L17. Reinforcement Learning (Extensions)

L17. Reinforcement Learning (Extensions)

1. One step→ Multiple steps ahead (links to Monte Carlo)

2. Model-free→ Include the model of environment (links to Dynamic programming)

3. Tabular → Continuous space (links to supervised learning)

4. Single agent → Multiple Agents (links to Game theory: Stochastic Game)

Dynamic Programming

Monte Carlo Control

+

Temporal Difference Control

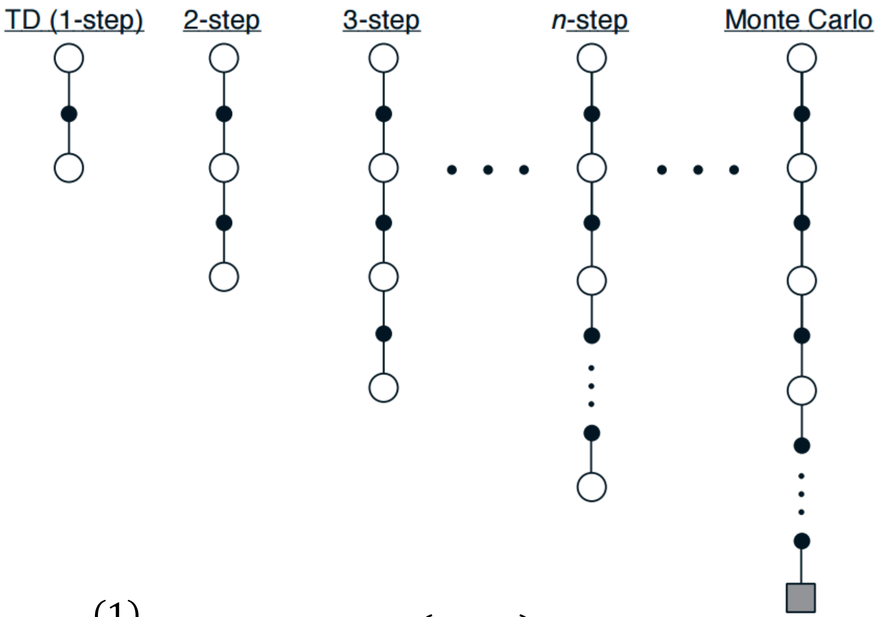
1. One step→ Multiple steps ahead (links to Monte Carlo)

2. Model-free→ Include the model of environment (links to Dynamic programming)

3. Tabular → Continuous space (links to supervised learning)

4. Single agent → Multiple Agents (links to Game theory: Stochastic Game)

n-Steps TD Prediction



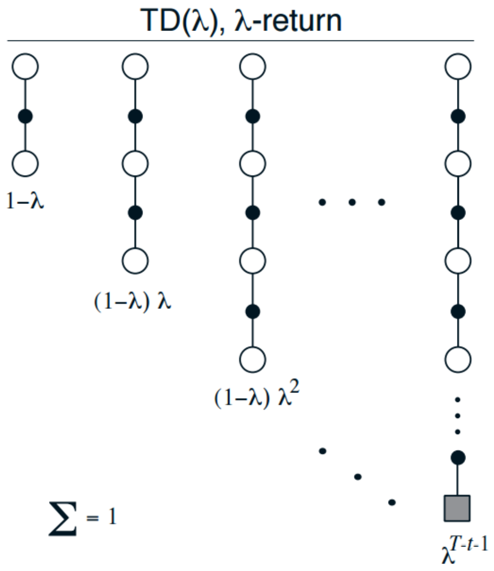
TD method

Monte Carlo method

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$$
$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$
$$R_t^{(3)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V_t(s_{t+3})$$
$$\vdots$$
$$R_t^{(n)} = r_{t+1} + \gamma V_t(s_{t+1}) + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$$
$$\vdots$$
$$R_t = r_{t+1} + \gamma V_t(s_{t+1}) + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$$

$$\Delta V_t(s_t) \leftarrow V_t(s_t) + \alpha \left[R_t^{(n)} - V_t(s_t)\right]$$

TD(λ) Method



$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)} = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)} + \lambda^{T-t-1}R_t$$

$$\lambda = 0,\ R_t^{\lambda=0} = R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$$

TD method

$$\lambda = 1,\ R_t^{\lambda=1} = R_t$$

Monte Carlo method

$$\Delta V_t(s_t) \leftarrow V_t(s_t) + \alpha\left[R_t^\lambda - V_t(s_t)\right]$$

1. One step→ Multiple steps ahead (links to Monte Carlo)

2. Model-free→ Include the model of environment (links to Dynamic programming)

3. Tabular → Continuous space (links to supervised learning)

4. Single agent → Multiple Agents (links to Game theory: Stochastic Game)

Planning and Learning

A Model:
anything that an agent can use to predict how the environment will respond to its actions

- Distribution models: produce a description of all possibilities and their probabilities
  - ✓ e.g., $T(s, a, s')$
  - ✓ Generate all possible episodes and their probabilities
- Sample models: produce just one of the possibility
  - ✓ e.g., Black jack simulator
  - ✓ Generate a an entire episode

Planning:
Refer to any computational process that takes a model as input and produces or imporves a policy for interacting with the modeled environment

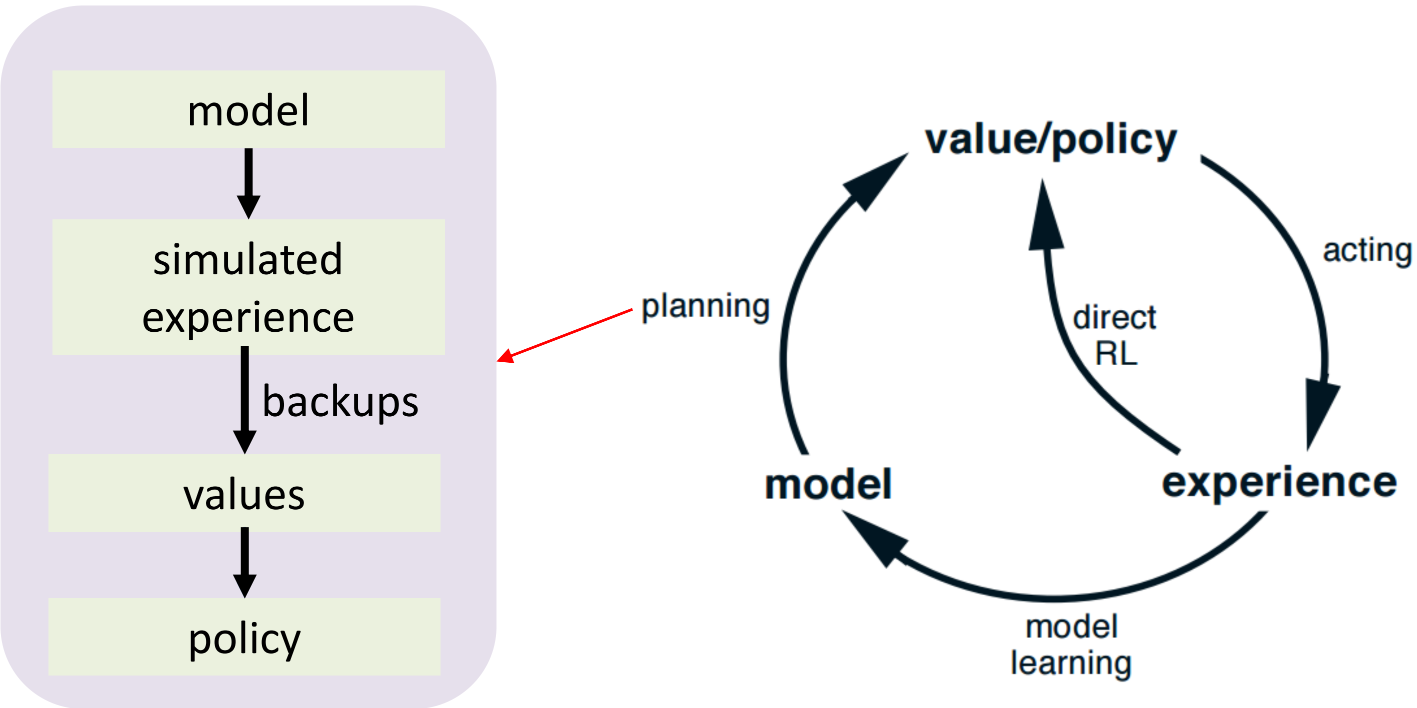
Model $\xrightarrow{\text{Planning}}$ Policy

- State-space planning: search through the state space for an optimal policy
  - ✓ e.g., Q learning
- Plan-space planning : Search through the space of plans
  - ✓ e.g., Route finding, Task ordering

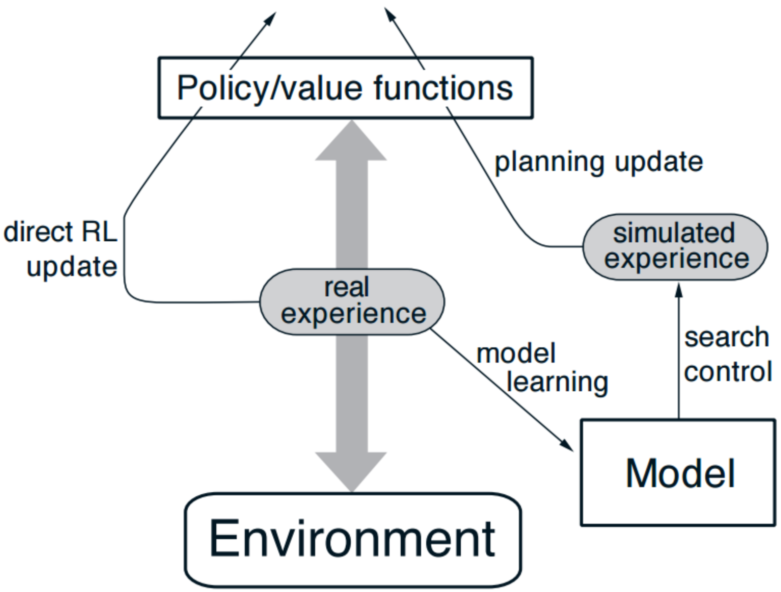
A Planning agent conduct two tasks simultaneously using experience:

- **Model-learning**: use experience to improve the model
  (make it more accurately match the real environment)

- **Direct reinforcement learning** : use experience to directly improve the value function and policy using RL techniques



Dyna-Q algorithm



Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:

(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S, Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:

$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

1. One step→ Multiple steps ahead (links to Monte Carlo)

2. Model-free→ Include the model of environment (links to Dynamic programming)

3. Tabular → Continuous space (links to supervised learning)

4. Single agent → Multiple Agents (links to Game theory: Stochastic Game)

Q-learning with function approximation

Q-learning (stochastic gradient update)

$$Q(s,a) \leftarrow Q(s,a) + \eta\left[\left(r + \gamma \max_{a} Q(s',a)\right) - Q(s,a)\right]$$

Problem:

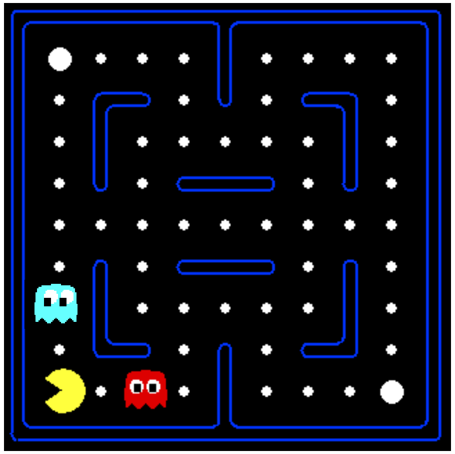
- Basic Q-Learning keeps a table of all q-values
- We cannot possibly learn about every single state!
  Too many states to visit them all in training
  Too many states to hold the q-tables in memory
- Doesn't generalize to unseen states/actions

Solution:

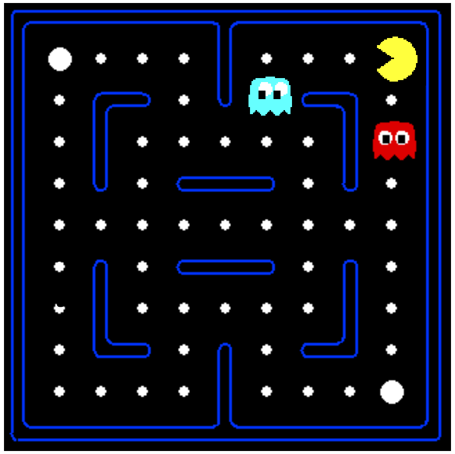
- Learn about some small number of training states from experience
- Generalize that experience to new, similar situations
  - ➢ fundamental idea in machine learning

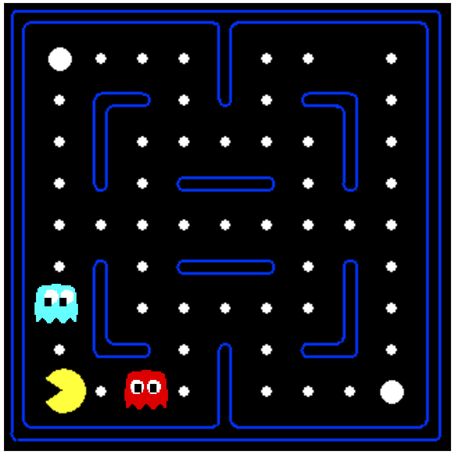
Q-learning with function approximation

Obtained experience:
This state is bad!



Complete new states





- Are these states are good or bad?
- For table representation of $Q(s,a)$, we cannot answer
- We can represent $Q(s,a)$ using the properties of the current state s:

$$Q(s,a;w) = w_1\phi_1(s,a) + w_2\phi_2(s,a) +,\ldots, +w_n\phi_n(s,a)$$
$$= w^T\phi(s,a)$$

$\phi_i(s,a)$: distance to closet ghost, number of ghost, distance to foods

Q-learning with function approximation

Approximate $Q(s,a)$ as a function:

$$Q(s,a;w) = w_1\phi_1(s,a) + w_2\phi_2(s,a) +, \ldots, +w_n\phi_n(s,a)$$
$$= w^T\phi(s,a)$$

w : weight vector$\phi(s,a)$: features vector

- Features, $\phi(s,a) = \{\phi_1(s,a), \ldots, \phi_n(s,a)\}$, are supposed to be properties of the state-action (s,a) pair that are indicative of the quality of taking action a and state s

Algorithm : Q-learning with function approximation

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

$$w \leftarrow w + \eta \left[\underbrace{\left(r + \gamma \max_a \hat{Q}\,(s', a; w) \right)}_{\text{Target}} - \underbrace{\hat{Q}\,(s, a; w)}_{\text{Estimate}} \right] \phi(s, a)$$

This is equivalent to find the weight w that maximizes the following objective function

$$\min_{\hat{Q}_\pi} \frac{1}{2} \sum_{(s,a,r,s')} \left(\underbrace{\left(r + \gamma \max_a \hat{Q}\,(s', a; w) \right)}_{\text{Target}} - \underbrace{\hat{Q}\,(s, a; w)}_{\text{Estimate}} \right)^2$$

For a single transition (s, a, r, s'), the error can be expressed as:

$$\text{Error}(w) = \frac{1}{2} \left(\left(r + \gamma \max_a \hat{Q}\,(s', a; w) \right) - \hat{Q}\,(s, a; w) \right)^2$$

$$= \frac{1}{2} \left(\left(r + \gamma \max_a \hat{Q}\,(s', a; w) \right) - \sum_{k=1}^{n} w_k \phi_k(s, a) \right)^2 \qquad \because \hat{Q}\,(s, a; w) = \sum_{k=1}^{n} w_k \phi_k(s, a)$$

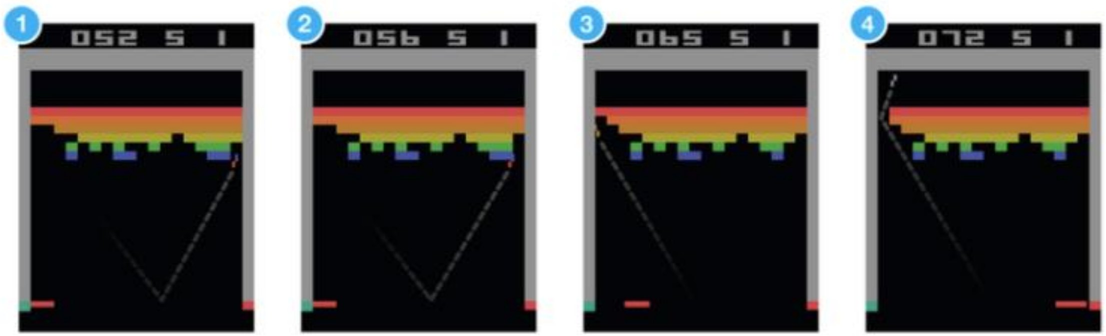
$$\frac{\partial \text{Error}(w)}{\partial w_k} = -\left(\left(r + \gamma \max_a \hat{Q}\,(s', a; w) \right) - \sum_{k=1}^{n} w_k \phi_k(s, a) \right) \phi_k(s, a)$$

$$w_k \leftarrow w_k + \eta \left[\left(r + \gamma \max_a \hat{Q}\,(s', a; w) \right) - \hat{Q}\,(s, a; w) \right] \phi(s, a)$$

Use a neural network for estimating $Q(s, a)$

Playing Atari [Google DeepMind, 2013]:

$$a^* = \pi(s = $$



$$)$$

Figure 1: Atari Breakout game. Image credit: DeepMind.

move left
move right
stroke

- last 4 frames (images) ⇒ 3-layer NN ⇒ keystroke (move left, right, stroke)
- ϵ-greedy, train over 10M frames with 1M replay memory
- Human-level performance on some games (breakout)

https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

Q-Learning with Neural Network

Screen size : 84×84 and convert to grayscale with 256 gray levels

State size = $256^{84\times 84\times 4} \approx 10^{67970}$ more than the number of atoms in the known universe

$$Q(s,a) = \left.\boxed{}\right\} 10^{67970}$$

→ Need to represent this large Q table using a function approximation (Deep learning)
→ Need to estimate Q-values for states that have never been seen before (generalization

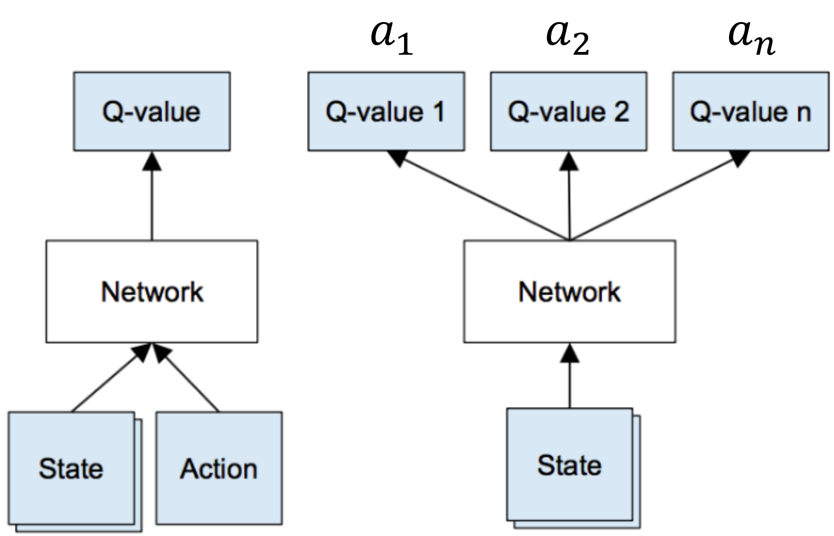


Figure 3: *Left:* Naive formulation of deep Q-network. *Right:* More optimized architecture of deep Q-network, used in DeepMind paper.

Q-Learning with Neural Network

$$L = \frac{1}{2}[\underbrace{r + max_{a'}Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Given a transition < *s*, *a*, *r*, *s'* >, the Q-table update rule in the previous algorithm must be replaced with the following:

1. Do a feedforward pass for the current state *s* to get predicted Q-values for all actions.
2. Do a feedforward pass for the next state *s'* and calculate maximum overall network outputs $max_{a'} Q(s', a')$.
3. Set Q-value target for action to $r + \gamma max_{a'} Q(s', a')$ (use the max calculated in step 2). For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.
4. Update the weights using backpropagation.

Deep Q-learning Algorithm

This gives us the final deep Q-learning algorithm with experience replay:

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
     select an action a
          with probability ε select a random action
          otherwise select a = argmax_a′ Q(s,a′)
     carry out action a
     observe reward r and new state s′
     store experience <s, a, r, s′> in replay memory D

     sample random transitions <ss, aa, rr, ss′> from replay memory D
     calculate target for each minibatch transition
          if ss′ is terminal state then tt = rr
          otherwise tt = rr + γmax_a′ Q(ss′, aa′)
     train the Q network using (tt - Q(ss, aa))^2 as loss

     s = s'
until terminated
```

1. One step → Multiple steps ahead (links to Monte Carlo)

2. Model-free → Include the model of environment (links to Dynamic programming)

3. Tabular → Continuous space (links to supervised learning)

4. Single agent → Multiple Agents (links to Game theory: Stochastic Game)

Prisoner's Dilemma

		Player 2	
		C	D
Player 1	C	(-1, -1)	(-4,0)
	D	(0,-4)	(-3,3)

- Nash Equilibrium Solution Concept

Repeated Game

What happens when a simple normal-from game is repeated infinitely?

	C	D
C	(-1, -1)	(-4,0)
D	(0,-4)	(-3,3)

	C	D
C	(-1, -1)	(-4,0)
D	(0,-4)	(-3,3)

	C	D
C	(-1, -1)	(-4,0)
D	(0,-4)	(-3,3)

…

	C	D
C	(-1, -1)	(-4,0)
D	(0,-4)	(-3,3)

- Assume each player does not know the action chosen by other player in the current game, but becomes to know after the single state game is over

- What kind of decision making strategies an agent need to use to maximize the payoff

  ✓ Tit-for-Tat: a strategy in witch the players starts by cooperating and thereafter chooses in round $j+1$ the action chosen by the other player in round j

  ✓ Triger strategy: Tit-for-Tat→ defect forever once the opponent defects

Stochastic Games

With Models	Stateless	State
Single Agent	Optimization	Markov Decision Process
Multiple Agents	Repeated Game	Stochastic Game

Data-driven approaches

NDP:RL::Stochastic Game: Multi-Agent RL

Without Models	Stateless	State
Single Agent	Model-free Optimization	Reinforcement Learning
Multiple Agents	Learning in Repeated Game	Multi-Agents Reinforcement Learning

A stochastic game is a generalization of repeated games

- agents repeatedly play games from a set of normal-form games
- the game played at any iteration depends on the previous game played and on the actions taken by all agents in that game

A stochastic game is a generalized Markov decision process

- there are multiple players
- one reward function for each agent
- the state transition function and reward functions depend on the action choices of all of the players

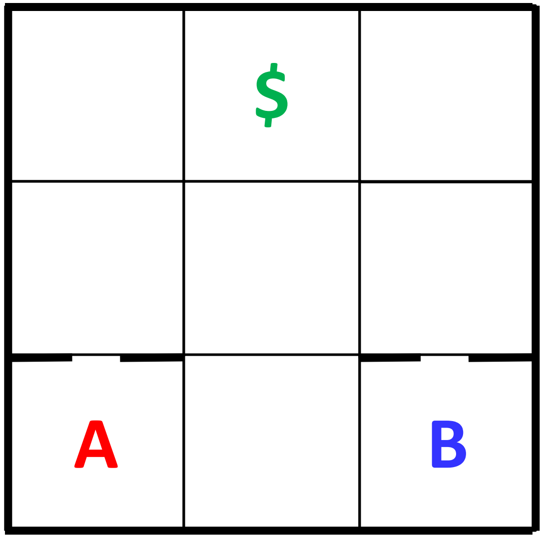
Stochastic Games

A **stochastic game** is a tuple (S, N, A, T, R), where

- S is a finite set of states
- N is a finite set of n players
- $A = A_1 \times \cdots \times A_n$, where A_1 is a finite set of actions available to player i
- $T: S \times A \times S \mapsto [0,1]$ is the transition probability function;
  - ✓ $T(s, a, s')$ is the probability of transitioning from state s to state s' after joint action a
- $R = r_1, \ldots, r_n$, where $r_i: S \times A \mapsto \mathbb{R}$ is a real-valued payoff function for player i
  - ✓ $R_i(s, a)$ is the reward function rewarded by taking a joint action a given state s

- This assumes strategy space is the same in all games otherwise just more notation
- Again we can have average or discounted payoffs.
- Interesting special cases:
  - zero-sum stochastic game
  - single-controller stochastic game transitions (but not payoffs) depend on only one agent

Stochastic (Markov) Games



- First to reach goal gets $100
- If both reaches the money at the same time, both win
- Semi wall (50% go through)
- Cannot occupy the same grid
- Coin flip if collide

2 players stochastic game

- S (States) : $s \in S$
- A_i (Actions for player i) : $a_1 \in A_1, a_2 \in A_2$
- T (Transitions) : $T(s, (a_1, a_2), s')$
- R_i (Reward s for player i) : $R_1(s, (a_1, a_2)),\ R_2(s, (a_1, a_2))$
- γ (Discount factor)

Narrowing down the definition

- $R_1 = -R_2$:
- $T(s, (a_1, a_2), s') = T(s, (a_1, a'_2), s')$ for any a'_2
  $R(s, (a_1, a_2)) = T(s, (a_1, a'_2))$ for any a'_2
- $|S| = 1$

- Zero-sum Stochastic Game
- MDP
- Repeated Game

2 players stochastic game

- Zero-sum Stochastic Game

MDP← Q-learning

Can we employ Q-learning approach to Stochastic Game?

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right\}$$

$$Q_i^*\big(s,(a_1,a_2)\big) = \sum_{s'} T(s,(a_1,a_2),s')\left\{R_i(s,(a_1,a_2),s') + \gamma \max_{(a'_1,a'_2)} Q_i^*\big(s',(a'_1,a'_2)\big)\right\}$$

- **Is this correct?**

- **This is not correct**

because it assumes that joint actions will benefit the agent i the most.

2 players stochastic game

- Zero-sum Stochastic Game

MDP ← Q-learning

Can we employ Q-learning approach to Stochastic Game?

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right\}$$

$$Q_i^*\big(s,(a_1,a_2)\big) = \sum_{s'} T(s,(a_1,a_2),s')\left\{R_i(s,(a_1,a_2),s') + \gamma \max_{(a'_1,a'_2)} Q_i^*\big(s',(a'_1,a'_2)\big)\right\}$$

$$Q_i^*\big(s,(a_1,a_2)\big) = \sum_{s'} T(s,(a_1,a_2),s')\left\{R_i(s,(a_1,a_2),s') + \gamma \operatorname{minmax}_{(a'_1,a'_2)} Q_i^*\big(s',(a'_1,a'_2)\big)\right\}$$

2 players stochastic game

- Zero-sum Stochastic Game

$$Q_i^*(s,(a_1,a_2)) = \sum_{s'} T(s,(a_1,a_2),s')\left\{R_i(s,(a_1,a_2),s') + \gamma \operatorname*{minmax}_{(a'_1,a'_2)} Q_i^*(s',(a'_1,a'_2))\right\}$$

$$Q_i^*(s,(a_1,a_2)) \leftarrow r_i + \gamma \operatorname*{minmax}_{(a'_1,a'_2)} Q_i^*(s',(a'_1,a'_2))$$

e.g., for player 1

$$Q_1^*(s,(a_1,a_2)) \leftarrow r_1 + \gamma \min_{a'_2} \max_{a'_1} Q_2^*(s',(a'_1,a'_2))$$

Comments:

- Value iteration works
- $\operatorname*{minmax}_{(a'_1,a'_2)} Q_i^*(s',(a'_1,a'_2))$ converges
- Unique solution to Q_i^*
- Policy can be computed independently
- Q_i^* is sufficient to optimally behave

2 players stochastic game

- **General**-sum Stochastic Game

$$Q_i^*\big(s,(a_1,a_2)\big) = \sum_{s'} T(s,(a_1,a_2),s')\left\{R_i(s,(a_1,a_2),s') + \gamma \underset{(a'_1,a'_2)}{\operatorname{minmax}} Q_i^*\big(s',(a'_1,a'_2)\big)\right\}$$

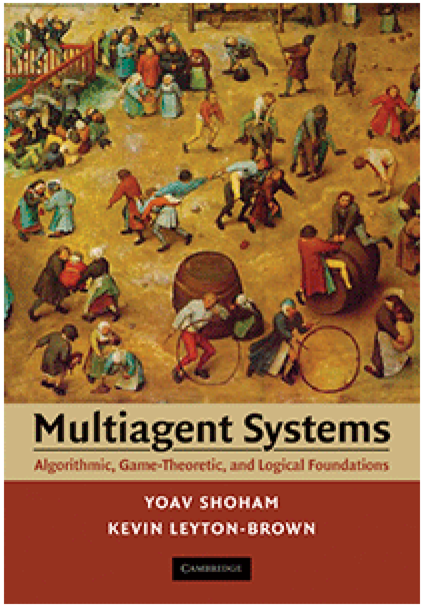
$$Q_i^*\big(s,(a_1,a_2)\big) = \sum_{s'} T(s,(a_1,a_2),s')\left\{R_i(s,(a_1,a_2),s') + \gamma \underset{(a'_1,a'_2)}{\operatorname{Nash}} Q_i^*\big(s',(a'_1,a'_2)\big)\right\}$$

$$Q_i^*\big(s,(a_1,a_2)\big) \leftarrow r_i + \gamma \underset{(a'_1,a'_2)}{\operatorname{Nash}} Q_i^*\big(s',(a'_1,a'_2)\big)$$

Comments:

- Value iteration **doesn't** works
- $\underset{(a'_1,a'_2)}{\operatorname{minmax}} Q_i^*\big(s',(a'_1,a'_2)\big)$ **doesn't** converges
- **No** Unique solution to Q_i^*
- Policy can **not** be computed independently
- Q_i^* is **not** sufficient to optimally behave

2017 Spring



Multiagent Systems

Algorithmic, Game-Theoretic, and Logical Foundations

Yoav Shoham
Stanford University
Kevin Leyton-Brown
University of British Columbia

Cambridge University Press, 2009
Order online: amazon.com