Machine Learning (2022) 111:3063–3083
https://doi.org/10.1007/s10994-022-06179-8

World-class interpretable poker

Dimitris Bertsimas[1] · Alex Paskov[2]

Received: 2 February 2021 / Revised: 21 February 2022 / Accepted: 8 April 2022 /
Published online: 9 June 2022

Abstract
We address the problem of interpretability in iterative game solving for imperfect-information games such as poker. This lack of interpretability has two main sources: first, the use of an uninterpretable feature representation, and second, the use of black box methods such as neural networks, for the fitting procedure. In this paper, we present advances on both fronts. Namely, first we propose a novel, compact, and easy-to-understand game-state feature representation for Heads-up No-limit (HUNL) Poker. Second, we make use of globally optimal decision trees, paired with a counterfactual regret minimization (CFR) self-play algorithm, to train our poker bot which produces an entirely interpretable agent. Through experiments against Slumbot, the winner of the most recent Annual Computer Poker Competition, we demonstrate that our approach yields a HUNL Poker agent that is capable of beating the Slumbot. Most exciting of all, the resulting poker bot is highly interpretable, allowing humans to learn from the novel strategies it discovers.

Keywords Optimal decision trees · Game solving · Deep reinforcement learning · Artificial intelligence · Interpretability · Poker · Regret minimization

1 Introduction

Over the past two decades, reinforcement learning has yielded phenomenal successes in the domain of perfect-information games: it has produced world-class bots capable of outperforming even the strongest human competitors in games such as Chess and GO (Silver et al., 2018). In contrast, progress in the domain of imperfect-information games, such

Editor: Henrik Boström.

✉ Dimitris Bertsimas
dbertsim@mit.edu

Alex Paskov
asp2205@columbia.edu

1 Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

2 Department of Applied Mathematics, Columbia University, New York City, NY 10027, USA

as Poker, where the players hold disjoint public/private information sets, has proven to be notoriously difficult.

Nonetheless, in recent years we have witnessed several breakthroughs which have yielded poker bots capable of achieving super-human performance (Brown & Sandholm, 2018) and capable of beating human professionals (Moravčık et al. 2017). While this is an impressive milestone, and certainly a critical step towards the ultimately applying control towards real life problems (i.e., non-recreational games), unfortunately to-date all proposed methods have suffered from a lack of interpretability. This is a serious drawback, and one which will severely limit the degree to which these methods will be accepted by regulators and the society at large.

The recent trend of using machine learning algorithms to guide poker agents (Li et al., 2020; Brown et al., 2019; Zarick et al., 2020; Moravčık et al., 2017) has shown promise, but still suffers from a lack of interpretability for humans. This has two main sources: first, the use of an uninterpretable feature representation; second, the use of black box methods such as neural networks, for the fitting procedure. In this paper, we present advances on both fronts. Namely, first we propose a novel, compact, and easy-to-understand game-state feature representation for Heads-up No-limit (HUNL) Poker. Second, we make use of globally optimal decision trees, paired with a counterfactual regret minimization (CFR) self-play algorithm, to train our poker bot. Through experiments against Slumbot, a strong poker agent and the winner of the most recent Annual Computer Poker Competition, we demonstrate that our approach yields a HUNL Poker agent that is capable of beating the Slumbot. Most exciting of all, the resulting poker bot is interpretable, allowing humans to learn from the novel strategies it discovers.

1.1 Literature

The use of uninterpretable card grouping, real-time move refinement, and Neural Networks in the creation of Poker bots for HUNL Poker is so widely employed that to the best of our knowledge, other work on the subject of using interpretable learning algorithms to produce master-level poker bots or rules for HUNL is sparse. Specifically for the case of HUNL Poker, the closest comes from Moravčık et al. (2017), only in that it used partially-interpretable card ranges as features to their learning algorithm. However, the subsequent use of Neural Networks and lack of further investigation into the performance of interpretable learning algorithms, such as Decision Trees or Linear Regression, resulted in no directly interpretable framework. In contrast, we employ interpretable model features and learning algorithms, so that the entire pipeline from generating features to training the model results in a wholly interpretable result. There also exists literature, Ganzfried and Yusuf (2017), along a similar line of this paper, specifically applying decision trees to find human-understandable rules for No-Limit Poker—though in a smaller, simplified version than HUNL. Ganzfried and Chiswick (2019) serves as a follow-up, using regression models to similarly find an over-arching, interpretable rule for the simplified game. Finally, Ganzfried and Sandholm (2010) uses an interpretable model of ”the equilibrium structure” of limit Texas Hold’em to improve strategy computation.

Furthermore, previous applications of Neural Networks to poker have relied on one-hot encodings or sparse embeddings of the cards and betting sequence in a game-state, with the intention of the Neural Network learning an appropriate representation (Brown et al., 2019; Li et al., 2020). However, such large representations are not compact enough for application to interpretable machine learning techniques, such as Decision Trees. As such, our representation of the game-state is the result of much experimentation, and ultimately represents a game-state in a compact and rich manner.

1.2 Contributions and structure

In this paper, we present a methodology we have developed for creating a HUNL Poker bot that is both highly performant and interpretable. We show that our player outperforms the strong poker AI Slumbot when trained using either of three learning algorithms: Optimal Classification Trees, Extreme Gradient Boosted Trees (XGBoost), and Feedforward Neural Networks. We also show that our interpretable framework can help improve human poker ability, by analyzing the perceptible strategy of the Optimal Classification Tree player. The main contributions of this paper are:

1. A new feature representation of a poker game-state, that is both directly interpretable to humans and a rich, compact representation of the game-state.
2. To our knowledge, this is the first methodology that produces an interpretable and highly performant Poker agent using interpretable machine learning techniques (Optimal Classification Trees) over Neural Networks. As a result, humans have the ability to study the produced strategies and in-turn directly improve their own abilities.
3. By applying several different learning algorithms in our framework, we produce three HUNL Poker bots that outperform the most recent Annual Computer Poker Competition winner in HUNL, Slumbot.

The structure of the paper is as follows. In Sect. 2, we describe HUNL Poker and outline the framework of our approach—in particular, the interpretable feature representation of a HUNL Poker game-state, the self-play CFR algorithm and optimizations used, and the learning algorithms. In Sect. 3, we analyze the effect of the learning algorithm by presenting game-play results against Slumbot. In Sect. 4, we examine the effect of self-play time on performance. In Sect. 5, we investigate the interpretability of our methods, and illustrate how we, as humans, can learn from the players. In Sect. 6, we summarize our findings and report our conclusions.

2 The Framework

2.1 Heads-up no-limit Texas Hold'em

Heads-Up No-Limit Texas Hold'em (HUNL) is a very popular poker variant, involving two players and a standard 52-card deck. As is typically the case in HUNL literature, both players start with \$20,000. At the start of a hand, the two players are dealt two private cards from the deck. In addition, the first player must put down \$50 and the second player \$100. Now, the first round of betting—the Preflop—begins with player 1. Here, and in all future turns, a player can either call, fold, check, or raise; if they call, they must match the amount of money the other player has put in; if they fold, they lose and forfeit all money in the pot; if it is the start of a betting round, the player may check, which corresponds to betting no additional money; finally, if the player raises, they must match the amount of money the other player has put in and then put in additional funds (note: the minimum raise is \$100, the player may not bet more money than they have, and they must add additional funds greater than all prior raises). After the first round of betting, the second round—the Flop—begins, in which three public cards are dealt. Two more rounds of betting follow this—the Turn and River, respectively—each adding one public card to the table. If the end of the River is reached, then the winner is determined by the player with the best 5-card hand, comprised of their two private cards and any three public cards. In the unlikely case of a split, the money is evenly distributed back to the two players.

Finally, if multiple hands are to be played, the role of the two players is swapped so that the players alternate between the initial bets of $50 and $100; each player's balance is also restored to $20,000 at the start of each new hand.

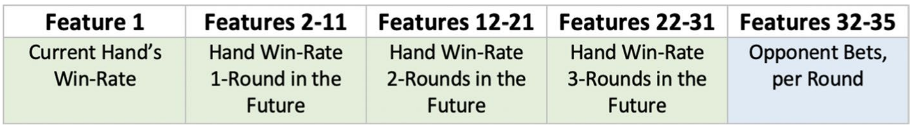
2.2 Interpretable feature representation

As is always the case in machine learning, the way the data is represented to the learning algorithm is critical. As such, we experimented with a variety of representations, with the goal of mapping a poker game-state to a vector of features that captures information in the game effectively, efficiently, and interpretably.

To transform a game-state to a vector, we use the fact that the state of a game of poker may be split into two components: the cards consisting of the players' private hand and the publicly visible cards, as well as the betting sequence of each player throughout the game so far. Therefore, representing a game-state involves both representing the cards and representing the betting history. The final feature representation for a game-state is simply the concatenation of the two vectors representing the card and betting history. Figure 1 illustrates the representation for a Pre-Flop hand, and the following two subsections provide a detailed explanation of this representation.

2.2.1 Card representation

Here, we present a novel approach to representing the public and private cards visible to a player as a set of interpretable features. Traditionally, representations are used in which strategically similar hands are grouped together by using K-Means Clustering on the current and potential equities of the hands (Ganzfried & Sandholm, 2014; Li et al., 2020). There is an extensive and rich history of literature on abstraction for Poker, covering all aspects of such abstractions (from actions to card representations): see (Brown & Sandholm, 2015; Brown et al., 2015; Ganzfried & Sandholm, 2013; Gilpin et al., 2008, 2007; Gilpin & Sandholm, 2006, 2007a, b, 2008). However, these lead to a representation that is fairly un-interpretable, because knowing which bucket a hand belongs to gives us little information about the hand itself. For instance, there is no natural ordering of the buckets, so having a hand that corresponds to bucket, say, 9000 of 10,000 does not indicate that the hand has a high win-rate relative to those in other buckets. It would also be very



Feature 1	Features 2-11	Features 12-21	Features 22-31	Features 32-35
Current Hand's Win-Rate	Hand Win-Rate 1-Round in the Future	Hand Win-Rate 2-Rounds in the Future	Hand Win-Rate 3-Rounds in the Future	Opponent Bets, per Round

Fig. 1 Visualization of the feature representation for a Pre-Flop hand. In total, this includes 35 features—4 pertaining to the betting history, and 31 pertaining to the public and private cards

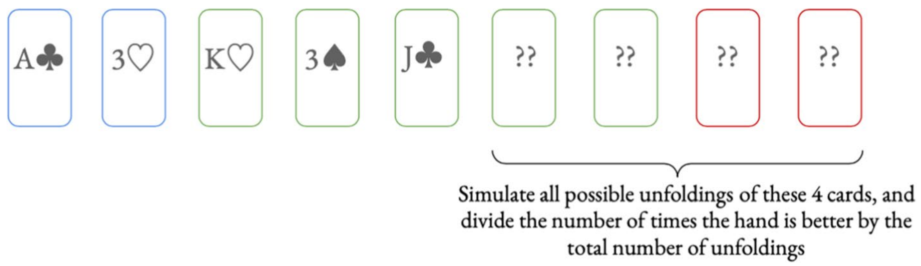

A♣
3♡
K♡
3♠
J♣
??
??
??
??
Simulate all possible unfoldings of these 4 cards, and divide the number of times the hand is better by the total number of unfoldings

Fig. 2 Visualization of calculating the equity of a hand, specifically during the Flop. The blue-outlined cards represent the player's private hand, the green-outlined cards represent the public cards, and the red-outlined cards represent the opponent's private hand (Color figure online)

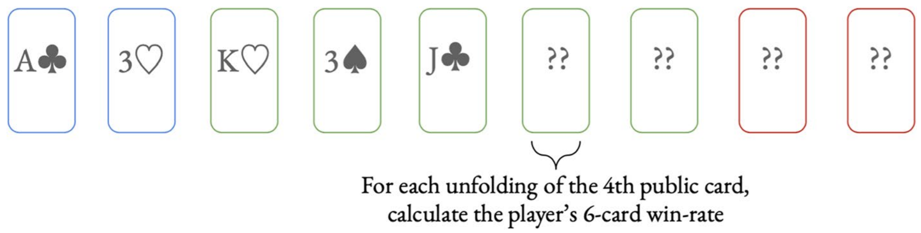

A♣
3♡
K♡
3♠
J♣
??
??
??
??
For each unfolding of the 4th public card, calculate the player's 6-card win-rate

Fig. 3 Visualization of calculating the equity of a hand $n = 1$ rounds into the future, specifically during the Turn. The blue-outlined cards represent the player's private hand, the green-outlined cards represent the public cards, and the red-outlined cards represent the opponent's private hand (Color figure online)

difficult for a human to sift through the card groupings, given the combinatorially large number of card possibilities, which is why we believe our card representation is easier to understand. These facts, combined with the one-hot nature of this representation, results in a poor feature representation for interpretable machine learning algorithms like Decision Trees, which may ask only a small number of questions before making a prediction. Our representation, defined below, seeks to solve all of these shortcomings.

First, keeping in-line with prior literature (Ganzfried & Sandholm, 2014), let us define the equity of a poker hand as the probability of it winning against a uniformly random drawing of any remaining public cards and the opponent's private cards, plus one half the probability of tying. For instance, the probability of a player winning against a uniform drawing of opponent and public cards, on a Pre-Flop with two kings as their private cards is 82.5%.

Given this definition of equity, we use this number as the first feature in our representation of the cards in any game-state.

Furthermore, a natural extension of this definition is to calculate the equity on future rounds of the poker game. In doing so, we would incorporate information regarding the player's hand's potential to improve or worsen as the rounds progress. To accomplish this for $n \geq 1$ rounds into the future, we enumerate every possible public hand reveal over the next n rounds, and calculate the associated equities for each of those scenarios using the previous definition of equity. Note, then, that the equity n rounds into the future is no longer a single number but rather a histogram of equities—with each data point in the histogram representing the equity of a possible card unfolding n rounds in the future.

For example, consider that the current round is the Flop, so that there are 3 public cards as illustrated in Figs. 2 and 3. A player's private hand consists of $A♣$ and $3♡$, and the public cards are $K\heartsuit$, $3\spadesuit$, and $J\clubsuit$. As Fig. 2 displays, we may calculate the hand's equity by enumerating all $\binom{47}{2} * \binom{45}{2}$ unfoldings, counting the number of unfoldings in which we end with a better hand than our opponent or tie, and dividing by the number of possible unfoldings. To calculate the equity on the next round (the Turn), Fig. 3 shows that we would enumerate all possible 47 revealings of the 4th public card, calculate an equity for each of those 47 scenarios, and store these in a histogram of equities. Finally, to calculate the equity 2 rounds into the future (the River), we would enumerate all possible $1081 = \binom{47}{2}$ revealings of the 4th and 5th public cards, calculate a equity for each of those 1081 scenarios, and store these in a histogram of equities.

Given these definitions, a natural way to compile all this information into a feature representation for a particular hand is to include its current equity, as well as its equities on all future rounds. However, since the histograms for future-round equities contain many values, we extract deciles from each histogram and use these values as features. Therefore, our card representation for a Pre-Flop hand contains 31 values—1 for its current equity, and 10 decile values for each of the 3 equity histograms on each of the 3 future betting rounds. Similarly, the card representation for a Turn hand contains 11 values—1 for its current equity, and 10 decile values for the equity histogram of the final betting round.

Finally, we also highlight that extracting deciles from each histogram helps reduce the game to a size feasible for the CFR self-play algorithm; see Sect. 2.3 Counterfactual Regret Minimization for details on feasibility. This is because many hands will have identical current equities and deciles for future-round equities. Also, then, varying the information extracted from each histogram (for example, extracting quantiles or ventiles) will also vary the degree of card-reduction and thus game-size reduction. In summary, this reduction is similar in result to the card-bucketing schemes used by prior poker bots, which serve to reduce the game-size as well.

2.2.2 Betting history representation

The second part of the feature representation involves extracting features pertaining to the betting history of the poker match. It is critical to also store this information, so that the agent may better understand game-play and in particular learn bluffing patterns based on how the opponent has bet. As such, we represent opponent bets with 4 numbers, each corresponding to the amount bet by the opponent on one of the four betting rounds. Note, if a round has not yet been reached (for instance, the River if the current game-state lies in the Pre-Flop), then the opponent bet for that round is 0.

We also point out that prior approaches to betting history representation (which have always been for Neural Networks) have also included features such as binary numbers indicating whether the opponent has bet, or even explicitly enumerating the opponent's bet within each round if multiple bets occurred (Brown et al., 2019; Li et al., 2020). While this provides additional betting features, including this more fine-grained information led to poorer HUNL Poker performance in our non-Neural Network-based bots, and a very marginal increase in performance for our Neural Network-based bot. As such, we found that our compact, 4-number betting history representation was the most effective.

2.3 Counterfactual regret minimization

2.3.1 Notation

We can visualize all possible future trajectories of a game of HUNL Poker as a tree, where the nodes of the tree represent various states in the game, and the branches emanating from each node as the possible decisions that can be made from each state. Note that there are three actors present in this tree: the two players, as well as the dealer. For example, at the beginning of the game, we are situated at the root of the tree and it is the dealer's turn. From this state, the dealer hands two cards to each of players one and two. As there are $\binom{52}{2} * \binom{50}{2}$ different ways that this allocation could be made, we will have a total of $\binom{52}{2} * \binom{50}{2}$ branches emanating from the root node of our tree. Once the cards have been dealt, we will follow that branch corresponding to the dealt cards to a new state of the game where both players each have two cards. In this new state, player one must now make a decision. The options available to them are to check, fold, or raise by some legal amount; again, there will be branches emanating from this state encoding each of those possible decisions. As before, once a decision has been made, we will follow the corresponding path to a new state of the game where it is now player two's turn. This process will continue until we arrive at the end of the game, which is represented via a terminal node in the tree. In this way, the entire evolution of a game can be encoded via a tree capturing the various decisions of the players and the dealer.

Formally, we let H denote the set of all possible states of the game, i.e., the set of all nodes in the tree, and $h \in H$ denote a particular state. Moreover, let $Z \subset H$ denote the set of terminal nodes in this game tree and $z \in Z$ denote a particular terminal state. Associated with each terminal state is a payout for each player i, which is given by the function $u_i : Z \rightarrow \mathbb{R}$. In other words, $u_i(z)$ gives the payout for player i in terminal state z.

Next, we define a strategy σ_i for player i as an assignment of probabilities to each possible move that the player may take at every possible game node. This, in other words, assigns a probability to each branch in every node in which it is player i's turn. For instance, if a particular strategy assigns probability 0.5 to folding and 0.5 to checking at a node h, then the player will fold 50% of the time and check 50% of the time, and never do any of the other potential actions. As such, specifying σ_i completely determines how player i will act at any position in the game. A strategy profile σ is a set of strategies, one for each player in the game.

Furthermore, due to the imperfect-information nature of HUNL Poker, some game-nodes will be indistinguishable from a particular player's perspective. For instance, assume at the start of the game player 1 is dealt $A♠$ and $K♠$, while player 2 is dealt $2♠$ and $3♠$. If player 2 had alternatively been dealt $5♡$ and $7♠$, both situations would be indistinguishable from player 1's perspective, as they do not see player 2's cards. Thus, even though both drawings would correspond to two different game nodes, player 1 cannot distinguish the two scenarios. So, we define an infoset (short for information set) I_i as a set containing all game-nodes indistinguishable from each other by player i. From the example just given, an infoset would contain all

$$\binom{50}{2}$$

nodes in which player 1 was just dealt $A♠$ and $K♠$ (so, each game-node in this infoset differs by the cards dealt to player 2).

Next, denote $v_i^\sigma(h)$ as the expected value of future rewards for player i in node h when all the players play according to σ. So, the higher $v_i^\sigma(h)$, the more rewarding h is expected to be for player i. Similarly, we can extend this to an infoset by defining the expected value of future rewards for player i for an infoset I_i as $v_i^\sigma(I_i)$.

Finally, we let $\pi^\sigma(h)$ denote the probability of reaching h under strategy σ. For a given node h, this is calculated as the product of the probabilities of taking each branch leading from the root to h. Additionally, it is useful to decompose $\pi^\sigma(h)$ into each actors' (the two players and the dealer) contribution to this probability. So, let $\pi_i^\sigma(h)$ refer to the probabilities coming from player i's decisions (i.e., branches), and $\pi_{-i}^\sigma(h)$ refer to the probabilities from all other actors, so that $\pi^\sigma(h) = \pi_i^\sigma(h) * \pi_{-i}^\sigma(h)$. For example for a given node h, $\pi_i^\sigma(h)$ is calculated as the product of the probabilities of taking each branch leading from the root to h in which it was player i's turn; $\pi_{-i}^\sigma(h)$ is simply the product of all remaining branches from the root to h. Finally, for an infoset I_i, we define $\pi^\sigma(I_i) = \sum_{h \in I_i} \pi^\sigma(h)$.

Please see the Appendix for a detailed worked example employing this notation.

2.3.2 The algorithm

Counterfactual Regret Minimization (CFR) is a popular iterative algorithm for finding a Nash equilibrium strategy profile, which is the strategy at which neither player can benefit from playing differently (Zinkevich et al., 2008). Therefore, by finding or approximating a Nash equilibrium, our poker agent can play HUNL Poker with extreme skill.

While many variants of CFR exist, for simplicity we first describe the simpler, vanilla algorithm and then specify modifications we used in the following subsection. At a high level, the algorithm works by repeatedly traversing the game tree (in essence, repeatedly playing poker) under a strategy profile, and accumulating "regrets". Conceptually, these regrets are a numeric value representing how much you regret not taking a particular action. Each iteration over the game tree accumulates new regrets, allowing the algorithm to re-weight and refine the strategy profile. Eventually, this strategy converges to a Nash equilibrium strategy through a weighting scheme, described below.

Formally, let t be the current iteration of the algorithm. First, we define the instantaneous regret of a move a_i at I_i as $r^t(I_i, a_i) = \pi_{-i}^{\sigma^t}(I_i) * (v_i^{\sigma^t}(I_i, a_i) - v_i^{\sigma^t}(I_i))$, which conceptually is the difference in reward between always choosing a_i versus playing according to σ^t at I_i, weighed by $\pi_{-i}^{\sigma^t}(I_i)$. Then, the counterfactual regret for infoset I_i for action a_i on an iteration T is $R_i^T(I_i, a_i) = T^{-1} \sum_{t=1}^T r_i^t(I_i, a_i)$, which is the sum of instantaneous regrets on all prior iterations of the algorithm for that particular infoset and action.

With these definitions, the CFR algorithm generates a strategy for each iteration based off of the accumulated regrets. Formally, player i's strategy on iteration $t+1$ is

$$\sigma_i^{t+1}(I_i, a_i) = \begin{cases} \dfrac{max\{0, R_i^t(I_i, a_i)\}}{\sum_{\alpha_i \in I_i} max\{0, R_i^t(I_i, \alpha_i)\}}, & \text{if } \sum\limits_{\alpha_i \in I_i} max\{0, R_i^t(I_i, \alpha_i)\} > 0, \\ \dfrac{1}{|\{a_i\}_{a_i \in I_i}|}, & \text{otherwise.} \end{cases}$$

Therefore, we see that the algorithm favors actions we regret not having taken (i.e., which would have led to greater rewards). Note, since on the first iteration no regrets have been observed, the initial policy followed by both players is uniformly random over the actions.

Following this definition, the CFR algorithm traverses the entire game tree for HUNL poker at each iteration, and then updates and records the strategies for both players. Note, however, that these iterative strategies do not converge to a Nash equilibrium strategy, but rather their average does. Formally, we define the average strategy $\overline{\sigma}_i^T$ as

$$\overline{\sigma}_i^T(I_i, a_i) = \frac{\sum_{t=1}^T \pi_i^{\sigma^t}(I_i) * \sigma_i^t(I_i, a_i)}{\sum_{t=1}^T \pi_i^{\sigma^t}(I_i)}.$$

In summary, the CFR algorithm iterates over the game tree, accumulates regrets and thus refines the average strategy. We may then attempt to learn the average strategy by training a machine learning algorithm on this data; see details in Sect. 2.4.

2.3.3 Computational optimizations for HUNL poker

In large imperfect-information games such as HUNL Poker—which has over 10^{161} possible game-states (Johanson, 2013)—it is impractical to store a strategy for the entire game and intractable to run CFR on the full game. To resolve these difficulties, we employ a series of tactics that either reduce the number of possible game-states or reduce the number of iterations needed for the CFR algorithm to converge to a desired proximity from a Nash equilibrium:

1. We use a variant of CFR called Discounted CFR, which empirically has been shown to accelerate the CFR algorithm by about two orders of magnitude (Brown & Sandholm, 2019). This variant differs from vanilla CFR by assigning less weight to the regrets accumulated in the earlier iterations. This significantly accelerates the empirical convergence, as earlier iterations are not nearly as valuable as information acquired later, and often weigh down changes in the strategy after many iterations. This variant works particularly well in games containing extremely poor action choices—and HUNL Poker is clearly such a game. We additionally included common techniques such as pruning, as well as the optimizations presented in Johanson et al. (2011). Overall, we estimate that these modifications improved the run-time of the CFR self-play by several orders of magnitude.
2. We reduce the number of potential actions our Poker agent may make. To motivate this, consider that in HUNL Poker there are often over 10,000 actions a player may make at any point in the game. However, considering every possible action is entirely unnecessary. For instance, betting \$141 versus \$142 in a game will often not result in a noticeable change in the outcome of the game. As such, we limit our agent to several choices of action at each round. The actions included were determined after analyzing actions taken by professional poker players and prior prizewinner poker agents. Overall, this reduces the number of game-states in HUNL Poker by many orders of magnitude, specifically by reducing the branching factor of nodes.
3. We group together game-states in which the agent's hands are strategically similar. For instance, being dealt an A♣ 5♠ during the preflop would be strategically similar to being dealt an A♡ 5♣. This grouping is typically done by placing strategically similar poker hands into the same buckets (Ganzfried & Sandholm, 2014; Li et al., 2020). However, this results in an un-interpretable representation of the cards, necessitating a new approach. Our interpretable feature implementation—outlined in Sect. 2.2—presents a novel representation that inherently performs such grouping. Overall, this reduces the number of game-states in HUNL Poker by many orders of magnitude, specifically by reducing the branching factor of chance nodes in the game tree.

2.4 Learning algorithms

We employ and compare a suite of learning algorithms to learn the average strategy found by the CFR self-play algorithm. The learners are purposefully chosen to range from high to low interpretability, so that we may analyze the trade-off between interpretability and performance of the Poker agent, see Sect. 3.

For each of the learning algorithms below, we train a model on data corresponding to many different HUNL Poker game-states represented via the interpretable feature representation outlined previously, and associated labels dictating the action the agent should take in each scenario, as learned by the CFR self-play.

Furthermore, we note that four models are trained for each of the learning algorithms employed—one for each of the four rounds of HUNL Poker. For instance, we produce four Neural Networks for the Neural Network agent, each of which dictate how to act on a particular round of HUNL Poker. We determined that this was beneficial for three reasons:

- First, our interpretable feature representation produces a different number of features for each round (for instance, 35 features on the Pre-Flop, and 15 features on the Turn). Since each of the learning algorithms employed learns a mapping from $\mathbb{R}^p$ to $\mathbb{R}$ for p fixed features, we need to train a separate model for each round.
- Second, the resulting agents achieved significantly better performance when decisions came from four models instead of one (in which missing features were filled with zeroes). This is likely due to the fact that different rounds of poker require different game-play; for instance, the bot may need to bluff significantly more on the Flop than on the River. As such, it would be difficult for a single model to learn all of the round-based nuances, in addition to the already difficult task of learning HUNL Poker strategies. Furthermore, we emphasize each of the four models can act as both the big blind and small blind player on its corresponding round. We experimented with creating 8 models (one for each round, and whether the player was a big/small blind) but did not see a reasonable increase in performance; using more models for a single agent also detracts from the interpretability.
- Third, by performing better and by learning how to play well in each of the rounds, the interpretable learning algorithms—CART and OCT—provide us with significantly richer and clearer strategies of how to play HUNL Poker. In addition to this, we may refer to any one of the four decision trees to improve their skills for a particular round.

2.4.1 CART

Classification and Regression Trees (CART) are an interpretable learning algorithm proposed in Breiman et al. (1984). For many years, CART was the leading method of generating a decision tree. The algorithm works by taking a greedy, top-down approach of determining the decision tree partitions. Specifically, the CART algorithm starts from the root node and determines which predictor to split on by solving an optimization problem; it then divides the point and recursively applies these steps to the two child nodes.

A primary advantage of using decision trees—in this case CART—is that they are highly interpretable, which will be vital for learning good strategies from the poker agent. However, a major shortcoming of decision trees produced by CART is that they generally achieve lower predictive accuracy than methods such as Neural Networks or Boosted

Trees, both discussed below. This lower performance is particularly prevalent in deep trees—which we train here in-order to learn the poker strategies—as the greedy nature of the CART algorithm begins to struggle to continue learning at larger depths. As such, we also employ a specialized alternative, Optimal Classification Trees, that generally perform significantly better than CART while maintaining its interpretability; see discussion below.

Finally, we note that the CART decision trees were trained with a max-depth of 15, which we determined struck a good balance between performance and interpretability. While deeper trees likely would have performed better, interpretability suffers significantly due to the exponential-increase in tree size.

2.4.2 Optimal classification trees

Optimal Classification Trees (OCT) are a learning algorithm developed by Bertsimas and Dunn (2017); see also Bertsimas and Dunn (2019). In particular, this learning algorithm uses mixed-integer optimization techniques to generate the entire decision tree in a single step—thus generating the tree in a non-greedy manner, unlike CART, and allowing each split to be determined with full knowledge of the other splits. As a result, this algorithm creates decision trees that significantly outperform the CART method and yield comparable performance to algorithms such as Boosted Trees and Random Forest, all while maintaining high interpretability. This high performance and high interpretability will be vital for the OCT's resultant poker interpretations, as it will clearly display meaningful and good poker strategies. Consider that if this method were not capable of great performance, its high interpretability would be useless, as it would display poor poker strategies.

As with CART, OCT produce a highly interpretable model, and Fig. 4 provides a demonstration of this interpretability by portraying a simple OCT for HUNL Poker Pre-Flop gameplay. Note, this example is only for illustration purposes and is not meant to demonstrate good poker strategy; see Sect. 5 for discussion of this.

In Fig. 4, the OCT model first assesses the magnitude of the opponent's bet, and proceeds to the left or right based off of whether is is greater than $12,700. If not, then the decision tree dictates the player call; otherwise, we proceed to the right child. There, the tree determines whether the player's current hand has a equity of greater than 0.48. If so,

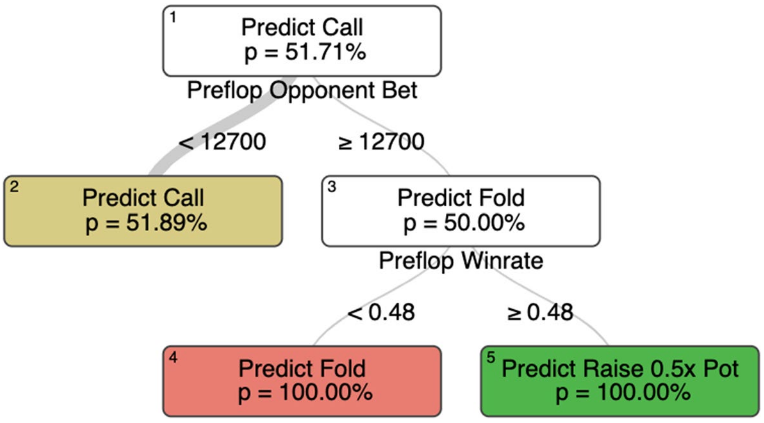


Fig. 4 Illustration of the interpretability of optimal classification trees. Predictions in non-terminal (leaf) nodes indicate what the agent thinks is the best action up to that point in the tree

then the player raises; otherwise, they fold and leave the poker game. Overall, then, we see this very simple model finds that the magnitude of the opponent's bet is the most relevant predictor of action in the Pre-Flop—it may then refine its choice of action based off of the equity of the hand.

As with CART, we determined that training OCT with a max-depth of 15 was the best balance of performance and interpretability. Also, the complexity parameter for split penalization and min-bucket parameter were both tuned via grid-search.

Finally, a noticeable trade-off with OCT is that training-time is longer than CART due to the complexity of solving the mixed-integer optimization problem. This is particularly true for deep trees—which we employ in this study—because of the exponential growth in the decision variables underlying the optimization problem. To speed this up, we take advantage of warm-starting the problem's solution. Specifically, we train the tree on increasingly larger subsets of the data (1/16 of the data, then 1/8 of the data, and so on) until a tree is trained off of the entire data-set; in doing so, we are able to warm-start the solution of the larger data-subset's model with the solution of the prior model. We estimate that this improved training time by several factors.

2.4.3 Extreme gradient boosted trees

To further reduce the bias of decision trees we employ Extreme Gradient Boosted Trees (XGBoost). This learning algorithm iteratively fits a sequence of decision trees, where the model at time step t is fit not only on the original data but also on the residuals of the model at time step $t - 1$. As such, the model focuses attention of trees further down in the sequence on the errors and shortcomings of the trees before them, which generally creates highly performant models. However, since the resulting model is a linear combination of thousands of decision trees, a critical trade-off of this approach is that the model is not interpretable.

The XGBoost models were tuned over a suite of hyper-parameters such as learning rate and the trees' max-depth. As is typical, the models performed better at larger depths, so the XGBoost models used had a maximum depth of 12.

2.4.4 Neural networks

Finally, we used a Feedforward Neural Network, which is the model that is furthest to the right on the interpretability-performance curve. As discussed in the Introduction, a Neural Network is the traditional method employed in machine learning-based poker agents and in similar areas like deep reinforcement learning. This method takes inspiration from the way the human brain works, by interconnecting a large number of nodes over a sequence of layers via a series of weights and activation functions. Unfortunately, despite the strong performance of Neural Networks, the complex structure of the model combined with the larger number of parameters present in it results in an un-interpretable model.

Like XGBoost, the Neural Networks were trained over a large number of hyper-parameters such as the number of layers, number of nodes per layer, and activation function used in each hidden layer. Ultimately, the Neural Networks performed best with seven hidden layers of 512 nodes each, using a RELU activation function for each hidden layer, and a soft-max activation function for the output layer. .

3 The Effect of the Learning Algorithm

For all results involving game-play in this paper, we benchmark our HUNL Poker agents against Slumbot, which is a very strong HUNL Poker bot and the winner of the most recent Annual Computer Poker Competition in HUNL. Because of its strong performance, this bot has been recently used as a benchmark for new HUNL Poker agents (Brown et al., 2018). We also emphasize that Slumbot was the best agent that is available for free-access, despite the existence of stronger poker AI agents such as Libratus and DeepStack, as mentioned previously. In addition, Slumbot, which is publicly available for play at slumbot.com, was produced by a variant of CFR self-play called Targeted CFR (Jackson, 2017) and uses a stored table for its average strategy, which is based on the standard bucketing abstractions discussed in Sect. 2.2—and as opposed to our approach of learning the average strategy via a learning algorithm.

In Table 1, we report the winnings in mbb/g from playing HUNL Poker with each pair of poker agents. Note, the averages are derived from 150,000 games of HUNL Poker, and include standard deviations. Also, each entry in Table 1 is presented from the perspective of the method in the 1st column—for example, 2.3 ± 1.5 indicates that the NN agent beats the XGBoost agent at that rate. We emphasize that the results in the 2nd column show performance comparisons between our poker agents and Slumbot; then, the 3rd, 4th, 5th, and 6th columns show performance comparisons directly between our poker agents.

From Table 1, we see that our Neural Network, XGBoost, and OCT-based HUNL Poker agents are able to outplay Slumbot by an impressive amount. For instance, the Neural Network is able to win almost \$5 per game against Slumbot on a \$100 big-blind and \$50 small-blind, which represents over a 6.5% averaged-return per game. Furthermore, we see Neural Networks yield the strongest bot, followed closely by XGBoost; OCT also yielded a strong player, while also providing important interpretability. Finally, note that CART yielded the weakest player, and is the only learning algorithm that resulted in poorer play against Slumbot.

We also see from Table 1 that OCT is an impressively efficient learning algorithm. Firstly, the OCT agent is able to play significantly better than CART agent, despite both agents being trained to the same depth and having the same final structure—that is, a decision tree. We therefore see that creating the tree to optimize over all possible node splits is a valuable trait. Secondly, while the OCT agent does not outplay the Neural Network-based agent, it plays almost as well despite only having a small fraction of the number of parameters that the Neural Network has. Specifically, while the depth 15

Table 1 Learning algorithm comparison—mbb/g over 150,000 games of HUNL Poker for each poker agent pairing

	SLUMBOT	NN	XGBOOST	OCT	CART
BLUEPRINT	19.9 ± 5.5	2.6 ± 1.0	4.1 ± 2.0	12.8 ± 3.5	23.6 ± 6.5
NN	18.4 ± 4.0		2.3 ± 1.5	6.4 ± 2.0	21.0 ± 3.5
XGBOOST	16.1 ± 3.5			4.5 ± 2.5	18.4 ± 3.0
OCT	9.8 ± 4.0				13.5 ± 4.0
CART	− 8.9 ± 4.5				

Presented averages are in games with a \$100 big-blind, as outlined in Sect. 2.1, and include standard deviations. The winnings presented come from the corresponding row-based agent's perspective

OCT contained up to 131, 069 parameters (one for each of its nodes and one for each of its branches), the Neural Network contains over 10 times more—1, 591, 296 parameters, one for each of its nodes and one for each of its connections. This again highlights the parameter-efficiency of the OCT method. In addition to this, we reiterate that the OCT has the valuable property of being highly interpretable, while the Neural Network does not.

4 The effect of self-play time

In Fig. 5, we depict the evolution of the XGBoost poker agent's performance as a function of the time the CFR self-play algorithm was run; we expect similar trends for the other tree methods, which would derive from the same average strategy. Specifically, the XGBoost player was periodically retrained according to the continuously refined average strategy; its performance was then measured by HUNL game-play against Slumbot. Note, the average winnings are reported from 50,000 games.

A few interesting observations are worth mentioning. First, the average winnings per game improve over time. This makes sense, as the CFR algorithm's average strategy is highly random at first, and gets refined overtime. We also see the benefits of the discounted regret weighting (i.e., using the Discounted CFR variant), as the XGBoost agent sees strong performance improvements very quickly. Second, after about 14 days, we observe that our poker bot advances to the point where it is now tied with the Slumbot. Beyond this point (i.e., after about two weeks), the improvement continues, though now at a lower rate, and after an additional two weeks the improvement appears to plateau at a average level of 4.3 dollars per game. While running the CFR for longer may have resulted in marginal improvements in performance, the trade-off between time and performance improvements rises sharply.

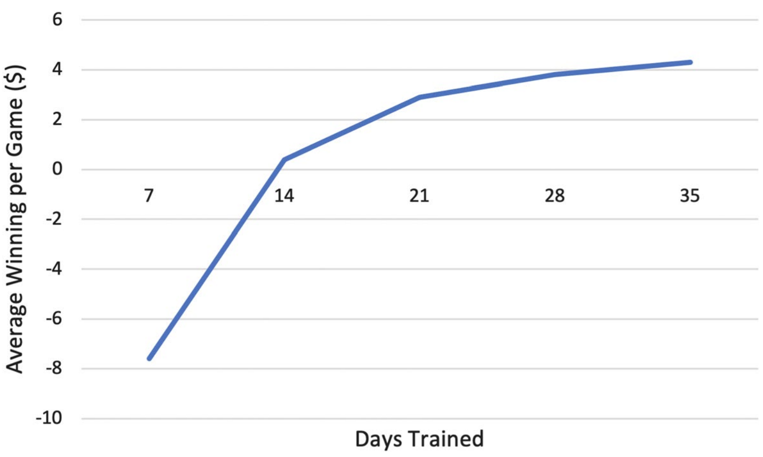


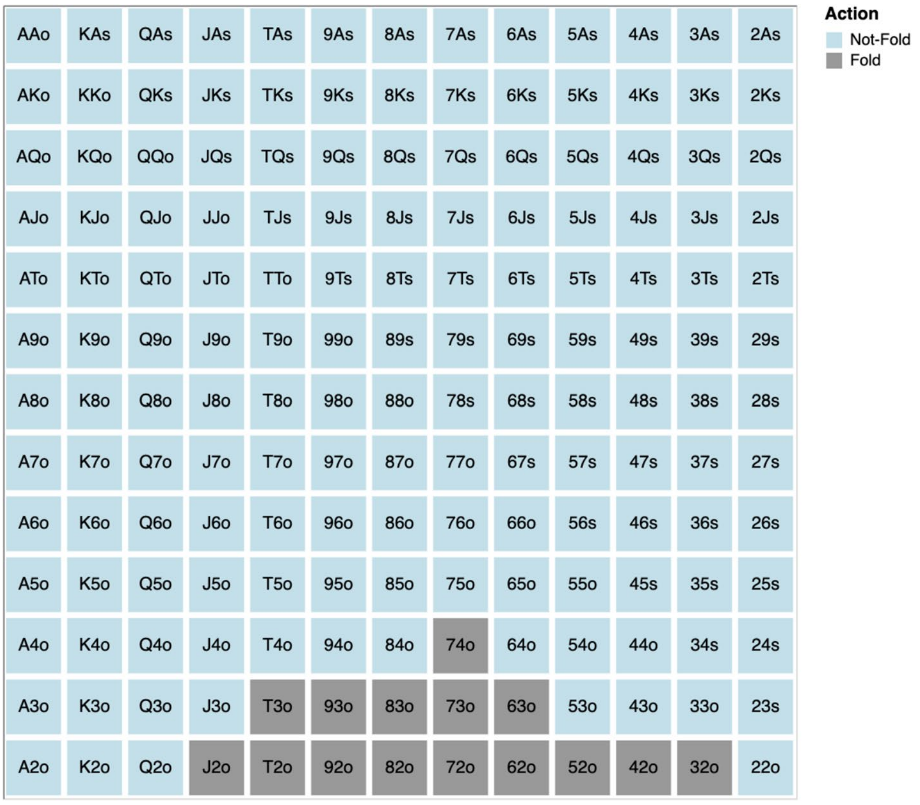
Fig. 5 Graph comparing the average winnings per game, when the XGBoost-based agent plays against Slumbot, as a function of the number of days of self-play. The average winnings derive from HUNL gameplay with standard buy-in's presented in Sect. 2.1, and are averages over 50,000 HUNL Poker games. Computation is done on computers from the MIT Sloan Cluster and MIT Supercloud Computing Center

Overall, the graph suggests that by refining the average HUNL poker strategy via CFR self-play for a little over a month, we arrive at a bot that is capable of very strong performance.

5 Can machines help interpretability?

In this section, we address perhaps the most important aspect of our work: the interpretability of our model. We begin with one of the most popular strategy visualization tools employed in the poker: the opening-move chart extracted from our model and shown in Fig. 6. We then present in Figs. 7 and 8 a novel way of representing a poker strategy using Optimal Decision Trees, which as we highlight below, enjoy numerous benefits over the static opening-move chart.

In Fig. 6, we present an opening-move chart for the OCT agent, which illustrates with which private cards the agent will immediately fold. Generating such a chart is very common when attempting to visualize the strategy employed by a poker agent. Note that an "o" in the cell means the two cards are of different suit, and an "s" means the two cards are



AAo	KAs	QAs	JAs	TAs	9As	8As	7As	6As	5As	4As	3As	2As
AKo	KKo	QKs	JKs	TKs	9Ks	8Ks	7Ks	6Ks	5Ks	4Ks	3Ks	2Ks
AQo	KQo	QQo	JQs	TQs	9Qs	8Qs	7Qs	6Qs	5Qs	4Qs	3Qs	2Qs
AJo	KJo	QJo	JJo	TJs	9Js	8Js	7Js	6Js	5Js	4Js	3Js	2Js
ATo	KTo	QTo	JTo	TTo	9Ts	8Ts	7Ts	6Ts	5Ts	4Ts	3Ts	2Ts
A9o	K9o	Q9o	J9o	T9o	99o	89s	79s	69s	59s	49s	39s	29s
A8o	K8o	Q8o	J8o	T8o	98o	88o	78s	68s	58s	48s	38s	28s
A7o	K7o	Q7o	J7o	T7o	97o	87o	77o	67s	57s	47s	37s	27s
A6o	K6o	Q6o	J6o	T6o	96o	86o	76o	66o	56s	46s	36s	26s
A5o	K5o	Q5o	J5o	T5o	95o	85o	75o	65o	55o	45s	35s	25s
A4o	K4o	Q4o	J4o	T4o	94o	84o	▨ 74o	64o	54o	44o	34s	24s
A3o	K3o	Q3o	J3o	▨ T3o	▨ 93o	▨ 83o	▨ 73o	▨ 63o	53o	43o	33o	23s
A2o	K2o	Q2o	▨ J2o	▨ T2o	▨ 92o	▨ 82o	▨ 72o	▨ 62o	▨ 52o	▨ 42o	▨ 32o	22o

Action
- ▢ Not-Fold
- ▨ Fold

Fig. 6 Opening-move chart of the OCT agent, portraying whether the agent will immediately fold when given each possible private card in the Pre-Flop. Note, the trailing "o" in each cell means the cards are off-suited (i.e., of different suits) and a "s" means the cards are suited (i.e., have the same suit)

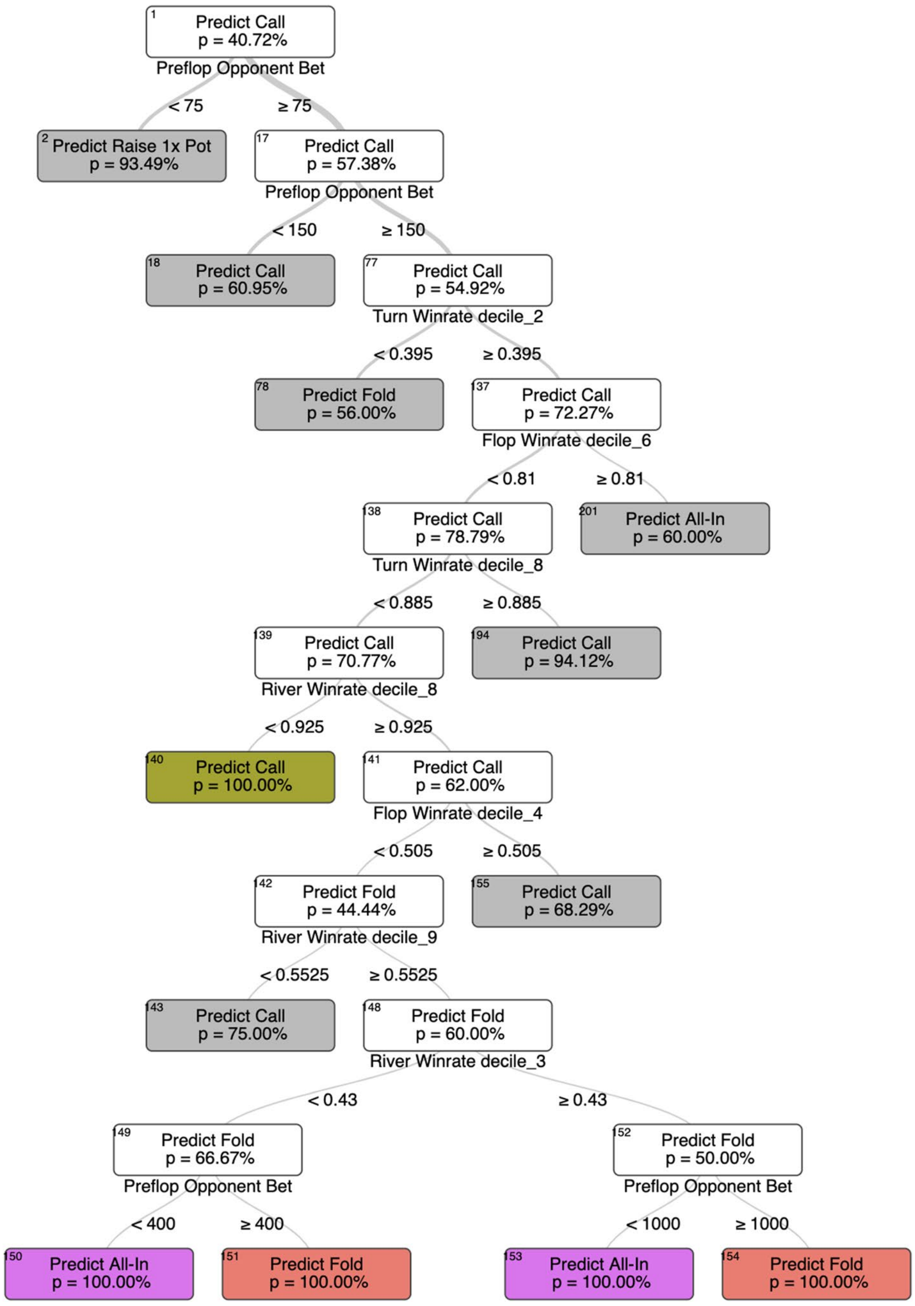

1 Predict Call p = 40.72%
Preflop Opponent Bet
< 75
≥ 75
2 Predict Raise 1x Pot p = 93.49%
17 Predict Call p = 57.38%
Preflop Opponent Bet
< 150
≥ 150
18 Predict Call p = 60.95%
77 Predict Call p = 54.92%
Turn Winrate decile_2
< 0.395
≥ 0.395
78 Predict Fold p = 56.00%
137 Predict Call p = 72.27%
Flop Winrate decile_6
< 0.81
≥ 0.81
138 Predict Call p = 78.79%
201 Predict All-In p = 60.00%
Turn Winrate decile_8
< 0.885
≥ 0.885
139 Predict Call p = 70.77%
194 Predict Call p = 94.12%
River Winrate decile_8
< 0.925
≥ 0.925
140 Predict Call p = 100.00%
141 Predict Call p = 62.00%
Flop Winrate decile_4
< 0.505
≥ 0.505
142 Predict Fold p = 44.44%
155 Predict Call p = 68.29%
River Winrate decile_9
< 0.5525
≥ 0.5525
143 Predict Call p = 75.00%
148 Predict Fold p = 60.00%
River Winrate decile_3
< 0.43
≥ 0.43
149 Predict Fold p = 66.67%
152 Predict Fold p = 50.00%
Preflop Opponent Bet
< 400
≥ 400
Preflop Opponent Bet
< 1000
≥ 1000
150 Predict All-In p = 100.00%
151 Predict Fold p = 100.00%
153 Predict All-In p = 100.00%
154 Predict Fold p = 100.00%

Fig. 7 Visualization of the interpretability of the optimal classification tree for preflop play. Note, gray nodes are compressed sections of the tree, to maintain visual simplicity. Here, we illustrate a branch corresponding to bluffing

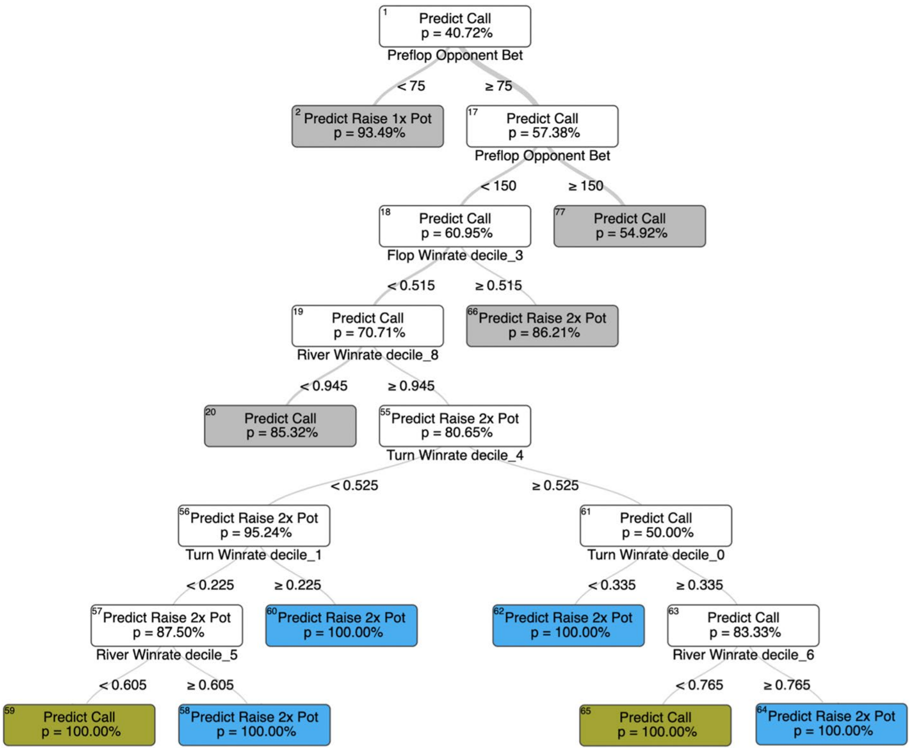

1 Predict Call p = 40.72%
Preflop Opponent Bet
< 75
≥ 75
2 Predict Raise 1x Pot p = 93.49%
17 Predict Call p = 57.38%
Preflop Opponent Bet
< 150
≥ 150
18 Predict Call p = 60.95%
77 Predict Call p = 54.92%
Flop Winrate decile_3
< 0.515
≥ 0.515
19 Predict Call p = 70.71%
66 Predict Raise 2x Pot p = 86.21%
River Winrate decile_8
< 0.945
≥ 0.945
20 Predict Call p = 85.32%
55 Predict Raise 2x Pot p = 80.65%
Turn Winrate decile_4
< 0.525
≥ 0.525
56 Predict Raise 2x Pot p = 95.24%
61 Predict Call p = 50.00%
Turn Winrate decile_1
Turn Winrate decile_0
< 0.225
≥ 0.225
< 0.335
≥ 0.335
57 Predict Raise 2x Pot p = 87.50%
60 Predict Raise 2x Pot p = 100.00%
62 Predict Raise 2x Pot p = 100.00%
63 Predict Call p = 83.33%
River Winrate decile_5
River Winrate decile_6
< 0.605
≥ 0.605
< 0.765
≥ 0.765
59 Predict Call p = 100.00%
58 Predict Raise 2x Pot p = 100.00%
65 Predict Call p = 100.00%
64 Predict Raise 2x Pot p = 100.00%

Fig. 8 Visualization of the interpretability of the optimal classification tree for preflop play. Note, gray nodes are compressed sections of the tree, to maintain visual simplicity. Here, we illustrate the logical decision-making of the tree

identically suited. For example, “T3o” means the agent was given a 10 and a 3 with different suits (perhaps a 10♡ and 3♣). From Fig. 6, we see that the agent will not immediately fold a vast majority of the time—in fact, if it is given suited cards it will never fold for the first move.

While such opening-move charts are very popular in the poker literature, and in fact can be interesting to examine, relative to a decision tree they are limited in several important dimensions. For one, it does not illustrate how the agent will play for the rest of the Pre-Flop or the rest of the HUNL Poker game. In addition, it does not provide intuition or reasoning as to why the agent folds in some situations and does not in others, which we argue is vital for educating oneself about good poker play and refining one’s own intuition. In contrast to this, Figs. 7 and 8 display portions of the OCT agent’s decision tree, which not only portray strategy for the entire Pre-Flop, but also provide explanation for the decisions by tracing down the decision tree—as the example with Fig. 4 demonstrated.

Before analyzing the Optimal Decision Trees, we reiterate that (as discussed in the Section 2.4) each non-leaf node in the decision tree poses a binary question about a particular feature; see also Section 2.2 for the specifics of the the card and betting history representation. In Fig. 7, we visualize a section of the OCT that corresponds to bluffing—that is, when the player tries to deceive their opponent by raising with weak cards. To see this, consider that the decision path to the purple and red nodes reveals a situation in which the agent does not have notably strong cards. Specifically, Node 39 along that path tells us that the agent will win a card showdown over 92.5% of the time with good end-game table cards (an 80-th percentile situation), but loses over 57% of the time with mildly poor end-game table cards (a 30-th percentile situation). Given that agent does not have notably strong cards, Nodes 49 and 52 determine whether or not the agent should bluff based on how confidently the opponent bets. If the opponent has bet too much (>$400 in Node 49 or >$1000 in Node 54), then the agent folds; otherwise, the agent goes All-In since it feels the opponent did not bet too confidently. In summary, the OCT agent is weighing the risks and benefits of taking a bluff on these paths. It bluffs if the opponent has not played too confidently, or if it still has a strong chance to recover if the opponent calls the bluff.

In Fig. 8, we visualize a section of the OCT that highlights its logical decision-making. Specifically, we see how it is weighing equity features. In Node 63, for instance, it asks if the agent's cards have a strong end-game equity, by examining the 60-th percentile of the current hand's River equity histogram. If this equity is high (76.5% to be exact), the agent raises by twice the amount in the pot; otherwise the agent calls. Identical decisions and logical weightings occur throughout this branch, again highlighting the OCT's rational decision-making process.

Ultimately, Figs. 7 and 8 also highlight the edge our card representation gives over past machine-learning based poker agents—which, as discussed previously, either use non-sparse card ranges as features or simply use card rank and suit embeddings as features (Moravčık et al. 2017; Brown et al., 2019). In comparison, our feature representation is interpretable and more compact (using at most 35 features to represent a poker game-state), leading to a richer representation. For instance, this is evident by noticing that the OCT heavily uses the equity decile features in determining its decisions—especially in situations that may require bluffing—as it is able to extract information on how the opponent sees the agent's card potential. This ultimately allows the agent to more efficiently analyze its current state, and make an effective decision. Finally, we emphasize that while this feature representation and methodology provides a significantly more interpretable picture for humans, it is still not fully interpretable due to the use of deciles (rather than less granular groupings) and deep decision trees.

6 Conclusion

In this paper, we propose a novel framework for constructing interpretable, machine-learning based HUNL Poker agents, and use it to build a poker bot that is simultaneously capable of outperforming the strong poker AI Slumbot and highly interpretable to humans, which may analyze the strategies it employs.

The nature of the advancement is two-fold: first, the proposal of a novel, compact, and easy-to-understand game-state feature representation for HUNL Poker. Second, the use of globally optimal decision trees, paired with a counterfactual regret minimization (CFR) self-play algorithm, to train our poker bot in an interpretable fashion. Already with the Optimal Decision Trees, we achieve world-class performance, i.e., a poker bot capable of outperforming the Slumbot by an average of $2.6 per game on a standard $100 big-blind. When this learning algorithm is replaced by a non-interpretable one, such as XGBoost and Neural Networks, our edge over the current champion grows even larger.

While it is exciting to produce a HUNL Poker agent capable of such performance, the most exciting and important property of our framework is its interpretability. This is because, as we demonstrated in Sect. 5, our agent can produce human-readable printouts of the strategies it uses. These strategies can then be studied by human opponents to inform their own poker strategies. Most important of all, as control steadily expands its sphere of influence, eventually outside the realm of recreational games and onto real-life problems, society will demand these systems be capable of explaining their reasoning and decision-making processes. And ours is a framework that sets the groundwork for doing exactly that.

Appendix: CFR notation example

Here, we provide a detailed worked example of the notation outlined in Sect. 2.3, on a simple form of poker known as Kuhn Poker. In this form of poker, there are only 3 cards—a Jack, a Queen, and a King. Also, there is only 1 round of betting, and each players action is restricted to a check, call, fold, or betting \$1. Due to the simplicity of this form of poker, the entire game tree is shown in Fig. 9, which allows us to exemplify the notation.

In the game tree, each node represents a game-state in which an actor (player 1, player 2, or the dealer) needs to take an action, and the branches emanating from that node are the valid actions. For example, at the root we see the dealer first deals a card to player one—thus, 3 branches emanate from the root, one for each card. At the leaves of this tree, we find the end-game states; the numbers corresponding to each of these are the payout function u_1's rewards to player 1.

Defining a strategy σ_i for player i would require assigning probabilities to each action in each game-state at which player i acts. For example, to define a strategy for player 1, we would assign probabilities to each of the grey branches in the game tree.

As defined in Sect. 2.3, an infoset I_i is a grouping of game-nodes that are indistinguishable from player $i's$ perspective. Recall that these arise due to the imperfect information in Poker—that is, that the opponents cards are private. In Fig. 9, the grey nodes with stars in them represent one infoset for player 1. Namely, player 1 may not distinguish being in these nodes, as they cannot see that player 2 has a queen in one versus a king in the other.

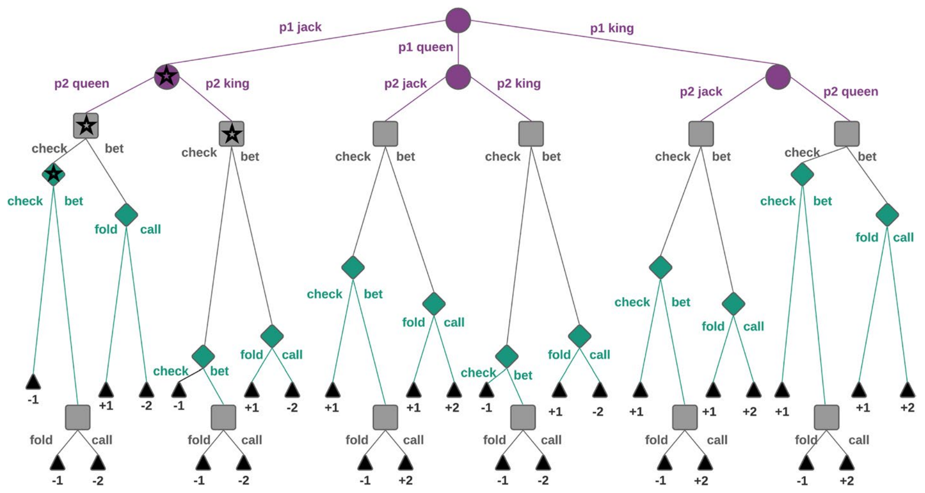


Fig. 9 Game tree of Kuhn Poker. The purple, grey, and green nodes correspond to states in which the dealer, player 1, and player 2 act, respectively. The black nodes are end-game nodes (Color figure online)

Finally, recall that $\pi^{\sigma}(h)$ denotes the probability of reaching h under strategy profile σ. Under the game tree above, for instance, $\pi^{\sigma}(h_1)$ in which h_1 is starred purple node in Fig. 9 would simply equal the probability that player 1 is dealt a jack—1/3. Alternatively, $\pi^{\sigma}(h_2)$ in which h_2 is starred green node in Fig. 9 is product of the probability player 1 is dealt a jack, player 2 is dealt a queen, and player 1 checks at the left grey starred node. As introduced in Sect. 2.3, we could decompose $\pi^{\sigma}(h_2)$ into $\pi^{\sigma}_1(h_2)$ and $\pi^{\sigma}_{-1}(h_2)$. The former would be calculated as player 1's contributions to $\pi^{\sigma}(h_2)$, which is the probability of player 1 checking at the left grey starred node; $\pi^{\sigma}_{-1}(h_2)$ would be calculated as all other actors' contributions to $\pi^{\sigma}(h_2)$, which in this case would be the probability player 1 is dealt a jack and player 2 is dealt a queen.

Author contributions DB had responsibility (a) of formulating the problem, (b) introducing the idea of interpretability and (c) writing and editing the manuscript. AP had responsibility (a) of formulating the problem, (b) coding the overall algorithm and (c) writing and editing the manuscript.

Funding Open Access funding provided by the MIT Libraries.

Data and code availibility Data and code is available on request from the authors.

Declarations

Consent to participate Both authors consent to participate in this research

Consent for publication Both authors consent to publications

Ethics approval Not applicable

References

Bertsimas, D., & Dunn, J. (2019). *Machine learning under a modern optimization lens*. Dynamic Ideas LLC.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth and Brooks.

Brown, N., Ganzfried, S., & Sandholm, T. (2015). Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas Hold'em agent (Vol. 1, pp. 7–15).

Brown, N., & Sandholm, T. (2015). Simultaneous abstraction and equilibrium finding in games. In *IJCAI*.

Brown, N., Sandholm, T., & Amos, B. (2018). Depth-limited solving for imperfect-information games. In *Advances in neural information processing systems* (Vol. 31, pp. 7663–7674). Curran Associates, Inc.

Brown, N., Lerer, A., Gross, S., & Sandholm, T. (2019). Deep counterfactual regret minimization. *Conference on Machine Learning, 97,* 793–802.

Brown, N., & Sandholm, T. (2018). Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science, 359*(6374), 418–424.

Brown, N., & Sandholm, T. (2019). Solving imperfect-information games via discounted regret minimization. *Proceedings of the AAAI Conference on Artificial Intelligence, 33,* 1829–1836.

Dunn, D. B. J. (2017). Optimal classification trees. *Machine Learning, 106*(7), 1039–1082.

Ganzfried, S., & Chiswick, M. (2019). Most important fundamental rule of poker strategy. *CoRR*, arxiv:1906.09895.

Ganzfried, S., & Sandholm, T. (2010). Computing equilibria by incorporating qualitative models. *AAMAS*.

Ganzfried, S., & Sandholm, T. (2013). Action translation in extensive-form games with large action spaces: axioms, paradoxes, and the pseudo-harmonic mapping. In *IJCAI*.

Ganzfried, S., & Sandholm, T. (2014). Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *Proceedings of the twenty-eighth AAAI conference on artificial intelligence* (pp. 682–690). AAAI Press.

Ganzfried, S., & Yusuf, F. (2017). Computing human-understandable strategies: deducing fundamental rules of poker strategy. *Games, 8*(4), 2017. https://doi.org/10.3390/g8040049.

Gilpin, A., & Sandholm, T. (2006). A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the 21st national conference on artificial intelligence* (Vol. 2).

Gilpin, A., & Sandholm, T. (2007a). Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *AAMAS '07*.

Gilpin, A., & Sandholm, T. (2007b). Lossless abstraction of imperfect information games. *J. ACM, 54,* 25.

Gilpin, A., & Sandholm, T. (2008). An experimental comparison using poker: Expectation-based versus potential-aware automated abstraction in imperfect information games. In *AAAI*.

Gilpin, A., Sandholm, T., & Sørensen, T. (2007). Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the 22nd national conference on artificial intelligence - Volume 1* (Vol. 1, pp. 50–57).

Gilpin, A., Sandholm, T., & Sørensen, T. (2008). A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs (Vol. 2, pp. 911–918). https://doi.org/10.1145/1402298.1402350.

Jackson, E. G. (2017). Targeted cfr. In *AAAI workshops*.

Johanson, M. (2013). Measuring the size of large no-limit poker games. In *CoRR*.

Johanson, M., Waugh, K., Bowling, M., & Zinkevich, M. (2011). Accelerating best response calculation in large extensive games (pp. 258–265).

Li, H., Hu, K., Ge, Z., Jiang, T., Qi, Y., & Song, L. (2020). Double neural counterfactual regret minimization. In *International conference on learning representations*.

Moravčık, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., et al. (2017). Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science, 356*(6337), 508–513.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science, 362,* 1140–1144.

Zarick, R., Pellegrino, B., Brown, N., & Banister, C. (2020). Unlocking the potential of deep counterfactual value networks. In *CoRR*, arXiv:abs/2007.10442.

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2008). Regret minimization in games with incomplete information. In *Advances in neural information processing systems* (Vol. 20, pp. 1729–1736). Curran Associates, Inc.