SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Using Normalizing Flows to Find Mixed-Strategy Equilibria

Janek Falkenstein

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Using Normalizing Flows to Find Mixed-Strategy Equilibria

Verwendung von Normalizing Flows zur Findung von Gleichgewichten mit gemischten Strategien

Author:	Janek Falkenstein
Supervisor:	Prof. Dr. Martin Bichler
Advisor:	M.Sc. Markus Ewert
Submission Date:	15.07.2024

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.07.2024 Janek Falkenstein

Abstract

This thesis adapts the Soft Actor-Critic algorithm for Bayesian games as a method to learn Bayesian Nash Equilibria. Additionally, the proposed method is extended with Normalizing Flows strategies, allowing for the representation of arbitrary, mixed strategies whose densities can be evaluated. The approach is validated across two auction games and two Blotto games with known analytical pure Equilbria. The experiments test various algorithm settings and demonstrate that both approaches effectively approximate pure Bayesian Nash Equilibria. Significant findings include the impact of the entropy parameter on results and the ineffectiveness of the original algorithms autotuning in this context. While evaluation scores are slightly lower than alternative algorithms, the proposed method achieves faster iteration times and holds potential for improved accuracy through better entropy parameter optimization. In addition, this algorithm offers significant advantages over existing methods by avoiding discretization, learning arbitrary mixed strategies, and enabling analytically evaluation of the strategy density, making it versatile for a wide range of applications.

Kurzfassung

Diese Masterarbeit passt den Soft Actor-Critic Algorithmus für Bayessche Spiele an, um Bayessche Nash-Gleichgewichte zu lernen. Zusätzlich wird die vorgeschlagene Methode mit Normalizing Flows Strategien erweitert, die es ermöglichen beliebige gemischte Strategien darzustellen und deren Wahrscheinlichkeitsdichte zu berechnen.

Der Ansatz wird in zwei Auktionsspielen und zwei Blotto-Spielen mit bekannten analytischen und reinen Gleichgewichten validiert. Die Experimente testen verschiedene Algorithmus-Anpassungen und zeigen, dass beide Ansätze reine Bayessche Nash-Gleichgewichte effektiv annähern. Wichtige Ergebnisse beinhalten den Einfluss des Entropie-Parameters auf die Resultate und die Unwirksamkeit des ursprünglichen Algorithmus' Autotuning in diesem Kontext.

Obwohl die Bewertungsergebnisse etwas niedriger sind als beim alternativen Algorithmus, erreicht die vorgestellte Methode schnellere Iterationszeiten und hat das Potenzial für verbesserte Genauigkeit durch bessere Optimierung des Entropie-Parameters. Darüber hinaus bietet dieser Algorithmus bedeutende Vorteile gegenüber bestehenden Methoden, indem er Diskretisierung vermeidet, beliebige gemischte Strategien lernt und eine analytische Bewertung der Strategiedichte ermöglicht, was ihn vielseitig für eine Vielzahl von Anwendungen macht.

Contents

1. Introduction

The *Nash Equilibrium* (NE) is an important solution concept in game theory (Holt et al. 2004). It is fundamental in analyzing and predicting the outcome of strategic interactions in various fields, including economics and politics. Since the introduction of the NE (Nash Jr 1950), many different methods to find equilibria have been developed. These methods often use iterative processes and collect experience over time to improve each agent's strategy. In such a learning method, an NE represents a state where no agent benefits from unilaterally deviating, thereby halting further learning by any single agent. Conversely, if a learning method does not converge to an NE, some learning agents would benefit from changing their strategy to improve payoffs. Nevertheless, convergence to an NE of learning processes is not trivial, even with rational agents. In fact, learning dynamics can result in never-ending cycles, diverge, or be chaotic (Sanders et al. 2018; Fudenberg et al. 2009; Benaım et al. 1999; Foster et al. 1997).

Understanding the learning dynamics and developing algorithms that converge quickly to NEs is especially influential in the realm of auction theory. For example, digital advertising auctions are a large and growing market where automatic learning bidding agents are applied (Choi et al. 2020). Results in this field can help bidders find an optimal strategy or researchers to design and analyze auctions.

Auctions are modeled as *Bayesian games* (Harsanyi 1967). In Bayesian games, each bidder draws a type from a probability distribution and plays an action based on this information. The players have incomplete information about each other: they know the probability distribution but not the specific types drawn. In auction games, the types are valuations for the items, and the action is a bid. Most commonly, the type and the action spaces are continuous. Besides auctions, these types of games can be applied to any scenario where modeling uncertainty about competitors' preferences is necessary. The analog to an NE in Bayesian games is the *Bayes Nash Equilibrium* (BNE).

With uncertainty about the preferences of competitors, the learning dynamics in Bayesian games are even more complex than for games with complete information. BNEs have only been proven to exist under specific conditions, and it remains uncertain when a BNE exists in general auction games with continuous type and action spaces (Carbonell-Nicolau et al. 2018; Athey 2001). Consequently, for realistic markets and auctions, the BNEs are often intractable.

Despite discouraging results regarding theoretical convergence, recent learning algorithms for Bayesian games have converged to verifiable BNEs in various auction games with continuous type and action spaces (Bichler et al. 2021; Bichler et al. 2023a; Li et al. 2021; Martin et al. 2022). However, their limitations include focusing on pure strategies or discretizing the action and type spaces. Discretization can limit solution quality (Kroer et al. 2015). Pure strategies are deterministic and always play the same action for a given type. Some Bayesian games do not have pure BNE but only a mixed BNE where strategies are distributions over

possible actions (Gairing 2008). This work presents an algorithm without discretization that can represent complex mixed strategies with evaluable probabilities.

Like recent successful algorithms, we use neural networks and apply gradient-based updates to handle the complexity of calculating exact best responses in Bayesian games. Gradient-based methods are versatile, easy to implement (Mazumdar et al. 2020), and evolve smoothly, reducing unwanted biases (Shamma et al. 2005). A challenge with gradient-based updates is the absence of explicit gradient information in most games. Specifically, an agent can only observe outcomes and not access the reward function. To overcome this problem in Bayesian games, researchers applied a form of the natural evolution strategies algorithm (Wierstra et al. 2014; Bichler et al. 2021; Li et al. 2021; Martin et al. 2022). In contrast, we adopt a popular *reinforcement learning* (RL) algorithm to find equilibria in Bayesian games.

RL literature has developed various methods to address the challenge of unavailable gradients. Even though traditional RL is mostly used in single-agent problems where the environment remains stationary, it has been increasingly applied to multi-agent settings (Zhang et al. 2021). In Bayesian games, each agent's type can be modeled as the observation of a Markov game, which is typically used in RL. Therefore, we believe RL is a promising method for developing algorithms capable of solving BNEs.

Specifically, we adapt *Soft Actor-Critic* (SAC) (Haarnoja et al. 2018b) to Bayesian games. SAC is stable and sample-efficient and a popular RL algorithm. It is designed for continuous action and type spaces, making it well-suited for many Bayesian games and auctions. During the learning phase, SAC trains a Gaussian distribution over actions. Typically, the mean of this distribution is taken as the final pure strategy. This vanilla approach can be valid for finding pure BNEs or mixed BNEs that follow a Gaussian distribution.

However, for learning complex mixed BNEs, a Gaussian distribution might not be sufficient. Multi-agent tasks often require strategies whose distributions cannot be well approximated by a Gaussian distribution (Ma et al. 2020). One method to represent and learn complex distribution is *Normalizing Flows* (NFs) and has been shown to be an advantage by representing more flexible distributions (Ma et al. 2020).

Multiple studies have extended SAC with NFs and showed that this can benefit exploration in a single-agent environment (Guerra et al. 2020; Mazoure et al. 2020; Ward et al. 2019). The properties of NF layers ensure that the density of a given action-type pair can be evaluated. Further, NFs can be used to explicitly model environmental constraints (Brahmanage et al. 2024; Rietz et al. 2024). We show that SAC extended to NF strategies and applied by each agent individually can be used to learn complex strategies and mixed BNEs.

The contributions of this thesis are summarized as follows:

1. **Adapting SAC for Bayesian Games:** We adapt SAC for Bayesian games and evaluate its performance on a set of Bayesian games with known analytical BNEs. This flexible architecture can handle continuous action and type spaces without discretization. The experiments show that it can rapidly and accurately approximate pure equilibria.

2. **Extending SAC for Bayesian Games with NFs:** We enhance SAC for Bayesian games to represent arbitrary, mixed strategies. NFs enable the evaluation of action-type pair

densities. We validate our approach by testing it on the same set of games with pure BNEs and demonstrate that it is equally effective in converging to known BNEs.

The following Chapter 2 reviews related work, focusing on equilibrium learning, particularly for Bayesian games and gradient-based methods. It also covers relevant findings from RL and explores the use of NFs in RL. Chapter 3 introduces notation, the games used in the experiments, and the formal definition of NFs. Chapter 4 introduces the adaptation of SAC for Bayesian games and its extension with NFs. It includes the experimental setup and training and evaluation procedures. Results are presented in the subsequent Chapter 5, followed by a discussion and suggestions for future research in Chapter 6. The thesis concludes by examining the implications of the findings in the final Chapter 7.

2. Related Work

Our proposed method combines research from various fields. The first part of this chapter covers the basics of learning in games with multiple agents, revisits fundamental NE algorithms, and displays convergence results. The focus lies on gradient learning and Bayesian games. The second part explores related work in RL, including policy gradient methods. Finally, this chapter examines the capabilities of NFs and their previous applications in RL.

2.1. Equilibrium Learning

Numerous methods for finding equilibria (i.e., solving games) have been developed. Some methods, such as the Lemke-Howson algorithm (Lemke et al. 1964) and the Support Enumeration algorithm, compute exact NEs. However, these algorithms are often restricted to specific settings (e.g., two-player games) or do not scale well regarding the number of players or actions.

Therefore, most game-solving algorithms are iterative, where multiple agents repeatedly play and learn to maximize their payoffs (Fudenberg et al. 2009). This approach can be called adaptive learning, equilibrium learning, or simply learning in games. The two basic iterative learning approaches are best response dynamics (e.g., Fictitious Play (G. W. Brown 1951; Robinson 1951)) and regret minimization (Hart et al. 2000). In Fictitious Play, players adjust their strategy based on the historical frequency of their opponents' past actions. In its simplest form, this method can solve zero-sum normal-form games. Many modern algorithms build on this best response principle.

A comprehensive framework defining which algorithms converge and which games are learnable does not exist. The question of convergence from general learning mechanisms to an equilibrium is very difficult and unanswered.

It has been shown that finding an NE in finite, complete information games is a PPAD-complete problem (Daskalakis et al. 2009). When multiple agents learn simultaneously, they create a non-stationary environment for each other, and the resulting dynamics are highly complex (Andrade et al. 2021). Additionally, there is the problem of multiple equilibria in some environments. Individual learning agents would have to coordinate on the same equilibrium to converge. Thus, learning dynamics of rational agents can result in never-ending cycles, diverge, or be chaotic (Sanders et al. 2018; Fudenberg et al. 2009; Benaım et al. 1999; Foster et al. 2001).

Nevertheless, specific findings detail under which conditions learning algorithms are guaranteed to converge, cannot converge, or occasionally fail to converge. For example, Fictitious Play has been proven to converge in any two-player zero-sum game with a finite

number of strategies (Robinson 1951), for potential games (Monderer et al. 1996), or if the game has generic payoffs and one agent does have exactly two actions to choose from (Berger 2005). In contrast, Fictitious Play can cycle indefinitely in non-zero-sum games (Shapley et al. 1963). Further findings include that uncoupled dynamics, where each player bases decisions solely on their own past actions and payoffs, cannot be guaranteed to converge to an equilibrium (Hart et al. 2003) and that regret minimization converges to a so-called coarse correlated equilibrium (Hart et al. 2000; Jafari et al. 2001). There are also some guarantees of convergence to an NE when monotonicity conditions hold (Mertikopoulos et al. 2019; Bichler et al. 2023c).

Numerous other convergence results are beyond the scope of this work. The literature on learning algorithms is extensive (Fudenberg et al. 2009). In conclusion, learning dynamics are known to converge to an NE only in specific settings. Next, we examine findings relevant to this thesis, focusing on algorithms that use gradient updates and those that solve Bayesian games.

2.1.1. Learning in Bayesian Games

Complete information games and their convergence and possible algorithms have been studied extensively (Fudenberg et al. 2009). In contrast, the literature on Bayesian games is sparse.

Theoretical Guarantees in Bayesian Games

With uncertainty about the preferences of competitors, the learning dynamics in Bayesian games are even more complex than in complete information games. In particular, solving Bayesian games with continuous type and action spaces is challenging.

In fact, BNEs have only been proven to exist in a limited set of auctions (Jackson et al. 2005) and it remains uncertain when a BNE exists in general auction games with continuous types and actions (Carbonell-Nicolau et al. 2018). While some games do not have any BNE (Hellman 2014) other games only have a mixed BNE (Gairing 2008). Pure BNEs only exist under certain conditions (Athey 2001) and determining whether a pure BNE exists in a Bayesian game is NP-complete (Conitzer et al. 2008). Specifically, computing pure BNE for simultaneous Vickrey auctions is PP-hard, a complexity class close to PSPACE (Cai et al. 2014). Bichler et al. (2023c) leverage the fact that finding an equilibrium in auctions is equivalent to finding the solution of an infinite-dimensional variational inequality. These inequalities have promising convergence properties for gradient optimization under monotonicity conditions. However, Bichler et al. (2023c) showe that the second- and the first-price auctions are not monotonous.

In conclusion, the theoretical guarantees for finding BNEs are far from promising. However, verifying a BNE ex-post is possible (Bichler et al. 2021). This means that even if an analytical BNE is not known, the result of an algorithm can be validated.

Algorithms to Solve Bayesian Games

One set of algorithms for Bayesian games uses simple best-response methods without gradient learning but imposes strong restrictions. These restrictions apply to the action space (Naroditskiy et al. 2007; Rabinovich et al. 2013), strategy space (Bosshard et al. 2020; Reeves et al. 2004), or games themselves (Vorobeychik et al. 2008) and simplify the calculation of the exact best response. Similarly, Ceppi et al. (2009) experiment with a support enumeration algorithm on a restricted discrete game.

More recent learning algorithms for Bayesian auction games with continuous actions and types use complex strategy representations and apply gradient-based updates to manage the intractability of calculating exact responses (Bichler et al. 2021; Bichler et al. 2023a; Li et al. 2021). Despite discouraging results regarding theoretical convergence, these algorithms converge in various auction games. The reasons behind the convergence is not well understood. These algorithms encourage the exploration and empirical testing of new methods.

Most of these works represent strategies as neural networks and base their algorithms on natural evolution strategies (Wierstra et al. 2014). Bichler et al. (2021) introduce *Neural Pseudogradient Ascent* (NPGA) as a method to solve for pure BNEs in symmetric auctions games and later extended their method to asymmetric auction (Bichler et al. 2023b). Li et al. (2021) also focus on symmetric auctions and solve for mixed BNEs by discretizing the action space in a second step. In contrast to these works, our method can learn arbitrarily mixed strategies and does not need to be discretized.

Simultaneous Online Dual Averaging (SODA) (Bichler et al. 2023a) is a method for auction games that learns distributional strategies (Milgrom et al. 1985), a form of mixed-strategies. The method discretizes the action and type spaces and the distributional strategies. Thus, strategies can be represented explicitly instead of using neural networks, and online convex optimization algorithms (e.g., dual averaging) can be used. Their empirical results show rapid convergence to approximate BNEs in various symmetric auctions. In addition, the authors show that if their algorithms converge to a pure strategy, it is an approximate equilibrium in the discretized game, and an equilibrium in the discretized game is an approximated equilibrium in the continuous game. In contrast to our method, this approach discretizes the game.

Martin et al. (2022) use a parallelized natural evolution strategy method to speed up computation time and include a random noise as input to their strategy network to enable mixed strategies. They demonstrate the effectiveness of their method on complete and incomplete information games (e.g., Colonel Blotto Games or Chopstick auctions). Their algorithm resembles the method proposed in this thesis, allowing for continuous actions and types and mixed strategies. However, it employs a standard neural network with random noise, which does not support evaluating the density of an action given a type. In contrast, we use NFs with a base distribution conditioned on the type. Additionally, their algorithm is based on natural evolution strategies, whereas our method leverages SAC.

2.1.2. Gradient Based Learning with Neural Networks

Gradient learning with neural networks is an algorithm type that has recently gained popularity, as seen in the recent BNE solvers. Agents use approximation functions (e.g., neural networks) for their own or opponents' strategies, and learning occurs through gradient descent on some objective function, often towards the best response. Gradient learning with neural networks is effective in games where complete update methods are intractable. For example, games with large, complex environments and continuous actions, states, or observations.

General convergence results also apply to gradient-based learning algorithms. Consequently, gradient learning can result in cycling or chaotic dynamics (Vlatakis-Gkaragkounis et al. 2020; Mertikopoulos et al. 2018a). Additionally, the combined gradient dynamics in the multi-agent setting do not necessarily correspond to a gradient flow (Mazumdar et al. 2020). Therefore, convergence guarantees for gradient flow do not apply to multi-agent gradient learning. However, some convergence guarantees exist for multi-agent gradient learning. For example, for convex optimization problems (Mertikopoulos et al. 2018b) or in simple two-player and two-action games (Singh et al. 2000).

2.2. Reinforcement Learning

Game theory studies learning in games with a focus on theoretical convergence guarantees and the dynamics that arise when multiple agents learn. In contrast, RL develops algorithms for specific environments, often prioritizing practical results for a single agent in a stationary environment. In such a setting, issues associated with non-stationary environments do not occur.

The games in RL are usually modeled as Markov games, called *Markov Decision Processes* (MDPs) in the case of a single agent, which can model many interesting sequential decision-making tasks (Bertsekas et al. 1996; Sutton et al. 1998). Unlike simpler repeated normal-form games studied in game theory, they involve different states and state transitions.

Value-based RL approaches aim to find the value of each state-action pair and then select the action with the highest value in a given state. *Policy-based* approaches learn the mapping from state to action directly. Policy-based algorithms can be either *on-policy*, learning only from samples generated by the current policy to stabilize training, or *off-policy*, utilizing past experiences from older policies to enhance sample efficiency.

A major challenge in RL is the exploration-exploitation tradeoff. It describes balancing the exploration of all possible policies with the exploitation of strategies that yield better outcomes.

2.2.1. Multi-Agent Reinforcement Learning

While many RL algorithms are developed for single-agent environments, they can also be applied to multi-agent settings by deploying them to each agent individually, a technique known as independent learning (M. Tan 1993). Recently, multi-agent learning has gained

popularity in the RL community (Zhang et al. 2021). Most of the advances are possible because deep neural networks approximate the policy or value function.

Multi-agent learning is often used for self-play where symmetric agents compete against each other to develop one final agent for games like poker, chess, Go, or autonomous driving (N. Brown et al. 2019; Silver et al. 2018; Silver et al. 2016; Vinyals et al. 2019; Shalev-Shwartz et al. 2016; Heinrich et al. 2016). Other approaches include algorithms that share networks or gradients between agents to model cooperative behavior (Foerster et al. 2018; Lowe et al. 2017; J. K. Gupta et al. 2017) and settings without symmetric agents (Pan et al. 2022).

2.2.2. Policy Gradient Algorithms

Policy-based approaches typically represent policies by function approximations such as neural networks. The core idea is to update the parameters of these functions along the gradient of the long-term reward, a method known as *Policy Gradient*. However, agents cannot access full gradients in reinforcement learning because they do not know the function mapping actions to rewards. Therefore, various methods exist to estimate the gradients, including REINFORCE (Williams 1992), actor-critic algorithms (Konda et al. 1999; Bhatnagar et al. 2009), and other vanilla policy gradient methods (Sutton et al. 1999; Baxter et al. 2001). Several advanced methods have successfully addressed issues like sample inefficiency, achieving great success in numerous applications (e.g., DDPG (Silver et al. 2014), PPO (Schulman et al. 2017), and TRPO (Schulman et al. 2015)).

One advanced method to estimate policy gradients is the SAC algorithm (Haarnoja et al. 2018b). It combines off-policy learning with a stable actor-critic architecture, effectively addressing the exploration-exploitation tradeoff and resulting in a stable, sample-efficient algorithm (Haarnoja et al. 2018c). SAC excels in continuous state and action spaces, making it well-suited for the experimental environment in this thesis. Although initially developed for single-agent environments and MDPs, we adapt it for Bayesian games in a multi-agent setting.

2.3. Normalizing Flows in Reinforcement Learning

NFs transform a simple base distribution into a more complex distribution using a series of invertible and differentiable transformations (i.e., a diffeomorphism) (Rezende et al. 2015). Specially designed invertible neural networks can parameterize these transformations, allowing for efficient sampling and density evaluation. NFs have been especially successful in generative modeling (Kobyzev et al. 2020) but have also shown promising results in various RL settings.

Boborzi et al. (2021) applied NFs to adversarial imitation learning. Tang et al. (2018) showed that incorporating NFs into Trust Region Policy Optimization (TRPO) outperforms baseline architectures, especially in high-dimensional tasks with complex dynamics. Luo et al. (2021) used NFs in value-based distributional RL to estimate the distribution of expected rewards.

Brahmanage et al. (2024) utilize NFs trained in a supervised manner to convert unconstrained actions into valid actions, subsequently adding these NFs to actor-critic algorithms. Rietz et al. (2024) propose using domain knowledge to explicitly encode constraints into the NFs. Similarly, Chen et al. (2024) tackle action constraints by introducing a framework that pairs an actor-critic method with NFs, rejecting invalid actions during training.

Haarnoja et al. (2018a) use the SAC algorithm to train hierarchical policies, highlighting that the invertibility of NFs allows higher layers to retain full expressiveness. Several papers have used SAC as the base algorithm to integrate NFs into RL, aiming to enhance exploration during training (Guerra et al. 2020; Mazoure et al. 2020; Ward et al. 2019). Additionally, Delalleau et al. (2019) extended SAC with NFs to handle a mix of discrete and continuous actions.

These promising results demonstrate the effectiveness of combining SAC with NFs and form the basis of the method tested in this thesis. However, all of these studies are done in the single-agent setting and for MDPs. Their goal is not to converge to an equilibrium with multiple learning agents. In contrast, we adapt SAC with NFs to Bayesian games with the goal of converging to a BNE in a multi-agent setting.

3. Background

This chapter introduces notation and reviews Bayesian games, NFs, and explains the formal games that are studied in the experiments of this thesis.

3.1. Bayesian Games

A *Bayesian game*, also known as an incomplete information game, is represented by a quintuple $G = (\mathcal{I}, \mathcal{A}, \mathcal{V}, F, u)$. Here, $\mathcal{I} = \{1, \ldots, n\}$ denotes the set of agents participating in the game. The possible action profiles are given by $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, with $\mathcal{A}_i \in \mathbb{R}^D$ representing the D-dimensional continuous actions space of agent $i \in \mathcal{I}$. The type or observation profiles, which are also continuous, are represented by $\mathcal{V} = \mathcal{V}_1 \times \cdots \times \mathcal{V}_n$, where $\mathcal{V}_i$ denotes the type space for agent i. The joint probability density function over type profiles is denoted by $F : \mathcal{V} \to [0, \infty)$, which is assumed to be common knowledge among all agents in Bayesian games.

Before the game, *ex-ante*, agents do not know their type but the probability distribution F. At the start of the game, a type profile $v \sim F$ is drawn. Each agent i is informed only of their own type $v_i \in \mathcal{V}_i$, leading to the *interim* stage of the game. Agents use this private information to select their actions $a_i \in \mathcal{A}_i$. Once agents are informed about the outcome of their utility function, the game is in the *ex-post* stage. The (ex-post) utility function of an agent, $u_i : \mathcal{A} \times \mathcal{V}_i \to \mathbb{R}$, depends on the actions of all agents but only on their own type.

Agents aim to maximize their individual utility or payoff u_i from an ex-ante view by learning some form of strategy. A *mixed strategy* for agent i is a function $\sigma_i : \mathcal{V}_i \to \Delta(\mathcal{A}_i)$, where $\Delta(\mathcal{A}_i)$ denotes the set of probability distributions over the action space $\mathcal{A}_i$. With $\sigma(a_i|v_i)$ we denote the probability of choosing action a_i given type v_i. In contrast, a mixed strategy in RL literature is known as a stochastic policy often denoted by π.

Let a_{-i}, v_{-i}, and σ_{-i} represent the action, type, and strategy profiles for all agents except agent i. The *ex-ante utility* is defined as the expected utility over the type distribution F and the action distributions induced by the mixed strategies:

$$\tilde{u}_i(\sigma_i, \sigma_{-i}) = \mathbb{E}_{v_i \sim F}\left[\mathbb{E}_{a_i \sim \sigma_i(v_i), a_{-i} \sim \sigma_{-i}(v_{-i})}\left[u_i(a_i, a_{-i}, v_i)\right]\right]. \tag{3.1}$$

A strategy profile $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a *Bayesian Nash Equilibrium* (BNE) if, for every agent i, the strategy σ_i maximizes the agent's ex-ante utility given the strategies of the other agents. Formally,

$$\tilde{u}_i(\sigma_i, \sigma_{-i}) \geq \tilde{u}_i(\sigma_i', \sigma_{-i}) \quad \forall \sigma_i' \text{ and } \forall i \in \mathcal{I}. \tag{3.2}$$

An *ϵ-BNE* is a strategy profile $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ where each agent's strategy σ_i ensures their ex-ante utility is within ϵ of the maximum possible utility given the current strategies of their

opponents. Formally,

$$\tilde{u}_i(\sigma_i, \sigma_{-i}) \geq \tilde{u}_i(\sigma_i', \sigma_{-i}) - \epsilon \quad \forall \sigma_i' \text{ and } \forall i \in \mathcal{I}. \tag{3.3}$$

3.2. Selected Bayesian Games

We test our method on Bayesian games where the BNE can be computed analytically. Specifically, we evaluate our approach using the following two auction and two Blotto games. Throughout all games, ties are resolved by randomly selecting a winner.

3.2.1. First Priced Sealed Bid Auction

In a *First Priced Sealed Bid Auction* (FP auction), each agent $i \in \mathcal{I}$ receives a private valuation. The bids $a_i \in \mathcal{A}_i$ are submitted simultaneously. The agent with the highest bid wins the item and pays their bid amount. The (ex-post) utility function for agent i is given by:

$$u_i(a_i, a_{-i}, v_i) = \begin{cases} v_i - a_i & \text{if } a_i > a_j \, \forall j \neq i \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

With the valuations v_i uniformly distributed between 0 and 1, Vickrey (1961) demonstrated that the pure BNE for this auction with two players is:

$$a_i = \frac{1}{2} \cdot v_i. \tag{3.5}$$

3.2.2. All Pay Auction with Private Valuations

In an *All Pay Auction with Private Valuation* (AP auction), each agent $i \in \mathcal{I}$ observes a private valuation and submits a bid $a_i \in \mathcal{A}_i$, and all agents pay their bid regardless of who wins. The agent with the highest bid wins the item. The utility function for agent i is given by:

$$u_i(a_i, a_{-i}, v_i) = \begin{cases} v_i - a_i & \text{if } a_i > a_j \, \forall j \neq i \\ -a_i & \text{otherwise} \end{cases} \tag{3.6}$$

As proven by Noussair et al. (2006), with valuations v_i uniformly distributed between 0 and 1, the BNE for this auction is:

$$a_i = \frac{1}{2} \cdot v_i^2. \tag{3.7}$$

3.2.3. Blotto Game with Incomplete Information

In a *Blotto game with incomplete information* (Blotto 1D) as defined by Adamo et al. (2009), each player $i \in \mathcal{I}$ has a private budget v_i drawn independently and uniformly from the interval $[0, 1]$. There are N prizes, each with a value W_k for $k \in N$, such that $\sum_{k=1}^{N} W_k = 1$. Players allocate their budgets across the N prizes simultaneously. Hence, their action $a_i \in \mathcal{A}_i$

is N-dimensional, representing the allocation of resources to each prize. The player who allocates the most resources to a prize wins that prize. The utility function for player i is defined by:

$$u_i(a_i, a_{-i}, v_i) = \sum_{k=1}^{N} W_k \cdot I(a_{ik} > a_{jk} \, \forall j \neq i) - a_{ik}, \tag{3.8}$$

where $I(\cdot)$ is an indicator function that is 1 if player i wins prize k and 0 otherwise.

The pure-strategy BNE for this game, as shown by Adamo et al. (2009), is given by:

$$a_i = (W_1 v_i, W_2 v_i, \ldots, W_N v_i). \tag{3.9}$$

3.2.4. Blotto Game with Multi-Dimensional Incomplete Information

In a *Blotto game with multi-dimensional incomplete information* (Blotto 3D) as defined by Kovenock et al. (2011), each player $i \in \mathcal{I}$ has a private valuation vector $v_i \in \mathbb{R}^N$ drawn independently from the distribution $\hat{P}$. To define $\hat{P}$, let $N = 3$ and $\mathcal{S}$ denote the set of points on the surface of the unit sphere in the first octant $\mathbb{R}^3_+$:

$$\mathcal{S} = \left\{ v \in \mathbb{R}^3_+ \mid \sum_{j=1}^{3} (v_j)^2 = 1 \right\}. \tag{3.10}$$

The distribution $\hat{P}$ is a 3-variate distribution defined by uniformly distributing mass over the set of points in $\mathcal{S}$.

The utility function for this game is the same as in the previous Blotto game. The pure-strategy BNE for this game, as shown by Kovenock et al. (2011), is given by:

$$a_i = \left(v_{i1}^2, v_{i2}^2, v_{i3}^2\right). \tag{3.11}$$

3.3. Normalizing Flows

The basic idea of NFs is to transform a sample from a simple base distribution $p_u(u)$ (e.g., a Gaussian) into a sample from a more complex distribution $p_x(x)$ using a transformation T:

$$x = T(u), \quad u \sim p_u(u) \tag{3.12}$$

The transformation T and its inverse are invertible and differentiable. Such transformations are called diffeomorphism. This allows to evaluate the density of the transformed distribution $p_x(x)$ by using the change of variable formula. Given two transformations T_1 and T_2, their composition $T_2 \circ T_1$ remains invertible and differentiable. Thus, composing K simpler transformation can create complex transformations without losing invertibility, differentiability, or the ability to calculate the density $p_x(x)$.

Let Σ_k be the Jacobian matrix of T_k^{-1} with respect to the output of T_{k-1}. Using equation the change of variable formula the final log density of x is given by:

$$\log p(x) = \log p_u(u) + \sum_{k=1}^{K} \log \det(\Sigma_k), \tag{3.13}$$

There are various ways to construct a flow. Since transformations are composable, a simple transformation model often serves as the base building block. While the transformations must be theoretically invertible and differentiable, it is important that both operations are also tractable. Particularly, the Jacobian determinant should be computable quickly, with most implementations allowing linear computation time relative to the input dimension.

One class of NFs is the autoregressive flows (see Figure 3.1). Let z_i denote the transformation and z'_i denote the transformation output at dimension i. Autoregressive flows consist of a *transformer* τ and a *conditioner* c_i. The transformer must be a strictly monotonic function and is parmaeterized by the conditioner. The constraint of c_i is that it can only use inputs $z_{<i}$ with dimensions smaller than i. This relationship is formalized as:

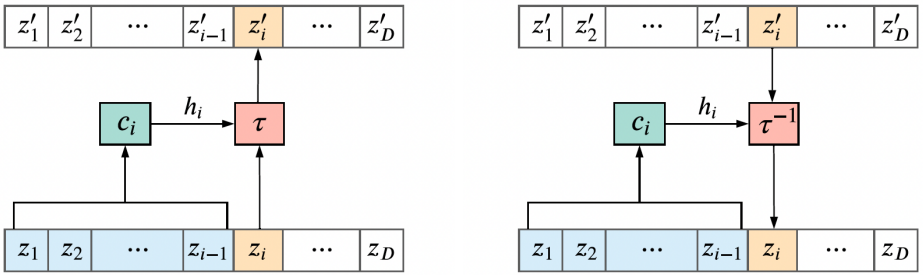
$$z'_i = \tau(z_i; h_i) \quad \text{where} \quad h_i = c_i(z_{<i}). \tag{3.14}$$

Different implementations of the conditioners and the transformer can be combined arbitrarily.

One implementation of the conditioners c_i is an independent neural network for each dimension. However, this approach does not scale well with increasing input dimensions. Another solution is the *coupling layer* (Dinh et al. 2016; Dinh et al. 2014), a special case of an autoregressive flow where the conditioner is not specified for each dimension i but remains constant and always takes as input all z with indexes lower than a certain threshold d. Only the inputs with indexes $> d$ can be transformed. Often, $d = \frac{D}{2}$ with D being the input dimensionality. To ensure all dimensions interact with each other, the dimensions are often permuted between layers. The concrete coupling layer $c(z_{<d})$ can be implemented as a simple neural network.

Multiple variants to implement the transformer τ exist. Affine transformers like Real NVP (Dinh et al. 2016) and Glow (Durk P Kingma et al. 2018) have a tractable inverse and Jacobian determinant but are limited in their expressiveness. Other implementations are more expressive but cannot be inverted efficiently (Papamakarios et al. 2021).

In contrast, *Neural-spline flows* (NSFs) (Durkan et al. 2019) implement autoregressive NFs with an analytical, fast-to-compute inverse and a tractable Jacobian determinant while being highly expressive. In NSFs, the transformer τ is parameterized by monotonic rational-quadratic splines. Splines are mathematical functions defined piecewise by polynomials, ensuring smooth transitions at the points where the polynomial segments connect. Rational quadratic splines in NSFs exploit monotonic segments of rational quadratic functions, making the transformer more expressive than other spline implementations. NSFs can be combined with coupling layers or other autoregressive flows and have been successfully applied to distributional RL (Luo et al. 2021).



(a) Forward (b) Inverse

Figure 3.1.: The i-th step of an autoregressive flow with conditioner c_i and reversible transformer τ. This figure, adapted from Papamakarios et al. (2021), illustrates both forward and inverse computations.

4. Methodology

This chapter outlines the derivation of the SAC algorithm, its adaptation to Bayesian games, and its extension with NF strategies. Additionally, it details the experimental setup, the objectives of the experiments, and the evaluation methods.

4.1. Soft Actor-Critic for Bayesian games

The SAC algorithm is designed for MDPs, not Bayesian games. We adapt the algorithm to our notation and games. Since the algorithm fundamentally changes for Bayesian games, this section explicitly states the transformed objective functions, update rules, and gradients. We will use the term strategy instead of policy.

SAC extends the standard RL objective, the (ex-ante) expected utility, with an entropy term that aims to maximize the randomness of the chosen strategy. This entropy term will stay the same for Bayesian games, but the expected utility can be simplified. In MDPs, the expected utility sums over a time horizon and depends on future rewards, state-action marginals, and a discount factor, which is a nontrivial element in defining the objective function (Thomas 2014).

Bayesian games are one-shot without future rewards. The expected utility only depends on the probability distribution of the type, actions chosen by the strategy, and the immediate reward. Consequently, the entropy objective used in SAC can be formulated for a Bayesian game with one agent as follows:

$$\sigma^* = \arg\max_{\sigma} \mathbb{E}_{v\sim F}\left[\mathbb{E}_{a\sim\sigma(v)}\left[u(a,v) + \alpha\mathcal{H}(\sigma(v))\right]\right] \tag{4.1}$$

with σ^* being the optimal strategy. The entropy parameter α determines the relative importance of the entropy and thus defines how random the strategy will be. Defining an optimal value for α is a critical challenge in the SAC algorithm.

4.1.1. Theoretical Foundations and Derivation

In line with Haarnoja et al. (2018b) to derive the algorithm, we first present the theoretical framework and then show how this can be approximated.

Theoretical Iterative Method

The basic idea of SAC is to find the Q-values of the current strategy and then update the strategy towards the exponential of the Q-values using a Boltzmann distribution. In RL

with MDPs, Q-values represent the expected reward for a given state-action pair and a fixed strategy, factoring in future rewards. Q-values in MDPs depend on the strategy because it determines the actions taken in the next state. Haarnoja et al. (2018b) update Q-values using an iterative Bellman backup operator and prove that this iterative process converges to the exact soft Q-values of the current strategy.

Unlike for MDPs, which account for discounted future rewards, the expected reward in Bayesian games depends solely on immediate utility due to their one-shot nature. This implies that Q-values are independent of the strategy, as no action selection is required in subsequent states. Therefore, in Bayesian games, Q-values represent the expected utility for a given type and action and are equivalent to the ex-post utility function of the game:

$$Q(v, a) = u(a, v). \tag{4.2}$$

With enough samples of $u(a, v)$, the Q-functions can estimate a current strategy's exact Q-values (i.e., the utility function).

To update the strategy toward the approximated Q-values, Haarnoja et al. (2018b) constrain the strategies to be in a distribution set Σ (e.g., a family parameterized distribution). Then, they minimize the Kullback-Leibler (KL) divergence between a strategy $\sigma \in \Sigma$ and the Boltzmann Distribution based on the Q-function and entropy weight α. We adjust their equation to our notations:

$$\sigma_{\text{new}} = \arg\min_{\sigma' \in \Sigma} \mathbb{D}_{\text{KL}} \left(\sigma'(v) \middle\| \frac{\exp\left\{ \frac{1}{\alpha} Q^{\sigma_{\text{old}}}(v, \cdot) \right\}}{Z^{\sigma_{\text{old}}}(v)} \right). \tag{4.3}$$

Iterating between updating the strategy and estimating Q-values is theoretically guaranteed to find the optimal solution regarding the objective function in Equation 4.1 (Haarnoja et al. 2018b).

Approximation with Gradient Descent

The exact calculation of the described iterative framework is only tractable in a tabular case. Approximations are necessary to represent the Q-function and the strategy in scenarios with large or continuous action and type spaces. Consequently, the SAC algorithm approximates this method using neural networks to represent the Q-function and the strategy. Instead of evaluating the Q-values and finding the exact minimum strategy with respect to the KL divergence, SAC implementation alternates between optimizing both networks using stochastic gradient descent. We will adjust the update rules for Bayesian games.

The Q-function parameters ψ in the original SAC for MDP games are trained by a loss that compares the current Q-value to the reward plus estimated discounted future rewards. SAC uses another set of so-called target Q-functions to estimate the value function and future rewards, which have been shown to stabilize training (Mnih et al. 2015). They become obsolete in Bayesian games.

Therefore, the loss for the Q-functions can be significantly simplified for Bayesian games without future rewards:

$$J_Q(\psi) = \mathbb{E}_{(v,a)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\psi(v,a) - u(a,v)\right)^2\right] \tag{4.4}$$

with $\mathcal{D}$ being the dataset (e.g., a batch from a replay buffer). This corresponds to a simple mean squared error between the estimated Q-values and the utility observed.

The gradients of this Q-function are given by:

$$\nabla_\psi J_Q(\psi) = \nabla_\psi Q_\psi(v,a)\left(Q_\psi(v,a) - u(a,v)\right). \tag{4.5}$$

To find the strategy that minimizes the KL divergence defined in Equation 4.3, SAC rewrites the KL divergence and uses it as the loss function directly. To find the strategy that minimizes the KL divergence in Equation 4.3, SAC uses the KL divergence directly as the loss function. Minimizing this loss function derives the desired strategy. Using the definition of the KL divergence, ignoring the partition function $Z^{\sigma_{\text{old}}}(v)$ (because it is constant with respect to the strategy), and multiplying by α, derives the following objective function for Bayesian games:

$$J_\pi(\phi) = \mathbb{E}_{v\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\sigma_\phi(v)}\left[\alpha\log(\sigma_\phi(a \mid v)) - Q_\psi(v,a)\right]\right]. \tag{4.6}$$

An advantage of SAC over other policy gradient methods is that the observed utility $u(a,v)$ retrieved from an unknown utility function of the environment is not part of the strategy loss function. Rather, the target density is the Q-function, which is a neural network. Thus J_π would be differentiable if the strategy would be deterministic.

The problem is that the strategy is non-differentiable due to a stochastic node. However, using the *reparametrization trick* (Diederik P Kingma et al. 2013), J_π can be made differentiable. The reparameterization trick transforms a stochastic variable into a deterministic one with added noise (see Figure 4.1). Specifically, for a variable a with distribution $p(a)$, it is reparameterized as $a = f_\phi(v,\epsilon)$, where ϵ is a noise variable with a simple distribution and ϕ are the parameters of the transformation function f_ϕ. For example, if $a \sim \mathcal{N}(\mu_\phi(v), \hat{\sigma}^2_\phi(v))$[1], we can reparameterize it as:

$$a = \mu_\phi(v) + \hat{\sigma}_\phi(v)\cdot\epsilon, \quad \epsilon \sim \mathcal{N}(0,1). \tag{4.7}$$

Applying the reparameterization trick, we can rewrite the loss function J_π as:

$$J_\pi(\phi) = \mathbb{E}_{v\sim\mathcal{D},\epsilon\sim p(\epsilon)}\left[\alpha\log(\sigma_\phi(f_\phi(v,\epsilon) \mid v)) - Q_\psi(v, f_\phi(v,\epsilon))\right] \tag{4.8}$$

where $a = f_\phi(v,\epsilon)$ with $\epsilon \sim p(\epsilon)$. This allows us to compute gradients with respect to ϕ and perform backpropagation through the strategy network. Given a type sample v and an epsilon sample ϵ, the gradients are the following:

$$\nabla_\phi J_\pi(\phi) = \nabla_\phi \alpha\log(\sigma_\phi(a \mid v)) + (\nabla_a \alpha\log(\sigma_\phi(a \mid v)) - \nabla_a Q_\psi(v,a))\nabla_\phi f_\phi(v,\epsilon) \tag{4.9}$$

with $a = f_\phi(v,\epsilon)$. Using Monte Carlo estimates, based on batches from a replay buffer, we can approximate the above gradients and perform stochastic gradient descent.

[1] To avoid confusion with our strategy $\sigma(v)$, we denote the standard deviation as $\hat{\sigma}$.

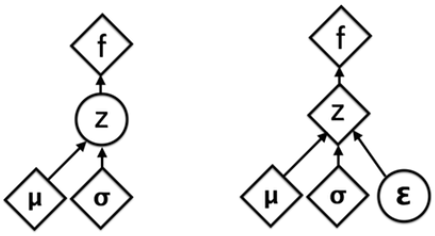


Figure 4.1.: Schematic representation of the reparameterization trick used in SAC to perform gradient descent on a stochastic function. Stochastic nodes are depicted as circles and deterministic nodes as diamonds.

In conclusion, the loss function of the strategy in SAC depends only on Q-function and not directly on observed utility. This, combined with the reparameterization trick, allows the use standard stochastic gradient descent for updates. Therefore, SAC has a lower variance estimator of the gradient than other policy gradient methods.

4.1.2. Strategy Representation

Haarnoja et al. (2018b) represent the strategy by a neural network that outputs a mean and standard deviation for each action dimension, forming a parameterized Gaussian.

Such a distribution is unbounded, but many RL tasks and Bayesian games actions are constrained to some finite interval. Therefore, SAC squashes the Gaussian samples by applying the hyperbolic tangent function tanh and adjusts the probability density function for the actions using the change of variable formula. The following shows the resulting mixed strategy:

$$u \sim \mathcal{N}(\mu_\phi(v), \text{diag}(\hat{\sigma}^2_\phi(v))) \tag{4.10}$$

$$\sigma_\phi(v) = \tanh(u). \tag{4.11}$$

This algorithm can be used with any parameterized distribution if it is differentiable, and the probability density can be evaluated for a value-action pair (Haarnoja et al. 2018b). This is where we extend the SAC algorithm by using NFs to represent the strategy.

SAC with NFs

It is possible to interpret the squashing of the tanh function as the first layer of an NF. We replace this transformation with a parameterized one. This approach has been implemented for MDPs (Haarnoja et al. 2018a; Guerra et al. 2020; Mazoure et al. 2020; Ward et al. 2019; Delalleau et al. 2019).

The NFs' base distribution $p_u(u)$ is defined by the mean and the standard deviation given the observed type v. A sample is drawn from this parameterized base distribution and pushed through the flows with parameters θ. We extend the mixed strategy to depend on θ: $\sigma_{\phi,\theta}(v)$.

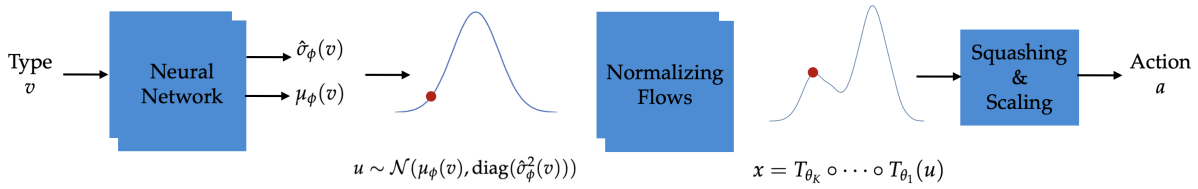


Figure 4.2.: A representation of the NF strategy: observing a type v, encoding it to a base distribution, transforming it with reversible NFs, and then squashing and scaling the NF sample to produce the final action a.

The final sample is squashed using tanh to adhere to the action constraints. The following equations enable sampling an action a based on an observation v, effectively defining the mixed strategy:

$$p_u(u) = \mathcal{N}(\mu_\phi(v), \mathrm{diag}(\hat{\sigma}^2_\phi(v))) \tag{4.12}$$

$$x = T_{\theta_K} \circ T_{\theta_{K-1}} \circ \cdots \circ T_{\theta_1}(u) \quad \text{with} \quad u \sim p_u(u) \tag{4.13}$$

$$\sigma_{\phi,\theta}(v) = \tanh(x) \tag{4.14}$$

The log density of an action a given the observation v is crucial for calculating the strategy gradient (see Equation 4.9). Using the change of variable formula from NFs (Equation 3.13) and accounting for the tanh squashing as in SAC, this log density can be evaluated effectively. The following describes the log density of an action a given the observation v under the current strategy:

$$\log(\sigma_{\phi,\theta}(a \mid v)) = \log p_u(u) + \sum_{k=1}^{K} \log\det(\Sigma_k) + \log\det(J_{\tanh^{-1}}(a)) \tag{4.15}$$

with Σ_k as the Jacobian matrix of T_k^{-1} with respect to the output of T_{k-1}.

Equation 4.14 enables sampling from a strategy, and Equation 4.15 evaluates the density of a strategy. Together, they are sufficient to prove that NFs can be incorporated into the SAC algorithm.

Incorporating Action Constraints

In all games of the experiments in this thesis, the actions are bounded to intervals from 0 to 1. The $\tanh(x)$ function squashes the actions to the range $[-1, 1]$. We then apply a transformation to scale and shift the actions to the desired range (i.e., $a = a \times \text{scale} + \text{bias}$) where both the scale and bias are set to 0.5. This could be adjusted for different action ranges. The log density is adjusted for this transformation by adding $\log(\text{scale})$ to the current density.

Blotto games have multi-dimensional actions that must sum up to the budget constraint. Therefore, we add a final softmax layer to the network to normalize the actions. The density adjustment is given by $\log\left(\sum_{k=1}^{n} a_k\right)$ where n is the number of action dimensions. In Blotto 3D, the budget is constant and equal to 1, making the output of the softmax sufficient. In Blotto

1D, the budget is dependent on the observation v_i, so we scale the output of the softmax by v_i and adjust the log density accordingly by $-\log(v_i)$.

4.1.3. Implementation Details

The SAC algorithm introduces further adjustments to optimize training (Haarnoja et al. 2018b). During a warm-up phase, SAC populates the replay buffer using random strategies. In addition, SAC uses two tricks from TD3 algorithms (Fujimoto et al. 2018). First, they use two Q-functions and take the minimum value to update the strategy. Second, they update the strategy less frequently than the Q-functions, introducing a hyperparameter. SAC also employs two target functions to estimate future rewards, which are unnecessary in Bayesian games and not part of our implementation.

4.1.4. Autotuning of the Entropy Parameter

The entropy parameter α determines the relative importance of the entropy term against the reward. Haarnoja et al. (2018b) state that α must be adapted to the reward scale. As the magnitude of the reward changes over time as the policy evolves, this is a non-trivial task. To address this, Haarnoja et al. (2018c) introduced an autotuning mechanism for α.

This approach is based on a minimum expected entropy, the target entropy, which is set to $-\dim(A_i)$, the dimension of the action space. A constrained optimization problem based on the target entropy constrains the strategies's average entropy but allows the entropy to vary at different states. The entropy parameter α is the dual variable of this optimization problem. Solving the dual problem is done via function approximations, and α is adjusted via gradient descent. Our implementation incorporates this autotuning method as default.

Using the implementation details in SAC for Bayesian games and applying independent multi-agent learning, we derive Algorithm 1.

4.2. Hyperparameters and Neural Network Architecture

In our implementation, the NF strategy utilizes one neural-spline flow layer (Durkan et al. 2019) with two hidden layers of 10 units each, implemented via the *normflows* package (Stimper et al. 2023). In NFs, the dimension of the base distribution must be the same as the target distribution. In Blotto 1D, the input is one-dimensional, while the output, the action, is three-dimensional. Therefore, we expand the input.

We conducted a hyperparameter tuning for the strategy and Q-function learning rates for each combination of game and strategy type, optimizing for the distance to the analytical BNE. In line with recommendations from Haarnoja et al. (2018c), the search interval for the Q-function learning rate was higher than that for the strategy learning rate. The learning rates did not exhibit a significant impact across different games. Table 4.1 lists the fixed hyperparameters used in the experiments. We also did preliminary experiments for the batch size and found 512 to be optimal.

Algorithm 1 SAC Algorithm for Bayesian games with multiple agents

Initialize environment, agents, replay buffers $\mathcal{D}_i$, and parameters ψ, ϕ, and α
while Training **do**
  for all Environment Steps **do**
    for all agents i **do**
      $v_i \sim F$
      if Warm-Up phase **then**
        $a_i \sim \mathcal{U}(\mathcal{A}_i)$
      else
        $a_i \sim \sigma_\phi(v_i)$ ▷ Use current strategy to select action
      end if
    end for
    for all agents i **do**
      $u = u_i(a, v_i)$
      $\mathcal{D}_i \leftarrow \mathcal{D}_i \bigcup (v_i, a_i, u)$
    end for
  end for
  for all Update Steps **do**
    for all agents i **do**
      $\psi_{i,k} \leftarrow \psi_{i,k} - \lambda_Q \nabla_{\psi_{i,k}} J_Q(\psi_{i,k}) \quad \forall k \in \{1, 2\}$ ▷ Update both Q-functions
      if Update Strategy **then** ▷ Strategy is updated less frequently
        $\phi_i \leftarrow \phi_i - \lambda_\sigma \nabla_{\phi_i} J_\sigma(\phi_i)$ ▷ Update strategy using min. Q-function
      end if
      if autotune **then**
        update α
      end if
    end for
  end for
end while

All other hyperparameters and neural network architectures are taken from Haarnoja et al. (2018c). The replay buffer size is 10^6. The optimizer is Adam (Diederik P Kingma et al. 2014). The Gaussian strategy is represented by a neural network with two hidden layers of 256 units each, followed by two linear layers to output the mean and log standard deviation. The networks for the two Q-functions also consist of two hidden layers with 256 units each.

Table 4.1.: Q-function and policy learning rates found during hyperparameter tuning and used in the experiments.

	Gaussian		Flow	
	Q-function lr	Policy lr	Q-function lr	Policy lr
FP	0.09340	0.00010	0.06620	0.00003
AP	0.08550	0.00012	0.08940	0.00068
Blotto 1D	0.03005	0.00035	0.03700	0.00046
Blotto 3D	0.00562	0.00005	0.03822	0.00040

4.3. Experiments

We apply the described SAC for Bayesian games algorithm with multiple players. Each player independently estimates Q-values and strategies. Since all agents are simultaneously learning, the environment is non-stationary. The experiments aim to understand the dynamics in this setting, whether SAC for Bayesian games converges to BNEs with Gaussian strategies and with NF strategies, and identify algorithm adjustments that enhance the results.

We conduct experiments with both strategy forms. The first set of experiments utilizes a Gaussian distribution, as is standard in SAC, while the second set extends SAC with NF strategies. We test both strategies with four games: FP auction, AP auction, Blotto 1D, and Blotto 3D. All games are played with two players. The Blotto games have three items. For Blotto 1D, the valuations are fixed to $W = (0.05, 0.025, 0.7)$. We expect the support for mixed strategies $\sigma_i(v_i)$ to concentrate on the pure BNE action.

Fixed Opponent In the first experiments, we fixed all but one agent to play the equilibrium strategy, creating a stationary environment. This setup helps evaluate the general usability of our method and helps identify the root cause of observations in the multi-agent setting. Issues solved in the single-agent setting suggest that they arise from the complex dynamics of multi-agent environments.

Entropy Parameter A crucial parameter in SAC is the entropy parameter α, which manages the balance between exploration and exploitation. SAC includes autotuning for this parameter, a feature we use as default in the experiments. However, autotuning is designed for a single-agent environment and could be less effective in multi-agent settings. Further, repeated

Bayesian games do not require as much exploration as MDPs with long trajectories. Hence, we investigate whether setting $\alpha = 0$ can enhance the learning of pure strategies. This setting aims to produce a distribution with lower entropy, prioritizing reward optimization over entropy. We also experimented with a fixed value $\alpha = 0.0025$ without autotuning, considering that a zero value might be overly restrictive and with an exponential decay starting with $\alpha = 0.25$ and ending at $\alpha = 0.001$ after 20,000 iterations.

Pre-Training Additionally, we tested pre-training where each agent's strategy outputs the identity, meaning agent i plays action $a_i = v_i$ given type v_i. In the context of auctions, this means we train the strategy for truthful bidding. This can be interpreted as providing the algorithm with initial information, enabling it to make more informed decisions from the start. One experiment combines setting $\alpha = 0$ with pertaining to see if pre-training is an alternative to the missing exploration caused by no entropy in the objective function.

Number of Gradient Steps In another set of experiments, we adjust the number of gradient steps after each game played. The basic setup repeatedly plays one game and takes one gradient step. Adjusting the number of gradient updates is a hyperparameter used by Haarnoja et al. (2018b). We test values 1, 2, 8, and 32. The total iterations are adjusted to ensure that each experiment ends after the same number of updates. Specifically, if we updated twice after each game, we conducted 10,000 iterations, each with two updates, instead of 20,000 iterations with one update.

Number of Games Similarly, we test the effect of adjusting the number of games played before each gradient step. Adjusting the number of games played simultaneously could increase stability during learning because the replay buffer is filled with more on-policy data. One experiment plays 1,000 games simultaneously, filling the replay buffer faster and with more data from the current policy. Another experiment plays 20,000 games simultaneously, and the batch and replay buffer sizes are also adjusted to 20,000. This means that every gradient step only uses experiences from the current strategy. Hence, the algorithm changes to an on-policy algorithm with a large batch size. In MDPs, rewards heavily depend on the strategy, as it determines which states are visited next, implying a substantial difference between on- and off-policy learning. In repeated Bayesian games, the reward remains independent of the current strategy. Nevertheless, in Bayesian games, we suspect that more on-policy samples could also make a difference as we learn the Q-values for the actions chosen by the current strategy more effectively.

4.4. Evaluation

We tracked two key metrics not used during training to evaluate our approach. The *root mean square error* (RSME) measures the Euclidean distance between the learned strategy σ

Table 4.2.: Summary of Experimental Settings and Properties. The rows represent different experiments, while the columns detail various settings.

	Fixed Opponent	Entropy Parameter	Pre-Training	Gradient Steps	Number of Games
Basic Setup	×	autotuning	×	1	1
Stationary Environment	✓	autotuning	×	1	1
$\alpha = 0$	×	0	×	1	1
$\alpha = 0.0025$	×	0.0025	×	1	1
Decaying α	×	Decay	×	1	1
Pre-Train	×	autotuning	✓	1	1
Pre-Train + $\alpha = 0$	×	0	✓	1	1
Updates 2	×	autotuning	×	2	1
Updates 8	×	autotuning	×	8	1
Updates 32	×	autotuning	×	32	1
Games 1,000	×	autotuning	×	1	1,000
Games 20,000	×	autotuning	×	1	20,000

and equilibrium strategy σ^* averaged across a batch of 2^{16} samples:

$$L_2(\sigma_i) = \left(\frac{1}{n_{\text{batch}}} \sum_{v_i} (\sigma_i(v_i) - \sigma^*(v_i))^2 \right)^{1/2}. \tag{4.16}$$

A utility loss assesses the difference in (ex-ante) utility between the learned and equilibrium strategies, providing insight into strategy effectiveness. Based on Bichler et al. (2021), we estimate the BNE-utility $\hat{u}_i(\sigma_i^*, \sigma_{-i^*}) \approx \tilde{u}_i(\sigma_i^*)$, and the utility of σ_i played against the BNE, $\hat{u}_i(\sigma, \sigma_{-i^*}) \approx \tilde{u}_i(\sigma_i, \sigma_{i-i^*})$, using a batch of 2^{18} samples. The *relative ex-ante utility loss* is then defined as:

$$\mathcal{L}(\sigma_i, \sigma_{-i}) = 1 - \frac{\hat{u}_i(\sigma_i, \sigma_{-i^*})}{\hat{u}_i(\sigma_i^*, \sigma_{-i^*})} \tag{4.17}$$

Each experiment includes six runs with different seeds. The reported results show the average losses across all agents and runs, as well as the standard deviation between runs. These metrics allow us to evaluate how closely the learned strategies match the theoretical equilibrium strategies and their effectiveness in maximizing agent utility. However, these losses are only applicable to games with an analytical solution.

If the mean can be directly retrieved from the Gaussian distribution, we additionally evaluate the losses using the strategy's mean action μ as a deterministic strategy instead of the mixed strategy σ. In RL applications of SAC, it is common practice to learn a mixed

strategy for exploration during training but to use the mean action as a deterministic strategy afterward. To differentiate these losses, we use a bar notation: $\bar{\mathcal{L}}$ and $\bar{L}_2$.

Further, evaluations include the visual canalization of the NF transformation by showing the distribution of the observations, the base distribution, the transformed distribution, and the final action distribution after squashing and scaling. Analyzing the exact transformation applied by the flow layer during training provides valuable insights into which parts of a distribution are challenging to learn or where the model might fail.

Additionally, evaluations include visualizing the NF transformation by showing base distribution $p_u(u)$, the distribution of transformed samples x, and the final distribution of the actions a after squashing and scaling. Analyzing the NF layers during training offers insights into challenging parts of the distribution or model failures.

5. Experiments and Results

This chapter presents the results and the impact of various adjustments to the algorithm. We analyze the outcomes of SAC for Bayesian games using Gaussian strategies, highlight specific findings for each game, and compare them to results with NF strategies.

5.1. SAC with Gaussian Strategies

The evaluation losses calculated using the mean actions ($\bar{\mathcal{L}}$ and $\bar{L}_2$) are significantly better than those using sampled actions from the Gaussian distribution ($\mathcal{L}$ and L_2). The FP auction especially benefits from using the mean. The $\bar{\mathcal{L}}$ loss is up to 90% better than the $\mathcal{L}$ loss for the FP auction and at least 50% better in the other three games. Table 5.1 demonstrates the benefit of using the mean as the deterministic strategy over sampling from the mixed strategy. All results in this section report losses calculated by the Gaussian distribution's mean action.

Table 5.1.: Improvement in using the mean actions ($\bar{\mathcal{L}}$ and $\bar{L}_2$) compared to sampling actions ($\mathcal{L}$ and L_2), demonstrating that using the mean action is a better strategy.

	L_2 Improvement (%)	$\mathcal{L}$ Improvement (%)
FP	65	90
AP	55	78
Blotto 1D	30	61
Blotto 3D	26	52

5.1.1. Basic Setup

Across all four games, the basic setup of SAC with Gaussian strategies adjusted for Bayesian games converges to an approximation of the analytical BNEs. As seen in Figure 5.1, both losses decrease continuously. There are no cycling or chaotic behaviors among the average of six runs. We can also see that all the games converge the latest after 10,000 iterations, with only relatively small improvements in the remaining iterations.

The two Blotto games resulted in the lowest utility loss $\bar{\mathcal{L}}$. The distance to the analytical BNE $\bar{L}_2$ is the shortest for Blotto 1D and the FP auction. In comparison, Blotto 3D performs significantly worse. A summary of the results can be found in Table 5.2. To illustrate specific behavior, we selected the best from six runs based on the $\bar{L}_2$ loss for each game.

Figure 5.2 displays the strategy for the FP auction at different iterations. After 500 iterations, the approximation of the analytical BNE, given by $a_i = \frac{1}{2} \cdot v_i^2$, becomes recognizable. Both

Table 5.2.: Average and standard deviation of losses over six runs for various games using the basic setup and Gaussian strategies.

	$\bar{\mathcal{L}}$	$\bar{L}_2$	sec/iter
FP	0.0206 (0.0138)	0.0388 (0.0113)	0.05
AP	0.0517 (0.0200)	0.0494 (0.0101)	0.05
Blotto 1D	0.0084 (0.0034)	0.0354 (0.0091)	0.08
Blotto 3D	0.0103 (0.0046)	0.0887 (0.0111)	0.08

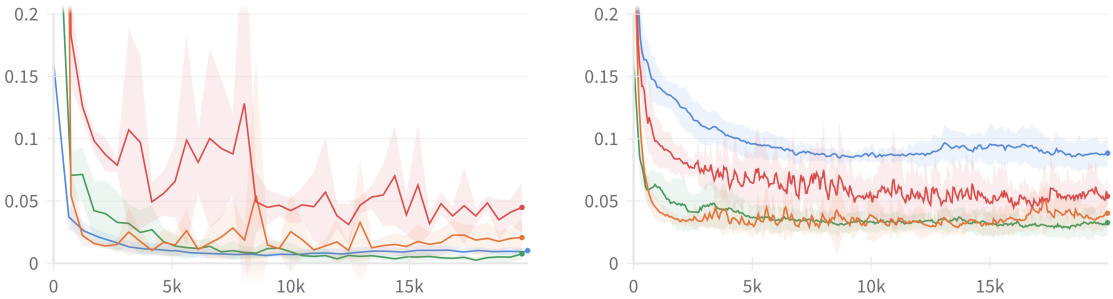


Figure 5.1.: Evaluation losses $\bar{\mathcal{L}}$ (left) and $\bar{L}_2$ (right) over time for the two auction and two Blotto games. The plots show the average of six runs with the standard deviation represented by shaded areas. Colors represent different games: FP (orange), AP (red), Blotto 1D (green), and Blotto 3D (blue).

agents occasionally transition from a denser strategy to one with a higher standard deviation. For instance, at iteration 5,000, agent 0 adopts a strategy with a particularly large standard deviation. After the final iteration, agent 1 had a larger standard deviation than agent 0.

The results for the AP auction are similar. The learned distribution approximates the BNE of $a_i = \frac{1}{2}v_i^2$ as visible in Figure 5.3 with a convergence speed similar to the FP auction. Again, the standard deviations of the Gaussian distribution vary throughout the learning process. The losses are worse for the AP auction than for the FP auction (see Table 5.2).

In Blotto 1D, we can again observe an approximation of this BNE in the chosen action distribution. Figure 5.4 shows each bidder's final three-dimensional strategy. The standard deviation of the Gaussian distribution across iterations is more consistent and lower than that of the strategies in AP and FP auctions. It takes more iteration to reach the minimum losses than for the two auction games (compare Figure 5.1).

In the Blotto 3D, the convergence to the BNE is slower than in the auction games and than that of the other Blotto game, with the loss metrics reaching a plateau after roughly 10,000 iterations (compare Figure 5.1). The low L_2 score (see Table 5.2) indicates that the final action distribution does not align with the analytical BNE correctly. This is also visible in Figure 5.5, where the actions of the final strategy are displayed. However, the utility scores are

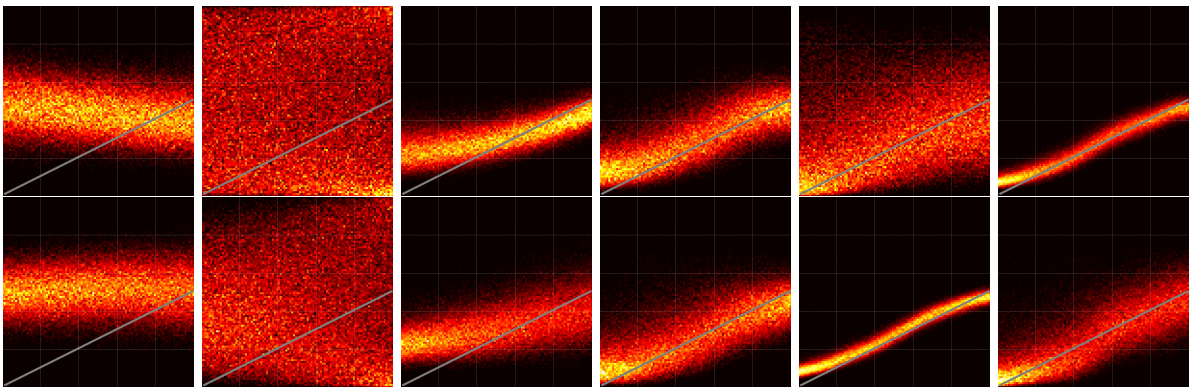


Figure 5.2.: Strategies of two agents playing the FP auction from iterations 0 to 20,000, with valuations v_i on the x-axis and actions a_i on the y-axis, indicating the ability approximate the analytical BNE depicted in grey.

significantly better than the ones from the auction games and are comparable to the other Blotto games.

5.1.2. Fixed Opponents

The results with one agent fixed to the equilibrium strategy (i.e., a stationary environment) were not different than those from the multi-agent setup, where all agents learn simultaneously (see Table A.1). The training curves, losses, and final strategies were nearly identical for both settings. Since only one agent's neural network must be trained, fixing one agent is twice as fast as with two agents.

5.1.3. Pre-training and the Entropy Parameter

To improve the results from the basic algorithm setup, we conducted experiments with different α parameter configurations and with pre-training as described in Section 4.3. The results are summarized in Table 5.3.

With $\alpha = 0$, the learned strategies have a much smaller standard deviation, as seen in Figure 5.6. It also seems to introduce vastly different results across runs with different seeds. Specifically, some instances result in players who stop learning and always bid 0 or 1. For instance, in the AP auction, one player consistently bids 0 while the other always bids 1, resulting in zero payoff for both. Similarly, in the Blotto 1D game, each agent consistently allocates their entire budget to a single item, irrespective of its valuation. Across all games, setting α to 0 did worsen both losses.

Using a constant $\alpha = 0.0025$, the evaluation losses generally improved compared to $\alpha = 0$, but they remained significantly worse than the basic setup with autotuning. Again, certain runs stagnated into states where both players' actions were ineffective. In contrast, an exponential decay for α eliminates divergence runs. This setting demonstrates the best

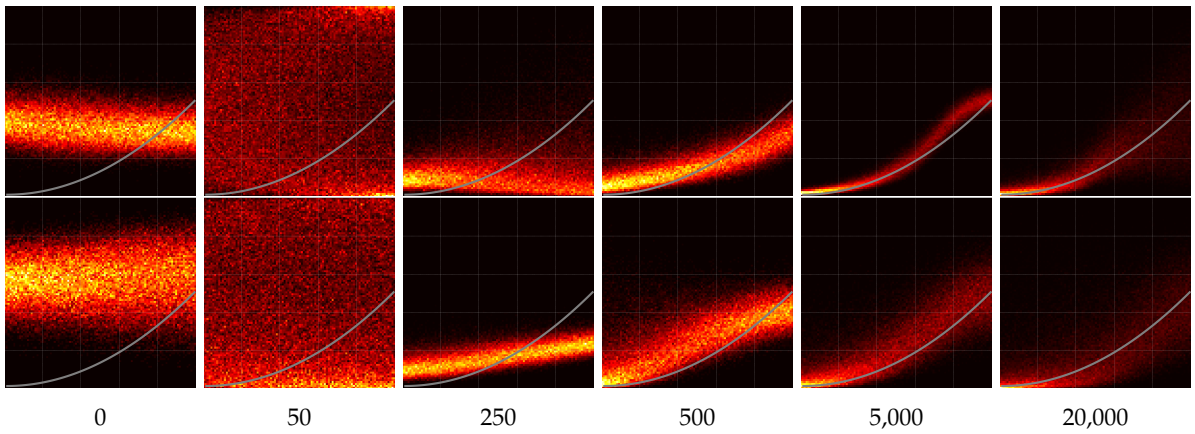


Figure 5.3.: Strategies of two agents playing the AP auction from iterations 0 to 20,000, with valuations v_i on the x-axis and actions a_i on the y-axis, indicating the ability approximate the analytical BNE depicted in grey.

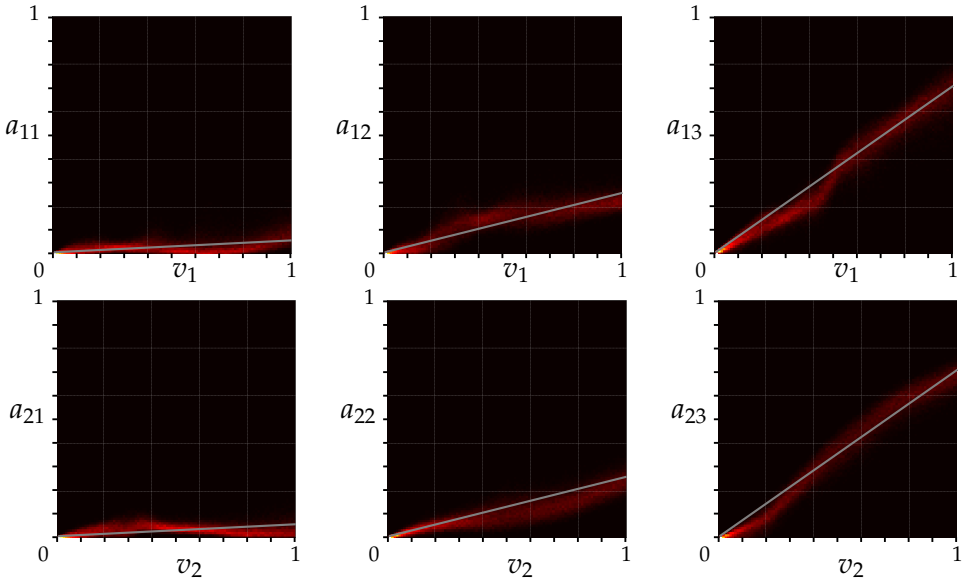


Figure 5.4.: Blotto 1D: Final distributions for both agents after 20,000 iterations with Gaussian strategies. Each image represents one dimension of a three-dimensional action. The analytical BNE $a_i = (0.05v_i, 0.25v_i, 0.7v_i)$ is depicted in grey.

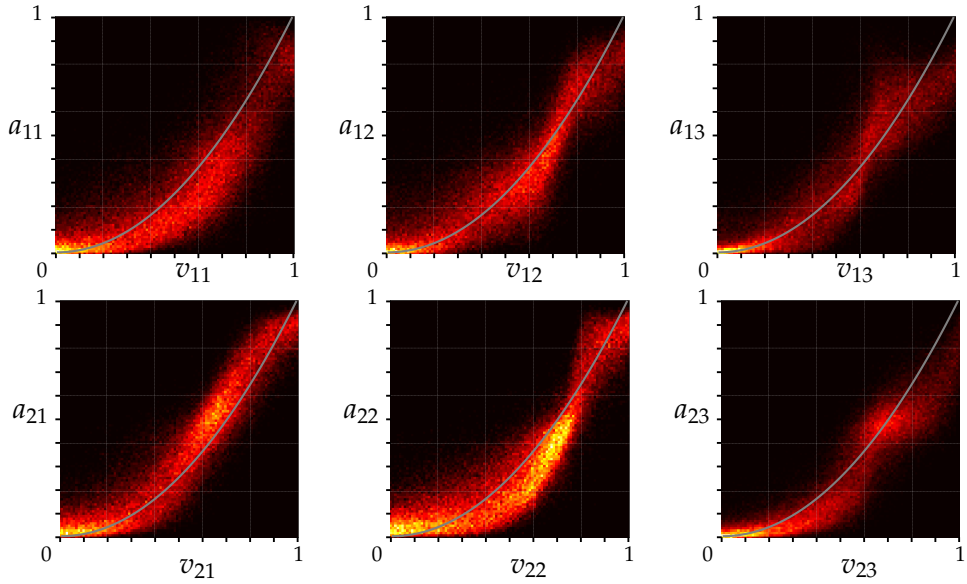


Figure 5.5.: Blotto 3D: Final distributions for both agents after 20,000 iterations with Gaussian strategies. Each image represents one dimension of a three-dimensional action over the corresponding observation dimension. The analytical BNE $a_i = \left(v_{i1}^2, v_{i2}^2, v_{i3}^2\right)$ is depicted in grey.

performance for the AP auction and Blotto 3D. Whereas, for the FP auction and Blotto 1D, the basic setup remains the most useful. The learning is slower and smoother than with autotuning (see Figure A.2).

Pre-training the strategies did not affect the results compared to the basic setup, as shown in Table 5.3. The evaluation losses, the final strategy, and the training curves did not reveal any significant difference between the two settings. Combining pre-training with $\alpha = 0$ leads to slower but more consistent learning than $\alpha = 0$ without pre-training. It also omits less but still some diverging runs. Nevertheless, the basic setup achieved better evaluation losses. This combination was only better than the basic setup when calculating the losses from the mixed strategy ($\mathcal{L}$ and L_2) and not the mean action ($\bar{\mathcal{L}}$ and $\bar{L}_2$).

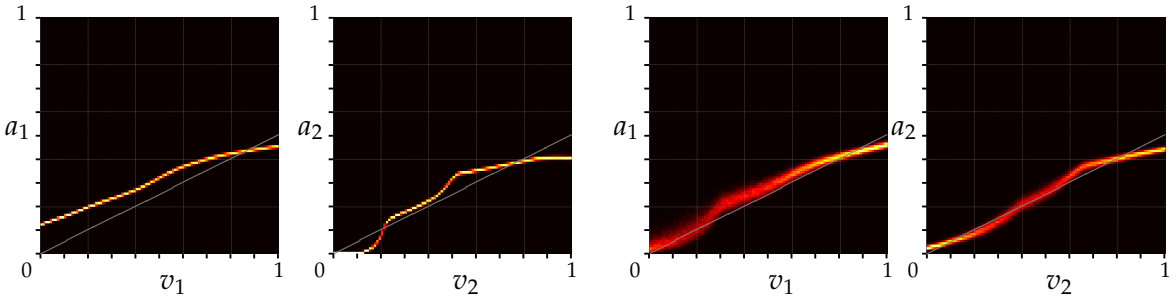


Figure 5.6.: The final strategy for an FP auction for both agents after 20,000 iterations, comparing two settings for the entropy parameter: $\alpha = 0$ (left) and decaying α (right).

Table 5.3.: Utility $\bar{\mathcal{L}}$ for various experiments with pre-training and different α values using Gaussian strategies. Bold values indicate the best results for each game.

	FP	AP	Blotto 1D	Blotto 3D
Basic Setup	**0.0206**	0.0517	0.0084	0.0103
Pre-train	0.0274	0.0430	**0.0039**	0.0350
Pre-train + $\alpha = 0$	0.0828	0.7020	0.0317	0.0500
$\alpha = 0$	0.2561	0.2077	0.0526	0.1832
$\alpha = 0.0025$	0.1543	0.2320	0.0192	0.0336
Decaying α	0.1078	**0.0392**	0.0123	**0.0086**

5.1.4. Number of Gradient Steps and Games

This section contains the results of experiments with more games to play before updating parameters and more gradient steps after playing one game. Both sets of experiments did not significantly improve the results, as shown in Table 5.4.

Playing 1,000 games simultaneously does not noticeably affect the evaluation losses and training curves compared to the basic setup. In contrast, using a replay buffer and batch size of 20,000 and playing 20,000 games simultaneously results in significantly worse performance. For the Blotto games, the basic experiment setup proves superior, while for the auction games, updating twice before adding experience to the replay buffer displays a slight advantage. The standard deviation between runs with different seeds increases when taking more than 2 gradient steps after each game. For instance, in the FP auction, employing 32 updates before playing a new game resulted in a standard deviation of 0.0802, compared to 0.0254 when updating once.

Table 5.4.: Utility $\bar{\mathcal{L}}$ for experiments with varying number gradient steps and games using Gaussian strategies. Bold values indicate the best results for each game.

	FP	AP	Blotto 1D	Blotto 3D
Basic Setup	0.0206	0.0517	**0.0084**	**0.0103**
Updates 2	**0.0136**	**0.0443**	0.0163	0.1119
Updates 8	0.0209	0.1892	0.0924	0.1662
Updates 32	0.0934	0.3766	0.1772	0.1375
Games 1,000	0.0513	0.0557	0.0091	0.0132
Games 20,000	0.0382	0.0907	0.0110	0.0192

5.1.5. Oscillating Behavior

In both auction games, oscillating behavior is evident in the $\bar{L}_2$ loss, actor losses, and in the value of the α parameter (see Figure 5.7). The Q-function losses remain stable. In the two

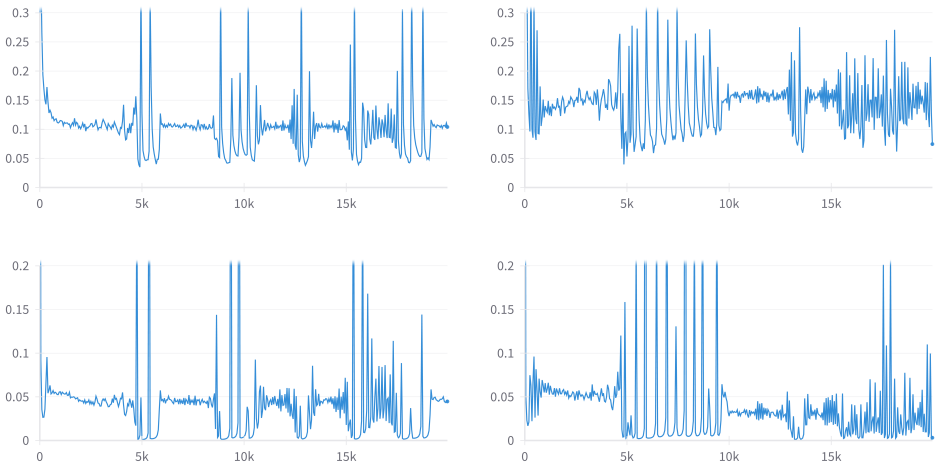


Figure 5.7.: $\bar{L}_2$ loss (top) and α value (bottom) for FP (left) and AP (right) auctions over 20,000 iterations of a single run with autotuning, showing large-scale oscillations.

Blotto games, the losses and the value of α do not exhibit the same scale of oscillation as seen in the AP and FP auctions (see Figure A.1). This reduced oscillation correlates with more stable training and better utility scores for the Blotto games.

With pre-training and adjusting the number of games or updates, the oscillating problem did not improve. This issue remained in single-agent settings when fixing all but one agent's strategy. However, in all experiments where the autotuning of α was disabled, the oscillating behavior was not visible (see Figure 6.1). This includes experiments with a decaying or constant α.

5.2. SAC with Normalizing Flow Strategies

This section presents the results of experiments employing NF strategies. NFs can model arbitrary strategies, making it less straightforward to retrieve the mean compared to Gaussian strategies, where the mean is an output of the neural network. Therefore, losses (L_2 and $\mathcal{L}$) in this section are calculated by drawing actions from the strategy distribution. For Gaussian strategies, drawing samples instead of using the mean results in significantly impaired evaluation losses (compare Table 5.1).

5.2.1. Comparing to Gaussian strategies

Across all four games, the basic experiments achieved results comparable to those of the basic experiments with a Gaussian strategy. This applies to the losses, the training curves, the final strategy, and the standard deviations between runs. Also, the experiments with the number of games played sequentially, and the number of updates before playing a game had very similar outcomes as those with a Gaussian strategy (see Table A.2). The most notable

difference between Gaussian and NF strategies is the time needed per iteration. See Table 5.5 to compare the results between Gaussian and NF strategies with the basic setup.

Table 5.5.: Comparison of losses $\mathcal{L}$ and L_2 and iteration time for Gaussian and NF strategies.

		$\mathcal{L}$	L_2	sec/iter
FP	Gaussian	0.1748	0.1254	0.05
	NFs	0.1998	0.1274	0.15
AP	Gaussian	0.1840	0.1155	0.05
	NFs	0.1747	0.1405	0.15
Blotto 1D	Gaussian	0.0208	0.0487	0.08
	NFs	0.0203	0.0483	0.33
Blotto 3D	Gaussian	0.0212	0.1201	0.08
	NFs	0.0201	0.1151	0.30

5.2.2. Pre-training and Entropy Parameter with Normalizing Flow Strategies

The experiments involving pre-training and setting the entropy parameter α to zero had a comparable outcome to the same set of experiments with the Gaussian strategy. The primary difference is that α significantly affects the calculated losses L_2 and $\mathcal{L}$, whereas the losses calculated with the mean action ($\bar{L}_2$ and $\bar{\mathcal{L}}$) are not significantly influenced.

As with Gaussian strategies, pre-training had no effect. Experiments with $\alpha = 0$ and $\alpha = 0.0025$ diverge in some runs, resulting in non-optimal average evaluation losses. However, using exponential decay for α greatly improved the results for all but the Blotto 1D game. While a small gain is observable in mean action losses $\bar{L}_2$ and $\bar{\mathcal{L}}$ when using a decaying α and Gaussian strategies (compare Table 5.3), the losses L_2 and $\mathcal{L}$ improve substantially (see Table 5.6 and Figure A.3).

Table 5.6.: Utility $\mathcal{L}$ for pre-training and α experiments with NF strategies. Bold values indicate the best results for each game. An exponential decay in α proved beneficial.

	FP	AP	Blotto 1D	Blotto 3D
Basic Setup	0.1998	0.1747	**0.0203**	0.0201
Pre-train	0.1166	0.1796	0.0223	0.0179
Pre-train + $\alpha = 0$	0.6296	0.5264	0.0439	0.0963
$\alpha = 0$	0.0659	0.9064	0.2182	0.2792
$\alpha = 0.0025$	0.2884	0.1927	0.0375	0.0513
Decaying α	**0.0120**	**0.0508**	0.0213	**0.0175**

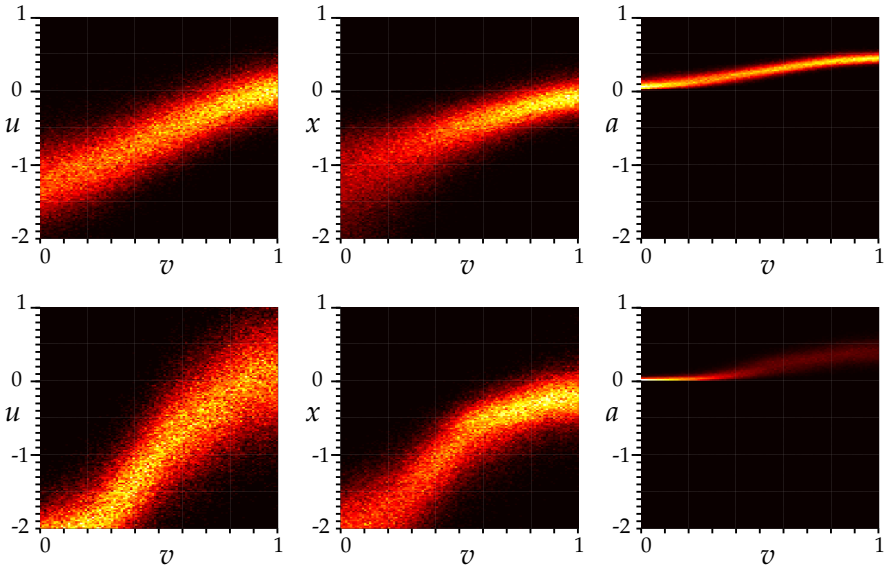


Figure 5.8.: Transformation of NF strategies in FP (top) and AP (bottom) auctions, showing the base distribution (left), NF transformation (middle), and final scaled distribution (right).

5.2.3. Transformation Components

This section compares the base distribution, the transformed distribution, and the final distribution, where the actions are squashed and rescaled. Figure 5.8 illustrates these distributions for the two auction games with one-dimensional actions mapped over the observations. Both the base distribution and the flow influence the final distribution.

For the two Blotto games, we sampled 2^{17} observations and plotted the resulting three-dimensional actions from the intermediate and final steps in Figure 5.9. The effects of the base distribution and the NF layers are visible, though the transformation effect is relatively small.

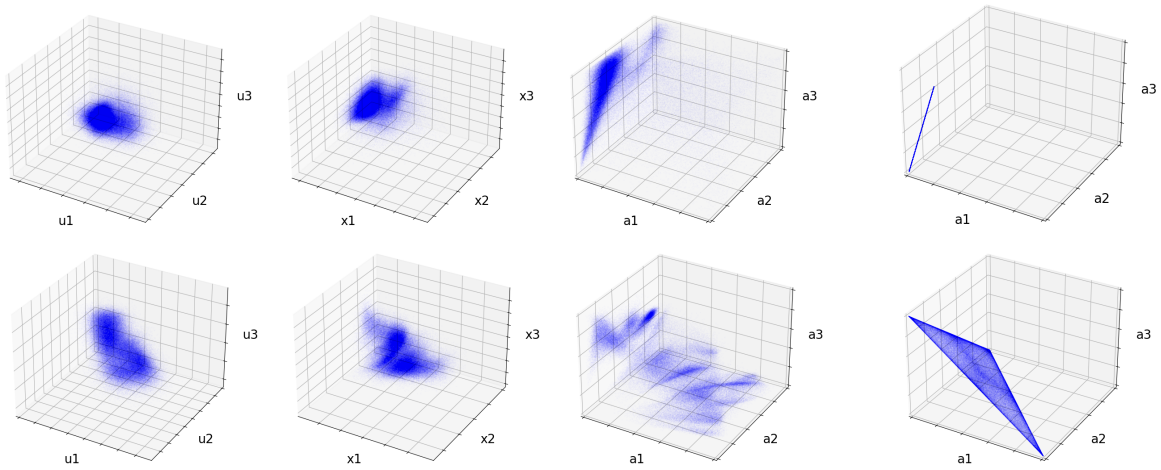


Figure 5.9.: The base distribution, transformation, and marginal action distribution after scaling and squashing for Blotto 1D (top) and Blotto 3D (bottom). The BNEs are shown on the right. The axes represent the three dimensions of one action in a Blotto game.

6. Discussion

6.1. Learning Pure Strategies in Bayesian Games

In general, the results in this thesis prove that SAC adapted for Bayesian games can learn approximations of pure BNEs in the described auction and Blotto games. The algorithm does not discretize the action or the type space like comparable algorithms, which can hinder solution quality (Kroer et al. 2015). Taking the mean of the learned strategy is best when applying it in games with pure BNEs. This approach aligns with how SAC is typically applied in single-agent RL.

SAC with NFs also allows for incorporating game constraints, like squashing, scaling, and normalizing a multi-dimensional action. This proved especially beneficial when the constraint depended on the observation, as is the case for Blotto 1D (see Appendix A.2.1).

Table 6.1 compares the utility score and time for the FP auction between NPGA and SAC for Bayesian games. Compared to NPGA, which learns only pure strategies, SAC shows inferior utility and L_2 losses. However, our algorithm is faster per iteration and does need a similar number of iterations to converge. We conducted another set of experiments with a similar runtime between NPGA and SAC for Bayesian games (see Appendix A.2.2), which improved the results for SAC, but NPGA still had the advantage. We believe an improved method for the entropy parameter could increase the performance of SAC for Bayesian games.

Table 6.1.: Comparison of NPGA (Bichler et al. 2021) and SAC for Bayesian games: $\bar{\mathcal{L}}$ score and iterations per second. Both algorithms were trained for 20,000 iterations. NPGA averaged over 10 runs and SAC over 6 runs, with standard deviation in brackets.

	$\bar{\mathcal{L}}$	sec/iter
NPGA	0.0001 (0.0009)	0.31
SAC	0.0136 (0.0034)	0.05

Another advantage of SAC is that it is an off-policy algorithm. This is beneficial in practice when insufficient experience is available based on the current strategy. A bidder could update its strategy after each auction outcome. Whereas in NPGA and the algorithm from Martin et al. (2022) need to collect experiences of multiple perturbations of the current strategies before updating. In SODA (Bichler et al. 2023a), information about all agents' strategies is needed to perform an update step.

6.2. Learning Mixed Strategies

The experiments with NFs showed that incorporating NFs in SAC does not hinder the algorithm from learning pure BNEs. The evaluation losses for all games were similar to those of SAC with Gaussian strategies. Looking at the specific transformation conducted by the NF layer, we see that NFs have an effect and are trained together with the base distribution. Since the NF layer does more than replicate the identity function, we suspect that SAC with NFs can learn to represent an arbitrary mixed strategy.

Introducing NFs into the algorithm results in longer training times due to the additional neural network. For instance, in our case, training with just one Neural Spline NF layer took approximately three times longer than training a single neural network for the Gaussian distribution. However, there is potential to optimize the training time by fine-tuning the number of layers and parameters of each neural network.

6.3. Stable Performance Across Experiments

SAC demonstrates robust performance. Hyperparameter tuning revealed that learning rates do not majorly affect performance and produced similar learning rates across the four games. Increasing the number of updates only had a noticeable adverse impact when set to eight or more. Playing more games with the current strategy, resulting in more on-policy data in the replay buffer, did not significantly impact performance. The best results were consistently achieved using the basic SAC setup. Only changing the method for determining α had a considerable impact.

These results align with the findings of Haarnoja et al. (2018b), who state that SAC is robust to hyperparameters, with the entropy parameter being the only hyperparameter that requires tuning.

6.4. Determining an Optimal Entropy Parameter

The basic setup employed autotuning of the entropy parameter with a target entropy of $-\dim(A_i)$ introduced by Haarnoja et al. (2018c). We suspect this is the reason for the oscillating behavior we observed through all experiments except when having a constant or exponential decaying α. Autotuning for α was developed for single-agent settings, which could be a reason why it did not work well in the experiments. However, even when fixing all agents but one to the equilibrium strategy, this oscillatory behavior persisted, implying that it does not purely stem from the non-stationary environment.

The results show that setting α to zero is not a valid alternative to autotuning. With $\alpha = 0$, the objective for our strategy becomes the traditional RL objective, maximizing only the expected reward and not the entropy of the strategy. This causes a loss of exploration, which is crucial for any RL algorithm. With $\alpha = 0.0025$, the results are slightly better but similar.

A decaying α improves results across games and avoids diverging runs. The FP and AP auctions most notably benefited from switching from autotuning to an exponential decaying

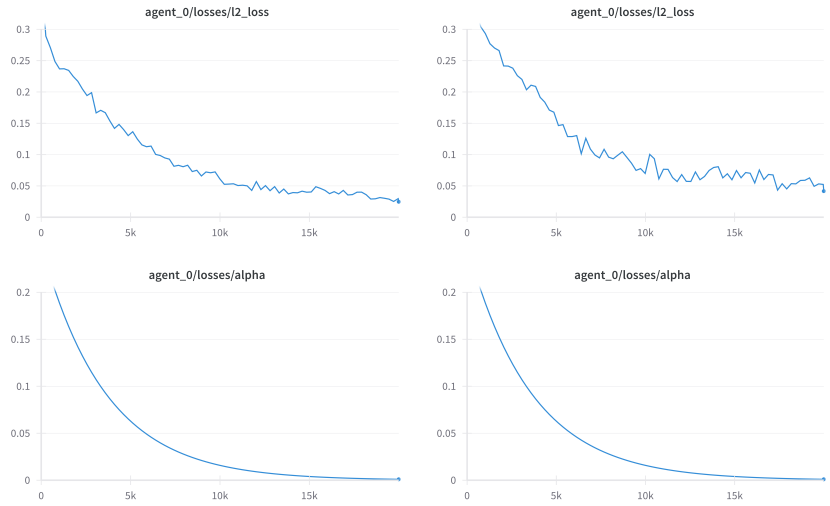


Figure 6.1.: The L_2 score and the α value of FP and AP auction with an exponential decaying α, displaying no oscillation and gradually decreasing L_2.

α. In these auctions, the oscillating behavior is especially prominent when using autotuning. Nevertheless, if the mean action is available, the losses $\bar{\mathcal{L}}$ and $\bar{L}_2$ were still slightly better than the losses using samples from the learned distribution ($\mathcal{L}$ and L_2).

We conclude that maintaining a relatively large α and using the mean as the final strategy may be advantageous when pure actions are desired. Conversely, when the goal is to learn a mixed strategy, a denser distribution achieved by using a small value for α is better. It is crucial to determine an optimal α value, which should be as small as possible while promoting sufficient exploration.

Exponential decaying of α showed promising results, but we expect that optimizing this process further holds much potential. Haarnoja et al. (2018c) discuss extensively the meaning of α and their autotuning solution. One goal of autotuning α is to allow for higher entropy in regions where the optimal action is uncertain and to be more deterministic in areas where the best action is clear. We believe this approach is also beneficial for Bayesian games, even though their exact implementation for autotuning α did not work.

6.5. Limitations

The current work serves as a proof of concept for demonstrating the viability of SAC for Bayesian games. SAC with NFs for Bayesian games has not been extensively tested for practical games lacking analytical or pure BNEs. The method's scalability remains unproven, with potential issues in training time due to the complexity added by NF layers. Methods like SODA have a considerable advantage regarding scalability and runtime. Additionally, the current losses are worse than those of comparable algorithms like NPGA. Finding an optimal strategy for entropy parameter α could improve the results. However, it is also possible that

the tradeoff between exploration and exploiting a low entropy strategy may be challenging to resolve in practice, and no optimal technique for choosing the entropy parameter exists.

6.6. Future Work

Future work remains to examine SAC for Bayesian games across various games and settings and evaluate its scalability with additional agents and items. This includes games that do not have pure BNEs. Additionally, SAC with NFs can be applied to non-Bayesian games to learn mixed strategies. Without any observation, the algorithm has a fixed base distribution and trains the NF layers to convert the base distribution into an optimal strategy.

Future work includes investigating the potential benefits of using NFs. For example, their properties of reversibility and monotonicity could be beneficial since strategies in auctions are typically assumed to be monotone. NFs are designed to perform efficiently with very high-dimensional data, such as images. This capability could present another advantage worth exploring. Trying out different NF implementations, besides Neural Splines, can also prove beneficial. NF layers that are less expressive or do not have an analytical inverse but are more efficient could increase the speed of SAC with NFs. In addition, NFs can be used to explicitly incorporate constraints as it has been done in single-agent settings (e.g., Brahmanage et al. (2024), Rietz et al. (2024), and Chen et al. (2024)).

NFs can also model pure strategies in Bayesian games. By treating the observation distribution $v \sim F$ as the base distribution, the observation is transformed deterministically into a sample of the target distribution. This method allows for learning reversible pure strategies, enabling the determination of which observation was needed to play a specific action. See an example of a transformation in Figure A.4. SAC is unsuitable for this approach as it needs the entropy of a stochastic strategy to explore sufficiently. However, replacing neural networks with NF layers in methods like NPGA could be promising for achieving a reversible strategy.

A key next step for SAC in Bayesian games is to establish a process to find an optimal α value. Possible approaches include investigating and adjusting the autotuning mechanism of the original SAC algorithm (e.g., setting a different target entropy), conducting hyperparameter tuning to identify a fixed value for each game, or examining other decay equations. Additionally, pretraining the strategy was beneficial in some of our runs when $\alpha = 0$, suggesting it could be helpful when experimenting with smaller α values. We believe exploring these options can significantly improve the results of SAC for Bayesian games with and without NFs.

7. Conclusion

This thesis introduces a novel method to learn BNEs. The method is based on a popular RL algorithm, SAC, designed to find optimal policies in MDPs. We apply SAC to Bayesian games in an independent multi-agent environment. Additionally, we use NFs, a concept that transforms a simple base distribution into a complex target distribution through a series of invertible transformations. We extend SAC for Bayesian games with NF strategies, allowing for the representation of arbitrary, mixed strategies whose densities can be evaluated. We validated both methods across two auction games and two Blotto games with pure BNEs and experimented with various algorithm settings.

The experiments demonstrate that both approaches effectively learn an approximation of the pure BNEs in the four Bayesian games. The algorithm is robust to most hyperparameters and converges within minutes. We found that the entropy parameter significantly impacts the results and that autotuning proves ineffective in our settings. Exponential decay emerged as the most effective function for the entropy parameter. Our method achieves results that are slightly inferior to NPGA, an algorithm to find pure BNEs, but SAC for Bayesian games achieves faster iteration times. We believe that optimizing the selection process for the entropy parameter holds significant potential for improving the algorithm's accuracy.

SAC for Bayesian games overcomes the limitations of alternative algorithms. First, it avoids the need to discretize the action or type space. This eliminates scaling issues stemming from the exponential growth of discrete points and the challenge of selecting appropriate discretization intervals. Second, it can learn pure and mixed strategies. Some Bayesian games lack a pure BNE, limiting the use of algorithms for merely pure strategies. Third, the density of the learned strategy can be evaluated analytically. This is crucial for analyzing concrete games, e.g., when designing auctions or exploring alternative actions as a player.

While this thesis acts as a proof of concept, future work must investigate the practicality of the SAC for Bayesian games algorithm. The method should be evaluated on diverse games, including games without pure BNEs and without analytical BNEs. Additionally, exploring and validating use cases that benefit from NFs' properties, such as monotonicity and reversibility, could reveal new applications for SAC with NFs for Bayesian games.

To summarize, we established a versatile algorithm that can find BNEs and is potentially useful in various applications. SAC for Bayesian games can help understand complex learning dynamics in Bayesian games and benefit bidders and auctioneers in designing optimal auctions and strategies. This makes it a valuable tool for game theorists, bidders, auctioneers, and other strategists involved in solving Bayesian games.

A. Appendix

A.1. Additional Tables and Figures

Table A.1.: Average $\bar{\mathcal{L}}$ loss for both agents in the basic setup and the single learning agent in the fixed opponent experiment. No significant difference is observed.

	FP	AP	Blotto 1D	Blotto 3D
Basic Setup (Multi-Agent)	**0.0206**	**0.0517**	0.0084	0.0103
Stationary Environment	0.0211	0.0561	**0.0069**	**0.0078**

Table A.2.: Utility $\mathcal{L}$ for experiments with varying number gradient steps and games using NFs strategies. Bold values indicate the best results for each game.

	FP	AP	Blotto 1D	Blotto 3D
Basic Setup	0.1998	**0.1747**	0.0203	0.0201
Updates 2	0.1891	0.2429	**0.0188**	0.0244
Updates 8	**0.1587**	0.2714	0.0822	0.0447
Updates 32	0.1960	0.6701	0.1546	0.0827
Games 1,000	0.2188	0.2145	0.0258	**0.0196**
Games 20,000	0.2991	0.5910	0.0307	0.0390

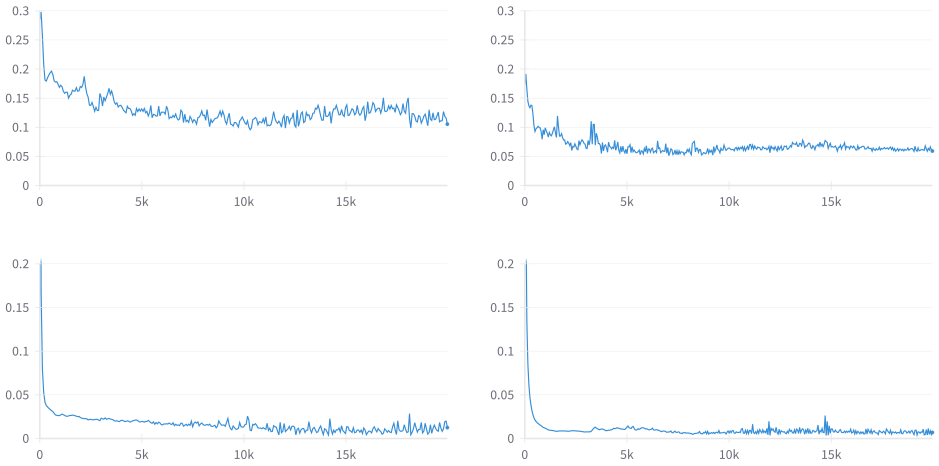
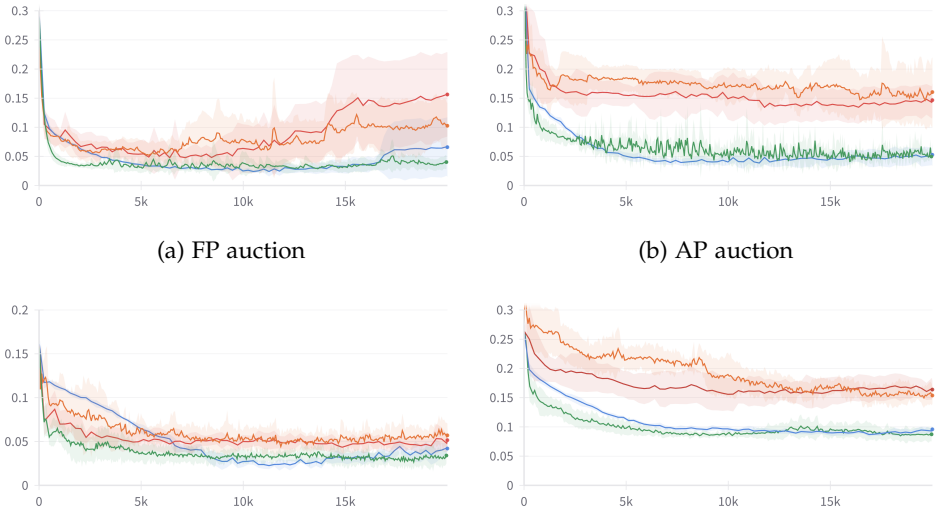


Figure A.1.: $\bar{L}_2$ loss (top) and α value (bottom) for Blotto 1D (left) and Blotto 3D (right) auctions over 20,000 iterations of a single run with autotuning, without large scale oscillation.



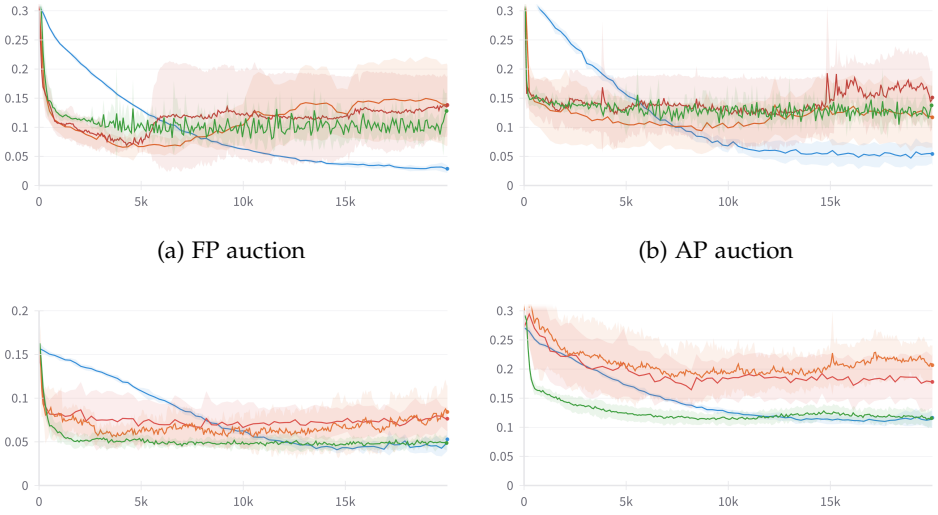
(a) FP auction

(b) AP auction

(c) Blotto 1D

(d) Blotto 3D

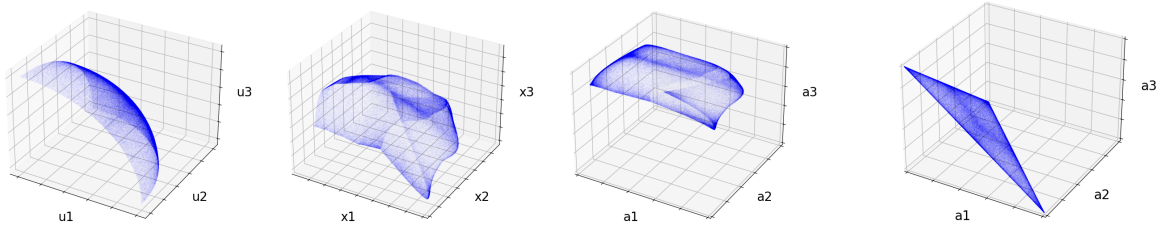
Figure A.2.: $\bar{\mathcal{L}}$ loss for the experiments and Gaussian strategies and different α settings: $\alpha = 0$ (orange), $\alpha = 0.0025$ (red), decaying α (blue), autotuning (green). Decaying α learns more slowly and smoothly but offers no advantage when using the strategy mean.



(a) FP auction (b) AP auction

(c) Blotto 1D (d) Blotto 3D

Figure A.3.: $\mathcal{L}$ loss for the experiments and Gaussian strategies and different α settings: $\alpha = 0$ (orange), $\alpha = 0.0025$ (red), decaying α (blue), autotuning (green). Decaying α learns more slowly and smoothly and improves the losses significantly for FP and AP auctions.



(a) Observations (b) Transformed (c) Scaled (d) BNE

Figure A.4.: An example of an NFs strategy in a Blotto 3D game without an explicit base distribution. The observation distribution serves as the base and is transformed to replicate the BNE, resulting in a pure strategy. This approach is unsuitable for SAC with Bayesian games due to the lack of entropy.

A.2. Additional Experiments

This section lists two additional experiments not mentioned in the main part of this thesis.

A.2.1. Modeling Constraints

Many games have strict constraints on their actions. SAC, by default, squashes actions with tanh and scales them to an interval. However, Blotto 1D has a conditional constraint on the action. The observation, v_i, determines the budget of player i. The multi-dimensional action must add up to the budget and cannot be larger than the budget.

We originally implemented this constraint directly inside the game environment. This means the environment accepted actions that did not follow this constraint and normalized the actions accordingly before determining the winners. This did not produce good results.

The alternative is to model the action constraint directly inside the model, which we have deployed throughout the experiments of this thesis. This method was superior. Table A.3 displays the utility scores of both methods.

Table A.3.: Comparison of two methods to model constraints, showing significant improvement if modeling the constraint inside the model instead of inside the environment.

	$\bar{\mathcal{L}}$	$\bar{L}_2$
Constraints in Model	0.0084	0.0354
Constraints in Env	0.1175	0.0934

A.2.2. Increasing Runtime

As compared in Chapter 6, NPGA has superior evaluation losses but takes more time per iteration than SAC for Bayesian games. In this comparison, both algorithms were run for 20,000 iterations, with NPGA being six times slower. To compare the method's effectiveness on a more leveled playing field, we let SAC for Bayesian games run for roughly the same times as NPGA (i.e., six times as many iterations). We used a decaying α starting at 0.25 and being at 0.005 after 120,000 iterations. We used Gaussian strategies and used the mean action loss $\bar{\mathcal{L}}$. The results can be seen in Table A.4 and the final strategy for an FP auction is depicted in Figure A.5.

Table A.4.: Comparison of NPGA (Bichler et al. 2021) and SAC for Bayesian games: $\bar{\mathcal{L}}$ score and iterations per second. NPGA was trained for 20,000 iterations, while SAC was trained for 120,000 iterations, resulting in similar runtimes.

	$\bar{\mathcal{L}}$	runtime (min)
NPGA	0.0001 (0.0009)	103
SAC	0.0057 (0.0027)	63

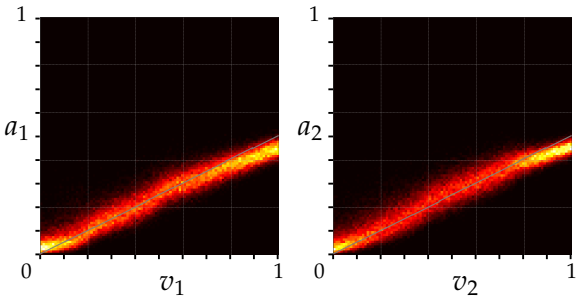


Figure A.5.: The final strategy for an FP auction for both agents after 120,000 iterations with a decaying α.

List of Figures

List of Tables

Bibliography

Adamo, Tim and Alexander Matros (2009). "A Blotto game with incomplete information". In: *Economics Letters* 105.1, pp. 100–102.

Andrade, Gabriel P, Rafael Frongillo, and Georgios Piliouras (2021). "Learning in matrix games can be arbitrarily complex". In: *Conference on Learning Theory*. PMLR, pp. 159–185.

Athey, Susan (2001). "Single crossing properties and the existence of pure strategy equilibria in games of incomplete information". In: *Econometrica* 69.4, pp. 861–889.

Baxter, Jonathan and Peter L Bartlett (2001). "Infinite-horizon policy-gradient estimation". In: *journal of artificial intelligence research* 15, pp. 319–350.

Benaım, Michel and Morris W Hirsch (1999). "Mixed equilibria and dynamical systems arising from fictitious play in perturbed games". In: *Games and Economic Behavior* 29.1-2, pp. 36–72.

Berger, Ulrich (2005). "Fictitious play in $2\times$ n games". In: *Journal of Economic Theory* 120.2, pp. 139–154.

Bertsekas, Dimitri and John N Tsitsiklis (1996). *Neuro-dynamic programming*. Athena Scientific.

Bhatnagar, Shalabh, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee (2009). "Natural actor–critic algorithms". In: *Automatica* 45.11, pp. 2471–2482.

Bichler, Martin, Max Fichtl, and Matthias Oberlechner (2023a). "Computing Bayes–Nash Equilibrium Strategies in Auction Games via Simultaneous Online Dual Averaging". In: *Operations Research*.

Bichler, Martin, Maximilian Fichtl, Stefan Heidekrüger, Nils Kohring, and Paul Sutterer (2021). "Learning equilibria in symmetric auction games using artificial neural networks". In: *Nature machine intelligence* 3.8, pp. 687–695.

Bichler, Martin, Nils Kohring, and Stefan Heidekrüger (2023b). "Learning equilibria in asymmetric auction games". In: *INFORMS Journal on Computing* 35.3, pp. 523–542.

Bichler, Martin, Stephan B Lunowa, Matthias Oberlechner, Fabian R Pieroth, and Barbara Wohlmuth (2023c). "On the Convergence of Learning Algorithms in Bayesian Auction Games". In: *arXiv preprint arXiv:2311.15398*.

Boborzi, Damian, Christoph-Nikolas Straehle, Jens S Buchner, and Lars Mikelsons (2021). "Learning normalizing flow policies based on highway demonstrations". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, pp. 22–29.

Bosshard, Vitor, Benedikt Bünz, Benjamin Lubin, and Sven Seuken (2020). "Computing Bayes-Nash equilibria in combinatorial auctions with verification". In: *Journal of Artificial Intelligence Research* 69, pp. 531–570.

Brahmanage, Janaka, Jiajing Ling, and Akshat Kumar (2024). "FlowPG: Action-constrained Policy Gradient with Normalizing Flows". In: *Advances in Neural Information Processing Systems* 36.

Brown, George W (1951). "Iterative solution of games by fictitious play". In: *Act. Anal. Prod Allocation* 13.1, p. 374.

Brown, Noam and Tuomas Sandholm (2019). "Superhuman AI for multiplayer poker". In: *Science* 365.6456, pp. 885–890.

Cai, Yang and Christos Papadimitriou (2014). "Simultaneous bayesian auctions and computational complexity". In: *Proceedings of the fifteenth ACM conference on Economics and computation*, pp. 895–910.

Carbonell-Nicolau, Oriol and Richard P McLean (2018). "On the existence of Nash equilibrium in Bayesian games". In: *Mathematics of Operations Research* 43.1, pp. 100–129.

Ceppi, Sofia, Nicola Gatti, and Nicola Basilico (2009). "Computing Bayes-Nash equilibria through support enumeration methods in Bayesian two-player strategic-form games". In: *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 2. IEEE, pp. 541–548.

Chen, Changyu, Ramesha Karunasena, Thanh Nguyen, Arunesh Sinha, and Pradeep Varakantham (2024). "Generative Modelling of Stochastic Actions with Arbitrary Constraints in Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 36.

Choi, Hana, Carl F Mela, Santiago R Balseiro, and Adam Leary (2020). "Online display advertising markets: A literature review and future directions". In: *Information Systems Research* 31.2, pp. 556–575.

Conitzer, Vincent and Tuomas Sandholm (2008). "New complexity results about Nash equilibria". In: *Games and Economic Behavior* 63.2, pp. 621–641.

Daskalakis, Constantinos, Paul W Goldberg, and Christos H Papadimitriou (2009). "The complexity of computing a Nash equilibrium". In: *Communications of the ACM* 52.2, pp. 89–97.

Delalleau, Olivier, Maxim Peter, Eloi Alonso, and Adrien Logut (2019). "Discrete and continuous action representation for practical rl in video games". In: *arXiv preprint arXiv:1912.11077*.

Dinh, Laurent, David Krueger, and Yoshua Bengio (2014). "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803*.

Durkan, Conor, Artur Bekasov, Iain Murray, and George Papamakarios (2019). "Neural spline flows". In: *Advances in neural information processing systems* 32.

Foerster, Jakob, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson (2018). "Counterfactual multi-agent policy gradients". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1.

Foster, Dean P and Rakesh V Vohra (1997). "Calibrated learning and correlated equilibrium". In: *Games and Economic Behavior* 21.1-2, pp. 40–55.

Foster, Dean P and H Peyton Young (2001). "On the impossibility of predicting the behavior of rational agents". In: *Proceedings of the National Academy of Sciences* 98.22, pp. 12848–12853.

Fudenberg, Drew and David K Levine (2009). "Learning and equilibrium". In: *Annu. Rev. Econ.* 1.1, pp. 385–420.

Fujimoto, Scott, Herke Hoof, and David Meger (2018). "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR, pp. 1587–1596.

Gairing, Martin (2008). "Malicious Bayesian congestion games". In: *International Workshop on Approximation and Online Algorithms*. Springer, pp. 119–132.

Guerra, Alex and Joseph Marino (2020). "Sequential Autoregressive Flow-Based Policies". In: *ICML workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*.

Gupta, Jayesh K, Maxim Egorov, and Mykel Kochenderfer (2017). "Cooperative multi-agent control using deep reinforcement learning". In: *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, pp. 66–83.

Haarnoja, Tuomas, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine (2018a). "Latent space policies for hierarchical reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 1851–1860.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018b). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR, pp. 1861–1870.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. (2018c). "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905*.

Harsanyi, John C (1967). "Games with incomplete information played by "Bayesian" players, I–III Part I. The basic model". In: *Management science* 14.3, pp. 159–182.

Hart, Sergiu and Andreu Mas-Colell (2000). "A simple adaptive procedure leading to correlated equilibrium". In: *Econometrica* 68.5, pp. 1127–1150.

– (2003). "Uncoupled dynamics do not lead to Nash equilibrium". In: *American Economic Review* 93.5, pp. 1830–1836.

Heinrich, Johannes and David Silver (2016). "Deep reinforcement learning from self-play in imperfect-information games". In: *arXiv preprint arXiv:1603.01121*.

Hellman, Ziv (2014). "A game with no Bayesian approximate equilibria". In: *Journal of Economic Theory* 153, pp. 138–151.

Holt, Charles A and Alvin E Roth (2004). "The Nash equilibrium: A perspective". In: *Proceedings of the National Academy of Sciences* 101.12, pp. 3999–4002.

Jackson, Matthew O and Jeroen M Swinkels (2005). "Existence of equilibrium in single and double private value auctions 1". In: *Econometrica* 73.1, pp. 93–139.

Jafari, Amir, Amy Greenwald, David Gondek, and Gunes Ercal (2001). "On no-regret learning, fictitious play, and nash equilibrium". In: *ICML*. Vol. 1, pp. 226–233.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Kingma, Durk P and Prafulla Dhariwal (2018). "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in neural information processing systems* 31.

Kobyzev, Ivan, Simon JD Prince, and Marcus A Brubaker (2020). "Normalizing flows: An introduction and review of current methods". In: *IEEE transactions on pattern analysis and machine intelligence* 43.11, pp. 3964–3979.

Konda, Vijay and John Tsitsiklis (1999). "Actor-critic algorithms". In: *Advances in neural information processing systems* 12.

Kovenock, Dan and Brian Roberson (2011). "A Blotto game with multi-dimensional incomplete information". In: *Economics Letters* 113.3, pp. 273–275.

Kroer, Christian and Tuomas Sandholm (2015). "Discretization of continuous action spaces in extensive-form games". In: *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pp. 47–56.

Lemke, Carlton E and Joseph T Howson Jr (1964). "Equilibrium points of bimatrix games". In: *Journal of the Society for industrial and Applied Mathematics* 12.2, pp. 413–423.

Li, Zun and Michael P Wellman (2021). "Evolution strategies for approximate solution of Bayesian games". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 6, pp. 5531–5540.

Lowe, Ryan, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch (2017). "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in neural information processing systems* 30.

Luo, Yudong, Guiliang Liu, Haonan Duan, Oliver Schulte, and Pascal Poupart (2021). "Distributional reinforcement learning with monotonic splines". In: *International Conference on Learning Representations*.

Ma, Xiaobai, Jayesh K Gupta, and Mykel J Kochenderfer (2020). "Normalizing flow model for policy representation in continuous action multi-agent systems". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1916–1918.

Martin, Carlos and Tuomas Sandholm (2022). "Finding mixed-strategy equilibria of continuous-action games without gradients using randomized policy networks". In: *arXiv preprint arXiv:2211.15936*.

Mazoure, Bogdan, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm (2020). "Leveraging exploration in off-policy algorithms via normalizing flows". In: *Conference on Robot Learning*. PMLR, pp. 430–444.

Mazumdar, Eric, Lillian J Ratliff, and S Shankar Sastry (2020). "On gradient-based learning in continuous games". In: *SIAM Journal on Mathematics of Data Science* 2.1, pp. 103–131.

Mertikopoulos, Panayotis, Christos Papadimitriou, and Georgios Piliouras (2018a). "Cycles in adversarial regularized learning". In: *Proceedings of the twenty-ninth annual ACM-SIAM symposium on discrete algorithms*. SIAM, pp. 2703–2717.

Mertikopoulos, Panayotis and Mathias Staudigl (2018b). "On the convergence of gradient-like flows with noisy gradient input". In: *SIAM Journal on Optimization* 28.1, pp. 163–197.

Mertikopoulos, Panayotis and Zhengyuan Zhou (2019). "Learning in games with continuous action sets and unknown payoff functions". In: *Mathematical Programming* 173, pp. 465–507.

Milgrom, Paul R and Robert J Weber (1985). "Distributional strategies for games with incomplete information". In: *Mathematics of operations research* 10.4, pp. 619–632.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Monderer, Dov and Lloyd S Shapley (1996). "Potential games". In: *Games and economic behavior* 14.1, pp. 124–143.

Naroditskiy, Victor and Amy Greenwald (2007). "Using iterated best-response to find Bayes-Nash equilibria in auctions". In: *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. Vol. 22. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, p. 1894.

Nash Jr, John F (1950). "Equilibrium points in n-person games". In: *Proceedings of the national academy of sciences* 36.1, pp. 48–49.

Noussair, Charles and Jonathon Silver (2006). "Behavior in all-pay auctions with incomplete information". In: *Games and Economic Behavior* 55.1, pp. 189–206.

Pan, Xuehai, Mickel Liu, Fangwei Zhong, Yaodong Yang, Song-Chun Zhu, and Yizhou Wang (2022). "Mate: Benchmarking multi-agent reinforcement learning in distributed target coverage control". In: *Advances in Neural Information Processing Systems* 35, pp. 27862–27879.

Papamakarios, George, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan (2021). "Normalizing flows for probabilistic modeling and inference". In: *Journal of Machine Learning Research* 22.57, pp. 1–64.

Rabinovich, Zinovi, Victor Naroditskiy, Enrico H Gerding, and Nicholas R Jennings (2013). "Computing pure Bayesian-Nash equilibria in games with finite actions and continuous types". In: *Artificial Intelligence* 195, pp. 106–139.

Reeves, Daniel M. and Michael P. Wellman (2004). "Computing Best-Response Strategies in Infinite Games of Incomplete Information". In: *Conference on Uncertainty in Artificial Intelligence*. URL: https://api.semanticscholar.org/CorpusID:6656229.

Rezende, Danilo and Shakir Mohamed (2015). "Variational inference with normalizing flows". In: *International conference on machine learning*. PMLR, pp. 1530–1538.

Rietz, Finn, Erik Schaffernicht, Stefan Heinrich, and Johannes A Stork (2024). "Towards Interpretable Reinforcement Learning with Constrained Normalizing Flow Policies". In: *arXiv preprint arXiv:2405.01198*.

Robinson, Julia (1951). "An iterative method of solving a game". In: *Annals of mathematics*, pp. 296–301.

Sanders, James BT, J Doyne Farmer, and Tobias Galla (2018). "The prevalence of chaotic dynamics in games with many players". In: *Scientific reports* 8.1, p. 4902.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). "Trust region policy optimization". In: *International conference on machine learning*. PMLR, pp. 1889–1897.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Shalev-Shwartz, Shai, Shaked Shammah, and Amnon Shashua (2016). "Safe, multi-agent, reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1610.03295*.

Shamma, Jeff S and Gürdal Arslan (2005). "Dynamic fictitious play, dynamic gradient play, and distributed convergence to Nash equilibria". In: *IEEE Transactions on Automatic Control* 50.3, pp. 312–327.

Shapley, Lloyd S et al. (1963). *Some topics in two-person games*. Rand Corporation.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). "Deterministic policy gradient algorithms". In: *International conference on machine learning*. Pmlr, pp. 387–395.

Singh, Satinder, Michael J Kearns, and Yishay Mansour (2000). "Nash Convergence of Gradient Dynamics in General-Sum Games." In: *UAI*, pp. 541–548.

Stimper, Vincent, David Liu, Andrew Campbell, Vincent Berenz, Lukas Ryll, Bernhard Schölkopf, and José Miguel Hernández-Lobato (2023). "normflows: A pytorch package for normalizing flows". In: *arXiv preprint arXiv:2302.12014*.

Sutton, Richard S and Andrew G Barto (1998). "The reinforcement learning problem". In: *Reinforcement learning: An introduction*, pp. 51–85.

Sutton, Richard S, David McAllester, Satinder Singh, and Yishay Mansour (1999). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12.

Tan, Ming (1993). "Multi-agent reinforcement learning: Independent vs. cooperative agents". In: *Proceedings of the tenth international conference on machine learning*, pp. 330–337.

Tang, Yunhao and Shipra Agrawal (2018). "Boosting trust region policy optimization by normalizing flows policy". In: *arXiv preprint arXiv:1809.10326*.

Thomas, Philip (2014). "Bias in natural actor-critic algorithms". In: *International conference on machine learning*. PMLR, pp. 441–448.

Vickrey, William (1961). "Counterspeculation, auctions, and competitive sealed tenders". In: *The Journal of finance* 16.1, pp. 8–37.

Vinyals, Oriol, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. (2019). "Alphastar: Mastering the real-time strategy game starcraft ii". In: *DeepMind blog* 2, p. 20.

Vlatakis-Gkaragkounis, Emmanouil-Vasileios, Lampros Flokas, Thanasis Lianeas, Panayotis Mertikopoulos, and Georgios Piliouras (2020). "No-regret learning and mixed nash equilibria: They do not mix". In: *Advances in Neural Information Processing Systems* 33, pp. 1380–1391.

Vorobeychik, Yevgeniy, Michael P Wellman, et al. (2008). "Stochastic search methods for nash equilibrium approximation in simulation-based games." In: *AAMAS (2)*, pp. 1055–1062.

Ward, Patrick Nadeem, Ariella Smofsky, and Avishek Joey Bose (2019). "Improving exploration in soft-actor-critic with normalizing flows policies". In: *arXiv preprint arXiv:1906.02771*.

Wierstra, Daan, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber (2014). "Natural evolution strategies". In: *The Journal of Machine Learning Research* 15.1, pp. 949–980.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8, pp. 229–256.

Zhang, Kaiqing, Zhuoran Yang, and Tamer Başar (2021). "Multi-agent reinforcement learning: A selective overview of theories and algorithms". In: *Handbook of reinforcement learning and control*, pp. 321–384.