Markov games as a framework for multi-agent reinforcement learning

Yongnan Ji

- Definitions
- Optimal Policies
- Finding Optimal Policies
- Learning Optimal Policies
- An Example

Definition

Markov decision process (MDP)

States S, Actions A

Transtion Action function $T : S \times A \rightarrow \mathrm{PD}(S)$

Reward Funtion $R : S \times A \rightarrow \mathbf{R}$

Agent's objective: Maximize $E\{\sum_{j=0}^{\infty} \gamma^{j} r_{t+j}\}$

Discount factor

Markov Game (stochastic game)

States S, Actions $A_1, A_2, \ldots, A_k$,

Transtion Action function $T : S \times A_1 \times A_2 \ldots \times A_k \to \mathrm{PD}(S)$

Reward Funtion $R_i : S \times A_1 \times A_2 \ldots \times A_k \to \mathrm{R}$

Agent’s objective: Maximize $E\{\sum_{j=0}^{\infty} \gamma^j r_{i,t+j}\}$

When there are only two agents

Agent

Action set S

Opponent

Action set O

Reward $R(s,a,o)$

Optimal Policies

Optimal Policies: the one that maximizes the expected sum of discounted reward and is undominated

Undominated: that there is no state from which any other policy can achieve a better expected sum of discounted reward.

Every MDP has at least one optimal policy and of the optimal policies for a given MDP, at least one is *stationary* and *deterministic*.

Finding Optimal Policies

Matrix Games

		Agent		
		rock	paper	scissors
Opponent	rock	0	1	-1
	paper	-1	0	1
	scissors	1	-1	0

$$
\begin{array}{rlrlrlll}
 & & & \pi_{\text{paper}} & - & \pi_{\text{scissors}} & \geq V & \textit{(vs. rock)} \\
- & \pi_{\text{rock}} & & & + & \pi_{\text{scissors}} & \geq V & \textit{(vs. paper)} \\
 & \pi_{\text{rock}} & - & \pi_{\text{paper}} & & & \geq V & \textit{(vs. scissors)} \\
 & \pi_{\text{rock}} & + & \pi_{\text{paper}} & + & \pi_{\text{scissors}} & = 1 &
\end{array}
$$

$$V = \max_{\pi \in \mathrm{PD}(A)} \min_{o \in O} \sum_{a \in A} R_{o,a} \pi_a ,$$

MDP's

value iteration

value of a state $V(s)$

quality of a state-action pair $Q(s,a)$

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$$

$$V(s) = \max_{a' \in A} Q(s,a')$$

Markov Games

$$Q(s,a,o) = R(s,a,o) + \gamma \sum_{s'} T(s,a,o,s')V(s')$$

$$V(s) = \max_{\pi \in \mathrm{PD}(A)} \min_{o \in O} \sum_{a \in A} Q(s,a,o)\pi_a,$$

Learning Optimal Policies

Value iteration is traditionally used

In this Q-learning formulation, the updates are synchronously performed without the use of the transition function, T.

$$Q(s, a) := r + \gamma V(s')$$

performed whenever it receives a reward of r when making a transition from s to s' after taking action a.

every action is tried in every state infinitely often $\longrightarrow$ $T(s, a, s')$

Initialize:

For all `s` in `S`, `a` in `A`, and `o` in `O`,
  Let `Q[s,a,o] := 1`
For all `s` in `S`,
  Let `V[s] := 1`
For all `s` in `S`, `a` in `A`,
  Let `pi[s,a] := 1/|A|`
Let `alpha := 1.0`

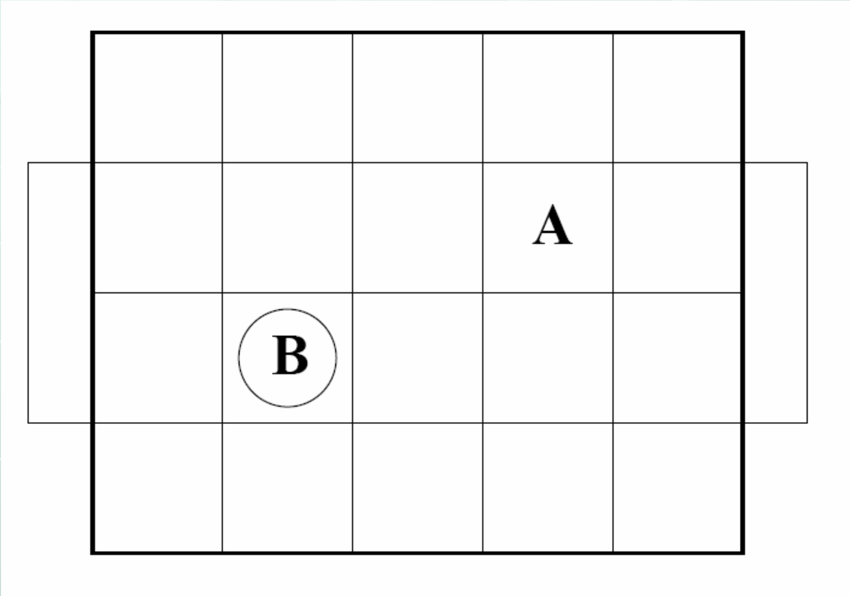
Choose an action:
With probability `explor`, return an action uniformly at random. Otherwise, if current state is `s`,   Return action `a` with probability `pi[s,a]`.

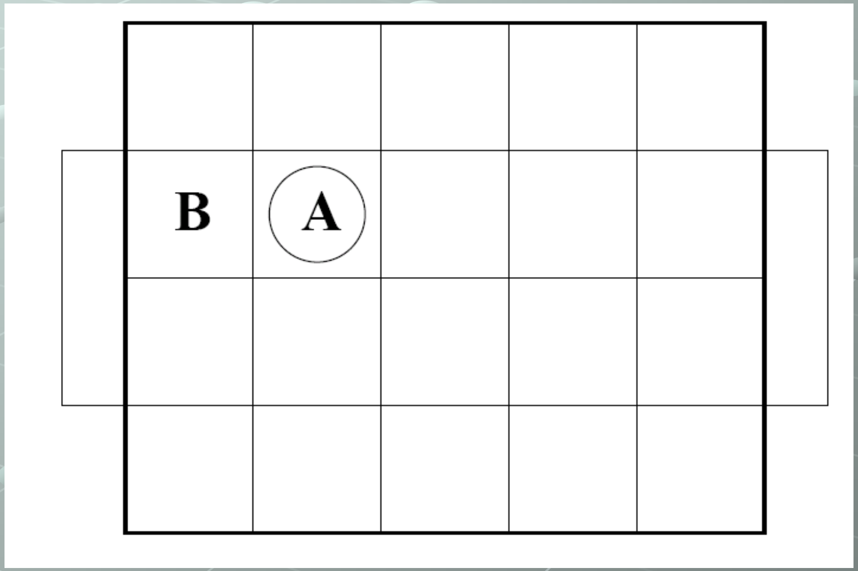
Learn:

After receiving reward `rew` for moving from state `s` to `s'`
    via action `a` and opponent's action `o`,
  Let `Q[s,a,o] := (1-alpha) * Q[s,a,o] + alpha * (rew + gamma * V[s'])`
  Use linear programming to find `pi[s,.]` such that:
    `pi[s,.] := argmax{pi'[s,.], min{o', sum{a', pi[s,a'] * Q[s,a',o']}}}`
  Let `V[s] := min{o', sum{a', pi[s,a'] * Q[s,a',o']}}`
  Let `alpha := alpha * decay`

Figure 1: The minimax-Q algorithm.

Example





Discount factor 0.9

Results

	MR		MM		QR		QQ	
	% won	games	% won	games	% won	games	% won	games
vs. random	99.3	6500	99.3	7200	99.4	11300	99.5	8600
vs. hand-built	48.1	4300	53.7	5300	26.1	14300	76.3	3300
vs. MR-challenger	35.0	4300						
vs. MM-challenger			37.5	4400				
vs. QR-challenger					0.0	5500		
vs. QQ-challenger							0.0	1200

Conclusion

- linear programming is somewhat problematic
- the minimax operator can be implemented extremely efficiently in some games
- Applying of cooperative and multi-player games could also prove fruitful