Approximate Exploitability: Learning a Best Response

Finbarr Timbers\textsuperscript{1}, Nolan Bard\textsuperscript{1}, Edward Lockhart\textsuperscript{1}, Marc Lanctot\textsuperscript{1}, Martin Schmid\textsuperscript{1}, Neil Burch\textsuperscript{1,2}, Julian Schrittweiser\textsuperscript{1}, Thomas Hubert\textsuperscript{1} and Michael Bowling\textsuperscript{1,2}

\textsuperscript{1}DeepMind
\textsuperscript{2}University of Alberta
finbarrtimbers@google.com

Abstract

Researchers have shown that neural networks are vulnerable to adversarial examples and subtle environment changes. The resulting errors can look like blunders to humans, eroding trust in these agents. In prior games research, agent evaluation often focused on the in-practice game outcomes. Such evaluation typically fails to evaluate robustness to worst-case outcomes. Computer poker research has examined how to assess such worst-case performance. Unfortunately, exact computation is infeasible with larger domains, and existing approximations are poker-specific. We introduce ISMCTS-BR, a scalable search-based deep reinforcement learning algorithm for learning a best response to an agent, approximating worst-case performance. We demonstrate the technique in several games against a variety of agents, including several AlphaZero-based agents. Supplementary material is available at https://arxiv.org/abs/2004.09677.

1 Introduction

The increasing incorporation of artificial intelligence into the systems of human life has created a growing impetus to understand the safety and robustness implications of using these algorithms in critical systems. Unexpected failures of AI systems will erode trust, particularly in high consequence settings. Such evaluation typically fails to evaluate robustness to worst-case outcomes. Computer poker research has examined how to assess such worst-case performance. Unfortunately, exact computation is infeasible with larger domains, and existing approximations are poker-specific. We introduce ISMCTS-BR, a scalable search-based deep reinforcement learning algorithm for learning a best response to an agent, approximating worst-case performance. We demonstrate the technique in several games against a variety of agents, including several AlphaZero-based agents. Supplementary material is available at https://arxiv.org/abs/2004.09677.

Prior research in computer poker has attempted to directly evaluate an agent’s performance against a worst-case (i.e. best responding) adversary, as game-theoretic algorithms approximating a Nash equilibrium are standard [Zinkevich et al., 2008] and seek to optimize this (for two-player zero-sum games). Exactly computing a policy’s exploitability – its loss relative to a Nash in the worst-case – was feasible in heads-up limit Texas hold’em poker (HUL) [Johanson et al., 2011], having roughly $10^{14}$ decision points. Unfortunately, other common games are vastly larger, such as heads-up no-limit Texas hold’em (HUNL) (with around $10^{150}$ decision points [Johanson, 2013]) or Go (approximately $10^{170}$ decision points [Allis, 1994]). In these games, computing exploitability or the best response strategy is intractable. However, the local best response (LBR) [Lisý and Bowling, 2016] approximation was used to provide a lower-bound on the exploitability of HUNL strategies. LBR uses explicit knowledge of the distribution over its opponent’s private state.
with a shallow search and simple poker-specific heuristic value function. Though simple, LBR’s reliance on domain knowledge and notable sensitivity to the various choices of abstractions and heuristics make it challenging to use broadly.

We generalize the approach introduced by LBR, introducing a scalable domain-independent alternative that uses deep reinforcement learning (RL) combined with search and knowledge of the opponent’s distribution over private states to learn a value function for the best response in a two-player zero-sum game. By fixing the policy of the agent being evaluated, the environment becomes single-agent from the perspective of the agent, and the optimal policy from RL is exactly a best response [Greenwald et al., 2017]. By exploiting access to the agent’s policy during evaluation, we hope to simplify the learning task, reducing the representational power needed by the network, and improving sample efficiency and accuracy.

Our contributions are as follows:

• In Section 3.2, we introduce our best response algorithm, ISMCTS-BR.

• In Section 4.1, we demonstrate the robustness of ISMCTS-BR across a large number of games & policies by showing that it is able to closely approximate the exact best response.

• In Section 4.2, we demonstrate that ISMCTS-BR can exploit strong, search-based agents, and provide evidence that deep reinforcement learning techniques can be vulnerable to distribution shift, highlighting the importance of worst-case performance measures in agent evaluation.

2 Background and Terminology

An extensive-form game is a sequential interaction between players \( i \in \{1, 2, \cdots, n\} \cup \{c\} \), where \( c \) is the chance player, i.e. a player following a static, stochastic policy to define the transition probabilities given states and actions. We use \( -i \) to refer to the non-chance opponent of player \( i \). In this paper, we focus on the two-player zero-sum setting, and in particular, those where beliefs about the opponent’s private state are tractable, i.e. sufficiently small to be represented in memory. The game starts in the root (or empty) history \( h = \emptyset \). On each turn, a player \( i \) (possibly chance) chooses an action \( a \in A_i \), changing the history to \( h' = ha \). The full history is sometimes also called a ground state because it uniquely identifies the true state, as it consists of all actions including those not observed by all players. For example, a history in poker includes all the dealt cards, even private ones, in addition to the sequence of actions taken. We define an information state (or infostate) \( s \in S \) for player \( i \) as the state as perceived by an agent which is consistent with its observations.\(^1\) Formally, each \( s \) is a set of histories, specifically \( h, h' \in s \Leftrightarrow \) the sequence of player \( i \)'s observations along \( h \) and \( h' \) are equal. Informally, this consists of all the histories which the player cannot distinguish given the information available to them. We call \( Z \) the set of terminal histories, which correspond to the end of a game. Each \( z \in Z \) has a utility for each player \( u_i(z) \). Since players cannot observe the ground state \( h \), policies are defined as \( \pi_i : S_i \rightarrow \Delta(A_i)(s) \), where \( \Delta(A_i)(s) \) is the set of probability distributions over the legal actions available to player \( i \) at \( s \), \( A_i(s) \).

For example, in Texas hold’em poker, the root history is when no cards have been dealt and no betting has happened. The history includes any private cards that have been dealt, while the information state for player \( i \) includes the private cards for that player, and the information that the other player has received cards, but not which cards those are. Both the history and the information state include all public cards and all betting actions. A terminal history is a history where either a showdown has occurred or a player has folded. \( u_i(z) \) is then the number of chips that each player has won.

In a perfect information game such as Chess or Go, the history is identical to the information state, since everything is public. As is common in perfect information games, we use an encoding of the current board state (or a stack of recent ones) as input for training, following [Silver et al., 2018]. \( u_i(z) \) is -1 for a loss, 1 for a win, and 0 for a draw when possible (e.g. in Chess). We assume finite games, so every history \( h \) is bounded in length. The expected value of a joint policy \( \pi \) (all players’ policies) for player \( i \) is defined as \( u_i(\pi) = E_{z \sim \pi} [u_i(z)] \), where the terminal histories \( z \in Z \) are composed of actions from the joint policy.

2.1 Optimal Policies

The standard game-theoretic solution concept for optimal policies in two-player zero-sum games is Nash equilibrium, defined as a strategy profile (i.e., joint policy) where none of the players benefits from deviating unilaterally. A strategy profile \( \pi = (\pi_1, \pi_2) \) is a Nash equilibrium iff

\[ \forall \pi'_i, i \in \{1, 2\} : u_i(\pi_i, \pi_{-i}) \geq u_i(\pi'_i, \pi_{-i}) \]  

(1)

In two-player zero-sum games, this concept is equivalent to the minmax solution concept, and the expected value for playing optimally is unique:

\[ v^*_i = \min_{\pi_1} \max_{\pi_2} u_1(\pi_1, \pi_2) = \min_{\pi_1} \max_{\pi_2} u_2(\pi_1, \pi_2). \]  

(2)

The value (2) is referred to as the game value (for player 1), \( v^*_1 \). In two-player zero-sum games, the game value is thus \( v^*_2 = -v^*_1 \). Both solutions concepts thus optimize against the worst-case opponent — the best response. We define the set of player \( i \) best responses to an opponent policy \( \pi_{-i} \),

\[ \text{BR}_i(\pi_{-i}) = \{ \pi_i | u_i(\pi_i, \pi_{-i}) = \max_{\pi'_i} u_i(\pi'_i, \pi_{-i}) \}. \]

For convenience, we refer to elements in this set, best responses to \( \pi_{-i} \), as \( b(\pi_{-i}) \). When a player uses an optimal policy \( \pi^*_i \), their utility is then guaranteed (on expectation) to achieve at least the game values: \( \forall i, \pi_{-i} : u_i(\pi^*_i, \pi_{-i}) \geq v^*_i \).

2.2 Exploitatibility

For suboptimal policies \( \pi \), player \( i \)'s incentive to deviate is:

\[ \delta_i(\pi) = u_i(b(\pi_{-i}), \pi_{-i}) - u_i(\pi). \]
One metric to quantify the distance from optimal is \( \text{NASHCONV}(\pi) = \sum_i \delta_i(\pi) \) and EXPLOITABILITY\( (\pi) = \text{NASHCONV}(\pi)/n \). In zero-sum games, when summing over players, the second terms sum to zero, so NASHCONV simplifies to \( \sum_i u_i(b(\pi_{-i}), \pi_{-i}) \). Furthermore, the \( \epsilon \)-minimax (or \( \epsilon \)-Nash equilibrium) policy is one where \( \max_i \delta_i(\pi) \leq \epsilon \).

Exploitation measures how well a strategy profile approximates a Nash equilibrium — the closer it is to zero, the closer the policy is to optimal. Unfortunately, exploitability can be difficult to compute exactly: in the worst case, it requires traversing the full game tree, making it intractable in large games. However, by approximating exploitability, we can address tractability while still assessing worst-case performance.

### 3 Approximate Best Response

By fixing the policy of one agent, the environment becomes a (stochastic) single agent environment where one can use standard reinforcement learning methods to learn a best response, as they are optimal policies in that environment [Greenwald et al., 2017]. The fixed agent is exploitable by the reward obtained by the optimal policy.

While exploitability is defined using an exact best-responding policy \( b(\pi_{-i}) \), we can approximate this policy and define a corresponding metric, the approximate best response (ABR\( \pi_{-i} \)), with the corresponding APPROXIMATE-NASHCONV (ANC) as:

\[
\text{ANC}(\pi) = \sum_i u_i(\text{ABR}(\pi_{-i}), \pi_{-i})
\]

We define approximate exploitability similarly. Since it uses an approximate best response, the approximate exploitability is a lower bound on the true value, with equality when the approximation matches the exact best response.

#### 3.1 Local Best Response

The poker community has used local best response (LBR) [Lišý and Bowling, 2016] as a proxy for the full best response [Moravčík et al., 2017; Brown et al., 2018]. LBR approximates the best response by performing a local search aided by explicit knowledge of the distribution over its opponent's private state and a value function, typically a hand-crafted heuristic function. LBR is fast to run and easy to understand; it can often be implemented in a small amount of code.

However, the results LBR obtains can be sensitive to the specific experimental setup. For example, when running against the same fixed policy using the same heuristic value approximator in action-abstracted no-limit poker, the values can vary wildly depending on the choice of action abstraction (e.g. from -867 mbb/h to 46 mbb/h [Gruslys et al., 2020] and from 496 mbb/h to 3763 mbb/h [Lišý and Bowling, 2016]). Additionally, while the check-call value function has had much empirical success in poker, it is not obvious what a strong choice for value function is in other games.

#### 3.2 IS-MCTS Best Response Search

LBR performs a depth-limited tree search, using a value function to truncate the search. This is similar to MCTS, which has been shown to perform remarkably well with a learned value function [Silver et al., 2018]. A natural extension to LBR then is to combine it with MCTS. Since we want an algorithm that works on imperfect information games, we use a variant of MCTS, information set MCTS (IS-MCTS) [Cowl- ing et al., 2010]. However, IS-MCTS does not sample from the belief distribution when searching at an information state. While empirically this works to exploit basic agents, this does not converge to the exact best response in the limit [Lišý et al., 2015].

Instead, we calculate the exact posterior distribution over histories with the same information state given the opponent’s policy, and use that to sample from opponent information states. To the best of our knowledge, ours is the first algorithm to apply IS-MCTS with the exact posterior distribution over opponent private states and PUCB.

We call this algorithm ISMCTS-BR and present it in Algorithm 1. GETNODE retrieves the node from the search tree corresponding to the information state \( s \). We abuse notation and let GETNODE operate on histories \( h \) as well, in which case GETNODE retrieves the node corresponding to the information state for the history from the current player’s perspective. SAMPLEHISTORYFROMINFOSTATE samples a history \( h \) (i.e. private information for the opponent) from the exact belief distribution induced by the opponent’s policy. In imperfect information games with perfect recall, the belief \( \text{Pr}(h|s) \) is a function of the opponent’s policy and can be obtained by applying Bayes’ rule given the reach probabilities of each history (see [Srinivasan et al., 2018, Section 3.2]). \( T \) and \( r \) are used to updated the search tree before proceeding to the next simulation. UPDATESEARCHTREE and CHOOSEACTION follow the standard MCTS algorithm [Silver et al., 2018]; an IS-MCTS implementation can be found in [Lanctot et al., 2019].

Running ISMCTS-BR requires having access to the opponent policy, as we query the opponent policy during Alg. 2 when the opponent would act. However, in practice, this is an easy restriction to relax, as replacing the exact opponent policy with a learned policy has been shown to work remarkably well [Silver et al., 2018].

---

In perfect information games, this is equivalent to MCTS.
We use an experimental setup identical to [Silver et al., 2018] except where otherwise noted. We train for only 100k learning steps rather than 800k steps. We use a distributed actor/learner setup to train our neural network. Each experiment requires roughly 128 Cloud TPUv4 chips for the actors, and 4 TPUv2 chips for the learners. The TPU version choice was arbitrary and based on internal availability at the time we ran our experiments. The results we report, unless otherwise indicated, are the average results over the last five network updates in the run; as the network is updated every 500 minibatch steps, and each minibatch uses 2048 examples, this is roughly 5 million states of data. On iteration $t$, network parameters $\theta_t$ are updated using a loss function that combines the mean-squared error between predicted expected reward $v_t$ and the Monte Carlo return for the episode $z_t$, the cross-entropy loss between the prior policy predicted by the network $p_{\pi}$ and the policy induced by the normalized visit counts during search $\pi_t$, and $\ell_2$ regularization:

$$f_\theta(h) = p_{\pi}(s),$$  
$$l_t = (z_t - v_t)^2 - \pi^T_t \log p_{\pi} + c\|\theta_t\|^2_2$$  
$$\theta_t = \text{GradDescent}(\theta_{t-1}, \alpha_t, \ell_t)$$

Since the reward and prior are learned for information states, not histories, the network learns an average across histories, allowing the agent to act according to the expected value of information state $s$.

We present results in imperfect information games, large perfect information games, Scotland Yard (a classic pursuit-evasion game on a graph), and two large variants of Texas hold’em poker: heads-up limit (HUL) and heads-up no-limit (HUNL). As these are widely-used domains, we provide descriptions in the appendix. Opponents range from basic and easy to exploit (e.g. uniform random) to agents that use search and are hard to exploit (e.g. AlphaZero).

### 4.1 Validation

We begin by validating ISMCTS-BR with policies & domains that are simple enough to allow us to compute the exact exploitability.

#### vs. Uniform Random

We evaluate ISMCTS-BR versus uniform random in numerous games. Table 1 shows that ISMCTS-BR consistently performs well, achieving $\geq95\%$ of achievable reward in most cases. $^4$ GoofspielN refers to a variant of Goofspiel with $N$ cards. SY refers to Scotland Yard. “Glasses”, “War Museum”, “Queen Mary’s Garden” and “Kennington Double” refer to custom graphs of Scotland Yard that are small enough to compute exact NashConv (see the appendix for the specific graphs).

![Algorithm 2: SIMULATION from Alg. 1.](image)

```plaintext
input : $h$ — root history for simulation
input : $\pi_{-i}$ — fixed policy for opponent
input : $f_\theta$ — policy & value function
input : $T$ — hashmap with information per infostate
input : $f_\theta$ — value and policy network
output: $a \in \mathcal{A}_i$ — action for the current infostate
1 Initialize $T$.

2 while $h$ is not terminal OR $h$ is not in $T$ do
3     $N = \text{GETNODE}(h, T)$
4     Add $N$ to $T$.
5     if $i$ is to act then
6         $a = \text{PUCB}(N)$.
7     else
8         $a = \pi_{-i}(h)$
9     end
10    $h = ha$
11 end
12 if $h$ is terminal then
13     $r$ is the terminal value from $h$
14 else
15     Add $h$ to $T$.
16     $r = \text{value from } f_\theta(h)$
17 end
18 return $(T, r)$;
```

Table 1: ISMCTS-BR ANC (\%) vs UniformRandom.

<table>
<thead>
<tr>
<th>Game</th>
<th>Unif. Rand.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leduc Poker</td>
<td>98%</td>
</tr>
<tr>
<td>Goofspiel4</td>
<td>97%</td>
</tr>
<tr>
<td>Goofspiel5</td>
<td>95%</td>
</tr>
<tr>
<td>Goofspiel6</td>
<td>97%</td>
</tr>
<tr>
<td>HUL.</td>
<td>90% $^3$</td>
</tr>
<tr>
<td>SY - Glasses</td>
<td>99%</td>
</tr>
<tr>
<td>SY - War Museum</td>
<td>96%</td>
</tr>
<tr>
<td>SY - Queen Mary’s Garden</td>
<td>98%</td>
</tr>
<tr>
<td>SY - Kennington Double</td>
<td>99%</td>
</tr>
<tr>
<td>Full Scotland Yard</td>
<td>99% $^4$</td>
</tr>
<tr>
<td>Chess</td>
<td>90% $^4$</td>
</tr>
<tr>
<td>Go</td>
<td>100% $^4$</td>
</tr>
<tr>
<td>Connect 4</td>
<td>100%</td>
</tr>
</tbody>
</table>

3[Johanson, 2011]
4We assume that an exact BR to uniform wins 100% in full Scotland Yard, Chess, and Go, as this value is intractable to compute.
percentage of the value achieved by an exact best response. Results are the mean over 200,000 games, and all have a standard error less than 30 mbb/h.

<table>
<thead>
<tr>
<th></th>
<th>ABR</th>
<th>LBR</th>
<th>NashConv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fold</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
</tr>
<tr>
<td>Call</td>
<td>2270</td>
<td>2310</td>
<td>2330</td>
</tr>
<tr>
<td>Random</td>
<td>7960</td>
<td>5690</td>
<td>8800</td>
</tr>
</tbody>
</table>

Table 3: Comparison of (approximate) NashConv values for HUL chumps. Units are in mbb/h. ANC is ISMCTS-BR’s Approximate-NashConv.

**Table 2: ABR vs poker chumps, percentage of the exact BR value achieved. HUNL(fcpa) limits actions to fold, call, pot bet, and all-in.**

<table>
<thead>
<tr>
<th>Policy</th>
<th>Fold</th>
<th>Raise</th>
<th>(\frac{1}{2}) Call</th>
<th>(\frac{1}{2}) Raise</th>
<th>Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leduc Poker</td>
<td>100%</td>
<td>99%</td>
<td>100%</td>
<td>99%</td>
<td></td>
</tr>
<tr>
<td>HUL</td>
<td>100%</td>
<td>89%</td>
<td>94%</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>HUNL(fcpa)</td>
<td>100%</td>
<td>(\geq 77% )(^5)</td>
<td>(\geq 95% )(^5)</td>
<td>99%</td>
<td></td>
</tr>
</tbody>
</table>

**4.2 Approximate Exploitability in Go**

We use ISMCTS-BR to evaluate two agents trained following [Silver et al., 2018]: one as described in the paper (Alp-Slumbot’s author, Eric Jackson, provided data files for Slumbot 2016 that were used with up-to-date code and a new configuration believed best matched Slumbot 2016.

Results in Table 4 show that ISMCTS-BR can exploit these more sophisticated policies. Against CFR+ with more than 10 iterations, ISMCTS-BR also yields exploitability close to the exact value. However, as the exploitability approaches zero with more iterations, small absolute variations cause large percentage variations.

Nash approximations also provide an opportunity to demonstrate the richness of evaluation enabled by learned best responses. We trained ISMCTS-BR in Leduc poker against CFR+ with 7 iterations and always fold, which have exact exploitabilities of 1.01 and 1, respectively. If practitioners only consider the exact exploitability, or the final exploitability produced by ABR, these policies would appear similar. However, Figure 1 shows that CFR+7 is more difficult to exploit — reflecting the strategy’s relative complexity.

Finally, we ran ISMCTS-BR vs Slumbot [Jackson, 2013], a strong agent in heads-up no-limit Texas hold’em. We evaluate a best effort reproduction of Slumbot 2016 using ISMCTS-BR and compare performance to prior LBR results for the original Slumbot 2016 [Lisý and Bowling, 2016]. ISMCTS-BR managed to exploit Slumbot by 1259 mbb/h when ABR acted using the FCPA betting abstraction. This outperforms the prior LBR results for responses that were not forced to check/call through the first two rounds, which won 522 and 763 mbb/h using the fold/call and 56 bets action abstractions, respectively. By constraining LBR’s actions per round — limiting it to check/call until the third round — LBR was able to win 4020 and 3763 mbb/h using the fold/call/pot/all-in and 56 bets action abstractions. While ISMCTS-BR uses the fcpa abstraction, we have not restricted it per round. Prior LBR results do not evaluate the fcpa abstraction without restrictions, but the contrast in performance when using the 56 bets abstraction is considerable, both against Slumbot and the other agents that were evaluated, suggesting that this domain-specific hand-tuning was important. The generality of ISMCTS-BR allows it to find a substantial degree of exploitability, even in high-quality agents, across disparate domains and despite a lack of such domain knowledge.

| ISMCTS-BR vs CFR+ | 99.9% | 95.7% | 99.5% | 94.9% | 99.9% | 97.6% |

Table 4: Approx. NashConv (% of exact NashConv) for ISMCTS-BR vs CFR+ with 10 iters.

---

\(^5\)We ran the LBR results using an implementation of [Lisý and Bowling, 2016]. Results are the mean over 200,000 games, and all have a standard error less than 30 mbb/h.

\(^{\text{a}}\)Leduc poker is small making exact exploitability simple, while values for HUL come from [Johanson et al., 2011, Table 1]. In HUNL, the prior LBR work [Lisý and Bowling, 2016] argues that always call has an exploitability of approximately 50 bb/h (big blinds per hand). We use this value for all chumps except always fold, as the same argument holds as an upper-bound on exact exploitability, likely causing an underestimate of ISMCTS-BR’s effectiveness.

---

vs. Poker Chumps. Simple poker policies called “chumps” have been evaluated in prior work. These policies either always fold, always call, always raise, or mix between call and raise with equal probability. Table 2 presents the results as a percentage of the value achieved by an exact best response\(^5\). In most cases, the reward achieved by ISMCTS-BR quickly converges to the same value as the exact best response. Practitioners should note that this evaluation was particularly helpful in developing ISMCTS-BR as it enabled testing on a range of behaviour at different game scales using exact exploitability values that were generally known.

We can compare LBR with ISMCTS-BR and the exact BR against chump policies in poker. Table 3 shows that ISMCTS-BR is able to improve upon LBR in HUL, except for the case of always call, where LBR’s value function exactly matches the opponent’s behaviour. LBR is notably weaker against uniform random. Results in HUNL echo this, with ISMCTS-BR performing similarly against always call (49.7 bb/h instead of LBR’s 49.0), but performing better than the best LBR result reported against the call/raise chump in [Lisý and Bowling, 2016]: 47.9 bb/h instead of LBR’s 24.4. This is a shortcoming of LBR: the specific choice of heuristic introduces bias, and it can fail to optimally exploit weak agents.\(^6\)

vs. Nash Approximations. One limitation of the previous opponents is that they behave the same regardless of the game’s state. To address this, we examine ISMCTS-BR in several small imperfect information games against Nash approximations generated with 10 iterations of CFR+ [Tammelin et al., 2015], ensuring the policy remains exploitable.

---

\(^{\text{a}}\)The specific choice of heuristic introduces bias, and it can fail to optimally exploit weak agents.

---

vs. Nash Approximations. One limitation of the previous opponents is that they behave the same regardless of the game’s state. To address this, we examine ISMCTS-BR in several small imperfect information games against Nash approximations generated with 10 iterations of CFR+ [Tammelin et al., 2015], ensuring the policy remains exploitable.
Figure 1: ISMCTS-BR training curves in Leduc poker vs. always fold and CFR+ with 7 iterations.

...) haZero MSE), and another using the 25-step TD error to train the value function (AlphaZero TD25). During the ISMCTS-BR search, we use the AlphaZero policy head as the opponent policy, rather than the full MCTS algorithm. When AlphaZero is selecting actions during play, it acts as normal, and uses 800 simulations.

In head-to-head play, AlphaZero MSE wins 23.25% of games vs AlphaZero TD25, indicating a relative Elo of -207. AlphaZero MSE was found to be less exploitable, with ISMCTS-BR at the end of training winning roughly 20% of games instead of the 40% against AlphaZero TD25 (Fig 2).

As AlphaZero is a MCTS agent with a variable number of simulations, we can create a curriculum — training ISMCTS-BR against AlphaZero with 1 simulation, then warm-starting the network and train it against AlphaZero with 10 simulations, then 50, 100, 200, 400, and 800. (Fig. 3). The curriculum improved the exploitability estimate significantly, returning results around 90%.

Figure 2: ISMCTS-BR results vs AlphaZero agents in Go.

Figure 3: ISMCTS-BR simulation curriculum results in Go against AlphaZero agents.

glect the agent’s worst-case performance, potentially deploying an agent that is vulnerable to failure in the wild. Head-to-head performance and exploitability are complementary, and practitioners much decide how practical worst-case scenarios are. Perhaps limited access to the agent makes it unrealistic for adversaries to probe its behaviour, allowing for more focus on the in-practice head-to-head performance.

5 Conclusion

We introduce ISMCTS-BR, an algorithm for learning an approximate best response that generalizes the poker specific local best response method, and validate it across a range of domains and agents. In addition to a lower-bound on the true exploitability, a learned best response provides both an assessment of how difficult it is to discover an agent’s flaws, and a mechanism to facilitate debugging by identifying game trajectories where the agent is weak. We believe this technique will support evaluation in large games and drive algorithmic improvements for robust agents, as prior work spurred computer poker research.

Although ISMCTS-BR is limited to tractable belief spaces, this could potentially be relaxed by using a sample model to generate actions for the opponent during search or replacing search with a stochastic planning algorithm such as [Ozair et al., 2021] as the best responder.

Acknowledgements

We would like to thank Eric Jackson for helping us evaluate against Slumbot, Kevin Waugh for repeatedly finding & fixing bugs in our code, and Josh Davidson for providing the Scotland Yard images.

References


