Particle Value Functions in Imperfect Information Games

Michal Šustr
Czech Technical University
Prague, Czechia
michal.sustr@aic.fel.cvut.cz

Vojtěch Kovařík
Czech Technical University
Prague, Czechia
vojta.kovarik@gmail.com

Viliam Lisý
Czech Technical University
Prague, Czechia
viliam.lisy@agents.fel.cvut.cz

ABSTRACT

In perfect-information games, approximating optimal strategies can be decomposed to solving depth-limited look-ahead subgames wherein the subgame leaves are evaluated using a value function. A similar approach is possible in imperfect-information games, but the value function must simultaneously evaluate whole sets of related states with players' belief distributions over these sets. Since existing methods represent these beliefs explicitly, their memory requirements can grow exponentially. To tackle this challenge, we introduce a sparse particle-based representation of value functions. Our first contribution is showing that this representation has strictly stronger representation power than the traditional approach. Second, we propose a domain-independent neural network architecture for approximating particle-based value functions. Finally, we empirically show the learnability and graceful degradation of this approach when only a subset of beliefs is used. Particle-based value functions dramatically expand the class of games where learning methods are applicable.

KEYWORDS

Game Theory, Nash Equilibrium, Value Function, Particle Algorithm

1 INTRODUCTION

Limited look-ahead search with a heuristic value function is a key AI technique in reinforcement learning and game playing. In perfect-information problems, this has commonly been the leading method in programs outperforming expert human players, e.g., in Chess [10] or Go [30]. Recent results in large *imperfect* information games (IIGs) indicate that depth-limited solving is also very effective in strong Poker AI [3, 20].

The value functions necessary for sound limited look-ahead game solving in IIGs need to evaluate whole sets of possible game histories, called public states, at once [28]. While in Poker, the size of a public state is relatively small and constant, in many other games, the number of possible game histories in a public state grows exponentially in the number of moves in the game. For example in Kriegspiel, a variant of Chess where players see only their pieces and only learn if their attempted move is illegal, the number of game histories that any of the players considers possible, i.e. the public state, initially grows almost as quickly as the number of all histories in the game. If the first capture does not occur within the first ten moves, and the players played only legal moves, the reached public state contains over 20^{10} different histories.

In order to implement a computationally efficient depth-limited search algorithm for arbitrary large games, we need to ensure that the set of histories in each public state, along with the probabilities of these histories occurring under a strategy, is **represented in a space of constant size**.

One way to achieve that is to represent the large public state by a sample of representative histories, called particles [35], of a bounded size. Intuitively, if there are only a few meaningful actions the opponent might have done, we can try to identify them and keep track of only the relevant histories during the game. If there are many such actions of the opponent, then a larger constant-size sample might also sufficiently capture the information necessary to choose the right move.

This paper proposes a value function that allows approximating counterfactual values based on only a sample of histories in a public state. We thus solve a fundamental challenge of extending existing algorithms that use full beliefs to use only the sparse representations. We are not aware of any fundamental difficulties in replacing the value function used in [3, 20] by the proposed one, but we leave the full integration for future work.

We present a neural network architecture inspired by Deep Sets [36] and multiple-instance learning [22] that allows mapping an arbitrarily large sample of histories and their reach probabilities to corresponding counterfactual values for any game represented as factored observation stochastic game (FOSG) [15].

We show both theoretically and empirically that this architecture can learn value functions of the same precision as existing architectures if it is provided with the complete set of all histories. We then demonstrate that reducing the proportion of histories on the input causes the outputs' precision to degrade gracefully. Finally, we show that using this value function with a well-selected subset of histories allows for depth-limited solving of a sparsified look-ahead tree.

2 RELATED WORK

Belief states, in the form of probability distributions over possible world states, are a key concept in single-agent Partially Observable Markov Decision Processes (POMDPs). Typically, the belief states are represented explicitly as a probability distribution over all possible states [32]. Since the number of possible states is often prohibitively large, some methods (e.g., POMCP [31]) represent the state a sample from the set of possible states, rather than using the explicit representation. However, these methods rely on pseudo-random rollouts to estimate the value of belief states and do not explicitly use value functions over the sets of samples. While there are methods that try to use rollouts directly without a value function even in partially observable games [17, 34], these methods converge too slowly to be practical to solve even medium-size problems.

An alternative approach to deal with prohibitively large state spaces is (automatically built) abstraction (e.g., [26] via belief compression). While there has been very active research in abstractions

in partially observable games (e.g., [7, 11]), abstraction-based strategies proved to be hugely exploitable in Poker [16], and the combination of maintaining the exact information about the current belief state with value function leads to super-human performance [3, 20]. Therefore, this paper focuses on combining value functions with sparse belief state representations.

This is a viable approach because even though the number of possible game states grows exponentially in many games, the set of states that are reachable by optimal play at a particular moment (i.e., the support of the belief state) tends to be small. Previous results show that for random normal-form games, the size of the support of the optimal play is logarithmic [18]. Double-Oracle algorithms in both normal-form [19] and extensive-form [2] games converge to small equilibria in a wide variety of real-world games. The number of actions required for optimal play, even in games with huge action spaces, has been proven to be linear in the amount of hidden information [27].

All existing value functions evaluate full belief states [3, 4, 20, 28]. We investigate whether it is possible to address this limitation and use a sparse belief state representation with particle value function evaluations. We answer this question in the positive both theoretically and empirically.

3 NOTATION AND BACKGROUND

We present our results using the formalism of factored-observation stochastic games [15] which is a variant of partially observable stochastic games [9] that distinguishes between private and public observations. This distinction is important, as sound algorithms that use decomposition in imperfect information games critically rely on the notion of common/public information [5, 28].

A two-player zero-sum factored-observation stochastic game is a tuple $\mathcal{G} = \langle \mathcal{N}, \mathcal{W}, w^0, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O} \rangle$, where:

- $\mathcal{N} = \{1, 2\}$ is a **player set**. We use symbol n for a player and $\text{-}n$ for its opponent.
- $\mathcal{W}$ is a set of **world states** and $w^0 \in \mathcal{W}$ is a designated initial world state.
- $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is a space of **joint actions**. The subsets $\mathcal{A}_n(w) \subset \mathcal{A}_n$ and $\mathcal{A}(w) = \mathcal{A}_1(w) \times \mathcal{A}_2(w) \subset \mathcal{A}$ specify the (joint) actions legal at $w \in \mathcal{W}$. For $a \in \mathcal{A}$, we write $a = (a_1, a_2)$. $\mathcal{A}_n(w)$ for $n \in \mathcal{N}$ are either all non-empty or all empty. A world state with no legal actions is **terminal**.
- After taking a (legal) joint action a at w, the **transition function** $\mathcal{T}$ determines the next world state w', drawn from the probability distribution $\mathcal{T}(w, a) \in \Delta(\mathcal{W})$.
- $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$ is the reward function, we consider the zero-sum setting with $\mathcal{R}_n(w, a) = -\mathcal{R}_{\text{-}n}(w, a)$ being the reward for taking the joint action a at w.
- $\mathcal{O} = (\mathcal{O}_{\text{priv}(1)}, \mathcal{O}_{\text{priv}(2)}, \mathcal{O}_{\text{pub}})$ is the **observation function**, where the functions $\mathcal{O}_{(\cdot)} : \mathcal{W} \times \mathcal{A} \times \mathcal{W} \to \mathbb{O}_{(\cdot)}$ specify the **private observation** that player n receives, resp. the **public observation** that every player receives, upon transitioning from world state w to w' via some joint action a.

A legal **world history** (or trajectory) is a finite sequence $h = (w^0, a^0, w^1, a^1, \ldots, w^t)$, where $w^k \in \mathcal{W}, a^k \in \mathcal{A}(w^k)$, and $w^{k+1} \in \mathcal{W}$ is in the support of $\mathcal{T}(w^k, a^k)$. We denote the set of all legal histories by $\mathcal{H}$.

Since the last world state in each $h \in \mathcal{H}$ is uniquely defined, the notation for $\mathcal{W}$ can be overloaded to work with $\mathcal{H}$ (e.g., $\mathcal{A}(h) := \mathcal{A}(\text{the last } w \text{ in } h)$, h being terminal, ...). We use $\mathcal{Z}$ to denote the set of all terminal histories, i.e. histories where the last world state is terminal.

The **cumulative reward** of n at h is $\sum_{k=0}^{t-1} r_n^k := \sum_{k=0}^{t-1} \mathcal{R}_n(w^k, a^k)$. When h is a terminal history (denoted as z), cumulative rewards are called **utilities**, and denoted as $u_n(z)$. We assume games are **zero-sum**, so $u_n(z) = -u_{\text{-}n}(z)\ \forall z \in \mathcal{Z}$.

The **public history** (or public state) corresponding to $h = (w^0, a^0, w^1, a^1, \ldots, w^t)$ is the observation sequence $s_{\text{pub}}(h) := (O^0_{\text{pub}}, O^1_{\text{pub}}, \ldots, O^t_{\text{pub}})$ of all public observations corresponding to h, where $O^k_{\text{pub}} = \mathcal{O}_{\text{pub}}(w^{k-1}, a^{k-1}, w^k)$ (and O^0_{pub} is some initial observation). The space $\mathcal{S}_{\text{pub}}$ of all such sequences can be viewed as the **public tree**.

A player's **private information state** corresponding to h is the sequence of the actions taken by the player and their private observations: $s_{\text{priv(n)}}(h) := (O^0_{\text{priv(n)}}, a^0_n, O^1_{\text{priv(n)}}, a^1_n, \ldots, O^t_{\text{priv(n)}})$, where $O^k_{\text{priv(n)}} = \mathcal{O}_{\text{priv(n)}}(w^{k-1}, a^{k-1}, w^k)$ (and $O^0_{\text{priv(n)}}$ are some initial observations). The set of all possible private infostates of player n is denoted $\mathcal{S}_{\text{priv(n)}}$. Those that can arise in a public s_{pub} are denoted $\mathcal{S}_{\text{priv(n)}}(s_{\text{pub}})$. Taken together, the public state and private information state form the player's **information state** (**infostate**) $s_n = (s_{\text{pub}}, s_{\text{priv(n)}})$. $\mathcal{S}_n$ and $\mathcal{S}_n(s_{\text{pub}})$ are defined analogously to $\mathcal{S}_{\text{priv(n)}}$ and $\mathcal{S}_{\text{priv(n)}}(s_{\text{pub}})$.

Example 1. *In the imperfect-information variant of card game Goofspiel, each player is given a deck of* bid cards $1, \ldots, N$ *and there is a third deck of so-called point cards* $1, \ldots, N$*. Each round, one of the point cards is drawn face up, the players bid one of their remaining bid cards (both at the same time), and the one with the higher bid wins the point card (nobody wins it in case of a tie). However, players do not learn the size of their opponent's bid — instead, a referee publicly announces who won the round (or whether the round was a tie). We assume that the point cards are "auctioned" in decreasing order.*

Formally, public observations are either "win", "tie", or "loss" (taken from player 1's point of view). As a result, the public tree in N-card Goofspiel is a depth-N ternary tree consisting of sequences of win/tie/loss outcomes. Each round, a player privately observes which cards they have left[1] *(a subset of* $\{1, \ldots, N\}$*) and submits an action "bid k" for one of the remaining cards.*

For example, suppose that $N = 3$*. If player 1 (PL1) bids 3 and PL2 bids 1, the resulting public observation is* $O^1_{pub} =$ win*, as PL1 won the bidding with a higher card. Players then receive private observations* $(1, 2)$ *and* $(2, 3)$*, resulting in private infostates* $(\emptyset,$ bid 3, $(1, 2))$*, resp.* $(\emptyset,$ bid 1, $(2, 3))$*.*

However, the public state $s_{\text{pub}} =$ win *can also arise through other play, if players bid 3 vs 2 or 2 vs 1. In this case, the sets of infostates* $\mathcal{S}_{\text{priv}(1)}(s_{\text{pub}})$*, resp.* $\mathcal{S}_{\text{priv}(2)}(s_{\text{pub}})$*, compatible with the public state* $s_{\text{pub}} =$ (win) *correspond to actions* {bid 3, bid 2} *for PL1, resp.* {bid 2, bid 1} *for PL2. As the game progresses, the size of* $\mathcal{S}_{\text{priv(n)}}(s_{\text{pub}})$ *increases* exponentially.

[1] The private observations are redundant since they can be inferred from past bid actions. We include them for illustration.

Lemma 2. *For Imperfect Information Goofspiel with N cards, in any depth k, there is a public state with at least*

$$X(k) = \frac{N!(N-1)!}{(N-k)!(N-1-k)!2^k}$$

histories and at least $2\sqrt{X(k)}$ *distinct information states.*

In the standard setup with $N = 13$ cards, some public states include over 5×10^7 private information states. See Appendix A for the proof and further details.

A **strategy profile** is a pair $\pi = (\pi_1, \pi_2)$, where each (behavioral) **strategy** $\pi_n : s_n \in \mathcal{S}_n \mapsto \pi_n(s_n) \in \Delta(\mathcal{A}_n(s_n))$ specifies the probability distribution from which player n draws their next action (conditional on having information s_n). We denote the set of all strategies of player n as Π_n and the set of all strategy profiles as Π.

The **reach probability** of a history $h \in \mathcal{H}$ under π is defined as $P^\pi(h) = P_1^\pi(h)\, P_2^\pi(h)\, P_c^\pi(h)$, where each $P_n^\pi(h)$ is a product of probabilities of the actions taken by player n between the root and h, and $P_c^\pi(h)$ is the product of stochastic transitions. The **expected utility** for player n of a strategy profile π is $u_n(\pi) = \sum_{z \in \mathcal{Z}} P^\pi(z) u_n(z)$. A profile $\pi^\star$ is a **Nash equilibrium** (or just **equilibrium**) if no player can achieve a higher expected utility by switching to a different strategy. Formally, $(\forall\, n \in \mathcal{N}) : u_n(\pi^\star) \geq \max_{\pi_n \in \Pi_n} u_n(\pi_n, \pi^\star_{-n})$. For zero-sum games, equilibria can be found using the following equation, typically solved by linear programming: [14, 21]

$$\max_{\pi_1 \in \Pi_1} \min_{\pi_2 \in \Pi_1} u_1(\pi_1, \pi_2) := u^\star. \tag{1}$$

Public belief states (PBS) [3, 20] generalize the notion of scalar "state value" to imperfect-information games. It is a generalization similar to belief states in POMDPs [1], i.e a conditional distribution of the current state given the history of observations and actions. However, in imperfect-information games, we need to condition on beliefs of both player for all infostates. Decomposition of the game into public states allows us to limit the number of infostates for which we need to maintain beliefs: the public observations guarantee that infostates are closed under a relation of indistinguishability [34]. However, we cannot generally reduce the amount of considered infostates further, as doing so could prevent the values from being properly defined [28]. Therefore, a **public belief state** (PBS) in a factored-observation stochastic game is triplet $\beta = (s_{\text{pub}}, b_1, b_2)$, where $b_n \in \Delta\mathcal{S}_{\text{priv(n)}}(s_{\text{pub}})$ are non-normalized **player beliefs**. For a specific policy π, the belief b_n is equivalent to the $\{P_n^\pi(s_n) \mid s_n \in \mathcal{S}_n(s_{\text{pub}})\}$ where $P_n^\pi(s_n)$ is a **player belief** for the particular infostate s_n. We call the tuple $b = (b_1, b_2)$ simply **beliefs**.

A **subgame** $\mathcal{G}(\beta)$ rooted in the PBS $\beta = (s_{\text{pub}}, b)$ is defined like the original factored-observation stochastic game $\mathcal{G}$, except that the initial world state emulates the distribution and infostates corresponding to b [15]. Just as in perfect-information subgames, the value of $\mathcal{G}(\beta)$ does not depend on anything (strategies, observations, etc.) that came before the root PBS. Subgames can be thus solved independently from other parts of the whole game [3]. This allows us to define the (non-normalized) **expected utility of a PBS** as[2]

$$u_n(\beta) = \sum_{h \in \mathcal{H}(s_{\text{pub}}(\beta))} P^\pi(h) u_n^{\pi^\star}(h), \tag{2}$$

where $\pi^\star$ is an equilibrium policy in the subgame rooted at β. Moreover, this value can be calculated recursively, in a way that can be viewed as an extension of the standard minimax algorithm from perfect information games that uses an oracle value function for the leaf evaluation:

Lemma 3. *The max-min optimization* (1) *can be recursively decomposed using the formula*

$$u_n(s_{\text{pub}}, b) = \max_{\pi_1|_{\mathcal{S}_1(s_{\text{pub}})}} \min_{\pi_2|_{\mathcal{S}_2(s_{\text{pub}})}} \sum_{s'_{\text{pub}} \in \text{succ}(s_{\text{pub}})} u_n(s'_{\text{pub}}, \tilde{b}),$$

where $\tilde{b}$ *denotes the belief obtained via a Bayesian update of* b *using the policy* $\pi|_{\mathcal{S}(s_{\text{pub}})}$ *and* $\text{succ}(s_{\text{pub}})$ *are successor public states of* s_{pub}.

The recursive decomposition of Lemma 3 has been used for iterative approximation of equilibria using regret minimization [5, 20]. As the games we typically solve are large (with largest variants of Poker having 10^{160} states) online algorithms have been used [3, 20]. They rely on function approximation to compute the counterfactual value constraints for each player for each infostate in a given PBS.

Finally, the term $u_n(\beta)$ can be viewed as a value function that assigns a single **value** to each public state (for a fixed policy). However, in practice, most algorithms use value functions where we have one value for each infostate [20, 28]. Fortunately, [3, Theorem 1] reconciles these seemingly incompatible approaches by proving that the infostate-based values correspond precisely to the (sub)gradients of the public-state-based values. This suggests that infostate values can be approximately computed when one only has access to the public-state based value function. In the following text, we can, therefore, use these two types of values mostly interchangeably.

In particular, when training approximate value functions, we can assume that we use a **training set** of the form $TS \subset \{(\beta^i, v^i) \mid \beta^i \in B, v^i \in \mathbb{R}^{S_1(s_{\text{pub}}(x^i)) \cup S_2(s_{\text{pub}}(x^i))}\}$ where B is the **set of public belief states** β in the game and targets v^i are counterfactual values for each infostate s_n for each player n within the public state $s_{\text{pub}}(\beta^i)$. The training set is either incrementally updated through self-play [3] or statically generated based on random sampling of beliefs [20, 28].

4 VALUE FUNCTIONS

While all the value functions appearing in the literature so far used positional encoding of infostates, it is also possible to use "non-positional" value functions, which avoid this. In this section, we contrast positional and non-positional value functions. To construct the latter type of a value function, we explain how to encode information about the game using features.

[2] Strictly speaking, the right-hand side of (2) should only use b, rather than π. However, $P_n^\pi(s_n)$ uniquely determines $P_n^\pi(h)$, so the right-hand side of (2) actually only depends on b.

4.1 Encoding using Features

In the previous section, we gave a "mathematical" description of value functions — it assumed that infostates were elements of an abstract set. To find a description that would be more suitable for encoding via a neural network, we use the concept of **features**. These will allow for more compact indexing and better generalization. Formally, we assume that each observation consist of a set of **elementary observations**, where each elementary observation is a pair $o = (f, x)$ where f is a name of some *feature* and x is the current value of that feature. While the value x is usually something game-specific, we typically encode it using a real number or a vector. Additionally, we will assume that the same is true of actions.

Armed with the concept of features, we can encode public states and infostates as vectors. Suppose we want to describe the situation at the start of the D-th round of the game. We can denote by $F^d = F^d_{\text{priv(1)}} \cup F^d_{\text{priv(2)}} \cup F^d_{\text{pub}}$ the set of elementary features that can plausibly be observed on the transition between the $(d-1)$-th and d-th round (i.e., they appear as part of some public or private observation received at the start of the d-th round, or as part of an action taken in $(d-1)$-th round). We call these the **imperfect recall features** (for round d). By putting the features from different rounds together, we obtain the **perfect recall features** (for round D): $F^{\leq D} := \dot{\bigcup}_{d \leq D} F^d$ (where the $\dot{\bigcup}$ denotes disjoint union). Using this notation, every public state in depth D can be encoded as an element of $\Pi_{f \in F^{\leq D}_{\text{pub}}} X_f$ and every private infostate can be encoded as an element of $\Pi_{f \in F^{\leq D}_{\text{priv(n)}}} X_f$. In the simplest (but common) case where all X_f are equal to $\mathbb{R}$, we thus encode public states and private infostates as real-valued vectors of length $|F^{\leq D}_{\text{pub}}|$, resp. $|F^{\leq D}_{\text{priv(n)}}|$. (To deal with features that are not defined in a particular state, we can pad the missing values by zeroes or some other unused value [28].)

Note that in perfect information games like Chess, while technically it is possible to use perfect recall public features, they are redundant as in these games the value function does not depend on the history of play. Instead, **imperfect recall public features** are used to represent the current world state (i.e. the chessboard). When there are known symmetries in the game with respect to the value of the state (such as rotational board symmetry in Go), these features can be simplified even more to save redundant computation for learning.

In a similar spirit, such simplifications can be done also in imperfect information games. For example, note that in some games (like Poker) the private observation features (i.e. hand cards) do not change throughout the game. This makes indexing based on perfect recall unnecessarily large since for each additional public action in the game we extend the private feature sequence with the same observation. In this case, we can use domain knowledge of the problem and construct **imperfect recall private features** for the hand cards.

Finally, while this encoding can be used in *any* domain, doing so might decrease the expressive power of the value function, making it impossible to find a good approximation. For example, in Goofspiel, using imperfect recall features would mean that the value function would not "remember" the order in which the player used their cards. On the other hand, training such a value function will typically be computationally less expensive, so the trade-off can be worth making.

4.2 Positional Value Functions

As we saw in the previous section, when computing value functions in imperfect information games, we need the public state and the belief b over all infostates $\mathcal{S}_{\text{priv(1)}}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv(2)}}(s_{\text{pub}})$ as input. And as an output, we use one value for each of these infostates. Typically, these value functions are learned using some neural network with parameters θ, using the training set described in the previous section. For games where the number of private infostates is small or constant (such as Poker), we can have a designated input and output neuron for each infostate $s \in \mathcal{S}_{\text{priv(n)}}(s_{\text{pub}})$ — we call such value networks **positional**. The easiest way to design such a network would be to use $|\mathcal{S}_{\text{priv(1)}}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv(2)}}(s_{\text{pub}})| + 1$ input neurons and $|\mathcal{S}_{\text{priv(1)}}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv(2)}}(s_{\text{pub}})|$ output neurons. Indeed, to encode the training set, we would fix an enumeration of public states and private infostates and encode each datapoint (s_{pub}, b, v) as (the enumeration of s_{pub}, reach probability of 1st infostate, reach probability of 2nd infostate, ...) on input and (value of 1st infostate, value of 2nd infostate, ...) on the output. In practice, it is advantageous to encode the public state not as a single number but rather using its features as described in Section 4.1. All existing value functions in the literature are either exactly of this form or very similar — we call them *positional* value functions [3, 4, 20, 28].

We formalize this by saying that v is a **positional value function** if it depends on a public state and a *full* belief (i.e., a vector $b \in \mathbb{R}^{\mathcal{S}_{\text{priv(1)}}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv(2)}}(s_{\text{pub}})}$) as input and output a *full* vector of infostate values (i.e., $v(b) \in \mathbb{R}^{\mathcal{S}_{\text{priv(1)}}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv(2)}}(s_{\text{pub}})}$). In contrast, the next section will discuss **non-positional value functions** which accept a public state s_{pub} and a *sparse* belief (i.e., a vector $b' \in R^{S'}$ defined on some *subset* S' of $\mathcal{S}_{\text{priv(1)}}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv(2)}}(s_{\text{pub}})$) as input and output a *sparse* vector of infostate values (i.e., $v(b') \in R^{S'}$).

We saw that positional value functions are straightforward to construct. Unfortunately, even when we use domain-specific simplifications, the number of infostates can grow exponentially in games like GoofSpiel or DarkChess. (Indeed, in N-card GoofSpiel, the encoding contains all possible card subsets $\binom{N}{k}$ after k bids. In DarkChess, players take a combinatorially large number of secret moves.) Since this translates into exponentially large inputs and outputs to the function approximator, using positional value functions is simply not feasible in these games. This motivates the use of *particle value functions*.

5 PARTICLE VALUE FUNCTIONS

In this section we introduce **particle value functions**, a novel neural network model for non-positional value functions. Motivated by the empirical evidence that equilibrium policies have small supports in many games (see Related Work), we design a model that can take an arbitrary subset of player beliefs for some public belief state and run regression to target infostate values, individually for each infostate. We show that if the model is sufficiently large, there exists a parametrization that represents a function identical to the positional value function. In the experiments, we show that the model can learn parameters that have similar or better performance than the corresponding positional value function.

5.1 Particle-Based Algorithm

We designed the particle value function with the intention of use in particle-based algorithms, similarly to how function approximation is used in POMDPs [35]. The algorithm is used both for incremental generation of the training set as well as an equilibrium approximation within public states reached during online play.

The algorithm keeps track of the current public state s_{pub} and maintains a subset $H \in \mathcal{H}(s_{\text{pub}})$ of representative histories – called **particles** – in s_{pub}. For each player, the algorithm maintains the reach probabilities $P^{\pi}_{n}(h)$ for all particles $h \in H$. When an infostate s_n is compatible with some of the particles $H(s_n) := \{h \mid s_n(h) = s_n \; \forall h \in H\}$, we assume that its reach probability is equal to the sum over the individual reach probabilities $P^{\pi}_{n}(s_n) = \sum_{h \in H(s_n)} P^{\pi}_{n}(h)$. And, as an approximation, we assume that if no such particle exists, the reach probability is equal to zero. This yields (approximate) beliefs b corresponding to π at s_{pub}. As a result, the particles H induce a well-defined PBS $\beta = (s_{\text{pub}}, b)$.

We then proceed similarly as in [3] to learn the value function via self-play. At each move, we approximate equilibrium cf. values in the current β. Instead of storing full beliefs for the successor states $\text{succ}(s_{\text{pub}}(\beta))$, we sample a new set of representative particles for those states. In online play, the actions of the opponent are not observed and we can reach an infostate or a public state not covered by the particles. To ensure we maintain a constant-sized representative sample, we can use particle regeneration [31], which can be done similarly as in [25, 29].

While in this paper we train the value functions based on a training set TS, we do no use them in the particle algorithm. We leave this for future work, as we are interested in analysis whether it is possible to use a trained particle value function practically, as this is a prerequisite for the full particle algorithm.

5.2 Particle Value Function Architecture

The particle value function uses sparse beliefs b' (a subset of full beliefs) as inputs for approximation of the target cf. values. It is a *set function*, which is a relatively non-standard function to learn using machine learning.

To enable the use of arbitrary subset of beliefs, we use addition of (transformed) inputs as proposed in [22, 36]. Even though the addition operation is very simple, architectures based on it have been proven to be universal approximators for any set functions [36]. Empirically it performs surprisingly well in a number of applied problems [6, 8, 24]. The addition causes the model to be permutationally invariant with respect to the order of the inputs and we do not need to use the positional encoding as existing methods. This saves the costly generation of the set of all possible private features, as we will use only private features for the currently evaluated infostates. Also, it makes the particle value function smaller compared to the positional one, as we do not need to encode exponentially large inputs. The inputs are rather expressed in terms of game-specific features for the infostates and player belief for those infostates. Finally, to avoid doing any positional encoding for the output as well, we use a projection operation to compute a target cf. value for individual infostates.

We will now describe the proposed particle value function $f_{\theta_{par}}$ in detail, see also Figure 1 . $f_{\theta_{par}}(s_{\text{pub}}, b', s_n) : \mathcal{S}_{\text{pub}} \times \mathbb{R}^{S'} \times S \to \mathbb{R}$

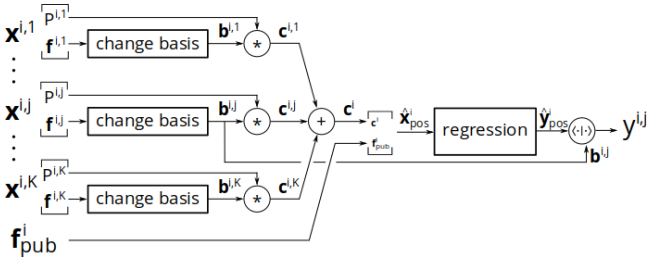

Figure 1: The architecture of the particle value function. The blocks "change basis" and "regression" can be implemented arbitrarily. All the "change basis" blocks share the same parameters θ_{cb}. The "regression" block uses parameters θ_r.

is parametrized with $\theta_{par} = (\theta_{cb}, \theta_r)$. The function takes as input public state $s_{\text{pub}} \in \mathcal{S}_{\text{pub}}$, sparse belief $b' \in \mathbb{R}^{S'}$ and a target infostate $s_n \in S'$ for which a scalar cf. value should be computed.

We use the training set TS and use samples $(\beta^i, v^i) \in TS$. So far, we described the particle value function in terms of elements of abstract sets, but we typically realize these functions as neural networks, for which we need to make create specific inputs $\boldsymbol{x}^i$ and outputs $\boldsymbol{y}^i$ based on *features*. Thus we describe how to construct $(\boldsymbol{x}^i, \boldsymbol{y}^i)$ for a given (β^i, v^i).

For (possibly sparse) beliefs $b(\beta^i)$ with size $K = |b(\beta^i)|$, we can construct inputs $\boldsymbol{x}^{i,j}, j \in (1, \ldots, K)$ for each infostate $s^{i,j}_n \in \mathcal{S}_1(s_{\text{pub}}) \cup \mathcal{S}_1(s_{\text{pub}})$. An individual infostate $s^{i,j}_n$ is then encoded as $\boldsymbol{x}^{i,j} = \begin{bmatrix} P^{i,j} & \boldsymbol{f}^{i,j} \end{bmatrix}$. The *player belief input* $P^{i,j}$ is $P^{\pi}_{n}(s^{i,j}_n)$, remarked for indexing purposes, for the infostate $s^{i,j}_n$. The *feature input* $\boldsymbol{f}^{i,j} = \begin{bmatrix} \boldsymbol{f}^{i}_{\text{pub}} & \boldsymbol{f}^{i,j}_{\text{priv(n)}} & \boldsymbol{f}^{i,j}_{n} \end{bmatrix}$ uses public features $\boldsymbol{f}^{i}_{\text{pub}}$ for s_{pub}, private features $\boldsymbol{f}^{i,j}_{\text{priv(n)}}$ for $s^{i,j}_n$, and an additional feature $\boldsymbol{f}^{i,j}_{n}$ for the index of the player n playing at $s^{i,j}_n$. The feature input is transformed via "change basis" (parametrized by θ_{cb}) into a "base" $\boldsymbol{b}^{i,j} = \text{change basis}(\boldsymbol{f}^{i,j})$. Then $\boldsymbol{b}^{i,j}$ is multiplied with player belief input $P^{i,j}$ to get a "base coordinate" $\boldsymbol{c}^{i,j} = P^{i,j}\boldsymbol{b}^{i,j}$. The individual $\boldsymbol{c}^{i,j}$ are grouped together using the addition operation $\boldsymbol{c}^{i} = \sum_j \boldsymbol{c}^{i,j}$ which we call **context**. Intuitively, the context represents the (now possibly compressed) beliefs $b(\beta^i)$. We take the context $\boldsymbol{c}^i$, concatenate it with $\boldsymbol{f}^{i}_{\text{pub}}$ and obtain $\hat{\boldsymbol{x}}^{i}_{\text{pos}} = \begin{bmatrix} \boldsymbol{c}^{i} & \boldsymbol{f}^{i}_{\text{pub}} \end{bmatrix}$ that we pass through "regression" (parametrized by θ_r) to compute $\hat{\boldsymbol{y}}^{i}_{\text{pos}} = \text{regression}(\hat{\boldsymbol{x}}^{i}_{\text{pos}})$.

Finally, we use inner product $\langle \hat{\boldsymbol{y}}^{i}_{\text{pos}}, \boldsymbol{b}^{i,j} \rangle = y^{i,j}$ to compute projection $y^{i,j}$, the target regressed cf. value of player n for the infostate $s^{i,j}_n$. We also considered other architecture variants to compute the outputs, such as concatenation of $\hat{\boldsymbol{y}}^{i,j}_{\text{pos}} = \begin{bmatrix} \boldsymbol{b}^{i,j} & \hat{\boldsymbol{y}}^{i}_{\text{pos}} \end{bmatrix}$ instead of the projection (see Appendix C for details). In Figure 2 we show that the variant which uses the projection can efficiently learn the unique solution, unlike the concatenation, for the following example of learning linear targets.

5.3 Learning Linear Targets

We called the individual components "change basis", "base", "base coordinate" and "regression" with these suggestive names, because

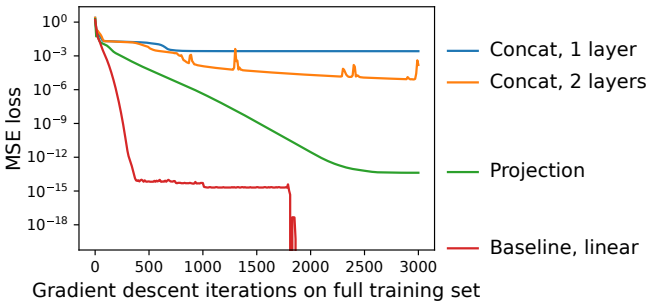

Figure 2: Comparison of various architectures for solving the linear regression task with $n = 10$. The baseline linear model trained on TS (Appendix D) quickly reaches low loss and eventually underflows machine float precision to zero. We learn the particle model variants on the training set with features TS_F (Appendix D). The variant with output projections takes longer to learn but eventually reaches a very low loss of 10^{-13}. While there exist solutions for the concatenation variant (Appendix C), they are hard to learn. The training can be even unstable, and the loss sometimes jumps several orders of magnitude.

this is exactly what they represent in the case of learning targets $\boldsymbol{y}$ that depend linearly on inputs $\boldsymbol{x}$, i.e. $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$ with $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. In Appendix D, we analyze this special case, as it gives insight into representation power of particle value functions and how they can achieve belief compression [26].

Consider what happens if $\text{rank}(\mathbf{A}) = m < n$, i.e. $\mathbf{A}$ is not full rank and it has some zero eigenvalues. Then it means that only an appropriate subset of inputs is needed to learn $\mathbf{A}$ [23]. This is analogous to using principal component analysis [12] to find a subspace with sparse beliefs: this method has been used for belief compression in POMDPs [26]. As equilibria often have small supports, belief compression over a small set of particles is therefore also viable for public belief states.

5.4 Particle Value Functions Can Represent Positional Value Functions

The intuition behind learning linear targets using the particle value function is also useful for understanding the proof of the following theorem. Proof is in the Appendix E.

Theorem 4. *For any positional value function $f_{\theta_{pos}}$, parametrized by θ_{pos}, there exists a particle value function $f_{\theta_{par}}$, parametrized by θ_{par}, such that $f_{\theta_{par}}(s_{\text{pub}}, b)(s_{\text{priv(n)}}) = f_{\theta_{pos}}(s_{\text{pub}}, b)(s_{\text{priv(n)}})$ for each $\beta \in B, \beta = (s_{\text{pub}}, b)$ and $s_n \in \mathcal{S}_1(s_{\text{pub}}) \cup \mathcal{S}_2(s_{\text{pub}})$.*

6 EXPERIMENTS

In the experiments, we evaluate value functions (VF) on two domains: Leduc Poker [33] and imperfect-information GoofSpiel with $N = 5$ cards. The error of equilibrium approximation within the depth-limited subgames is computed using trunk exploitability [28]. Full experiments details are in Appendix F.

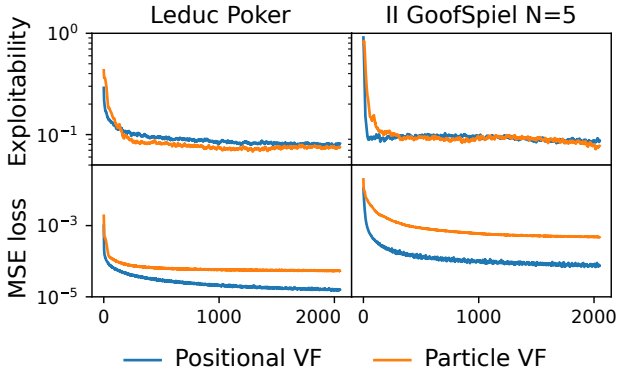

Figure 3: Comparison of the positional and particle value functions during training of the neural networks.

First, we show empirically that the particle VF is able to learn a value approximation that results in similar or better trunk exploitability as the positional VF. In Figure 3 we plot trunk exploitability and mean square error loss as a function of training iterations of the neural networks. The model parameters are trained to minimize the MSE loss between the predictions and targets from the training set TS. The functions are then used to predict the cf. values for leaf infostates of the depth-limited subgame and are used within 100 inner loop iterations of CFR+. We compute the trunk exploitability 2048 times during the training which corresponds to 155 and 932 epochs over the training sets. The plots are averages over 10 runs with differently randomized training sets and neural network parameters.

Both architectures are trained on exactly the same canonical data TS, so the evaluations can be directly compared. Initially, both the MSE loss and the trunk exploitability improves faster for the positional VF. However, the particle VF matches the exploitability of positional VF after ≈ 11 and ≈ 134 epochs. When the full training is over, the particle VF produces strategies with a slightly lower exploitability within a factor of 0.1, in spite of the MSE loss being several orders of magnitude worse than the loss of the positional VF. While it has been shown that the loss correlates with exploitability and hence loss minimization can be treated as a surrogate problem [20, 28], the relation has not been shown to be linear. We speculate that the particle VF has inductive biases which produce cf. values that are more helpful for the CFR+ iterations in the depth-limited subgame.

Finally, we show that using the particle VF with a well-selected subset of histories allows for depth-limited solving of a sparsified depth-limited look-ahead tree. We compute an equilibrium $\pi^\star$ for the whole game and let players use it to play a small number of moves (3 in Leduc and 2 in GoofSpiel). We collect all the histories H that are within the belief supports of $\pi^\star$ and use a fraction of them to build sparsified depth-limited look-ahead trees.

In Figure 4 we take a trained particle VF and show that trunk exploitability degrades gracefully with the fraction of histories in H selected based on their reach probability under $\pi^\star$. We fix the

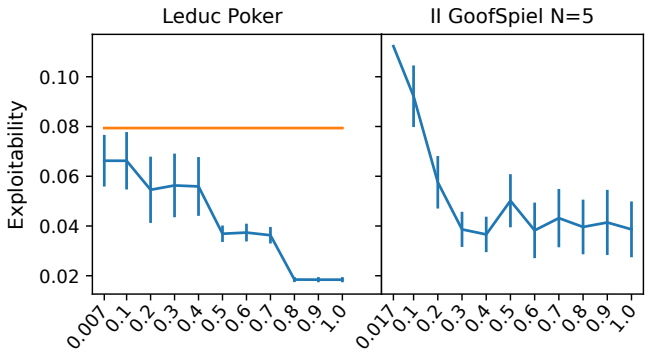

Figure 4: Trunk exploitability evaluated on sparse trunks built with fractions of root histories (particles), averaged over 10 runs. The vertical bars indicate the standard deviation. As we select smaller fractions of particles, the performance degrades gracefully. The left-most points with smallest fractions corresponds to using a single particle (in GoofSpiel) or two particles (in Leduc). The horizontal bar in the Leduc plot shows the exploitability of a fully uniform strategy. We omit this information for II GoofSpiel from the figure – its value is 0.42, $\approx$ 3.5 times higher than the single-history exploitability.

strategy for the non-selected infostates as uniform for the evaluation. We find that it is possible to compute a strategy with low trunk exploitability even when using a fraction of histories from the beliefs support. This implies that the particle VF can be used within the full particle algorithm even when the set of particles is smaller the supports.

7 CONCLUSION

We introduced a novel neural network architecture, called particle value function, that can approximate counterfactual values used for iterative Nash equilibrium approximation. Unlike the current method, positional value functions, which must use full beliefs for inputs, particle value functions accept arbitrary subsets of beliefs. We proved that particle value functions are at least as expressive as positional value functions. We empirically demonstrated that when provided with the full belief, particle value functions sometimes even outperform the positional ones.

Our main focus is on the situations, where using the full belief is impossible because of its size. We show that in that case, using only the particles with non-zero reach probability still leads to near-optimal play. Moreover, further reduction of the number of considered particles gracefully reduces the quality of the computed strategies. In Imperfect Information Goofspiel with 5 cards, only the 30% histories visited with the highest probability are sufficient to achieve practically the same exploitability as using the full belief.

The particle value function can be used to approximate equilibrium strategies in the full game, similarly to how it has been done for the positional value function. However, using particle value functions dramatically expands the class of games where limited look-ahead game solving methods are applicable, as many games have public states, which are prohibitively large for evaluation by positional value functions. Additionally, the proposed architecture can also be used for the single-agent version of the problem (i.e., POMDPs), and it can also be extended to continuous action spaces in both POMDPs and FOSGs.

ACKNOWLEDGMENTS

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures. This work was supported by Czech science foundation grant no. 18-27483Y.

REFERENCES

[1] Dimitri P Bertsekas. 1995. *Dynamic programming and optimal control*. Vol. 1. Athena scientific Belmont, MA.

[2] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. 2014. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research* 51 (2014), 829–866.

[3] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games. *arXiv preprint arXiv:2007.13544* (2020).

[4] Noam Brown, Tuomas Sandholm, and Brandon Amos. 2018. Depth-limited solving for imperfect-information games. In *Advances in Neural Information Processing Systems*. 7663–7674.

[5] Neil Burch, Michael Johanson, and Michael Bowling. 2014. Solving imperfect information games using decomposition. In *AAAI*. 602–608.

[6] Hanqing Chao, Yiwei He, Junping Zhang, and Jianfeng Feng. 2019. Gaitset: Regarding gait as a set for cross-view gait recognition. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 8126–8133.

[7] Andrew Gilpin and Tuomas Sandholm. 2007. Lossless abstraction of imperfect information games. *Journal of the ACM (JACM)* 54, 5 (2007), 25–es.

[8] William L Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. *arXiv preprint arXiv:1806.01445* (2018).

[9] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. 2004. Dynamic programming for partially observable stochastic games. In *AAAI*, Vol. 4. 709–715.

[10] Feng-Hsiung Hsu. 2006. *Behind Deep Blue: Building the Computer that Defeated the World Chess Championship*. Princeton University Press.

[11] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. 2013. Evaluating State-Space Abstractions in Extensive-Form Games. *12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013* 1, 271–278.

[12] Ian T Jolliffe. 1986. Principal components in regression analysis. In *Principal component analysis*. Springer, 129–155.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Daphne Koller and Nimrod Megiddo. 1996. Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory* 25, 1 (1996), 73–92.

[15] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. 2019. Rethinking formal models of partially observable multiagent decision making. *arXiv preprint arXiv:1906.11110* (2019).

[16] Viliam Lisy and Michael Bowling. 2017. Eqilibrium approximation quality of current no-limit poker bots. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

[17] Viliam Lisý, Marc Lanctot, and Michael Bowling. 2015. Online Monte Carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 27–36.

[18] Andrew McLennan and Johannes Berg. 2005. Asymptotic expected number of Nash equilibria of two-player normal form games. *Games and Economic Behavior* 51, 2 (2005), 264–295.

[19] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. 2003. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 536–543.

[20] Matej Moravcik, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356, 6337 (2017), 508–513.

[21] J v Neumann. 1928. Zur theorie der gesellschaftsspiele. *Mathematische annalen* 100, 1 (1928), 295–320.

[22] Tomáš Pevný and Petr Somol. 2017. Using neural network formalism to solve multiple-instance problems. In *International Symposium on Neural Networks*. Springer, 135–142.

[23] Elad Plaut. 2018. From principal subspaces to principal components with linear autoencoders. *arXiv preprint arXiv:1804.10253* (2018).

[24] Huilin Qu and Loukas Gouskos. 2020. Jet tagging via particle clouds. *Physical Review D* 101, 5 (2020), 056019.

[25] Mark Richards and Eyal Amir. 2012. Information set generation in partially observable games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[26] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. 2005. Finding approximate POMDP solutions through belief compression. *Journal of artificial intelligence research* 23 (2005), 1–40.

[27] Martin Schmid, Matej Moravcik, and Milan Hladik. 2014. Bounding the support size in extensive form games with imperfect information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

[28] Dominik Seitz, Vojtech Kovarík, Viliam Lisý, Jan Rudolf, Shuo Sun, and Karel Ha. 2019. Value Functions for Depth-Limited Solving in Imperfect-Information Games beyond Poker. *arXiv preprint arXiv:1906.06412* (2019).

[29] Dominik Seitz, Nikita Milyukov, and Viliam Lisý. 2021. Learning to guess opponent's information in large partially observable games. In *AAAI Workshop on Reinforcement Learning in Games*.

[30] David Silver, 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. *Science* 362, 6419 (2018), 1140–1144.

[31] David Silver and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2*. 2164–2172.

[32] Trey Smith and Reid Simmons. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. 520–527.

[33] Finnegan Southey, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. 2012. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411* (2012).

[34] Michal Šustr, Vojtěch Kovařík, and Viliam Lisý. 2019. Monte Carlo continual resolving for online strategy computation in imperfect information games. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 224–232.

[35] Sebastian Thrun. 1999. Monte Carlo POMDPs.. In *NIPS*, Vol. 12. 1064–1070.

[36] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. 2017. Deep sets. *arXiv preprint arXiv:1703.06114* (2017).

A SIZE OF PUBLIC STATES

Lemma 2. *For Imperfect Information Goofspiel with N cards, in any depth k, there is a public state with at least*

$$X(k) = \frac{N!(N-1)!}{(N-k)!(N-1-k)!2^k}$$

histories and at least $2\sqrt{X(k)}$ *distinct information states.*

Proof. The proof is by a repeated application of the pigeonhole principle. First we count all histories in depth k without any draws. The first player can choose any sequence of k out of N cards. The second player can choose any of his remaining cards, with the exception of the card selected by player 1.

$$nonDraw(k) = \frac{N!}{(N-k)!}\frac{(N-1)!}{(N-1-k)!}$$

All these non-draw histories must fall to one of the public states without any draw in the history of public observations and there are 2^k of them in depth k. Hence, there is at least one public state with

$$X(k) = \frac{N!(N-1)!}{(N-k)!(N-1-k)!2^k} \text{ histories.}$$

Furthermore, we realise that each pair of the histories counted above must be distinguishable by at least one of the players. This is because each history is uniquely identified only by the cards played by both players. Now assume that player 1 partitions the $X(k)$ histories in the public state into m information sets. Then the largest information set will include at least $\frac{X(k)}{m}$ histories. Since player 2 must be able to tell all these histories apart, she has to have at least $\frac{X(k)}{m}$ information sets. Therefore, the number of information sets of both players is at least

$$Y(m) = m + \frac{X(k)}{m}.$$

This value is minimised if $m = \sqrt{X(k)}$, when $Y(m) = 2\sqrt{X(k)}$. □

For example in Imperfect Information Goofspiel with the standard number of 13 cards, the number of histories and information sets is shown in Table 1

B RECURSIVE MAX-MIN OPTIMIZATION

Lemma 3. *The max-min optimization* (1) *can be recursively decomposed using the formula*

$$u_n(s_{\text{pub}}, b) = \max_{\pi_1|_{\mathcal{S}_1(s_{\text{pub}})}} \min_{\pi_2|_{\mathcal{S}_2(s_{\text{pub}})}} \sum_{s'_{\text{pub}} \in \text{succ}(s_{\text{pub}})} u_n(s'_{\text{pub}}, \tilde{b}),$$

where $\tilde{b}$ *denotes the belief obtained via a Bayesian update of* b *using the policy* $\pi|_{\mathcal{S}(s_{\text{pub}})}$ *and* $\text{succ}(s_{\text{pub}})$ *are successor public states of* s_{pub}.

Proof. The *base case* is a terminal PBS $\beta_\perp$, where the $s_{\text{pub}}(\beta_\perp)$ is terminal, i.e. all histories h that have an observation sequence $s_{\text{pub}}(\beta_\perp)(h)$ are terminal. The utility $u_n(\beta_\perp)$ depends only on the beliefs of the players, the players have no choice between actions and they simply receive a scalar utility which is minimax optimal.

The value of $u_n(\beta) = u_n(s^d_{\text{pub}}, b_1, b_2)$ of a non-terminal $s^d_{\text{pub}}(\beta)$ at depth d with a set of successor states $s^{d+1}_{\text{pub}} \in \text{succ}(s^d_{\text{pub}}(\beta))$ at depth $d+1$ can be written as

$$u_n(s^d_{\text{pub}}, b_1, b_2) = \max_{\pi_1^d} \min_{\pi_2^d} \sum_{s^{d+1}_{\text{pub}}} u_n(s^{d+1}_{\text{pub}}, b_1 \otimes \pi_1^d, b_2 \otimes \pi_2^d),$$

where we used the symbol $\otimes$ to denote succintly the Bayesian beliefs update for the successor public states between the depths d and $d+1$.

The *induction step* expands the summation of expected utilities to depth $d+2$:

$$\begin{aligned}
&u_n(s^d_{\text{pub}}, b_1, b_2)\\
&= \max_{\pi_1^d} \min_{\pi_2^d} \sum_{s^{d+1}_{\text{pub}}}\\
&\quad \max_{\pi_1^{d+1}} \min_{\pi_2^{d+1}} \sum_{s^{d+2}_{\text{pub}}} u_n(s^{d+2}_{\text{pub}}, b_1 \otimes \pi_1^d \otimes \pi_1^{d+1}, b_2 \otimes \pi_2^d \otimes \pi_2^{d+1})\\
&= \max_{\pi_1^d} \min_{\pi_2^d} \sum_{s^{d+1}_{\text{pub}}} \sum_{s^{d+2}_{\text{pub}}} u_n(s^{d+2}_{\text{pub}}, b_1 \otimes \pi_1^d, b_2 \otimes \pi_2^d)\\
&= \max_{\pi_1|_{\mathcal{S}_1(s_{\text{pub}})}} \min_{\pi_2|_{\mathcal{S}_2(s_{\text{pub}})}} \sum_{s'_{\text{pub}} \in \text{succ}(s_{\text{pub}})} u_n(s'_{\text{pub}}, \tilde{b}),
\end{aligned}$$

where we used the minimax property [21] to swap out the terms as follows (we omit some notation for clarity):

$$\begin{aligned}
&\max_d \min_d \sum_{d+1} \max_{d+1} \min_{d+1} \sum_{d+2} = \max_d \min_d \max_{d+1} \sum_{d+1} \min_{d+1} \sum_{d+2}\\
&= \max_d \min_d \max_{d+1} \min_{d+1} \sum_{d+1} \sum_{d+2} = \min_d \max_d \max_{d+1} \min_{d+1} \sum_{d+1} \sum_{d+2}\\
&= \min_d \max_d \min_{d+1} \sum_{d+1} \sum_{d+2} = \max_d \min_d \min_{d+1} \sum_{d+1} \sum_{d+2}\\
&= \max_d \min_d \sum_{d+1} \sum_{d+2}.
\end{aligned}$$

Finally, in the root PBS $\beta_\varnothing$ all beliefs b are equal to 1, as it's the start of the game, and the $u_1(\beta_\varnothing) = u^\star$, i.e. the result of the original problem (1). □

C ARCHITECTURE VARIANTS

Let us investigate architecture variants for the particle value function by using the linear regression example. While we used an inner product to compute individual outputs $y^{i,j}$, there are other natural choices. For example, we could use a concatenation $\hat{\boldsymbol{y}}^{i,j}_{\text{pos}} = \begin{bmatrix}\boldsymbol{b}^{i,j} & \hat{\boldsymbol{y}}^i_{\text{pos}}\end{bmatrix}$, regressed to compute $y^{i,j}$ through a new block "projection regression" , parametrized with θ_{pr} and thus augmenting θ_{par} with new set of parameters. However the concatenation requires the use of RELUs within the "projection regression" to find a solution with zero loss (for the regression task), so the function is no longer linear and there may be local optima that are not global.

When we tried experimentally to learn the proposed particle model with either projection or the concatenation, we found that it is hard even for this simple problem and the loss plateaus at 10^{-3}. If we use two layers of RELU neurons instead of just one, the loss is smaller, but still significantly worse than our proposed projection operation. Additionally, the training is also unstable: the loss occasionally jumps several orders of magnitude. See Figure 2 for details.

Depth	0	1	2	3	4	5	6	7	8	9	10	11	12
Histories	1	78	5148	283140	1.3e7	4.6e8	1.3e10	2.7e11	4.0e12	4.0e13	2.4e14	7.3e14	7.3e14
Info. sets	2	17.66	144	1064	7139	42834	2.2e5	1.0e6	4.0e6	1.3e7	3.1e7	5.4e7	5.4e7

Table 1: The lower bound on the number of histories and information sets in the largest public state of Imperfect Information Goofspiel with 13 cards.

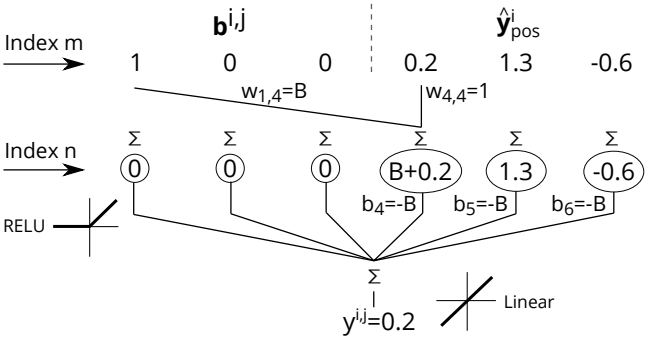

Figure 5: An alternative architecture for computing the individual outputs $y^{i,j}$ uses concatenation of the "base" $\boldsymbol{b}^{i,j}$ and "regressed values" $\hat{\boldsymbol{y}}^i_{\mathbf{pos}}$ as input. We show there exists a solution for one layer of fully connected RELU neurons with one additional output neuron as a summation. The weights of RELU neurons are $w_{m,n}$, $m, n \in (1, \ldots, I)$ where input size I is $I = I_b + I_y = |\boldsymbol{b}^{i,j}| + |\hat{\boldsymbol{y}}^i_{\mathbf{pos}}|$. The solution critically relies on a large constant $B > 0$ that ensures that we can copy input, at position as indicated in the one-hot encoded $\boldsymbol{b}^{i,j}$, to the output. The constant B must be larger than any possible input. The selection is done by setting weights $w_{k-I_b,k} = B$ and $w_{k,k} = 1$ and biases $b_k = -B$ with $\forall k \in \{I_b + 1, \ldots, I\}$. The weights for the output neuron $w_{\mathbf{out}}$ are all 1. All other weights and biases are zero.

D LEARNING LINEAR TARGETS

Consider a standard regression problem of learning a linear function $f_{\theta_r}(\boldsymbol{x})$, parametrized by $\theta_r \in \mathbb{R}^{n \times n}$, i.e. f_{θ_r} is a single fully connected layer of n perceptron with a linear activation function. The function f_{θ_r} correponds to the positional value function in this example. (We set the features f_{pub} as empty.)

The training set is $TS = \{(\boldsymbol{x}^i, \boldsymbol{y}^i) \in \mathcal{X} \times \mathcal{Y} \mid i \in \{1, \ldots, L\}\}$, where the inputs $\boldsymbol{x}^i \in \mathbb{R}^n$ are some random points and the targets $\boldsymbol{y}^i = \mathbf{A}\boldsymbol{x}^i$ are generated based on some unknown matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ we would like to learn. If $\mathbf{X} = \begin{bmatrix} \boldsymbol{x}^{1,\top} & \ldots & \boldsymbol{x}^{L,\top} \end{bmatrix} \in \mathbb{R}^{L \times n}$ has $\text{rank}(\mathbf{X}) = n$ we can learn $\mathbf{A}$ using linear regression as $\arg\min_{\theta_r \in \mathbb{R}^{n \times n}} \left\| f_{\theta_r}(\mathbf{X}) - \mathbf{Y} \right\|_2^2$.

We'd like to solve the linear regression task with a particle value function $f_{\theta_{par}}$. As before, $\theta_{par} = (\theta_{cb}, \theta_r)$. We use $\theta_{cb} \in \mathbb{R}^{n \times n}$, also a single fully connected layer of n perceptron with linear activation function. We will construct a derived training set TS_F for the particle value function, which will add "features" to each of the inputs and transform them into a set (represented by a matrix).

The derived training set is $TS_F = \{(F(\boldsymbol{x}^i), G(\boldsymbol{y}^i)) \mid (\boldsymbol{x}^i, \boldsymbol{y}^i) \in TS\}$ where feature augmentation $F : \mathbb{R}^n \to \mathbb{R}^{n \times (n+1)}$ and target augmentation $G : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ are defined as

$$F(\boldsymbol{P}^i) = \begin{bmatrix} \boldsymbol{P}^i & \boldsymbol{f}^i \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}^i & \mathbf{I} \end{bmatrix} \quad \text{and}$$
$$G(\boldsymbol{y}^i) = \begin{bmatrix} \boldsymbol{y}^i & \boldsymbol{y}^i & \ldots & \boldsymbol{y}^i \end{bmatrix},$$

where $\mathbf{I}$ is identity matrix.

For the particle value function we solve the regression

$$\arg\min_{\theta_{par} \in \mathbb{R}^{n \times n} \times \mathbb{R}^{n \times n}} \sum_{i=1}^{L} \sum_{j=1}^{n} \left\| f_{\theta_{par}}(\boldsymbol{x}^i, j) - \boldsymbol{y}^{i,j} \right\|_2^2,$$

where samples $(\boldsymbol{x}^i, \boldsymbol{y}^i) \in TS_F$. As the function $f_{\theta_{par}}$ is a linear combination of linear functions there is a single local optimum for the optimization task. The solution is trivially $\theta_{cb} = \mathbf{I}$ and $\theta_r = \mathbf{A}$.

Moreover, if feature augmentation F used a regular matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$ instead of identity $\mathbf{I}$, the solution can still be found with $\theta_{cb} = \mathbf{T}^{-1}$ as the inverse will **change the basis** appropriately to $\mathbf{I} = \mathbf{T}\mathbf{T}^{-1}$. This shows that the names given to the each of the blocks of the particle value function can be indeed interpreted as basic constructs from linear algebra.

E REPRESENTATION POWER OF PARTICLE VALUE FUNCTIONS

Theorem 4. *For any positional value function $f_{\theta_{pos}}$, parametrized by θ_{pos}, there exists a particle value function $f_{\theta_{par}}$, parametrized by θ_{par}, such that $f_{\theta_{par}}(s_{\text{pub}}, b)(s_{\text{priv(n)}}) = f_{\theta_{pos}}(s_{\text{pub}}, b)(s_{\text{priv(n)}})$ for each $\beta \in B, \beta = (s_{\text{pub}}, b)$ and $s_n \in \mathcal{S}_1(s_{\text{pub}}) \cup \mathcal{S}_2(s_{\text{pub}})$.*

Proof. By construction. For the particle value function with parameters $\theta_{par} = (\theta_{cb}, \theta_r)$ we find θ_{cb} such that for any input they recreate one-hot positional encoding as "base" $\boldsymbol{b}^{i,j}$, that is then multiplied with the player belief input to produce $\boldsymbol{c}^{i,j}$. By summing over all $\boldsymbol{c}^{i,j}$ and concatenating public features we recreate the input for the positional value function. The parameters $\theta_r = \theta_{pos}$. The output is a projection to the "base" $\boldsymbol{b}^{i,j}$ for each positionally encoded infostate $s_n^{i,j}$.

Let $\mathbf{F} \in \mathbb{R}^{f \times s}$ be a matrix of features $\mathbf{F} = \begin{bmatrix} \boldsymbol{f}^1 & \boldsymbol{f}^2 & \ldots & \boldsymbol{f}^s \end{bmatrix}$, where columns are features $\boldsymbol{f}^i$ of size f for every infostate in the game $s_n^i \in \mathcal{S}_1 \cup \mathcal{S}_2$ and $s = |\mathcal{S}_1 \cup \mathcal{S}_2|$. The matrix $\mathbf{F}$ has full rank column space: we use perfect recall public and private features in $\boldsymbol{f}^i$, which uniquely identify every infostate $s_n \in \mathcal{S}_n$ and the player features f_n disambiguate the players as well. Let $p = |\mathcal{S}_{\text{priv}(1)}(s_{\text{pub}}) \cup \mathcal{S}_{\text{priv}(2)}(s_{\text{pub}})|$ and $\mathbf{E} \in \mathbb{R}^{s \times p}$ be a matrix constructed as follows:

$$\mathbf{E}_{i,j} = \begin{cases} 1 & \text{if } PE(s_n^i) = j \\ 0 & \text{otherwise,} \end{cases}$$

where $PE(\cdot)$ is some fixed enumeration of all possible private histories. We will find a matrix $\mathbf{T} \in \mathbb{R}^{s \times p}$ which satisfies

$$\mathbf{F}^\top \mathbf{F}\mathbf{T} = \mathbf{E}.$$

Since $\mathbf{F}$ has full column rank, $\mathbf{F}^\top\mathbf{F}$ is a regular matrix and $\mathbf{T}$ can be computed as

$$\mathbf{T} = (\mathbf{F}^\top\mathbf{F})^{-1}\mathbf{E}.$$

The "change basis" can be constructed as a single layer of perceptron neurons with linear activation function, where their parameters θ_{cb} represent the matrix $\mathbf{FT}$. They encode the basis transformation for the inputs as well the output projection. □

F EXPERIMENT DETAILS

Algorithms with depth-limited lookahead use value functions to evaluate the public state leaves at depth d. The depth-limited subgame rooted at β is called a **trunk** [5] and we denote it as $Tr(\beta, d)$. A trunk has a strategy profile $\pi^{Tr} \subset \pi \in \Pi$ restricted only to the infostates within the (depth-limited) trunk. The error of equilibrium approximation within the trunk can be computed using trunk exploitability [28] as

$$\begin{aligned} expl(\pi^{Tr}) = &- \max_{\pi_1 \setminus \pi_1^{Tr} \in \Pi_1} \min_{\pi_2 \in \Pi_2} u_1(\pi_1^{Tr} \cup \pi_1, \pi_2) \\ &- \max_{\pi_1 \in \Pi_1} \min_{\pi_2 \setminus \pi_2^{Tr} \in \Pi_2} u_2(\pi_1, \pi_2^{Tr} \cup \pi_2) \end{aligned}$$

in the subgame $G(\beta)$.

For the positional value function we use 5 fully connected layers with hidden size of 5× the input. For the particle value function we used 3 fully connected layers with hidden size of 3× the input for the "change basis" module and use the same architecture for the "regression" module as used for the positional value function. All hidden layers use RELU activation function and the networks are trained using the Adam optimizer [13].

For the first experiment, the trunk is rooted in the initial PBS of the game (at $d = 0$) and it has depth $d = 5$ for Leduc and $d = 2$ for GoofSpiel. We generate a training set TS based on random sampling of PBS as in [20, 28], with sizes $|TS| = 54000$ and $|TS| = 9000$ respectively. To compute the target values we run 100 CFR+ iterations in the subgames. To compute the trunk strategy we run 100 iterations of CFR+ in the trunk. We then compute the trunk exploitability for the (average) strategy as approximated by CFR+.

For the second experiment, if players do not know opponent's strategy and they have played some number of moves before the current trunk, we technically have to use we need to a resolving game [5] to be able to evaluate trunk exploitability correctly. To avoid creating resolving games we make that the $\pi^\star$ is common knowledge between the players by introducing a chance node which reflects the beliefs for $\pi^\star$. Then we can use only a value-solving game in this experiment, which produces consistent strategies in this case, and we can still use trunk exploitability for evaluation.