Deep Reinforcement Learning (Sp25)

Instructor: Dr. Mohammad Hossein Rohban

Summary of: Lecture 26

Summarized By: Behnia Soleymani

- **I. Continue From Last Summary: Game Theory**
  - We can group games by how the agents' interests line up:
    - **Cooperative Games**: All agents work towards a common goal and get similar rewards. Team success is the aim.
    - **Competitive (Zero-Sum) Games**: Agents have opposite goals. The total rewards always add up to zero ($\Sigma_i R_i(a) = 0$). If one agent wins, another loses that exact amount.
    - **Mixed/General-Sum Games**: This is the most common type. Agents have their own reward goals, which might not be directly opposed or perfectly aligned. Many real-world situations fit here.
  - To figure out what might happen in a game, we use solution concepts:
    - **Mixed Strategy (σ_i)**: An agent doesn't just pick one action but chooses randomly from their actions according to a probability distribution.
    - **Mixed Strategy NE (σ^*)**: A set of mixed strategies $\sigma^* = (\sigma_1^*, \ldots, \sigma_N^*)$ is an NE if no agent can get a better average reward by changing their mixed strategy, assuming others keep playing their part of σ^*. Formally, $E[R_i(\sigma_i^*, \sigma_{-i}^*)] \geq E[R_i(\sigma_i', \sigma_{-i}^*)]$ for any other mixed strategy σ_i'.
    - **Nash's Theorem**: A famous result that says every Normal Form Game with a limited number of players and actions has at least one Nash Equilibrium (which might involve mixed strategies).
- **II. Moving to Sequential Choices: Multi-Agent Reinforcement Learning (MARL)**
  - MARL takes game theory ideas and uses them in situations where agents make many decisions one after another over time, learning as they go (like in regular Reinforcement Learning). Its where multi-agent games meet sequential decision-making.
  - The main model for MARL is the Stochastic Game (SG), or Markov Game. It's like an MDP but for N agents:
    - **Parts**:
      - S: A shared set of states, describing the environment's current situation.
      - $\{A^j\}_{j\in N}$: Each agent j has its own set of possible actions A^j. A joint action $a = (a^1, \ldots, a^N)$ is one action from each agent.
      - $T(s'|s, a)$: The transition function, giving the chance of moving to state s' from state s if joint action 'a' is taken.
      - $\{R^j\}_{j\in N}$: Each agent j has a reward function $R^j(s, a)$, giving the immediate reward for joint action 'a' in state s.
      - γ: A discount factor (between 0 and 1), making future rewards a bit less valuable than immediate ones.

    * **Policy (Strategy)** π^j: For agent j, this is a plan that says what action (or distribution over actions) to take in each state, $\pi^j : S \to \Omega(A^j)$.
    * **Value Function** $V^j_\pi(s)$: The expected total discounted future reward for agent j, if it starts in state s and all agents follow the joint policy $\pi = (\pi^1, \dots, \pi^N)$.
  - The idea of Nash Equilibrium also applies to Stochastic Games: a joint policy π^* is an NE if no agent j can get a better value $V^j(\pi^*_j, \pi^*_{-j})(s)$ by just changing its own policy to π_j, for any state s.

- **III. Big Challenges in MARL**
  - Making good MARL agents is tough because of these issues:
    * **Non-stationarity (Changing Environment)**: From one agent's viewpoint, the environment seems to keep changing because other agents are also learning and changing their strategies. This breaks a key assumption of many basic RL methods.
    * **Scalability (Growing Complexity)**: The number of possible joint actions ($|A_1| \times \cdots \times |A_N|$) gets huge very quickly as you add more agents. Storing values for all joint actions becomes impossible.
    * **Credit Assignment (Who Did What?)**: In team settings, especially if there's only one team reward, it's hard to tell which agent's actions helped or hurt the team's performance.
    * **Opponent Modelling (Guessing Others)**: To act smart, an agent often needs to guess or model what other agents will do or what their plans are. This is hard if opponents are also learning.
    * **Equilibrium Selection (Which NE?)**: Games can have many Nash Equilibria. If agents learn on their own and aim for different NEs, they might not coordinate well. Picking or agreeing on a good NE is tricky.

- **IV. Common MARL Algorithms**
  - **Independent Q-Learning (IQL)**:
    * How it works: The simplest idea. Each agent runs its own Q-learning, learning a Q-value $Q^i(s, a^i)$ based only on its own actions and rewards. It basically ignores that other agents are learning.
    * Update: $Q^i(s, a^i) \leftarrow Q^i(s, a^i) + \alpha(r^i + \gamma \max_{a'^i} Q^i(s', a'^i) - Q^i(s, a^i))$.
    * Good: Easy to set up. Bad: Faces the non-stationarity problem and often doesn't work well or converge reliably.
  - **Opponent Modelling: Fictitious Play**
    * How it works: A basic way to model opponents. An agent assumes opponents are playing a fixed (but possibly random) strategy.

    - Method: Agent i keeps track of how often opponent j has played action a^j in state s. Based on these counts, it estimates opponent j's strategy $Pr_t^i(a^j|s)$. Agent i then plays its best response to these estimated strategies.

  - **Joint Action Learners (JALs)**: These algorithms learn Q-values that depend on the actions of all agents, $Q(s, a^1, \dots, a^N)$.
    - **Joint Q-Learning (JQL)**:
      - Use Case: Good for games where all agents get the same reward $R(s, a)$ and teams are small. The goal is usually a "Pareto-dominating" NE (an NE that's at least as good for everyone and strictly better for someone, compared to other NEs), which is unique in these games.
      - Q-value: Agents learn a shared Q-function $Q(s, a)$.
      - Update: $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'^i} Q(s', a'^i, \hat{a}'_{-i}) - Q(s, a))$.
      - $\hat{a}'_{-i}$ are the guessed actions of other agents in the next state s'. Agent i needs an opponent model (like fictitious play) to make these guesses.
      - Convergence: JQL can find the true Nash Q-values in cooperative games if all agents use JQL (called "self-play"), explore enough, and the learning rate α is managed well (e.g., $\Sigma\alpha_n \to \infty, \Sigma(\alpha_n)^2 < \infty$).
    - **Minimax Q-Learning - For Competitive (Zero-Sum) SGs**:
      - Use Case: Good for two-player zero-sum games (or one agent vs. a team trying to minimize its reward). Aims for a min-max equilibrium, where the expected utilities are unique.
      - Q-value: Agent j learns $Q^j(s, a^j, a_{-j})$, its value if it takes action a^j and opponents take a_{-j}.
      - **Value of next state for agent j ($V^j(s')$)**: Agent j calculates this assuming it will pick its action to maximize its Q-value, while opponents will pick their actions to minimize agent j's Q-value: $V^j(s') = \max_{a'^j} \min_{a'_{-j}} Q^j(s', a'^j, a'_{-j})$.
      - Update: $Q^j(s, a^j, a_{-j}) \leftarrow (1 - \alpha)Q^j(s, a^j, a_{-j}) + \alpha(r^j + \gamma V^j(s'))$.
      - Convergence: Can find the min-max Q-values in self-play with enough exploration and good learning rates. Fictitious play can also be used here; for zero-sum games in self-play, it helps converge to the min-max action.
    - **General-Sum Stochastic Games**: When rewards are not purely cooperative or zero-sum, the problem is much harder. There can be many different Nash Equilibria. This is a big research area in MARL.

- **V. Key Things Needed for Convergence (for basic JQL & Minimax-Q):**

- For these early MARL algorithms to reliably find their target equilibria, a few things are usually needed:
  - **Self-play**: All agents in the game are using the same learning algorithm.
  - **Lots of Exploration**: Every state (or important state-action combination) must be tried out many, many times so the agents learn about all possibilities (e.g., using ϵ-greedy or Boltzmann exploration).
  - **Smart Learning Rate (α)**: The learning rate α should get smaller over time, but not too fast (e.g., $\alpha_t = 1/t$, or conditions like $\Sigma\alpha_t = \infty, \Sigma(\alpha_t)^2 < \infty$).
  - **Right Game Type**:
    - JQL works for cooperative SGs to find the unique Pareto-dominating Nash Q-values.
    - Minimax-Q works for competitive (zero-sum) SGs to find the unique min-max equilibrium Q-values.