Approximate exploitability: Learning a best response in large games

Finbarr Timbers
DeepMind
Edmonton, Alberta, Canada
finbarrtimbers@google.com

Edward Lockhart
DeepMind
London, United Kingdom
locked@google.com

Martin Schmid
DeepMind
Edmonton, Alberta, Canada
mschmid@google.com

Marc Lanctot
DeepMind
Edmonton, Alberta, Canada
lanctot@google.com

Michael Bowling
DeepMind
Edmonton, Alberta, Canada
bowlingm@google.com

ABSTRACT

A common metric in games of imperfect information is exploitability, i.e. the performance of a policy against the worst-case opponent. This metric has many nice properties, but is intractable to compute in large games as it requires a full search of the game tree to calculate a best response to the given policy. We introduce a new metric, approximate exploitability, that calculates an analogous metric to exploitability using an *approximate* best response. This method scales to large games with tractable belief spaces. We focus only on the two-player, zero-sum case. Additionally, we provide empirical results for a specific instance of the method, demonstrating that it can effectively exploit agents in large games. We demonstrate that our method converges to exploitability in the tabular setting and the function approximation setting for small games, and demonstrate that it can consistently find exploits for weak policies in large games, showing results on Chess, Go, Heads-up No Limit Texas Hold'em, and other games.

KEYWORDS

Imperfect information games; online search; Nash equilibrium; Monte Carlo tree search; evaluation

1 INTRODUCTION

Artificial intelligence research has long used games as a venue for progress, including results in Checkers [19], Chess ([7]), Jeopardy [9], Poker [17], [5], and Go [20], among other results. In particular, two-player, zero-sum games have been the focus of a lot of research, and been benchmarks for research. We focus on that case in this paper. In perfect information games, head-to-head play has commonly been used as the metric; if two agents play some number of games against each other, and one beats the other, than the winning agent is assumed to be "superior" to the losing agent, for some definition of superior.

However, this only provides a relative ranking, and the order of the ranking can be affected by a variety of small details, including, for instance, the stack size used in the Alberta Computer Poker Competition (ACPC) [3]. It also suffers from issues when agents performance are not strictly transitive [2, 18]. Absolute metrics, such as **exploitability**, which measures the distance from a Nash Equilibrium, require a large amount of engineering effort to calculate in large games [13]. Exploitability provides guarantees for the worse-case performance of agents, however, while it is (barely) feasible for Heads-up Limit Texas Hold'em (HULH), having roughly 10^{14} decision points, other games are much, much larger, such as Heads-up No Limit Texas Hold'em (HUNL) (with around 10^{160} decision points [12]) or Go (approximately 10^{170} decision points [1]). In these games, calculating exploitability is intractable, and other metrics must be used. In Go, and other games of perfect information, Elo is commonly used, which provides a relative ranking.

As computing the best response strategy is intractable, the computer poker community has recently opted to approximate it by **local best response** (LBR) [16]. LBR combines a small search with a simple poker-specific heuristic value function. While LBR has proven to work surprisingly well in poker, it relies on a hand-designed poker heuristic.

An obvious extension is to learn a policy that can exploit the agent we wish to evaluate. Fixing the agent's policy makes the environment a single agent one [4]. One can thus use standard reinforcement learning (RL) algorithms to compute a best response [10]. Additionally, learning the best response has another, potentially very appealing, benefit. Exploitability itself does not tell the full story about the strength of the agent. A strong chess agent that can be beaten by a particularly clever line of play is a better chess player than an agent that always resigns. Yet, both of these agents technically have the same exploitability, i.e. 1. Since we are learning the best response, one would expect that our method can indicate whether an exploiting agent is "strong" or "weak".

We present a large scale implementation of this framework, where we learn an (approximate) best response in large games of perfect and imperfect information. Our experimental section includes small games where the exact solution is tractable, allowing to compare the resulting learned best response to the exact best response. We also present experimental results in large games of perfect and imperfect information.

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 every player other than i and the chance player. In this paper, we focus on the 2 player setting, and in particular, the two-player, zero-sum setting 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 chooses an action $a \in \mathcal{A}_i$, changing the history to $h' = ha$.[1] Here h is called a prefix history of h', denoted $h \sqsubset h'$. The full history is sometimes also called a *ground* state because it uniquely identifies the true state, since chance's actions are included. In poker, a history includes all the players' private cards in addition to the sequence of actions taken. We define an **information state** $s \in \mathcal{S}$ for player i as the state as perceived by an agent which is consistent with its observations. 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 from each other given the information available to them. In poker, an information state groups together all the histories that differ only in the private cards of $-i$.

We call $\mathcal{Z}$ the set of terminal histories, which correspond to the end of a game. Each $z \in \mathcal{Z}$ has a utility for each player $u_i(z)$. We also define $\tau(s)$ as the player who is choosing an action at s, and use $\mathcal{Z}(h)$ to denote the subset of terminal histories that share h as a prefix. Since players cannot observe the ground state h, policies are defined as $\pi_i : \mathcal{S}_i \rightarrow \Delta(\mathcal{A}_i)(s)$, where $\Delta(\mathcal{A}_i)(s)$ is the set of probability distributions over $\mathcal{A}_i(s)$.

As an example, in Texas No-limit Hold'em Poker, the root history is when no cards have been dealt and no betting has happened. The history includes the 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 consists of the state when either a showdown has occurred or a player has folded. The $u_i(z)$ 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. $u_i(z)$ is either -1 for a loss, +1 for a win, and 0 for a draw when one is possible (e.g. in Chess).

We assume finite games, so every history h is bounded in length. The expected value of a joint policy $\bar{\pi}$ (all players' policies) for player i is defined as

$$v_{i,\bar{\pi}} = \mathbb{E}_{z\sim\bar{\pi}}[u_i(z)], \tag{1}$$

where the terminal histories $z \in \mathcal{Z}$ are composed of actions drawn from the joint policy. We assume that players have **perfect recall**, i.e. they do not forget anything they have observed while playing. Under perfect recall, we can obtain the distribution of the states using Bayes' rule (see [21, Section 3.2]).

2.1 Optimal Policies

The standard game-theoretical solution concept for optimal policies is **Nash equlibrium**, defined as a strategy profile where none of the players benefits from deviating.

$$\forall \pi'_i :\ v_i(\pi_i, \pi_{-i}) \geq v_1(\pi'_i, \pi_{-i}) \tag{2}$$

[1]Note that we include c, chance, as a player, so this will include all chance actions, e.g. all cards (both public and private) that are dealt.

In two player zero sum games, this concept is equivalent to the **minmax** solution concept (this is true for both perfect and imperfect information games).

$$\max_{\pi_1}\min_{\pi_2} v_1(\pi_1, \pi_2) = \min_{\pi_2}\max_{\pi_1} v_1(\pi_1, \pi_2) = gv \tag{3}$$

The unique value of (3) is referred to as the **game value** - gv. Both solutions concepts thus optimize against the worst case opponent — the **best response**. The best response to a policy π_i is defined as $b(\pi_i) = \text{argmin}_{\pi_{-i}} v_i(\pi_i, \pi_{-i})$. The optimal policy π^*_i is then guaranteed (on expectation) to receive at least the game value $\forall \pi'_{-i}\ v_i(\pi^*_i, \pi_{-i}) \geq gv_i$.

2.2 Exploitability

For suboptimal policies, the worst-case value is strictly less than the game value: $\delta_i(\pi_i) = gv_i - v_i(\pi_i, b(\pi_i))$. A common measure then is $\text{NashConv}(\pi) = \sum_i \delta_i(\pi)$ and exploitability $= \frac{\text{NashConv}}{|\mathcal{N}|}$. Furthermore, the ϵ-minmax (or ϵ-Nash equilibrium) policy is one where $\max_i \delta_i(\pi) \leq \epsilon$.

While computing the game value is in general as hard as computing an optimal policy, one can compute the exploitability of a policy profile without access to the game value. Simple algebra reveals that NashConv (where the last operation follows since $gv_1 = -gv_2$):

$$\sum_i \delta_i(\pi) = \sum_i gv_i - v_i(\pi_i, b(\pi_i)) = \sum_i v_i(\pi_i, b(\pi_i)) \tag{4}$$

Exploitability thus measures the quality of a strategy profile — the lower the exploitability, the closer to the optimal policy. However, exploitability is difficult to calculate in large games, as it requires a full tree traversal. This is clearly intractable in games such as HUNL poker, with approximately 10^{160} decision points - more than the number of atoms in the universe.

3 APPROXIMATE BEST RESPONSE

By fixing the policy of one agent, the environment becomes a (stochastic) single agent environment [4]. The best response is then exactly the optimal policy in that environment, and the exploitability is then exactly the optimal reward the agent can achieve in this environment.

$$\sum_i v_i(\pi_i, b(\pi_i)) \tag{5}$$

As this is a single agent environment, one can opt for standard reinforcement learning methods to learn this optimal best-responding policy [10]. And while exploitability is defined using the optimal best-responding policy $b(\pi_i)$, we are learning to approximate this policy, and can thus define a corresponding metric. We denote the resulting approximate best response as $abr_i(\pi_i)$, and define the corresponding approximate exploitabilty as:

$$\sum_i v_i(\pi_i, abr_i(\pi_i)) \tag{6}$$

Approximate exploitability is thus exploitability with an approximate best response rather than the exact best response; similarly, we have ApproximateNashConv (ANC).

3.1 Local best response

Rather than learning the best response, recently the poker community has used local best response (LBR) [16] as an approximate best responder [6, 17]. LBR approximates best response by performing a local search aided with a poker specific value function. This has proven to be a surprisingly good indicator of an agent's performance in poker, but it relies on a game specific hand-crafted heuristic evaluation. Furthermore, our experiments suggest that LBR does a poor job approximating a best response in small poker variants. Note that when we discuss "strength" here, we implicitly use a ranking from people familiar with the field. The "strength" that we discuss is almost identical to exploitability; a "strong" agent would be one that is hard to exploit, whereas a weak agent would be open to exploits. In practice, it's impossible to calculate exploitability in large games, so we rely on fallible approximations, such as whether or not an agent can be beat by humans (professional or otherwise), or how an agent performs against techniques that have some theoretical guarantees. For instance, DeepStack [17] or Libratus [5] would be considered "strong" agents as they have both beaten professional human players and have theoretical guarantees for convergence to a Nash Equilibrium, while UniformRandom is considered a "weak" agent as it can be trivially beaten by an amateur human player. The authors recognize that this is somewhat hand-wavy and unsatisfactory, and hope that this paper will help formalize this notion of strength and provide a more consistent notion of strength.

3.2 IS-MCTS ABR

We now describe a particular instance of ABR where we learn the approximate best-response.

At an abstract level, LBR is a method which performs a depth-limited tree search, using a value function to truncate the search. This sounds remarkably similar to MCTS, which has been shown to perform remarkably well with a learned value function [20]. As such, a natural extension to LBR is to combine it with MCTS. We call this algorithm **approximate best response**, which is information state MCTS (**IS-MCTS**) with access to the opponent's policy during search. The ABR IS-MCTS algorithm is defined in Algorithm 1.

We use information state MCTS rather than providing ABR the full state (i.e. providing ABR the private state of the opponent) as we want to measure the distance from a Nash Equilibrium to enable us to have an objective measure of the strength of the agent. Consider, for instance, the sequential version of Rock/Paper/Scissors where players take turns selecting actions, which form their private information[2]. If ABR were given access to the history of the game, then the game would become trivial, as ABR would be able to exploit every strategy[3].

Our goal is to approximate an exact best response, which is calculated on an information state basis. Intuitively, we are trying to learn a strategy that, when the exploiter is given unlimited access to the strategy of the opponent, can create a valid, exploiting, counter-strategy; we are not interested in how well the exploiter does when given additional information that an opponent would not have access to. In short, we want to simulate what would happen if someone trained for years to exploit the opponent; we do not try and simulate what would happen if someone cheated against the opponent.

We use a distributed actor/learner setup. We have n actors following Algorithm 1. After each actor finishes playing a game, the terminal utility vector $\mathbf{u}(z) = \begin{bmatrix} u_0(z) \\ u_1(z) \end{bmatrix}$ along with the trajectory of state action pairs visited during play are sent to the learner, who uses them to train the value and policy network. The learner stores all data in a replay buffer and uniformly samples batches from it. The buffer has a maximum size, and when we reach that size, we remove the oldest data point. We also remove data points once they've been sampled 30 times. In practice, the actors are so slow to record data that we almost never hit the size cap.

Algorithm 1: Approximate best response policy.

input : π_{-i} — fixed policy for opponent
input : s — infostate to choose an action at
input : f_θ — value and policy network
output : $a \in \mathcal{A}_i$ — action for the current infostate

$\mathcal{N}$ = GetNodeFromSearchTree(s)
for $s = 1, \ldots, n$ **do**
    h = SampleHistoryFromInfostate(s).
    $\mathcal{T}, r$ = RunIS-MCTS($h, \mathcal{N}, f_\theta$)[4]
    UpdateSearchTree($\mathcal{T}, r$).
a = ChooseAction($\mathcal{N}$)
return a;

Following [20], the neural network, with parameters θ, is trained to predict the search probabilities π_t as a policy vector $\mathbf{p}_t$, along with the expected reward v_t using a loss function that combines the mean-squared error, cross-entropy loss, and L^2 weight regularization:

$$(\mathbf{p}, v) = f_\theta(s), \tag{7}$$

$$l = (z - v)^2 + \boldsymbol{\pi}^T \log \boldsymbol{p} + c\|\theta\|_2^2 \tag{8}$$

$$\theta_t = \textsc{GradDescent}(\theta_{t-1}, \alpha_t, l_t) \tag{9}$$

Here, v is the predicted value of a state, $\mathbf{p}$ is the predicted search outcome which is used as a prior, and z is the Monte-Carlo return for the episode.

The reward and prior are learned for information states, not histories, and thus the network learns an average across histories, which should converge to the correct value once enough data has been sent to the learner, allowing the agent to make decisions according to the information state value. As we send batches of data to the learner, and only start learning once we have a complete batch of data, the updates should be reasonably accurate. In practice, the agent was able to perform well as long as the games were recorded in batches of size $B \geq 2$. When running the tabular algorithm with $B = 1$, the

[2] i.e. player 0 chooses an action, which becomes their private information, then player 1 chooses an action, and then both actions are revealed and the game ends.

[3] At least while playing as player 1.

[4] $\mathcal{T}$ is the sequence of states and actions taken starting from h. r is the return associated with that particular Monte Carlo rollout, which either comes from a terminal history if the search reaches a terminal history, or from the value network otherwise. GetNodeFromSearchTree(s) retrieves the node corresponding to the current information state from the search tree. SampleHistoryFromInfostate samples a new history (i.e. private information for the opponent) from the exact belief distribution according to the opponent's policy. RunIS-MCTS, UpdateSearchTree, and ChooseAction follow the standard MCTS search algorithm following [20]; one such implementation is available in [15].

Parameter	Value
UCT C	2.6
L2 Coefficient	$5e-6$
Num simulations	800
Virtual loss	4
Num hidden units	128
Learning rate	10^{-5}
Num exploration steps	1
Stop exploring after	40
Min size to learn	512
Learning batch size	512
Actor batch size	32

Table 1: Hyper-parameters used.

searches were unable to accurately estimate the value of the information state, thus performing suboptimally. Following [8] we run our searches in multiple threads using tree parallelization with local mutexes on a node level to enable scaling to large games.

The specific variant of MCTS that we use is the same variant as used by [20], as we train a policy head in addition to the value head and use that to set the prior for our nodes during expansion. This can also be used as the policy in PUCT. However, we ran a number of trials to test PUCT vs UCT, and found that UCT performed better, so we do not use the policy head during search.

4 EXPERIMENTAL EVALUATION

4.1 Experimental setup

We performed hyper-parameter tuning using repeated grid searches on the small games. The best hyper-parameters are reported in Table 1, which correspond to all the experiments.

We ran ABR against UniformRandom in a number of small games, as reported in Figure 1 and 2, using both the tabular and function approximation versions. We note that both versions came close to the exact exploitability. This is unsurprising as the games are small enough for the networks to easily memorize the whole game.

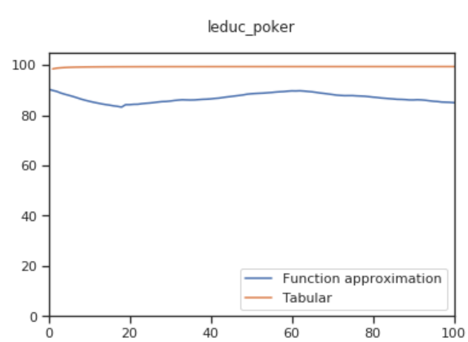


Figure 1: ABR results against UniformRandom in Leduc Poker. The y-axis is the cumulative average of the reward ABR receives divided by the reward the exact best responder receives.

Then, we ran ABR against A2C. A2C was trained for a week in selfplay. The number of steps varied. This policy is not designed to be strong, but rather to serve as an example of an exploitable policy. We used the A2C implementation from [15] with the default hyper-parameters. When played head-to-head against itself and CFR+ [22] trained for 1000 iterations against itself, we got the following results as described in Table 2. CFR+ is a known low-exploitability agent with an exploitability of approximately 10^{-5} after 1000 iterations in Leduc Poker[5]. However, when played head to head against a known exploitable agent, CFR+ failed to exploit the opponent fully, winning roughly 30% of the total possible reward, on average in Leduc. Performance was better in Liar's Dice, where CFR+ was able to win 87% of the possible reward in Liar's Dice. This is not a property of CFR+, but rather a property of an algorithm that learns a Nash Equilibrium strategy. While a Nash Equilibrium can be a strong policy, it is not an optimal exploiter policy. See [14] for more details.[6]

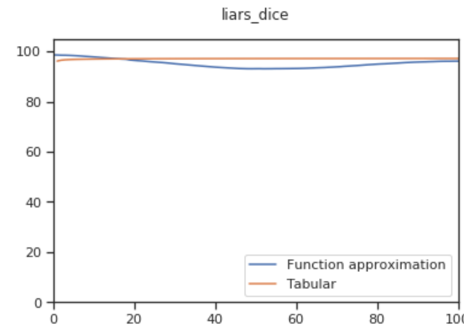


Figure 2: ABR results against UniformRandom in Liar's Dice. The y-axis is the cumulative average of the reward ABR receives divided by the reward the exact best responder receives.

Leduc Poker	CFR+1000	A2C
CFR+1000	-0.094	0.72
A2C	-0.72	0.05

Liar's Dice	CFR+1000	A2C
CFR+1000	-0.03	0.655
A2C	-0.655	-0.06

Table 2: Results from playing head to head. Column is player 0, row is player 1. Reward shown is the average over 1024 games for the row player.

When we instead play ABR against A2C, we get the results detailed in Table 3— namely, ABR wins approximately 85% of the possible maximum in Leduc, and 97% of the possible maximum in Liar's Dice. This demonstrates that head-to-head play is insufficient to gauge agent performance in the imperfect information setting, and we instead need exploitability (or a similar metric) to do so, as we're able to see that the Liar's Dice A2C agent is much stronger than the Leduc agent.

[5] Using the default implementation from [15].

[6] Consider Rock/Paper/Scissors; the Nash Equilibrium strategy is UniformRandom, which will tie AlwaysRock, despite AlwaysRock having a simple exploit.

Policy	ANC	ANC (%)	LBR	LBR (%)	NashConv
AlwaysFold	1.50	100	1.496	99.7	1.50
AlwaysCall	1.61	69	2.31	99.1	2.33
UniformRandom	8.71	98.9	5.69	64.6	8.80

Table 3: Results from playing ABR against various chump policies in Heads Up Limit Texas Hold'em. Units are in big blinds per game. ANC is ApproximateNashConv. ANC(%) is equal to ANC/NashConv · 100%.

Game	ANC	ANC (%)	NashConv
Liar's Dice	1.52	97.4	1.561490
Leduc Poker	4.07	85.8	4.747222
Kuhn Poker	0.95[7]	103.9	0.916667

Table 4: ABR w/ function approximation against UniformRandom. ANC is ApproximateNashConv.

We also present results using function approximation against UniformRandom in Table 4. We see that ABR is able to come reasonably close to the exact exploitability.

4.2 Discussion

Figure 3 demonstrates the result of the ABR IS-MCTS algorithm in a variety of games. In no game is it below 50%, and in every game other than Hex and Go, it is above 70%. It is important to note that these results are on a relatively small number of games. In chess, [20] took roughly 100 million episodes to attain a performance of 1000 Elo, while our results have a maximum of 100 thousand episodes. The performance appears to learn quite quickly, winning 80% of the time in Backgammon, a large game, with only 1 million episodes. Similarly, we see that the average reward converges closely to the exact NashConv value in the imperfect information games. One drawback is that the average reward is misleading as a metric in chess, as the games take a different amount of time depending on the outcome. We are measuring the average reward on a state level, so if there is a significant difference in game length, the average reward won't correspond to the percentage of wins. In chess, this appears to be happening, as the average game length when drawing is 440 plies, but the average game length when winning is 116 plies. This is an obvious drawback of our evaluation method, and is an implementation detail, which will be fixed in future versions of our experiment setup.

It is important to note that there is a substantial amount of learning happening in our chess agent; when the learning steps are restricted in experiments, our chess agent performs much worse, receiving an average reward of 0. In this case, what seems to be happening is that the actors who win games finish their batch of games much quicker than the actors who draw, causing the average reward as seen by the learner to have a substantial bump initially. This then levels out to the equilibrium average reward over time when the actors who draw finish their batches. In chess, other than a few initial losses in the first batch, our agent does not lose any games.

[7]This is obviously greater than the exact exploitability, which is impossible. We believe this to be due to the stochastic nature of the sampling which we're doing.

An obvious shortcoming of our approach is that we need tractable belief spaces. We have ideas on how to deal with this, such as adding an additional belief head to our network. We plan to deal with this in successor work.

We think that the tabular results demonstrate that our method converges reasonably quickly, and the function approximation results versus A2C demonstrate that the algorithm can scale to exploit learned agents in large games. Future work will analyze the ability of ABR to exploit strong agents in large games.

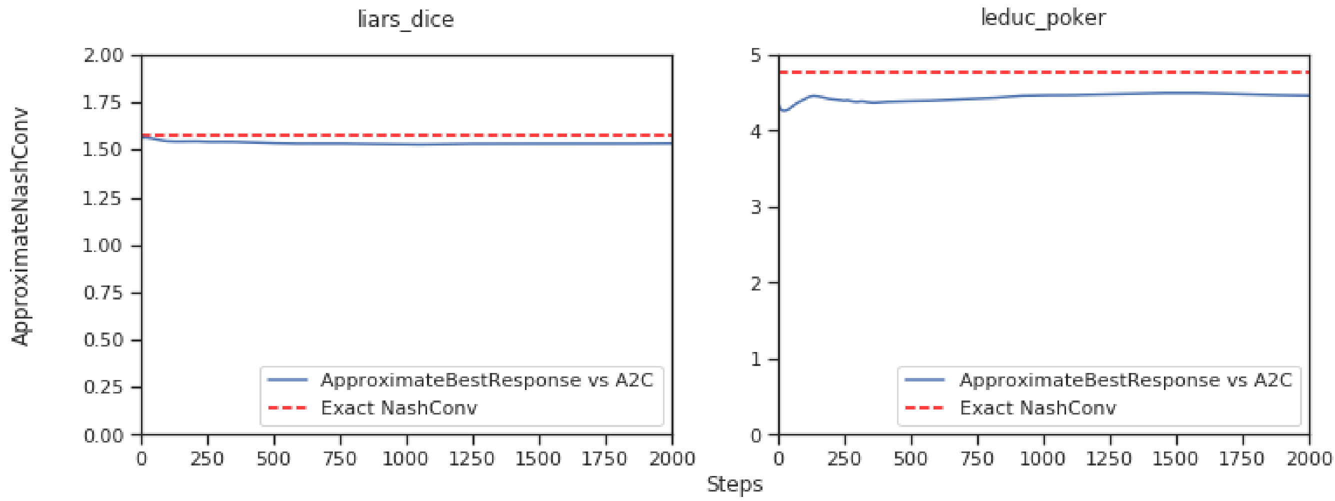
An additional point to be made is that it is still computationally expensive to train an agent to optimality. In particular, training an agent in a large game like Chess or Go requires a massive distributed computation that uses a large number of actors. Our experiments, for instance, used 200 actors, each with 4 CPUs and 8 GB of RAM, and the learner used 10 CPUs with 20 GB of RAM. For experiments using a residual convolutional network, which ended up not being necessary, we had to use a number of TPUs, which are quite expensive. As such, there is *no reason* to prefer ABR over exact exploitability if your game is sufficiently small to make that calculation feasible. ABR as a technique is preferable to exact exploitability in large games only, as it scales much better. Additionally, ABR provides a general technique that can be used in a variety of games. LBR is a strong technique for poker games, and although we demonstrate slightly better results in a small number of scenarios, that is no reason to avoid LBR. It is the fact that ABR can be used in almost any two-player, zero-sum game without additional work that is the strength of the technique.

To illustrate the difference in computational power required, a best response in Leduc Poker for a tabular policy can be calculated in 30μs, while it takes 150s to use ABR to calculate the approximate exploitability of the UniformRandom agent in Kuhn Poker, a factor of 5 million times more expensive. However, it is feasible to run ABR on, say, Go or No Limit Texas Hold'em, while calculating exploitability on such a game is prohibitively expensive (in particular, storing the result of such a calculation is not physically possible, requiring O(10^{160}) information state policies to be stored, which is more than the number of atoms in the universe.

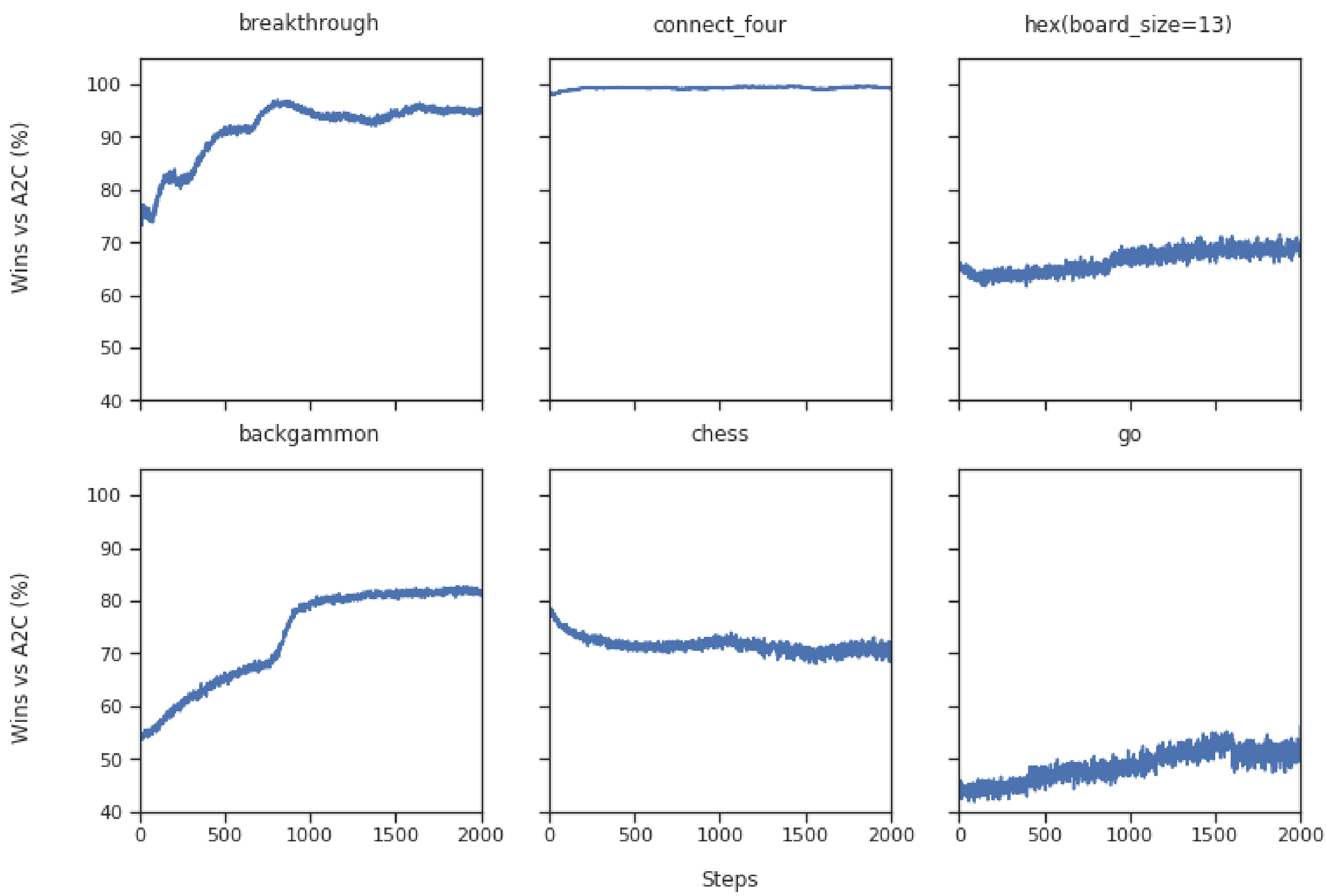
5 CONCLUSION

We introduce a new metric to support evaluation in large games of imperfect information. This metric provides a lower bound on the exploitability of algorithms in a wide class of games, and helps to provide counter examples by showing how to exploit agents. We hope that this metric will be used to advance the state of the art in large games research.

Additionally, there are a number of techniques that compute Nash Equilibria which have struggled to work in large-scale games of imperfect information, such as [11]. One potential area of future work is to use our method to extend [11] to large games, or as the exploiter in a setting similar to [23].



(a) ABR against A2C in imperfect information games. We include the NashConv of UniformRandom for context, indicating that ABR can do roughly as well against A2C as it does against UniformRandom, despite A2C presumably being stronger than UniformRandom.



(b) Win rate against A2C. Note: We use win rate here rather than ANC as it is easier to interpret for games with a win/loss outcome, rather than one with varying outcomes, such as Poker. Win rate is equivalent to a scalar transformation of ANC.

Figure 3: Results from various games playing ABR against A2C.

REFERENCES

[1] V. L. Allis. 1994. *Searching for solutions in games and artificial intelligence*. Ph.D. Dissertation. University of Limburg.

[2] David Balduzzi, Karl Tuyls, Julien Pérolat, and Thore Graepel. 2018. Re-evaluating evaluation. *CoRR* abs/1806.02643 (2018). arXiv:1806.02643 http://arxiv.org/abs/1806.02643

[3] Nolan DC Bard. 2016. Online Agent Modelling in Human-Scale Problems. (2016).

[4] Michael Bowling. 2003. *Multiagent learning in the presence of agents with limitations*. Ph.D. Dissertation.

[5] Noam Brown and Tuomas Sandholm. 2017. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 360, 6385 (December 2017).

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

[7] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence* 134, 1-2 (2002), 57–83.

[8] Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. 2008. Parallel monte-carlo tree search. In *International Conference on Computers and Games*. Springer, 60–71.

[9] David A Ferrucci. 2012. Introduction to “this is watson”. *IBM Journal of Research and Development* 56, 3.4 (2012), 1–1.

[10] Amy Greenwald, Jiacui Li, and Eric Sodomka. 2017. Solving for best responses and equilibria in extensive-form games with reinforcement learning methods. In *Rohit Parikh on Logic, Language and Society*. Springer, 185–226.

[11] Johannes Heinrich and David Silver. 2016. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. *CoRR* abs/1603.01121 (2016).

[12] Michael Johanson. 2013. Measuring the size of large no-limit poker games. *arXiv preprint arXiv:1302.7008* (2013).

[13] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. 2011. Accelerating best response calculation in large extensive games. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[14] Michael Johanson, Martin Zinkevich, and Michael Bowling. 2008. Computing robust counter-strategies. In *Advances in neural information processing systems*. 721–728.

[15] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. 2019. OpenSpiel: A Framework for Reinforcement Learning in Games. *CoRR* abs/1908.09453 (2019). arXiv:cs.LG/1908.09453 http://arxiv.org/abs/1908.09453 http://arxiv.org/abs/1908.09453.

[16] Viliam Lisý and Michael H. Bowling. 2016. Eqilibrium Approximation Quality of Current No-Limit Poker Bots. *CoRR* abs/1612.07547 (2016). arXiv:1612.07547 http://arxiv.org/abs/1612.07547

[17] Matej Moravčík, 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* 358, 6362 (October 2017).

[18] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. 2019. -Rank: Multi-Agent Evaluation by Evolution. *Scientific Reports* 9, 1 (2019), 9937. https://doi.org/10.1038/s41598-019-45619-9

[19] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. 1996. CHINOOK the world man-machine checkers champion. *AI Magazine* 17, 1 (1996), 21–21.

[20] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 632, 6419 (2018), 1140–1144.

[21] Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. 2018. Actor-critic policy optimization in partially observable multiagent environments. In *Advances in Neural Information Processing Systems*. 3422–3435.

[22] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. 2015. Solving Heads-up Limit Texas Hold'em. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*.

[23] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* (2019). https://doi.org/10.1038/s41586-019-1724-z