Reinforcement Learning in Games

Game Theory · Module 9 — Learning In Games

Node 9.6

Position in Knowledge Graph

System: Game Theory **Structural Domain:** Learning in Games **Node:** 9.6

Reinforcement learning in games drops the most restrictive assumption of the classical learning framework: that players know the payoff function. In fictitious play (9.1), regret matching (9.2), and CFR (9.3), each player observes the full loss vector — the payoff of every action, not just the one chosen. RL agents observe only the payoff of the action they took (the "bandit" feedback model), and must balance *exploration* (trying suboptimal actions to learn about them) with *exploitation* (playing the currently best-known action). This shift from full-information to partial-information learning changes everything: convergence is harder to guarantee, rates are slower, and the multi-agent setting introduces non-stationarity that breaks single-agent RL guarantees. Despite these challenges, RL in games has produced some of the most spectacular AI results of the past decade: AlphaGo and AlphaZero (perfect-information board games), AlphaStar (StarCraft), OpenAI Five (Dota 2), and actor-critic methods applied to poker and other imperfect-information domains. This node develops the theory of RL in games — Q-learning, policy gradient, self-play, multi-agent RL — and maps the structural connections between RL and the game-theoretic learning dynamics of the rest of Module 9.

Formal Definition

Single-agent RL (background)

An agent interacts with an environment modeled as a Markov Decision Process (MDP): (S, A, P, R, γ) where S is the state space, A the action space, $P(s'|s,a)$ the transition kernel, $R(s,a)$ the reward function, and $\gamma \in [0,1)$ the discount factor. A policy $\pi : S \to \Delta(A)$ maps states to action distributions. The value function:

$$V^\pi(s) \;=\; \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\middle|\; s_0 = s\right],$$

and the action-value function:

$$Q^\pi(s,a) \;=\; R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s').$$

The optimal policy π^* satisfies $V^{\pi^*}(s) = \max_\pi V^\pi(s)$ for all s.

Q-learning (Watkins 1989)

A model-free algorithm that learns Q^* directly from experience:

$$Q(s,a) \;\leftarrow\; Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right],$$

where α is the learning rate and (s, a, R, s') is an observed transition. Under standard assumptions (every state-action pair visited infinitely often, α decaying appropriately), Q-learning converges to Q^* in single-agent settings.

Multi-agent extension: stochastic games

A **stochastic game** (Shapley 1953) generalizes MDPs to n players: $(S, \{A_i\}, P, \{R_i\}, \gamma)$, where the transition and rewards depend on the joint action profile. Each player's MDP is non-stationary from their perspective, because the other players' policies change over time. This non-stationarity is the fundamental obstacle: single-agent RL convergence theorems assume a stationary environment, which multi-agent play violates.

Independent Q-learning

Each player runs Q-learning independently, treating other players as part of the environment:

$$Q_i(s, a_i) \;\leftarrow\; Q_i(s, a_i) + \alpha \left[R_i(s, a_i, a_{-i}) + \gamma \max_{a_i'} Q_i(s', a_i') - Q_i(s, a_i) \right].$$

No convergence guarantee. Since each player's "environment" changes as others learn, the stationarity assumption of Q-learning is violated. Independent Q-learning can diverge, cycle, or converge to suboptimal equilibria. Empirically, it sometimes works in simple games but frequently fails in complex ones.

Self-play

An agent plays against copies of itself (or historical versions of itself). The resulting training process generates a sequence of policies $\pi^0, \pi^1, \pi^2, \ldots$ where each π^{t+1} is trained against π^t (or a mixture of past policies). In two-player zero-sum games, self-play with appropriate averaging converges to Nash — this is because self-play is effectively a form of fictitious play or no-regret learning where the "opponent model" is the agent's own past.

Policy gradient methods

Instead of learning Q-values, directly parameterize the policy π_θ and update via gradient ascent on expected reward:

$$\theta \;\leftarrow\; \theta + \alpha \nabla_\theta J(\theta), \qquad J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_t \gamma^t R_t \right].$$

The REINFORCE estimator (Williams 1992):

$$\nabla_\theta J(\theta) \;=\; \mathbb{E} \left[\sum_t \nabla_\theta \ln \pi_\theta(a_t | s_t) \cdot G_t \right],$$

where $G_t = \sum_{\tau \geq t} \gamma^{\tau - t} R_\tau$ is the return from time t.

Actor-critic methods

Combine a policy (the "actor") with a value-function estimator (the "critic"). The actor updates via policy gradient using the critic's value estimates as a baseline, reducing variance:

$$\theta \;\leftarrow\; \theta + \alpha \nabla_\theta \ln \pi_\theta(a_t | s_t) \cdot \left(R_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \right),$$

where V_ϕ is the critic's learned value function. In multi-agent settings, each player can maintain their own actor-critic, leading to independent actor-critic (IAC), or a centralized critic can observe the joint state while actors observe only local information (centralized training, decentralized execution — CTDE).

Intuition

The exploration-exploitation tradeoff. In full-information settings (9.2–9.4), the learner sees the payoff of every action at each round, so there is no need to explore — you can evaluate all actions without trying them. In RL, you only observe the payoff of the action you took. To estimate the value of an untried action, you must try it — potentially sacrificing payoff in the process. This tradeoff is fundamental and does not exist in the full-information regime.

Non-stationarity is the killer. The single-agent RL convergence theory (Q-learning converges, policy gradient is unbiased, etc.) assumes a *stationary* environment. In a game, the environment is the other players, who are also learning. From each player's perspective, the reward function and transition kernel change over time as opponents change their policies. This makes the "MDP" that each player faces non-stationary, breaking every single-agent guarantee. The multi-agent RL (MARL) field is largely about finding conditions under which convergence can be restored despite this non-stationarity.

Self-play as a structural fix. In two-player zero-sum games, self-play sidesteps the non-stationarity problem by making the opponent a function of the learner's own history. If the learner improves, the opponent improves at the same rate (because it is the same agent). This creates a "co-evolution" dynamic that, under appropriate conditions, converges to Nash. Self-play is the RL analogue of the no-regret convergence theorem: in zero-sum games, the structure of the game forces convergence even though each individual learner faces a moving target.

Deep RL: function approximation changes the game. When Q-functions or policies are represented by neural networks rather than tables, new phenomena arise: approximation error, catastrophic forgetting, training instability, and mode collapse. These are not game-theoretic issues per se, but they interact with the game-theoretic convergence properties in complex ways. AlphaZero's success shows that deep RL self-play can work spectacularly in practice (in perfect-information games); AlphaStar shows that it can work in imperfect-information games with massive state spaces; but theoretical guarantees for deep RL in games remain sparse.

Structural Role

9.6 is the bridge from classical game-theoretic learning (9.1–9.5) to modern AI:

- **From 9.1 (fictitious play):** Self-play is RL's version of fictitious play — iterate best responses against the opponent's historical play.
- **From 9.2 (no-regret):** Policy gradient with entropy regularization can be interpreted as running a no-regret algorithm (MWU variant) in policy space.
- **From 9.3 (CFR):** CFR requires a model of the game tree; RL does not. RL can learn in games where the tree is unknown or too large to enumerate.
- **From 9.5 (convergence):** RL inherits all the convergence difficulties of the full-information setting, plus the additional problems of partial information and function approximation.

9.6 also connects outward to the AI milestones that brought game-theoretic concepts to public attention:

- **AlphaGo (2016, Silver et al.):** Deep RL + Monte Carlo tree search in the perfect-information game of Go.
- **AlphaZero (2017, Silver et al.):** Self-play RL from scratch (no human data) in Go, chess, and shogi. Demonstrates that self-play + deep RL + MCTS can rediscover and surpass centuries of human knowledge in perfect-information games.

- **AlphaStar (2019, Vinyals et al.):** Deep RL self-play in StarCraft II, an imperfect-information, real-time strategy game with massive state and action spaces. Used a league of agents with diverse strategies to avoid self-play convergence failures.
- **OpenAI Five (2019):** Deep RL in Dota 2 (5v5 team game), trained via self-play with long-horizon RL.

Mathematical Development

Failures of independent Q-learning

Example: coordination game. Two players, two actions each: (A, A) = (1, 1), (B, B) = (1, 1), (A, B) = (0, 0), (B, A) = (0, 0). Two Nash equilibria: (A, A) and (B, B). Independent Q-learning with ϵ-greedy exploration: each player estimates Q-values for A and B based on observed rewards. If player 1 happens to explore B while player 2 plays A, both observe zero reward, and player 1 decreases $Q_1(B)$. The learning dynamics are path-dependent: the equilibrium reached depends on the random exploration sequence. In some runs, the players converge to (A, A); in others, to (B, B); and in degenerate cases, they cycle between the two.

Example: Prisoner's Dilemma. Independent Q-learning converges to (D, D) — the stage Nash — because defection dominates. No exploration can discover that (C, C) is Pareto-superior, because each player's Q-values correctly reflect the fact that cooperating against a defector is worse than defecting. This is not a failure of the algorithm; it is a correct equilibrium computation. But it shows that RL finds Nash, not welfare optima.

Example: matching pennies. Independent Q-learning can fail to converge entirely. The Q-values oscillate as each player chases the other's strategy. With function approximation (deep Q-learning), the oscillations can be amplified, leading to divergence.

Nash Q-learning (Hu and Wellman 2003)

A modification of Q-learning where each player, instead of maximizing their own Q-value, computes a Nash equilibrium of the stage game defined by the current Q-values:

$$Q_i(s, a_i, a_{-i}) \leftarrow Q_i(s, a_i, a_{-i}) + \alpha \left[R_i + \gamma \, \text{Nash}_i(Q(s', \cdot)) - Q_i(s, a_i, a_{-i}) \right],$$

where $\text{Nash}_i(Q(s', \cdot))$ is player i's payoff in the Nash equilibrium of the matrix game $Q(s', \cdot, \cdot)$.

Convergence. Nash Q-learning converges in two-player zero-sum stochastic games (because Nash of a zero-sum matrix game is unique and efficiently computable) and in certain special classes of general-sum games. It does not converge in general (because computing Nash of a general matrix game is PPAD-hard and may not be unique).

Policy gradient in multi-agent settings

Each player i parameterizes their policy as π_{θ_i} and updates via:

$$\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} J_i(\theta_1, \ldots, \theta_n),$$

where J_i is player i's expected reward under the joint policy. This is **gradient play** — each player follows the gradient of their own objective. The fixed points of gradient play are the first-order stationary points of the game, which include Nash equilibria but also non-Nash stationary points.

Convergence in zero-sum games. In two-player zero-sum games, policy gradient with appropriate learning rates converges to Nash in time-averaged play (connecting to the gradient-dynamics analysis of 9.5).

Divergence in general-sum games. In general-sum games, gradient play can cycle (9.5), and policy gradient inherits this problem. Adding entropy regularization (SAC, maximum-entropy RL) can stabilize the dynamics by making the objective strongly concave, but changes the target equilibrium (the entropy-regularized Nash is not the same as Nash).

Self-play convergence

Theorem (informal, Heinrich and Silver 2016). In two-player zero-sum games, neural fictitious self-play (NFSP) — where each player trains a best-response network via RL and an average-strategy network via supervised learning on its own past play — converges to an approximate Nash equilibrium. The rate depends on the function approximation quality and the mixing of best-response and average-strategy play.

NFSP is the RL analogue of CFR: it approximates the per-info-set regret-matching process of CFR using neural networks to represent the strategy, and substitutes RL (DQN) for the exact value computation. The convergence guarantee is weaker than CFR's (no exploitability bound in the general case), but NFSP can handle games too large for explicit CFR.

AlphaZero: the deep RL paradigm for perfect-information games

AlphaZero (Silver et al. 2017) combines:

1. **Monte Carlo tree search (MCTS):** At each position, build a search tree by simulating many rollouts, guided by a neural network that estimates value and policy.
2. **Self-play:** Generate training data by playing games against itself.
3. **Neural network training:** Train the network to predict (a) the MCTS policy output and (b) the game outcome from each position.

The training loop is: self-play $\to$ generate data $\to$ train network $\to$ use improved network for next round of self-play. After many iterations, the policy and value estimates converge to near-optimal play. In chess, Go, and shogi, AlphaZero achieved superhuman performance within hours of training.

Game-theoretic interpretation. AlphaZero's self-play is a form of policy iteration in a perfect-information zero-sum game. Each iteration of self-play computes an improved best response to the current policy (via MCTS), and the network training aggregates these improvements into a single policy. This is structurally similar to fictitious play, but with deep function approximation replacing the empirical frequency.

Limitation: perfect information only. AlphaZero relies on MCTS, which requires the ability to simulate the game from any state. In imperfect-information games (poker), you cannot simulate from a single state because you do not know the opponent's private information. This is why poker AI uses CFR-based methods rather than AlphaZero-style MCTS.

AlphaStar and league training

AlphaStar (Vinyals et al. 2019) tackled StarCraft II, an imperfect-information, real-time strategy game with $\sim 10^{26}$ possible actions per step. Key innovations:

1. **League training:** Instead of pure self-play (which can converge to a narrow strategy that beats itself but loses to diverse opponents), AlphaStar maintains a "league" of agents with diverse strategies. New agents are trained against the league, and the league is updated to include strong and diverse policies.
2. **Population-based training:** Multiple agents are trained in parallel with different hyperparameters, and the best performers seed the next generation.
3. **Exploiter agents:** Special agents trained to exploit weaknesses in the current league members, forcing the league to be robust.

The league structure is an engineering solution to the non-convergence problem in general-sum games: by maintaining a diverse population, AlphaStar avoids the cycling and instability that plague simple self-play in non-zero-sum settings.

Worked Example

Independent Q-learning in Rock-Paper-Scissors

Two players, each running ϵ-greedy Q-learning with $\epsilon = 0.1$, $\alpha = 0.01$, against each other. Payoff matrix for Row:

$$A = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}.$$

Initialize $Q_i(a) = 0$ for all i, a. Each round: both players choose actions (ϵ-greedy on Q-values), observe rewards, update Q-values.

After 10,000 rounds, the Q-values oscillate without converging to a fixed point. The empirical action frequencies converge (slowly) to approximately $(1/3, 1/3, 1/3)$ for both players — the Nash equilibrium — because the time-averaging effect smooths out the oscillations. But the Q-values themselves do not stabilize: $Q_i(R)$, $Q_i(P)$, $Q_i(S)$ fluctuate around 0 as the opponents' strategies shift.

Compare to regret matching (9.2), which converges cleanly and monotonically. The RL version achieves the same long-run result but with much higher variance and no per-iteration exploitability guarantee.

Self-play policy gradient in a simple poker game

Consider Kuhn Poker (the toy game from 9.3). Train two policy-gradient agents via self-play:

1. Parameterize each player's policy by a small neural network mapping info-set features to action probabilities.
2. Play 1,000 hands of self-play, collecting trajectories.
3. Compute policy-gradient updates for both players.
4. Repeat.

After 100,000 episodes, the policies approximate the Nash equilibrium of Kuhn Poker (see 9.3 for the analytic solution). The approximation quality depends on the network architecture and learning rate. Typical exploitability: 0.01-0.05 chips, compared to CFR's 0.001 after the same number of game evaluations. The RL approach is less sample-efficient but does not require knowledge of the game tree.

Poker Anchor

Concept mapping

Formal object	Poker instantiation
MDP / stochastic game	The poker game as a sequential decision process with unknown opponent policy
Q-function $Q(s, a)$	EV of taking action a at game state s (hand + board + history)
Policy π_θ	A neural-network strategy mapping info sets to action distributions
Self-play	Training by playing against yourself (or past versions of yourself)
Exploration (ϵ-greedy)	Occasionally making "random" plays to discover new strategic information
Function approximation	Using a neural net to generalize across similar poker situations

Formal object	Poker instantiation
Non-stationarity	Your opponent is also learning, so the "environment" changes over time

Concrete situation: why poker AI uses CFR, not pure RL

CFR (9.3) has two advantages over RL for poker:

1. **Full-information feedback.** CFR's tree traversal computes counterfactual values for *every* action at *every* info set, even ones not taken. RL only observes the reward of the action taken. This makes CFR far more sample-efficient: one CFR iteration updates regrets at every info set, while one RL episode updates the policy only along the single trajectory played.
2. **Convergence guarantees.** CFR has a provable $O(T^{-1/2})$ exploitability bound. RL in multi-agent settings has no comparable guarantee. The non-stationarity of multi-agent learning means that RL can cycle, diverge, or converge to exploitable strategies.

However, RL has advantages in settings where CFR is impractical:

- **Unknown game rules.** CFR requires a complete model of the game tree. RL can learn from interaction without knowing the rules.
- **Continuous action spaces.** CFR requires discretizing actions; RL can handle continuous actions via policy gradient.
- **Very large games.** When the game tree is too large even for MCCFR with abstraction, RL with function approximation can still learn (AlphaStar, OpenAI Five).

The state of the art in poker AI is a hybrid: CFR for the core equilibrium computation (where convergence guarantees matter), RL-style techniques for value estimation in subgame solving (DeepStack's neural-net value function).

Concrete situation: training a poker bot via self-play RL

If you were to train a HUNL poker bot using pure RL self-play (no CFR, no game tree model), you would:

1. Parameterize the bot's strategy as a neural network mapping (hand features, board features, action history) to action probabilities.
2. Play millions of hands against itself.
3. After each hand, compute policy-gradient updates using the hand's reward.
4. Repeat, periodically evaluating exploitability against a known GTO strategy.

This approach works in principle but is enormously less efficient than CFR. The key bottleneck is exploration: in HUNL, the space of possible game states is 10^{161}, and each self-play hand explores only one trajectory. CFR, by contrast, updates all info sets in each iteration. Empirically, pure RL self-play in HUNL requires orders of magnitude more compute than CFR to reach the same exploitability.

What this understanding buys you

CFR and RL are complementary, not competing. CFR gives you convergence guarantees and sample efficiency in games with known structure. RL gives you flexibility in games with unknown structure or continuous actions. Modern poker AI (DeepStack, Libratus, Pluribus) uses both: CFR for equilibrium computation, neural networks (trained by supervised learning or RL) for value estimation in real-time subgame solving.

Self-play is not magic. Self-play works well in two-player zero-sum games because of the minimax theorem: improving against yourself improves against everyone. In general-sum or multiplayer games, self-play can converge to narrow strategies that exploit their own weaknesses without being robust to diverse opponents. This is why AlphaStar uses league training rather than pure self-play, and why Pluribus's self-play blueprint is supplemented with online subgame solving.

The exploration-exploitation tradeoff in poker study. When you sit down to play and decide between "play GTO" (exploit) and "try a new line to see what happens" (explore), you are instantiating the exploration-exploitation tradeoff of RL. GTO play exploits your current knowledge; deviating from GTO explores new strategic territory. The theory says some exploration is necessary for long-run improvement, but too much exploration costs money in the short run. The optimal balance depends on your current skill level, the stakes, and how far your strategy is from optimal.

Cross-Links

- **9.1 Fictitious play** — self-play is the RL analogue.
- **9.2 No-regret learning** — policy gradient with entropy is an approximate no-regret method.
- **9.3 CFR** — the model-based alternative with convergence guarantees.
- **9.5 Convergence to equilibrium** — RL inherits all convergence difficulties plus sampling noise.
- **7.3 Replicator dynamics** — population-based RL training echoes evolutionary dynamics.
- **3.4 Backward induction** — AlphaZero's MCTS is a learned version of backward induction.
- **MDP and control theory** — single-agent RL foundations.
- **Deep learning** — function approximation for large state spaces.

Structural Compression

Invariant: Reinforcement learning in games replaces full-information feedback with bandit feedback (observing only the payoff of the chosen action), introducing exploration-exploitation tradeoffs and non-stationarity from other learners; self-play in two-player zero-sum games converges to Nash under appropriate conditions, but independent RL in general games has no convergence guarantee to Nash.

Mechanism: Q-learning or policy gradient updates estimate action values or policy gradients from sampled trajectories. Self-play generates training data against the agent's own (evolving) policy. Multi-agent non-stationarity is mitigated by population-based training (league), centralized critics (CTDE), or averaging over policy histories (NFSP). Function approximation (neural networks) enables scaling to large state spaces at the cost of losing convergence guarantees.

Cross-System Echo: Multi-agent RL is the computational instantiation of **co-evolution in biology**: two populations adapting to each other in real time, with no fixed fitness landscape. The Red Queen effect — "you have to keep running just to stay in the same place" — is the biological analogue of non-stationarity in multi-agent learning. League training (AlphaStar) is the computational analogue of maintaining biodiversity to prevent evolutionary dead ends. The structural lesson is universal: in any system where multiple adaptive agents interact, the "environment" is not fixed, and single-agent optimization principles must be augmented with game-theoretic reasoning about the co-adaptation of others.

Exercises

1. Implement independent Q-learning for two players in Rock-Paper-Scissors. Run for 100,000 rounds and plot the Q-values and empirical action frequencies over time. Does the empirical frequency converge to Nash? Do the Q-values converge?

2. Show that independent Q-learning can converge to different Nash equilibria in a coordination game depending on the exploration sequence. Run 100 independent trials and report the distribution of outcomes.

3. Derive the policy-gradient theorem: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_t \nabla_\theta \ln \pi_\theta(a_t|s_t) \cdot G_t]$. Why is this an unbiased estimator of the gradient of expected reward?

4. Explain why AlphaZero's approach (MCTS + self-play + neural networks) cannot be directly applied to poker. What specific structural feature of poker (imperfect information) prevents MCTS from being used without modification?

5. Compare the sample efficiency of CFR and self-play RL on Kuhn Poker. How many game evaluations does each require to reach exploitability below 0.01 chips? What accounts for the difference?

6. Describe the non-stationarity problem in multi-agent RL and explain how league training (AlphaStar) mitigates it. What is the game-theoretic interpretation of maintaining a diverse league?

7. **Argue, structurally,** why the combination of RL (for value estimation) and CFR (for equilibrium computation) is more powerful than either alone for solving large imperfect-information games. What does each method contribute that the other cannot?

Game Theory · Module 9 — Learning In Games · Node 9.6
Concepts: markov-chains, convergence, adaptive-systems, stochastic-optimization
Prerequisites: 9.1, 9.2, 9.5
Enables: 10.3

hbar.university — own.your.cognition