Reinforcement Learning

Hwanjo Yu

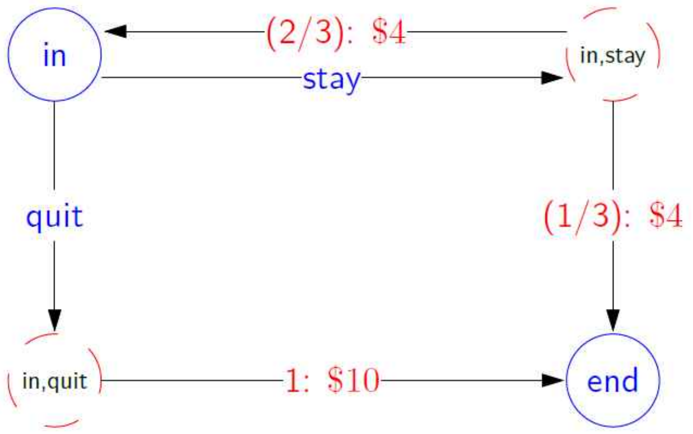
POSTECH

http://di.postech.ac.kr/hwanjoyu

Review: MDPs

Definition: Markov Decision Process

- States: the set of states
- $s_{\text{start}} \in \text{States}$: starting state
- $\text{Actions}(s)$: possible actions from state s
- $T(s, a, s')$: probability of s' if take action a in state s
- $\text{Reward}(s, a, s')$: reward for the transition (s, a, s')
- $\text{IsEnd}(s)$: whether s is an end of game
- $0 \le \gamma \le 1$: discount factor (default: 1)



Review: MDPs

- Following a **policy** $\pi(s)$ 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

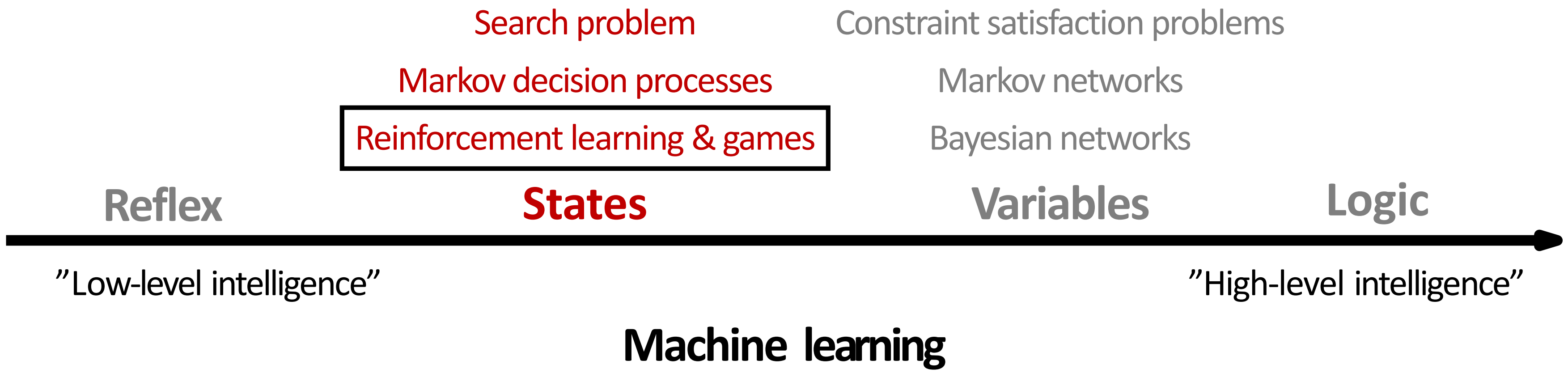
$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

Unknown transitions and rewards

Definition: Markov Decision Process

- States: the set of states
- $s_{\text{start}} \in \text{States}$: starting state
- $\text{Actions}(s)$: possible actions from state s
- $T(s, a, s')$: ?
- $\text{Reward}(s, a, s')$: ?
- $\text{IsEnd}(s)$: whether at end of game
- $0 \leq \gamma \leq 1$: discount factor (default: 1)

Reinforcement learning!


Search problem
Markov decision processes
Reinforcement learning & games
Constraint satisfaction problems
Markov networks
Bayesian networks
Reflex
States
Variables
Logic
"Low-level intelligence"
"High-level intelligence"
Machine learning

Mystery game

Example: mystery game

- For each round r = 1, 2, …
  - You choose A or B.
  - You move to a new state and get some rewards.
- You should take good actions to get rewards, but in order to know which actions are good, we need to explore and try different actions.

From MDPs to reinforcement learning

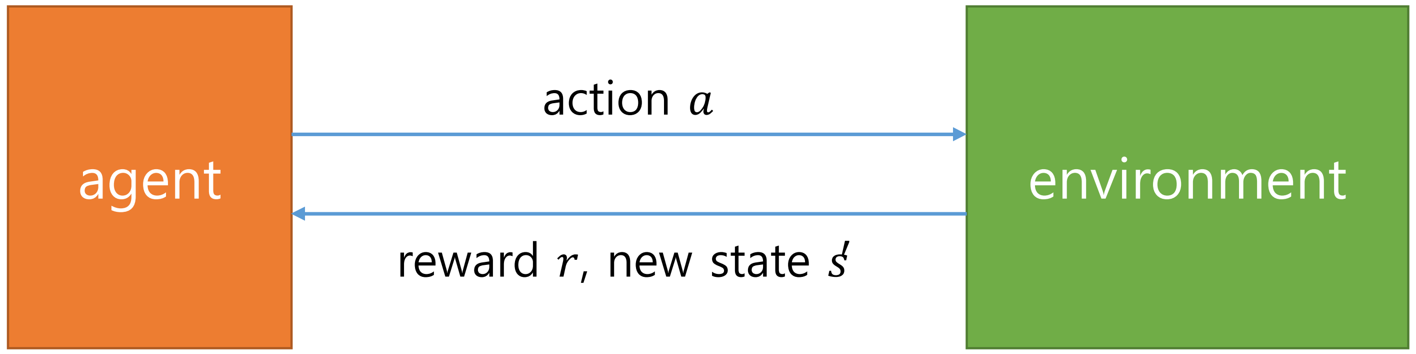
Markov decision process (offline)

- 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 $t = 1,2,3, \ldots$
  - Choose action $a_t = \pi_{\text{act}}\,(s_{t-1})$ (how?)
  - Receive reward r_t and observe new state s_t
  - Update parameters (how?)

Roadmap

Monte Carlo methods

SARSA, Q-learning

Exploitation / exploration

Function approximation

Model-based Monte Carlo

Key idea: model-based learning

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

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

- Transitions:

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

- Rewards:

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

Model-based Monte Carlo

Example: model-based Monte Carlo

- Data (following policy π):
  - **S1**; A, 3, **S1**; B, 0, **S1**; A, 5, **S1**; A, 7, **S2**
- Estimate:
  - $\hat{T}(\mathrm{S1}, \mathrm{A}, \mathrm{S1}) = \frac{2}{3}$
  - $\hat{T}(\mathrm{S1}, \mathrm{A}, \mathrm{S2}) = \frac{1}{3}$
  - $\widehat{\mathrm{Reward}}(\mathrm{S1}, \mathrm{A}, \mathrm{S1}) = \frac{1}{2}(3 + 5) = 4$
  - $\widehat{\mathrm{Reward}}(\mathrm{S1}, \mathrm{A}, \mathrm{S2}) = 7$
- Estimates converge to true values (under certain conditions)

Problem

- Data (following policy π):
  - **S1**; A, 3, **S2**; B, 0, **S1**; A, 5, **S1**; A, 7, **S1**
- Problem:
  - won't even see (s, a) if $a \neq \pi(s)$
- Solution:
  - Need π to explore explicitly (more on this later)

Key idea: exploration

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

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):

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

Recall:

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

Utility:

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

Estimate:

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

Model-free Monte Carlo

Example: model-free Monte Carlo

- Data (following policy π):
  - **S1**; A, 3, **S1**; B, 0, **S1**; A, 5, **S1**; A, 7, **S2**
- Estimate (assume $\gamma = 1$):
  - $\hat{Q}_\pi(\mathrm{S1}, \mathrm{A}) = \frac{1}{3}(15 + 12 + 7) \cong 11.33$

- Note: we are estimating Q_π now, not Q_{opt}; can use policy improvement to get new policy
- Caveat: converges, but still need follow π that explores (**on-policy** algorithm whereas model-based Monte Carlo is **off-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):

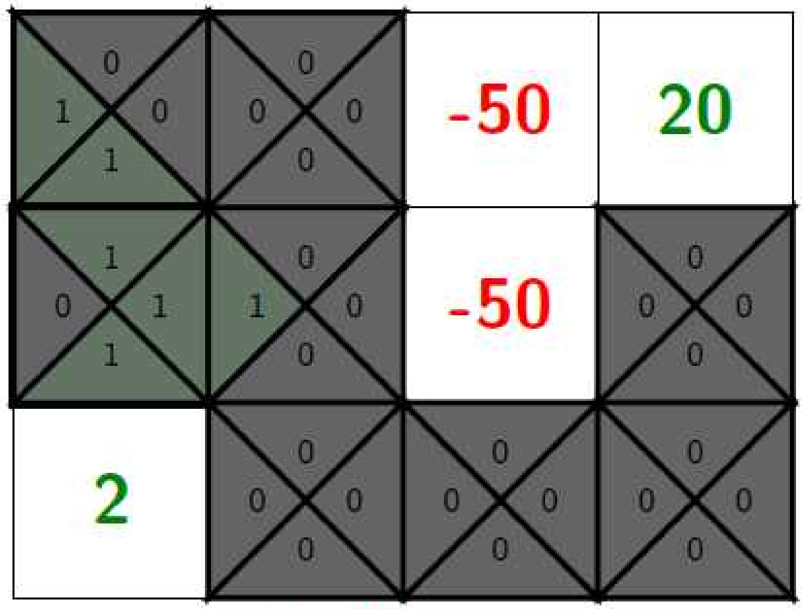
- On each (s, a, u):
  - $\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta u$

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:

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

Volcanic model-free Monte Carlo



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

Roadmap

Monte Carlo methods

SARSA, Q-learning

Exploitation / exploration

Function approximation

SARSA

Data (following policy π):

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

Algorithm: model-free Monte Carlo updates

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

Algorithm: SARSA

- When receive (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 vs SARSA

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

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

- u is only based on one path, so could have large variance, need to wait until end
- $\hat{Q}_\pi(s', a')$ based on estimate, which is more stable, update immediately

Question

Which of the following algorithms allows you to estimate $Q_{\text{opt}}(s, a)$ (select all that apply)?

a. model-based Monte Carlo

b. model-free Monte Carlo

c. SARSA

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

- MDP recurrence (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:

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

Off-policy vs On-policy

On-policy:
- evaluate or improve the data-generating policy
- model-free Monte Carlo, SARSA

Off-policy
- evaluate or learn using data from another policy
- Model-based Monte Carlo, Q-learning

Reinforcement learning algorithms

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

Roadmap

Monte Carlo methods

SARSA, Q-learning

Exploitation / exploration

Function approximation

Exploration

Algorithm: reinforcement learning template

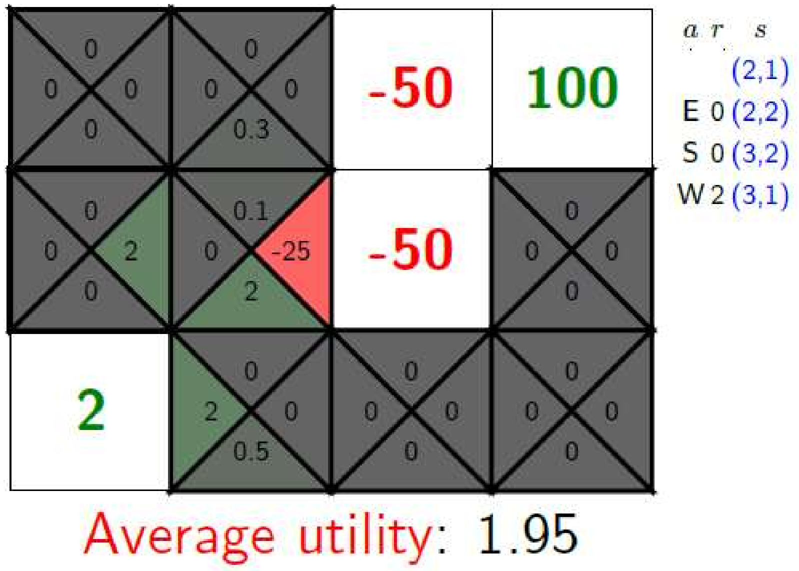
- For $t = 1,2,3, \ldots$
  - Choose action $a_t = \pi_{\text{act}}(s_{t-1})$ (how?)
  - Receive reward r_t and observe new state s_t
  - Update parameters (how?)

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

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

No exploration, all exploitation

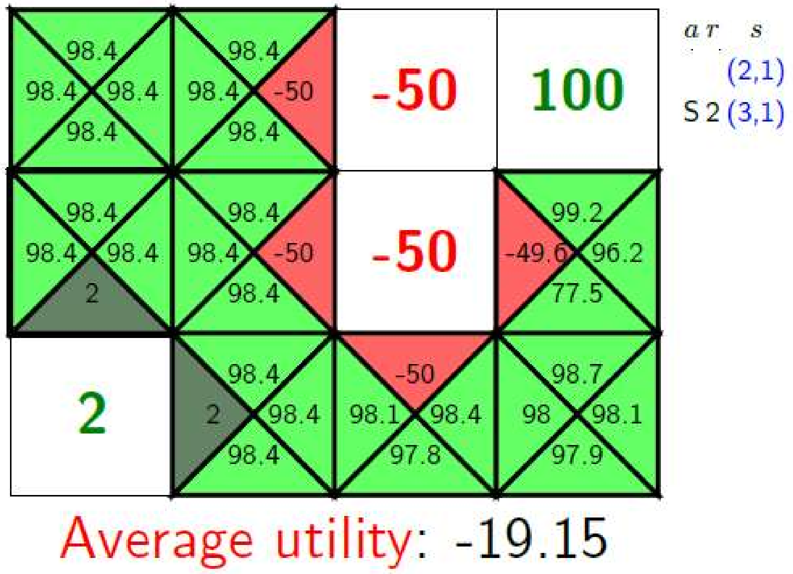
- Attemp 1: Set $\pi_{\text{act}}(s) = arg \max_{a \in \text{Actions}(s)} \hat{Q}_{\text{opt}}(s, a)$



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

No exploitation, all exploration

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

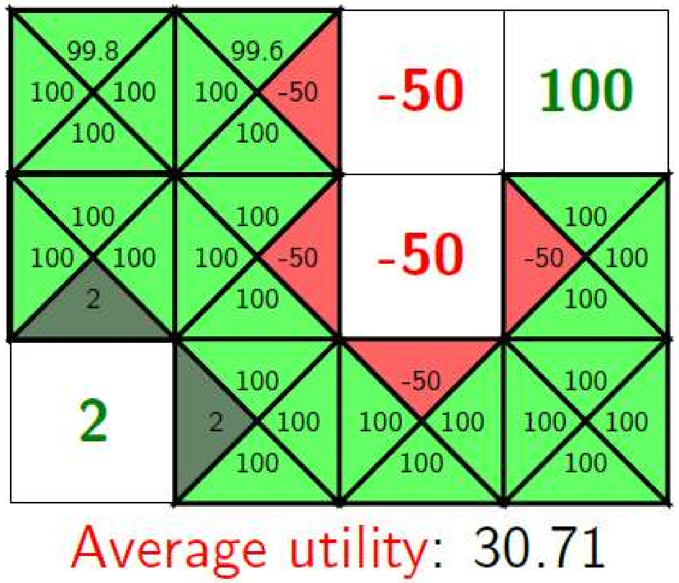


- Problem: average utility is low because exploration is not guided

ϵ-greedy

Algorithm: ϵ-greedy policy

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



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)

Roadmap

Monte Carlo methods

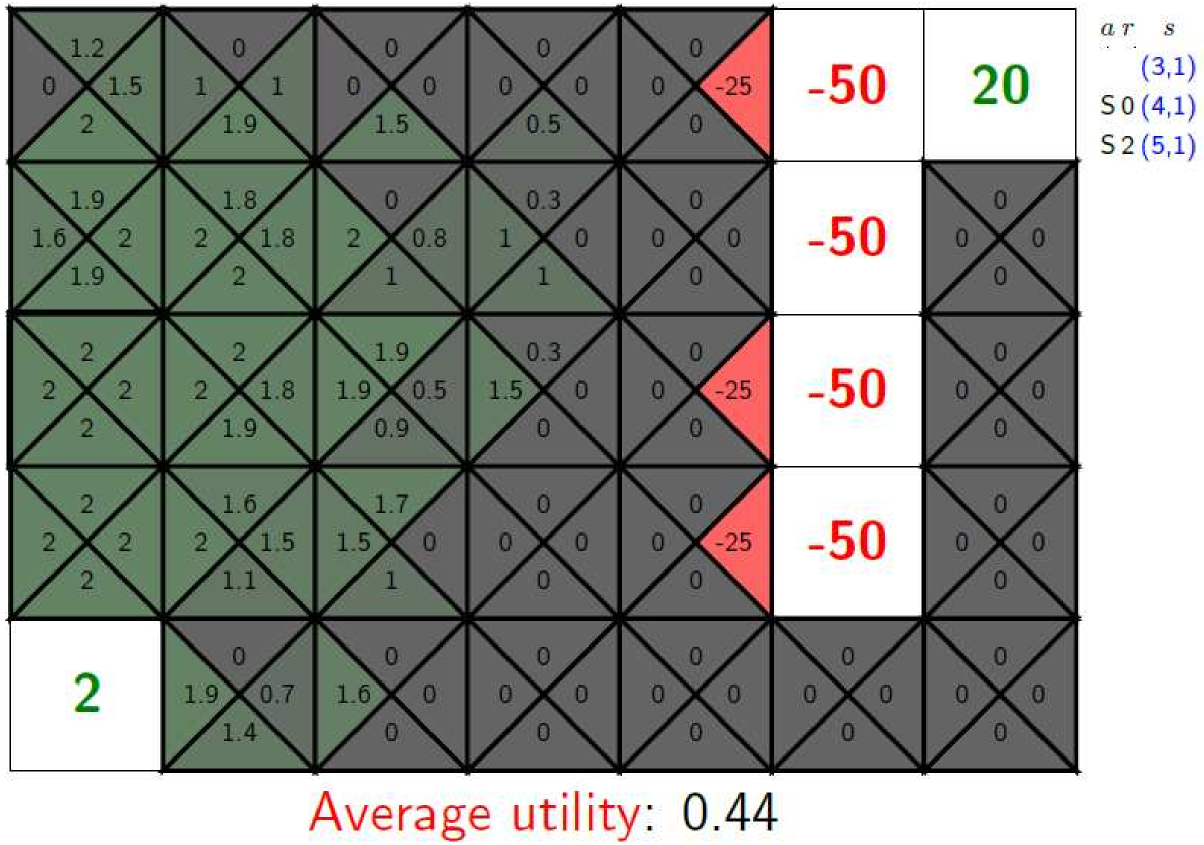
SARSA, Q-learning

Exploitation / exploration

Function approximation

Generalization

Problem: large state spaces, hard to explore



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

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

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:

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

Covering the unknown

- ϵ-greedy: balance the exploration/exploitation tradeoff
- Function approximation: can generalize to unseen states

Summary: MDP and reinforcement learning

MDP: cope with uncertainty (unlike search problems)

- Solutions are **policies** rather than paths
- **Policy evaluation** computes policy value (expected utility)
- **Value iteration** computes optimal value (maximum expected utility) and optimal policy

Reinforcement learning: learn and take actions online

- **Monte Carlo**: estimate transitions, rewards, Q-values from data only
- **SARSA, Q-learning**: estimate Q-values from data and previous estimation
- **Exploitation / exploration**: balance learning and maximizing utility
- **Function approximation**: use machine learning to generalize to unseen states

Challenges in reinforcement learning

Binary classification (sentiment classification, SVMs):

- Stateless, full feedback

Reinforcement learning (flying helicopters, Q-learning):

- Stateful, partial feedback

Key idea: partial supervision

- Reward feedback, but not given the solution directly.

Key idea: state

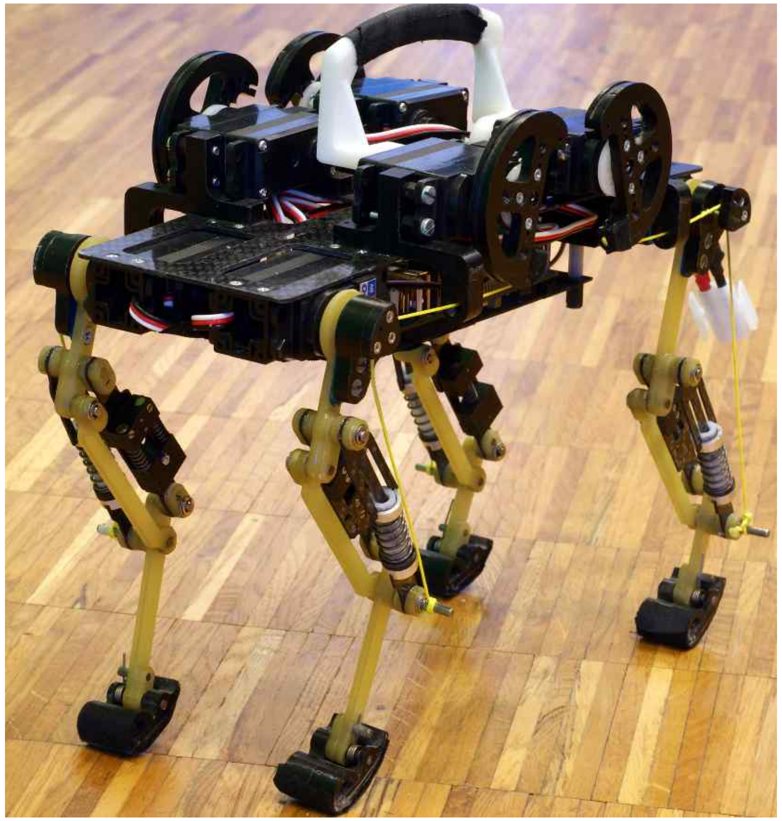
- Rewards depend on previous actions => can have delayed rewards.

Crawling robot

Goal: maximize distance travelled by robot

Markov decision process (MDP):

- States: positions (4 possibilities) for each of 2 servos
- Actions: choose a servo, move it up/down
- Transitions: move into new position (unknown)
- Rewards: distance travelled (unknown)



Deep reinforcement learning

Playing Atari [Google DeepMind, 2013]:

- Just use a neural network for $\hat{Q}_{\text{opt}}(s, a)$
- Last 4 frames (images) => 3-layer NN => keystroke
- ϵ-greedy, train over 10M frames with 1M replay memory
- https://www.youtube.com/watch?v=V1eYniJ0Rnk

Deep reinforcement learning

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

- Andrej Karpathy's blog post

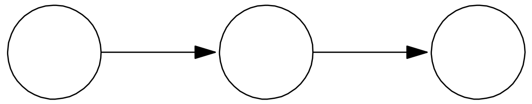
http://karpathy.github.io/2016/05/31/rl

Application

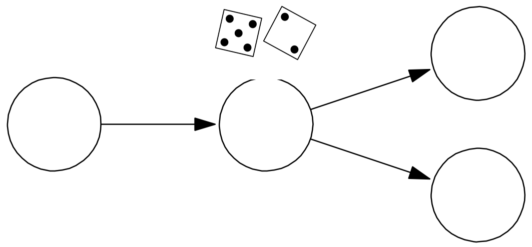
- Autonomous helicopters: control helicopter to do maneuvers in the air
- Backgammon: TD-Gammon plays 1-2 million games against itself, human-level performance
- Elevator scheduling; send which elevators to which floors to maximize throughput of building
- Managing datacenters; actions: bring up and shut down machine to minimize time/cost

State-based Models

- Search problems: you control everything



- MDP, RL: against nature (e.g., Blackjack)



Next time…

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