MULTI-AGENT REINFORCEMENT LEARNING

Jonatan Milewski
Outline

• Introduction
• Reinforcement Learning
  • Q-Learning
• Multi-Agent Reinforcement Learning
  • Minimax Q-Learning
  • Q-Learning in general-sum games
• Conclusion
Introduction

- Context: **Repeated Games** & **Stochastic Games**
- Want to learn the best strategy against the opponent(s)
- Might not know all the payoff values beforehand
- Might not know the transition probabilities between states (in a stochastic game)
- Can use Reinforcement Learning!
Reinforcement Learning

- Inspired by behaviorist psychology
- Learn by interacting with the environment
- Trial-and-error approach
- Positive feedback encourages given behavior
- Negative feedback discourages given behavior
- Balance between exploration and exploitation
- Long-term payoff
Q-Learning

- Environment consists of states
- From each state agent can choose an action
- Each action has an associated reward
- After performing action, agent moves to another state (maybe)
Q-Learning

- Each **state-action** pair has a corresponding **Q-value**: represents **expected cumulative payoff** from performing action in the given state
- Update Q each time:
  \[
  Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_a Q(s', a))
  \]
  Learning rate (between 0 and 1)
  Discount factor (between 0 and 1)

- Goal: Find **“optimal policy”** i.e. actions that maximize \(V(s)\)
  \[
  V(s) \leftarrow \max_a Q(s, a)
  \]
Q-Learning

\[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_a Q(s', a)) \]

- How are actions chosen?
  - Randomly, with probability exploration
  - According to max Q(s,a) with probability exploitation
Q-Learning

\[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_a Q(s', a)) \]

• How are actions chosen?
  • Randomly, with probability exploration
  • According to max Q(s,a) with probability \( 1 - \text{exploitation} \)

• What’s a good learning rate (alpha) ?
  • \(1/k\)
  • 0.1
  • ...

Jonatan Milewski
Multi-Agent Reinforcement Learning
Outline

• Introduction
• Reinforcement Learning
  • Q-Learning
• Multi-Agent Reinforcement Learning
  • Minimax Q-Learning
  • Q-Learning in general-sum games
• Conclusion
Play This Game Repeatedly:

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L2</td>
</tr>
<tr>
<td>L1</td>
<td>0,0</td>
</tr>
<tr>
<td>R1</td>
<td>-10,10</td>
</tr>
</tbody>
</table>

```
L1  \rightarrow \quad  R1  \leftarrow \\
\leftarrow \quad \times \quad \rightarrow
```

```
L2  \leftarrow \\
\leftarrow \quad \quad \rightarrow \quad \leftarrow \\
R2  \rightarrow
```
Q-Learning in Zero-Sum Stochastic Games

- Naïve approach: apply Q-learning directly
  - Might not work well against a good opponent with a complex strategy
  - No guarantee of convergence
- Better approach: play MaxMin and converge to Nash
Minimax Q-Learning

- Q-values are over joint actions: $Q(s, a, o)$
  - $s =$ state
  - $a =$ your action
  - $o =$ action of the opponent
- Instead of playing action with highest $Q(s, a, o)$, play **MaxMin**

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

$$V(s) \leftarrow \max_{\pi_s} \min_o \sum_a Q(s, a, o)\pi_s(a)$$

probability of playing $a$ when following strategy $\pi_s$
Minimax Q-Learning

How are actions chosen?

- At the beginning set $\pi_s$ to select actions uniformly at random for each state
- Before each step:
  - Play random action with probability $\text{explor}$
  - Play according to $\pi_s$ with probability $1 - \text{explor}$
- After each step:
  - Update $\pi_s$ to the MaxMin strategy (based on $Q(s,a,o)$)
Minimax Q-Learning

• Does it work?
  • Performs better than naïve Q-learning
  • Guarantees convergence to Nash equilibrium (under certain conditions)
  • No guarantee of rate of convergence 😞
Q-Learning in General-Sum Games

• A much harder problem
• Nash Q-Learning:

\[ Q(s, a_1, \ldots, a_n) \leftarrow (1 - \alpha)Q(s, a_1, \ldots, a_n) + \alpha(R + \gamma \text{NashV}(s')) \]

• NashV(s) is the payoff value from computing a Nash equilibrium
• Must keep track of all players’ Q-values to compute NashV(s)
• Assumes all players play the same Nash equilibrium
Belief-based Reinforcement Learning

\[ Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha) Q_t(s_t, a_t) + \alpha_t (r(s_t, a_t) + \beta V_t(s_{t+1})) \]

\[ V_t(s) \leftarrow \max_{a_i} \sum_{a_{-i} \subseteq A_{-i}} Q_t(s, (a_i, a_{-i})) Pr_i(a_{-i}) \]

- Uses some beliefs about opponents’ strategy to calculate \( V(s) \)
- Ideally beliefs are updated after each move
Other approaches

• A Nash equilibrium is not always the “best” way to play
• Can use other solution concepts:
  • Correlated equilibrium
  • Pareto-optimality
  • Regret
  • …
• Methods developed for specific kinds of games
  • E.g. “coordination games” (Battle of the Sexes)
Conclusion

• Reinforcement learning can be useful in learning strategies in stochastic games
• It is not necessary to know the payoff matrix and transition probabilities beforehand
• Many methods’ success depends on the accuracy of assumptions about other players’ strategies
THANK YOU

Jonatan Milewski
References