Reinforcement Learning

Multi-agent Reinforcement Learning

Subramanian Ramamoorthy
School of Informatics

28 March, 2017
Agents often face \textit{Strategic} Adversaries

Key issue we seek to model: Misaligned/conflicting interest
On Self-Interest

What does it mean to say that agents are self-interested?

• It does not necessarily mean that they want to cause harm to each other, or even that they care only about themselves.

• Instead, it means that each agent has his own description of which states of the world he likes—which can include good things happening to other agents

—and that he acts in an attempt to bring about these states of the world (better term: inter-dependent decision making)
A Simple Model of a Game

• Two decision makers
  – Robot (has an action space: $a$)
  – Adversary (has an action space: $\theta$)

• Cost or payoff (to use the term common in game theory) depends on actions of both decision makers:
  $R(a, \theta)$ – denote as a matrix corresponding to product space

$$
\begin{array}{ccc}
\Theta \\
1 & -1 & 0 \\
-1 & 2 & -2 \\
2 & -1 & 1 \\
\end{array}
$$

This is the normal form – simultaneous choice over moves
Representing Payoffs

In a general, bi-matrix, normal form game: $(n, A_1...n, R_1...n)$

Action sets of players
Payoff function: $A \rightarrow \mathbb{R}$

The combined actions $(a_1, a_2, ..., a_n)$ form an action profile $a \in A$
Example: Rock-Paper-Scissors

- Famous children’s game
- Two players; Each player simultaneously picks an action which is evaluated as follows,
  - Rock beats Scissors
  - Scissors beats Paper
  - Paper beats Rock

\[
\begin{pmatrix}
R & P & S \\
R & 0 & -1 & 1 \\
P & 1 & 0 & -1 \\
S & -1 & 1 & 0 \\
\end{pmatrix} \quad \begin{pmatrix}
R & P & S \\
R & 0 & 1 & -1 \\
P & -1 & 0 & 1 \\
S & 1 & -1 & 0 \\
\end{pmatrix}
\]
TCP Game

• Imagine there are only two internet users: you and me
• Internet traffic is governed by TCP protocol, one feature of which is the *backoff* mechanism: when network is congested then backoff and reduce transmission rates for a while
• Imagine that there are two implementations: C (correct, does what is intended) and D (defective)
• If you both adopt C, packet delay is 1 ms; if you both adopt D, packet delay is 3 ms
• If one adopts C but other adopts D then D user gets no delay and C user suffers 4 ms delay
TCP Game in Normal Form

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>-1, -1</td>
<td>-4, 0</td>
</tr>
<tr>
<td>D</td>
<td>0, -4</td>
<td>-3, -3</td>
</tr>
</tbody>
</table>

*Note* that this is another way of writing a bi-matrix game: First number represents payoff of row player and second number is payoff for column player.
Some Famous Matrix Examples
- What are they Capturing?

• Prisoner’s Dilemma: Cooperate or Defect (same as TCP game)

\[ R_1 = \begin{pmatrix} C & D \\ C & D \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \quad R_2 = \begin{pmatrix} C & D \\ C & D \end{pmatrix} \begin{pmatrix} 3 & 4 \\ 0 & 1 \end{pmatrix} \]

• Bach or Stravinsky (von Neumann called it Battle of the Sexes)

\[ R_1 = \begin{pmatrix} B & S \\ B & S \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \quad R_2 = \begin{pmatrix} B & S \\ B & S \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \]

• Matching Pennies: Try to get the same outcome, Heads/Tails

\[ R_1 = \begin{pmatrix} H & T \\ H & T \end{pmatrix} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \quad R_2 = \begin{pmatrix} H & T \\ H & T \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \]
A common-payoff game is a game in which for all action profiles $a \in A_1 \times \cdots \times A_n$ and any pair of agents $i, j$, it is the case that $u_i(a) = u_j(a)$.
A two-player normal-form game is constant-sum if there exists a constant $c$ such that for each strategy profile $a \in A_1 \times A_2$ it is the case that $u_1(a) + u_2(a) = c$

<table>
<thead>
<tr>
<th></th>
<th>Heads</th>
<th>Tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heads</td>
<td>1, -1</td>
<td>-1, 1</td>
</tr>
<tr>
<td>Tails</td>
<td>-1, 1</td>
<td>1, -1</td>
</tr>
</tbody>
</table>

Pure competition:
One player wants to coordinate
Other player does not!
Defining the “action space”

What can players do?

- **Pure strategies** \((a_i)\): select an action.

- **Mixed strategies** \((\sigma_i)\): select an action according to some probability distribution.
Strategies

Notation.

- $\sigma$ is a joint strategy for all players.

$$R_i(\sigma) = \sum_{a \in A} \sigma(a) R_i(a)$$

- $\sigma_{-i}$ is a joint strategy for all players except $i$.

- $\langle \sigma_i, \sigma_{-i} \rangle$ is the joint strategy where $i$ uses strategy $\sigma_i$ and everyone else $\sigma_{-i}$.
Solution Concepts

Many ways of describing what one ought to do:

- Dominance
- Minimax
- Pareto Efficiency
- Nash Equilibria
- Correlated Equilibria

Remember that in the end game theory aspires to predict behaviour given specification of the game.

*Normatively*, a solution concept is a *rationale* for behaviour.
Concept: Dominance

- An action is **strictly dominated** if another action is always better, i.e.,

\[ \exists a'_i \in A_i \ \forall a_{-i} \in A_{-i} \quad R_i(\langle a'_i, a_{-i} \rangle) > R_i(\langle a_i, a_{-i} \rangle). \]

- Consider prisoner’s dilemma.

\[
R_1 = \begin{pmatrix} C & D \\ C & D \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \quad R_2 = \begin{pmatrix} C & D \\ C & D \end{pmatrix} \begin{pmatrix} 3 & 4 \\ 0 & 1 \end{pmatrix}
\]

- For both players, D dominates C.
Concept: Iterated Dominance

- Actions may be dominated by mixed strategies.

\[
R_1 = \begin{pmatrix}
A & D & E \\
B & 1 & 1 \\
C & 4 & 0 \\
\end{pmatrix}
\quad
R_2 = \begin{pmatrix}
A & D & E \\
B & 4 & 0 \\
C & 1 & 2 \\
\end{pmatrix}
\]

- If strictly dominated actions should not be played.

\[
R_1 = \begin{pmatrix}
A & D & E \\
B & 1 & 1 \\
C & 4 & 0 \\
\end{pmatrix}
\quad
R_2 = \begin{pmatrix}
A & D & E \\
B & 4 & 0 \\
C & 1 & 2 \\
\end{pmatrix}
\]

- This game is said to be dominance solvable.
Concept: Minimax

- Consider matching pennies.

\[
R_1 = \begin{pmatrix}
H & T \\
T & H
\end{pmatrix} \begin{pmatrix}
1 & -1 \\
-1 & 1
\end{pmatrix} \\
R_2 = \begin{pmatrix}
H & T \\
T & H
\end{pmatrix} \begin{pmatrix}
-1 & 1 \\
1 & -1
\end{pmatrix}
\]

- Q: What do we do when the world is out to get us?
  A: Make sure it can’t.

- Play strategy with the best worst-case outcome.

\[
\arg\max_{\sigma_i \in \Delta(A_i)} \min_{a_{-i} \in A_{-i}} R_i(\langle \sigma_i, \sigma_{-i} \rangle)
\]

- Minimax optimal strategy.
Minimax

• Back to matching pennies.

\[ R_1 = \begin{bmatrix} H \\ T \end{bmatrix} \begin{pmatrix} H & T \\ 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} = \sigma_1^* \]

• Consider Bach or Stravinsky.

\[ R_1 = \begin{bmatrix} B \\ S \end{bmatrix} \begin{pmatrix} B & S \\ 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1/3 \\ 2/3 \end{pmatrix} = \sigma_1^* \]

• Minimax optimal guarantees the safety value.

• Minimax optimal never plays dominated strategies.
Computing Minimax: Linear Programming

- Minimax optimal strategies via linear programming.

\[
\arg\max_{\sigma_i \in \Delta(A_i)} \min_{a_i \in A_i} R_i(\langle \sigma_i, \sigma_{-i} \rangle)
\]
Pick-a-Hand

• There are two players: chooser (player I) & hider (player II)

• The hider has two gold coins in his back pocket. At the beginning of a turn, he puts his hands behind his back and either takes out one coin and holds it in his left hand, or takes out both and holds them in his right hand.

• The chooser picks a hand and wins any coins the hider has hidden there.

• She may get nothing (if the hand is empty), or she might win one coin, or two.
Pick-a-Hand, Normal Form:

- Hider could minimize losses by placing 1 coin in left hand, most he can lose is 1
- If chooser can figure out hider’s plan, he will surely lose that 1
- If hider thinks chooser might strategise, he has incentive to play R2, ...
- All hider can guarantee is max loss of 1 coin
- Similarly, chooser might try to maximise gain, picking R
- However, if hider strategizes, chooser ends up with zero
- So, chooser can’t actually guarantee winning anything
Pick-a-Hand, with Mixed Strategies

- Suppose that chooser decides to choose R with probability $p$ and L with probability $1 - p$
- If hider were to play pure strategy R2 his expected loss would be $2p$
- If he were to play L1, expected loss is $1 - p$
- Chooser maximizes her gains by choosing $p$ so as to maximize $\min\{2p, 1 - p\}$

- Thus, by choosing R with probability $1/3$ and L with probability $2/3$, chooser assures expected payoff of $2/3$, regardless of whether hider knows her strategy
Mixed Strategy for the Hider

• Hider will play R2 with some probability q and L1 with probability 1–q
• The payoff for chooser is 2q if she picks R, and 1 – q if she picks L
• If she knows q, she will choose the strategy corresponding to the maximum of the two values.

• If hider knows chooser’s plan, he will choose q = 1/3 to minimize this maximum, guaranteeing that his expected payout is 2/3 (because 2/3 = 2q = 1 – q)
• Chooser can assure expected gain of 2/3, hider can assure an expected loss of no more than 2/3, regardless of what either knows of the other’s strategy.
Safety Value as Incentive

- Clearly, without some extra incentive, it is not in hider’s interest to play *Pick-a-hand* because he can only lose by playing.
- Thus, we can imagine that chooser pays hider to entice him into joining the game.
- 2/3 is the maximum amount that chooser should pay him in order to gain his participation.
Equilibrium as a Saddle Point

The saddle point in Matching Pennies, with and without a plane at $z = 0$. 
Concept: Nash Equilibrium

- What action should we play if there are no dominated actions?

- Optimal action depends on actions of other players.

- A **best response set** is the set of all strategies that are optimal given the strategies of the other players.

  \[
  \text{BR}_i(\sigma_{-i}) = \{\sigma_i \mid \forall \sigma'_i \quad R_i(\langle \sigma_i, \sigma_{-i} \rangle) \geq R_i(\langle \sigma'_i, \sigma_{-i} \rangle)\}
  \]

- A **Nash equilibrium** is a joint strategy, where all players are playing best responses to each other.

  \[
  \forall i \in \{1 \ldots n\} \quad \sigma_i \in \text{BR}_i(\sigma_{-i})
  \]
Nash Equilibrium

- A Nash equilibrium is a joint strategy, where all players are playing best responses to each other.

\[ \forall i \in \{1 \ldots n\} \quad \sigma_i \in BR_i(\sigma_{-i}) \]

- Since each player is playing a best response, no player can gain by unilaterally deviating.

- Dominance solvable games have obvious equilibria.
  - Strictly dominated actions are never best responses.
  - Prisoner’s dilemma has a single Nash equilibrium.
Nash Equilibrium - Example

- Consider the coordination game.

\[
R_1 = \begin{pmatrix}
A & B \\
2 & 0 \\
0 & 1 \\
\end{pmatrix} \quad R_2 = \begin{pmatrix}
A & B \\
2 & 0 \\
0 & 1 \\
\end{pmatrix}
\]

- Consider Bach or Stravinsky.

\[
R_1 = \begin{pmatrix}
B & S \\
2 & 0 \\
0 & 1 \\
\end{pmatrix} \quad R_2 = \begin{pmatrix}
B & S \\
1 & 0 \\
0 & 2 \\
\end{pmatrix}
\]
Nash Equilibrium - Example

- Consider matching pennies.

\[ R_1 = \begin{pmatrix} H & T \\ T & 1 \\ 1 & -1 \end{pmatrix} \quad R_2 = \begin{pmatrix} H & T \\ T & -1 \\ 1 & 1 \end{pmatrix} \]

- No pure strategy Nash equilibria. Mixed strategies?

\[ BR_1 \left( \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} \right) = \{ \sigma_1 \} \]

- Corresponds to the minimax strategy.
Many well known techniques from reinforcement learning, e.g., value/policy iteration can still be applied to solving these games.
Stochastic Games (SG)

Defined by the tuple \((n, S, A_1, \ldots, n, T, R_1, \ldots, n)\)

- No. agents
- Set of states
- Set of actions available to each agent

\[ A = A_1 \times A_2 \times \ldots \times A_n \]

Transition dynamics
\[ S \times A \times S \rightarrow [0, 1] \]

Reward function of \(i^{th}\) agent
\[ S \times A \rightarrow R \]
\[ R = R_1 \times R_2 \times \ldots \times R_n \]

We wish to learn a stationary, possibly stochastic, policy:
\[ \rho : S \rightarrow Pr(A_i) \]

Objective continues to be maximization of expected future reward
A First Algorithm for SG Solution [Shapley]

1. Initialize $V$ arbitrarily.

2. Repeat,
   (a) For each state, $s \in S$, compute the matrix,

   \[
   G_s(V) = \left[ g_{a \in A} : R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s') \right].
   \]

   (b) For each state, $s \in S$, update $V$,

   \[
   V(s) \leftarrow \text{Value} \left[ G_s(V) \right].
   \]

This classic algorithm (from 1953) is akin to Value Iteration for MDPs.
- Max operator has been replaced by “Value”, which refers to \textit{equilibrium}.
- i.e., the matrix game is being solved at each state (step 2b)
The Policy Iteration Algorithm for SGs

1. Initialize $V$ arbitrarily.
2. Repeat,

$$
\begin{align*}
\rho_i & \leftarrow \text{Solve}_i \left[ G_s(V) \right] \\
V(s) & \leftarrow E \left\{ \sum \gamma^t r_t | s_0 = s, \rho_i \right\} .
\end{align*}
$$

Table 2: Algorithm: Pollatschek & Avi-Itzhak. The function $G_s$ is the same as presented in Table 1.

- This algorithm is akin to Policy Iteration for MDPs.
- Each player selects equilibrium policy according to current value function (using the same G matrix as in Shapley’s algorithm)
- Value function is then updated based on rewards as per equil. policy
Q-Learning for SGs

1. Initialize $Q(s \in S, a \in A)$ arbitrarily, and set $\alpha$ to be the learning rate.

2. Repeat,
   
   (a) From state $s$ select action $a_i$ that solves the matrix game $[Q(s, a)_{a \in A}]$, with some exploration.
   (b) Observing joint-action $a$, reward $r$, and next state $s'$,

   $$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma V(s')),$$

   where,

   $$V(s) = \text{Value} \left( \left[ Q(s, a)_{a \in A} \right] \right).$$

Table 3: Algorithm: Minimax-Q and Nash-Q. The difference between the algorithms is in the Value function and the $Q$ values. Minimax-Q uses the linear programming solution for zeros-sum games and Nash-Q uses the quadratic programming solution for general-sum games. Also, the $Q$ values in Nash-Q are actually a vector of expected rewards, one entry for each player.

- Q-learning version of Shapley’s algorithm (maintaining value over joint actions)
- Algorithm converges to stochastic game’s equilibrium, even if other player doesn’t, provided everyone executes all actions infinitely often.
What do we do if we have no Model? Fictitious Play [Robinson ‘51]

1. Initialize $V$ arbitrarily, $U_i(s \in S, a \in A_i) \leftarrow 0$, and $C_i(s \in S, a \in A_i) \leftarrow 0$.

2. Repeat: for every state $s$, let joint action $a = (a_1, a_2)$, such that $a_i = \arg\max_{a_i \in A_i} \frac{U^i(s, a_i)}{C_i(s, a_i)}$. Then,

$$C_i(s, a_i) \leftarrow C_i(s, a_i) + 1$$

$$U_i(s, a_i) \leftarrow U_i(s, a_i) + P_i(s, a) + \gamma \left( \sum_{s' \in S} T(s, a, s') V(s') \right)$$

$$V(s) \leftarrow \max_{a_1 \in A_1} \frac{U_1(s, a_1)}{C_1(s, a_1)}$$

Table 5: Algorithm: Fictitious play for two-player, zero-sum stochastic games using a model.

- Assumes opponents play stationary strategies
- Maintains information about average value of each action
- Finds equilibria in zero-sum and some general sum games
Summary: General Tactic for SGs

Matrix Game Solver + Temporal Differencing = Stochastic Game Solver
Summary: Many Approaches

$$\text{Matrix Game Solver} + \text{Temporal Differencing} = \text{Stochastic Game Solver}$$

<table>
<thead>
<tr>
<th>MG</th>
<th>TD</th>
<th>Game Theory</th>
<th>RL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>TD(0)</td>
<td>Shapley</td>
<td>MiniMax-Q</td>
</tr>
<tr>
<td>LP</td>
<td>TD(1)</td>
<td>Pollatschek &amp; Avi-Itzhak</td>
<td>–</td>
</tr>
<tr>
<td>LP</td>
<td>TD((\lambda))</td>
<td>Van der Wal[25]</td>
<td>–</td>
</tr>
<tr>
<td>Nash</td>
<td>TD(0)</td>
<td>–</td>
<td>Nash-Q</td>
</tr>
<tr>
<td>FP</td>
<td>TD(0)</td>
<td>Fictitious Play</td>
<td>Opponent-Modeling / JALs</td>
</tr>
</tbody>
</table>

LP: linear programming  
FP: fictitious play
Optional Reference/Acknowledgements

Learning algorithms for stochastic games are from the paper: M. Bowling, M. Veloso, An analysis of stochastic game theory for multiagent reinforcement learning, CMU-CS-00-165, 2000.

Several slides are adapted from the following sources:

• Tutorial at IJCAI 2003 by Prof Peter Stone, University of Texas
• Y. Peres, Game Theory, Alive (Lecture Notes)