Learning to guess opponent's information in large partially observable games

Dominik Seitz*, Nikita Milyukov, Viliam Lisý
seitzdom,milyunik,lisyvili @fel.cvut.cz

Abstract

Online game playing algorithms in imperfect-information games (IIG) keep track of a set of states they consider possible, given the encountered situations, to compute a strategy. Due to exponential branching of the opponent's private information, a large portion, or even all, of the tracked states may suddenly become incompatible with a received observation. In that case, new compatible states quickly need to be identified. This problem is called information set generation. Our contributions include (1) introducing a domain-independent way to encode both an agent's sequence of actions and observations (AOH) as well as the corresponding uncertainty over the opponent's information (OPI). (2) A neural network architecture capable of learning the mapping from AOH to a heuristic for efficient generation of OPI. (3) Algorithms for generating the data for training the neural network using a game simulator. In our empirical evaluation on three large domains, we show that with time budgets suitable for online game playing, our algorithm significantly outperforms a baseline both in the number of states generated and in the ability to find states where the baseline is unable to find any.

Introduction

To assess the performance of game playing algorithms, it has become increasingly popular to let them compete against humans. This is especially of interest for games where it is infeasible to find exact solutions due to their sizes. Computing strategies for game playing can be performed in two ways. Offline approaches find strategies for all possible situations that can occur prior to playing the game, which ensures that an agent will never encounter an unknown state. The drawbacks of this approach are the computational requirements to optimally solve large games and being able to store the full game in memory, which can be infeasible. Online game playing algorithms, which are the focus of this work, in contrast, compute strategies only for actually encountered situations while playing the game and have proven to be able to beat humans in large perfect- and imperfect-information games (Hsu 2004; Silver et al. 2017; Moravčík et al. 2017; Brown et al. 2020). In IIG, an agent only knows about his information set, a set of possible states given his actions and observations. Online game playing algorithms optimize strategies for both players, even in parts of the game where they are not actually playing. However, the branching of the opponent's private information is often exponential in the number of opponent actions, i.e., Kriegspiel, whereas in poker the opponent's private information does not change. Hence, while keeping track of a constant size set of states an agent considers possible, he might end up in a situation where all tracked states become inconsistent with a newly received observation. In that case, the agent needs to quickly assess which opponent actions are compatible with his own actions and observations. In literature this is referred to as information set generation (ISG) (Richards and Amir 2012). Traditionally, games are represented as extensive form games (EFG) (Morgenstern and Von Neumann 1953) and state-of-the-art solutions address the problem of information set generation by implementing the domain in a logic-based game description language (GDL), e.g.(Thielscher 2010). When asking for nodes in an information set, a constraint satisfaction problem is constructed using the GDL. Its solution are all states compatible with the sequences of observations in that information set (Richards and Amir 2012). The GDL description however, requires expert human effort to encode and is a major speed limitation when used during online game play. In addition to that, EFGs lack any notion of observations. Factored observation stochastic games (FOSG)(Kovařík et al. 2019), a variant of partially observable stochastic games (Hansen, Bernstein, and Zilberstein 2004), in contrast describe information in games in terms of sequences of actions and private as well as public information, where players receive subsets of each . For any state in the game, a FOSG maintains a 'memory' of sequences of actions and observations that lead to it. Hence, the uncertainty in a game can be viewed strictly in terms of actions and observations, rather than just abstract information-partitioning used by EFGs. This enables us to ask which opponent actions and observations are compatible with what the player did and observed. Recent results on Atari games (Mnih et al. 2013) were solely based on an agent having access to a simulator in order to train. Having to express every single game concisely in a description language would be expensive. In this work we provide a solution to the problem of information set generation in large IIG focused on online game playing. In this setting, agents only

*Corresponding author

have access to a simulator which provides observations from the environments. To encode them, we use FOSG which is easier to attain when applied to a large set of games.

Contributions Our contributions are the following: (1) We show how to use FOSG to encode which opponent's private information (OPI) is compatible with a player's action-observation-history (AOH). We extend this encoding to distributions over possible opponent private information. (2) We present a domain-independent neural network architecture capable of learning a heuristic, which speeds up the efficient generation of OPI. (3) We provide algorithms for generating the data for training the neural network using a game simulator. (4) In our empirical evaluation on three large domains, we show that with time budgets suitable for online game playing, our algorithm significantly outperforms a baseline both in terms of the number of valid states generated and in the ability to find states where the baseline is unable to find any.

The purpose of our algorithm is first and foremost to generate states, not to explicitly model the opponent. Our framework, however, allows for this as well, as it is possible to use any (e.g., near-optimal) opponent strategy to generate the training data. The goal of this paper is to show that even using arbitrary (uniform) strategies, we are able to find nodes anywhere in the tree, even when training only from a small fraction of all information sets in domains with non-trivial properties. This shows that the neural network is able to learn the logic of an arbitrary game just by training on a relatively small number of example matches and give meaningful estimations about possible opponent information.

Background

Factored-observation stochastic games (FOSG), a variant of partially-observable stochastic games, describes information in games in terms of sequences of actions and private as well as public observations (Kovařík et al. 2019). A two-player zero-sum factored-observation stochastic game is a tuple $G = \langle W, w^o, A, T, R, O \rangle$; W is the set of *world states*; $w^0 \in W$ denoting the initial state; $A = A_1 \times A_2$ is the space of *joint actions*, with subsets $A_p(w) \subset A_p$ denoting the actions available to each *player* $p \in \{1, 2\}$ at $w \in W$. If players perform a joint action $a = (a_1, a_2)$ at w, player 1 is awarded (but doesn't observe) the *reward* $R_1(w, a)$. In contrast, player 2 receives its opposite $R_2(w, a) = -R_1(w, a)$. Afterwards, the *transition function* T determines the next world state w', drawn from the probability distribution $T(w, a) \in \Delta(W)$. The *observation function* $O = (O_{\text{priv}(1)}, O_{\text{priv}(2)}, O_{\text{pub}})$ yields the factorized observations, the private part (for each player) and the public part (for both of them). Upon reaching w', each player receives $O_p(w, a, w') = \big(O_{\text{priv}(p)}(w, a, w'), O_{\text{pub}}(w, a, w')\big)$.

A *history* in FOSG is a finite sequence $h = (w^0, a^0, w^1, a^1, \ldots, w^t)$, where $w^k \in W$, $a^k \in A(w^k)$, and $T(w^k, a^k, w^{k+1}) > 0$. The game always ends after a finite number of transitions, whereas the set of terminal histories is denoted $\mathcal{Z}$. A particular history can be viewed from both players' perspectives in terms of their knowledge or uncertainty about the current state. This is referred to as an *information-state* (info-state) or *action-observation history* (AOH) and describes the sequence of actions performed by a particular player and the observations received along the way. It is defined as

$$s_p(h) := \big(O_p(-, -, w^0), a_p^0, O_p(w^0, a^0, w^1), a_p^1, \ldots \\ \ldots, a_p^{t-1}, O_p(w^{t-1}, a^{t-1}, w^t)\big),$$

where $O_p^0(-, -, w^0)$ denotes the initial observation at the start of the game. The corresponding *information* set (infoset) $I_p(s_p) := \{h \in \mathcal{H} \mid s_p(h) = s_p\}$, is the set of all histories compatible with a information-state. We will refer to histories as trajectories τ interchangeably throughout the paper.

Example of an Information Set Consider a variant of Goofspiel where two players and a referee each hold Jack, Queen and King. The referee publicly shows one of his three cards, both players then privately bet one of their cards to win the public card. The public card is then awarded to the player with the higher private bet. Hence, the players learn who won which card, but they do not know with which private card. Now, consider the public state after the first round where the King was shown and player one won. If player one played King, his action-observation-history is $s_1(h) := (O_{\text{pub}}(King), a_1(King), O_{\text{pub}}(Win))$. The information set $s_1(h)$ contains both the world state where player two played "Jack" and one where he played "Queen".

Figure 1 showcases the above described game using FOSG. We only show the sub-tree where player one bet "King". It has three children, two with a "win" and one with a "draw" public observation. Note that through the course of the game FOSG keeps track of sequences of actions and observations, for example shown by "K,P1 Win", a descendant of the public state which contains only "K". The figure contains two world states compatible with that particular sequence of public observations. One where player two bet "Jack" and one where he bet "Queen".

Encoding Action-Observation Sequences

Figure 1 demonstrates that one player's action-observation-history (AOH) determines a set of compatible opponent action-observation-histories. We refer to this set as *opponent private information* (OPI) which defines a player's uncertainty over both private actions as well as private observations of his opponent. We established that the corresponding infoset for some info-state s_i is the set of all histories compatible with it. In this paper we want to learn a function $histories(S_i) \mapsto \mathcal{P}(\mathcal{H})$ such that $histories(s_i) := I(s_i) = \{h \in \mathcal{H} \mid s_i(h) = s_i\}$. To implement it, there are three ingredients necessary: The first is to encode the AOH and the corresponding OPI, which will be shown in this section. The second is to acquire training data necessary for approximating such a function and the third is to create and train a model which is able to learn such a function in a domain-independent manner.

In this section, we will present a way of encoding both AOH and OPI in a domain-independent way using FOSG.

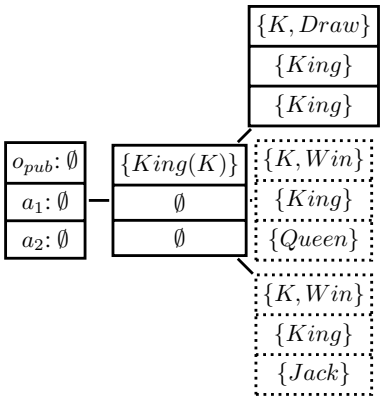


Figure 1: We show a FOSG representation of world states of the card game described in the text. Each node is divided in three parts. The top part denotes the public observation, middle part player one's private action and the lower part player two's private action. The root state contains the public observation that the referee announced the King. We display only the states where player one privately bet "King". The two states where player one won are in the same information set (dotted).

We will first show how this encoding can convey if opponent information is possible, given a player's observations. Furthermore, we will extend this encoding such that it can be used to reflect how likely some opponent information is, assuming a strategy profile σ. We need this representation to efficiently search for plausible opponent actions and private observations.

Encoding a Turn Assuming all information in a FOSG gets revealed once some player observes it, a world state can be uniquely identified by the sequences of public and both players' private information. To make notation easier to understand, we will assume a game with only public observations throughout the coming sections. However, our framework supports private observations as well by, instead of considering each opponent action a, representing the opponent information by pairs of $(a_{-i}, O_{\text{priv(-i)}})$ given by the cartesian product of the opponent's private observations and actions. We will use the term *turn* to refer to each pair of $(a_p, O_{\text{pub}}) \in A_p \times O_{\text{pub}}$, as in "the player did something" and received an observation [1]. We will now describe how these sequences of actions and observations can be encoded using one-hot encoding. We will reuse the example in Figure 1. After the first round of this game, there are three possible public observations that can occur (Loss,Draw,Win). To identify them, we can just enumerate the possibilities and assign indices to each outcome. Now, we are able to encode the public observation by allocating a vector of zeroes with length 3 and assigning 1 to the position which corresponds to the received public observation, i.e., $O_{\text{pub}} = \{Win\} \rightarrow [0, 0, 1]$. Analogously, we consider all possible actions and perform the same for the private information part, i.e., $a_1 = \{King\} \rightarrow [0, 0, 1]$ (Jack,Queen,King). Concatenating the two, we get one vector containing the encoded AOH of one player.

[1]The term "turn" is generally used to describe "full" events happening in a game., i.e. a public state with both players info states

Encoding an Action-Observation History Viewing an AOH as a sequence of pairs of public and private observation, we can perform the above described procedure for every element in the AOH. Doing so encodes a full AOH for one player. Using the vectorized representation the full action-observation history hence becomes a sequence, where each element contains a pair of public and private information.

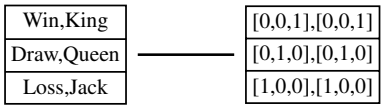


Figure 2: Encoding of a full AOH

Encoding Opponent's Private Information We established how we can encode one player's action-observation-history, where the player knows exactly what he did and observed. We now want to encode his knowledge about what the opponent might have done or observed. Since an information set can have more than one history, we extend our representation by allowing multiple entries of 1 per turn. Figure 3 shows how to encode the opponent's private information in the "Win" public state from player one's perspective.

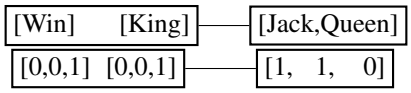


Figure 3: Encoding the player's AOH (left) and the corresponding opponent's private information for the "win" public state (on the right). Since from the player's perspective the opponent could have played both Jack and Queen we assign a 1 for each of them, however the opponent could not have played King.

Encoding Sampled Opponent's Private Information The above encoding reflects if a certain piece of OPI is *possible* (0 or 1) with an AOH, however we can also use it to represent a sampled subset of the infoset. Also, under some strategy σ, a certain history might be more likely than others. We will now show how the above encoding can be further extended to reflect this. Let σ denote a strategy. Let $s_1 := (O_{\text{pub}}(King), a_1(King), O_{\text{pub}}(Win))$ (from the above example). Assume we tried to reach s_1 by choosing actions according to σ from the root. Every time we hit s_1 either by $a_2(Jack)$ or $a_2(Queen)$, we track their occurrence. After 10 samples, we can normalize over the occurrences and get an intuition of how "likely" they are, assuming σ.

Figure 4 shows an example where $a_2(Jack)$ and $a_2(Queen)$ have both been observed 5 times.

$$\boxed{[5 \times Jack, 5 \times Queen, 0 \times King]}$$
$$\boxed{[0.5, \quad 0.5, \quad 0]}$$

Figure 4: Encoding sampled opponent private information for the 'Win' (P2 loss) public state. We sample 10 times into s_1 using σ. We receive $5xJack$ and $5xQueen$. We normalize over the number of occurrences and receive 0.5 each.

This encoding is lossy since it does not directly enable for an exact recovery of the original information set. However, encoding private information for each turn individually allows us to represent opponent private information which is possibly exponential in the number of actions. Furthermore, the encoding shown in this section is just an example implementation of a general encoding procedure. In the remainder of the sections, we will assume a general encoding function which encodes the player's action-observation history as Π_{in} and the opponent's private information as Π_{out}.

Learning to Guess Opponent's Information

We introduced the notion of opponent private information for a given player's information set and showed how to encode both an information set's AOH as well as OPI. Now we will define the function which maps from AOH to OPI. Assume a FOSG G with some deterministic transition function and a chance player. Let s_i denote the information-state of player i with $I_i(s_i)$ denoting its information set. The goal of the paper is to implement an approximation of the function:

$$hist : \bar{s}_i \in S_i \mapsto I_i(\bar{s}_i) = \{h \in H : s_i(h) = \bar{s}_i\} \quad (1)$$

This is usually not provided in a game definition and expensive to get in a domain-independent way. In practice, for some info-state $\bar{s}_i$, we want to sample elements of $hist(\bar{s}_i)$. A naive approach would randomly sample actions for both players and chance and compare the received observations with $\bar{s}_i$ and continue the search if compatible, backtrack and choose different actions if otherwise.

We propose *Heuristic Rejection Sampling* (HRS) (Algorithm 1), which receives a heuristic γ which induces an ordering of actions at each decision point of the opponent. In the case where γ is equal for all actions, it coincides with (Parker, Nau, and Subrahmanian 2005), i.e. naive search and takes random actions and compares the received observation with the one in the requested $\bar{s}_i$. If it is compatible, it continues as such. If incompatible, it returns to the parent state and takes the next action. Let $n = \mid \bar{s}_i \mid$ denote the length of $\bar{s}_i$ in turns, where $\bar{s}_i = a_i^1 o_i^1, \dots a_i^n o_i^n$. A full trajectory τ with $t = \mid \tau \mid$ is given by $\tau = a_i^1 a_{-i}^1 o_i^1 \dots a_i^t a_{-i}^t o_i^t$. We will use $\tau \oplus a$ to denote the concatenation of τ and a. By $w(\tau)$ we denote the last world state in τ. We denote $\mathcal{A}_{-i}^{\gamma}$ as the set of opponent actions ordered in decreasing order by γ. We describe HRS in Algorithm 1.

We propose learning the search heuristic γ where:

$$\gamma^{\sigma} : S_i \to \mathbb{R}^{\mathbb{N} \times (\mathcal{O}_{-i} \times \mathcal{A}_{-i})} \quad (2)$$

Algorithm 1 Heuristic Rejection Sampling (HRS)

```
Input: τ                                          {Current Trajectory}
t =| τ |                                          {Current length of τ}
if s_i(τ) = s̄_i then
    output(τ)
else
    (a_i, o_i) = s̄_i[t]                           {(a_i, o_i) in s̄_i at depth t}
    A_{-i} = {a_{-i} ∈ A^γ_{-i}}                  {A^γ_{-i} ordered by γ}
    for a_{-i} ∈ A_{-i} do
        w' ~ T(w(τ), (a_i, a_{-i}))               {Sample w' using a_{-i}}
        o'_i = O_i(w, (a_i, a_{-i}), w')          {Receive o'_i}
        if o'_i = o_i then
            s_i(τ ⊕ (a_i, a_{-i})) ⊏ s̄_i          {is prefix of}
            HRS(τ ⊕ (a_i, a_{-i}, o'_i))          {Recursive call}
```

such that

$$\gamma^{\sigma}(\bar{s}_i)[d, (a_{-i}, o_{-i})] \in [0, 1] \quad (3)$$

refers to $\gamma^{\sigma}(\bar{s}_i)$ indexed[2] at depth d and action-observation pair (a_{-i}, o_{-i}). It denotes the probability that opponent and chance perform action a_{-i} and yielding observation o_i [3] at depth d while playing strategy σ_{-i} and reaching $\bar{s}_i$. Suppose the opponent has actions $a_{-i}^{k=1}, a_{-i}^2 \dots a_{-i}^{K=|\mathcal{A}_{-i}|}$ at depth d then it holds that:

$$\sum_{k=1}^{K} \gamma^{\sigma}(\bar{s}_i)[d, (a_{-i}^k, o_i)] = 1$$

When γ is used in Algorithm 1 to order the set of opponent actions at each turn, a_{-i} are then chosen in a decreasing order, taking actions with the highest values first. If two values are equal, the action is chosen which is higher in the original enumeration.

We will use $\Pi_{in}(s_i)$ to denote the encoding of s_i, where $\Pi_{in}(s_i)[d]$ will refer to turn d. Analogously, we encode the corresponding OPI as $\gamma(s_i) = \Pi_{out}(s_i)$ and index opponent action a_{-i} at depth d by $\Pi_{out}(s_i)[d, (a_{-i}, o_{-i})]$. The mapping is then given by $\Pi_{in} \underset{\gamma}{\to} \Pi_{out}$.

There are several procedures to effectively obtain training data for approximating γ.

Single trajectory sampling STS samples a terminal trajectory $z \in \mathcal{Z}$ by taking random actions a'_i, a'_{-i} and receiving corresponding observations o'_i. At each depth, the current trajectory is encoded such that for each z we generate a training sample Π_{in}, Π_{out} as shown in Algorithm 2.

Proposition 1. *The function which minimizes L_2-loss on samples constructed by Algorithm 2 approximates the presented $\gamma^{\sigma}(\bar{s}_i)[d, (a_{-i}, o_{-i})] \in [0, 1]$.*

Proof sketch. Let $s_1 := O_{\text{pub}}(King), a_1(King), O_{\text{pub}}(Win)$. Its corresponding

[2] From now on, we will use the [] operator to denote indexing into an object.

[3] In our example o_i coincides with o_{-i}

Algorithm 2 Single Trajectory Sampling (STS)

Input: τ {Current Trajectory}
$t =\mid \tau \mid$ {Current length of τ}
while $\tau \notin \mathcal{Z}$ **do**
  $\Pi_{in}[t] = \Pi_{in}(\tau[t])$ {Encode AOH of τ}
  $\Pi_{out}[t] = \Pi_{out}(\tau[t])$ {Encode OPI of τ}
  $a_i, a_{-i} \sim \sigma(\tau[t])$ {Sample new actions}
  $w' \sim T(w(\tau[t]), (a_i, a_{-i}))$ {Receive new world state}
  $o_i = O_i(w, (a_i, a_{-i}), w')$ {Receive observation}
  $STS(\tau \oplus (a_i, a_{-i}, o_i))$ {Recursive call}
Return Π_{in}, Π_{out} {Training sample}

information set contains two histories, one where the opponent plays $a_2(Jack)$ and one where $a_2(Queen)$ under some strategy σ_2. We refer to the encoding of s_1 as $\Pi_{in}(s_1)$ and the encoding of the corresponding opponent information as $\Pi_{out}(s_1)$ with d denoting the depth. We encode the corresponding opponent information for the history containing $a_2(Jack)$ as $\Pi_{out}[d, a_2(Jack)] = 1$ and 0 for the case of $Queen$. Now let $p = \gamma^{\sigma}(s_1)[d, a_2(Jack)]$ denote the probability of $a_2(Jack)$ being in s_1 assuming the opponent plays σ_2. Suppose we sample using Algorithm 2 and hit s_1 N times yielding k pairs of $(\Pi_{in}(s_1), \Pi_{out}(s_1)[d, a_2(Jack)])$. Let Π^n denote the nth sampled pair and let $\hat{\gamma}$ denote a function $\hat{\gamma} : \mathbb{R}^{\mathbb{N} \times (\mathcal{O}_1 \times \mathcal{A}_1)} \mapsto [0, 1]$. We minimize the MSE on each sample given by:

$$\mathcal{L}^N_{\hat{\gamma}} = \frac{1}{N} \sum_{n=1}^{N} (\hat{\gamma}(\Pi^n_{in}) - \Pi^n_{out})^2$$

Since the input to $\hat{\gamma}$ is always Π^n_{in}, but the target is either $[0, 1]$, $\hat{\gamma}(\Pi^n_{in})$ approximates the mean of the N samples which is $\frac{k}{N}$, i.e., the expected value of N Bernoulli coin tosses with k successes. We converge to p if:

$$\lim_{N \to \infty} \hat{\gamma}(\Pi^n_{in})[d, a_2(Jack)] \to p$$

Since we minimize the MSE on each element of Π_{out} individually, this holds for each element of Π_{out}. □

Batch trajectory sampling (BTS) samples a terminal state $z \in \mathcal{Z}$ by taking a trajectory τ with random actions and receiving corresponding observations. However, whenever an unseen info-state s_i is encountered which was reached by $a^1_i, a^1_{-i}, o^1_i, \ldots a^n_i, a^n_{-i}, o^t_i$ ($t =\mid \tau \mid$), we sample trajectories τ^s using a_i, o_i but choosing actions a_{-i} from strategy σ_{-i} at depth $t - 1$. We sample until we hit s_i S times and we compute the average over the S samples $\bar{\Pi}_{out}$. For each such z generate a training sample Π_{in}, Π_{out} such that for every depth $t =\mid \tau \mid$ we generate the input/output pairs as shown in Algorithm 3:

Proposition 2. *The function which minimizes L_2-loss on samples constructed using BTS approximates the presented $\gamma(\bar{s}_i)[d, (a_{-i}, o_{-i})] = [0, 1]$ for each element in Π_{out}.*

Proof sketch. Analogous to STS, the same properties hold for BTS. Instead of sampling $\Pi_{out}(\bar{s_i})[d, a_2(Jack)] = 1$ and 0 otherwise, we can sample $\Pi_{out}(\bar{s_i})[d, a_2(Jack)] \in (0, 1)$. However in each of the N samples, $a_2(Jack) \in (0, 1)$. Hence, over N samples we approximate the average p over all sampled values for $\Pi_{out}(a_2(Jack))$, similar to STS. □

Algorithm 3 Batch Trajectory Sampling (BTS)

Input: τ, S {Current Trajectory, Num Samples/Depth}
$t =\mid \tau \mid$ {Current length of τ}
while $\tau \notin \mathcal{Z}$ **do**
  $(a, o) = \tau[t]$
  $\Pi_{in}[t] = \Pi_{in}(\tau[t])$ {Input sample}
  while $s < S$ **do**
    $a_{-i} \sim \sigma_{-i}(\tau[t-1])$
    $w' \sim T(w(\tau[t-1]), (a_i, a_{-i}))$
    $o'_i = O_i(w, (a_i, a_{-i}), w')$ {Receive o'_i}
    $\tau^s = \tau[t-1] \oplus (a_i, a_{-i}, o'_i)$
    if $o'_i = o_i$ **then**
      $\Pi^{sum}_{out}(\tau)[t] += \Pi_{out}(\tau^s)$ {Add τ^s to the sum}
      $s += 1$ {Increment}
  $\bar{\Pi}_{out}[t] = \frac{\Pi^{sum}_{out}}{S}$ {Output sample}
  $a'_i, a'_{-i} \sim \sigma(\tau[t])$ {Sample actions to extend τ}
  $w' \sim T(w(\tau[t]), (a'_i, a'_{-i}))$ {New world state}
  $o'_i = O_i(w, (a_i, a_{-i}), w')$ {New observation}
  $BTS(\tau \oplus (a'_i, a'_{-i}, o'_i), S)$ {Recursive call}
Return $(\Pi_{in}, \bar{\Pi}_{out})$

Function Approximator Both Algorithms 2 and 3 generate a training sample in the form a pair of sequences. The AOH consists of pairs of (action,observation), i.e., the action-observation history and the OPI of the corresponding opponent private information,i.e., distributions over opponent action-observation histories. A suitable function approximator needs to (1) be able to perform sequence to sequence mapping (2) be able to process variable-length input/output sequences and (3) needs to be able to model dependencies of elements in a sequence. Hence, we chose to approximate the function γ using a LSTM neural network (Hochreiter and Schmidhuber 1997) Figure 5 shows our encoder/decoder sequence to sequence architecture.

Neural Information Set Generation (NISG) Our algorithm NISG represents an extension of Algorithm 1 where $\hat{\gamma}$ is provided by a neural network of the shape shown in Figure 5. $\hat{\gamma}$ predicts the corresponding OPI sequence with orderings for each pair a_{-i}, o_{-i} at each turn. The algorithm then operates just like HRS (Algorithm 1) by choosing opponent actions/ private observations according to $\hat{\gamma}$.

Experiments

We test our algorithm on Imperfect-Information Goofspiel (GS) (Lisý, Lanctot, and Bowling 2015; Lanctot et al. 2009), Stratego (STR) (Ismail 2004) and Phantom Tic-Tac-Toe (PTTT) (Lanctot et al. 2012). We chose these domains to showcase the domain-independent capability of our framework. Goofspiel has private actions and public observations

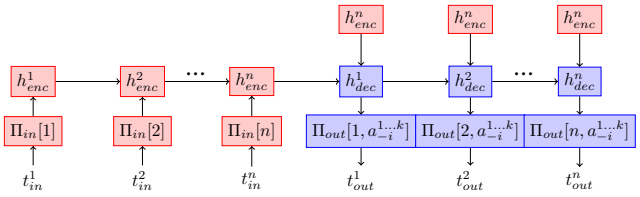


Figure 5: We map a sequence of encoded action-observation pairs to a sequence of distributions over opponent action-observation histories. Each element of the sequence Π_{in} is processed into a hidden state as one timestep t^n_{in} in the red encoder resulting in one hidden state h_n which is then used to construct the full output Π_{out} turn by turn. Each timestep t^n_{out} of the blue decoder has k features given by the number of opponent actions.

with no early terminal states. PTTT has private actions, allows early terminal states, illegal moves, and has sparse public observations early into the game. Stratego has public actions, its imperfect information stems from the initial board setup, it also has sparse public observations and early terminal states. We will examine (1) which algorithm is more suitable to generate training data, (2) which hyperparameter setting to choose for the number of samples in BTS (Algorithm 3) and then show our main experiment in which we perform random trajectories τ from the root to a terminal state, at each info-state in τ, we sample histories using random HRS and NISG within a certain time budget. We track how many nodes have been identified by both algorithms.

Choice of Training Data Algorithm We introduced both single trajectory (STS) and batch trajectory sampling (BTS) to generate training data. We are now interested in how well the function γ can be learned using either STS or BTS data. We trained neural networks on 2000 trajectories of each data type and observe that STS and BTS networks have very similar validation loss on unseen BTS generated data in GS30, while networks trained on BTS data outperforming STS in STR and PTTT. When used in NISG , they are able to find a similar amount of nodes with BTS outperforming STS in deeper games. Due that we decided to show our main results using BTS networks.

Sufficient BTS Samples per Depth Now we are interested in how many samples per depth we should perform when generating training data using BTS which is controlled by the hyperparameter S in Algorithm 3. We ran a sweep over $S = 500, 1000, 2000, 10000$ samples per depth in each domain for randomly chosen information sets. We observe that the distribution in the OPI matrices converges around $500 - 1000$ in all three domains.

Generating States under a Time Budget Our main experiment consists of generating histories in a time budget of 500ms. The reason for that is that it is adjusted to current "thinking times" of online game playing algorithms. If an information set is reached where the agent lacks knowledge, gathering compatible states is an important element, however the agent should be able to allocate as much of the given time budget as possible to compute a strategy.

We perform 2000 test trajectories in each domain. At each depth, we generate states and maintain statistics over how many histories have been found. Note that our testing procedure ensures that trajectories are chosen which have not been used as training data.

NISG used networks trained on just 500 BTS trajectories ($S = 500$). In Figure 6 (top and middle), we report the sum of nodes generated (y-axis) within the time budget of 500ms at each depth (x-axis). The y-axis is furthermore divided by the number of samples which have been performed at each depth (some trajectories terminate early). In the bottom subfigure, we display the percentage of tested information sets at each depth for which no state could be found for GS15,20 and 30[4].

We observe that, while only having been trained on 500 trajectories, NISG is able to outperform random HRS in the number of generated states. Note that not all of the 500 training as well as the 2000 test trajectories reached to the maximum depth displayed for Stratego and PTTT, since they contain early terminal states. In PTTT and Stratego, games with large branching factor and sparse observations, the number of generated states significantly exceeds the baseline. In all GS variants, NISG is able to find nodes for a significantly larger amount of infosets compared to the baseline as shown.

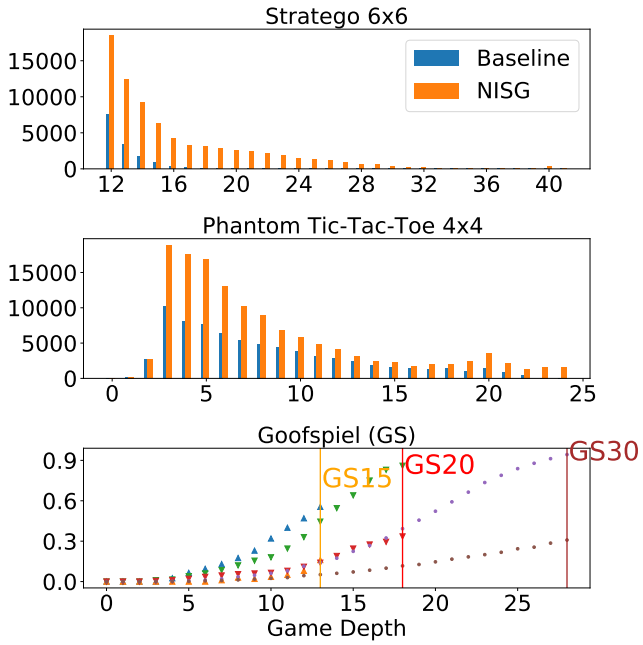
Summary of Other Empirical Results We also performed an analysis on how the amount of training trajectories influences the ability to find states. We observe that in all domains the number of states increases with the number of training trajectories. Despite the small number of training trajectories, the networks are able to generalize well to unseen opponent's private information shown by validation losses close to training losses. While we clarified our motivation to use 500ms time budgets at the start of this section, we did also experiment with higher time budgets and concluded that there were no significant relative changes between the algorithm's performances. Apart from Goofspiel, we deliberately chose two domains (PTTT and Stratego) where a large portion of infosets might contain only very sparse observations. Despite this sparsity, the neural networks are able to learn having only been trained on a small fraction of the tree. We observed (training,validation) MSE losses of $GS30 = (0.001, 0.002)$,$PTTT = (0.02, 0.04)$ and $Stratego = (0.02, 0.03)$.

Conclusion

We addressed the problem of information set generation focused on online game playing where agents only have access to a simulator. We introduced a domain-independent way based on FOSG to encode an agent's action-observation-history and the opponent's private information. We extended

[4]Note that in GS the number of samples per depth is constant since there are no early terminal states

Figure 6: We performed 2000 test trajectories on Stratego (top) and PTTT (middle) with a time budget of 500ms. We generated nodes at each depth with random HRS (blue) and NISG using a neural networks trained on 500 training BTS trajectories (orange). The x-axis denotes the game depth. The y-axis shows the overall sum of nodes divided by the number of samples at each depth which have been performed during the 2000 test trajectories. The bottom plot shows GS15,20,30. The y-axis shows the percentage of infosets at each depth for which no state could be identified within 500ms by each algorithm. The x-axis denotes game depth. NISG in GS15 is shown in yellow with the corresponding HRS result in blue(triangle facing up), for GS20 in NISG is marked red/ HRS green(triangle facing down), and GS30 brown/purple(dots).



this encoding such that it can be used to provide distributions over possible opponent action-observation histories. We provided algorithms to generate training data for learning a function which maps from AOH to OPI. We presented a neural network architecture capable of learning this function. We introduced an algorithm which uses this neural network as a heuristic during search. In our empirical evaluation on three large non-trivial domains, we showed that our algorithm is able to generate significantly more states than the baseline. It is also able to identify nodes in depths where a depth-first search baseline is unable to do so.

References

Brown, N.; Bakhtin, A.; Lerer, A.; and Gong, Q. 2020. Combining deep reinforcement learning and search for imperfect-information games. *arXiv preprint arXiv:2007.13544* .

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, 709–715.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Hsu, F.-H. 2004. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press.

Ismail, M. 2004. *Multi-agent stratego*. Ph.D. thesis, B. Sc thesis, Rotterdam University, The Netherlands.[2].

Kovařík, V.; Schmid, M.; Burch, N.; Bowling, M.; and Lisý, V. 2019. Rethinking Formal Models of Partially Observable Multiagent Decision Making. *arXiv preprint arXiv:1906.11110* .

Lanctot, M.; Burch, N.; Zinkevich, M.; Bowling, M.; and Gibson, R. G. 2012. No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 65–72.

Lanctot, M.; Waugh, K.; Zinkevich, M.; and Bowling, M. 2009. Monte Carlo sampling for regret minimization in extensive games. In *Advances in neural information processing systems*, 1078–1086.

Lisý, V.; Lanctot, M.; and Bowling, M. 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*, 27–36. International Foundation for Autonomous Agents and Multiagent Systems.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356(6337): 508–513.

Morgenstern, O.; and Von Neumann, J. 1953. *Theory of games and economic behavior*. Princeton university press.

Parker, A.; Nau, D.; and Subrahmanian, V. 2005. Game-tree search with combinatorially large belief states. In *IJCAI*, 254–259.

Richards, M.; and Amir, E. 2012. Information set generation in partially observable games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*. Citeseer.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature* 550(7676): 354–359.

Thielscher, M. 2010. A general game description language for incomplete information games. In *AAAI*, volume 10, 994–999. Citeseer.