DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker

Matej Moravčík[♠,♥,†], Martin Schmid[♠,♥,†], Neil Burch[♠], Viliam Lisý[♠,♣],
Dustin Morrill[♠], Nolan Bard[♠], Trevor Davis[♠],
Kevin Waugh[♠], Michael Johanson[♠], Michael Bowling[♠,*]

[♠]Department of Computing Science, University of Alberta,
Edmonton, Alberta, T6G2E8, Canada

[♥]Department of Applied Mathematics, Charles University,
Prague, Czech Republic

[♣]Department of Computer Science, FEE, Czech Technical University,
Prague, Czech Republic

[†]These authors contributed equally to this work and are listed in alphabetical order.

[*]To whom correspondence should be addressed; E-mail: bowling@cs.ualberta.ca

Artificial intelligence has seen several breakthroughs in recent years, with games often serving as milestones. A common feature of these games is that players have perfect information. Poker is the quintessential game of imperfect information, and a longstanding challenge problem in artificial intelligence. We introduce DeepStack, an algorithm for imperfect information settings. It combines recursive reasoning to handle information asymmetry, decomposition to focus computation on the relevant decision, and a form of intuition that is automatically learned from self-play using deep learning. In a study involving 44,000 hands of poker, DeepStack defeated with statistical significance professional poker players in heads-up no-limit Texas hold'em. The approach is theoretically sound and is shown to produce more difficult to exploit strategies than prior approaches. [1]

Games have long served as benchmarks and marked milestones of progress in artificial intelligence (AI). In the last two decades, computer programs have reached a performance that

[1]This is the author's version of the work. It is posted here by permission of the AAAS for personal use, not for redistribution. The definitive version was published in *Science*, (March 02, 2017), doi: 10.1126/science.aam6960.

exceeds expert human players in many games, e.g., backgammon (*1*), checkers (*2*), chess (*3*), Jeopardy! (*4*), Atari video games (*5*), and go (*6*). These successes all involve games with information symmetry, where all players have identical information about the current state of the game. This property of perfect information is also at the heart of the algorithms that enabled these successes, e.g., local search during play (*7, 8*).

The founder of modern game theory and computing pioneer, von Neumann, envisioned reasoning in games without perfect information. "Real life is not like that. Real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory." (*9*) One game that fascinated von Neumann was poker, where players are dealt private cards and take turns making bets or bluffing on holding the strongest hand, calling opponents' bets, or folding and giving up on the hand and the bets already added to the pot. Poker is a game of imperfect information, where players' private cards give them asymmetric information about the state of game.

Heads-up no-limit Texas hold'em (HUNL) is a two-player version of poker in which two cards are initially dealt face-down to each player, and additional cards are dealt face-up in three subsequent rounds. No limit is placed on the size of the bets although there is an overall limit to the total amount wagered in each game (*10*). AI techniques have previously shown success in the simpler game of heads-up limit Texas hold'em, where all bets are of a fixed size resulting in just under 10^{14} decision points (*11*). By comparison, computers have exceeded expert human performance in go (*6*), a perfect information game with approximately 10^{170} decision points (*12*). The imperfect information game HUNL is comparable in size to go, with the number of decision points exceeding 10^{160} (*13*).

Imperfect information games require more complex reasoning than similarly sized perfect information games. The correct decision at a particular moment depends upon the probability distribution over private information that the opponent holds, which is revealed through their past actions. However, how our opponent's actions reveal that information depends upon their knowledge of our private information and how our actions reveal it. This kind of recursive reasoning is why one cannot easily reason about game situations in isolation, which is at the heart of heuristic search methods for perfect information games. Competitive AI approaches in imperfect information games typically reason about the entire game and produce a complete strategy prior to play (*14–16*). Counterfactual regret minimization (CFR) (*14,17,18*) is one such technique that uses self-play to do recursive reasoning through adapting its strategy against itself over successive iterations. If the game is too large to be solved directly, the common response is to solve a smaller, abstracted game. To play the original game, one translates situations and actions from the original game to the abstract game.

Although this approach makes it feasible for programs to reason in a game like HUNL, it does so by squeezing HUNL's 10^{160} situations down to the order of 10^{14} abstract situations. Likely as a result of this loss of information, such programs are behind expert human play. In 2015, the computer program Claudico lost to a team of professional poker players by a margin of 91 mbb/g (*19*), which is a "huge margin of victory" (*20*). Furthermore, it has been recently shown that abstraction-based programs from the Annual Computer Poker Competition have

massive flaws (*21*). Four such programs (including top programs from the 2016 competition) were evaluated using a local best-response technique that produces an approximate lower-bound on how much a strategy can lose. All four abstraction-based programs are beatable by over 3,000 mbb/g, which is four times as large as simply folding each game.

DeepStack takes a fundamentally different approach. It continues to use the recursive reasoning of CFR to handle information asymmetry. However, it does not compute and store a complete strategy prior to play and so has no need for explicit abstraction. Instead it considers each particular situation as it arises during play, but not in isolation. It avoids reasoning about the entire remainder of the game by substituting the computation beyond a certain depth with a fast approximate estimate. This estimate can be thought of as DeepStack's intuition: a gut feeling of the value of holding any possible private cards in any possible poker situation. Finally, DeepStack's intuition, much like human intuition, needs to be trained. We train it with deep learning (*22*) using examples generated from random poker situations. We show that DeepStack is theoretically sound, produces strategies substantially more difficult to exploit than abstraction-based techniques, and defeats professional poker players at HUNL with statistical significance.

DeepStack

DeepStack is a general-purpose algorithm for a large class of sequential imperfect information games. For clarity, we will describe its operation in the game of HUNL. The state of a poker game can be split into the players' private information, hands of two cards dealt face down, and the public state, consisting of the cards laying face up on the table and the sequence of betting actions made by the players. Possible sequences of public states in the game form a public tree with every public state having an associated public subtree (Fig. 1).

A player's strategy defines a probability distribution over valid actions for each decision point, where a decision point is the combination of the public state and the hand for the acting player. Given a player's strategy, for any public state one can compute the player's range, which is the probability distribution over the player's possible hands given that the public state is reached.

Fixing both players' strategies, the utility for a particular player at a terminal public state, where the game has ended, is a bilinear function of both players' ranges using a payoff matrix determined by the rules of the game. The expected utility for a player at any other public state, including the initial state, is the expected utility over reachable terminal states given the players' fixed strategies. A best-response strategy is one that maximizes a player's expected utility against an opponent strategy. In two-player zero-sum games, like HUNL, a solution or Nash equilibrium strategy (*23*) maximizes the expected utility when playing against a best-response opponent strategy. The exploitability of a strategy is the difference in expected utility against its best-response opponent and the expected utility under a Nash equilibrium.

The DeepStack algorithm seeks to compute and play a low-exploitability strategy for the

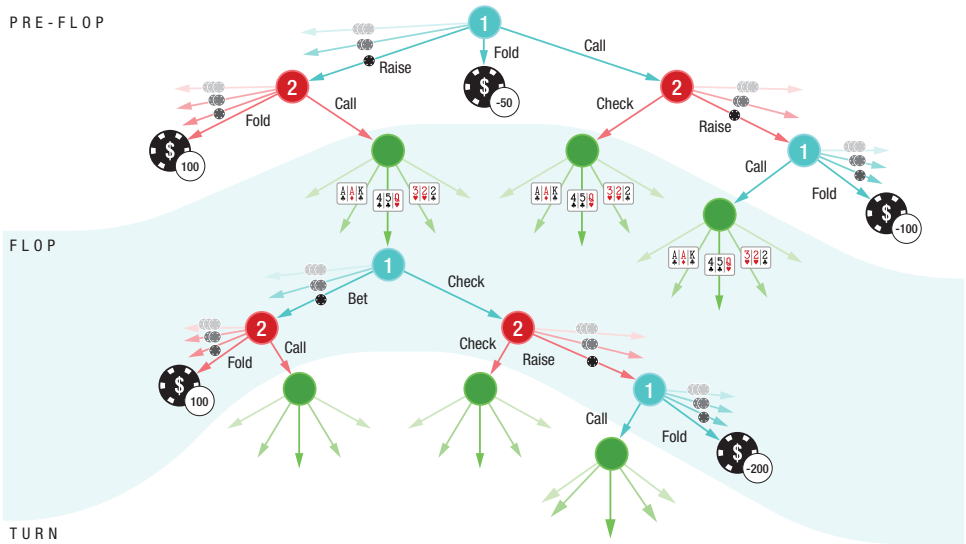

Figure 1: A portion of the public tree in HUNL. Nodes represent public states, whereas edges represent actions: red and turquoise showing player betting actions, and green representing public cards revealed by chance. The game ends at terminal nodes, shown as a chip with an associated value. For terminal nodes where no player folded, the player whose private cards form a stronger poker hand receives the value of the state.

game, i.e., solve for an approximate Nash equilibrium. DeepStack computes this strategy during play only for the states of the public tree that actually arise. Although computed during play, DeepStack's strategy is static, albeit stochastic, because it is the result of a deterministic computation that produces a probability distribution over the available actions.

The DeepStack algorithm (Fig. 2) is composed of three ingredients: a sound local strategy computation for the current public state, depth-limited lookahead using a learned value function to avoid reasoning to the end of the game, and a restricted set of lookahead actions. At a conceptual level these three ingredients describe heuristic search, which is responsible for many of AI's successes in perfect information games. Until DeepStack, no theoretically sound application of heuristic search was known in imperfect information games. The heart of heuristic search methods is the idea of "continual re-searching", where a sound local search procedure is invoked whenever the agent must act without retaining any memory of how or why it acted to reach the current state. At the heart of DeepStack is continual re-solving, a sound local strategy computation which only needs minimal memory of how and why it acted to reach the current public state.

Continual re-solving. Suppose we have taken actions according to a particular solution strategy but then in some public state forget this strategy. Can we reconstruct a solution strategy for the subtree without having to solve the entire game again? We can, through the process of

re-solving (*17*). We need to know both our range at the public state and a vector of expected values achieved by the opponent under the previous solution for each opponent hand (*24*). With these values, we can reconstruct a strategy for only the remainder of the game, which does not increase our overall exploitability. Each value in the opponent's vector is a counterfactual value, a conditional "what-if" value that gives the expected value if the opponent reaches the public state with a particular hand. The CFR algorithm also uses counterfactual values, and if we use CFR as our solver, it is easy to compute the vector of opponent counterfactual values at any public state.

Re-solving, however, begins with a strategy, whereas our goal is to avoid ever maintaining a strategy for the entire game. We get around this by doing continual re-solving: reconstructing a strategy by re-solving every time we need to act; never using the strategy beyond our next action. To be able to re-solve at any public state, we need only keep track of our own range and a suitable vector of opponent counterfactual values. These values must be an upper bound on the value the opponent can achieve with each hand in the current public state, while being no larger than the value the opponent could achieve had they deviated from reaching the public state. This is an important relaxation of the counterfactual values typically used in re-solving, with a proof of sufficiency included in our proof of Theorem 1 below (*10*).

At the start of the game, our range is uniform and the opponent counterfactual values are initialized to the value of being dealt each private hand. When it is our turn to act we re-solve the subtree at the current public state using the stored range and opponent values, and act according to the computed strategy, discarding the strategy before we act again. After each action, either by a player or chance dealing cards, we update our range and opponent counterfactual values according to the following rules: (i) Own action: replace the opponent counterfactual values with those computed in the re-solved strategy for our chosen action. Update our own range using the computed strategy and Bayes' rule. (ii) Chance action: replace the opponent counterfactual values with those computed for this chance action from the last re-solve. Update our own range by zeroing hands in the range that are impossible given new public cards. (iii) Opponent action: no change to our range or the opponent values are required.

These updates ensure the opponent counterfactual values satisfy our sufficient conditions, and the whole procedure produces arbitrarily close approximations of a Nash equilibrium (see Theorem 1). Notice that continual re-solving never keeps track of the opponent's range, instead only keeping track of their counterfactual values. Furthermore, it never requires knowledge of the opponent's action to update these values, which is an important difference from traditional re-solving. Both will prove key to making this algorithm efficient and avoiding any need for the translation step required with action abstraction methods (*25, 26*).

Continual re-solving is theoretically sound, but by itself impractical. While it does not ever maintain a complete strategy, re-solving itself is intractable except near the end of the game. In order to make continual re-solving practical, we need to limit the depth and breadth of the re-solved subtree.

Limited depth lookahead via intuition. As in heuristic search for perfect information games, we would like to limit the depth of the subtree we have to reason about when re-solving. However, in imperfect information games we cannot simply replace a subtree with a heuristic or precomputed value. The counterfactual values at a public state are not fixed, but depend on how players play to reach the public state, i.e., the players' ranges (*17*). When using an iterative algorithm, such as CFR, to re-solve, these ranges change on each iteration of the solver.

DeepStack overcomes this challenge by replacing subtrees beyond a certain depth with a learned counterfactual value function that approximates the resulting values if that public state were to be solved with the current iteration's ranges. The inputs to this function are the ranges for both players, as well as the pot size and public cards, which are sufficient to specify the public state. The outputs are a vector for each player containing the counterfactual values of holding each hand in that situation. In other words, the input is itself a description of a poker game: the probability distribution of being dealt individual private hands, the stakes of the game, and any public cards revealed; the output is an estimate of how valuable holding certain cards would be in such a game. The value function is a sort of intuition, a fast estimate of the value of finding oneself in an arbitrary poker situation. With a depth limit of four actions, this approach reduces the size of the game for re-solving from 10^{160} decision points at the start of the game down to no more than 10^{17} decision points. DeepStack uses a deep neural network as its learned value function, which we describe later.

Sound reasoning. DeepStack's depth-limited continual re-solving is sound. If DeepStack's intuition is "good" and "enough" computation is used in each re-solving step, then DeepStack plays an arbitrarily close approximation to a Nash equilibrium.

Theorem 1 *If the values returned by the value function used when the depth limit is reached have error less than ϵ, and T iterations of CFR are used to re-solve, then the resulting strategy's exploitability is less than $k_1\epsilon + k_2/\sqrt{T}$, where k_1 and k_2 are game-specific constants. For the proof, see (10).*

Sparse lookahead trees. The final ingredient in DeepStack is the reduction in the number of actions considered so as to construct a sparse lookahead tree. DeepStack builds the lookahead tree using only the actions fold (if valid), call, 2 or 3 bet actions, and all-in. This step voids the soundness property of Theorem 1, but it allows DeepStack to play at conventional human speeds. With sparse and depth-limited lookahead trees, the re-solved games have approximately 10^7 decision points, and are solved in under five seconds using a single NVIDIA GeForce GTX 1080 graphics card. We also use the sparse and depth-limited lookahead solver from the start of the game to compute the opponent counterfactual values used to initialize DeepStack's continual re-solving.

Relationship to heuristic search in perfect information games. There are three key challenges that DeepStack overcomes to incorporate heuristic search ideas in imperfect information

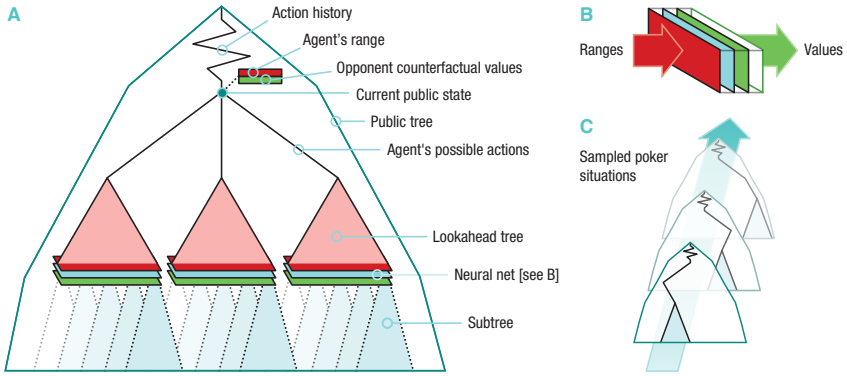

Figure 2: **DeepStack overview.** **(A)** DeepStack reasons in the public tree always producing action probabilities for all cards it can hold in a public state. It maintains two vectors while it plays: its own range and its opponent's counterfactual values. As the game proceeds, its own range is updated via Bayes' rule using its computed action probabilities after it takes an action. Opponent counterfactual values are updated as discussed under "Continual re-solving". To compute action probabilities when it must act, it performs a re-solve using its range and the opponent counterfactual values. To make the re-solve tractable it restricts the available actions of the players and lookahead is limited to the end of the round. During the re-solve, counterfactual values for public states beyond its lookahead are approximated using DeepStack's learned evaluation function. **(B)** The evaluation function is represented with a neural network that takes the public state and ranges from the current iteration as input and outputs counterfactual values for both players (Fig. 3). **(C)** The neural network is trained prior to play by generating random poker situations (pot size, board cards, and ranges) and solving them to produce training examples. Complete pseudocode can be found in Algorithm S1 (*10*).

games. First, sound re-solving of public states cannot be done without knowledge of how and why the players acted to reach the public state. Instead, two additional vectors, the agent's range and opponent counterfactual values, must be maintained to be used in re-solving. Second, re-solving is an iterative process that traverses the lookahead tree multiple times instead of just once. Each iteration requires querying the evaluation function again with different ranges for every public state beyond the depth limit. Third, the evaluation function needed when the depth limit is reached is conceptually more complicated than in the perfect information setting. Rather than returning a single value given a single state in the game, the counterfactual value function needs to return a vector of values given the public state and the players' ranges. Because of this complexity, to learn such a value function we use deep learning, which has also been successful at learning complex evaluation functions in perfect information games (*6*).

Relationship to abstraction-based approaches. Although DeepStack uses ideas from abstraction, it is fundamentally different from abstraction-based approaches. DeepStack restricts the number of actions in its lookahead trees, much like action abstraction (*25, 26*). However,

each re-solve in DeepStack starts from the actual public state and so it always perfectly understands the current situation. The algorithm also never needs to use the opponent's actual action to obtain correct ranges or opponent counterfactual values, thereby avoiding translation of opponent bets. We used hand clustering as inputs to our counterfactual value functions, much like explicit card abstraction approaches (*27, 28*). However, our clustering is used to estimate counterfactual values at the end of a lookahead tree rather than limiting what information the player has about their cards when acting. We later show that these differences result in a strategy substantially more difficult to exploit.

Deep Counterfactual Value Networks

Deep neural networks have proven to be powerful models and are responsible for major advances in image and speech recognition (*29, 30*), automated generation of music (*31*), and game-playing (*5, 6*). DeepStack uses deep neural networks with a tailor-made architecture, as the value function for its depth-limited lookahead (Fig. 3). Two separate networks are trained: one estimates the counterfactual values after the first three public cards are dealt (flop network), the other after dealing the fourth public card (turn network). An auxiliary network for values before any public cards are dealt is used to speed up the re-solving for early actions (*10*).

Architecture. DeepStack uses a standard feedforward network with seven fully connected hidden layers each with 500 nodes and parametric rectified linear units (*32*) for the output. This architecture is embedded in an outer network that forces the counterfactual values to satisfy the zero-sum property. The outer computation takes the estimated counterfactual values, and computes a weighted sum using the two players' input ranges resulting in separate estimates of the game value. These two values should sum to zero, but may not. Half the actual sum is then subtracted from the two players' estimated counterfactual values. This entire computation is differentiable and can be trained with gradient descent. The network's inputs are the pot size as a fraction of the players' total stacks and an encoding of the players' ranges as a function of the public cards. The ranges are encoded by clustering hands into 1,000 buckets, as in traditional abstraction methods (*27, 28, 33*), and input as a vector of probabilities over the buckets. The output of the network are vectors of counterfactual values for each player and hand, interpreted as fractions of the pot size.

Training. The turn network was trained by solving 10 million randomly generated poker turn games. These turn games used randomly generated ranges, public cards, and a random pot size (*10*). The target counterfactual values for each training game were generated by solving the game with players' actions restricted to fold, call, a pot-sized bet, and an all-in bet, but no card abstraction. The flop network was trained similarly with 1 million randomly generated flop games. However, the target counterfactual values were computed using our depth-limited

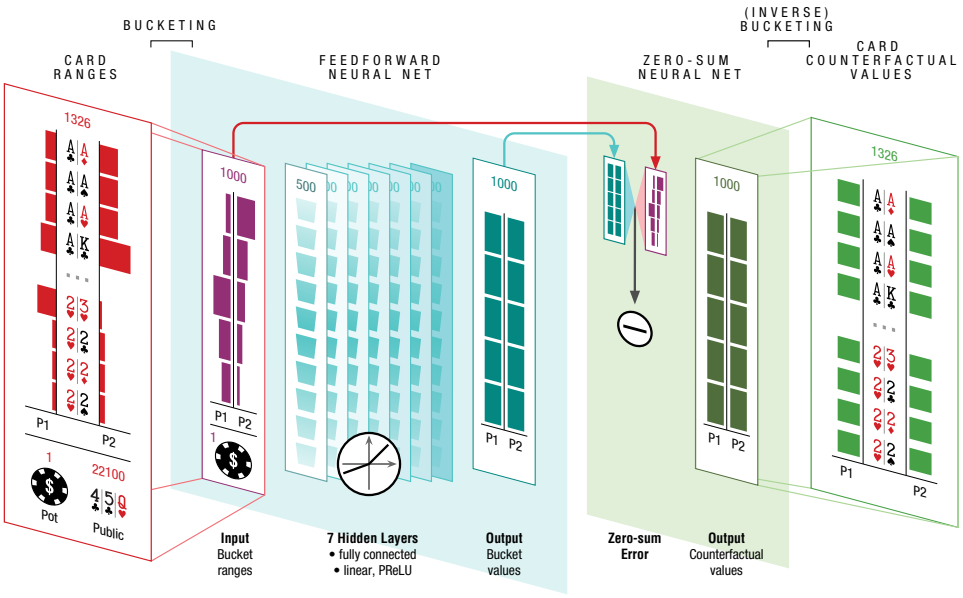

Figure 3: **Deep counterfactual value network.** The inputs to the network are the pot size, public cards, and the player ranges, which are first processed into hand clusters. The output from the seven fully connected hidden layers is post-processed to guarantee the values satisfy the zero-sum constraint, and then mapped back into a vector of counterfactual values.

solving procedure and our trained turn network. The networks were trained using the Adam gradient descent optimization procedure (*34*) with a Huber loss (*35*).

Evaluating DeepStack

We evaluated DeepStack by playing it against a pool of professional poker players recruited by the International Federation of Poker (*36*). Thirty-three players from 17 countries were recruited. Each was asked to complete a 3,000 game match over a period of four weeks between November 7th and December 12th, 2016. Cash incentives were given to the top three performers ($5,000, $2,500, and $1,250 CAD).

Evaluating performance in HUNL is challenging because of the large variance in per-game outcomes owing to randomly dealt cards and stochastic choices made by the players. The better player may lose in a short match simply because they were dealt weaker hands or their rare bluffs were made at inopportune times. As seen in the Claudico match (*20*), even 80,000 games may not be enough to statistically significantly separate players whose skill differs by a considerable margin. We evaluate performance using AIVAT (*37*), a provably unbiased low-variance technique for evaluating performance in imperfect information games based on carefully constructed control variates. AIVAT requires an estimated value of holding each hand in each public

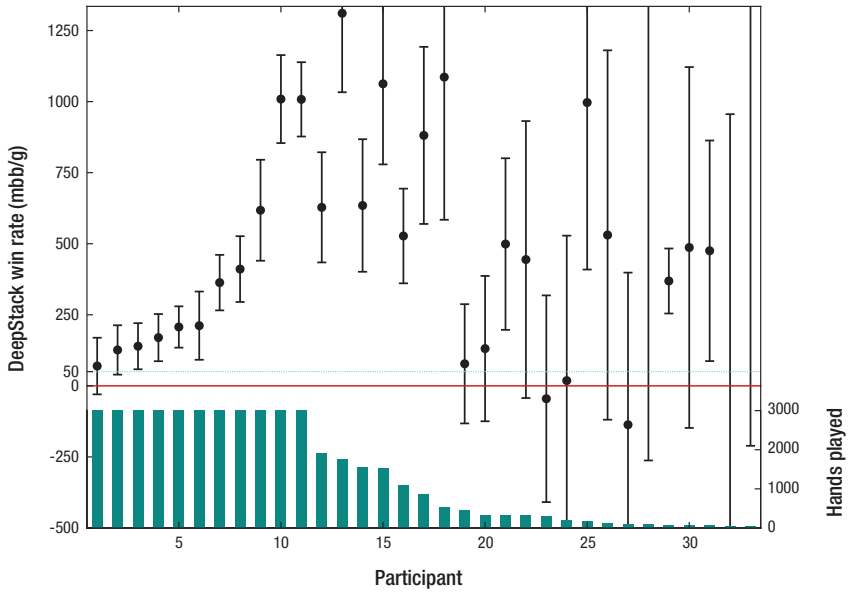

Figure 4: **Performance of professional poker players against DeepStack.** Performance estimated with AIVAT along with a 95% confidence interval. The solid bars at the bottom show the number of games the participant completed.

state, and then uses the expected value changes that occur due to chance actions and actions of players with known strategies (i.e., DeepStack) to compute the control variate. DeepStack's own value function estimate is perfectly suited for AIVAT. Indeed, when used with AIVAT we get an unbiased performance estimate with an impressive 85% reduction in standard deviation. Thanks to this technique, we can show statistical significance (*38*) in matches with as few as 3,000 games.

In total 44,852 games were played by the thirty-three players with 11 players completing the requested 3,000 games. Over all games played, DeepStack won 492 mbb/g. This is over 4 standard deviations away from zero, and so, highly significant. Note that professional poker players consider 50 mbb/g a sizable margin. Using AIVAT to evaluate performance, we see DeepStack was overall a bit lucky, with its estimated performance actually 486 mbb/g. However, as a lower variance estimate, this margin is over 20 standard deviations from zero.

The performance of individual participants measured with AIVAT is summarized in Figure 4. Amongst those players that completed the requested 3,000 games, DeepStack is estimated to be winning by 394 mbb/g, and individually beating 10 out of 11 such players by a statistically significant margin. Only for the best performing player, still estimated to be losing by 70 mbb/g, is the result not statistically significant. More details on the participants and their results are presented in (*10*).

Table 1: **Exploitability bounds from Local Best Response.** For all listed programs, the value reported is the largest estimated exploitability when applying LBR using a variety of different action sets. Table S2 gives a more complete presentation of these results (*10*). ‡: LBR was unable to identify a positive lower bound for DeepStack's exploitability.

Program	LBR (mbb/g)
Hyperborean (2014)	4675
Slumbot (2016)	4020
Act1 (2016)	3302
Always Fold	750
DeepStack	**0** ‡

Exploitability

The main goal of DeepStack is to approximate Nash equilibrium play, i.e., minimize exploitability. While the exact exploitability of a HUNL poker strategy is intractable to compute, the recent local best-response technique (LBR) can provide a lower bound on a strategy's exploitability (*21*) given full access to its action probabilities. LBR uses the action probabilities to compute the strategy's range at any public state. Using this range it chooses its response action from a fixed set using the assumption that no more bets will be placed for the remainder of the game. Thus it best-responds locally to the opponent's actions, providing a lower bound on their overall exploitability. As already noted, abstraction-based programs from the Annual Computer Poker Competition are highly exploitable by LBR: four times more exploitable than folding every game (Table 1). However, even under a variety of settings, LBR fails to exploit DeepStack at all — itself losing by over 350 mbb/g to DeepStack (*10*). Either a more sophisticated lookahead is required to identify DeepStack's weaknesses or it is substantially less exploitable.

Discussion

DeepStack defeated professional poker players at HUNL with statistical significance (*39*), a game that is similarly sized to go, but with the added complexity of imperfect information. It achieves this goal with little domain knowledge and no training from expert human games. The implications go beyond being a milestone for artificial intelligence. DeepStack represents a paradigm shift in approximating solutions to large, sequential imperfect information games. Abstraction and offline computation of complete strategies has been the dominant approach for almost 20 years (*33, 40, 41*). DeepStack allows computation to be focused on specific situations that arise when making decisions and the use of automatically trained value functions. These are two of the core principles that have powered successes in perfect information games, albeit conceptually simpler to implement in those settings. As a result, the gap between the largest

perfect and imperfect information games to have been mastered is mostly closed.

With many real world problems involving information asymmetry, DeepStack also has implications for seeing powerful AI applied more in settings that do not fit the perfect information assumption. The abstraction paradigm for handling imperfect information has shown promise in applications like defending strategic resources (*42*) and robust decision making as needed for medical treatment recommendations (*43*). DeepStack's continual re-solving paradigm will hopefully open up many more possibilities.

References and Notes

1. G. Tesauro, *Communications of the ACM* **38**, 58 (1995).
2. J. Schaeffer, R. Lake, P. Lu, M. Bryant, *AI Magazine* **17**, 21 (1996).
3. M. Campbell, A. J. Hoane Jr., F. Hsu, *Artificial Intelligence* **134**, 57 (2002).
4. D. Ferrucci, *IBM Journal of Research and Development* **56**, 1:1 (2012).
5. V. Mnih, *et al.*, *Nature* **518**, 529 (2015).
6. D. Silver, *et al.*, *Nature* **529**, 484 (2016).
7. A. L. Samuel, *IBM Journal of Research and Development* **3**, 210 (1959).
8. L. Kocsis, C. Szepesvári, *Proceedings of the Seventeenth European Conference on Machine Learning* (2006), pp. 282–293.
9. J. Bronowski, The ascent of man, Documentary (1973). Episode 13.
10. See Supplementary Materials.
11. In 2008, Polaris defeated a team of professional poker players in heads-up limit Texas hold'em (*44*). In 2015, Cepheus essentially solved the game (*18*).
12. V. L. Allis, Searching for solutions in games and artificial intelligence, Ph.D. thesis, University of Limburg (1994).
13. M. Johanson, Measuring the size of large no-limit poker games, *Technical Report TR13-01*, Department of Computing Science, University of Alberta (2013).
14. M. Zinkevich, M. Johanson, M. Bowling, C. Piccione, *Advances in Neural Information Processing Systems 20* (2008), pp. 905–912.
15. A. Gilpin, S. Hoda, J. Peña, T. Sandholm, *Proceedings of the Third International Workshop On Internet And Network Economics* (2007), pp. 57–69.

16. End-game solving (*17, 45, 46*) is one exception to computation occurring prior to play. When the game nears the end, a new computation is invoked over the remainder of the game. Thus, the program need not store this part of the strategy or can use a finer-grained abstraction aimed to improve the solution quality. We discuss this as re-solving when we introduce DeepStack's technique of continual re-solving.

17. N. Burch, M. Johanson, M. Bowling, *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence* (2014), pp. 602–608.

18. M. Bowling, N. Burch, M. Johanson, O. Tammelin, *Science* **347**, 145 (2015).

19. We use milli-big-blinds per game (mbb/g) to measure performance in poker, where a milli-big-blind is one thousandth of the forced big blind bet amount that starts the game. This normalizes performance for the number of games played and the size of stakes. For comparison, a win rate of 50 mbb/g is considered a sizable margin by professional players and 750 mbb/g is the rate that would be lost if a player folded each game. The poker community commonly uses big blinds per one hundred games (bb/100) to measure win rates, where 10 mbb/g equals 1 bb/100.

20. J. Wood, Doug Polk and team beat Claudico to win $100,000 from Microsoft & The Rivers Casino, *Pokerfuse,* http://pokerfuse.com/news/media-and-software/26854-doug-polk-and-team-beat-claudico-win-100000-microsoft/ (2015).

21. V. Lisý, M. Bowling, *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games* (2017). https://arxiv.org/abs/1612.07547.

22. DeepStack is not the first application of deep learning to the game of poker. Previous applications of deep learning, though, are either not known to be theoretically sound (*47*), or have only been applied in small games (*48*).

23. J. F. Nash, *Proceedings of the National Academy of Sciences* **36**, 48 (1950).

24. When the previous solution is an approximation, rather than an exact Nash equilibrium, sound re-solving needs the expected values from a best-response to the player's previous strategy. In practice, though, using expected values from the previous solution in self-play often works as well or better than best-response values (*10*).

25. A. Gilpin, T. Sandholm, T. B. Sørensen, *Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems* (2008), pp. 911–918.

26. D. Schnizlein, M. Bowling, D. Szafron, *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (2009), pp. 278–284.

27. A. Gilpin, T. Sandholm, T. B. Sørensen, *Proceedings of the Twenty-Second Conference on Artificial Intelligence* (2007), pp. 50–57.

28. M. Johanson, N. Burch, R. Valenzano, M. Bowling, *Proceedings of the Twelfth International Conference on Autonomous Agents and Multi-Agent Systems* (2013), pp. 271–278.

29. A. Krizhevsky, I. Sutskever, G. E. Hinton, *Advances in Neural Information Processing Systems 25* (2012), pp. 1106–1114.

30. G. Hinton, *et al.*, *IEEE Signal Processing Magazine* **29**, 82 (2012).

31. A. van den Oord, *et al.*, *CoRR* **abs/1609.03499** (2016).

32. K. He, X. Zhang, S. Ren, J. Sun, *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1026–1034.

33. J. Shi, M. L. Littman, *Proceedings of the Second International Conference on Computers and Games* (2000), pp. 333–345.

34. D. P. Kingma, J. Ba, *Proceedings of the Third International Conference on Learning Representations* (2014).

35. P. J. Huber, *Annals of Mathematical Statistics* **35**, 73 (1964).

36. International Federation of Poker. http://pokerfed.org/about/ (Accessed January 1, 2017).

37. N. Burch, M. Schmid, M. Moravcik, M. Bowling, *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games* (2017). http://arxiv.org/abs/1612.06915.

38. Statistical significance where noted was established using a two-tailed Student's t-test at the 0.05 level with $N \geq 3000$.

39. Subsequent to our study, the computer program Libratus, developed at CMU by Tuomas Sandholm and Noam Brown, defeated a team of four professional heads-up poker specialists in a HUNL competition held January 11-30, 2017. Libratus has been described as using "nested endgame solving" (*49*), a technique with similarities to continual re-solving, but developed independently. Libratus employs this re-solving when close to the end of the game rather than at every decision, while using an abstraction-based approach earlier in the game.

40. D. Billings, *et al.*, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (2003), pp. 661–668.

41. T. Sandholm, *AI Magazine* **31**, 13 (2010).

42. V. Lisy, T. Davis, M. Bowling, *Proceedings of the Thirtieth Conference on Artificial Intelligence* (2016), pp. 544–550.

43. K. Chen, M. Bowling, *Advances in Neural Information Processing Systems 25* (2012), pp. 2078–2086.

44. J. Rehmeyer, N. Fox, R. Rico, *Wired* **16.12**, 186 (2008).

45. S. Ganzfried, T. Sandholm, *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems* (2015), pp. 37–45.

46. M. Moravcik, M. Schmid, K. Ha, M. Hladík, S. J. Gaukrodger, *Proceedings of the Thirtieth Conference on Artificial Intelligence* (2016), pp. 572–578.

47. N. Yakovenko, L. Cao, C. Raffel, J. Fan, *Proceedings of the Thirtieth Conference on Artificial Intelligence* (2016), pp. 360–367.

48. J. Heinrich, D. Silver, *arXiv preprint arXiv:1603.01121* (2016).

49. N. Brown, T. Sandholm, *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games* (2017).

50. M. Zinkevich, M. Littman, *Journal of the International Computer Games Association* **29**, 166 (2006). News item.

51. D. Morrill, Annual computer poker competition poker GUI client, https://github.com/dmorrill10/acpc_poker_gui_client/tree/v1.2 (2012).

52. O. Tammelin, N. Burch, M. Johanson, M. Bowling, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015), pp. 645–652.

53. R. Collobert, K. Kavukcuoglu, C. Farabet, *BigLearn, NIPS Workshop* (2011).

54. S. Ganzfried, T. Sandholm, *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence* (2014), pp. 682–690.

Acknowledgements. The hand histories of all games played in the human study as well as those used to generate the LBR results against DeepStack are in (*10*). We would like to thank the IFP, all of the professional players who committed valuable time to play against DeepStack, the anonymous reviewers' invaluable feedback and suggestions, and R. Holte, A. Brown, and K. Blažková for comments on early drafts of this article. We especially would like to thank IBM for their support of this research through an IBM faculty grant. The research was also supported by Alberta Innovates through the Alberta Machine Intelligence Institute, the Natural Sciences and Engineering Research Council of Canada, and Charles University (GAUK) Grant no. 391715. This work was only possible thanks to computing resources provided by Compute Canada and Calcul Québec. MM and MS are on leave from IBM Prague. MJ serves as a Research Scientist, and MB as a Contributing Brain Trust Member, at Cogitai, Inc. DM owns 200 shares of Gamehost Inc.