo

On each (s, a, u):

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

Policy evaluation

Plan: define recurrences relating value and Q-value



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

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

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

answer in chat

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

MDPs: Q-learning

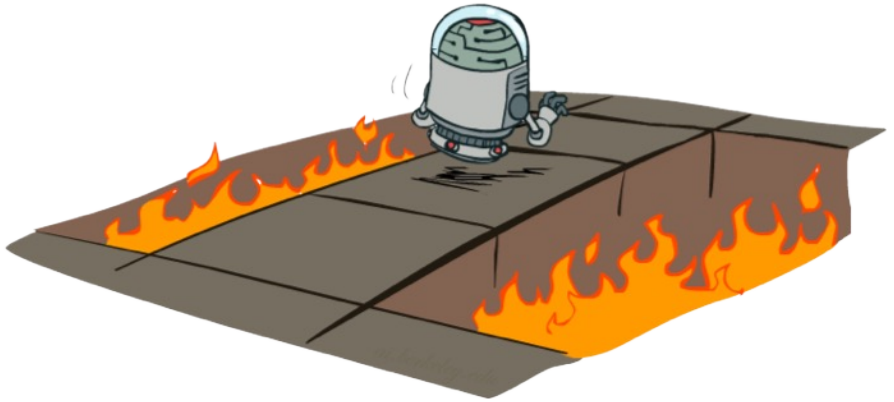
Q-learning

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**

Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Detour: Q-Value Iteration

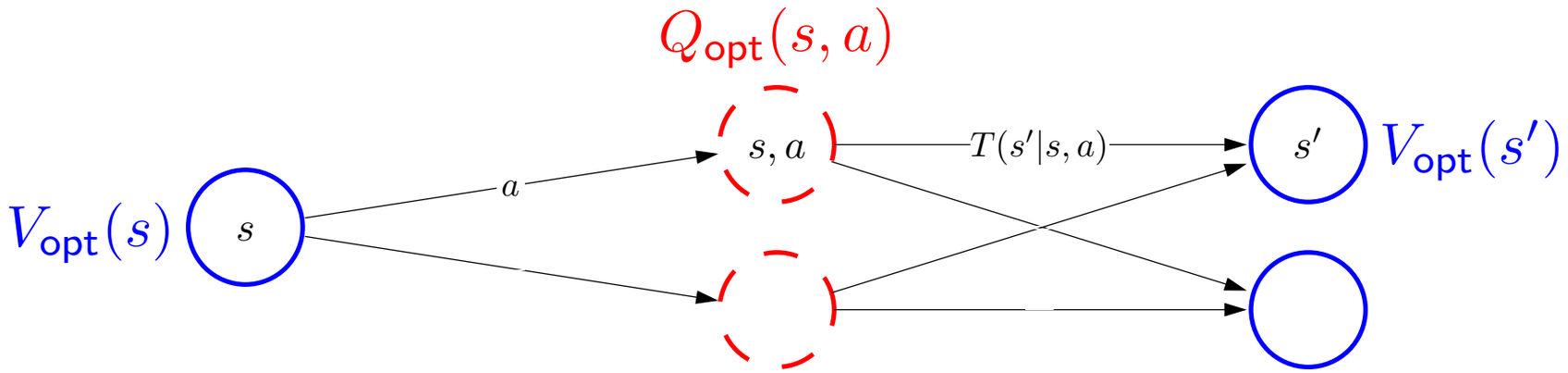
- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Optimal values and Q-values



Optimal value if take action a in state s:

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

Optimal value from state s:

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a) & \text{otherwise.} \end{cases}$$

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'):

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

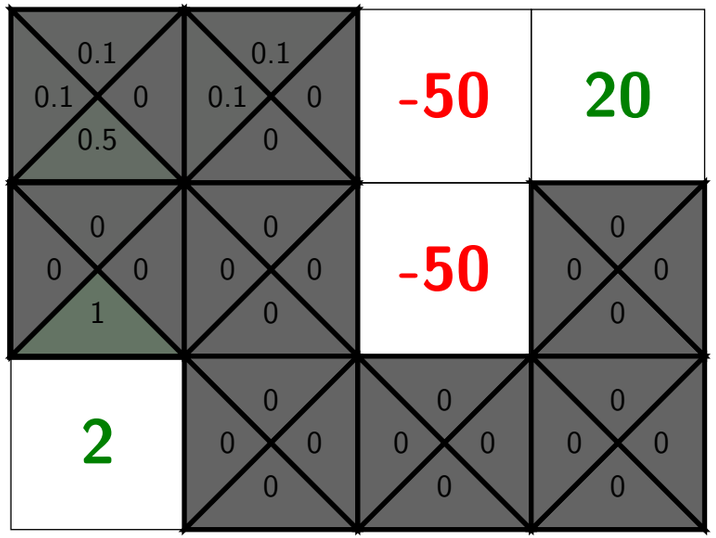
Algorithm: Q-learning [Watkins/Dayan, 1992]

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

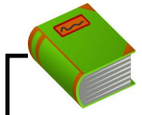
Run (or press ctrl-enter)



Utility: 2

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

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	$\hat{T}, \hat{R}$	$s_0, a_1, r_1, s_1, ...$
Model-Free Monte Carlo	$\hat{Q}_\pi$	u
SARSA	$\hat{Q}_\pi$	$r + \hat{Q}_\pi$
Q-Learning	$\hat{Q}_{\text{opt}}$	$r + \hat{Q}_{\text{opt}}$

MDPs: epsilon-greedy

Exploration

Algorithm: reinforcement learning template

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

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

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

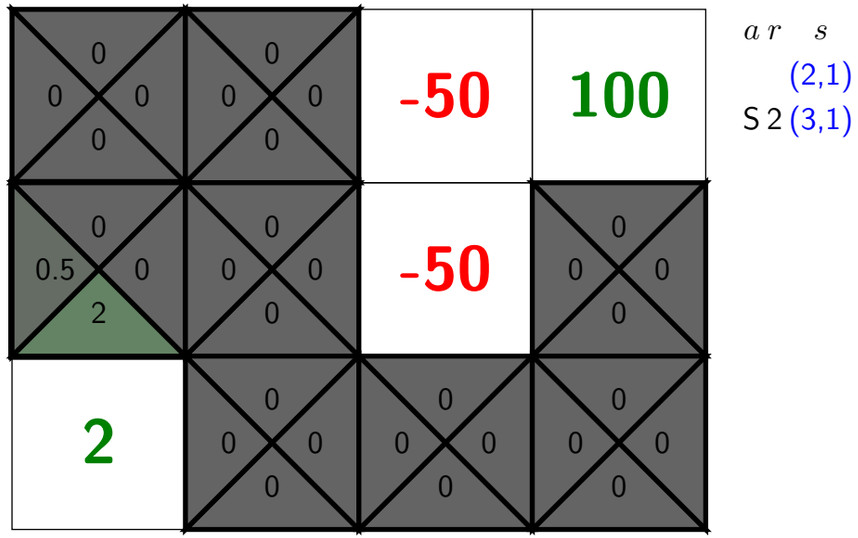
$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

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)
S 2 (3,1)

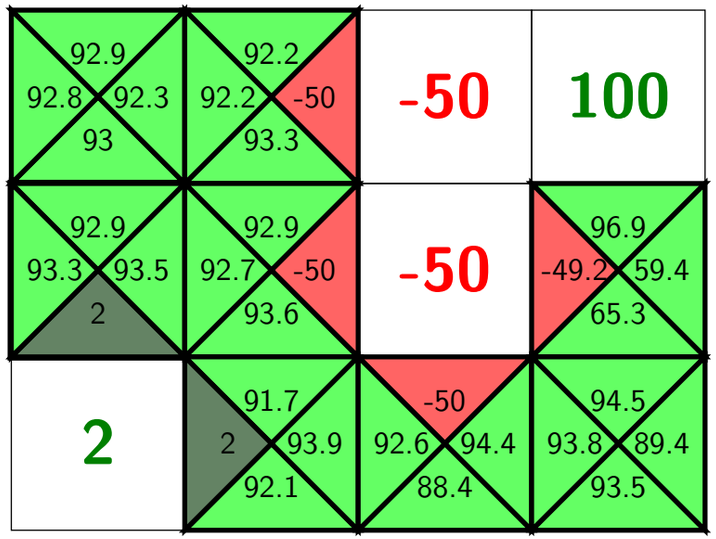
Average (lifetime) utility: 2

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: -17.11

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

Exploration/exploitation tradeoff

Key idea: balance

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

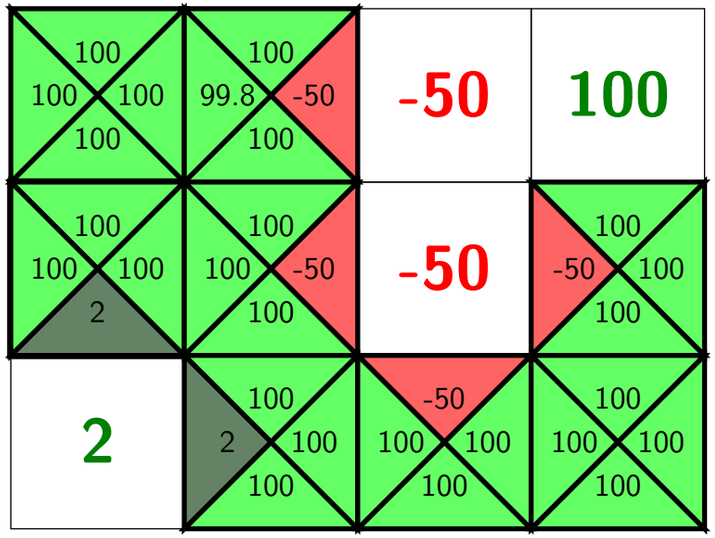
Examples from life: restaurants, routes, research

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)



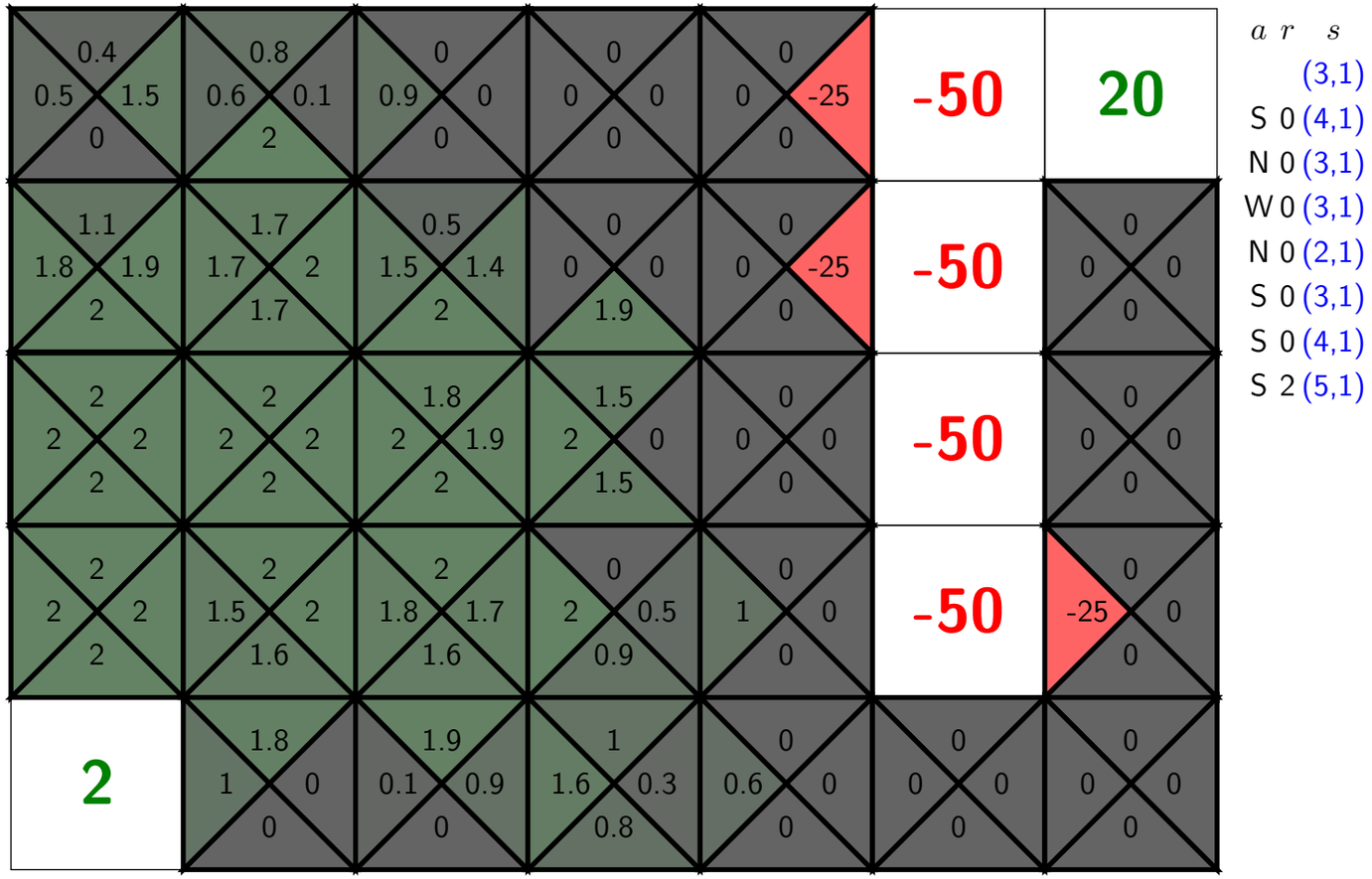
a	r	s
		(2,1)
E	0	(2,2)
S	0	(3,2)
E	0	(3,3)
E	0	(3,4)
N	0	(2,4)
N	100	(1,4)

Average (lifetime) utility: 31.75

MDPs: function approximation

Generalization

Problem: large state spaces, hard to explore



a	r	s
		(3,1)
S	0	(4,1)
N	0	(3,1)
W	0	(3,1)
N	0	(2,1)
S	0	(3,1)
S	0	(4,1)
S	2	(5,1)

Average (lifetime) utility: 0.42

Q-learning

Stochastic gradient update:

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

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

$$\phi_1(s,a) = \mathbf{1}[a = \text{W}] \qquad \phi_7(s,a) = \mathbf{1}[s = (5,*)]$$
$$\phi_2(s,a) = \mathbf{1}[a = \text{E}] \qquad \phi_8(s,a) = \mathbf{1}[s = (*,6)]$$
$$\ldots \qquad \ldots$$

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:

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


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

MDPs: recap

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

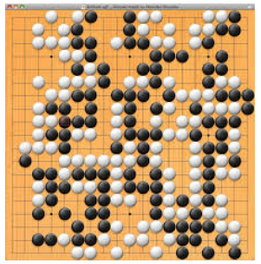
Playing Atari [Google DeepMind, 2013]:

- 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)

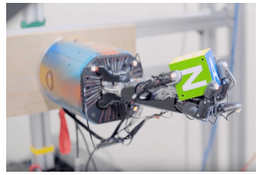
Deep reinforcement learning

- 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)



Applications

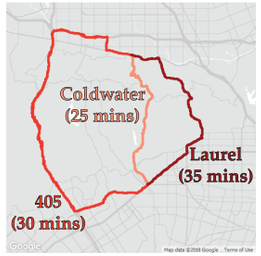


Robotics Applications: learning dexterous manipulation, control helicopter to do maneuvers in the air

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

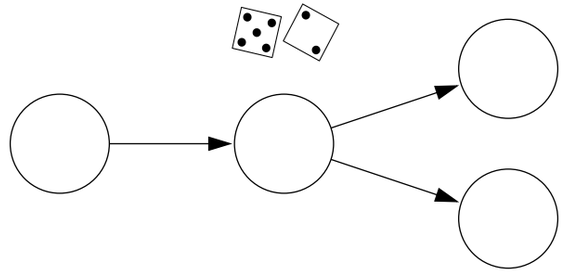
Games: DQN solving Atari Games, openAI Five playing Dota.

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)