Markov Games

Zero-sum Multi-Agent Reinforcement Learning

How to learn optimal agent behaviour in the presence of an opponent (two-player zero-sum game).

Based on the paper

Michael L. Littman (1994), Markov Games as a Framework for Multi-Agent Reinforcement Learning, In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 157–163, Morgan Kaufmann, San Francisco, California.

Markov Models of Stochastic Environments

- Markov decision processes (MDPs)
- Markov games
- Partially observable MDPs (POMDPs)
- Incomplete-information games

	Single agent	multiple agents
State known	MDP	MG
State observed indirectly	POMDP	I-I G

Markov Decision Process (MDP)

$$\text{An MDP is a tuple } \langle S, A, r, p \rangle: \quad \begin{cases} S = \text{set of states} \\ A = \text{set of actions} \\ r : S \times A \to \mathbb{R} \quad \text{(reward function)} \\ p : S \times A \to PD(S) \quad \text{(transition function)} \end{cases}$$

Objective: Find a strategy (policy) π which maximizes the expected sum of discounted rewards,

$$v(s, \pi) = r(s, a_\pi) + \beta \sum_{s'} p(s' | s, a_\pi) v(s', \pi)$$

Theorem: There exists an optimal policy π^* such that for any $s \in S$

$$v(s, \pi^*) = \max_a \left\{ r(s, a) + \beta \sum_{s'} p(s' | s, a) v(s', \pi^*) \right\}$$

If r and p are known, π^* can be found through iterative search methods (e.g., value iteration).

If r and p are unknown, the agent needs to interact with the environment to find its optimal policy through *reinforcement learning*.

Reinforcement Learning

Model-Based or Model-Free

Model-based reinforcement learning: The agent can collect statistics on experience tuples $\langle s, a, r, s' \rangle$ and update counts and rewards

$$n(s, a) \leftarrow n(s, a) + 1, \qquad n(s, a, s') \leftarrow n(s, a, s') + 1, \qquad \rho(s, a) \leftarrow \rho(s, a) + r$$

and compute estimates

$$\hat{p}(s' | s, a) = \frac{n(s, a, s')}{n(s, a)} \quad \text{and} \quad \hat{r}(s, a) = \frac{\rho(s, a)}{n(s, a)}.$$

Then, given this estimated model, an (approximate) optimal policy can be determined.

Model-free reinforcement learning: For large MDPs, the model-based approach can be rather space demanding, calling for model-free approaches (e.g., *Q-learning*).

Q-learning

The theorem said that there exists an optimal policy π^* such that for any $s \in S$

$$v(s, \pi^*) = \max_a \left\{ r(s,a) + \beta \sum_{s'} p(s'|s,a) v(s', \pi^*) \right\}.$$

Now, define for state-action pair (s,a) the *quality function*

$$Q^*(s,a) = r(s,a) + \beta \sum_{s'} p(s'|s,a) v(s', \pi^*)$$

i.e., the total discounted reward attained by taking action a in state s and then following the optimal policy thereafter. Without access to r and p, only the Q function can be used to reconstruct v and π, as

$$v(s) = \max_a Q(s,a) \quad \text{and} \quad \pi(s) = \arg\max_a Q(s,a).$$

The Q function can be estimated from experience tuples $\langle s, a, r, s' \rangle$. Given such a tuple, the Q-learning rule is

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \left(r + \beta \max_{a'} Q(s', a') \right),$$

where the learning rate α must be slowly decreased for Q to converge to Q^*.

Markov Game (MG)

An MG is a tuple $\langle S, \{A_1, \ldots, A_k\}, \{r_1, \ldots, r_k\}, p \rangle$: $\begin{cases} S = \text{set of states} \\ A_i = \text{set of actions for agent } i \\ r_i : S \times A_1 \times \cdots \times A_k \to \mathbb{R} \ \ \text{(reward for agent } i) \\ p : S \times A_1 \times \cdots \times A_k \to PD(S) \ \ \text{(transition func.)} \end{cases}$

We consider the case of two players (*agent* and *opponent*) with diametrically opposed goals. That is,

$$A_1 = A, \quad A_2 = O, \quad r_1 = -r_2 \ \ \text{(zero-sum)}$$

such that we have a single reward function, $r(s,a,o)$, that our agent tries to maximize while the opponent tries to minimize it.

For every MDP, at least one optimal policy, π, is *stationary* (i.e., π fixed over time) and *deterministic* (i.e., same action each time).

For every MG, at least one optimal policy is stationary, but there need not be a deterministic optimal policy. Instead, the optimal stationary policy is sometimes *probabilistic*.

Optimal Game Policies

Examples

Game	Stationary	Deterministic
Tic-tac-toe	yes	yes
Backgammon	yes	yes
Rock, paper, scissors	yes	no

Any deterministic policy can be consistently defeated in the game of "rock, paper, scissors".

The need for probabilistic action choice is due to the agent's uncertainty of its opponents current move and its requirement to avoid being "second guessed".

The optimal policy in the game of "rock, paper, scissors" is a probability distribution over actions, $\pi \in PD(A)$:

$$\pi_{\text{rock}} + \pi_{\text{paper}} + \pi_{\text{scissors}} = 1.$$

Minimax: Maximize Reward in Worst Case

In an MDP, an optimal policy is *undominated*: There is no state from which another policy performs better.

For many Markov games, there is no policy that is undominated because performance depends critically on the choice of the opponent.

Minimax: Eliminate the choice of the opponent and maximize the reward in the worst case (i.e., the opponent choice that makes it look the worst).

Example (rock, paper, scissors): The minimal expected reward should be as large as possible.

	Agent's choice			Expected rewards			
Opponent's choice	rock	paper	scissors	rock	paper	scissors	Constraints
rock	0	1	−1		π_{paper}	$-\ \pi_{\text{scissors}}$	$\geq V$
paper	−1	0	1	$-\ \pi_{\text{rock}}$		$+\ \pi_{\text{scissors}}$	$\geq V$
scissors	1	−1	0	π_{rock}	$-\ \pi_{\text{paper}}$		$\geq V$

Solution: Identify largest V less than the minimal expected reward. That is, identify the largest V such that the constraints hold (including $\pi_{\text{rock}} + \pi_{\text{paper}} + \pi_{\text{scissors}} = 1$).

Minimax

Rock, Paper, Scissors

We have the following constraints:

$$\begin{array}{rcl}
0 \cdot \pi_{\text{rock}} + 1 \cdot \pi_{\text{paper}} + -1 \cdot \pi_{\text{scissors}} & \geq & V \\
-1 \cdot \pi_{\text{rock}} + 0 \cdot \pi_{\text{paper}} + 1 \cdot \pi_{\text{scissors}} & \geq & V \\
1 \cdot \pi_{\text{rock}} + -1 \cdot \pi_{\text{paper}} + 0 \cdot \pi_{\text{scissors}} & \geq & V
\end{array}$$

Linear programming is a general technique for solving problems of this kind. In this example, we find $V = 0$ and $\pi_{\text{rock}} = \pi_{\text{paper}} = \pi_{\text{scissors}} = 1/3$.

We can abbreviate this linear program as

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

where $\sum_a r(o, a)\pi_a$ is the expected reward for using policy π against the opponent's action o.

Value Iteration

In an MDP, we have the following relationship between the Q function (expressing the *quality* of the state-action pair (s, a)) and the value function:

$$Q(s, a) = r(s, a) + \beta \sum_{s' \in S} p(s'|s, a)v(s') \tag{1}$$

$$v(s) = \max_{a' \in A} Q(s, a') \tag{2}$$

The optimal policy in MDPs can be found through value iteration which starts with estimates for Q and v and generates new estimates by treating the equal signs in (1) and (2) as assignment operators.

Using the minimax approach, a similar formulation can be given for Markov games:

$$Q(s, a, o) = r(s, a, o) + \beta \sum_{s' \in S} p(s'|s, a, o)v(s') \tag{3}$$

$$v(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} Q(s, a, o)\pi_a, \tag{4}$$

where $Q(s, a, o)$ expresses the expected reward for taking action a when the opponent chooses o from state s, and $v(s)$ is the expected reward for the optimal policy starting from state s.

Value Iteration in Markov Games

It has been shown that a value iteration algorithm based on (3) and (4) converges to the correct values.

$$Q(s, a, o) = r(s, a, o) + \beta \sum_{s' \in S} p(s'|s, a, o)v(s') \tag{3}$$

$$v(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} Q(s, a, o)\pi_a. \tag{4}$$

Note, that $v(s)$ needs to be computed through linear programming.

However, in games with alternating turns, the value function need not be computed by linear programming since there is an optimal deterministic policy. In this case we can write

$$v(s) = \max_a \min_o Q(s, a, o).$$

Relation Between Value Iteration and Q-learning

In value iteration (for MDPs) the Q function is updated through (1):

$$Q(s, a) = r(s, a) + \beta \sum_{s' \in S} p(s'|s, a)v(s'). \tag{1}$$

In Q-learning, the Q function is updated through

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r(s, a) + \beta \max_{a'} Q(s', a') \right).$$

Since $v(s') = \max_{a'} Q(s', a')$ and given a learning rate of $\alpha = 1$ this update formula becomes

$$Q(s, a) \leftarrow r(s, a) + \beta\, v(s').$$

The probability with which this update happens is precisely $p(s'|s, a)$, and hence it is possible for the agent to avoid explicitly using p.

The Minimax Q-learning Algorithm

For each experience tuple $\langle s, a, o, r, s' \rangle$

$$Q(s,a,o) \leftarrow (1-\alpha)Q(s,a,o) + \alpha\left(r(s,a,o) + \beta v(s')\right)$$

Find $\pi(s,\cdot)$ such that

$$\pi(s,\cdot) \leftarrow \arg\max_{\pi'(s,\cdot)} \min_{o'} \sum_{a'} \pi'(s,a')Q(s,a',o')$$

$$v(s) \leftarrow \min_{o'} \sum_{a'} \pi(s,a')Q(s,a',o')$$

$$\alpha \leftarrow \gamma \cdot \alpha$$

The decay factor $\gamma \in [0;1[$ makes sure that the learning rate, α, is slowly decreased.

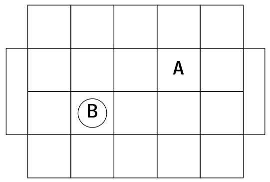
Actions are selected according to the following scheme:

With probability explor, return an action uniformly at random;
otherwise, return action a with probability $\pi(s,a)$.

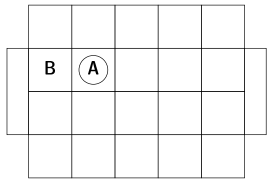
Experiments: Soccer

The following rules governs the game:

- Each player can choose one of 5 actions: *north, south, east, west*, and *stand*.
- Once both players have selected their actions, the two moves are executed in random order.
- Possession of the ball goes to stationary player and the move is cancelled if a player tries a move which brings it to the square occupied by the other player.
- A goal is scored when the player with the ball steps into the appropriate goal.



An initial board



Requires probabilistic choice for A

Experiment Set-Up

- Four different policies learned, using minimax-Q and Q-learning (“max” instead of “minimax” and no information kept about opponent actions):
  - MR: minimax-Q trained against random
  - MM: minimax-Q trained against minimax-Q
  - QR: Q trained against random
  - QQ: Q trained against Q
- Discount factor $\beta = 0.9$ (encourages scoring sooner rather than later).
- Learning from 10^6 experience tuples.
- In minimax-Q: $\mathsf{explor} = 0.2$ and decay factor $\gamma = 10^{\log 0.01/10^6} = 0.9999954$ ($\alpha = 0.01$ at end of run).
- Each of the resulting policies were evaluated in three ways:
  - Head-to-head with a random policy for 10^5 steps
  - Head-to-head with a deterministic hand-built policy for 10^5 steps
  - Head-to-head with a challenger, trained against the respective champion, for 10^5 steps

To emulate the discount factor, every step has a 0.1 probability of being declared a draw.

In 10^5 steps, the hand-built policy completed 5600 games against the random opponent and won 99.5% of them.

Results – I

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

- Vs. random: QR performs (expectably) quite remarkable, since it was trained specifically to beat this opponent, whereas MR chooses actions with an idealized opponent in mind.
- Vs. hand-built: MM did marginally better than MR. However, in the limit, this should not be the case since an agent trained by minimax-Q should be insensitive to the opponents against which it was trained.
- The success of QQ against the hand-built agent was pure luck!

Results – II

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

- Vs. challengers: Minimax policies are designed to do good against any opponent. So we should expect MR and MM to break even against their challengers. Lack of convergence explains why this is not quite observed.
- The Q-learned agents did really bad against their challengers, as Q-learning is designed to find deterministic policies and every deterministic offense has a perfect defense in this game.
- Extending Q-learning to find optimal probabilistic policies is exactly what minimax-Q is designed for.