Reinforcement Learning from Self-Play in Imperfect-Information Games

Johannes Heinrich

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

March 9, 2017

Abstract

This thesis investigates artificial agents learning to make strategic decisions in imperfect-information games. In particular, we introduce a novel approach to reinforcement learning from self-play.

We introduce Smooth UCT, which combines the game-theoretic notion of fictitious play with Monte Carlo Tree Search (MCTS). Smooth UCT outperformed a classic MCTS method in several imperfect-information poker games and won three silver medals in the 2014 Annual Computer Poker Competition.

We develop Extensive-Form Fictitious Play (XFP) that is entirely implemented in sequential strategies, thus extending this prominent game-theoretic model of learning to sequential games. XFP provides a principled foundation for self-play reinforcement learning in imperfect-information games.

We introduce Fictitious Self-Play (FSP), a class of sample-based reinforcement learning algorithms that approximate XFP. We instantiate FSP with neural-network function approximation and deep learning techniques, producing Neural FSP (NFSP). We demonstrate that (approximate) Nash equilibria and their representations (abstractions) can be learned using NFSP end to end, i.e. interfacing with the raw inputs and outputs of the domain.

NFSP approached the performance of state-of-the-art, superhuman algorithms in Limit Texas Hold'em - an imperfect-information game at the absolute limit of tractability using massive computational resources. This is the first time that any reinforcement learning algorithm, learning solely from game outcomes without prior domain knowledge, achieved such a feat.

Acknowledgements

Heartfelt thanks to:

David Silver, for inspiring and guiding me in the fascinating world of reinforcement learning and artificial intelligence

Peter Dayan, for his thought-provoking feedback and ideas

Philip Treleaven, for his mentorship and support

Peter Cowling and **Thore Graepel**, for their valuable feedback and a memorable viva

The **DeepMind** team who greatly inspired, taught and supported me: **Marc Lanctot, Georg Ostrovski, Helen King, Marc Bellemare, Joel Veness, Tom Schaul, Thomas Degris, Nicolas Heess, Volodymyr Mnih, Remi Munos, Demis Hassabis**

John Shawe-Taylor, for his feedback and support

James Pitkin, for the fun times

Sam Devlin, for his friendship and guidance

The **CDT in Financial Computing & Analytics**, for the support and great research environment

NVIDIA, for the hardware grant

My family, for their love and support

Martina Ankenbrand, for our love and adventures

Strategy without tactics is the slowest route to victory.

Tactics without strategy is the noise before defeat.

— Sun Tzu

One day Alice came to a fork in the road and saw a Cheshire cat in a tree. “Which road do I take?” she asked. “Where do you want to go?” was his response. “I don’t know”, Alice answered. “Then,” said the cat, “it doesn’t matter.”

— Lewis Carroll, Alice in Wonderland

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Games are domains of conflict or cooperation (Myerson, 1991) between decision-making intelligences, who are termed agents. Our world is full of imperfect-information games. Examples include airport security, traffic control, financial trading, sales & marketing, tax evasion, sports and poker. This thesis addresses the problem of artificial agents learning to make strategic decisions in imperfect-information games.

Agents exist in an environment. An agent's experience in its environment is a sequence of observations it perceives, actions it chooses, and rewards it receives. The real-valued reward signal determines how the agent perceives its experience, e.g. falling off a bike would be a painful, negative-reward experience. The observations encode the information that the agent gathers via its senses, e.g. sight or touch but also non-human senses such as a phone's accelerometer. An agent's goal is to maximise its total long-term reward. To achieve its goal it has to experientially learn a policy for choosing optimal (reward-maximising) actions, which may include forgoing short-term reward to achieve higher long-term reward. For example, an agent might try different restaurants and dishes on the menu instead of settling on a single choice.

Reinforcement learning is a paradigm of goal-directed intelligent learning from an agent's experience (Sutton and Barto, 1998). A reinforcement-learning agent is in charge of exploring its environment to gather the experience it may need for learning a policy on its own. In particular, positive (negative) rewards reinforce (inhibit) the agent's decisions. This could be described as trial-and-error learning. Unlike in supervised learning, there is no external teacher[1] who presents the agent with the optimal action (label) for a given situation (input). Unlike in traditional programming, the agent's decisions are not hard-coded. These distinctive properties of reinforcement learning were key to achieving superhuman performance in Backgammon (Tesauro, 1995), Go (Silver et al., 2016a), and Atari games (Mnih et al., 2015).

Game theory is the classic science of strategic decision making in games. Strategic decisions take into account the motives and intelligence of other agents, who might be cooperative or adversarial. Game theorists have traditionally studied the decision making of rational agents (Myerson, 1991). Rational agents maximise their rewards and are assumed to have the ability to do so. A seminal game-theoretic result is that every game has a Nash equilibrium (Nash, 1951), i.e. an assignment of policies from which no rational agent would choose to deviate.

1.1 Strategic Decision Making

In a single-agent task, such as a robot lifting inanimate objects or a machine classifying images, the agent's environment is usually assumed to be stationary, i.e. its dynamics do not change if the task is repeated. This property is very useful to a reinforcement learner, as it can gather experience from repeatedly performing the task. In contrast, in a multi-agent game an agent's decisions affect fellow agents' experience, causing them to dynamically adapt their behaviour. Due to these reciprocal effects, experience of specific strategic scenarios can be scarce. This poses significant challenges to experiential learning.

1.1.1 Adaptive (Exploitative) Approach

Consider an agent repeatedly playing a game against an opponent. Each round of the game adds to the experience of both agents. In particular, the opponent will learn from this experience and thus (indefinitely) alter its internal state, e.g. memory and policy. Technically, as the opponent cannot be forced to forget the past, an agent

[1]A teacher could be encountered within the agent's environment.

can therefore never repeat the same iteration of the game against the same opponent. How could the agent still effectively learn from trial and error? It would have to generalise from unique experiences. First, there might be generalisable patterns in the opponent's decision making that allow the agent to transfer knowledge across the sequence of repeated games. This could be the case when the opponent is using a simple heuristic or algorithm for making its decisions or when it has a bounded memory or other limitations. Second, the agent could play against different opponents and attempt to generalise from these experiences. This would be possible if there were (identifiable) groups of opponents that think similarly.

While the adaptive approach has the potential of being optimal, i.e. reward maximising, it poses several challenges. First, acquiring experience and learning from trial and error can be particularly costly in an adversarial game, such as airport security or poker. Second, accelerating the experience acquisition by using data or experience of other agents may not be an option for imperfect-information games. Finally, a superior opponent, who learns faster, could outplay an adaptive agent by anticipating its decisions. For example, by employing a theory of mind (Premack and Woodruff, 1978), an opponent could be already thinking about what the agent is thinking about him (Yoshida et al., 2008), i.e. the opponent's decision making would be one step ahead.

1.1.2 Static (Defensive) Approach

In a Nash equilibrium every agent is maximising its reward and therefore may as well continue using its respective policy. Thus, a Nash equilibrium can be regarded as a fixed point of multi-agent learning. In fact, by definition (Nash, 1951), Nash equilibria are the only policies that rational agents can ever converge on.

What are the use cases of Nash equilibria? First, learning a Nash equilibrium, in principle, only requires knowledge of the game's rules or access to a black-box simulator. In particular, no prior experience or domain knowledge is required. Second, a Nash equilibrium may be predictive of sophisticated, i.e. approximately rational, agents' behaviour. This can be useful in designing multi-agent systems that cannot be gamed, e.g. auctions (Roth, 2002) or blockchains (Nakamoto, 2008).

Third, decentrally deployed cooperative agents are sometimes managed by a single entity, e.g. a city's traffic lights system. All optimal policies of these agents would be Nash equilibria. The goal would be to find the best Nash equilibrium, i.e. a global optimum. Fourth, in adversarial (two-player zero-sum) games Nash equilibria offer attractive worst-case guarantees (Neumann, 1928). In particular, choosing a Nash equilibrium is optimal against a worst-case opponent, i.e. an opponent that always knows the agent's policy and plays perfectly against it. This is useful in security domains, where an attacker might choose to only act on detected weaknesses (Tambe, 2011). Furthermore, a Nash equilibrium is unbeatable (in expectation) in alternating adversarial games, e.g. poker. Finally, a Nash equilibrium could be used as an initial policy and then tailored to fellow agents by learning from experience (Johanson et al., 2008). This could mitigate some of the costs of experience acquisition (compare Section 1.1.1).

Using static Nash equilibria poses the following challenges. First, choosing a policy from a Nash equilibrium is not always optimal. For example, an irrational opponent might make mistakes that the agent could learn to exploit for higher reward than a Nash equilibrium might offer (see Section 1.1.1). Also, in games with more than two players (or general-sum games) all agents generally have to select the same Nash equilibrium in order to be offered its optimality guarantees, e.g. if two cooperative neighbouring city councils operate their own respective traffic lights systems, they might have to coordinate on the same Nash equilibrium to achieve optimal traffic flow. Second, the reward signals of fellow agents are subjective and generally unperceivable by other agents. For instance, fellow agents' reward signals might encode individual, altruistic, moral and utilitarian motives (Hula et al., 2015). Therefore, a Nash equilibrium of a subjective model of a game might be unapplicable to the environment if the assumed rewards do not match the real reward signals of fellow agents. This is less of an issue for purely adversarial games, such as poker, football or tax evasion, where the opponent's reward might just be assumed to be the negative of the relevant agent's reward. In this case, a Nash equilibrium would offer the typical worst-case guarantees with respect to the relevant agent's reward signal, e.g. breaking even in symmetric or alternating games.

1.1.3 Self-Play

Self-play is the activity of imagining or simulating the play of a game against oneself. For instance, a human agent could sit down in front of a chess board and play out game positions either mentally or physically on the board. This educational activity yields experience that humans can learn from. Similarly, an artificial agent can learn from self-play (Shannon, 1950; Samuel, 1959). Many practical successes of artificial intelligence in games have been based on self-play (Tesauro, 1995; Veness et al., 2009; Bowling et al., 2015; Brown et al., 2015; Silver et al., 2016a).

Self-play can be alternatively interpreted as a fictitious interaction of multiple agents. This is because a self-playing agent would have to imagine itself as each player[2] in the game, taking into account the respective players' private information that would be available only to them in an imperfect-information game. This is technically equivalent to imagining multiple agents interacting in the game. Therefore, our discussion of Nash equilibria as fixed points of multi-agent learning (see Section 1.1.2) also applies to self-play. In particular, Nash equilibria are the only policies that rational learning from self-play can converge on (Bowling and Veloso, 2001). Vice versa, Nash equilibria can, in principle, be learned from self-play.

1.2 Research Question

Our overall goal is to answer the question,

> Can an agent feasibly learn *approximate* Nash equilibria of *large-scale imperfect-information* games from self-play?

1.2.1 Motivation

Real-world games are typically **large-scale** and only partially observable, requiring agents to make decisions under **imperfect information**. By large-scale, we mean the game to have a variety of situations that cannot be exhaustively enumerated with

[2] A game's player is merely the specification of a role. This role can be implemented by an agent.

reasonable computational resources. In particular, specific actions for each situation cannot be feasibly provided by a lookup table.

We have established the utility of Nash equilibria in Section 1.1.2. In large-scale games we generally have to settle for an **approximate** Nash equilibrium, i.e. an assignment of policies from which agents have little incentive to deviate.

1.2.2 Related Work

In this section we discuss prior work related to our research question, focusing on the respective aspects that have not fully addressed our question.

Optimisation Nash equilibria have been traditionally computed by mathematical programming techniques (Koller et al., 1996; Gilpin et al., 2007; Hoda et al., 2010; Bosansky et al., 2014). An advantage of some of these methods is that they produce a perfect equilibrium. However, these methods generally need to enumerate and encode the various aspects of a game and thus do not apply to large-scale games.

Fictitious play (Brown, 1951) is a classic game-theoretic model of learning from self-play (Robinson, 1951; Fudenberg and Levine, 1995; Monderer and Shapley, 1996; Hofbauer and Sandholm, 2002; Leslie and Collins, 2006). Fictitious players repeatedly play a game, at each iteration choosing the best policy in hindsight. This is quite similar to how a reinforcement learner might act. However, fictitious play has been mainly studied in single-step games, which can only inefficiently represent sequential, multi-step games (Kuhn, 1953). A sequential variant of fictitious play was introduced but lacks convergence guarantees in imperfect-information games (Hendon et al., 1996).

Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2007) is a full-width[3] self-play approach guaranteed to converge in sequential adversarial (two-player zero-sum) games. Monte Carlo CFR (MCCFR) (Lanctot et al., 2009) extends CFR to learning from sampled subsets of game situations. Outcome Sampling (OS) (Lanctot et al., 2009; Lisý et al., 2015) is an MCCFR variant that samples single game trajectories. It can be regarded as an experiential, reinforcement learning method. However, OS, MCCFR and CFR are all table-lookup approaches that lack

[3] Generally, each iteration has to consider all game situations or actions.

the ability to learn abstract patterns and use them to generalise to novel situations, which is essential in large-scale games. An extension of CFR to function approximation (Waugh et al., 2015) is able to learn such patterns but has not been combined with sampling. Therefore, it still needs to enumerate all game situations. Finally, the aforementioned CFR variants, except OS, are unsuitable for agent-based learning from experience.

CFR-based methods have essentially solved two-player Limit Texas Hold'em poker (Bowling et al., 2015) and produced a champion program for the No-Limit variant (Brown et al., 2015). Both of these games can be considered large-scale. However, in order to achieve computational tractability, a model of the game had to be exploited with domain-specific human-engineered routines, e.g. compression techniques, abstractions and massive distributed computing. In contrast, an agent would learn solely from its experience and generalise to unseen situations rather than exhaustively enumerate them. Such an agent may also be versatilely applied to other games by connecting it to the respective environments' streams of observations, actions and rewards.

Abstraction The limited scalability of CFR and mathematical programming methods has traditionally been circumvented by solving abstractions of games. An abstraction reduces the game to a manageable size. Forming an abstraction generally requires prior domain knowledge (experience). In early work on computer poker (Billings et al., 2003; Zinkevich et al., 2007; Johanson et al., 2012), agents did not learn abstractions from their own experience. Instead they were injected with a hard-coded abstraction, which was handcrafted based on human domain expertise. Apart from the resource requirements for creating such abstractions, hard-coded abstraction can eternally bias an agent and prevent it from achieving its goals. In particular, handcrafted abstractions can result in unintuitive pathologies of agents' policies (Waugh et al., 2009a). Recent work (Sandholm and Singh, 2012; Kroer and Sandholm, 2014; Brown and Sandholm, 2015) has explored automated abstraction techniques, which refine an abstraction during training based on (computed) game outcomes. Similarly, an end-to-end approach to experiential learning from self-play could allow an agent to learn both the game's optimal (Nash equilibrium) policy as well as a suitable game abstraction (features) to represent it.

Reinforcement learning has been successful in learning end to end (Riedmiller, 2005; Heess et al., 2012; Mnih et al., 2015, 2016), i.e. interfacing with the raw stream of the environment's observations, actions, and rewards. Many applications of experiential learning from self-play can be well described as reinforcement learning. In adversarial games, agents try to maximise (minimise) their own (opponents') rewards in a *Minimax* fashion (Shannon, 1950; Littman, 1996). While classical reinforcement learning methods have achieved high-quality solutions to recreational, perfect-information games (Tesauro, 1995; Veness et al., 2009; Silver et al., 2016a), these methods fail to converge (on Nash equilibria) in imperfect-information games (Ponsen et al., 2011; Lisý, 2014) (see Appendix A). However, for the adaptive approach to strategic decision making (see Section 1.1.1) convergence to Nash equilibria is not required. In this case, the problem reduces to a Markov decision process (MDP) (technically, a single-agent domain) for which reinforcement learning offers convergence guarantees (Sutton and Barto, 1998). Regarding learning from self-play, Leslie and Collins (2006) introduced a reinforcement learning algorithm that implements (approximate) fictitious play in single-step games and thus offers its convergence guarantees.

1.3 Approach

This section describes the recurring theme of this thesis and the specific steps we take towards answering our research question.

1.3.1 Theme

We craft a novel approach to learning from self-play, by combining the strengths of reinforcement learning (sequential experiential learning) and fictitious play (convergence on Nash equilibria). This is motivated by a striking similarity between the concepts of fictitious play and reinforcement learning. Both techniques consider agents who choose policies that are best in hindsight, i.e. according to their experience.

We develop and evaluate our methods in poker games, which have traditionally exemplified the challenges of strategic decision making (Von Neumann and Morgenstern, 1944; Kuhn, 1950; Nash, 1951; Billings et al., 2002; Sandholm, 2010; Rubin and Watson, 2011; Bowling et al., 2015).

1.3.2 Outline

We proceed as follows:

- Chapter 2 reviews the literature and background material that we build upon.
- Chapter 3 presents a case study of introducing the notion of fictitious play into a reinforcement learning algorithm. The resulting approach, Smooth UCT, significantly improves on the original algorithm's convergence in self-play and was successful in the Annual Computer Poker Competition (ACPC) 2014.
- Chapter 4 extends fictitious play to sequential games, preserving its convergence guarantees. The resulting full-width approach, Extensive-Form Fictitious Play (XFP), provides a principled foundation for reinforcement learning from self-play.
- Chapter 5 extends XFP to learning from sampled experience. The resulting approach, Fictitious Self-Play (FSP), utilises standard supervised and reinforcement learning techniques for approximating (sequential) fictitious play.
- Chapter 6 addresses practical issues of FSP by instantiating it with deep learning techniques. The resulting approach, Neural Fictitious Self-Play (NFSP), has enabled agents to experientially learn a near state-of-the-art policy for a large-scale poker game from self-play. This result provides an answer to the research question of the thesis.
- Chapter 7 concludes the thesis.

Chapter 2

Background and Literature Review

2.1 Reinforcement Learning

Reinforcement learning is the science of optimal sequential decision making in the presence of a reward feedback signal. A reinforcement learning agent is not generally presented a desired or optimal action (output) for a given situation (input). Instead it needs to discover desired solutions from its own or others' experience of interacting with its domain. A real-valued reward signal provides the agent with feedback on its choices and experience. A common goal for the agent is to learn an optimal policy that maximises its expected total reward.

2.1.1 Task

A reinforcement learning **agent** exists in an **environment** that it can interact with. At each each **time step** t the learning agent perceives the current **observation** $O_t \in \mathscr{O}$ and then chooses an **action** $A_t \in \mathscr{A}$ from a set of available actions. After performing its action, which generally has an effect on the environment, the agent experiences a **reward** $R_{t+1} \in \mathbb{R}$ and its next observation O_{t+1} of the environment. The sequence of observations, actions and rewards constitutes the agent's **experience**. The agent's **information state** $U_t \in \mathscr{U}$ is a (possibly sufficient) statistic of the agent's past experience. A **policy**, $\pi(a \mid u)$, is a probability distribution over available actions given information states. In a finite, **episodic task**, the agent strives to maximise its **return**[1], $G_t = \sum_{k=t}^{T} R_{k+1}$, by learning an optimal policy from its experience.

A reinforcement learning task has been traditionally formulated as a **Markov decision process (MDP)** (Puterman, 1994; Sutton and Barto, 1998). A MDP consists of a set of Markov states $\mathscr{S}$, a set of actions $\mathscr{A}$ and probability distributions that determine its dynamics. The transition function, $\mathscr{P}_{ss'}^{a} = \mathbb{P}\left(S_{t+1} = s' \,|\, S_t = s, A_t = a\right)$, determines the probability of transitioning to state s' after taking action a in state s. Rewards are emitted from a conditional probability distribution with expected values determined by the reward function, $\mathscr{R}_{ss'}^{a} = \mathbb{E}\left[R_{t+1} \,|\, S_t = s, A_t = a, S_{t+1} = s'\right]$.

A MDP defines a **perfect-information**, stationary environment. In this environment the agent observes the MDP's Markov state, $O_t = S_t$, which is a sufficient statistic of the agent's past experience,

$$\mathbb{P}\left(o_{t+1}, r_{t+1} \,|\, o_1, a_1, \ldots, o_t, a_t\right) = \mathbb{P}\left(o_{t+1}, r_{t+1} \,|\, o_t, a_t\right).$$

In MDP environments we usually do not distinguish between the environment state and the agent's information state and observation, as we assume them to be equal,

$$S_t = U_t = O_t.$$

A **partially observable Markov decision process (POMDP)** (Kaelbling et al., 1998) extends the MDP model to environments that are not fully observable. In addition to the components of an MDP, a POMDP contains a set of observations $\mathscr{O}$ and an observation function $\mathscr{Z}_{so} = \mathbb{P}\left(O_t = o \,|\, S_t = s\right)$ that determines the probability of observing o in state s.

A POMDP defines an **imperfect-information**, stationary environment. In this environment the agent only receives a partial observation of the underlying MDP's state. In principle, the agent can remember the full history of past observations and

[1] In this thesis, we restrict our presentation and use of reinforcement learning to episodic tasks and thus ignore the common discount factor that is convenient for non-episodic tasks.

actions, which by definition is a sufficient statistic of its experience and thus an ideal information state,

$$U_t = O_1 A_1 O_2 ... A_{t-1} O_t. \tag{2.1}$$

The POMDP can be reduced to a standard, perfect-information MDP by redefining it over these full-history information states and extending the respective transition and reward functions (Silver and Veness, 2010).

2.1.2 Value Functions

Many reinforcement learning methods learn to predict the expected return of the agent (Sutton and Barto, 1998).

The **state-value function**, $V^{\pi} : \mathscr{U} \to \mathbb{R}$, maps an information state u to the expected return from following policy π after experiencing u,

$$V^{\pi}(u) = \mathbb{E}^{\pi}\left[G_t \,|\, U_t = u\right], \tag{2.2}$$

where $\mathbb{E}^{\pi}$ denotes the expectation with the agent's actions, $\{A_k\}_{k \geq t}$, and thus subsequent experience sampled from policy π.

The **action-value function**, $Q^{\pi} : \mathscr{U} \times \mathscr{A} \to \mathbb{R}$, maps information state-action pairs (u, a) to the expected return from taking action a in information state u and following policy π thereafter,

$$Q^{\pi}(u, a) = \mathbb{E}^{\pi}\left[G_t \,|\, U_t = u, A_t = a\right]. \tag{2.3}$$

Full-Width Updates A key ingredient of many reinforcement learning algorithms are the **Bellman equations** (Bellman, 1957). They relate the values of subsequent states. For MDPs, where the agent's information states are equivalent to the MDP's

Markov states, we can derive the Bellman equations,

$$\begin{aligned}
V^{\pi}(u) &= \mathbb{E}^{\pi}\left[\sum_{k=t}^{T} R_{k+1} \,\middle|\, U_t = u\right] \\
&= \mathbb{E}^{\pi}\left[R_{t+1} + \sum_{k=t+1}^{T} R_{k+1} \,\middle|\, U_t = u\right] \\
&= \sum_{a \in \mathscr{A}} \pi(a\,|\,u) \sum_{u' \in \mathscr{U}} \mathscr{P}^{a}_{uu'} \left(\mathscr{R}^{a}_{uu'} + \mathbb{E}^{\pi}\left[\sum_{k=t+1}^{T} R_{k+1} \,\middle|\, U_{t+1} = u'\right]\right) \\
&= \sum_{a \in \mathscr{A}} \pi(a\,|\,u) \sum_{u' \in \mathscr{U}} \mathscr{P}^{a}_{uu'} \left(\mathscr{R}^{a}_{uu'} + V^{\pi}(u')\right).
\end{aligned} \tag{2.4}$$

The Bellman equations define a **full-width update** of a state value from its subsequent state values,

$$V^{\pi}(u) \leftarrow \sum_{a \in \mathscr{A}} \pi(a\,|\,u) \sum_{u' \in \mathscr{U}} \mathscr{P}^{a}_{uu'} \left(\mathscr{R}^{a}_{uu'} + V^{\pi}(u')\right) \tag{2.5}$$

This update requires knowledge or a model of the MDP dynamics.

Similar equations and full-width updates can be derived for the action-value function,

$$Q^{\pi}(u,a) = \sum_{u' \in \mathscr{U}} \mathscr{P}^{a}_{uu'} \left(\mathscr{R}^{a}_{uu'} + \sum_{a \in \mathscr{A}} \pi(a\,|\,u') Q(u',a)\right). \tag{2.6}$$

Sampled Updates Consider the problem of estimating a state or action value from sampled experience, without requiring a model of the environment's dynamics.

Assume an episodic environment and consider experience sampled from following a policy π. Then the sampled return after visiting information state U_t at time t,

$$G_t = \sum_{k=t}^{T} R_{k+1}, \tag{2.7}$$

is an unbiased estimate of the state value, $V^{\pi}(U_t)$. **Monte Carlo** methods accumulate these sampled returns to update the value function

$$V(U_t) \leftarrow V(U_t) + \alpha_t (G_t - V(U_t)), \tag{2.8}$$

where α_t is some scalar step-size that can be time- and state-dependent.

Especially if a state's value depends on many subsequent actions and delayed rewards, sampled returns can have high variance. By *bootstrapping* value function updates from current value estimates, **temporal-difference** methods reduce variance at the cost of introducing bias (Sutton and Barto, 1998).

Considering the Bellman equation,

$$V^{\pi}(U_t) = \mathbb{E}^{\pi}[G_t] = \mathbb{E}^{\pi}[R_{t+1} + V^{\pi}(U_{t+1})], \tag{2.9}$$

we can replace samples of G_t with $R_t + V^{\pi}(U_{t+1})$. **Temporal-difference learning** (Sutton, 1988) uses the current estimate $V(U_{t+1})$ of $V^{\pi}(U_{t+1})$ to perform the following update

$$V(U_t) \leftarrow V(U_t) + \alpha_t (R_t + V(U_{t+1}) - V(U_t)). \tag{2.10}$$

As $V(U_{t+1})$ is an approximation, this update uses a biased estimate of the target value. The difference

$$\delta_t = R_t + V(U_{t+1}) - V(U_t) \tag{2.11}$$

is the **temporal-difference error**. Temporal-difference learning minimises the expected temporal-difference error by incrementally matching values of subsequent states.

2.1.3 Policy Evaluation and Policy Improvement

The preceding section discussed estimating values of single states or state-action pairs. In order to guide sequential decision making in the whole environment, it is useful to estimate the policy's value function for all states. This is known as **policy evaluation** (Sutton and Barto, 1998). After evaluating a policy π, i.e. computing the value function V^{π} or Q^{π}, the policy can be improved by raising its probability of taking actions that are predicted to lead to higher returns. In particular, for an evaluated action-value function, Q^{π}, **greedy policy improvement**

$$\pi'(a|u) = \begin{cases} 1 & \text{for } a = \arg\max_{a'} Q^{\pi}(u,a') \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

is guaranteed to improve the agent's performance, i.e. $V^{\pi'}(u) \geq V^{\pi}(u)\ \forall s \in \mathscr{S}$ (Bellman, 1957). For any MDP there are unique **optimal value functions**,

$$V^*(u) = \max_{\pi} V^{\pi}(u)\ \forall u \in \mathscr{U}, \tag{2.13}$$

$$Q^*(u,a) = \max_{\pi} Q^{\pi}(u,a)\ \forall (u,a) \in \mathscr{U} \times \mathscr{A}, \tag{2.14}$$

that cannot be further improved (Sutton and Barto, 1998). An **optimal policy**, π^*, is any policy that only takes actions of maximum value,

$$\pi^*(a\,|\,u) > 0 \rightarrow Q^*(u,a) = \max_{a'} Q^*(u,a'), \tag{2.15}$$

and thus maximises the agent's expected total reward.

Policy iteration (Bellman, 1957; Puterman and Shin, 1978) alternates policy evaluation with policy improvement. In particular, for a known model of the MDP we can evaluate a policy by repeatedly applying the full-width update (2.5) to all states. Under appropriate conditions, the Banach fixed-point theorem ensures convergence of this procedure to the value function of the evaluated policy. Alternating this policy evaluation procedure with greedy policy improvement converges to the optimal value function (Puterman and Shin, 1978). In general, we do not need to wait for policy evaluation to converge before improving the policy. Value iteration,

$$V^{k+1}(u) \leftarrow \max_{a \in \mathscr{A}} \sum_{u' \in \mathscr{U}} \mathscr{P}^{a}_{uu'} \left(\mathscr{R}^{a}_{uu'} + V^{k}(u') \right) \quad \forall u \in \mathscr{U}, \tag{2.16}$$

also converges to the optimal value function (under appropriate conditions) and performs both (approximate) policy evaluation and greedy policy improvement in one step (Puterman and Shin, 1978). In an episodic MDP that can be represented by a finite tree, there is no need to iteratively apply the value iteration algorithm. If the computation is started from the leaves of the tree, one pass over all values of the tree is sufficient to compute the optimal value function. Value iteration is one example of **Generalised Policy Iteration (GPI)**, a fundamental principle that underlies many reinforcement learning methods (Sutton and Barto, 1998). GPI describes the concept of interleaving potentially partial policy evaluation with policy improvement.

Whereas (full-width) policy and value iteration update all states of the environment at each iteration, sample-based methods selectively apply their updates to states that were experienced by an agent interacting with the environment. **On-policy** agents evaluate the same policy that they use for sampling their experience. In the **off-policy** setting an agent evaluates its policy from experience that was sampled from a different policy, possibly even by a different agent. **Q-learning** (Watkins, 1989) is a popular off-policy temporal-difference learning algorithm. It evaluates the greedy policy by temporal-difference learning (2.10) from sampled experience tuples, $(U_t, A_t, R_{t+1}, U_{t+1})$,

$$Q(U_t, A_t) \leftarrow R_{t+1} + \max_{a \in \mathscr{A}} Q(U_{t+1}, a).$$

The experience can be sampled from any policy without biasing the update. This is because both random variables, U_{t+1}, R_{t+1}, of the update are determined by the environment. A common approach is to sample experience from an exploratory **ε-greedy policy**, which chooses a random action with probability ε and the action of highest estimated value otherwise. This ensures that all states are visited and evaluated, enabling Q-learning to converge to the optimal policy (Watkins and Dayan, 1992). Vanilla Q-learning updates the value function after each transition in the environment in an **online** fashion. Fitted Q Iteration (FQI) (Ernst et al., 2005) is a **batch** reinforcement learning (Lange et al., 2012) method that applies Q-learning to past, memorized experience (Lin, 1992).

2.1.4 Function Approximation

In domains with a very large or continuous state space, learning a separate value for each state can be unfeasible. However, there might be similarities between states leading to similar values. Generalising learned values across states could increase learning performance.

One way to achieve generalisation is to approximate the value function by a simpler function of the agent's information state. In order to define such functions, we need to encode the agent's state appropriately. A common choice is to represent the information state u as a vector of features, $\phi(u) \in \mathbb{R}^n$. Examples of features include the agent's sensor reading, e.g. the pixels it observes on a screen (Mnih et al., 2015), or a binary k-of-n encoding of the cards it is holding in a card game.

A **linear function approximator** represents the value of a state as a linear combination

$$V(u;\theta) = \theta^T \phi(u), \tag{2.17}$$

where $\theta \in \mathbb{R}^n$ is a learned weight vector. **State aggregation** is a special linear function approximator that encodes each state as a unit vector with exactly one non-zero element. **Table lookup** is a special case of state aggregation, that encodes each state uniquely and thus represents the value of each state separately. The representational power of a linear function approximator is limited by the quality of the features. A **multi-layer perceptron** is a common **feedforward neural network** (Bengio, 2009; Schmidhuber, 2015) and a universal function approximator capable of approximating any continuous function on a compact subset of $\mathbb{R}^n$ (Cybenko, 1989). It performs successive stages (layers) of computation. Each layer k is parameterized by a weight matrix W^k and a bias vector b^k and transforms the previous layer's output x^{k-1} to an output vector

$$x^k = \rho\left(W^k x^{k-1} + b^k\right),$$

where ρ is a usually non-linear activation function, e.g. rectified linear $\rho(x) = \max(0,x)$. Each row of this computation, $x_i^k = \rho\left(W_i^k x^{k-1} + b_i^k\right)$, models an **artifical**

neuron. A rectified linear unit (ReLU) is an artificial neuron activated by a rectified linear function. When approximating a value function, the neural network would map the agent's feature vector, $x_0 = \phi(u)$, to the approximated value of the state, $V(u;\theta) = x^n$, where x^n is the final layer's output and $\theta = \{W^1,...,W^n,b^1,...,b^n\}$ is the set of network parameters. In particular,

$$V(u;\theta) = W^2\rho\left(W^1\phi(u)+b^1\right)+b^2, \tag{2.18}$$

specifies a single-layer perceptron approximation of a value function.

In principle, we could approximately evaluate a policy by **Stochastic Gradient Descent (SGD)** on the mean squared error objective

$$\mathscr{L}^\pi(\theta) = \mathbb{E}^\pi\left[\left(V^\pi(U_t) - V(U_t;\theta)\right)^2\right]. \tag{2.19}$$

This is a **supervised learning** formulation of policy evaluation that requires knowledge of the target value function (labels), V^π. In practice, we substitute these optimal targets with sampled or biased targets,

$$\hat{V}_t = \begin{cases} \sum_{a\in\mathscr{A}} \pi(a\,|\,U_t) \sum_{u\in\mathscr{U}} \mathscr{P}^a_{U_t u}\left(\mathscr{R}^a_{U_t u} + V(u;\theta)\right) & \text{(DP)} \\ G_t & \text{(MC)} \\ R_{t+1} + V(U_{t+1};\theta) & \text{(TD)} \end{cases} \tag{2.20}$$

Temporal-difference learning (Sutton, 1988) stochastically optimises the squared temporal-difference error (compare Equation 2.11) with sampled gradient updates,

$$\begin{aligned} \theta &\leftarrow \theta + \alpha_t\left(\hat{V}_t - V(U_t;\theta)\right)\nabla_\theta V(U_t;\theta) \\ &= \theta + \alpha_t\left(R_{t+1} + V(U_{t+1};\theta) - V(U_t;\theta)\right)\nabla_\theta V(U_t;\theta). \end{aligned}$$

On-policy linear temporal-difference learning (Sutton, 1988, 1996) uses linear function approximation and is guaranteed to converge to a bounded region of the desired value function (Gordon, 2001). On the other hand, off-policy and non-linear temporal-difference learning were shown to diverge in counter examples (Baird et al., 1995; Tsitsiklis and Van Roy, 1997). In practice, temporal-difference learning with neural network function approximation was incredibly successful in a variety of applications (Tesauro, 1992; Lin, 1992; Mnih et al., 2015). Neural Fitted Q Iteration (NFQ) (Riedmiller, 2005) and Deep Q Network (DQN) (Mnih et al., 2015) are extensions of FQI that use neural network function approximation with batch and online (fitted) Q-learning updates respectively. These approaches are considered to stabilise reinforcement learning with neural-network function approximation. In particular, they break the correlation of online updates through experience replay (Lin, 1992; Riedmiller, 2005; Mnih et al., 2015). Furthermore, they reduce non-stationarity through fitted target (see Equation 2.20) network parameters (Riedmiller, 2005; Mnih et al., 2015).

2.1.5 Exploration and Exploitation

In order to fully evaluate its policy an agents needs to explore all states and actions in the environment. On the other hand, in order to perform well the agent must choose actions of highest (estimated) value. The agent faces the following dilemma. While **exploitation** of current knowledge, e.g. value function, achieves higher short-term reward, **exploration** of the environment acquires new knowledge that could result in more reward in the long term.

A simple approach for trading off exploitation against exploration is the **ε-greedy policy**, which chooses a random action with probability ε and the action of highest estimated value otherwise,

$$\varepsilon\text{-greedy}\,[Q]\,(a\,|\,u) = \begin{cases} 1-\varepsilon+\frac{\varepsilon}{|\mathscr{A}(u)|}, & a = \arg\max_{a'\in\mathscr{A}(u)} Q(u,a) \\ \frac{\varepsilon}{|\mathscr{A}(u)|}, & a \neq \arg\max_{a'\in\mathscr{A}(u)} Q(u,a) \end{cases}$$

A more sophisticated approach, that uses the scale of the estimated action values to inform exploration, is the **Boltzmann policy**,

$$\text{Boltzmann}\,[Q]\,(a\,|\,u) = \frac{\exp\frac{Q(u,a)}{\tau}}{\sum_{a'\in\mathscr{A}}\exp\frac{Q(u,a')}{\tau}},$$

where the **temperature** τ controls the randomness of the policy. In practice, the ε-greedy and Boltzmann policies' exploration parameters, ε and τ respectively, are reduced over time to focus the policy on the highest-value actions. This idea is related to the principle of **optimism in the face of uncertainty**, which prescribes adjusting exploration of actions with the degree of uncertainty of their values. In a stationary environment, values become more certain over time and therefore exploration can be reduced. The Upper Confidence Bounds (UCB) algorithm (Auer et al., 2002) is a principled approach to optimism in the face of uncertainty for (non-adversarial) multi-armed bandit problems (Lai and Robbins, 1985), i.e. single-state MDPs. UCB applies exploration bonuses, derived from confidence bounds, to actions in proportion to their uncertainty, and maximises the resulting values,

$$a_t = \arg\max_a Q_t(a) + \sqrt{\frac{2\log N_t}{N_t(a)}}, \tag{2.21}$$

where $N_t(a)$ counts the number of times action a has been chosen, $N_t = t$ is the total number of plays and $Q_t(a)$ is the mean reward from choosing action a.

2.1.6 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) (Coulom, 2006; Kocsis and Szepesvári, 2006; Browne et al., 2012) is a class of simulation-based search algorithms, many of which apply Monte Carlo reinforcement learning to local search trees (Silver, 2009). The following properties render MCTS applicable to large-scale environments. By planning online, it can focus its search effort on a relevant (local) subset of states. Instead of a full-width evaluation of the local subtree, as e.g. in Minimax search, MCTS selectively samples trajectories of the game and thus mitigates the *curse of dimensionality*. These simulations are guided by an action-selection policy that explores the most promising regions of the state space. The search tree is incrementally expanded across explored states, which produces a compact, asymmetric search tree.

A MCTS algorithm requires the following components. Given a state and action, a **black box simulator** of the game samples a successor state and reward. A

reinforcement learning algorithm uses simulated trajectories and outcomes (returns) to update statistics at visited nodes in the search tree. A **tree policy** chooses actions based on a node's statistics. A **rollout policy** determines default behaviour for states that are out of the scope of the search tree.

For a specified amount of planning time the algorithm repeats the following. It starts each Monte Carlo simulation at the root node and follows its tree policy until either reaching a terminal state or the boundary of the search tree. Leaving the scope of the search tree, the rollout policy is used to play out the simulation until reaching a terminal state. In this case, the tree is expanded by a state node where the simulation left the tree. This approach selectively grows the tree in areas that are frequently encountered in simulations. After reaching a terminal state, the rewards are propagated back so that each visited state node can update its statistics.

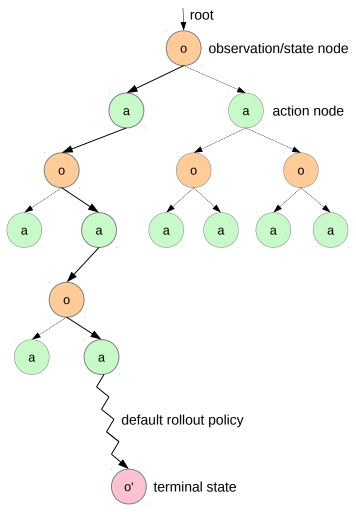


Figure 2.1: MCTS schematic

Common MCTS keeps track of the following node values. $N(u)$ is the number of visits by a Monte-Carlo simulation to node u. $N(u,a)$ counts the number of times action a has been chosen at node u. $Q(u,a)$ is the estimated value of choosing action a at node u. The action value estimates are usually updated by Monte Carlo evaluation,

$$Q(u,a) = \frac{1}{N(u,a)} \sum_{k=1}^{N(u,a)} G(u,k),$$

where $G(u,k)$ is the k-th return experienced from state u.

Kocsis and Szepesvári (2006) suggested using UCB to select actions in the Monte Carlo search tree. The resulting MCTS method, UCB Applied to Trees (UCT), selects greedily between action values that have been enhanced by an exploration bonus,

$$\pi_{tree}(s) = \arg\max_{a} Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}. \tag{2.22}$$

The exploration bonus parameter c adjusts the balance between exploration and exploitation. For suitable c, UCT converges to optimal policies in Markovian environments (Kocsis and Szepesvári, 2006). The appropriate scale of c depends on the size of the rewards.

Other MCTS methods have been proposed in the literature. Silver and Veness (2010) adapt MCTS to POMDPs and prove convergence given a true initial belief state. Auger (2011) and Cowling et al. (2012) extend MCTS to imperfect-information games and use the regret-minimising bandit method EXP3 (Auer et al., 1995) for action selection. Lisy et al. (2013) prove convergence of MCTS in simultaneous-move games for regret-minimising tree policies, including the EXP3 variant. Cowling et al. (2015) propose MCTS methods for inference in multi-player imperfect-information games. Lisý et al. (2015) introduce Online Outcome Sampling, an online MCTS variant that is guaranteed to converge in two-player zero-sum imperfect-information games. Ponsen et al. (2011) compared the full-game search performance of Outcome Sampling and UCT in a poker game. They concluded that UCT quickly finds a good but suboptimal policy, while Outcome Sampling initially learns more slowly but converges to the optimal policy over time.

2.2 Game Theory

Game theory is the science of strategic decision making in multi-agent scenarios.

2.2.1 Extensive-Form Games

Extensive-form games (Kuhn, 1953; Myerson, 1991) are a model of multiple players' sequential interaction. The representation is based on a game tree and consists of the following components: $\mathscr{N} = \{1, ..., n\}$ denotes the set of players. $\mathscr{S}$ is a set of **states** corresponding to nodes in a finite rooted game tree. For each state node $s \in \mathscr{S}$ the edges to its successor states define a set of **actions** $\mathscr{A}(s)$ available to a player or chance in state s. The **player function** $P : \mathscr{S} \to \mathscr{N} \cup \{c\}$, with c denoting chance, determines who is to act at a given state. **Chance** may be considered a particular player that follows a fixed randomized strategy that determines the distribution of chance events at chance nodes. For each player i there is a corresponding set of **information states** $\mathscr{U}^i$ and an **information function** $I^i : \mathscr{S} \to \mathscr{U}^i$ that determines which states are indistinguishable for the player by mapping them onto the same information state $u \in \mathscr{U}^i$. In this dissertation, we mostly assume games with **perfect recall**, i.e. each player's current information state u^i_t implies knowledge of the sequence of his information states and actions, $u^i_1, a^i_1, u^i_2, a^i_2, ..., u^i_t$, that led to this information state. The **return function** $R : \mathscr{S} \to \mathbb{R}^n$ maps terminal states to a vector whose components correspond to each player's return. A game is **zero-sum** if and only if the sum of all players' returns is always zero, i.e. $\sum_{i \in \mathscr{N}} R^i(s) = 0$ for all terminal states s.

By composing (surjective) functions, $f^i_A : \mathscr{S}^i \to \tilde{\mathscr{S}}^i \subseteq \mathscr{S}^i$, with the game's information functions, I^i, we obtain alternative information functions, $\tilde{I}^i = f^i_A \circ I^i$, that induce an (information-state) **abstraction** of the extensive-form game. This is conceptually equivalent to introducing state-aggregating function approximators (Section 2.1.4) for each player.

A player's **behavioural strategy**, $\pi^i(a \,|\, u^i)$, is a probability distribution over actions given information states, and Σ^i is the **set of all behavioural strategies** of player i. A **strategy profile** $\pi = (\pi^1, ..., \pi^n)$ is a collection of strategy for all players. π^{-i} refers to all strategies in π except π^i.

2.2.2 Nash Equilibria

By extension of the return function R, $R^i(\pi)$ is the expected return of player i if all players follow the strategy profile π. For $\varepsilon \geq 0$, the set of **ε-best responses** of player i to their opponents' strategies π^{-i},

$$\mathrm{BR}^i_\varepsilon(\pi^{-i}) = \left\{ \pi^i \in \Sigma^i : R^i(\pi^i, \pi^{-i}) \geq \max_{\pi' \in \Sigma^i} R^i(\pi', \pi^{-i}) - \varepsilon \right\}, \tag{2.23}$$

contains all strategies which achieve an expected return against π^{-i} that is suboptimal by no more than ε. The set of **best responses**, $\mathrm{BR}^i(\pi^{-i}) = \mathrm{BR}^i_0(\pi^{-i})$, contains all optimal strategies against π^{-i}. A **Nash equilibrium** (Nash, 1951) of an extensive-form game is a strategy profile π such that $\pi^i \in \mathrm{BR}^i(\pi^{-i})$ for all $i \in \mathcal{N}$, i.e. a strategy profile from which no return-maximising player would choose to deviate. An **ε-Nash equilibrium** is a strategy profile π such that $\pi^i \in \mathrm{BR}^i_\varepsilon(\pi^{-i})$ for all $i \in \mathcal{N}$.

For a two-player zero-sum game, we say that a strategy profile π is **exploitable** by an amount ε if and only if

$$\varepsilon = \frac{R^1\left(\mathrm{BR}^1(\pi^2), \pi^2\right) + R^2\left(\pi^1, \mathrm{BR}^2(\pi^1)\right)}{2}.$$

Exploitability measures how well a worst-case opponent would perform by best responding to the strategy profile. In two-player zero-sum games Nash equilibria have attractive worst-case guarantees.

Theorem 2.2.1 (Minimax (Neumann, 1928))**.** *For any two-player zero-sum game, there is a value $v \in \mathbb{R}$ such that*

$$\max_{\pi^1 \in \Sigma^1} \min_{\pi^2 \in \Sigma^2} R^1(\pi^1, \pi^2) = \min_{\pi^2 \in \Sigma^2} \max_{\pi^1 \in \Sigma^1} R^1(\pi^1, \pi^2) = v$$

Thus, playing a Nash equilibrium strategy guarantees player 1 (player 2) at least the game's unique value v ($-v$). Hence, a Nash equilibrium is unexploitable, i.e. has zero exploitability. In general-sum and multi-player games different Nash equilibria can result in different expected returns.

2.2.3 Normal Form

An extensive-form game induces an equivalent **normal-form game** as follows (Kuhn, 1953; Myerson, 1991). For each player $i \in \mathcal{N}$ their deterministic behavioural strategies, $\Delta^i_p \subset \Sigma^i$, define a set of normal-form actions, called **pure strategies**. Restricting the extensive-form return function R to pure strategy profiles yields an (expected) return function for the normal-form game.

Each pure strategy can be interpreted as a full-game plan that specifies deterministic actions for all situations that a player might encounter. Before playing an iteration of the game each player chooses one of their available plans and commits to it for the iteration. A **mixed strategy** Π^i for player i is a probability distribution over their pure strategies. Let Δ^i denote the **set of all mixed strategies** available to player i. A **mixed strategy profile** $\Pi \in \times_{i \in \mathcal{N}} \Delta^i$ specifies a mixed strategy for each player. Finally, $R^i(\Pi)$ determines the expected return of player i for the mixed strategy profile Π.

Throughout this dissertation, we use small Greek letters for extensive-form, behavioural strategies and large Greek letters for pure and mixed strategies of a game's normal form.

2.2.4 Sequence Form

The sequence form (Koller et al., 1994; Von Stengel, 1996) is a compact representation of an extensive-form game, which is well-suited for computing Nash equilibria of two-player zero-sum games by linear programming (Koller et al., 1996). It decomposes players' strategies into sequences of actions and probabilities of realizing these sequences.

For any player $i \in \mathcal{N}$ of a perfect-recall extensive-form game, each of their information states $u^i \in \mathcal{U}^i$ uniquely defines a **sequence** σ_{u^i} of actions that the player has to take in order to reach information state u^i. Let $\sigma_u a$ denote the sequence that extends σ_u with action a. A **realization plan**, x, maps each player i's sequences, $\{\sigma_{u^i}\}_{u^i \in \mathcal{U}^i}$, to **realization probabilities**, such that $x(\emptyset) = 1$ and $x(\sigma_{u^i}) = \sum_{a \in \mathcal{A}(u^i)} x(\sigma_{u^i} a) \forall s^i \in \mathcal{U}^i$. A behavioural strategy π induces a realization plan

$$x_\pi(\sigma_u) = \prod_{(u',a) \in \sigma_u} \pi(a \mid u'),$$

where the notation (u', a) disambiguates actions taken at different information states. Similarly, a realization plan induces a behavioural strategy,

$$\pi(a \mid u) = \frac{x(\sigma_u a)}{x(\sigma_u)}, \quad x(\sigma_u) \neq 0,$$

where π is defined arbitrarily at information states that are never visited, i.e. when $x(\sigma_u) = 0$. As a pure strategy is just a deterministic behavioural strategy, it has a realization plan with binary values. As a mixed strategy is a convex combination of pure strategies, $\Pi = \sum_i w_i \Pi_i$, its realization plan is a similarly weighted convex combination of the pure strategies' realization plans, $x_\Pi = \sum_i w_i x_{\Pi_i}$.

Two strategies π_1 and π_2 of a player are **realization equivalent** (Von Stengel, 1996) if and only if for any fixed strategy profile of the other players both strategies, π_1 and π_2, produce the same probability distribution over the states of the game. As a realization plan defines probabilities of strategies realizing states, we have the following theorem.

Theorem 2.2.2 (Von Stengel, 1996)**.** *Two strategies are realization equivalent if and only if they have the same realization plan.*

Kuhn's Theorem (Kuhn, 1953) links extensive-form, behavioural strategies with (realization-equivalent) normal-form, mixed strategies.

Theorem 2.2.3 (Kuhn, 1953)**.** *For a player with perfect recall, any mixed strategy is realization equivalent to a behavioural strategy, and vice versa.*

2.2.5 Fictitious Play

Fictitious play (Brown, 1951) is a classic game-theoretic model of learning from self-play. Fictitious players repeatedly play a game, at each iteration choosing a best response to their opponents' empirical, average strategies. The average strategies of fictitious players converge to Nash equilibria in certain classes of games, e.g. two-player zero-sum and many-player potential games (Robinson, 1951; Monderer and Shapley, 1996). Fictitious play is a standard tool of game theory and has motivated substantial discussion and research on how Nash equilibria could be realized in practice (Brown, 1951; Fudenberg and Levine, 1995; Fudenberg, 1998; Hofbauer and Sandholm, 2002). Leslie and Collins (2006) introduced generalised weakened fictitious play. It has similar convergence guarantees as common fictitious play, but allows for approximate best responses and perturbed average strategies, making it particularly suitable for machine learning.

Definition 2.2.4 (compare Definition 3 (Leslie and Collins, 2006))**.** A generalised weakened **fictitious play** is a sequence of mixed strategies, $\{\Pi_k\}$, $\Pi_k \in \times_{i \in \mathcal{N}} \Delta^i$, s.t.

$$\Pi^i_{k+1} \in (1-\alpha_{k+1})\Pi^i_k + \alpha_{k+1}(\mathrm{BR}^i_{\varepsilon_k}(\Pi^{-i}_k) + M^i_{k+1}), \quad \forall i \in \mathcal{N},$$

with $\alpha_k \to 0$ and $\varepsilon_k \to 0$ as $k \to \infty$, $\sum_{k=1}^{\infty} \alpha_k = \infty$, and $\{M^i_k\}$ sequences of perturbations that satisfy, for any $T > 0$,

$$\lim_{k\to\infty} \sup_l \left\{ \left\| \sum_{j=k}^{l-1} \alpha_{j+1} M^i_{j+1} \right\| \text{ s.t.} \sum_{j=k}^{l-1} \alpha_{j+1} \leq T \right\} = 0, \quad \forall i \in \mathcal{N}.$$

An appropriate choice of perturbations is e.g. M_t a martingale that is uniformly bounded in L^2 and $\alpha_k = \frac{1}{k}$ (Benaïm et al., 2005). Original fictitious play Brown (1951) is a generalised weakened fictitious play with stepsize $\alpha_k = \frac{1}{k}$, $\varepsilon_k = 0$ and $M_k = 0\ \forall k$. Cautious or smooth fictitious play (Fudenberg and Levine, 1995) uses ε-best responses defined by a Boltzmann distribution over the action values.

Leslie and Collins (2006) proposed a self-play reinforcement learning method for normal-form games that follows a generalised weakened fictitious play. In particular, they simulated two agents that play according to their average fictitious-play strategies and evaluate action values from the outcomes of these simulations

$$Q^i_k\left(a^i\right) \approx R^i\left(a^i, \Pi^{-i}_{k-1}\right)$$

Each agent updates its average fictitious play policy by moving it towards a best response in form of a Boltzmann distribution over its Q-value estimates.

Hendon et al. (1996) introduced two definitions of fictitious play in extensive-form games, based on sequential and local best responses respectively. They showed that each convergence point of these variants is a sequential equilibrium, but convergence in imperfect-information games could not be guaranteed.

2.2.6 Best Response Computation

This section describes the computation of best responses in extensive-form games. Consider a player i of an extensive-form game and let π^{-i} be a strategy profile of fellow agents. This strategy profile defines stationary stochastic transitions at all states that are not controlled by player i. Thus, the game becomes a single-agent (information-state) MDP. Computing an optimal policy of such an MDP, e.g. by dynamic programming, produces a best response, $\mathrm{BR}^i(\pi^{-i})$, to the fellow agents' strategy profile, π^{-i}. Johanson et al. (2011) discuss efficient techniques for computing best responses in extensive-form poker games.

We present dynamic-programming values and recursions for computing best responses in extensive-form games. At each player's terminal information state, $\bar{u}^i$, we compute his value as a weighted sum over his returns in all states that belong to his information set weighted by the realization probabilities of his fellow agents and chance. In particular,

$$v(\bar{u}^i) = \sum_{s \in I^{-1}(\bar{u}^i)} x_{\pi^{-i}}(\sigma_s) R^i(s), \tag{2.24}$$

where $x_{\pi^{-i}}(\sigma_s)$ is the product of fellow agents' and chance's realization probabilities based on their respective imperfect-information views of state s. These computed values are recursively propagated back. In particular, each information state's value and best-response action can be recursively defined,

$$\begin{aligned} v(u^i) &= \max_{a \in \mathscr{A}(u^i)} \sum_{u \in \mathscr{U}^i} \delta^a_{u^i u} v(u), \\ \beta^i(u^i) &= \arg\max_{a \in \mathscr{A}(u^i)} \sum_{u \in \mathscr{U}^i} \delta^a_{u^i u} v(u), \end{aligned} \tag{2.25}$$

where $\delta^a_{u^i u} \in \{0, 1\}$ indicates whether the player can transition (in one step) to information state u after taking action a in information state u^i. Furthermore, the sum of each player's values at the game's root produces the expected reward his best response achieves. This can be used for computing the exploitability of a strategy profile.

2.3 Poker

Poker has traditionally exemplified the challenges of strategic decision making (Von Neumann and Morgenstern, 1944; Kuhn, 1950; Nash, 1951). It has also become a focus of research in computational game theory (Billings et al., 2002; Sandholm, 2010; Rubin and Watson, 2011; Bowling et al., 2015).

In addition to its large number of game states, it features elements of imperfect information and uncertainty. These properties are found in many real-world problems that require strategic decision making, e.g. security, trading and negotiations. Unlike complex real-world problems, however, poker games have clear rules and structure. This enables rapid simulation and algorithm development. Thus, it is an excellent domain for furthering methods of multi-agent artifical intelligence, reinforcement learning and game theory.

In this section we introduce the rules of the poker games that are used in our experiments. In addition, we give an overview of common properties of poker games and discuss why these are challenging for self-play reinforcement learning. Finally, we review algorithmic approaches that have been successfully applied to computer poker.

2.3.1 Rules

There exist many different poker games (Sklansky, 1999). We restrict our presentation to (Limit) Texas Hold'em, Leduc Hold'em and Kuhn poker. Texas Hold'em is the most popular poker game among humans and has also been a main challenge in algorithmic game-theory (Billings et al., 2002; Sandholm, 2010). Leduc Hold'em and Kuhn poker are smaller poker variants that are well-suited for experiments.

Limit Texas Hold'em is played with a 52 card deck that contains 13 card ranks and 4 suits. Each episode (*hand*) of the game consists of up to four betting rounds. At the beginning of the first round, called *preflop*, each player is dealt two private cards. In the second round, the *flop*, three public community cards are revealed. A fourth and fifth community card are revealed in the last two rounds, the *turn* and *river*, respectively. Figure 2.2 shows an exemplary situation of a *flop* in two-player Texas Hold'em.

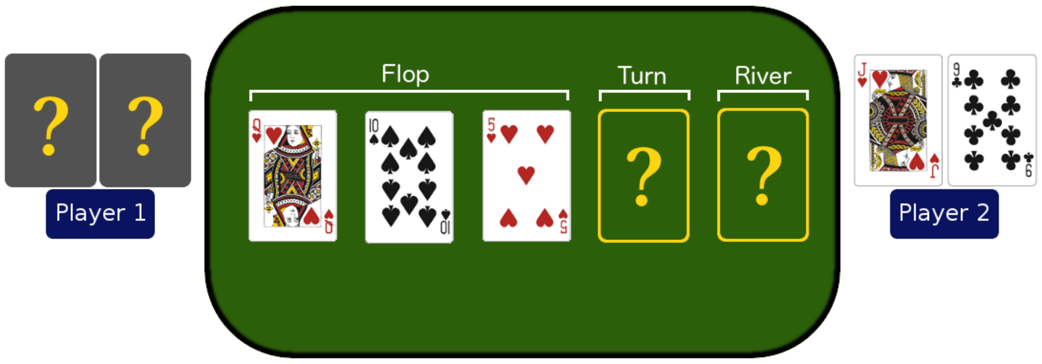


Figure 2.2: Exemplary Texas Hold'em game situation

Players have the following options when it is their turn. A player can fold and forfeit any chips that he has bet in the current episode. After folding a player is not allowed to participate any further until the beginning of the next episode of the game. A player can call by matching the highest bet that has been placed at the current betting round. If no bet has been placed at the current round then this action is called a check. A player can raise by betting the round's currently highest bet increased by a fixed amount, i.e. bet size. The bet size is two units preflop and on the flop and four units on the turn and river, called small and big bet respectively.

A marker, called the button, defines the positions of players. After each episode of the game it is moved by one position clockwise. At the beginning of the game the first and second player after the button are required to place forced bets of one and two units respectively, called the small and big blind. Afterwards, the first betting round commences with the player following the big blind being the first to act. On the flop, turn and river the first remaining player after the button is the first to act. The total number of bets or raises per round is limited (*capped*). In particular, players are only allowed to call or fold once the current bet has reached four times the round's bet size. The game transitions to the next betting round once all remaining players have matched the current bet. However, each remaining player is guaranteed to act at least once at each round.

An episode ends in one of two ways. If there is only one player remaining, i.e. all other players folded, then he wins the whole pot. If at least two players reach the end of the river then the game progresses to the showdown stage. At the showdown each remaining player forms the best five card combination from his private cards and the public community cards. There are specific rules and rankings of card combinations (Sklansky, 1999), e.g. three of a kind beats a pair. The player with the best card combination wins the pot. There is also the possibility of a split if several players have the same combination.

No-Limit Texas Hold'em has almost the same rules as the Limit variant. The main difference is that players do not need to bet in fixed increments but are allowed to relatively freely choose their bet sizes including betting all their chips, known as moving all-in.

Leduc Hold'em (Southey et al., 2005) is a small variant of Texas Hold'em. There are three card ranks, J, Q and K, with two of each in the six card deck. There are two betting rounds, preflop and the flop. Each player is dealt one private card preflop. On the flop one community card is revealed. Instead of blinds, each player is required to post a one unit ante preflop, i.e. forced contribution to the pot. The bet sizes are two units preflop and four units on the flop with a cap of twice the bet size at each round.

Kuhn poker (Kuhn, 1950) is a simpler variant than Leduc Hold'em. There are 3 cards, a J, Q, and K, in the deck. Each player is dealt one private card and there is only one round of betting without any public community cards. Each player is required to contribute a one unit ante to the pot. The bet size is one unit and betting is capped at one unit.

2.3.2 Properties

Poker presents a variety of difficulties to learning algorithms.

Imperfect Information In imperfect-information games players only observe their information state, U_t^i, and generally do not know which exact game state, S_t, they are in. They might form beliefs, $\mathbb{P}\left(S_t \mid U_t^i\right)$, which are generally affected by fellow players' strategies at preceding states, S_k, $k < t$. This implicit connection to prior states renders common policy iteration (Section 2.1.3) and Minimax search techniques ineffective in imperfect-information games (compare Appendix A) (Frank and Basin, 1998). This is problematic as many reinforcement learning algorithms are related to (generalised) policy iteration (Sutton and Barto, 1998). It also poses a problem to local search which has been highly successful in large perfect-information games, e.g. computer Go (Silver et al., 2016a). Given a subgame with perfect player beliefs based on a full-game Nash equilibrium, resolving the subgame does not generally recover a Nash equilibrium (Burch et al., 2014; Ganzfried and Sandholm, 2015; Lisý et al., 2015). Finally, Nash equilibria of imperfect-information games are generally stochastic. By contrast, many reinforcement learning methods are designed for MDPs, which always have an optimal deterministic strategy. Furthermore, even with a stochastic policy, the standard GPI approach to reinforcement learning may be unsuitable for learning Nash equilibria of imperfect-information games (see Section A.1).

Size A two-player zero-sum game can theoretically be solved by linear programming (Koller et al., 1996). In practice, however, linear programming does not scale to the size of Limit Texas Hold'em with current computational resources. Note that the size of Limit Texas Hold'em, which has about 10^{18} game states, might not appear to be large in comparison to much larger perfect-information games such as chess or Go. However, these game sizes are not directly comparable because local search and other solution techniques, e.g. Minimax search, are not readily applicable to imperfect-information games. In particular, Minimax search has to parse the tree only once to produce a Nash equilibrium, whereas CFR or fictitious play need to repeatedly perform computations at all states. In practice, large imperfect-information games are usually abstracted to a tractable size (Johanson et al., 2013). We discuss common abstraction techniques for poker in the next section.

Multi-Player Almost all poker variants are defined for two and more players. Apart from the escalating size of multi-player games there are more fundamental problems in finding optimal strategies. The worst-case guarantees of the Minimax Theorem 2.2.1 do not apply to multi-player games. Therefore, it is unclear whether and when a Nash equilibrium is actually a desired solution. Furthermore, many game-theoretic approaches do not have any convergence guarantees for even zero-sum multi-player games (Risk and Szafron, 2010). However, in practice these methods have produced competitive policies (Risk and Szafron, 2010) and in some games converge close to a Nash equilibrium (Ganzfried and Sandholm, 2009).

Variable Outcomes Poker is a game of high variance. The private holdings and public community cards are dealt by chance. This might pose a challenge to sample-based evaluation. In addition, instead of a binary outcome as in many board games, the outcome of poker is based on the monetary units that have been wagered. A lot of bets and raises lead to large pots. E.g. in Limit Texas Hold'em (LHE) the outcomes range between -48 and 48. This can pose a challenge for value function approximation, e.g. neural networks.

2.3.3 Abstraction

This section describes common abstraction techniques for Limit Texas Hold'em. In order to reduce the game tree to a tractable size, various hand-crafted abstractions have been proposed in the literature (Billings et al., 2003; Gilpin and Sandholm, 2006; Johanson et al., 2013). A common approach is to group information states by the strategic similarity of the corresponding cards and leave the players' action sequences unabstracted. Various metrics have been proposed for defining card similarity. Hand strength-based metrics use the expected winning percentage of a combination of private and community cards against a random holding (Billings et al., 2003; Gilpin and Sandholm, 2006; Zinkevich et al., 2007). Distribution-aware methods compare probability distributions of winning against a random holding instead of just the expected value (Johanson et al., 2013). A clustering by these metrics produces an abstraction function, f_A, that assigns each card combination to a cluster. Together with the action sequences, the clusters define the abstracted information states.

After learning an approximate Nash equilibrium π in the abstracted game, we can recover a strategy for the real game by composition $\pi \circ f_A$, i.e. mapping the real information states to the abstraction and then selecting an action from the strategy. The hope is that $\pi \circ f_A$ also approximates a Nash equilibrium in the full game and that the quality of the approximation improves with the quality of the abstraction. However, it has been shown that finer abstractions do not always yield better approximations in the real game and can result in unintuitive pathologies (Waugh et al., 2009a).

2.3.4 Current Methods

Efficiently computing Nash equilibria of imperfect-information games has received substantial attention by researchers in computational game theory (Sandholm, 2010; Bowling et al., 2015). Game-theoretic approaches have played a dominant role in furthering algorithmic performance in computer poker. The most popular modern techniques are either optimisation-based (Koller et al., 1996; Gilpin et al., 2007; Hoda et al., 2010; Bosansky et al., 2014) or perform (counterfactual) regret minimisation (Zinkevich et al., 2007; Sandholm, 2010).

Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2007) is a full-width self-play approach guaranteed to converge in sequential adversarial (two-player zero-sum) games. MCCFR (Lanctot et al., 2009) extends CFR to learning from sampled subsets of game situations. OS (Lanctot et al., 2009; Lisý et al., 2015) is an MCCFR variant that samples single game trajectories. It can be regarded as an experiential, reinforcement learning method[2]. However, OS, MCCFR and CFR are all table-lookup approaches that lack the ability to learn abstract patterns and use them to generalise to novel situations, which is essential in large-scale games. An extension of CFR to function approximation (Waugh et al., 2015) is able to learn such patterns but has not been combined with sampling. Therefore, it still needs to enumerate all game situations. CFR-based methods have essentially solved two-player Limit Texas Hold'em poker (Bowling et al., 2015) and produced a champion program for the No-Limit variant (Brown et al., 2015).

[2] We can reinterpret the counterfactual regret as a novel kind of (advantage) value function.

Chapter 3

Smooth UCT Search

In this chapter we address the subquestion,

> Can UCT be utilised for learning approximate Nash equilibria in imperfect-information games?

This is motivated by UCT's success in perfect-information games. Furthermore, as extensive-form games are essentially trees, an MCTS algorithm is a suitable choice for a case study of reinforcement learning in imperfect-information extensive-form games.

3.1 Introduction

MCTS (see Section 2.1.6) has been incredibly successful in perfect-information games (Browne et al., 2012; Gelly et al., 2012; Silver et al., 2016a). Although these methods have been extended to imperfect-information domains (Silver and Veness, 2010; Auger, 2011; Cowling et al., 2012), so far they have not achieved the same level of practical performance (Brown et al., 2015) or theoretical convergence guarantees as competing methods (Lanctot et al., 2009; Lisý et al., 2015). In particular, the prominent UCT algorithm has failed to converge in practice (Ponsen et al., 2011; Lisý, 2014).

In this chapter, we investigate UCT-based MCTS in poker as a case study of reinforcement learning from self-play in imperfect-information games. We focus on convergence in the full-game MCTS setting, which is a prerequisite for online

MCTS. In particular, we introduce Smooth UCT, which combines the notion of fictitious play with MCTS. Fictitious players perform the best response to other players' average behaviour. We introduce this idea into MCTS by letting agents mix in their average strategy with their usual utility-maximising actions. Intuitively, apart from mimicking fictitious play, this might have further potential benefits, e.g. breaking correlations and stabilising self-play learning due to more smoothly changing agent behaviour.

3.2 MCTS in Extensive-Form Games

In perfect-information games as well as single-agent POMDPs, it is sufficient to search a single tree that includes the information states and actions of all players. The asymmetry of players' information in an extensive-form game with imperfect information does not allow for a single, collective search tree. In order to extend the single-agent Partially Observable Monte-Carlo Planning (POMCP) method (Silver and Veness, 2010) to extensive-form games, we build on the idea of using separate search trees that are spanned over the respective players' information states of the extensive-form game (Auger, 2011; Cowling et al., 2012).

In particular, for each player i we grow a tree T^i over their information states $\mathscr{S}^i$. $T^i(s^i)$ is the node in player i's tree that represents their information state s^i. In a game with perfect recall, player i's sequence of previous information states and actions, $s^i_1, a^i_1, s^i_2, a^i_2, ..., s^i_k$, is incorporated in the information state s^i_k and therefore T^i is a proper tree. Imperfect-recall abstractions (Waugh et al., 2009b) can produce recombining trees. Figure 3.1 illustrates the multiple-tree approach.

Algorithm 1 describes a general MCTS approach for the multi-player imperfect-information setting of extensive-form games. The game mechanics are sampled from transition and return simulators $\mathscr{G}$ and $\mathscr{R}$. The transition simulator takes a game state and action as inputs and stochastically generates a successor state $S_{t+1} \sim \mathscr{G}(S_t, A^i_t)$, where the action A^i_t belongs to the player i who makes decisions at state S_t. The return simulator generates all players' returns at terminal states, i.e. $R \sim \mathscr{R}(S_T)$. The extensive-form information function, $I^i(s)$, determines the acting player *i*'s information state. The OUT-OF-TREE property keeps track of which player has left the scope of their search tree in the current episode.

This algorithm can be specified by the action selection and node updating functions, SELECT and UPDATE. These functions are responsible to sample from and update the tree policy. In this work, we focus on UCT-based methods.

3.3 Extensive-Form UCT

Extensive-Form UCT uses UCB to select from and update the tree policy in Algorithm 1. Algorithm 2 presents this instantiation. It can be seen as a multi-agent version of Partially Observable UCT (PO-UCT) (Silver and Veness, 2010). PO-UCT searches a tree over the histories of an agent's observations and actions in a POMDP, whereas extensive-form UCT searches multiple agents' respective trees over their information states and actions. In a perfect-recall game an information state implies knowledge of the preceding sequence of information states and actions and is therefore technically equivalent to a full history (compare Equation 2.1).

3.4 Smooth UCT

Smooth UCT is a MCTS algorithm that selects actions from Smooth UCB, a variant of the multi-armed bandit algorithm UCB.

Smooth UCB is a heuristic modification of UCB, that is inspired by fictitious play. Fictitious players learn to best respond to their fellow players' average strategy profile. By counting how often each action has been taken, UCB already keeps track

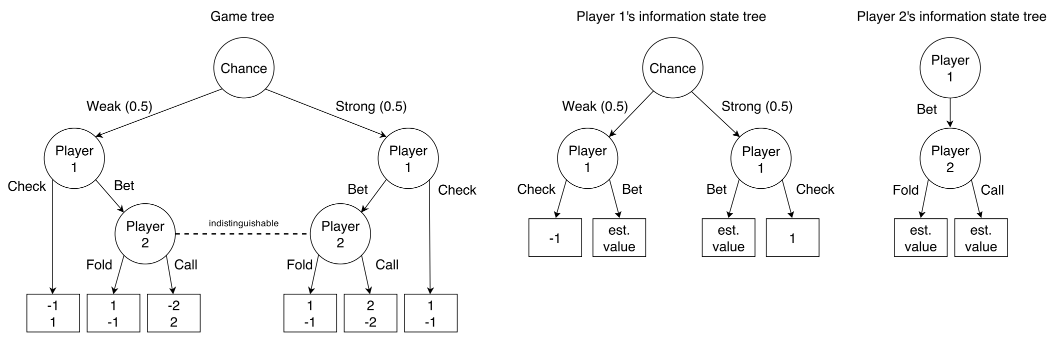


Figure 3.1: The full game tree and the players' individual information-state trees of a simple extensive-form game.

Algorithm 1 Extensive-Form MCTS

```
function SEARCH
    while within computational budget do
        Sample initial game state $s_0$
        SIMULATE($s_0$)
    end while
    return $\pi_{tree}$
end function

function ROLLOUT($s$)
    $a \sim \pi_{rollout}(s)$
    $s' \sim \mathcal{G}(s,a)$
    return SIMULATE($s'$)
end function

function SIMULATE($s$)
    if ISTERMINAL($s$) then
        return $r \sim \mathscr{R}(s)$
    end if
    $i =$ PLAYER$(s)$
    if OUT-OF-TREE($i$) then
        return ROLLOUT($s$)
    end if
    $u^i = I^i(s)$
    if $u^i \notin T^i$ then
        EXPANDTREE($T^i, u^i$)
        $a \sim \pi_{rollout}(s)$
        OUT-OF-TREE$(i) \leftarrow$ true
    else
        $a =$ SELECT$(u^i)$
    end if
    $s' \sim \mathcal{G}(s,a)$
    $r \leftarrow$ SIMULATE$(s')$
    UPDATE$(u^i, a, r^i)$
    return $r$
end function
```

of an average strategy π. It can be readily extracted by setting $\pi_t(a) = \frac{N_t(a)}{N_t}$, where $N_t(a)$ is the number of times action a has been chosen and $N_t = t$ the total number of plays. However, UCB does not use this strategy and therefore potentially ignores some useful information in its search.

The basic idea of Smooth UCB is to mix in the average strategy when selecting actions in order to induce the other agents to respond to it. This resembles the

Algorithm 2 UCT

```
SEARCH($\Gamma$), SIMULATE($s$) and ROLLOUT($s$) as in Algorithm 1

function SELECT($u^i$)
    return $\arg\max_a Q(u^i,a) + c\sqrt{\frac{\log N(u^i)}{N(u^i,a)}}$
end function

function UPDATE($u^i, a, r^i$)
    $N(u^i) \leftarrow N(u^i) + 1$
    $N(u^i,a) \leftarrow N(u^i,a) + 1$
    $Q(u^i,a) \leftarrow Q(u^i,a) + \frac{r^i - Q(u^i,a)}{N(u^i,a)}$
end function
```

idea of fictitious play, where agents are supposed to best respond to the average strategy. The average strategy might have further beneficial properties. Firstly, it is a stochastic strategy and it changes ever more slowly over time. This can decrease correlation between the players' actions and thus help to stabilise the self-play process. Furthermore, a more smoothly changing strategy can be relied upon by other agents and is easier to adapt to than an erratically changing greedy policy like UCB.

Smooth UCB requires the same information as UCB but explicitly makes use of the average strategy via the action counts. In particular, it mixes between UCB and the average strategy with probability η_k, where η_k is a sequence that decays to 0 or a small constant for $k \to \infty$. On the one hand, decaying η_k allows the agents to focus their experience on play against their opponents' average strategies, which fictitious players would want to best respond to. On the other hand, decaying η_k would also slow down exploration and updates to the average strategy. Intuitively, the annealing schedule for η_k has to balance these effects. In this work, we empirically chose

$$\eta_k = \max\left(\gamma, \eta_0\left(1 + d\sqrt{N_k}\right)^{-1}\right), \quad k > 0, \tag{3.1}$$

where N_k is the total number of plays, γ a lower limit on η_k, η_0 an initial value and d a constant that parameterises the rate of decay. Intuitively, the decay by $\sqrt{N_k}$ and the lower bound of γ were chosen to avoid a premature slowdown of exploration and updates to the average strategies. Note that the effective step size of these updates is $\frac{\eta_k}{k}$, due to averaging.

As UCT is obtained from applying UCB at each information state, we similarly define Smooth UCT as an MCTS algorithm that uses Smooth UCB at each information state. Algorithm 3 instantiates extensive-form MCTS with a Smooth UCB tree policy. For a constant $\eta = 1$ (via $\eta_0 = 1, d = 0$), we obtain UCT as a special case. Compared to UCT, the UPDATE operation is left unchanged. Furthermore, for a cheaply determined η the SELECT procedure has little overhead compared to UCT.

Algorithm 3 Smooth UCT

```
SEARCH($\Gamma$), SIMULATE($s$) and ROLLOUT($s$) as in Algorithm 1
UPDATE($u^i, a, r^i$) as in Algorithm 2

function SELECT($u^i$)
    $\eta \leftarrow \max\left(\gamma, \eta_0\left(1 + d\sqrt{N(u^i)}\right)^{-1}\right)$ as in Equation 3.1
    $z \sim U[0,1]$
    if $z < \eta$ then
        return $\arg\max_a Q(u^i, a) + c\sqrt{\frac{\log N(u^i)}{N(u^i,a)}}$
    else
        $\forall a \in A(u^i) : p(a) \leftarrow \frac{N(u^i,a)}{N(u^i)}$
        return $a \sim p$
    end if
end function
```

3.5 Experiments

We evaluated Smooth UCT in the Kuhn, Leduc and Limit Texas Hold'em poker games.

3.5.1 Kuhn Poker

Ponsen et al. (2011) tested UCT against Outcome Sampling in Kuhn poker. We conducted a similar experiment, comparing UCT to Smooth UCT. Learning performance was measured in terms of the average policies' mean squared errors with respect to the closest Nash equilibrium determined from the known parameterization of equilibria in Kuhn poker (Hoehn et al., 2005). Smooth UCT's mixing parameter schedule (3.1) was manually tuned in preliminary experiments on Kuhn poker. In particular, we varied one parameter at a time and measured the resulting achieved exploitability. The best results were achieved with $\gamma = 0.1$, $\eta = 0.9$ and $d = 0.001$. We calibrated the exploration parameters of UCT and Smooth UCT by training 4 times for 10 million episodes each with parameter settings of $c = 0.25k$, $k = 1, ..., 10$. The best average final performance of UCT and Smooth UCT was achieved with 2 and 1.75 respectively. In each main experiment, each algorithm trained for 20 million episodes. The results, shown in figure 3.2, were averaged over 50 repeated runs of the experiment. We see that Smooth UCT approached a Nash equilibrium, whereas UCT exhibited divergent performance.

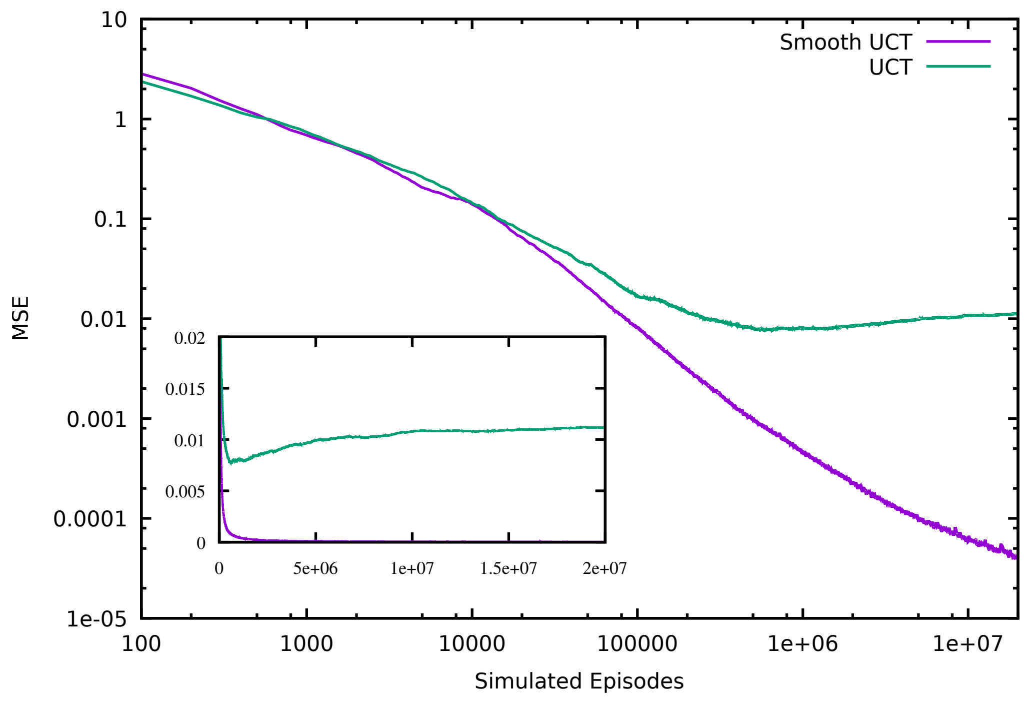


Figure 3.2: Learning curves in Kuhn poker.

3.5.2 Leduc Hold'em

We also compared Smooth UCT to OS (see Section 2.3.4) and UCT in Leduc Hold'em. Due to not knowing the Nash equilibria in closed form, we measured learning performance in terms of exploitability of the average strategy profile (see Equation 2.2.2). Smooth UCT's mixing parameter schedule (3.1) was manually tuned in preliminary experiments on Leduc Hold'em. In particular, we varied one parameter at a time and measured the resulting achieved exploitability. The best

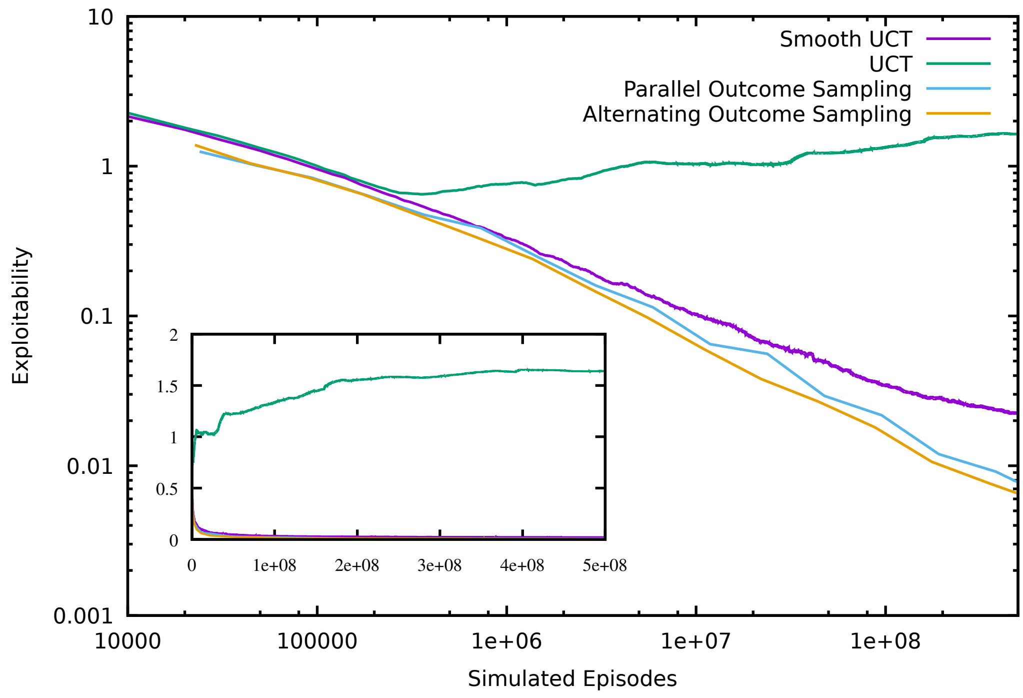


Figure 3.3: Learning curves in Leduc Hold'em.

results were achieved with $\gamma = 0.1$, $\eta = 0.9$ and $d = 0.002$. Smooth UCT and UCT trained 5 times for 500 million episodes each with exploration parameter settings of $c = 14 + 2k$, $k = 0, ..., 4$. We report the best average performance which was achieved with $c = 20$ and $c = 18$ for UCT and Smooth UCT respectively. We compared to both Parallel and Alternating Outcome Sampling (Lanctot, 2013). Both variants trained for 500 million episodes with exploration parameter settings of $\varepsilon = 0.4 + 0.1k$, $k = 0, ..., 4$. We report the best results which were achieved with $\varepsilon = 0.5$ for both variants.

Figure 3.3 shows that UCT diverged rather quickly and never reached a level of low exploitability. Smooth UCT, on the other hand, learned just as fast as UCT but continued to approach a Nash equilibrium. For the first million episodes, Smooth UCT's performance is comparable to Outcome Sampling. Afterwards, the slope of its performance curve begins to slowly increase. Alternating Outcome Sampling achieved an exploitability of 0.0065, whereas Smooth UCT was exploitable by 0.0223 after 500 million episodes.

We investigated Smooth UCT's long-term performance, by running it for 10

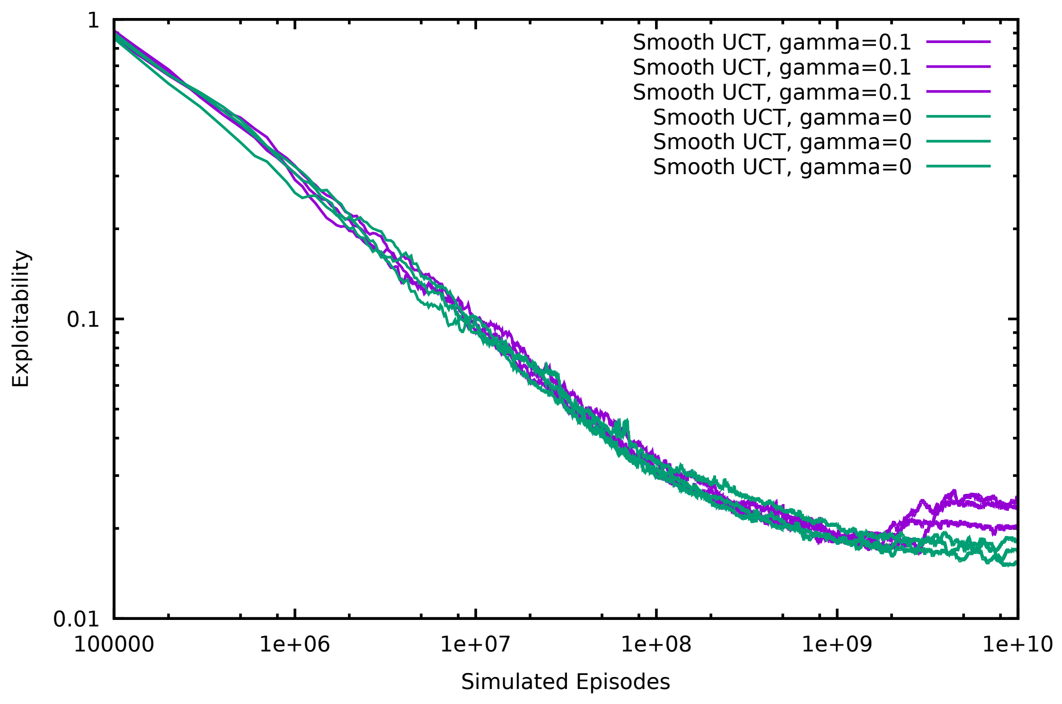


Figure 3.4: Long-term learning curves in Leduc Hold'em.

billion episodes with the same parameters as in the previous experiment. Figure 3.4 shows that Smooth UCT indeed struggles with convergence to a perfect Nash equilibrium. Setting γ to zero in the mixing parameter schedule (3.1) appears to stabilise its performance at an exploitability of around 0.017.

3.5.3 Limit Texas Hold'em

Finally, we consider LHE with two and three players. The two-player game tree contains about 10^{18} nodes. In order to reduce the game tree to a tractable size, various abstraction techniques have been proposed in the literature (Billings et al., 2003; Johanson et al., 2013). The most common approach is to group information states according to strategic similarity of the corresponding cards and leave the players' action sequences unabstracted. The resulting information state *buckets* define a state-aggregating function approximator (compare Section 2.1.4), i.e. abstraction.

In this section, we apply MCTS to an abstraction. This abstraction is handcrafted with a common bucketing metric called expected hand strength squared (Zinkevich et al., 2007). At a final betting round, there are five public community cards that players can combine their private cards with. The hand strength of such

Game	Preflop	Flop	Turn	River
two-player LHE	169	1000	500	200
three-player LHE	169	1000	100	20

Table 3.1: $\mathbb{E}[\mathrm{HS}^2]$ discretization grids used in experiments.

a combination is defined as its winning percentage against all combinations that are possible with the respective community cards. On any betting round, $\mathbb{E}[\mathrm{HS}^2]$ denotes the expected value of the squared hand strength on the final betting round. We have discretized the resulting $\mathbb{E}[\mathrm{HS}^2]$ values with an equidistant grid over their range, $[0,1]$. Table 3.1 shows the grid sizes that we used in our experiments. We used an imperfect-recall abstraction (Waugh et al., 2009b) that does not let players remember their $\mathbb{E}[\mathrm{HS}^2]$ values of previous betting rounds. An imperfect-recall abstraction is simpler to generate and implement. Furthermore, it allows for a finer discretization of hand strength values, as the information state assignment (bucketing) at later rounds does not need to condition on the assignment at previous rounds.

In LHE, there is an upper bound on the possible terminal pot size given the betting that has occurred so far in the episode. This is because betting is capped at each betting round. To avoid potentially excessive exploration we dynamically update the exploration parameter during a simulated episode at the beginning of each betting round and set it to

$$c = \min\left(C, \text{potsize} + k * \text{remaining betting potential}\right), \tag{3.2}$$

where C and $k \in [0,1]$ are constant parameters and the remaining betting potential is the maximum possible amount that players can add to the pot in the remainder of the episode. Note that the remaining betting potential is $48, 40, 32, 16$ at the beginning of the 4 betting rounds respectively. In particular, for $k = 1$ at each information state the exploration parameter is bounded from above by the maximum achievable potsize in the respective subtree. Furthermore, in two-player LHE, for $k = 0.5$ it is approximately bounded by the maximum achievable total return. This is because each player contributes about half of the potsize.

Annual Computer Poker Competition We submitted SmooCT, a program trained with an early version of the Smooth UCT algorithm (Heinrich and Silver, 2014), to the 2014 ACPC, where it ranked second in all two- and three-player LHE competitions. Our experiments in this dissertation improve on the performance of SmooCT, the competition program. In particular, SmooCT's implementation contained the following differences. First, it used a different mixing parameter schedule (compare (Heinrich and Silver, 2014)). Second, it had a bug in its preflop abstraction that prevented the agents from distinguishing between suited[1] and unsuited cards preflop. Third, it used a coarser abstraction bucket granularity (compare Table 3.1). Fourth, for its 3-player instance we manually refined its abstraction granularity for often-visited subtrees.

Two-player We trained strategies for two-player LHE by UCT and Smooth UCT. Both methods performed a simulation-based search in the full game. For evaluation, we extracted a greedy policy profile that at each information state takes the action with the highest estimated value. Learning performance was measured in milli-big-blinds won per hand, mbb/h, in actual play against benchmark opponents. To reduce variance of the evaluation, we averaged the results obtained from symmetric play that permuted the positions of players, reusing the same random card seeds. We had access to a benchmark server of the ACPC which enabled us to evaluate against the contestants of the 2014 competition.

Based on the experiments in Kuhn and Leduc poker, we selected a parameterization for Smooth UCT's mixing parameter schedule (3.1) and set it to $\gamma = 0.1$, $\eta = 0.9$ and $d = 20000^{-1}$. In the exploration schedule (3.2) we set $k = 0.5$ and $C = 24$, which corresponds to half of the maximum potsize achievable in two-player LHE. This results in exploration parameter values of $c \in [10, 24]$, which is centred around 17, a value that has been reported in a previous calibration of UCT in LHE by Ponsen et al. (2011).

UCT and Smooth UCT planned for 14 days each, generating about 62.1 and 61.7 billion simulated episodes respectively; note that Smooth UCT had almost

[1]Cards of the same suit.

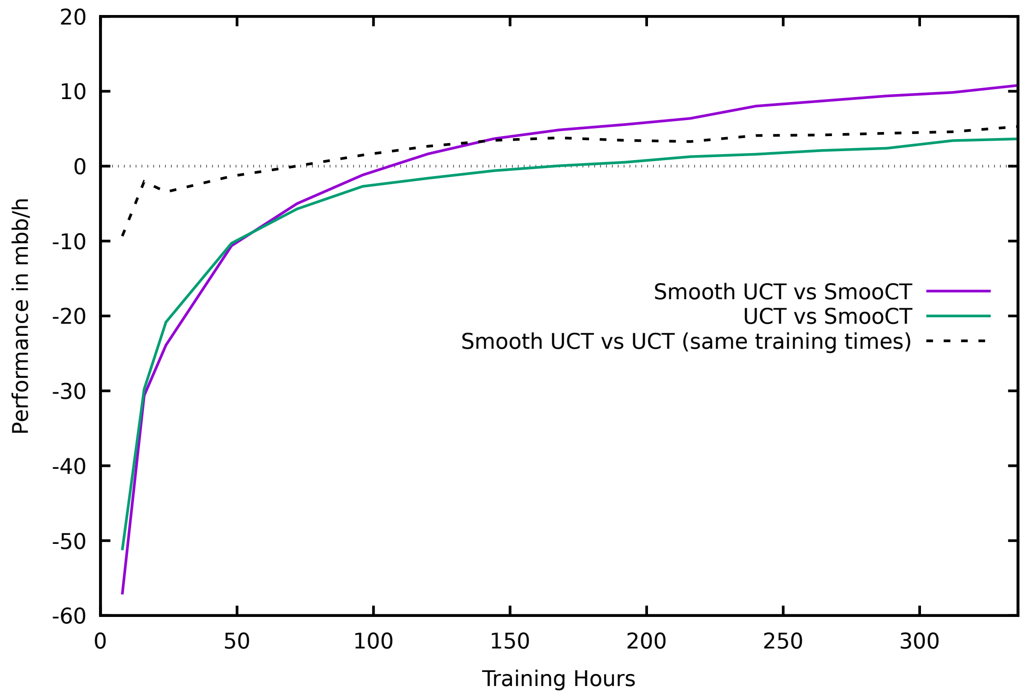


Figure 3.5: Learning performance in two-player Limit Texas Hold'em, evaluated against SmooCT, the runner-up in the ACPC 2014. The estimated standard error at each point of the curves is less than 1 mbb/h.

no computational overhead compared to UCT. We trained on a modern desktop machine, using a single thread and less than 200 MB of RAM. Each greedy strategy profiles' uncompressed size was 8.1 MB.

During training, snapshots of the strategies were taken at regular time intervals in order to measure learning performance over time. In particular, we evaluated each snapshot by symmetric play for 2.5 million games against SmooCT, the silver-medal contestant of the 2014 ACPC. In addition, we compared UCT's and Smooth UCT's snapshots by symmetric play against each other. The results in figure 3.5 show that UCT performed slightly better for a training time of under 72 hours. After 72 hours Smooth UCT outperformed UCT and was able to widen the gap over time.

Table 3.2 presents an extensive evaluation of greedy policies that were obtained from UCT and Smooth UCT. The table includes results against all but one contestants of the 2014 ACPC, in the order of their ranking in the competition. We had to omit the one contestant because it was broken on the benchmark server. The table also includes matches between Smooth UCT and UCT. Smooth UCT performed better than UCT against all but one of the top-7 benchmark agents. Performance against the weaker 6 ACPC contestants was more even, with Smooth UCT performing better in just half of the match-ups. Finally, Smooth UCT won in the match-up against UCT and achieved a higher average performance.

Smooth UCT lost against all but 2 of the top-7 contestants of the 2014 ACPC. This might be partly due to the much bigger and sophisticated abstractions and resources used by most of these agents. However, Smooth UCT achieved strong average performance against the whole field of agents. Similar results were achieved by SmooCT in the 2014 two-player LHE competition (ACPC). This suggests that Smooth UCT, using tiny resources and a small abstraction, can train highly competitive policies that perform well in an ecosystem of variable player types. Regarding the losses against the top agents, it remains unclear whether Smooth UCT would be able to efficiently compute good approximate Nash equilibria if it were provided with a larger abstraction.

Match-up	Smooth UCT	UCT
escabeche	-23.49 $\pm$ 3.2	-30.26 $\pm$ 3.2
SmooCT	10.78 $\pm$ 0.8	3.64 $\pm$ 0.9
Hyperborean	-24.81 $\pm$ 4.2	-25.03 $\pm$ 4.3
Feste	28.45 $\pm$ 4.0	20.02 $\pm$ 4.1
Cleverpiggy	-25.22 $\pm$ 4.0	-30.29 $\pm$ 4.0
ProPokerTools	-18.30 $\pm$ 4.0	-19.84 $\pm$ 3.9
652	-20.76 $\pm$ 4.0	-19.49 $\pm$ 4.0
Slugathorus	93.08 $\pm$ 5.7	93.13 $\pm$ 5.8
Lucifer	139.23 $\pm$ 4.7	138.62 $\pm$ 4.7
PokerStar	167.65 $\pm$ 4.9	173.19 $\pm$ 5.0
HITSZ_CS_14	284.72 $\pm$ 4.5	281.55 $\pm$ 4.5
chump9	431.11 $\pm$ 7.7	435.26 $\pm$ 7.8
chump4	849.30 $\pm$ 8.5	789.28 $\pm$ 8.7
Smooth UCT	0	-5.28 $\pm$ 0.6
UCT	5.28 $\pm$ 0.6	0
average	51.38	48.33
average*	135.50	128.89

Table 3.2: Two-player Limit Texas Hold'em winnings in mbb/h and their standard errors. The average results are reported with and without including chump4 and chump9.

Three-player Next we performed a similar evaluation in three-player LHE. Smooth UCT used the same mixing parameter schedule as in two-player LHE. The exploration schedule (3.2) was set to $k = 0.5$ and $C = 36$, which corresponds to half of the maximum potsize achievable in three-player LHE. UCT and Smooth UCT planned for 10 days each, generating about 50.9 and 49.4 billion simulated episodes respectively. Once again we trained on a modern desktop machine, using a single thread and less than 3.6 GB of RAM. Each final greedy strategy profiles' uncompressed size was about 435 MB.

The 2014 ACPC featured two three-player LHE competitions that were won by Hyperborean_tbr and Hyperborean_iro. In both competitions SmooCT and KEmpfer finished second and third out of 5 respectively.

Table 3.3 presents our three-player results. Smooth UCT outperformed UCT in all but 3 match-ups and achieved a higher average performance overall.

Match-up	Smooth UCT	UCT
Hyperborean_iro, KEmpfer	27.06 ± 8	11.35 ± 8
Hyperborean_tbr, KEmpfer	8.84 ± 9	2.79 ± 9
Hyperborean_tbr, SmooCT	-17.10 ± 8	-31.46 ± 9
Hyperborean_tbr, HITSZ_CS_14	69.91 ± 10	74.73 ± 10
SmooCT, KEmpfer	42.30 ± 8	50.35 ± 8
SmooCT, HITSZ_CS_14	133.49 ± 9	125.79 ± 9
KEmpfer, HITSZ_CS_14	171.96 ± 9	194.55 ± 9
2x SmooCT	6.17 ± 1	-5.77 ± 1
2x Smooth UCT	0	-8.16 ± 1
2x UCT	7.51 ± 1	0
average	50.02	46.02

Table 3.3: Three-player Limit Texas Hold'em winnings in mbb/h and their standard errors.

3.6 Conclusion

We have introduced Smooth UCT, a MCTS algorithm for extensive-form games with imperfect information. In two small poker games, it was able to learn as fast as UCT but approached an (approximate) Nash equilibrium whereas UCT diverged. Furthermore, in two- and three-player LHE, a game of real-world scale, Smooth

UCT outperformed UCT and achieved three silver medals in the 2014 ACPC. The results suggest that highly competitive strategies can be learned by a full-game simulation-based search with Smooth UCT.

Introducing the average strategy into the self-play learning process significantly improved the performance of UCT. However, in our experiments in Leduc Hold'em Smooth UCT performed worse than OS and plateaued at an approximate Nash equilibrium. Hence, Smooth UCT's heuristic implementation of fictitious play is promising but imperfect. In the next chapter we introduce an extension of fictitious play to extensive-form games and thus develop a more principled foundation for self-play reinforcement learning.

Chapter 4

Fictitious Play in Extensive-Form Games

In this chapter we address the subquestion,

> Can fictitious play be extended to extensive-form games?

The motivation is to create a self-play algorithm that offers convergence guarantees in extensive-form games. In particular, fictitious play exhibits key similarities to reinforcement learning. Thus, an extensive-form variant could serve as a principled foundation for the design of self-play reinforcement learning methods.

4.1 Introduction

Fictitious play is a classic example of self-play learning that has inspired artificial intelligence algorithms in games. Despite the popularity of fictitious play to date, it has seen use in few large-scale applications, e.g. (Lambert III et al., 2005; McMahan and Gordon, 2007; Ganzfried and Sandholm, 2009). One possible reason for this is its reliance on a normal-form representation. While any extensive-form game can be converted into a normal-form equivalent (Kuhn, 1953), the resulting number of actions is typically exponential in the number of game states. The extensive form offers a much more efficient representation via behavioural strategies, whose number of parameters is linear in the number of information states.

In this chapter, we extend fictitious play to extensive-form games. We begin with a discussion of various strategy updates. Using one of these updates, we introduce Extensive-Form Fictitious Play (XFP). It is realization equivalent to normal-form fictitious play and therefore inherits its convergence guarantees. However, it can be implemented using only behavioural strategies and therefore its computational complexity per iteration is linear in the number of game states rather than exponential. We empirically compare our methods to CFR (Zinkevich et al., 2007) and a prior extensive-form fictitious play algorithm (Hendon et al., 1996). Furthermore, we empirically evaluate XFP's robustness to approximation errors, which is useful for machine learning applications.

4.2 Best Response Computation

At each iteration, fictitious players compute best responses to their fellow players' average strategy profile. In extensive-form games, these best responses can be efficiently computed by dynamic programming, as described in Section 2.2.6.

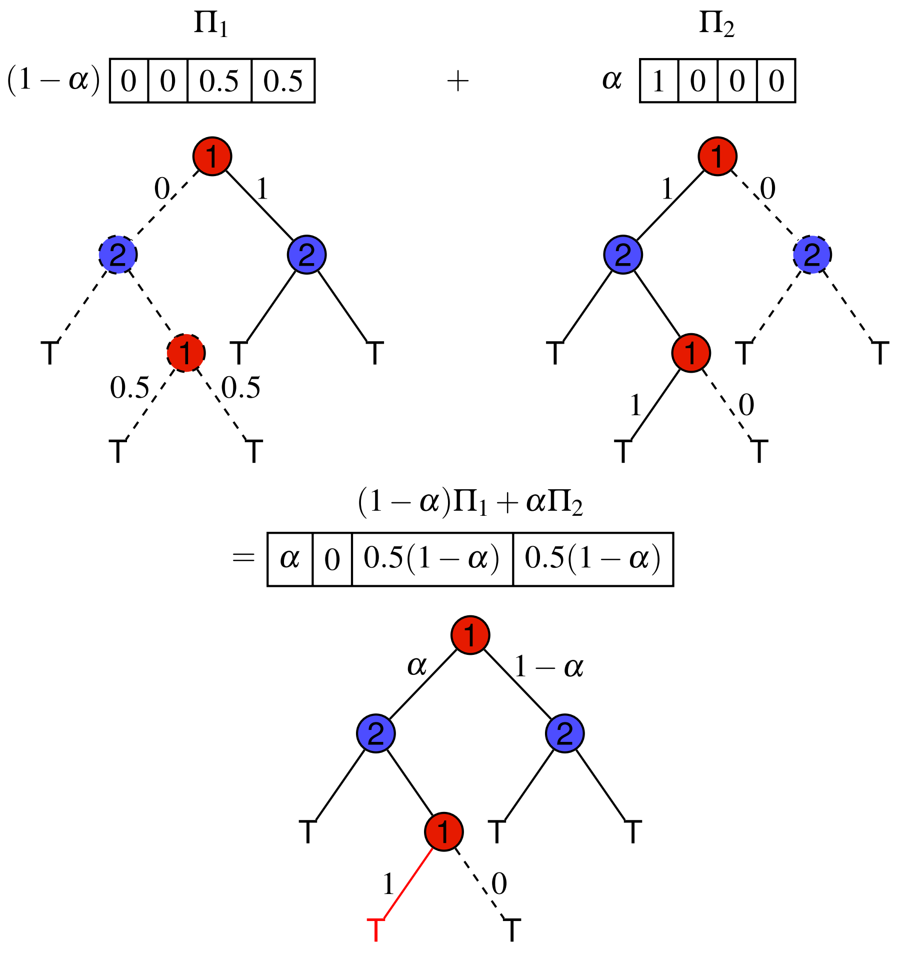
4.3 Strategy Updates

This section explores various options for aggregating extensive-form, behavioural strategies.

4.3.1 Mixed Strategies

We begin by analysing the mixed strategy updates that a common, normal-form fictitious player would perform. A mixed strategy is a probability distribution over normal-form actions. Therefore, a weighted aggregate of two mixed strategies can be computed by taking a convex combination of the two corresponding vectors of probabilities.

Figure 4.1 depicts such an aggregate of two mixed strategies and its equivalent representation in extensive form. The aggregate behaviour at player 1's initial state is unsurprisingly composed of the two strategies' behaviour at that state in proportion to their combination's weights, α and $1-\alpha$. Interestingly, the aggregate behaviour at player 1's second state is entirely determined by his second strategy, Π_2. This is because a normal-form aggregate selects (pure) strategies for entire playouts of the game and does not resample at each information state that is encountered dur-



Π_1
$(1-\alpha)$ 0 0 0.5 0.5
+
Π_2
α 1 0 0 0
$(1-\alpha)\Pi_1 + \alpha\Pi_2$
$=$ α 0 $0.5(1-\alpha)$ $0.5(1-\alpha)$

Figure 4.1: Illustration of mixed strategy updates in extensive form

ing such playouts. Because strategy Π_1 never reaches the second state, its behaviour at that state has no impact on the normal-form aggregate behaviour.

4.3.2 Unweighted Behavioural Strategies

A simple option for aggregating behavioural strategies is to linearly combine them with equal weights at each state. In this section we discuss certain drawbacks of such updates.

Firstly, this form of aggregaton would clearly yield a different result than the normal-form aggregation shown in Figure 4.1. Hence, replacing normal-form fictitious play updates with unweighted behavioural updates in the extensive form would yield a different variant of fictitious play (Hendon et al., 1996), calling into question

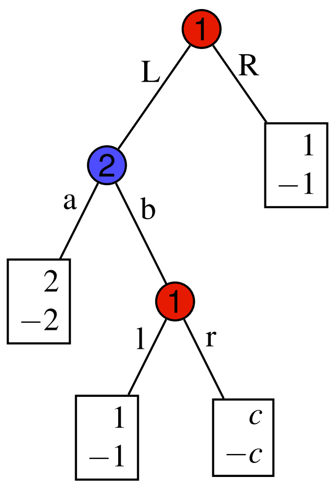


Figure 4.2: Game used in proof of proposition 4.3.1

its convergence guarantees.

Secondly, consider two strategies that achieve a similar return performance against an opponent strategy profile. For a combination of two such strategies, we might consider it desirable that the aggregate strategy achieve a performance similar to its constituent strategies. However, if we combine strategies unweightedly then their aggregate performance can be arbitrarily worse, as shown by the following result.

Proposition 4.3.1. *There exists a (parameterised) game, two strategies, π_1 and π_2, and an opponent strategy profile, π^{-i}, such that both π_1 and π_2 have equal return performance against π^{-i} but the unweightedly aggregated strategy, $(1-\lambda)\pi_1 + \lambda\pi_2$, performs arbitrarily worse.*

Proof. Consider the game in Figure 4.2 and strategies $\pi_1 \equiv$ L,l and $\pi_2 \equiv$ R,r. Both achieve a return of 1 against an opponent strategy profile, $\pi^{-i} \equiv b$, that always takes action b. However, an unweighted aggregate, $(1-\lambda)\pi_1 + \lambda\pi_2$, will achieve a return of $1 + \lambda(1-\lambda)(c-1) < 1$ for $c < 1$. As c could be arbitrarily low, the result follows. $\square$

In a reinforcement learning setting, this problematic scenario may occur when aggregating two agents' policies that focus on different areas of the state space. An unweighted aggregate of two such policies could be tainted by suboptimal behaviour in states that one agent but not the other chose to avoid.

4.3.3 Realization-Weighted Behavioural Strategies

This section introduces an approach to combining strategies that overcomes the return performance corruption that unweighted aggregation may cause. Furthermore, this method of combining strategies is equivalent to the corresponding normal-form aggregation of the respective mixed strategies.

The main idea is to analyse the extensive-form behaviour of a convex combination of normal-form, mixed strategies. It turns out that the aggregate behaviour can be efficiently expressed in closed form as a combination of the realization-equivalent behavioural strategies' behaviours. The following proposition formalizes our analysis.

Proposition 4.3.2. *Let $\pi_1, ..., \pi_n$ be behavioural strategies of an extensive-form game. Let $\Pi_1, ..., \Pi_n$ be mixed strategies that are realization-equivalent to $\pi_1, ..., \pi_n$ respectively. Let $w_1, ..., w_n \in \mathbb{R}_{\geq 0}$ be weights that sum to 1. Then the convex combination of mixed strategies,*

$$\Pi = \sum_{k=1}^{n} w_k \Pi_k,$$

is realization-equivalent to the behavioural strategy,

$$\pi(u) \propto \sum_{k=1}^{n} w_k x_{\pi_k}(\sigma_u) \pi_k(u) \quad \forall u \in \mathscr{U}, \tag{4.1}$$

where $\sum_{k=1}^{n} w_k x_{\pi_k}(\sigma_u)$ is the normalizing constant for the strategy at information state u.

Proof. The realization plan of $\Pi = \sum_{k=1}^{n} w_k \Pi_k$ is

$$x_\Pi(\sigma_u) = \sum_{k=1}^{n} w_k x_{\Pi_k}(\sigma_u), \quad \forall u \in \mathscr{U}.$$

and due to realization equivalence, $x_{\Pi_k} = x_{\pi_k}$ for $k = 1, ..., n$. This realization plan induces a realization-equivalent behavioural strategy

$$\begin{aligned}\pi(a\,|\,u) &= \frac{x_\Pi(\sigma_u a)}{x_\Pi(\sigma_u)} \\ &= \frac{\sum_{k=1}^n w_k x_{\Pi_k}(\sigma_u a)}{\sum_{k=1}^n w_k x_{\Pi_k}(\sigma_u)} \\ &= \frac{\sum_{k=1}^n w_k x_{\pi_k}(\sigma_u)\pi_k(a\,|\,u)}{\sum_{k=1}^n w_k x_{\pi_k}(\sigma_u)}.\end{aligned}$$

□

Each constituent strategy's behaviour $\pi_k(u)$ at an information state s is weighted by its probability of realizing this information state relative to the aggregate strategy's realization probability,

$$\frac{w_k x_{\pi_k}(\sigma_u)}{x_\pi(\sigma_u)}. \tag{4.2}$$

This weight is the conditional probability of following strategy π_k locally at u given that the aggregate strategy π is played globally. Hence, it is 0 for a constituent strategy that never reaches the respective information state and it can be as large as 1 if it is the only strategy in the combination that reaches the information state. In Section 5.4, we exploit this conditional probability to devise sample-based approximations of strategy aggregates.

The following corollary analyses the performance of realization-weighted aggregates.

Corollary 4.3.3. *Let π^{-i} be a strategy profile of a player i's fellow players in an extensive-form game. Let $\pi_1^i, ..., \pi_n^i$ be behavioural strategies of player i, achieving a return of $G_1, ..., G_n$ against π^{-i} respectively. Let $w_1, ..., w_n \in \mathbb{R}_{\geq 0}$ be weights that sum to 1 and*

$$\pi^i(u) \propto \sum_{k=1}^n w_k x_{\pi_k^i}(\sigma_u)\pi_k^i(u) \quad \forall u \in \mathscr{U}^i,$$

a behavioural strategy. Then π^i's performance against π^{-i} is a similarly weighted *convex combination of its constitutent strategies' returns,*

$$\sum_{k=1}^{n} w_k G_k.$$

Proof. Given Proposition 4.3.2, π^i is realization equivalent to a mixed strategy $\Pi^i = \sum_{k=1}^{n} w_k \Pi_k^i$, where each constitutent strategy, Π_k^i is realization equivalent to π_k^i. Following the definition of a mixed strategy, i.e. each constituent strategy Π_k^i is played with probability w_k, the expected return is a similarly weighted convex combination. □

Corollary 4.3.3 shows that for realization-weighted strategy aggregation the aggregate return is convex in the constituent strategy's returns. Thus, the return degradation shown for unweighted aggregates in Proposition 4.3.1 does not occur with realization-weighted strategy aggregation.

4.4 Extensive-Form Fictitious Play

In this section we combine the extensive-form best response computation and average strategy updates into a fictitious play that is entirely implemented in behavioural strategies. Thus, for sequential games computation in the exponentially less efficient normal-form representation can be avoided. Nonetheless, this extensive-form variant of fictitious play is realization-equivalent to its normal-form counterpart and thus inherits its convergence guarantees.

The following theorem presents an extensive-form fictitious play that inherits the convergence results of generalised weakened fictitious play by realization-equivalence.

Theorem 4.4.1. *Let π_1 be an initial behavioural strategy profile. The extensive-form process*

$$\beta_{t+1}^i \in \mathrm{BR}_{\varepsilon_{t+1}}^i(\pi_t^{-i}),$$

$$\pi_{t+1}^i(u) = \pi_t^i(u) + \frac{\alpha_{t+1} x_{\beta_{t+1}^i}(\sigma_u)}{(1-\alpha_{t+1}) x_{\pi_t^i}(\sigma_u) + \alpha_{t+1} x_{\beta_{t+1}^i}(\sigma_u)} \left(\beta_{t+1}^i(u) - \pi_t^i(u)\right), \quad (4.3)$$

for all players $i \in \mathcal{N}$ and all their information states $u \in \mathcal{U}^i$, with $\alpha_t \to 0$ and $\varepsilon_t \to 0$ as $t \to \infty$, and $\sum_{t=1}^{\infty} \alpha_t = \infty$, is realization-equivalent to a generalised weakened fictitious play in the normal-form and thus the average strategy profile converges to a Nash equilibrium in all games with the fictitious play property.

Proof. By induction. Assume π_t and Π_t are realization equivalent and $\beta_{t+1} \in b_{\varepsilon_{t+1}}(\pi_t)$ is an ε_{t+1}-best response to π_t. By Kuhn's Theorem, let B_{t+1} be any mixed strategy that is realization equivalent to β_{t+1}. Then B_{t+1} is an ε_{t+1}-best response to Π_t in the normal-form. By Lemma 6, the update in behavioural strategies, π_{t+1}, is realization equivalent to the following update in mixed strategies

$$\Pi_{t+1} = (1 - \alpha_{t+1})\Pi_t + \alpha_{t+1} B_{t+1}$$

and thus follows a generalised weakened fictitious play. $\square$

While the generality of this theorem, i.e. variable step size and approximate best responses, is not really required for a full-width approach, it is useful for guiding the development of sample-based, approximate reinforcement learning techniques.

Algorithm 4 implements XFP, an extensive-form fictitious play according to Theorem 4.4.1. The initial average strategy profile, π_1, can be defined arbitrarily, e.g. uniform random. At each iteration the algorithm performs two operations. First it computes a best response profile to the current average strategies. Secondly it uses the best response profile to update the average strategy profile. The first iteration's computational requirements are linear in the number of game states. For each player the second operation can be performed independently from their opponents and requires work linear in the player's number of information states. Furthermore, if a deterministic best response is used, the realization weights of Theorem 4.4.1 allow ignoring all but one subtree at each of the player's decision nodes.

Algorithm 4 Extensive-Form Fictitious Play (XFP)

```
function FICTITIOUSPLAY($\Gamma$)
    Initialize $\pi_1$ arbitrarily
    $j \leftarrow 1$
    while within computational budget do
        $\beta_{j+1} \leftarrow$ COMPUTEBESTRESPONSES($\pi_j$)
        $\pi_{j+1} \leftarrow$ UPDATEAVERAGESTRATEGIES($\pi_j, \beta_{j+1}$)
        $j \leftarrow j+1$
    end while
    return $\pi_j$
end function

function COMPUTEBESTRESPONSES($\pi$)
    Compute a best-response strategy profile, $\beta \in \mathrm{BR}(\pi)$, with dynamic programming, using Equations 2.24 and 2.25.
    return $\beta$
end function

function UPDATEAVERAGESTRATEGIES($\pi_j, \beta_{j+1}$)
    for each player $i$ do
        Compute an updated strategy $\pi^i_{j+1}$ by applying Equation 4.3 to all information states of the player.
    end for
    return $\pi_{j+1}$
end function
```

4.5 Experiments

4.5.1 Realization-Weighted Updates

In this section, we investigate the effects of unweighted and realization-weighted strategy updates on the performance of extensive-form fictitious play.

We define updates with effective step sizes, $\lambda_{t+1} : \mathcal{U} \to [0,1]$,

$$\pi_{t+1}(u) = \pi_t(u) + \lambda_{t+1}(u)(\beta_{t+1}(u) - \pi_t(u)), \quad \forall u \in \mathcal{U},$$

where $\beta_{t+1} \in b(\pi_t)$ is a sequential best response and π_t is the iteratively updated average strategy profile. Stepsize $\lambda^1_{t+1}(u) = \frac{1}{t+1}$ produces the sequential extensive-form fictitious play (Hendon et al., 1996), which applies unweighted strategy updates. XFP is implemented by stepsize $\lambda^2_{t+1}(u) = \frac{x_{\beta_{t+1}}(\sigma_u)}{t x_{\pi_t}(\sigma_u) + x_{\beta_{t+1}}(\sigma_u)}$. Note that both updates use a basic step size of $\alpha_t = \frac{1}{t}$ (compare Equation 4.3).

The initial average strategy profiles were initialized randomly as follows. At each information state u, we drew the weight for each action from a uniform distribution and normalized the resulting strategy at u. We trained each algorithm for 400000 iterations and measured the exploitability of the average strategy profile after each iteration. The experiment was repeated five times and all learning curves are plotted in figure 4.3. The results show noisy behaviour of each fictitious play process that used stepsize λ^1. Each instance of XFP reliably approached an approximate Nash equilibrium.

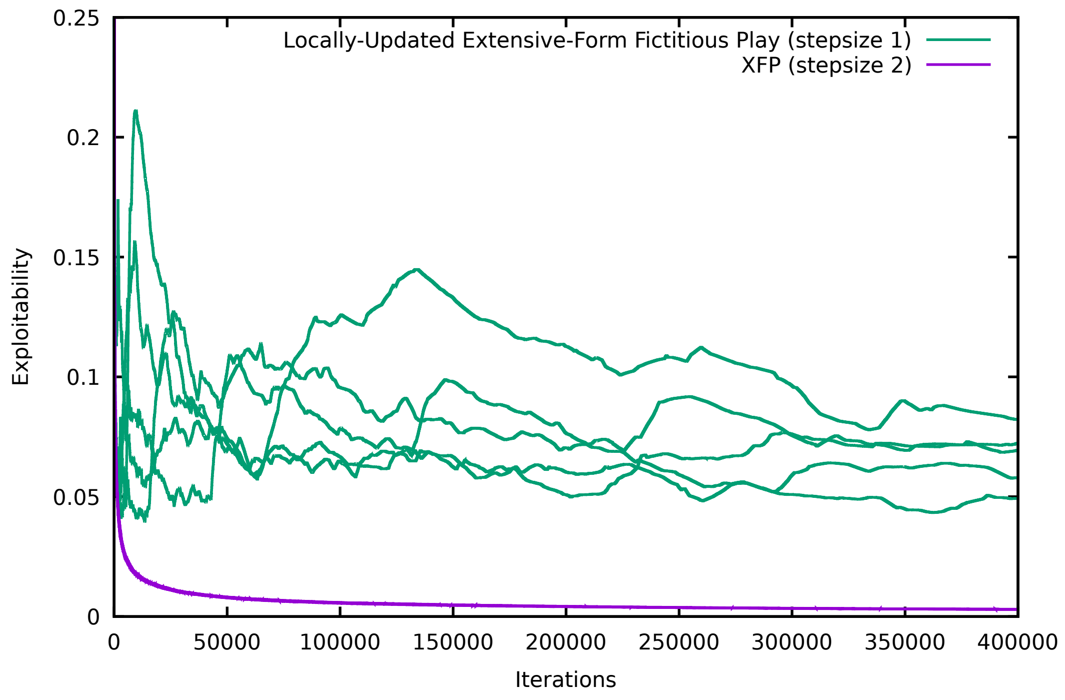


Figure 4.3: Learning curves of extensive-form fictitious play processes in Leduc Hold'em, for stepsizes λ^1 and λ^2.

4.5.2 GFP and Comparison to CFR

In Appendix B we introduce Geometric Fictitious Play (GFP), a novel, experimental variant of extensive-form fictitious play. This section compares the performance of XFP, GFP and CFR in Leduc Hold'em.

Note that the GFP update (see Equation B.2) weights the vanilla average strategy by $(1-\alpha)^{D+1}$, where D is the maximum depth of the player's information-state tree. For Leduc Hold'em we have $D = 3$. To control for the different effective stepsizes of XFP and GFP, we included variants with scaled and normalized stepsizes. In particular, we normalized GFP's stepsize by $\frac{\alpha_k}{1-(1-\alpha_k)^{D+1}}$ to match vanilla XFP's. Vice versa, we scaled XFP's stepsize by $\frac{1-(1-\alpha_k)^{D+1}}{\alpha_k}$ to match vanilla GFP's. For each variant, we repeated the experiment 3 times with randomly initialized average strategies (as above). All learning curves are plotted in Figure 4.4. The results show that GFP outperformed XFP in Leduc Hold'em. This improvement does not appear to be merely due to the larger effective stepsize, as scaling XFP's stepsize had a detrimental effect.

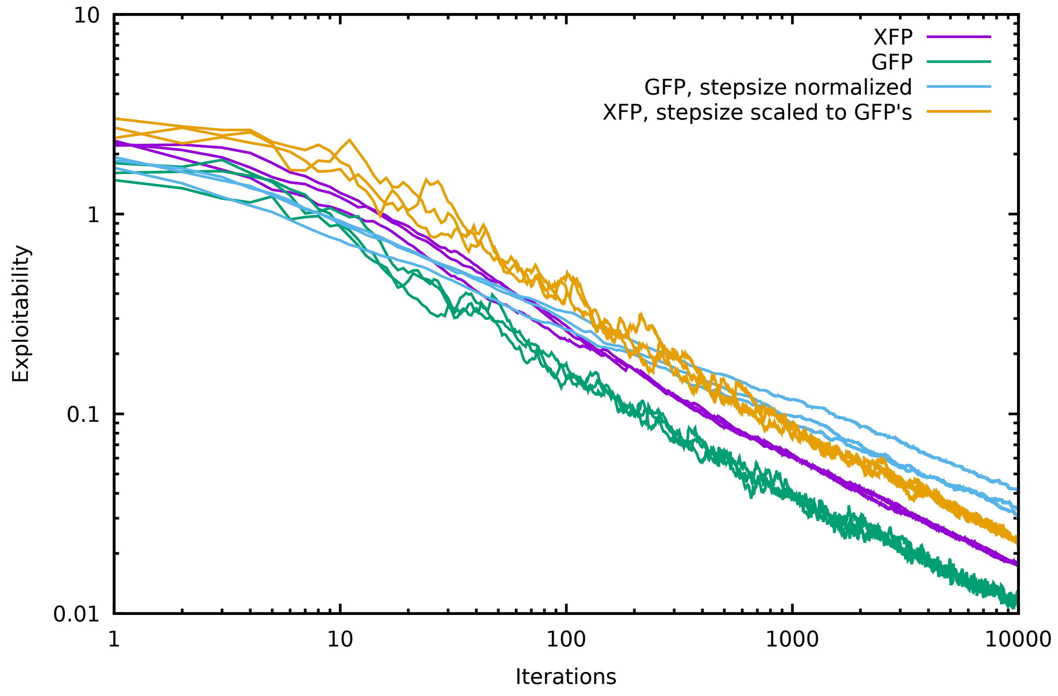


Figure 4.4: Performance of XFP and GFP variants in Leduc Hold'em.

We also compared the performance of XFP and GFP to CFR, the main game-theoretic approach to computing Nash equilibria from self-play. We initialized all variants' initial strategy to uniform random at all information states. As the algorithms are deterministic, we present a single run of all algorithms in Figure 4.5. Like CFR, both XFP and GFP appear to converge at a rate of $\mathcal{O}(k^{-\frac{1}{2}})$, where k is the iteration count, as conjectured by Karlin (1959).

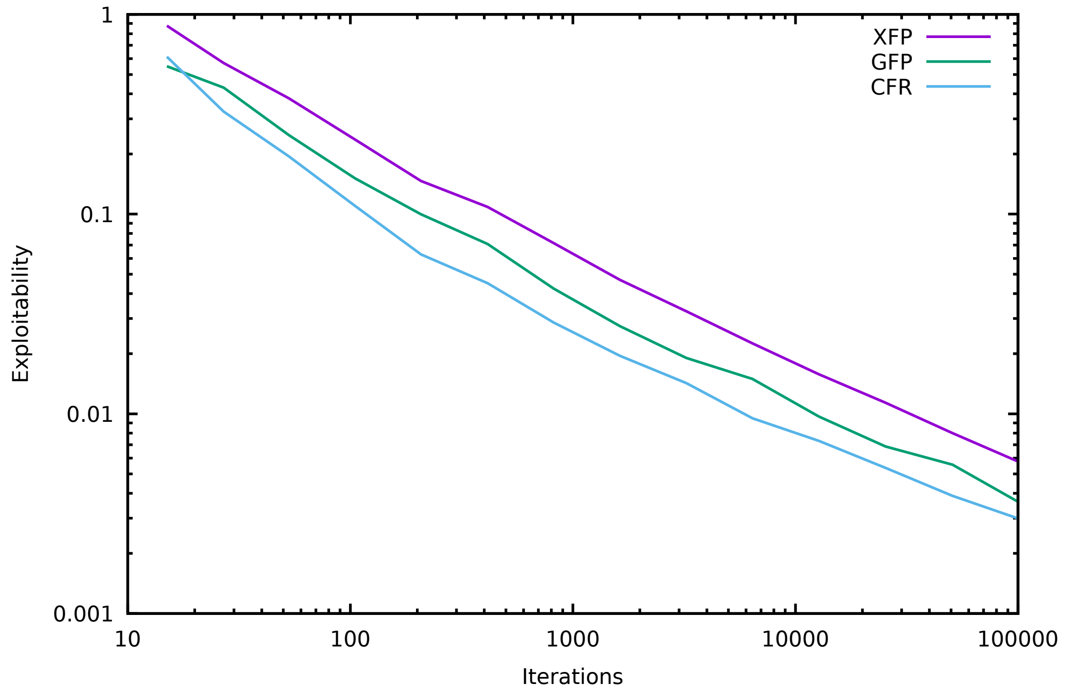


Figure 4.5: Comparison of XFP, GFP and CFR in Leduc Hold'em.

4.5.3 Robustness of XFP

Our ultimate goal is to develop self-play reinforcement learning algorithms that approximate XFP and achieve its convergence properties. This is partly motivated by the fact that generalised weakened fictitious play (Leslie and Collins, 2006) is robust to various approximations. As machine learning algorithms will incur various approximation and sampling errors, we investigate the potential effects of such errors on XFP's performance. In particular, we explore what happens when the perfect averaging used in XFP is replaced by an incremental averaging process closer to gradient descent. Furthermore, we explore what happens when the exact best response used in XFP is replaced by an approximation with epsilon error.

Figure 4.6 shows the performance of XFP with default, $1/T$, and constant stepsizes for its strategy updates. We see improved asymptotic but lower initial performance for smaller stepsizes. For constant stepsizes the performance seems to plateau rather than diverge. These results may guide algorithmic design decisions for machine learning of average strategies, e.g. a decaying learning rate of gradient descent might be a reasonable choice.

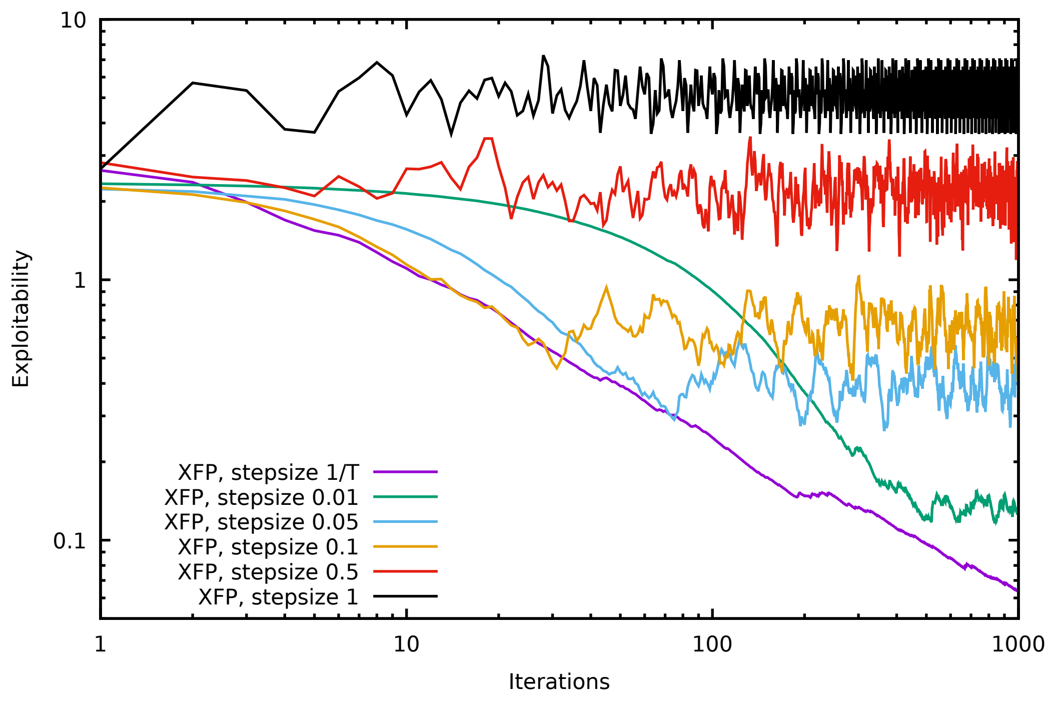


Figure 4.6: The impact of constant stepsizes on the performance of full-width fictitious play in Leduc Hold'em.

Reinforcement learning agents typically add random exploration to their policies and may use noisy stochastic updates to learn action values, which determine their approximate best responses. Therefore, we investigated the impact of random noise added to the best response computation, which XFP performs by dynamic programming. At each backward induction step, we return a uniform-random action's value with probability ε and the best action's value otherwise. Figure 4.7 shows monotonically decreasing performance with added noise. However, performance remains stable and keeps improving for all noise levels.

4.6 Conclusion

We have shown that fictitious play can be entirely implemented in behavioural strategies of the extensive-form representation. The resulting approach, XFP, is realization-equivalent to common, normal-form fictitious play and therefore preserves all its convergence guarantees. However, XFP is much more efficient in extensive-form games as its computational requirements in time and space are linear in the number of game states, rather than exponential.

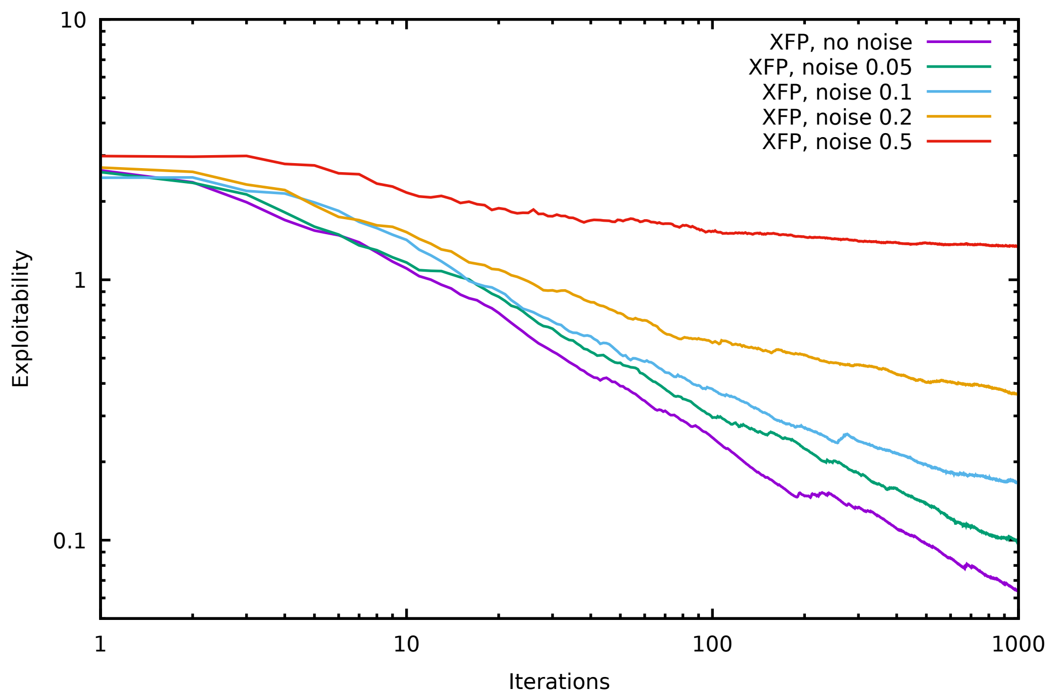


Figure 4.7: The performance of XFP in Leduc Hold'em with uniform-random noise added to the best response computation.

We complemented our theoretical results with the following empirical findings. Firstly, our experiments have shown that (realization-weighted) XFP significantly outperforms a prior extensive-form fictitious play algorithm (Hendon et al., 1996) that ignored such weightings. Secondly, in Leduc Hold'em we have seen GFP improve on the performance of XFP and both achieved a similar level of performance as CFR. Furthermore, both XFP and GFP appear to converge at a rate of $\mathscr{O}(k^{-\frac{1}{2}})$, where k is the iteration count, as conjectured by Karlin (1959). Finally, our experiments have found XFP to be robust to various approximation errors and fixed step sizes. This finding is aligned with the theoretical guarantees of generalised weakened fictitious play allowing for certain approximations and perturbations.

This chapter introduced realization-weighted, full-width updates of fictitious players' strategies. These updates' weights are in fact conditional probabilities that prescribe a way to sample the updated strategies. In the next chapter we develop sample-based methods that build upon this observation.

Chapter 5

Fictitious Self-Play

In this chapter we address the subquestion,

> Can XFP be approximated by agents learning from sampled experience?

Such agents, in principle, would be able to learn approximate Nash equilibria from self-play in games where fictitious play is guaranteed to converge.

5.1 Introduction

In XFP players compute their best responses, i.e. goal-directed strategies, by dynamic programming on their information state trees. In addition, they compute routine strategies that average their past best responses. Thus, XFP is reminiscent of an actor-critic architecture (Barto et al., 1983; Sutton et al., 1999a). In particular, XFP moves players' average strategies (actors) towards their best responses (critics). XFP has further properties that render it suitable for machine learning. First, best responses are just strategies that are best in hindsight, i.e. optimal policies with respect to an agent's experience. This enables the use of standard reinforcement learning techniques. Second, an average strategies' weights (compare Equation 4.1) are in fact conditional probabilites (see Equation 4.2) that are automatically realized when sampling from the agent's average policy. This enables the use of standard supervised learning techniques for mimicking the sampled behaviour (labels).

In this chapter, we develop a sample-based machine learning approach to finding Nash equilibria in imperfect-information games. We introduce Fictitious Self-

Play (FSP), a class of learning algorithms that approximate XFP. In FSP players repeatedly play a game and learn from their experience. In particular, players learn approximate best responses from their experience of play against their opponents. In addition, they learn a model of their own average, routine strategy from their experience of their own behaviour. In more technical terms, FSP iteratively samples episodes of the game from self-play. These episodes constitute experience that is used by reinforcement learning to produce approximate best responses and by supervised learning to produce models of average strategies.

5.2 Experiential Learning

XFP and related game-theoretic approaches face the following two challenges when applied to large domains.

XFP is a full-width approach that suffers from the curse of dimensionality. At each iteration, computation needs to be performed at all states of the game irrespective of their relevance. However, generalised weakened fictitious play (Leslie and Collins, 2006) only requires approximate best responses and even allows some perturbations in the updates. An extensive-form game usually has sequential structure that could be exploited when computing approximate best responses and strategy updates, e.g. by sampling likely trajectories of the game.

XFP is not compatible with function approximation. Each state is represented in isolation; preventing agents from generalising their knowledge between similar states. Currently, researchers in computational game theory usually circumvent this problem by hand-crafting abstractions of the game and applying their algorithm to the smaller, abstracted version of the game. It would be desirable that the algorithm itself can find such abstractions that best approximate the learned strategies and action values.

FSP addresses both problems by introducing two innovations. Firstly, it replaces XFP's full-width strategy and action-value updates with approximate, sample-based learning methods. Secondly, FSP agents can represent their strategies and action-value estimates with function approximators instead of using table-lookup representations. Approximate best responses are learned by reinforcement learning from experience of play against the opponents' average strategies. The average strategy updates can be formulated as a supervised learning task, where each player learns a transition model of their own behaviour. We introduce reinforcement learning-based best response computation in section 5.3 and present supervised learning-based strategy updates in section 5.4.

5.3 Best Response Learning

This section develops reinforcement learning of approximate best responses from sampled experience. We first break up the iterations of fictitious play into a sequence of MDPs. Then, we describe how agents can sample experience of these MDPs. Next, we present a suitable memory architecture for off-policy learning from such experience. Finally, we discuss the approximation effects of reinforcement learning on the fictitious play process.

5.3.1 Sequence of MDPs

Consider an extensive-form game Γ and some strategy profile π. Then for each player $i \in \mathcal{N}$ the strategy profile of their opponents, π^{-i}, defines an MDP, $\mathcal{M}(\pi^{-i})$ (Greenwald et al., 2013). Player i's information states define the states of the MDP (Silver and Veness, 2010). The MDP's dynamics are given by the rules of the extensive-form game, the chance function and the opponents' fixed strategy profile. The rewards are given by the game's return function.

An ε-optimal policy of the MDP, $\mathcal{M}(\pi^{-i})$, yields an ε-best response of player i to the strategy profile π^{-i}. Thus fictitious players' iterative computation of approximate best responses can be formulated as a sequence of MDPs to solve approximately, e.g. by applying reinforcement learning to samples of experience from the respective MDPs. In particular, to approximately solve the MDP $\mathcal{M}(\pi^{-i})$ we sample player i's experience from their opponents' strategy profile π^{-i}. Player i's strategy should ensure sufficient exploration of the MDP but can otherwise be arbitrary if an off-policy reinforcement learning method is used, e.g. Q-learning (Watkins and Dayan, 1992).

5.3.2 Sampling Experience

Each agent i tries to evaluate and maximise its action values, $Q^i(s,a) \approx \mathbb{E}_{\beta^i,\pi^{-i}}\left[G^i_t \,\middle|\, S_t = s, A_t = a\right]$, of playing its (approximate) best response policy β^i against its fellow agents' average strategy profile, π^{-i}. The approximate best response policy can be defined with respect to the Q-values, e.g. $\beta^i = \varepsilon\text{-greedy}\left[Q^i\right]$ or $\beta^i = \text{Boltzmann}\left[Q^i\right]$.

In an on-policy setting we can simply let each agent i play its best response policy against its opponents' fixed average strategy profile, π^{-i}, in alternating self-play. Through iteration of policy evaluation and policy improvement, each agent would learn an approximate best response. In an off-policy setting we can let all agents play their average strategies, π, in simultaneous self-play and have them collect their experience in a memory. This experience can then be evaluated with off-policy reinforcement learning, again yielding an approximate best response through iterative policy evaluation and improvement.

In this chapter we use off-policy, batch reinforcement-learning methods to learn from memorized experience. At each iteration k, FSP samples each agent i's experience from two types of strategy profiles, the average strategy profile π_k and the strategy profile $(\beta^i_{k+1}, \pi^{-i}_k)$. Both profiles generate experience of play against the current average strategy profile of the agent's opponents. Each agent i adds this experience to its replay memory, $\mathcal{M}^i_{RL}$. The data is stored in the form of transition tuples, $(u_t, a_t, r_{t+1}, u_{t+1})$, that are typical for reinforcement learning.

5.3.3 Memorizing Experience

We use a finite memory of fixed size. If the memory is full, new transitions replace existing transitions in a first-in-first-out order. Thus, each agent i's memory, $\mathcal{M}^i_{RL}$, contains experience of a sliding window of past MDPs, i.e.

$$\mathcal{M}^i_{RL} \approx \frac{1}{s+1} \sum_{k=t-s}^{t} \mathcal{M}(\pi^{-i}_k),$$

where s depends on the capacity of the memory. As fictitious players' average strategies, π_k, change more slowly over time,

$$\lim_{t\to\infty} \frac{1}{s+1} \sum_{k=t-s}^{t} \Pi_k = \Pi_t$$

the memory's windowing artifacts have a limited effect asymptotically,

$$\frac{1}{s+1} \sum_{k=t-s}^{t} \mathscr{M}(\pi_k^{-i}) \to \mathscr{M}(\pi_t^{-i}) \text{ for } t \to \infty.$$

5.3.4 Best Response Quality

While generalised weakened fictitious play allows ε_k-best responses at iteration k, it requires that the deficit ε_k vanishes asymptotically, i.e. $\varepsilon_k \to 0$ as $k \to \infty$. Learning such a valid sequence of ε_k-optimal policies of a sequence of MDPs would be hard if these MDPs were unrelated and knowledge could not be transferred. However, in fictitious play the MDP sequence has a particular structure. The average strategy profile at iteration k is realization-equivalent to a linear combination of two mixed strategies, $\Pi_k = (1-\alpha_k)\Pi_{k-1} + \alpha_k B_k$. Thus, in a two-player game, the MDP $\mathscr{M}(\pi_k^{-i})$ is structurally equivalent to an MDP that initially picks between $\mathscr{M}(\pi_{k-1}^{-i})$ and $\mathscr{M}(\beta_k^{-i})$ with probability $(1-\alpha_k)$ and α_k respectively. Due to this similarity between subsequent MDPs it is possible to transfer knowledge. The following proposition bounds the increase of the optimality deficit when transferring an approximate solution between subsequent MDPs in a fictitious play process.

Proposition 5.3.1. *Consider a two-player zero-sum extensive-form game with maximum return range* $\hat{R} = \max_{\pi\in\Delta} R^1(\pi) - \min_{\pi\in\Delta} R^1(\pi)$. *Consider a fictitious play process in this game. Let* Π_k *be the average strategy profile at iteration k,* B_{k+1} *a profile of* ε_{k+1}*-best responses to* Π_k*, and* $\Pi_{k+1} = (1-\alpha_{k+1})\Pi_k + \alpha_{k+1}B_{k+1}$ *the usual fictitious play update for some stepsize* $\alpha_{k+1} \in (0,1)$. *Then for each player i,* B_{k+1}^i *is an* $[\varepsilon_{k+1} + 2\alpha_{k+1}\hat{R}]$*-best response to* Π_{k+1}.

Proof. W.l.o.g. we prove the result for a player i at a fixed iteration $k \in \mathbb{N}$. Furthermore, set $\alpha = \alpha_{k+1}$ and $\varepsilon = \varepsilon_{k+1}$. Define $v_k = R^i(\mathrm{BR}^i(\Pi_k^{-i}), \Pi_k^{-i})$, the value of a best response to strategy profile Π_k^{-i}. Consider,

$$\begin{aligned} v_{k+1} &= R^i(\mathrm{BR}^i(\Pi_{k+1}^{-i}), \Pi_{k+1}^{-i}) \\ &\leq (1-\alpha)R^i(\mathrm{BR}^i(\Pi_k), \Pi_k) + \alpha\hat{R} \\ &= (1-\alpha)v_k + \alpha\hat{R} \end{aligned}$$

Then,

$$\begin{aligned} R^i(B_{k+1}^i, \Pi_{k+1}) &= (1-\alpha)R^i(B_{k+1}^i, \Pi_k^{-i}) + \alpha R^i(B_{k+1}^i, B_{k+1}^{-i}) \\ &\geq (1-\alpha)(v_k - \varepsilon) - \alpha\hat{R} \\ &\geq v_{k+1} - (1-\alpha)\varepsilon - 2\alpha\hat{R} \\ &\geq v_{k+1} - \left(\varepsilon + 2\alpha\hat{R}\right) \end{aligned}$$

$\square$

This bounds the absolute amount by which reinforcement learning needs to improve the best response profile to achieve a monotonic decay of the optimality gap ε_k. However, ε_k only needs to decay asymptotically. Given $\alpha_k \to 0$ as $k \to \infty$, the bound suggests that in practice a finite amount of learning per iteration might be sufficient to achieve asymptotic improvement of best responses. In Section 5.6.1, we empirically analyse whether this is indeed the case in practice. Furthermore, in our experiments on the robustness of (full-width) XFP (see Section 4.5.3), it robustly approached lower-quality strategies for noisy, low-quality best responses.

5.4 Average Strategy Learning

This section develops supervised learning of approximate average strategies from sampled experience. We first translate fictitious players' averaging of strategies into an equivalent problem of agents learning to model themselves. Then, we describe how appropriate experience for such models may be sampled. Next, we present suitable memory architectures for off-policy learning from such experience. Finally, we discuss the approximation effects of machine learning on the fictitious play process.

5.4.1 Modeling Oneself

Consider the point of view of a particular player i who wants to learn a behavioural strategy π that is realization-equivalent to a convex combination of their own normal-form strategies, $\Pi = \sum_{j=1}^{k} w_j B_j$, $\sum_{j=1}^{k} w_j = 1$. This task is equivalent to learning a model of the player's behaviour when it is sampled from Π. In addition to describing the behavioural strategy π explicitly, Proposition 4.3.2 prescribes a way to sample data sets of such convex combinations of strategies.

Corollary 5.4.1. *Let $\{B_k\}_{1\leq k\leq n}$ be mixed strategies of player i, $\Pi = \sum_{k=1}^{n} w_k B_k$, $\sum_{k=1}^{n} w_k = 1$ a convex combination of these mixed strategies and μ^{-i} a completely mixed sampling strategy profile that defines the behaviour of player i's fellow agents. Then the expected behaviour of player i, when sampling from the strategy profile (Π, μ^{-i}), defines a behavioural strategy,*

$$\pi(a\,|\,u) = \mathbb{P}\left(A_t^i = a\,|\,U_t^i = u, A_t^i \sim \Pi\right) \quad \forall u \in \mathscr{U}^i \,\forall a \in \mathscr{A}(u), \tag{5.1}$$

that is realization equivalent to Π.

Proof. This is a direct consequence of realization equivalence. In particular, consider player i sampling from $\Pi = \sum_{k=1}^{n} w_k B_k$. For $k = 1, ..., n$, let β_k be a behavioural strategy that is realization equivalent to B_k. Choose $u \in \mathscr{U}^i$ and $a \in \mathscr{A}(u)$. Then, when sampling from (Π, μ^{-i}), the probability of player i sampling action a in information state u is

$$\begin{aligned}
\mathbb{P}(u, a\,|\,\Pi, \mu^{-i}) &= \sum_{k=1}^{n} w_k x_{\beta_k}(\sigma_u)\beta_k(a\,|\,u) \sum_{s\in I^{-1}(u)} x_{\mu^{-i}}(\sigma_s), \\
&= \left(\sum_{k=1}^{n} w_k x_{\beta_k}(\sigma_u)\beta_k(a\,|\,u)\right)\left(\sum_{s\in I^{-1}(u)} x_{\mu^{-i}}(\sigma_s)\right),
\end{aligned}$$

where $x_{\mu^{-i}}(\sigma_s)$ is the product of fellow agents' and chance's realization probabilities based on their respective imperfect-information views of state s. Conditioning on reaching u yields

$$\mathbb{P}(a \mid u, \Pi, \mu^{-i}) \propto \sum_{k=1}^{n} w_k x_{\beta_k}(\sigma_u) \beta_k(a \mid u),$$

which is equivalent to the explicit update in Equation 4.1.

□

Hence, the behavioural strategy π^i can be learned approximately from a data set consisting of trajectories sampled from (Π^i, μ^{-i}). Recall that we can sample from $\Pi = \sum_{k=1}^{n} w_k B_k$ by sampling whole episodes from each constituent $\beta_k^i \equiv B_k^i$ with probability w_k.

5.4.2 Sampling Experience

In fictitious play, our goal is to keep track of the average mixed strategy profile

$$\Pi_{k+1} = \frac{1}{k+1} \sum_{j=1}^{k+1} B_j = \frac{k}{k+1} \Pi_k + \frac{1}{k+1} B_{k+1}.$$

Both Π_k and B_{k+1} are available at iteration k and we can therefore apply Proposition 5.4.1 to sample experience of a behavioural strategy π_{k+1}^i that is realization-equivalent to Π_{k+1}^i. In particular, for a player i we would repeatedly sample episodes of him following his strategy π_{k+1}^i against a fixed, completely mixed strategy profile of his opponents, μ^{-i}. Note, that player i can follow π_{k+1}^i by choosing between π_k^i and β_{k+1}^i at the root of the game with probabilities $\frac{k}{k+1}$ and $\frac{1}{k}$ respectively and committing to his choice for an entire episode. The player would experience sequences of state-action pairs, (s_t, a_t), where a_t is the action he chose at information state s_t following his strategy. This experience constitutes data that approximates the desired strategy that we want to learn, e.g. by fitting the data set with a function approximator. Alternatively, players can collect state-policy pairs, (s_t, ρ_t), where $\rho_t = \mathbb{E}_t \left[\left(1_{\{A_t=1\}}, \ldots, 1_{\{A_t=n\}} \right) \right]$ is the player's policy at state S_t.

Instead of resampling a whole new data set of experience at each iteration, we can incrementally update our data set from a stream of best responses, $\beta_j^i, j = 1, \ldots, k$. In order to constitute an unbiased approximation of an average of best responses, $\frac{1}{k}\sum_{j=1}^{k} B^i_j$, we need to accumulate the same number of sampled episodes from each B^i_j and these need to be sampled against the same fixed opponent strategy profile μ^{-i}. However, we suggest using the average strategy profile π_k as the (now varying) sampling distribution μ_k. Sampling against π_k has the benefit of focusing the updates on states that are more likely in the current strategy profile. When collecting samples incrementally, the use of a changing sampling distribution π_k can introduce bias. However, in fictitious play π_k is changing more slowly over time and thus it is conceivable for this bias to decay over time as well.

5.4.3 Memorizing Experience

In principle, collecting sampled data from an infinite stream of best responses would require an unbounded memory. To address this issue, we propose a table-lookup counting model and the use of reservoir sampling (Vitter, 1985) for general supervised learning.

Table-Lookup A simple, but possibly expensive, approach to memorizing experience of past strategies, is to count the number of times each action has been taken. For each sampled tuple, (s^i_t, a^i_t), we update the count of the chosen action.

$$N(s^i_t, a^i_t) \leftarrow N(s^i_t, a^i_t) + 1$$

Alternatively, we can accumulate local policies $\rho^i_t = \beta^i_t(s^i_t)$, where s^i_t is agent i's information state and β^i_t is the strategy that the agent pursued at this state when this experience was sampled.

$$\forall a \in \mathscr{A}(s_t) : N(s_t, a) \leftarrow N(s_t, a) + \rho^i_t(a)$$

The average strategy can be readily extracted.

$$\forall a \in \mathscr{A}(s_t) : \pi(s_t, a) \leftarrow \frac{N(s_t, a)}{N(s_t)}$$

As each sampled tuple, (s^i_t, a^i_t) or (s^i_t, ρ^i_t), only needs to be counted once, we can parse the experience of average strategies in an online fashion and thus avoid infinite memory requirements.

Reservoir In large domains, it is unfeasible to keep track of action counts at all information states. We propose to track a random sample of past best response behaviour with a finite-capacity memory by applying reservoir sampling to the stream of state-action pairs sampled from the stream of best responses.

Reservoir sampling (Vitter, 1985) is a class of algorithms for keeping track of a finite random sample from a possibly large or infinite stream of items, $\{x_k\}_{k\geq 1}$. Assume a reservoir (memory), $\mathscr{M}$, of size n. For the first n iterations, all items, $x_1,...,x_n$, are added to the reservoir. At iteration $k > n$, the algorithm randomly replaces an item in the reservoir with x_k with probability $\frac{n}{k}$, and discards x_k otherwise. Thus, at any iteration k, the reservoir contains a uniform random sample of the items that have been seen so far, $x_1,...,x_k$. Exponential reservoir sampling (Osborne et al., 2014) replaces items in the reservoir with a probability that is bounded from below, i.e. $\max(p, \frac{n}{k})$, where p is a chosen minimum probability. Thus, once $k = \frac{n}{p}$ items have been sampled, the items' likelihood of remaining in the reservoir decays exponentially.

Adding agents' experience of their (approximate) best responses to their respective supervised learning memories (reservoirs), $\mathscr{M}_{SL}$, with vanilla reservoir sampling approximates a $\frac{1}{T}$ step size of fictitious play, where T abstracts the iteration counter. Using exponential reservoir sampling results in an approximate step size of $\max(\alpha, \frac{1}{T})$, where α depends on the chosen minimum probabiltiy and capacity of the reservoir.

5.4.4 Average Strategy Approximation

Let $\tilde{\Pi}_k^i$ be an approximation of the average strategy profile $\Pi_k = \frac{1}{k}\sum_{j=1}^{k} B_j$, learned from a perfect fit of samples from the stream of best responses, $B_k, k \geq 1$, e.g. with the counting model. As we collect more samples from this stream, the approximation error, $\tilde{\Pi}_k^i - \Pi_k^i$, decays with increasing k. However, these approximation errors can bias the sequence of learned best responses, B_{k+1}, as they are trained with respect to the approximate average strategies, $\tilde{\Pi}_k$. While generalised weakened fictitious play only requires asymptotically-perfect best responses, this could slow down convergence to an impractical level. In Section 5.6.1, we empirically analyse the approximation error, $\tilde{\Pi}^i_k - \Pi^i_k$.

5.5 Algorithm

FSP can be implemented in a variety of ways, e.g. on- or off-policy and online or batch variants are possible. The key idea is that the two fictitious play operations, namely best response computation and average strategy updates, are implemented via reinforcement and supervised learning respectively. Which exact machine learning methods to use, is up to the user and can be tailored to the demands of the domain. It is critical though that one ensures that agents learn from appropriate experience, as discussed in Sections 5.3.2 and 5.4.2. Algorithm 5 presents FSP in this abstract generality.

Algorithm 5 Abstract Fictitious Self-Play

```
Initialize completely mixed average strategy profile $\pi_1$
for $k = 1, ..., K-1$ do
    for each player $i$ do
        Update approximate best response, $\beta^i_{k+1}$, via reinforcement learning from experience in the MDP induced by fellow agents' average strategy profile, $\mathcal{M}(\pi^{-i}_k)$
        Update average strategy, $\pi^i_{k+1}$, via supervised learning from experience of own historical best response behaviour, $\sum_{j=1}^{k+1} B^i_j$ or $\Pi^i_k + \frac{1}{k+1}\left(B^i_{k+1} - \Pi^i_k\right)$
    end for
end for
return $\pi_K$
```

In this chapter we restrict ourselves to batch reinforcement learning from memorized experience and a table-lookup, counting model to keep track of players' average strategies. We first discuss a general batch algorithm for FSP in Section 5.5.1 and then instantiate it with Q-learning and a table-lookup counting model in Section 5.5.2.

5.5.1 Batch

This section introduces a batch variant of FSP, presented in Algorithm 6. This variant separates the sampling of agents' experience to fill their memories from their learning from these memories. It does not specify particular off-policy reinforcement learning or supervised learning techniques, as these can be instantiated by a variety of algorithms.

Algorithm 6 Batch Fictitious Self-Play

```
Initialize completely mixed average strategy profile $\pi_1$
Initialize replay memories $\mathcal{M}_{RL}$ and $\mathcal{M}_{SL}$
Initialize the constant numbers of episodes that are sampled simultaneously, $n$, and alternatingly, $m$
for $k = 1, ..., K-1$ do
    SAMPLESIMULTANEOUSEXPERIENCE($n, \pi_k, \mathcal{M}_{RL}$)
    Each player $i$ updates their approximate best response strategy
        $\beta^i_{k+1} \leftarrow$ REINFORCEMENTLEARNING($\mathcal{M}^i_{RL}$)
    SAMPLEALTERNATINGEXPERIENCE($m, \pi_k, \beta_{k+1}, \mathcal{M}_{RL}, \mathcal{M}_{SL}$)
    Each player $i$ updates their average strategy
        $\pi^i_{k+1} \leftarrow$ SUPERVISEDLEARNING($\mathcal{M}^i_{SL}$)
end for
return $\pi_K$

function SAMPLESIMULTANEOUSEXPERIENCE($n$, $\pi$, $\mathcal{M}_{RL}$)
    Sample $n$ episodes from strategy profile $\pi$
    For each player $i$ store their experienced transitions, $(u^i_t, a_t, r_{t+1}, u^i_{t+1})$, in their reinforcement learning memory $\mathcal{M}^i_{RL}$
end function

function SAMPLEALTERNATINGEXPERIENCE($m, \pi, \beta, \mathcal{M}_{RL}, \mathcal{M}_{SL}$)
    for each player $i \in \mathcal{N}$ do
        Sample $m$ episodes from strategy profile $(\beta^i, \pi^{-i})$
        Store player $i$'s experienced transitions, $(u^i_t, a_t, r_{t+1}, u^i_{t+1})$, in their reinforcement learning memory $\mathcal{M}^i_{RL}$
        Store player $i$'s experienced own behaviour, $(u^i_t, a_t)$ or $(u^i_t, \beta^i(u^i_t))$, in their supervised learning memory $\mathcal{M}^i_{SL}$
    end for
end function
```

Reinforcement Learning For each agent i, the functions SAMPLESIMULTANEOUSEXPERIENCE and SAMPLEALTERNATINGEXPERIENCE both generate experience of play against fellow agents' average strategy profile, π^{-i}_k, at iteration k. As this is the strategy profile the agent wants to learn to best respond to, all this experience is added to the agent's reinforcement learning memory, $\mathcal{M}^i_{RL}$. Thus, an approximate best response to π^{-i}_k can be learnt via off-policy reinforcement learning from the memory. In this work, we implement the memories with circular buffers, which produce sliding-window approximations as discussed in Section 5.3.3.

Supervised Learning In order to enable agents to learn their own average strategies, the function SAMPLEALTERNATINGEXPERIENCE also generates experience of the agents' own best response behaviour. This experience is added to the agents' respective supervised learning memories, $\mathcal{M}^i_{SL}, i = 1,...,N$. For each agent a fixed number (m) of episodes is added from each of their past best responses. Thus, at iteration k an agent i's supervised learning memory approximates its historical, average strategy, $\sum_{j=1}^{k+1} B^i_j$.

5.5.2 Table-lookup

Algorithm 7 instantiates Batch FSP with table-lookup methods. In particular, to learn approximate best responses we use Q-learning with experience replay on the reinforcement learning memories. Given (approximately) evaluated Q-values, a Boltzmann distribution over these values yields an approximate best response. To keep track of the average strategies, we aggregate the experience from the supervised learning memories into a counting model that was introduced in Section 5.4.3.

5.6 Experiments

This section presents experiments with table-lookup FSP. We begin by investigating the approximation errors induced by our learning- and sample-based approach. We then compare FSP to (full-width) XFP to illustrate the benefits of sampling and learning from experience.

5.6.1 Empirical Analysis of Approximation Errors

Section 5.3.4 expressed concerns about the quality of approximate best responses that are produced by reinforcement learning with a finite computational budget. Here, we empirically investigate the quality of approximate best responses that table-lookup FSP produces with Q-learning. In particular, we measure the optimality gap of the approximate ε-best responses, β_{k+1}, to the average strategy profiles, π_k,

$$\varepsilon = \frac{\sum_{i=1}^{2} R^i\left(\mathrm{BR}(\pi_k^{-i}), \pi_k^{-i}\right) - R^i\left(\beta^i_{k+1}, \pi_k^{-i}\right)}{2}. \tag{5.2}$$

Algorithm 7 Table-lookup Batch FSP with Q-learning and counting model

```
Initialize Q-learning parameters, e.g. learning stepsize
Initialize all agents' table-lookup action-values, ${Q^i}_{i \in \mathcal{N}}$
Intiialize all agents' table-lookup counting models, ${N^i}_{i \in \mathcal{N}}$
Initialize and run Algorithm 6 (Batch FSP) with REINFORCEMENTLEARNING
and SUPERVISEDLEARNING methods below

function REINFORCEMENTLEARNING($\mathcal{M}^i_{RL}$)
    Restore previous iteration's $Q^i$-values
    Update (decay) learning stepsize and Boltzmann temperature
    Learn updated $Q^i$-values with Q-learning from $\mathcal{M}^i_{RL}$
    return Boltzmann[$Q^i$]
end function

function SUPERVISEDLEARNING($\mathcal{M}^i_{SL}$)
    Restore previous iteration's counting model, $N^i$, and average strategy, $\pi^i$
    for each $(u_t, \rho_t)$ in $\mathcal{M}^i_{SL}$ do
        $\forall a \in \mathcal{A}(u_t) : N^i(u_t, a) \leftarrow N^i(u_t, a) + \rho_t(a)$
        $\forall a \in \mathcal{A}(u_t) : \pi^i(u_t, a) \leftarrow \frac{N^i(u_t,a)}{N^i(u_t)}$
    end for
    Empty $\mathcal{M}^i_{SL}$
    return $\pi^i$
end function
```

Figure 5.1 shows the quality of the best responses improve over time. As we use a fixed computational budget per iteration, this demonstrates that knowledge is indeed transferred between iterations. It appears that Q-learning can keep up with the slower changing average strategy profiles it learns against.

Section 5.4.4 expressed concerns about the quality of approximate average strategies that are produced from a finite amount of samples from each constituent (past) best response. Here, we empirically investigate the resulting approximation error. In particular, we additionally track a perfect average strategy, which we compute by periodically averaging the respective best responses according to XFP's full-width update (see Theorem 4.4.1). Note that these perfect average strategies do not affect FSP in any way; they are only computed for evaluation purposes. We then compute the squared distance (error) between this perfect average and the approximate average strategy contained in the counting model of table-lookup FSP. Figure 5.2 shows the approximate average strategy profile approach the quality of several

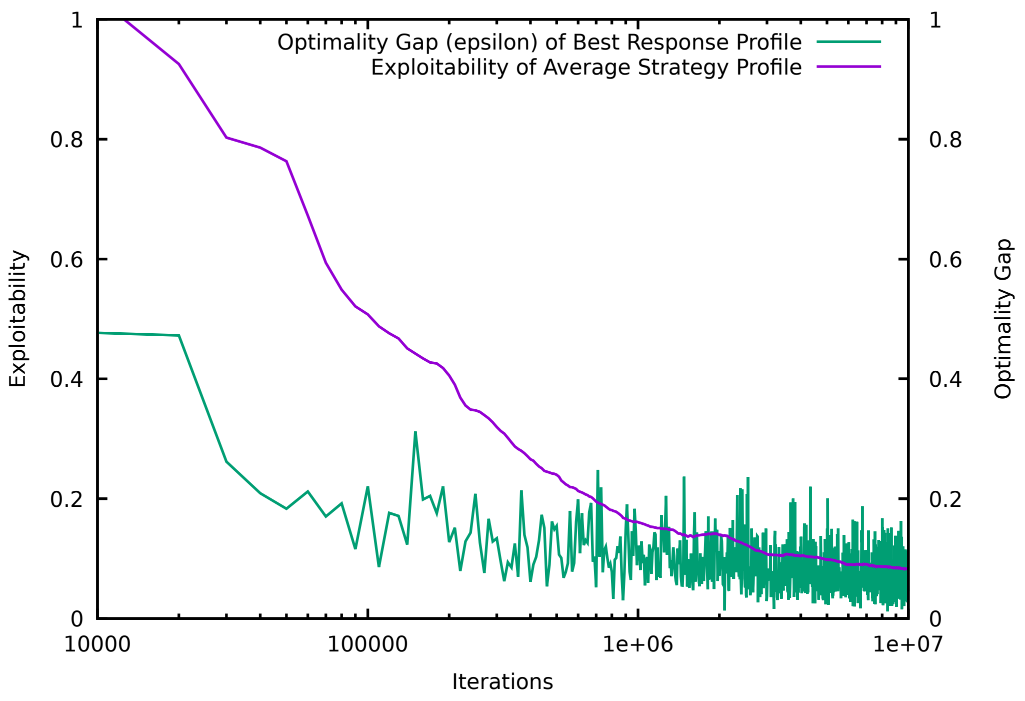


Figure 5.1: Analysis of the best response approximation quality of table-lookup FSP in Leduc Hold'em.

perfect average strategies, which were computed at intervals of 10, 100 and 1000 iterations respectively. This suggests that, in principle, we can track and learn the average strategy of fictitious play from sampled experience.

5.6.2 Sample-Based Versus Full-Width

We tested the performance of FSP Algorithm 7 with a fixed computational budget per iteration and evaluated how it scales to larger games in comparison to XFP.

We manually tuned the FSP parameters in preliminary experiments on 6-card Leduc Hold'em. In particular, we varied single or small subsets of parameters at a time and measured the resulting achieved exploitability. The best results were achieved with the following calibration, which we used in all subsequent experiments and games presented in this section. In particular, for each player i, we used a replay memory, $\mathscr{M}_{RL}^{i}$, with space for 40000 episodes. We sampled 2 episodes simultaenously from strategy profile π and 1 episode alternatingly from (β^i, π^{-i}) at each iteration for each player respectively, i.e. we set $n = 2$ and $m = 1$ in algorithm 6. At each iteration k each agent replayed 30 episodes with Q-learning stepsize

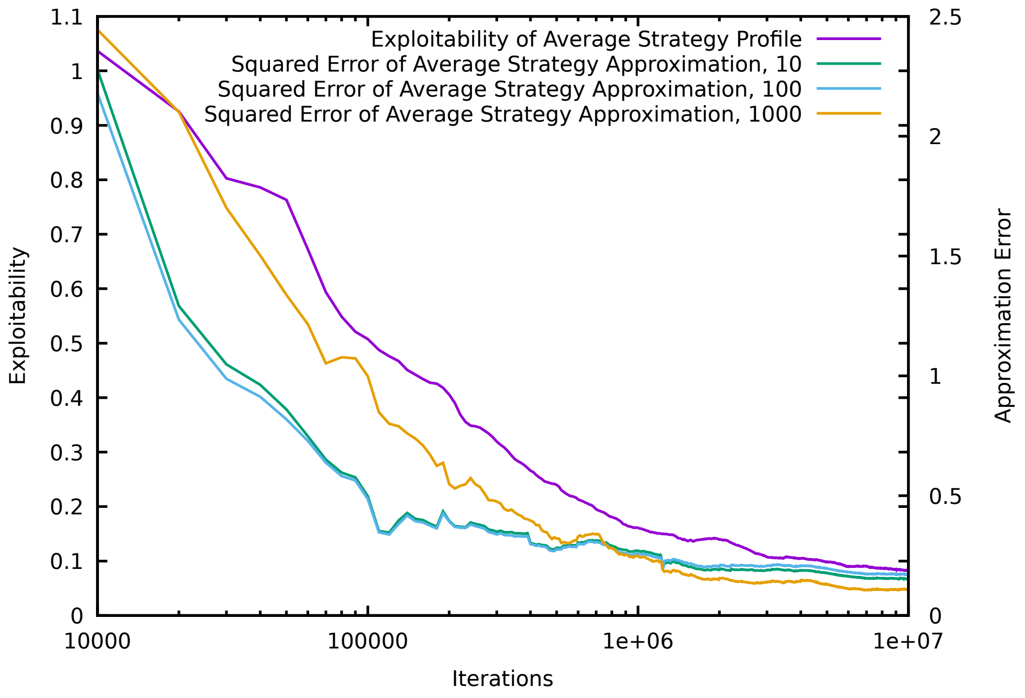


Figure 5.2: Analysis of the average strategy approximation quality of table-lookup FSP in Leduc Hold'em.

$\frac{0.05}{1+0.003\sqrt{k}}$. It returned an (approximate) best response policy that at each information state was determined by a Boltzmann distribution over the estimated Q-values, using temperature $(1+0.02\sqrt{k})^{-1}$. The Q-value estimates were maintained across iterations, i.e. each iteration started with the Q-values of the previous iteration.

We compared XFP and FSP in parameterizations of Leduc Hold'em and River poker. In each experiment, both algorithms' initial average strategy profiles were initialized to a uniform distribution at each information state. Each algorithm trained for 300 seconds. The average strategy profiles' exploitability was measured at regular intervals.

Leduc Hold'em Figure 5.3 compares both algorithms' performance in 6- and 60-card Leduc Hold'em, where the card decks consist of 3 and 30 ranks with 2 suits respectively. While XFP clearly outperformed FSP in the small 6-card variant, in the larger 60-card Leduc Hold'em it learned more slowly. This might be expected, as the computation per iteration of XFP scales linearly in the squared number of cards. FSP, on the other hand, operates only on information states whose number

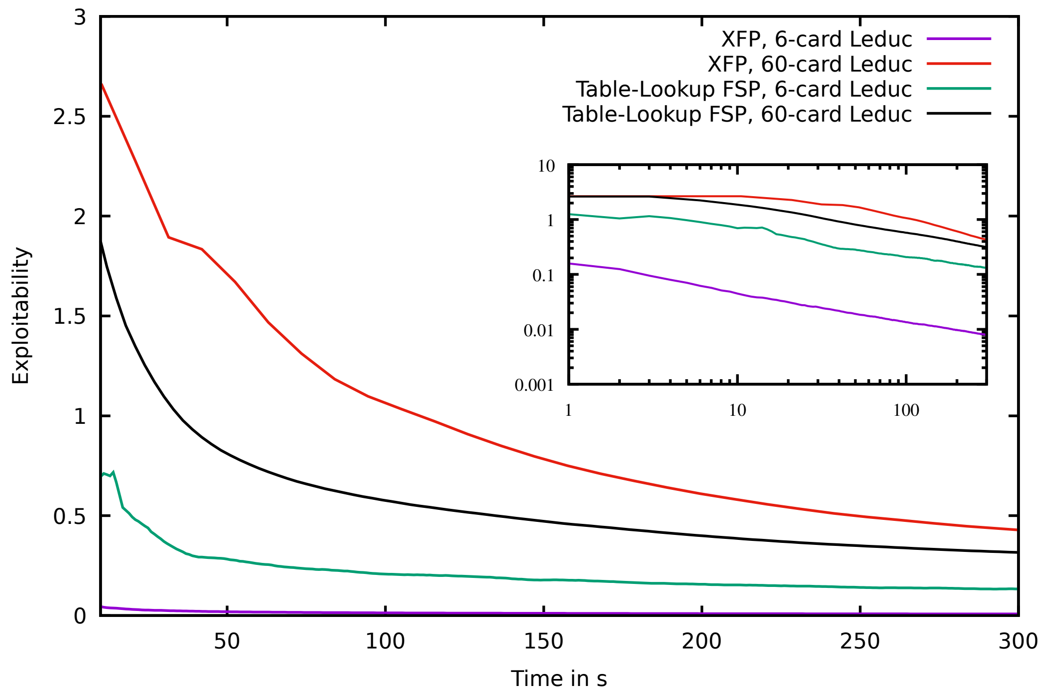


Figure 5.3: Comparison of XFP and table-lookup FSP in Leduc Hold'em with 6 and 60 cards. The inset presents the results using a logarithmic scale for both axes.

scales linearly with the number of cards in the game.

River Poker We compared the algorithms in two instances of River Poker, a simplified model of an end-game situation in Limit Texas Hold'em. We set the potsize to 6, allowed at most 1 raise and used a fixed set of community cards[1]. The first instance assumes uninformed, uniform beliefs of the players that assigns equal probability to each possible holding. The second instance assumes that players have inferred beliefs over their opponents' holdings. An expert poker player has provided us with belief distributions that model a real Texas Hold'em scenario, where the first player called a raise preflop, check/raised on the flop and bet the turn. The distributions assume that player 1 holds one of 14% of the possible holdings [2] with probability 0.99 and a uniform random holding with probability 0.01. Similarly, player 2 is likely to hold one of 32% holdings [3].

According to figure 5.4, FSP improved its average strategy profile much faster

[1]KhTc7d5sJh

[2]K4s-K2s,KTo-K3o,QTo-Q9o,J9o+,T9o,T7o,98o,96o

[3]QQ-JJ,99-88,66,AQs-A5s,K6s,K4s-K2s,QTs,Q7s,JTs,J7s,T8s+,T6s-T2s,97s,87s,72s+,AQo-A5o,K6o,K4o-K2o,QTo,Q7o,JTo,J7o,T8o+,T6o-T4o,97o,87o,75o+

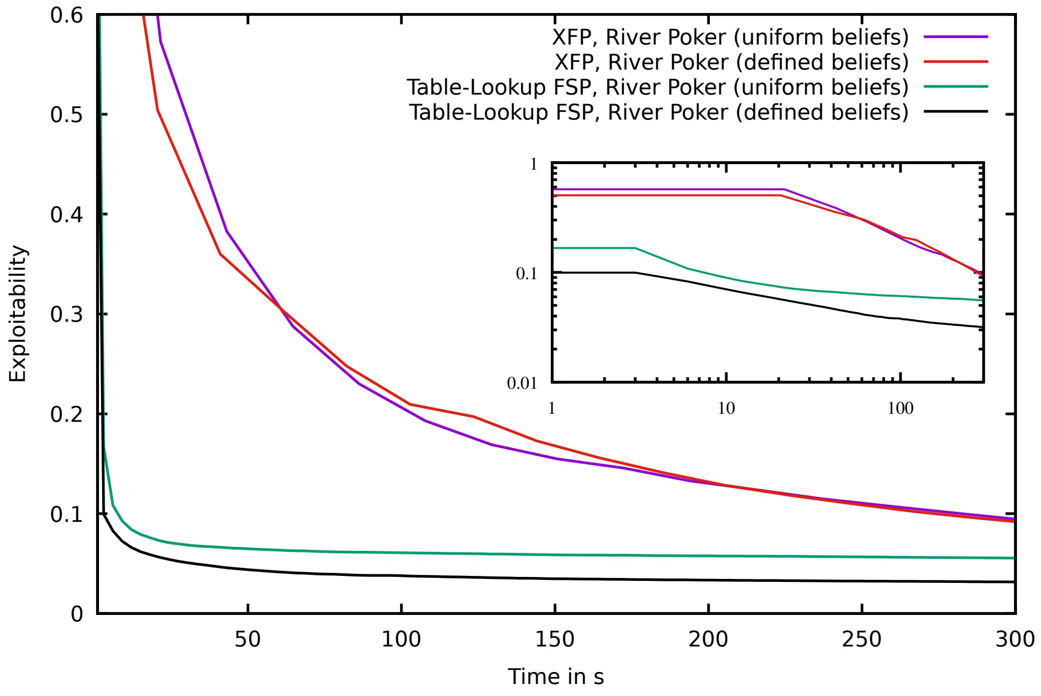


Figure 5.4: Comparison of XFP and table-lookup FSP in River poker. The inset presents the results using a logarithmic scale for both axes.

than the full-width variant in both instances of River poker. In River poker with defined beliefs, FSP obtained an exploitability of 0.05 after 33 seconds, whereas after 300 seconds XFP was exploitable by more than 0.09. Furthermore, XFP's performance was similar in both instances of River poker, whereas FSP lowered its exploitability by more than 40%, when defined beliefs were used.

River poker has about 10 million states but only around 4000 information states. For a similar reason as in the Leduc Hold'em experiments, this might explain the overall better performance of FSP. Furthermore, the structure of the game assigns non-zero probability to each state of the game and thus the computational cost of XFP is the same for both instances of River poker. It performs computation at each state no matter how likely it is to occur. FSP on the other hand is guided by sampling and is therefore able to focus its computation on likely scenarios. This allows it to benefit from the additional structure introduced by the players' beliefs into the game.

5.7 Conclusion

In this chapter we have introduced FSP, a sample-based machine learning approach to learning Nash equilibria from self-play. FSP is designed to approximate extensive-form fictitious play. It can be instantiated with a variety of classic, single-agent reinforcement learning and standard supervised learning methods.

Our experiments on two poker games have shown that sample-based FSP outperformed full-width XFP in games with a large number of cards as well as non-uniform belief distributions. We have attributed this success to two factors. Firstly, reinforcement learning agents only observe their respective information states and thus implicitly average over states that they can't distinguish. XFP, which scales in the number of games states rather than information states, can be more adversely affected by an increase in the size of the game. Secondly, for non-uniform belief distributions some states of the game can be more likely and important. A sample-based approach can exploit this by sampling and focusing on these potentially more important states.

While FSP's strategy updates would asymptotically converge to the desired strategies, it remains an open question whether guaranteed convergence can be achieved with a finite computational budget per iteration. However, we have presented some intuition why this might be the case and our experiments provide first empirical evidence that the resulting approximation erors indeed decay in practice.

This chapter's table-lookup methods have barely tapped into FSP's potential. Function approximation can provide automated abstraction and generalisation in large extensive-form games. Continuous-action reinforcement learning can learn best responses in continuous action spaces. FSP may, therefore, scale to large and even continuous-action game-theoretic applications. The next chapter provides evidence of this claim, by combining FSP with deep learning techniques and successfully applying it to a large-scale game.

Chapter 6

Practical Issues in Fictitious Self-Play

In this chapter we address our main research question (see Section 1.2).

6.1 Introduction

Common game-theoretic methods (Section 2.3.4), XFP (Section 4.4) and table-lookup FSP (Section 5.5.2) lack the ability to learn abstract patterns and use them to generalise across related information states. This results in limited scalability to large games, unless the domain is abstracted to a manageable size, typically using human expert knowledge, heuristics or modelling. This domain knowledge is expensive and hard to come by. Thus, relying on it prevents self-play learning methods from scaling at the pace of computational technology.

In this chapter we introduce NFSP, a deep reinforcement learning method for learning approximate Nash equilibria of imperfect-information games without prior domain knowledge. NFSP combines FSP with neural network function approximation. We begin by exploring fictitious play with anticipatory dynamics. These dynamics enable fictitious self-players to learn simultaneously by letting them choose a mixture of their average strategy and best response strategy. We then introduce the Online FSP Agent, an online reinforcement learner for FSP. Finally, we set up NFSP as a multi-agent learning domain of simultaneously learning Online FSP Agents that use neural networks to represent their average and best-response policies. These agents train their networks by deep learning from memorized experience. We empirically evaluate the convergence of NFSP in Leduc Hold'em and compare it to DQN's performance. We also apply NFSP to LHE, a game of real-world scale, and visualize the neural representations (abstractions) it has learned.

6.2 Simultaneous Learning

The Batch FSP Algorithm 6 required a meta controller that coordinated FSP agents so that that they appropriately sampled experience to approximate fictitious play. From an artifical intelligence and multi-agent learning perspective, this is somewhat unappealing. If individual agents could autonomously learn a Nash equilibrium from simultaneous experience of their uncoordinated interaction, this would provide an example of how such strategies might arise in practice. Furthermore, there are also practical advantages to simultaneous learning. First, sampling experience simultaneously rather than alternatingly, in principle, is n times more sample efficient, where n is the number of agents. Second, simultaneously learning agents can be directly applied to a real-world black-box environment, such as a city's traffic lights system.

If we want all agents to learn simultaneously while playing against each other, we face the following dilemma. In principle, each agent could play its average policy, π, and learn a best response with off-policy Q-learning, i.e. evaluate and maximise its action values, $Q^i(s,a) \approx \mathbb{E}_{\beta^i,\pi^{-i}}\left[G_t^i \,\middle|\, S_t = s, A_t = a\right]$, of playing its best response policy β^i against its fellow agents' average strategy profile, π^{-i}. However, in this case the agent would not generate any experience of its own best response behaviour, β^i, which is needed to train its average policy, π^i, that approximates the agent's average of past best responses.

To address this problem, we propose an approximation of **anticipatory dynamics** of continuous-time dynamic fictitious play (Shamma and Arslan, 2005). In this variant of fictitious play, players choose best responses to a short-term prediction of their fellow agents' (anticipated) average normal-form strategies, $\Pi_t^{-i} + \eta \frac{d}{dt}\Pi_t^{-i}$, where $\eta \in \mathbb{R}$ is the **anticipatory parameter**. The authors show that for appropriate, game-dependent choice of η stability of fictitious play at equilibrium points can be improved. We use $B_{k+1}^i - \Pi_k^i \approx \frac{d}{dk}\Pi_k^i$ as a discrete-time approximation of the derivative that is used in the anticipatory dynamics. Note that $\Delta\Pi_k^i \propto B_{k+1}^i - \Pi_k^i$ is indeed the normal-form update direction of common discrete-time fictitious play.

6.3 Online FSP Agent

We introduce the Online FSP Agent, an online reinforcement learner for simultaneous FSP with anticipatory dynamics. Online FSP agents choose their actions from the mixture policy $\sigma \equiv (1-\eta)\Pi + \eta B$. This enables each agent to compute an approximate best response, β^i, to its fellow agents' anticipated average strategy profile, $\sigma^{-i} \equiv \Pi^{-i} + \eta(B^{-i} - \Pi^{-i})$, by iteratively evaluating and maximising their action values, $Q^i(s,a) \approx \mathbb{E}_{\beta^i,\sigma^{-i}}\left[G_t^i \,\middle|\, S_t = s, A_t = a\right]$. Additionally, as each agent's own best response policy is now sampled (in proportion to the anticipatory parameter), they can now learn their average policy from that experience. Algorithm 8 presents a general Online FSP Agent, that can be instantiated with a variety of reinforcement and supervised learning methods.

6.4 Neural Fictitious Self-Play

.

NFSP deploys individual deep-learning Online FSP Agents to represent the self-playing intelligence from all players' points of view. It implements a multi-agent interpretation of self-play.

Deep Learning Best Responses DQN is an off-policy deep reinforcement learning approach for MDP environments (Mnih et al., 2015). As before (see FSP Chapter 5), we can directly plug the algorithm into the FSP framework and thus utilise it to learn approximate best responses.

DQN agents learn an ε-greedy policy by deep fitted Q-learning from replayed experience (Mnih et al., 2015). Similarly, FSP agents can learn approximate best responses with the DQN algorithm applied to their reinforcement learning memories. In particular, the agent trains a neural network, $Q(s,a;\theta^Q)$, to predict action values from experience of play against its fellow agents' (recent) anticipated average

Algorithm 8 Online FSP Agent

```
Initialize replay memories $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir)
Initialize average policy $\pi$
Initialize best-response policy $\beta$
Initialize anticipatory parameter $\eta$
for each episode do
    Set policy $\sigma \leftarrow \begin{cases} \beta, & \text{with probability } \eta \\ \pi, & \text{with probability } 1-\eta \end{cases}$
    Observe initial information state $s_1$ and reward $r_1$
    for $t = 1, T$ do
        Sample action $a_t$ from policy $\sigma$
        Execute action $a_t$ in environment and observe reward $r_{t+1}$ and next information state $s_{t+1}$
        Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reinforcement learning memory $\mathcal{M}_{RL}$
        if following best response policy $\sigma = \beta$ then
            Store behaviour tuple $(s_t, a_t)$ in supervised learning memory $\mathcal{M}_{SL}$
        end if
        Update $\pi$ with supervised learning on experience of own average behaviour, $\mathcal{M}_{SL}$
        Update $\beta$ with reinforcement learning on experience of interactions with environment, $\mathcal{M}_{RL}$
    end for
end for
```

strategies. This is achieved with stochastic gradient descent on the mean squared error,

$$\mathcal{L}\left(\theta^Q\right) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s',a';\theta^{Q'}) - Q(s,a;\theta^Q)\right)^2\right], \tag{6.1}$$

where $\theta^{Q'}$ are the fitted network parameters, i.e. we periodically update $\theta^{Q'} \leftarrow \theta^Q$. The resulting network defines the agent's approximate best response strategy,

$$\beta = \varepsilon\text{-greedy}\left[Q(\cdot\,;\theta^Q)\right], \tag{6.2}$$

which selects a random action with probability ε and otherwise chooses the action that maximises the predicted action values.

Deep Learning Average Strategies Multinomial logistic regression is a common technique for learning from demonstrations (Argall et al., 2009). We apply this method to learning average policies from the agents' respective supervised learning memories. These memories effectively contain demonstrations produced by the agents' own past best responses.

The agent's average policy is represented by a neural network, $\Pi(s,a;\theta^{\Pi})$, that maps states to a Softmax output of action probabilities. The network is trained to imitate the agent's own past best response behaviour by multinomial logistic regression applied to its supervised learning memory, $\mathscr{M}_{SL}$. In particular, we use stochastic gradient descent on the negative log-probability of past actions taken,

$$\mathscr{L}(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathscr{M}_{SL}}\left[-\log\Pi(s,a;\theta^{\Pi})\right]. \tag{6.3}$$

The agent's average policy is defined by the network's Softmax outputs,

$$\pi(a\,|\,s) = \Pi(s,a;\theta^{\Pi}). \tag{6.4}$$

Algorithm NFSP, presented in Algorithm 9, executes deep-learning instances of the Online FSP Agent (Algorithm 8) for each player in the game. As these agents interact, they learn their own average strategies and (approximate) best responses to their fellow agents' anticipated average strategies. Thus, NFSP is a simultaneous self-play instance of FSP (Algorithm 5).

6.5 Encoding a Poker Environment

Observations One of our goals is to minimise reliance on prior knowledge. Therefore, we attempt to define an objective encoding of information states in poker games. In particular, we do not engineer any higher-level features. Poker games usually consist of multiple rounds. At each round new cards are revealed to the players. We represent each rounds' cards by a k-of-n encoding. For example, LHE has a card deck of 52 cards and on the second round three cards are revealed. Thus, this round is encoded with a vector of length 52 and three elements set to 1 and the rest to 0. In Limit Hold'em poker games, players usually have three actions to choose from, namely {fold, call, raise}. Note that depending on context, calls and

Algorithm 9 Neural Fictitious Self-Play (NFSP) with DQN

```
Initialize game $\Gamma$ and execute an agent via RunAgent for each player in the game
function RunAgent($\Gamma$)
    Initialize replay memories $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir)
    Initialize average-policy network $\Pi(s,a;\theta^{\Pi})$ with random parameters $\theta^{\Pi}$
    Initialize action-value network $Q(s,a;\theta^{Q})$ with random parameters $\theta^{Q}$
    Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
    Initialize anticipatory parameter $\eta$
    for each episode do
        Set policy $\sigma \leftarrow \begin{cases} \epsilon\text{-greedy}(Q), & \text{with probability } \eta \\ \Pi, & \text{with probability } 1-\eta \end{cases}$
        Observe initial information state $s_1$ and reward $r_1$
        for $t = 1, T$ do
            Sample action $a_t$ from policy $\sigma$
            Execute action $a_t$ in game and observe reward $r_{t+1}$ and next information state $s_{t+1}$
            Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reinforcement learning memory $\mathcal{M}_{RL}$
            if agent follows best response policy $\sigma = \epsilon\text{-greedy}(Q)$ then
                Store behaviour tuple $(s_t, a_t)$ in supervised learning memory $\mathcal{M}_{SL}$
            end if
            Update $\theta^{\Pi}$ with stochastic gradient descent on loss
                $\mathcal{L}(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log\Pi(s,a;\theta^{\Pi})\right]$
            Update $\theta^{Q}$ with stochastic gradient descent on loss
                $\mathcal{L}\left(\theta^{Q}\right) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s',a';\theta^{Q'}) - Q(s,a;\theta^{Q})\right)^2\right]$
            Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
        end for
    end for
end function
```

raises can be referred to as checks and bets respectively. Betting is capped at a fixed number of raises per round. Thus, we can represent the betting history as a tensor with 4 dimensions, namely $\{$player, round, number of raises, action taken$\}$. E.g. heads-up LHE contains 2 players, 4 rounds, 0 to 4 raises per round and 3 actions. Thus we can represent a LHE betting history as a $2 \times 4 \times 5 \times 3$ tensor. In a heads-up game we do not need to encode the fold action, as a two-player game always ends if one player gives up. Thus, we can flatten the 4-dimensional tensor to a vector of length 80. Concatenating with the card inputs of 4 rounds, we encode an information state of LHE as a vector of length 288. Similarly, an information state of Leduc Hold'em can be encoded as a vector of length 30, as it contains 6 cards with 3 duplicates, 2 rounds, 0 to 2 raises per round and 3 actions.

Actions Fixed-limit poker variants limit the number of players' bets per round. Thus, a raise is illegal once such limit is reached. Furthermore, a fold may be considered illegal if the agent does not face any prior bet, e.g. common poker software deactivates this action for human players. We address these issues by modifying the environment. In particular, an illegal raise or fold is automatically results in a call. Thus, the environment is effectively offering two duplicate actions to the agent in those cases.

Rewards We directly map players' monetary payoffs to rewards, i.e. a loss of 1 monetary unit corresponds to a reward of -1. The agents' betting during an episode yield intermediate rewards. For example, if an agent bet 2 units on the flop, this would result in an immediate reward of -2, which the agent would experience together with its observation of the next information state. Alternatively, total rewards could be distributed only at the end of an episode.

6.6 Experiments

We evaluate NFSP and related algorithms in Leduc and Limit Texas Hold'em poker games.

6.6.1 Leduc Hold'em

We empirically investigate the convergence of NFSP to Nash equilibria in Leduc Hold'em. We also study whether removing or altering some of NFSP's components breaks convergence.

First, we explored high-level architectural and parameter choices for NFSP in preliminary experiments on Leduc Hold'em. In this initial exploration we found rectified linear activations to perform better than alternative activation functions, e.g. tanh or sigmoid. Furthermore, network architectures with one hidden layer of 64 neurons appeared to be a sweet spot that fabourably traded off network size and resulting computational cost against achieved performance, i.e. low exploitability. Second, we manually tuned the NFSP parameters in further preliminary experiments on Leduc Hold'em for a fully connected neural network with 1 hidden layer of 64 neurons and rectified linear activations. In particular, we varied single or small subsets of parameters at a time and measured the resulting achieved exploitability. We fixed the best calibration we could find and then repeated learning in Leduc Hold'em for various network architectures with the same parameters. In particular, we set the sizes of memories to 200k and 2m for $\mathscr{M}_{RL}$ and $\mathscr{M}_{SL}$ respectively. $\mathscr{M}_{RL}$ functioned as a circular buffer containing a recent window of experience. $\mathscr{M}_{SL}$ was updated with reservoir sampling (Vitter, 1985). The reinforcement and supervised learning rates were set to 0.1 and 0.005, and both used vanilla SGD without momentum for stochastic optimisation of the neural networks. Each agent performed 2 stochastic gradient updates of mini-batch size 128 per network for every 128 steps in the game. The target network of the DQN algorithm was refitted every 300 updates. NFSP's anticipatory parameter was set to $\eta = 0.1$. The ε-greedy policies' exploration started at 0.06 and decayed to 0, proportionally to the inverse square root of the number of iterations.

Figure 6.1 shows NFSP approaching Nash equilibria for various network architectures. We observe a monotonic performance increase with size of the networks. NFSP achieved an exploitability of 0.06, which full-width XFP typically achieves after around 1000 full-width iterations.

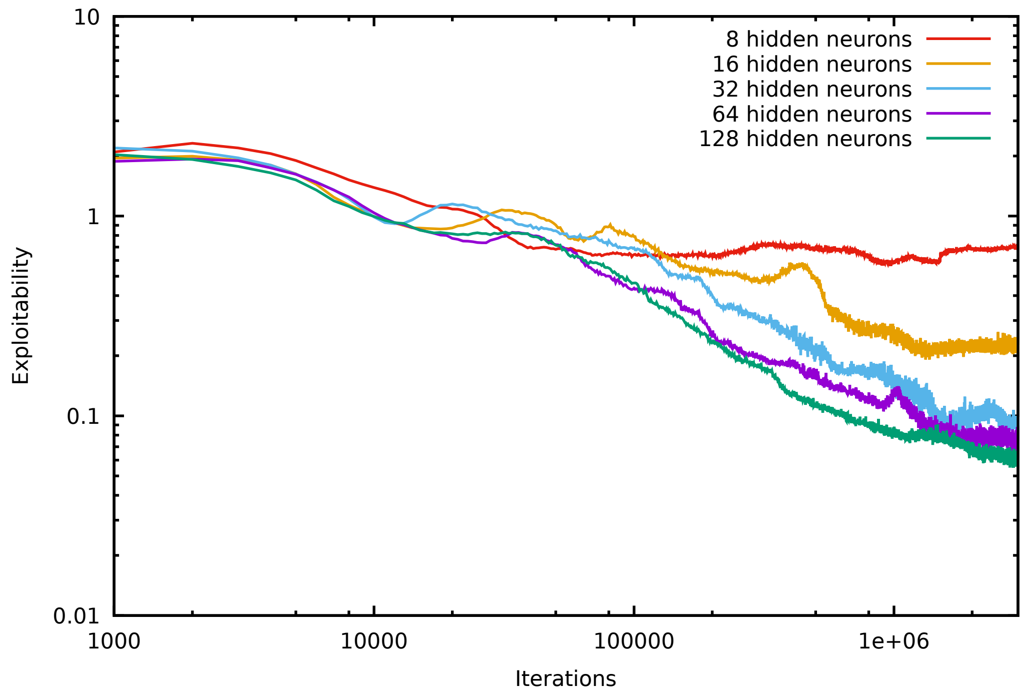


Figure 6.1: Learning performance of NFSP in Leduc Hold'em for various network sizes.

In order to investigate the relevance of various components of NFSP, e.g. reservoir sampling and anticipatory dynamics, we conducted an experiment that isolated their effects. Figure 6.2 shows that these modifications led to decremental performance. In particular, using a fixed-size sliding window to store experience of the agents' own behaviour led to divergence. NFSP's performance plateaued for a high anticipatory parameter of 0.5, that most likely violates the game-dependent stability conditions of dynamic fictitious play (Shamma and Arslan, 2005). Finally, using exponentially-averaged reservoir sampling for supervised learning memory updates led to noisy performance.

6.6.2 Comparison to DQN

Several stable algorithms have previously been proposed for deep reinforcement learning, notably the DQN algorithm (Mnih et al., 2015). However, the empirical stability of these algorithms was only previously established in single-agent, perfect (or near-perfect) information MDPs. Here, we investigate the stability of DQN in multi-agent, imperfect-information games, in comparison to NFSP.

DQN learns a deterministic, greedy strategy. Such strategies are sufficient to

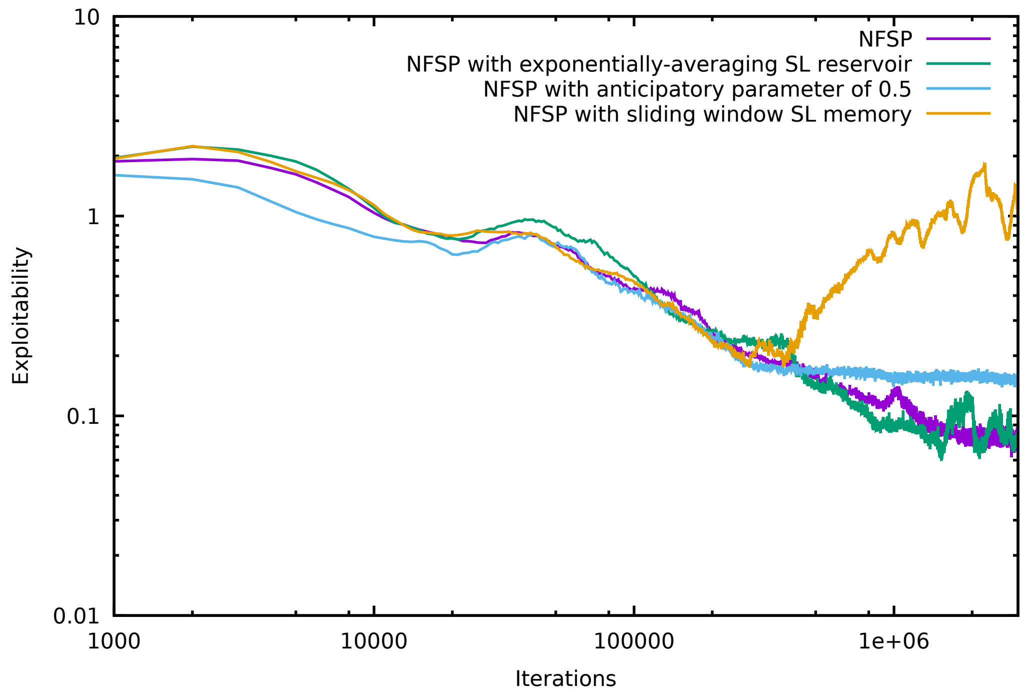


Figure 6.2: Breaking learning performance in Leduc Hold'em by removing essential components of NFSP.

behave optimally in single-agent domains, i.e. MDPs for which DQN was designed. However, imperfect-information games generally require stochastic strategies to achieve optimal behaviour. One might wonder if the average behaviour of DQN converges to a Nash equilibrium. To test this, we augmented DQN with a supervised learning memory and train a neural network to estimate its average strategy. Unlike NFSP, the average strategy does not affect the agent's behaviour in any way; it is passively observing the DQN agent to estimate its evolution over time. We implement this variant of DQN by using NFSP with an anticipatory parameter of $\eta = 1$. We trained DQN with all combinations of the following parameters: Learning rate $\{0.2, 0.1, 0.05\}$, decaying exploration starting at $\{0.06, 0.12\}$ and reinforcement learning memory $\{\text{2m reservoir}, \text{2m sliding window}\}$. Other parameters were fixed to the same values as NFSP; note that these parameters only affect the passive observation process. We then chose the best-performing result and compared to NFSP's performance that was achieved in the previous section's experiment. DQN achieved its best-performing result with a learning rate of 0.2, exploration starting

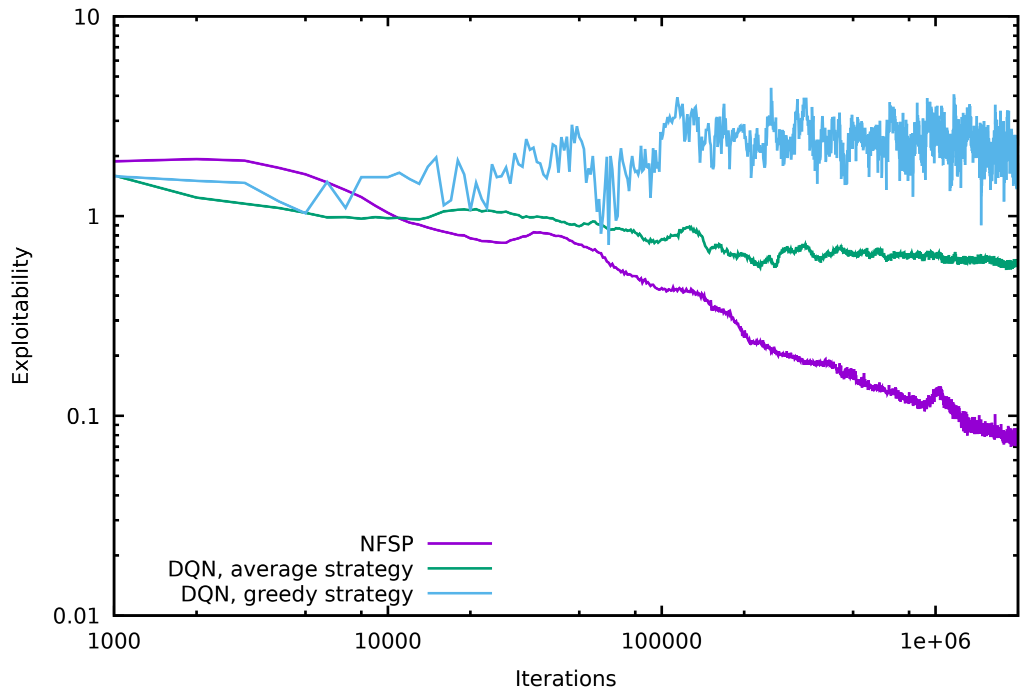


Figure 6.3: Comparing performance to DQN in Leduc Hold'em.

at 0.12 and a sliding window memory of size 2m.

Figure 6.3 shows that DQN's deterministic strategy is highly exploitable, which is expected as imperfect-information games usually require stochastic policies. DQN's average behaviour does not approach a Nash equilibrium either. In principle, DQN also learns a best-response to the historical experience generated by fellow agents. So why does it perform worse than NFSP? The problem is that DQN agents exclusively generate self-play experience according to their ε-greedy strategies. These experiences are both highly correlated over time, and highly focused on a narrow distribution of states. In contrast, NFSP agents use an ever more slowly changing (anticipated) average policy to generate self-play experience. Thus, their experience varies more smoothly, resulting in a more stable data distribution, and therefore more stable neural networks. Note that this issue is not limited to DQN; other common reinforcement learning methods have been shown to exhibit similarly stagnating performance in poker games, e.g. UCT (see Section 3.5.2).

6.6.3 Limit Texas Hold'em

We applied NFSP to LHE, a game that is popular with humans. Since in 2008 a computer program beat expert human LHE players for the first time in a public competition, modern computer agents are widely considered to have achieved superhuman performance (Newall, 2013). The game was *essentially solved* by Bowling et al. (2015). We evaluated our agents against the top 3 computer programs of the most recent (2014) ACPC that featured LHE. Learning performance was measured in milli-big-blinds won per hand, mbb/h, i.e. one thousandth of a big blind that players post at the beginning of a hand.

We successively tried 9 configurations of NFSP in LHE. The configurations' parameters were chosen based on intuition and experience from the respective prior runs in LHE as well as our experiments in Leduc Hold'em. We achieved the best performance with the following parameters. The neural networks were fully connected with four hidden layers of $1024, 512, 1024$ and 512 neurons with rectified linear activations. The memory sizes were set to 600k and 30m for $\mathscr{M}_{RL}$ and $\mathscr{M}_{SL}$ respectively. $\mathscr{M}_{RL}$ functioned as a circular buffer containing a recent window of experience. $\mathscr{M}_{SL}$ was updated with exponentially-averaged reservoir sampling (Osborne et al., 2014), replacing entries in $\mathscr{M}_{SL}$ with minimum probability 0.25. We used vanilla SGD without momentum for both reinforcement and supervised learning, with learning rates set to 0.1 and 0.01 respectively. Each agent performed 2 stochastic gradient updates of mini-batch size 256 per network for every 256 steps in the game. The target network was refitted every 1000 updates. NFSP's anticipatory parameter was set to $\eta = 0.1$. The ε-greedy policies' exploration started at 0.08 and decayed to 0, more slowly than in Leduc Hold'em. In addition to NFSP's main, average strategy profile we also evaluated the best response and greedy-average strategies, which deterministically choose actions that maximise the predicted action values or probabilities respectively.

To provide some intuition for win rates in heads-up LHE, a player that always folds will lose 750 mbb/h, and expert human players typically achieve expected win rates of 40-60 mbb/h at online high-stakes games. Similarly, the top half of

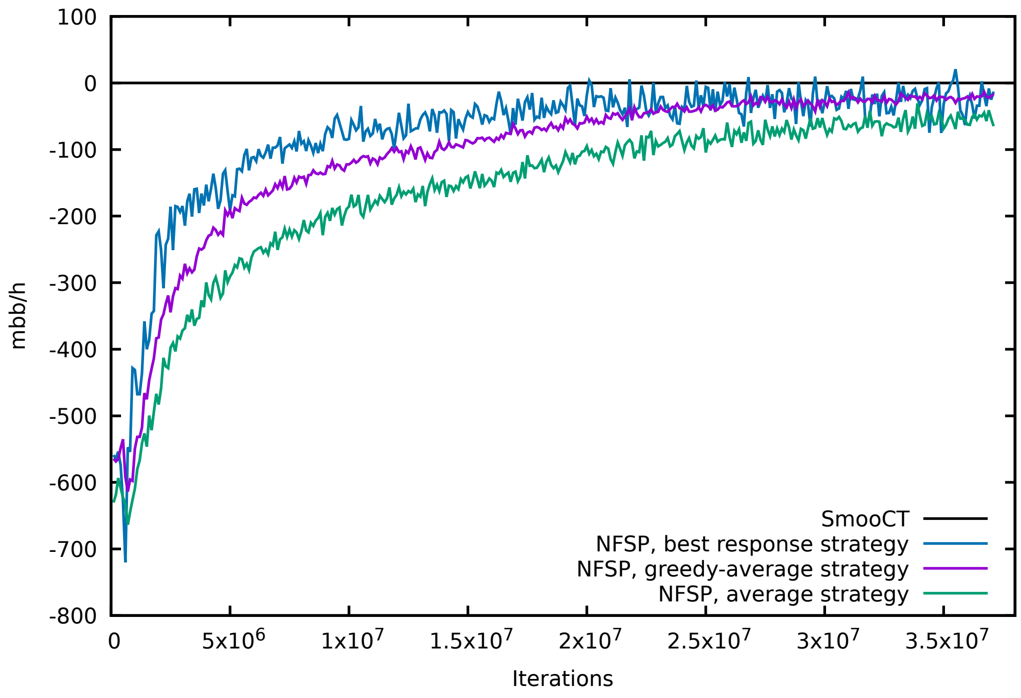


Figure 6.4: Win rates of NFSP against SmooCT in Limit Texas Hold'em. The estimated standard error of each evaluation is less than 10 mbb/h.

Match-up	Win rate (mbb/h)
escabeche	-52.1 $\pm$ 8.5
SmooCT	-17.4 $\pm$ 9.0
Hyperborean	-13.6 $\pm$ 9.2

Table 6.1: Win rates of NFSP's greedy-average strategy against the top 3 agents of the ACPC 2014.

computer agents in the ACPC 2014 achieved up to 50 mbb/h between themselves. While training, we periodically evaluated NFSP's performance against SmooCT from symmetric play for 25000 hands each. Figure 6.4 presents the learning performance of NFSP. NFSP's average and greedy-average strategy profiles exhibit a stable and relatively monotonic performance improvement, and achieve win rates of around -50 and -20 mbb/h respectively. The best response strategy profile exhibited more noisy performance, mostly ranging between -50 and 0 mbb/h. We also evaluated the final greedy-average strategy against the other top 3 competitors of the ACPC 2014. Table 6.1 presents the results. NFSP achieves winrates similar to those of the top half of computer agents in the ACPC 2014 and thus is competitive with superhuman computer poker programs.

6.7 Visualization of a Poker-Playing Neural Network

To analyse the features (abstractions) that NFSP learned for LHE, we visualized the neural networks' last hidden layers' activations with the t-SNE algorithm (Maaten and Hinton, 2008). In particular, by simulating games of NFSP's final average strategy profile, we created a data set consisting of the information states, the last hidden layers' activations, the output action probabilities and various custom features. We applied the t-SNE algorithm to about 100000 vectors of the networks' activations for both players respectively. We used the information state descriptions, action probabilities and custom features to illustrate the embeddings.

Figures 6.5 and 6.6 present both players' embeddings with various colourings. We observe that the activations cluster by the game's rounds and the players' local policies (action probabilities). It is interesting to see the colour gradients between the fold and call actions as well the call and raise actions. Indeed, in many poker situations human players would gradually switch from call to fold (or raise) based on their hand strength. The embeddings also appear to group states based on whether the respective player has the initiative, i.e. was the last to bet or raise. This is a common strategic feature that is taught in many books on poker strategy (Sklansky, 1999). Finally, situations with similar pot sizes are mapped to nearby locations. The pot size is strategically important, as it determines the odds that players get on their bets and thus their optimal action frequencies.

Figure 6.7 presents a selection of game situations from the first player's embedding. Group A shows clusters of pocket pairs that the network tends to raise preflop against a call. Group B presents pairs that player 1 check/called on the flop and turn after defending his big blind preflop (call against a raise). In this situation the player is now facing a final bet on the river (last round). The colouring suggests that the network gradually moves from a call to a fold based on the quality of the pair (middle pair to bottom pair). Group C shows pairs that player 1 holds facing a bet on the flop after defending his big blind preflop. This is one of the most common situations in heads-up LHE. Indeed, these situations' embedding appears to form a prominent ridged structure. The colouring suggests that the network gradually switches from a call to a raise, based on the quality of the pair (bottom to top pair). Group D presents straight draws on the turn, i.e. four cards to a straight that are worthless unless a matching fifth card completes the hand on the river. Group E presents straight draws that player 1 has bet with as a bluff on the turn and is now facing a decision on the river after his draw did not complete to a straight. Both groups, D and E, suggest that the network has learned features that encode straight draws. Furthermore, the mapping of these bluffs to similar features may enable the agent to identify situations in which it was bluffing.

Figure 6.8 presents a selection of game situations from the second player's embedding. Group A shows high cards[1] that face a bet on the river after the turn was checked by both players. This is a common situation where the player has to decide whether their relatively weak holding might be sufficient to win the pot. Indeed, the colouring suggests that the network gradually moves from a fold to a call based on the quality of the high cards. Group B presents high cards that face a check/raise after having placed a bet on the flop. Again we see a gradual switch from fold to call based on the quality of the high cards. The examples indicate that the network would fold weak high-card holdings that have little hope for improvement. Furthermore, it would continue with high-card holdings that can beat a bluff and have some chance for improvement. Group C presents a variety of diamond and spade flushes[2], suggesting that the network has learned features that encode flushes. Group D presents a variety of straight draws that face a bet on the flop, providing further evidence for the network having learned features that encode this category of holdings. Group E shows a variety of bluffs that have either bet (first four examples) or raised the turn (last four examples). In these situations the network faces a check on the river and has to decide whether to follow through with its bluff. Mapping these situations to similar feature vectors may enable the agent to learn decisions on following through with bluffs on the river. When to follow through with a bluff is a common question

[1]Cards that have not formed a pair or better.

[2]Five cards of the same suit.

amongst amateur as well as expert players in the poker community.

The illustrations and discussion suggest that the network learned to represent a variety of specific domain knowledge (card rankings such as pairs, straights and flushes), strategic concepts (bluffs, check/raises, playing strong hands aggressively and weak hands passively) and even heuristic features that human experts use, e.g. initiative.

6.8 Conclusion

We have introduced NFSP, an end-to-end deep reinforcement learning approach to learning approximate Nash equilibria of imperfect-information games from self-play. Unlike previous game-theoretic methods, NFSP learns solely from game outcomes without prior domain knowledge. Unlike previous deep reinforcement learning methods, NFSP is capable of learning (approximate) Nash equilibria in imperfect-information games. Our experiments have shown NFSP to converge reliably to approximate Nash equilibria in a small poker game, whereas DQN's greedy and average strategies did not. In LHE, a poker game of real-world scale, NFSP learnt a strategy that approached the performance of state-of-the-art, superhuman algorithms based on significant domain expertise. To our knowledge, this is the first time that any reinforcement learning algorithm, learning solely from game outcomes without prior knowledge, has demonstrated such a result. Furthermore, the visualization of the learned neural networks has empirically demonstrated NFSP's capability to learn richly featured abstractions end to end.

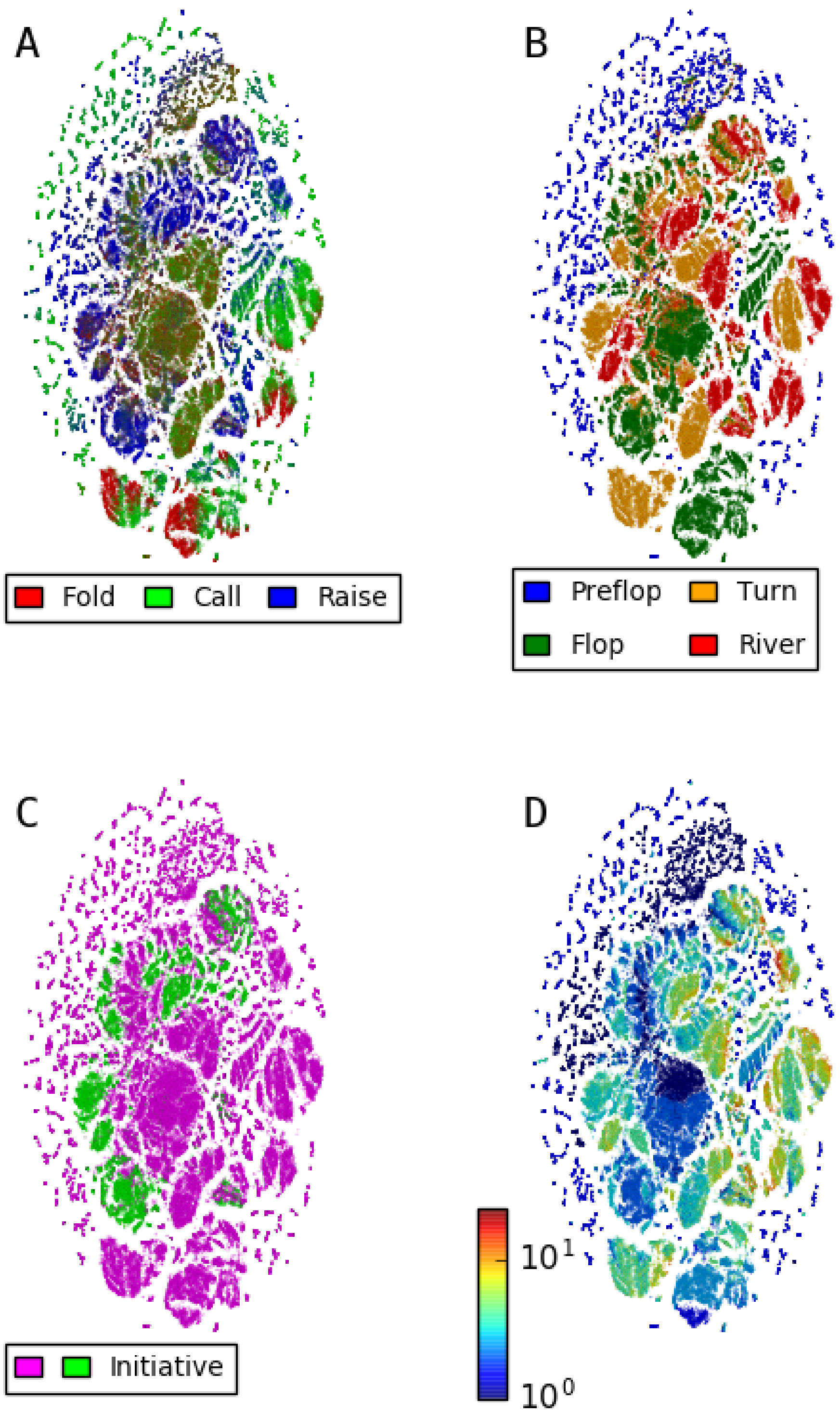


Figure 6.5: t-SNE embeddings of the first player's last hidden layer activations. The embeddings are coloured by A) action probabilities; B) round of the game; C) initiative feature; D) pot size in big bets (logarithmic scale).

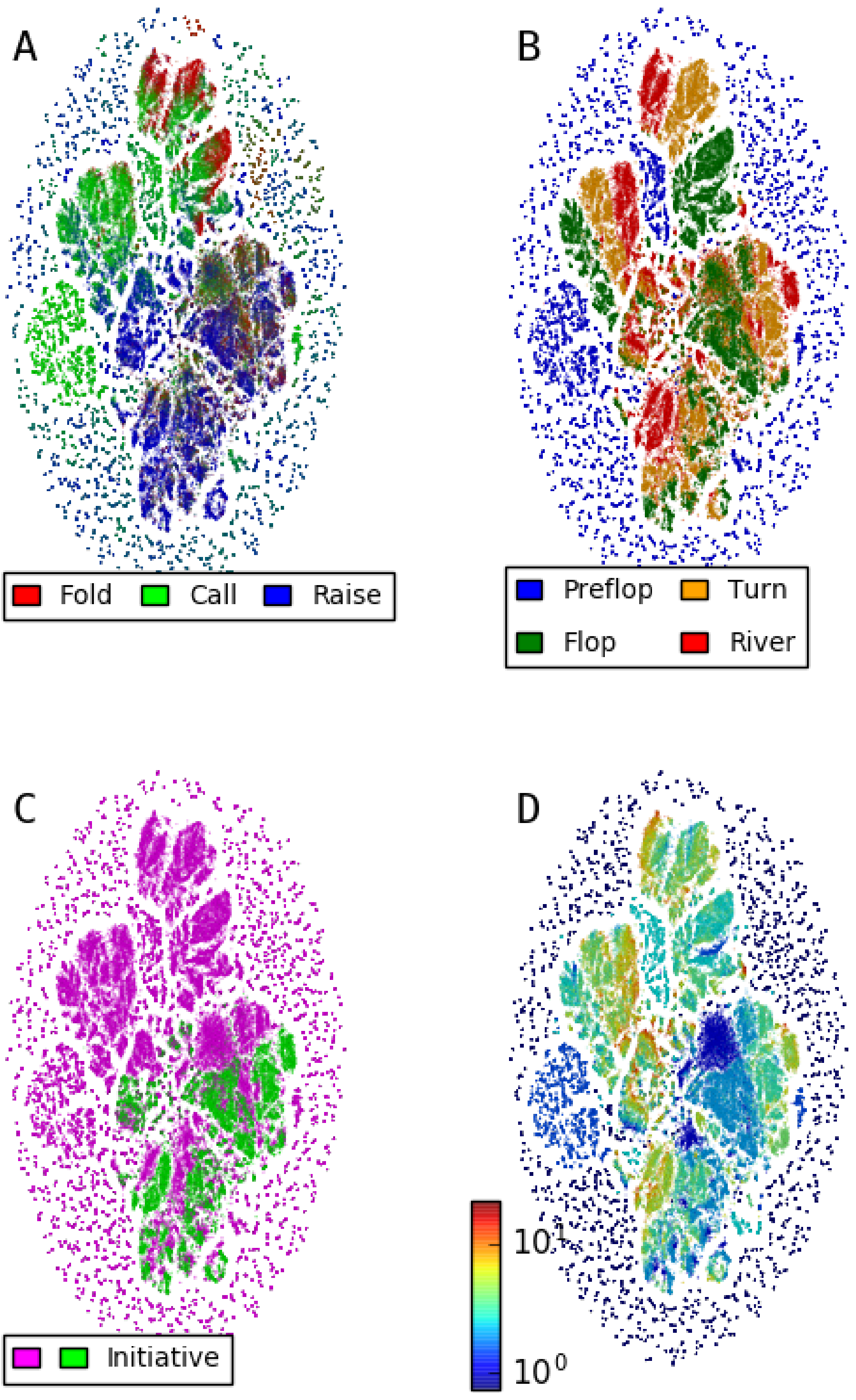


Figure 6.6: t-SNE embeddings of the second player's last hidden layer activations. The embeddings are coloured by A) action probabilities; B) round of the game; C) initiative feature; D) pot size in big bets (logarithmic scale).

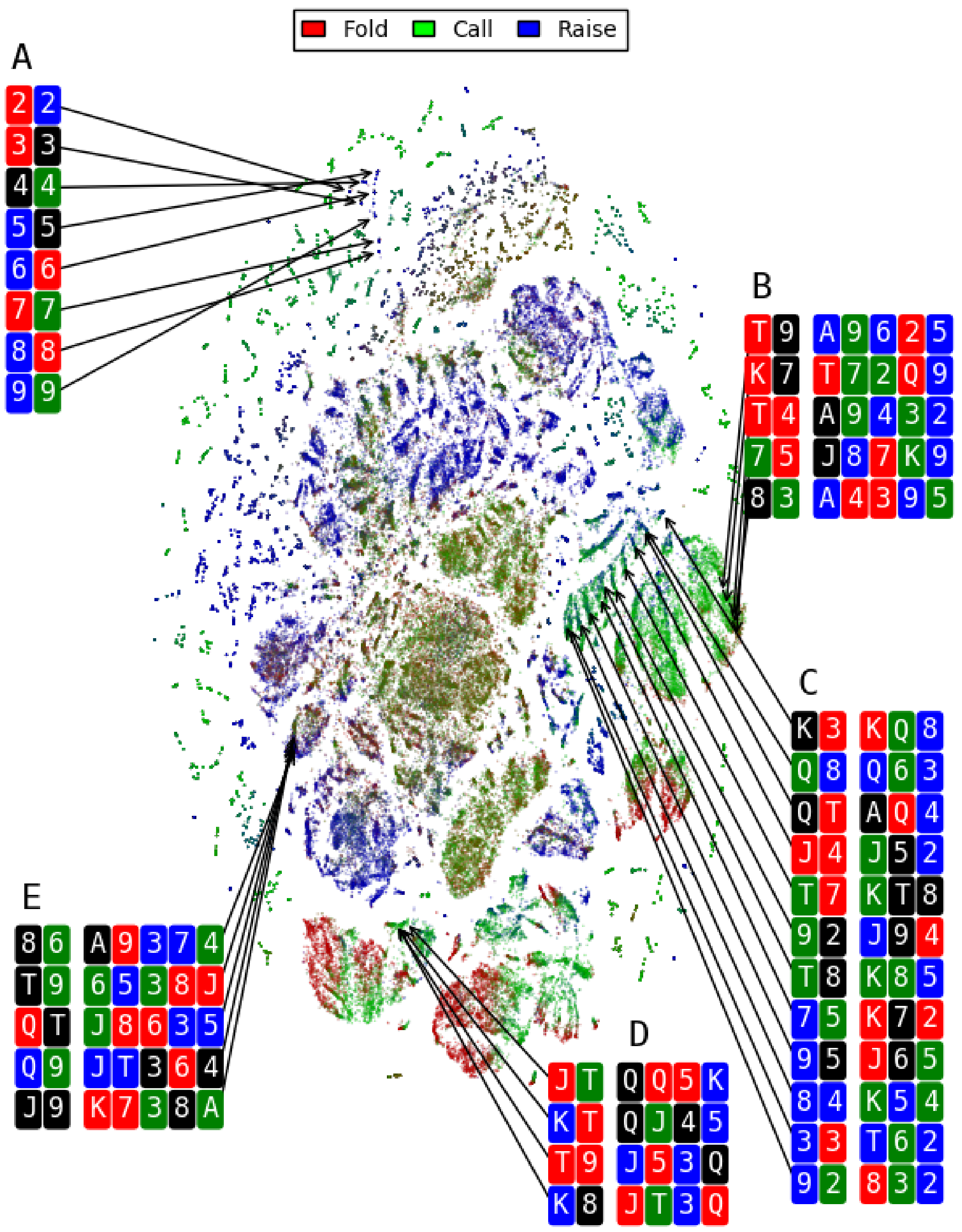


Figure 6.7: A) pairs preflop vs call; B) pairs check/calling down from flop after big-blind defense (rc/crc/crc/cr); C) pairs on flop facing continuation bet after big-blind defense (rc/cr); D) straight draws facing a bet on the turn; E) uncompleted straight draws on the river after having bluff-bet the turn.

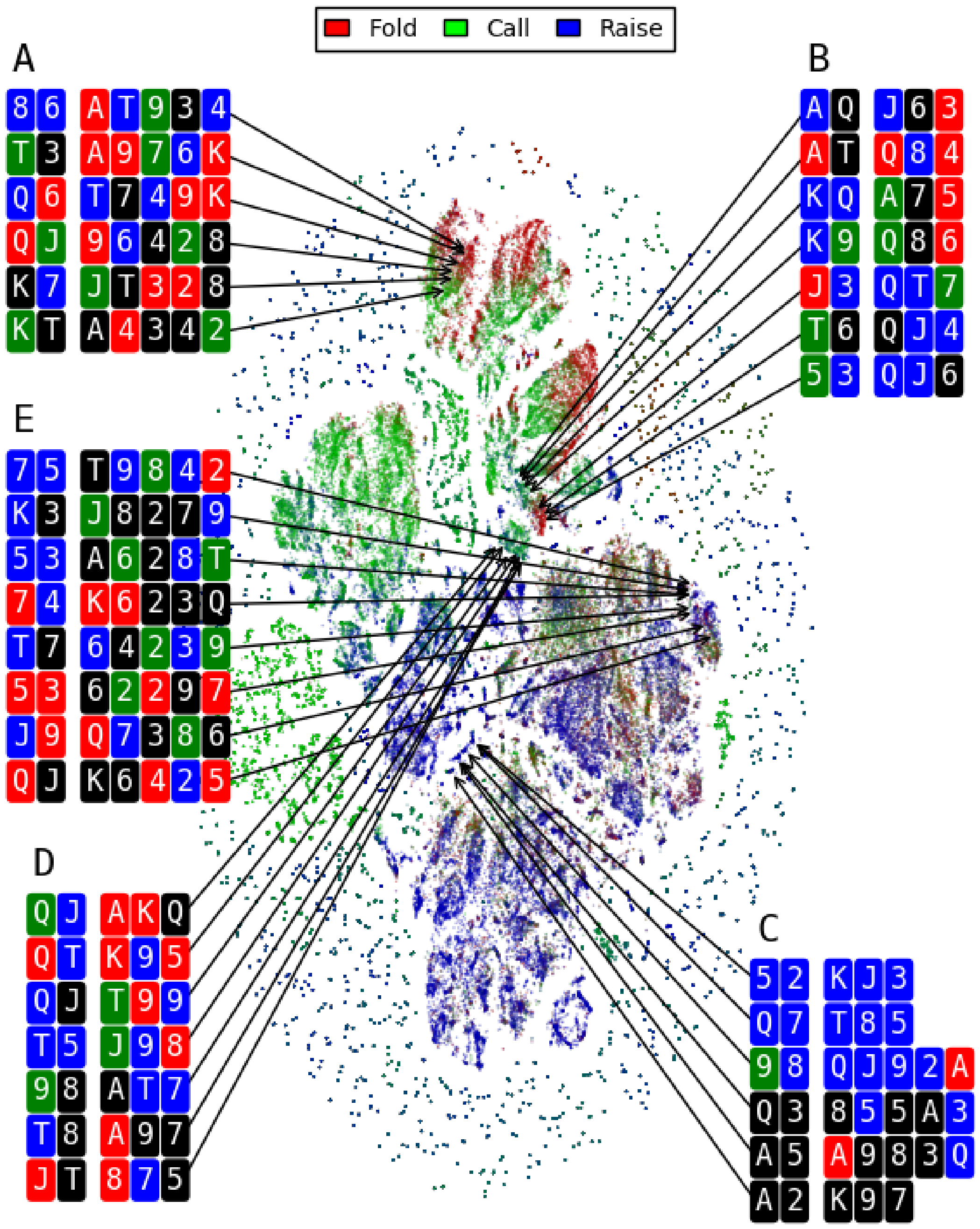


Figure 6.8: A) high cards vs river bet after check through turn; B) high cards facing check/raise on flop after continuation bet (rc/crr); C) flushes facing a check; D) straight draws facing a bet on the flop; E) facing check on river after having bluff-raised (first four) or bluff-bet (last four) the turn.

Chapter 7

Conclusion

We set out to address the question,

> Can an agent feasibly learn *approximate* Nash equilibria of *large-scale imperfect-information* games from self-play?

In this chapter, we discuss our contributions towards answering this question. We also present an outlook on the future of self-play.

7.1 Contributions

We first critically review our contributions and then contextualize them.

7.1.1 Review

Smooth UCT (Chapter 3) introduced the notion of fictitious play into UCT, a popular Monte Carlo Tree Search (MCTS) algorithm. Our work on Smooth UCT can be regarded as a proof of concept. It empirically demonstrated that the combination of fictitious play with reinforcement leaning can indeed learn high-quality approximate Nash equilibria in sequential, imperfect-information games. While the algorithm was practically successful in the 2014 Annual Computer Poker Competition (ACPC), our subsequent work on Extensive-Form Fictitious Play (XFP) and Fictitious Self-Play (FSP) has revealed that it might not have a sound theoretical foundation. In principle, this flaw could be fixed by modifying Smooth UCT to an FSP algorithm[1].

[1]This could be achieved with alternating self-play and a forgetting variant of UCT, e.g. similar to sliding-window or discounted UCB (Garivier and Moulines, 2008)

XFP (Chapter 4) extends fictitious play to sequential games. In particular, it achieves the same convergence guarantees as its single-step (normal-form) counterpart. Moreover, it represents agents' policies in a sequential (extensive) form. Thus, XFP is useful as a blueprint for designing sequential decision making algorithms, e.g. reinforcement learning, that converge in self-play.

FSP (Chapter 5) extends XFP to learning from sampled experience, which is essential to take on large-scale domains that are not exhaustively enumerable. In order to generalise to unseen, i.e. non-sampled, situations, FSP proposes the use of machine learning. In particular, FSP replaces the two full-width computations of XFP, i.e. best response computation and policy averaging, with reinforcement and supervised learning respectively. Furthermore, classical reinforcement and supervised learning techniques, which are designed for stationary environments, can be utilised. While we empirically investigated FSP's approximation errors and convergence, we did not prove theoretical guarantees for its convergence with a finite computational budget per iteration.

NFSP (Chapter 6) instantiates FSP with neural-network function approximation and deep learning techniques. In Limit Texas Hold'em, a large-scale poker game, Neural Fictitious Self-Play (NFSP) approached the performance of state-of-the-art, superhuman algorithms based on significant domain expertise. To our knowledge, this is the first time that any reinforcement learning agent, learning solely from its own experience without prior domain knowledge, has demonstrated such a result in a large-scale game of imperfect information.

7.1.2 Contexts

Perfect to Imperfect Information TD-Gammon (Tesauro, 1995) and AlphaGo (Silver et al., 2016a) achieved superhuman performance in perfect-information games by reinforcement learning from self-play. NFSP achieved a similar result in a real-world-scale imperfect-information game. This is a step towards other real-world applications, which typically involve imperfect information.

Handcrafted to Learned Abstractions Many game-theoretic approaches for computing Nash equilibria typically solve handcrafted abstractions of the game (Johanson et al., 2013), i.e. approximate it first. In contrast, NFSP agents experientally learn approximate Nash equilibria and an (implicit) abstraction end to end, i.e. from the raw inputs of the environment. A visualization of the agents' neural networks (abstractions) revealed that they encoded complex strategic features akin to those a domain expert might choose to craft by hand (see Section 6.7).

Experiential Learning from a Black Box With the exception of Outcome Sampling (OS) (Lanctot et al., 2009), many game-theoretic methods for computing Nash equilibria require explicit knowledge of the game's dynamics. In contrast, FSP and NFSP are experiential reinforcement learning approaches that can be applied to a black box of the game, e.g. learn real-time in a traffic lights system from raw camera inputs.

Realisation of Nash Equilibria Game theory has raised the question of how and whether Nash equilibria are realised in practice (Harsanyi, 1973; Selten, 1975; Fudenberg and Levine, 1995). Leslie and Collins (2006) provided an example of how agents interacting in a single-step game might converge on a Nash equilibrium. NFSP provides a similar example for sequential games. The learning process can be interpreted as habituation to a routine policy, which averages policies that were deemed optimal in the past. Such process is reminiscent of an actor-critic architecture (Barto et al., 1983).

7.2 Future of Self-Play

Local Planning Learning in a model of a subgame[2], in the hope of transferring a policy to the real environment, is known as local planning. By refining an agent's policy from self-play in the currently encountered subgame, local planning has been incredibly successful in perfect-information games, e.g. Chess (Campbell et al., 2002) or Go (Gelly et al., 2012; Silver et al., 2016a). In imperfect-information games, however, strictly local[3] planning does not generally recover a global Nash equilibrium (Burch et al., 2014; Ganzfried and Sandholm, 2015; Lisý et al., 2015). The consequences of this issue and how they could be overcome, e.g. global search

[2]A subgame encompasses a subset of locally relevant game situations.

[3]Sampling from the planning agents beliefs about the state of the game.

targeting (Lisý et al., 2015), present a promising avenue of research.

Moravčík et al. (2017) have recently[4] presented DeepStack, an approach to local planning based on decomposition of imperfect-information games (Burch et al., 2014). They utilise deep learning from approximate, game-theoretic solutions of the game in order to innovatively bootstrap depth-limited, local planning in imperfect-information games. A key aspect of this approach is to compactly summarize a global Nash equilibrium with a set of counterfactual values and probabilities of realising information states (Burch et al., 2014; Moravčík et al., 2017). This compact summary of a Nash equilibrium, however, may be hard to come by as it requires prior approximate solutions of the game. Furthermore, unlike in perfect-information games, local planning in imperfect-information games may not recover from an inaccurate compact summary of the Nash equilibrium.

We may want to investigate the following questions. As DeepStack relies on a quality summary of the global Nash equilibrium, are there alternative approaches to (semi-)local planning that do not have such requirements? For example, if there was a mutually-observed state in the sequence of play should we rather plan from that prior state, not requiring any compact summary of a Nash equilbrium? Do such mutually-observed states commonly occur in real-world imperfect-information games? Examples may include entering a new room with different people, temporary occlusions or synchronizing information through communication. Finally, how should we plan locally in general-sum multi-player games, e.g. by using player beliefs (Cowling et al., 2015) or equilibrium selection (Nash and Shapley, 1950; Szafron et al., 2013)? In particular, are there any performance guarantees of multi-player Nash equilibria that would justify a need for local planning that is able to recover global Nash equilibria?

Temporal Abstraction In principle, an environment defines an agent's primitive actions at the finest granularity, e.g. muscle controls. A great many such actions might be required to perform a higher-level action such as taking a three-point shot (rather than passing) in basketball. Arguably, strategic decision making (in the multi-agent

[4]This work was published after the defence of this thesis.

sense) is mainly important at such higher levels of decisions. This is because primitive or lower-level actions might have negligible reciprocal effects on fellow agents. Therefore, temporal abstraction, e.g. hierarchical reinforcement learning (Dayan and Hinton, 1993; Sutton et al., 1999b; Dietterich, 2000; Barto and Mahadevan, 2003), may be of particular importance to strategic decision making.

We may want to investigate the following questions. We may see full-game strategies as the most temporally extended courses of action, i.e. options. Consequently, a normal-form game would be a temporal abstraction. Can an agent learn such a normal-form, temporal abstraction from experience in an extensive-form game? In particular, how does it decide which learned strategies to put in the abstract normal-form action set? For example, fictitious play could be considered adding all past best responses. Is there a small set of distinctive strategies (options) that would enable effective strategic reasoning, i.e. a kind of basis for the strategic space of the game (Bosansky et al., 2014)? Does learning finer-grained temporal abstractions in an extensive-form game pose the same challenges as temporal abstraction in sequential single-agent domains (McGovern and Barto, 2001; Mannor et al., 2004; Konidaris et al., 2010; Castro and Precup, 2011)? In particular, are there any challenges that are specific to multi-agent strategic decision making? For example, could mutually-observed states not only serve as natural roots for local planning (see above) but also define strategic termination conditions of options?

Game Models To learn from self-play, an agent requires a model of a multi-agent environment. Learning multi-agent models poses distinctive challenges. In contrast to learning a model of a single-agent environment (Sutton, 1990; Lin, 1992; Silver et al., 2016b), a multi-agent model would have to identify fellow agents, i.e. assumed intelligences. Furthermore, such a model includes not only possible information states and actions of fellow agents but also beliefs about their subjective reward signals. Learning Nash equilibria in such models touches on questions of bounded rationality (Simon, 1972; Selten, 1990), e.g. a fellow agent might be irrational or just optimising an unconventional reward signal.

A learned multi-agent model could also include likely or definite behaviours that fellow agents are assumed to exhibit (Johanson et al., 2008). Thus, there is a continuum from the static, self-play approach (Sections 1.1.2 and 1.1.3) to the adaptive approach (Section 1.1.1) in strategic decision making. At one extreme, only the choices of fellow agents are modelled, without any assumptions on their likelihood. At the other extreme, a completely defined model of fellow agents' policies is learned. The latter model is a Markov decision process that can be solved with typical (single-agent) reinforcement learning methods. The other models in the continuum, which leave some choices of fellow agents undetermined, define games that require an agent to learn from self-play.

We may want to investigate the following questions. Given that all learning from self-play is model-based, what motivates an agent to learn and solve such a subjective model in the first place? In particular, should a general reinforcement learner (Hutter, 2005; Legg, 2008) ever play a Nash equilibrium, i.e. a solution to a multi-agent model? In the case of the agent lacking a model of fellow agents' behaviour, couldn't self-play substitute for such a model by learning reasonable behaviours from the fellow agents' points of view? For example, humans may use a theory of mind (Premack and Woodruff, 1978; Yoshida et al., 2008) to plan in games. How can we enable an artifical agent to learn such a multi-agent model from experience?

Appendix A

Analysis of Self-Play Policy Iteration

Rational learning from self-play can only converge on Nash equilibria (Bowling and Veloso, 2001). This chapter presents analyses which suggest that Generalised Policy Iteration (GPI), a fundamental principle of many reinforcement learning algorithms (Sutton and Barto, 1998), might be unsuitable for convergent learning of Nash equilibria from self-play in games that are not perfect-information two-player zero-sum.

A.1 Nash Equilibria of Imperfect-Information Games

We analyse the value functions of a Nash equilibrium and conclude that rational policy improvement may not be able to represent the equilibrium. Following the definition of a Nash equilibrium (Nash, 1951), we know that all actions in the support of the equlibrium have the same (maximum) value.

Theorem A.1.1. *Let π be a Nash equilibrium of an extensive-form game, $i \in \mathcal{N}$ any agent of the game and $u^i \in \mathcal{U}^i$ any of its information states that has positive probability of being reached under the equilibrium, π. Furthermore, let $\overline{\mathcal{A}}(u^i) \subseteq \mathcal{A}(u^i)$ be the agents's actions in the support of its equilibrium strategy, π^i, at u^i. Then, its action-value function satisfies*

$$\begin{aligned} Q^\pi(u^i,a) &= Q^\pi(u^i,a') \quad \text{for all } a,a' \in \overline{\mathcal{A}}(u^i), \\ Q^\pi(u^i,a) &\geq Q^\pi(u^i,a') \quad \text{for all } a \in \overline{\mathcal{A}}(u^i), a' \in \mathcal{A}(u^i) \end{aligned}$$

Proof. If one of the properties was not satisfied, then there would be an action $a \in \mathscr{A}(u^i)$ with $\pi^i(a \mid u^i) > 0$ such that $Q^{\pi}(u^i, a) < \arg\max_{a' \in \mathscr{A}(u^i)} Q^{\pi}(u^i, a')$, which violates the assumption that π is a Nash equilibrium, as the agent would be able to improve its return. $\square$

Consider a policy iteration algorithm that has evaluated the Nash equilibrium strategy, i.e. computed its value function. Consider policy improvement being performed via a mapping from action-value functions to policies.

Definition A.1.2. A monotonic policy mapping from the action-value function, Q, to a policy, π_Q, satisfies for any information state u

$$\begin{aligned} \pi_Q(a \mid u) &> \pi_Q(a' \mid u) \quad \text{if } Q(u,a) > Q(u,a'), \\ \pi_Q(a \mid u) &= \pi_Q(a' \mid u) \quad \text{if } Q(u,a) = Q(u,a') \end{aligned}$$

It is hard to justify a non-monotonic stochastic policy mapping for a Generalised Policy Iteration (GPI)-based agent from knowledge of the (current) evaluated action-value function alone. Why would it ever prefer actions with lower values, and why should it choose a non-uniform distribution over same-value actions (unless it uses a deterministic policy for convenience)? However, Nash equilibria of imperfect-information games generally require stochastic strategies with non-uniform distributions, e.g. the equilibrium of Kuhn poker (Kuhn, 1950) balances players betting frequencies in proportion to the pot size. Thus, both a deterministic (greedy) as well as a monotonic policy mapping would break a perfectly-evaluated Nash equilibrium strategy.

We also considered policy gradient methods (Sutton et al., 1999a), which are an instance of GPI. Each agent's policy gradient would be zero for a Nash equilibrium strategy profile. However, at any information state the action-value function would be flat across all dimensions of actions in the support of the equilibrium. There is no curvature that attracts a gradient descent method to the Nash equilibrium point. Therefore, an agent learning with a stochastic policy gradient method (Sutton et al., 1999a) against a fixed Nash equilibrium would not be expected to recover a Nash strategy. If all agents learned simultaneously, they might push each other back towards the equilibrium by penalising deviations. Bowling and Veloso (2001) demonstrated that this can indeed be achieved if agents dynamically adapt their learning rates.

A.2 Policy Iteration in Perfect-Information Games

Having discussed the potential unsuitability of policy iteration for self-play learning in imperfect-information games, we investigate its applicability to games with perfect information.

A general finite perfect-information game with any number of players can, in principle, be solved by dynamic programming from the leaves of the game, e.g. exhaustive Minimax search. It requires only a single exhaustive traversal of the game tree to compute a Nash equilibrium strategy.

However, consider an iterative learning process based on the same dynamic programming principles, e.g. GPI. Figure A.1 shows a simple two-player general-sum game. Consider a policy iteration process that alternates policy evaluation and greedy policy improvement. If two actions have the same value, then the greedy action is chosen uniform randomly. Since player two is indifferent between all of his actions he will randomly choose one of his 4 deterministic strategies, $\{Ll, Lr, Rl, Rr\}$, at each iteration. We make the following observations.

- The iterative policy iteration process never converges as player 1 is forced to switch his optimal policy infinitely often due to player 2's infinite (random) switches.

- The average policies converge but not to a Nash equilibrium. The average policy of player 2 converges to a uniform random distribution over his actions at each state. Observe that at each iteration player 2 plays one of his 4 policies with equal probability. Hence, player 1 will respond with one of four best responses to player 1's policies. The average policy of player 1 thus converges to $P(L) = \frac{3}{8}$, $P(R) = \frac{5}{8}$. This is not a best response to the average strategy of player 2. Therefore, the converged average strategies do not form a Nash equlibrium.

- Now consider a policy iteration process where players evaluate their policies against the opponent's average strategy, i.e. as in fictitious play. In this case, it is easy to see that player 1 will converge to the pure strategy R, a best response to player 2's average strategy. Any strategy of player 2 is a best response as all yield reward 0. This process thus converges to a Nash equilibrium in this game.

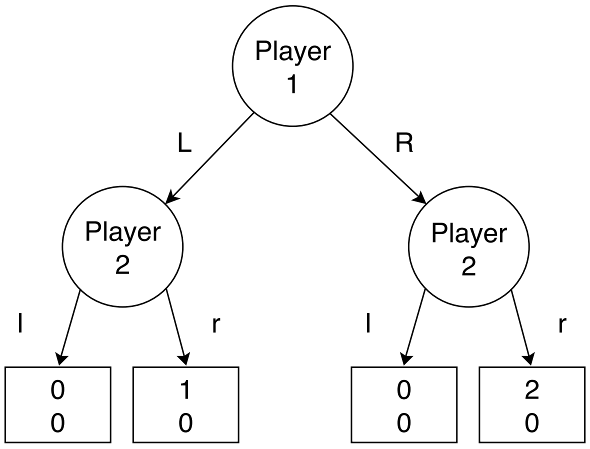


Figure A.1: A small two-player general-sum game.

We can trivially extend the presented two-player game to a zero-sum three player game, by adding a third player that has only one action which always leads to the initial state of the original game. Finally, at each terminal node, add a reward for player 3 that is the negative of player 1's reward. The previous arguments also apply to this three-player zero-sum game.

Note that in a two-player zero-sum game, if a player is indifferent between actions at a subsequent node, then his opponent would also be indifferent between the player's choice as he would get the same value due to the zero-sum property. Thus, the players' values that are backed up by dynamic programming are the same for any distribution over the players' same-value actions. Therefore, in finite perfect-information zero-sum extensive-form games (iterative) deterministic policy iteration converges to a Nash equilibrium after finitely many steps.

Appendix B

Geometric Fictitious Play

According to the common interpretation of fictitious play, players choose (and play) a best response to their fellow players' empirical, average behaviour. We propose an alternative interpretation. Fictitious players have a routine (average) strategy, π_k, that they choose at each iteration. They update their routine strategy by introducing innovative behaviour, $\Delta\Pi_k \in \alpha_k \left(\mathrm{BR}(\Pi_{k-1}) - \Pi_k\right)$, with respect to their fellow players' current strategy profile, π_k. In particular, innovation slows down over time as α_k decays and the routine strategies, π_k, become more sophisticated. This interpretation produces an equivalent sequence of average, routine strategies and thus follows common fictitious play.

In XFP, players would introduce their (innovative) best response strategy at the root of the game in proportion to a diminishing fraction (stepsize). This is because the standard fictitious play update is realization equivalent to a mixed strategy that samples the previous average strategy and the innovative best response in proportion to the stepsize. Here, we consider a more aggressive update that introduces local, sequential best responses at each information state. In particular, we define the local policy operator,

$$\mathrm{L}\left[\beta, \pi, s'\right](a \,|\, s) = \begin{cases} \beta(a \,|\, s) & \text{for } \sigma_{s'} \subseteq \sigma_s \\ \pi(a \,|\, s) & \text{otherwise} \end{cases}, \qquad \text{(B.1)}$$

that maps two strategies of a player, β^i and π^i, and one of their information states, s^i, to a strategy that plays β^i at all information states in the (local) subtree rooted at s^i and π^i otherwise. Any strategy in $\left\{L[\beta,\pi^i,s^i] : \beta \in \mathrm{BR}^i(\pi^i)\right\}$ is a local, sequential best response of player i from s^i with routine strategy π^i to their fellow players' strategy profile π^{-i}. For a best response $B^i \in \mathrm{BR}\left(\Pi^{-i}\right)$, consider the geometrically-weighted update,

$$(1-\alpha)^{D+1}\Pi^i + \alpha \sum_{d=0}^{D} (1-\alpha)^d \sum_{s \in \mathscr{S}_d^i} x_{\Pi^i}(\sigma_s) \mathrm{L}\left[B^i, \Pi^i, s\right], \tag{B.2}$$

where D is the maximum depth of player i's information-state tree and $\mathscr{S}_d^i$ are their information states at depth d of the tree. Note that, $\sum_{s \in \mathscr{S}_d^i} x_{\Pi^i}(\sigma_s) = 1$ and $\alpha \sum_{d=0}^{D} (1-\alpha)^d = 1 - (1-\alpha)^{D+1}$. Geometric Fictitious Play (GFP) is a fictitious play algorithm that uses this geometrically-weighted update.

While this looks like a busy update, it has a simple interpretation and also an efficient recursive expression for behavioural strategies. The update is equivalent to beginning the game with routine strategy, π^i, and at each step in the game, including at the root, choosing to switch to the best response for the remainder of the game with probability α. Accordingly, for an information state s at depth d, the realization-equivalent behavioural update is

$$\pi(s) \propto (1-\alpha)^{d+1} x_\pi(\sigma_s)\pi(s) + \underbrace{\alpha \sum_{j=0}^{d} (1-\alpha)^j x_\pi\left(\sigma_s[:j]\right) x_\beta\left(\sigma_s[j:]\right)}_{\alpha[\sigma_s]} \beta(s), \tag{B.3}$$

where $\sigma_s[:j]$ is the partial sequence of the first j actions in σ_s and $\sigma_s[j:] = \sigma_s \setminus \sigma_s[:j]$. Futhermore, the local stepsize, $\alpha[\sigma_s]$, can be recursively determined,

$$\alpha[\sigma_s a] = \alpha[\sigma]\beta(a \mid s) + \alpha(1-\alpha)^{|\sigma_s|+1} x_\pi(\sigma_s a), \tag{B.4}$$

with $\alpha[\emptyset] = \alpha$. Finally, we have

$$\begin{aligned}\pi(s) &\propto (1-\alpha)^d x_\pi(\sigma_s)\left[(1-\alpha)\pi(s)+\alpha\beta(s)\right]+\alpha[\sigma_s[:d-1]]\beta(s)\\ &\propto (1-\alpha)\pi(s)+\alpha\beta(s)+\underbrace{c}_{\geq 0}\beta(s) \quad \text{for } x_\pi(\sigma_s)\neq 0. \end{aligned} \tag{B.5}$$

Thus, Geometric Fictitious Play (GFP)'s local stepsize at each information state is at least α, wheras XFP's, $\frac{\alpha x_\beta(\sigma_s)}{(1-\alpha)x_\pi(\sigma_s)+\alpha x_\beta(\sigma_s)}$, is strictly smaller than GFP's and can be as small as zero.

Why would we want to apply such an update? The standard XFP update is very sparse, e.g. for a deterministic best response it updates the policy along only a single trajectory in a player's information state tree. For a stochastic best response, its realization weight is thinly spread across the tree. GFP, on the other hand, achieves a minimum update of at least stepsize α at each information state (see Equation B.5). Furthermore, in addition to XFP's (global, normal-form) probability of α of playing the best response, GFP plays additional local best responses with probability $\alpha(1-\alpha)^d$ from each level d of the tree (see Equation B.2). These local best responses are distributed across each level in proportion to the routine strategy's realization probabilities, thus carrying the updates to the potentially most relevant states. The result is that the GFP update's probability of playing the routine strategy by step d of the game decays exponentially, $(1-\alpha)^d$. In particular, it still plays the best response with only probability α at the root of the game. Intuitively, we might be able to get away with these more aggressive updates deeper into the tree, because these updates have less repercussions for the player's own subsequent information states and also their fellow players' belief states. In our experiments in Section 4.5.2 GFP outperformed XFP in Leduc Hold'em.

Bibliography

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (1995). Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 322–331. IEEE.

Auger, D. (2011). Multiple tree for partially observable Monte-Carlo tree search. In *Applications of Evolutionary Computation*, pages 53–62. Springer.

Baird, L. et al. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37.

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

Benaïm, M., Hofbauer, J., and Sorin, S. (2005). Stochastic approximations and differential inclusions. *SIAM Journal on Control and Optimization*, 44(1):328–348.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., and Szafron, D. (2003). Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th International Joint Conference on Artifical Intelligence*, pages 661–668.

Billings, D., Davidson, A., Schaeffer, J., and Szafron, D. (2002). The challenge of poker. *Artificial Intelligence*, 134(1):201–240.

Bosansky, B., Kiekintveld, C., Lisy, V., and Pechoucek, M. (2014). An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, pages 829–866.

Bowling, M., Burch, N., Johanson, M., and Tammelin, O. (2015). Heads-up limit holdem poker is solved. *Science*, 347(6218):145–149.

Bowling, M. and Veloso, M. (2001). Rational and convergent learning in stochastic games. In *Proceedings of the 17th International Joint Conference on Artifical Intelligence*, volume 17, pages 1021–1026.

Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity analysis of production and allocation*.

Brown, N., Ganzfried, S., and Sandholm, T. (2015). Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit texas hold'em agent. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 7–15. International Foundation for Autonomous Agents and Multiagent Systems.

Brown, N. and Sandholm, T. W. (2015). Simultaneous abstraction and equilibrium finding in games. AAAI.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Burch, N., Johanson, M., and Bowling, M. (2014). Solving imperfect information games using decomposition. In *28th AAAI Conference on Artificial Intelligence*.

Campbell, M., Hoane, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, 134(1):57–83.

Castro, P. S. and Precup, D. (2011). Automatic construction of temporally extended actions for mdps using bisimulation metrics. In *European Workshop on Reinforcement Learning*, pages 140–152. Springer.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *5th International Conference on Computer and Games*.

Cowling, P. I., Powley, E. J., and Whitehouse, D. (2012). Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143.

Cowling, P. I., Whitehouse, D., and Powley, E. J. (2015). Emergent bluffing and inference with monte carlo tree search. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 114–121. IEEE.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–271. Morgan Kaufmann Publishers.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. In *Journal of Machine Learning Research*, pages 503–556.

Frank, I. and Basin, D. (1998). Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123.

Fudenberg, D. (1998). *The theory of learning in games*, volume 2. MIT press.

Fudenberg, D. and Levine, D. K. (1995). Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, 19(5):1065–1089.

Ganzfried, S. and Sandholm, T. (2009). Computing equilibria in multiplayer stochastic games of imperfect information. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pages 140–146.

Ganzfried, S. and Sandholm, T. (2015). Endgame solving in large imperfect-information games. In *Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent Systems*.

Garivier, A. and Moulines, E. (2008). On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*.

Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., and Teytaud, O. (2012). The grand challenge of computer go: Monte Carlo tree search and extensions. *Communications of the ACM*.

Gilpin, A., Hoda, S., Pena, J., and Sandholm, T. (2007). Gradient-based algorithms for finding Nash equilibria in extensive form games. In *Internet and Network Economics*, pages 57–69. Springer.

Gilpin, A. and Sandholm, T. (2006). A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1007.

Gordon, G. J. (2001). Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems*. Citeseer.

Greenwald, A., Li, J., Sodomka, E., and Littman, M. (2013). Solving for best responses in extensive-form games using reinforcement learning methods. *The 1st Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*.

Harsanyi, J. C. (1973). Games with randomly disturbed payoffs: A new rationale for mixed-strategy equilibrium points. *International Journal of Game Theory*, 2(1):1–23.

Heess, N., Silver, D., and Teh, Y. W. (2012). Actor-critic reinforcement learning with energy-based policies. In *EWRL*, pages 43–58.

Heinrich, J. and Silver, D. (2014). Self-play Monte-Carlo tree search in computer poker. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Hendon, E., Jacobsen, H. J., and Sloth, B. (1996). Fictitious play in extensive form games. *Games and Economic Behavior*, 15(2):177–202.

Hoda, S., Gilpin, A., Pena, J., and Sandholm, T. (2010). Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512.

Hoehn, B., Southey, F., Holte, R. C., and Bulitko, V. (2005). Effective short-term opponent exploitation in simplified poker. In *AAAI*, volume 5, pages 783–788.

Hofbauer, J. and Sandholm, W. H. (2002). On the global convergence of stochastic fictitious play. *Econometrica*, 70(6):2265–2294.

Hula, A., Montague, P. R., and Dayan, P. (2015). Monte carlo planning method estimates planning horizons during interactive social exchange. *PLoS Comput Biol*, 11(6):e1004254.

Hutter, M. (2005). *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin.

Johanson, M., Bard, N., Burch, N., and Bowling, M. (2012). Finding optimal abstract strategies in extensive-form games. In *26th AAAI Conference on Artificial Intelligence*.

Johanson, M., Burch, N., Valenzano, R., and Bowling, M. (2013). Evaluating state-space abstractions in extensive-form games. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 271–278.

Johanson, M., Waugh, K., Bowling, M., and Zinkevich, M. (2011). Accelerating best response calculation in large extensive games. In *Proceedings of the 22nd international joint conference on Artificial intelligence*, volume 11, pages 258–265.

Johanson, M., Zinkevich, M., and Bowling, M. (2008). Computing robust counter-strategies. In *Advances in neural information processing systems*, pages 721–728.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134.

Karlin, S. (1959). *Mathematical methods and theory in games, programming and economics*. Addison-Wesley.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer.

Koller, D., Megiddo, N., and Von Stengel, B. (1994). Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 750–759. ACM.

Koller, D., Megiddo, N., and Von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259.

Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. G. (2010). Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *Advances in neural information processing systems*, pages 1162–1170.

Kroer, C. and Sandholm, T. (2014). Extensive-form game abstraction with bounds. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 621–638. ACM.

Kuhn, H. W. (1950). Simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103.

Kuhn, H. W. (1953). Extensive games and the problem of information. *Contributions to the Theory of Games*, 2(28):193–216.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.

Lambert III, T. J., Epelman, M. A., and Smith, R. L. (2005). A fictitious play approach to large-scale optimization. *Operations Research*, 53(3):477–489.

Lanctot, M. (2013). *Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games*. Ph.D. thesis, University of Alberta.

Lanctot, M., Waugh, K., Zinkevich, M., and Bowling, M. (2009). Monte Carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems 22*, pages 1078–1086.

Lange, S., Gabel, T., and Riedmiller, M. (2012). Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer.

Legg, S. (2008). *Machine super intelligence*. PhD thesis, University of Lugano.

Leslie, D. S. and Collins, E. J. (2006). Generalised weakened fictitious play. *Games and Economic Behavior*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Lisý, V. (2014). Alternative selection functions for information set monte carlo tree search. *Acta Polytechnica: Journal of Advanced Engineering*, 54(5):333–340.

Lisy, V., Kovarik, V., Lanctot, M., and Bosansky, B. (2013). Convergence of Monte Carlo tree search in simultaneous move games. In *Advances in Neural Information Processing Systems*, pages 2112–2120.

Lisý, V., Lanctot, M., and Bowling, M. (2015). Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent Systems*.

Littman, M. L. (1996). *Algorithms for sequential decision making*. PhD thesis, Brown University.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th international conference on Machine learning*.

McMahan, H. B. and Gordon, G. J. (2007). A fast bundle-based anytime algorithm for poker and other convex games. In *International Conference on Artificial Intelligence and Statistics*, pages 323–330.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Monderer, D. and Shapley, L. S. (1996). Fictitious play property for games with identical interests. *Journal of economic theory*, 68(1):258–265.

Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in no-limit poker. *arXiv preprint arXiv:1701.01724*.

Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, pages 286–295.

Nash, J. and Shapley, L. (1950). A simple three-person poker game. *Essays on Game Theory*.

Neumann, J. v. (1928). Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320.

Newall, P. (2013). *Further Limit Hold 'em: Exploring the Model Poker Game*. Two Plus Two Publishing, LLC.

Osborne, M., Lall, A., and Van Durme, B. (2014). Exponential reservoir sampling for streaming language models. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Ponsen, M., de Jong, S., and Lanctot, M. (2011). Computing approximate Nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, 42(1):575–605.

Premack, D. and Woodruff, G. (1978). Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(04):515–526.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pages 317–328. Springer.

Risk, N. A. and Szafron, D. (2010). Using counterfactual regret minimization to create competitive multiplayer poker agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 159–166.

Robinson, J. (1951). An iterative method of solving a game. *Annals of Mathematics*, pages 296–301.

Roth, A. E. (2002). The economist as engineer: Game theory, experimentation, and computation as tools for design economics. *Econometrica*, 70(4):1341–1378.

Rubin, J. and Watson, I. (2011). Computer poker: A review. *Artificial Intelligence*, 175(5):958–987.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

Sandholm, T. (2010). The state of solving large incomplete-information games, and application to poker. *AI Magazine*, 31(4):13–32.

Sandholm, T. and Singh, S. (2012). Lossy stochastic game abstraction with bounds. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 880–897. ACM.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Selten, R. (1975). Reexamination of the perfectness concept for equilibrium points in extensive games. *International journal of game theory*, 4(1):25–55.

Selten, R. (1990). Bounded rationality. *Journal of Institutional and Theoretical Economics*, pages 649–658.

Shamma, J. S. and Arslan, G. (2005). Dynamic fictitious play, dynamic gradient play, and distributed convergence to Nash equilibria. *IEEE Transactions on Automatic Control*, 50(3):312–327.

Shannon, C. E. (1950). Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*.

Silver, D. (2009). Reinforcement learning and simulation-based search. *Doctor of philosophy, University of Alberta*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016a). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. (2016b). The predictron: End-to-end learning and planning. *Preprint*.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pages 2164–2172.

Simon, H. A. (1972). Theories of bounded rationality. *Decision and organization*, 1(1):161–176.

Sklansky, D. (1999). *The theory of poker*. Two Plus Two Publishing LLC.

Southey, F., Bowling, M., Larson, B., Piccione, C., Burch, N., Billings, D., and Rayner, C. (2005). Bayes bluff: Opponent modelling in poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence*.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, pages 1038–1044.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999a). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063.

Sutton, R. S., Precup, D., and Singh, S. (1999b). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211.

Szafron, D., Gibson, R., and Sturtevant, N. (2013). A parameterized family of equilibrium profiles for three-player kuhn poker. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 247–254. International Foundation for Autonomous Agents and Multiagent Systems.

Tambe, M. (2011). *Security and game theory: algorithms, deployed systems, lessons learned.*

Tesauro, G. (1992). Practical issues in temporal difference learning. In *Reinforcement Learning*, pages 33–53. Springer.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Veness, J., Silver, D., Blair, A., and Uther, W. (2009). Bootstrapping from game tree search. In *Advances in neural information processing systems*, pages 1937–1945.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software*.

Von Neumann, J. and Morgenstern, O. (1944). Theory of games and economic behavior.

Von Stengel, B. (1996). Efficient computation of behavior strategies. *Games and Economic Behavior*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge England.

Waugh, K., Morrill, D., Bagnell, J. A., and Bowling, M. (2015). Solving games with functional regret estimation. In *29th AAAI Conference on Artificial Intelligence*.

Waugh, K., Schnizlein, D., Bowling, M., and Szafron, D. (2009a). Abstraction pathologies in extensive games. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 781–788.

Waugh, K., Zinkevich, M., Johanson, M., Kan, M., Schnizlein, D., and Bowling, M. H. (2009b). A practical use of imperfect recall. In *Proceedings of the 8th Symposium on Abstraction, Reformulation and Approximation*. AAAI Press.

Yoshida, W., Dolan, R. J., and Friston, K. J. (2008). Game theory of mind. *PLoS Comput Biol*, 4(12):e1000254.

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. (2007). Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, pages 1729–1736.