Imperfect Information Games: Modeling Responses to Irrational Behaviour using Deep Learning

Anonymous CVPR submission

Paper ID ****

Abstract

This work explores the impact of incentive incompatibility in the game of Leduc poker. We model a set of traditional poker playing profiles varying in computational complexity and play these profiles against DeepStack, an AI agent that optimizes towards the Nash Equilibrium. Once we identify the subset of profiles that fares best against DeepStack, we then modify DeepStack to be open-loop to learn an opponent's playing style and then simulate incentive incompatibility with the intent to undermine DeepStack's ability to successfully learn the opponent's strategy.

1. Introduction

Modeling strategies in imperfect-information games, such as Heads-Up No-Limits (HUNL) Poker and Leduc Hold'em, is challenging as the full state (the cards of the opponent) is unknown. Because of this challenge, imperfect-information games require more complex reasoning as each decision during play depends on a probability distribution of the opponent's private information, which can only be discerned through the opponent's action history.

Recent breakthrough work (see [6], [2]) has found success in optimizing towards Nash-equilibrium strategies, yet previous work has not exploited the average player's tendency to deviate from rationality in the context of games [5], exhibiting changes in risk-tolerance over the course of play [4]. The objective of this work is to explore incentive incompatibility by identifying a set of traditional poker playing profiles that fare best against an open-loop DeepStack, and then integrate these profiles in a player that demonstrates incentive incompatibility, undermining DeepStack's ability to learn the opponent's playing style.

2. Related Work

Counterfactual regret minimization is a foundational approach to improve performance in imperfect information games. Additionally, two notable approaches have been explored to overcome the challenges of states lacking unique values.

2.1. Counterfactual Regret Minimization

Recursive reasoning approaches to solve imperfect-information games have been explored since 2008. Counterfactual regret minimization (CFR) exploits the degree of incomplete information in an extensive game and uses self-play to recursively reason.

Self-play enables an AI agent to adapt its playing strategy against itself over many iterations, leveraging "regrets" of prior decisions to inform future play. The AI agent's goal is to optimize its game tree traversal based on its opponent to converge on the Nash equilibrium. The limitation of this approach comes when dealing with larger game trees where such computation during play is too expensive. In these cases, abstraction is used to simplify the game which is then approximately solved via tabular CFR. Two types of abstractions have been historically employed – information abstraction, where information sets are bundled, and action abstraction, where a subset of actions are removed from the game tree model.

Recent work has explored an innovation on the tabular CFR approach that introduces Deep CFR, a form of CFR that uses function approximation with deep neural networks to approximate the tabular CFR behavior without the need for game abstraction [1].

2.2. DeepStack & Libratus

AI DeepStack's technique modifies the definition of state into a joint belief state across all players [6], proving inferior to the top two HUNL Poker players. DeepStack's architecture consists of a standard feedforward neural network with seven fully connected hidden layers, each with 500 nodes and parametric rectified linear units (32) as output. This network is then embedded in another network imposing counterfactual regret minimization to satisfy the zero-sum property. Libratus approximates the optimal strategy for the earlier part of HUNL Poker and then approximates the best play in the latter parts of the game. This approach

enabled Libratus to devise better strategies for particular subgames while fitting these strategies within the overarching blueprint. Libratus' architecture differentiates from DeepStack due to it's self-improver module that facilitates an open-loop to progressively learn from playing history.

2.3. DeepStack-Leduc

Extending the approach of [2], Schmid et. al re-implemented the DeepStack AI agent for HUNL Leduc Hold'em. This adaptation to a simplified game laid the groundwork for rapidly exploring the performance impact of introducing irrationality in play.

3. Problem Formulation

3.1. Nash Equilibrium Strategy

The Nash Equilibrium strategy represents a profile of strategies in imperfect-information games in which no player can improve by shifting to a different strategy. Mathematically, a Nash Equilibrium in principle can be found by solving simultaneous equations:

$$x_1^* = \max_{x_i \in X_i} f_1(x_1^*, x_2)$$

$$x_2^* = \max_{x_i \in X_i} f_2(x_1, x_2^*)$$

Where x_i^* represents the optimal action Player i should take to converge on the Nash equilibrium of the game, x_i represents the action taken by Player i, X_i represents the set of all possible strategies Player i can take, and f_i is the function that maximizes the pay-off to Player i in response to the opponent's action.

When the opponent plays the Nash Equilibrium strategy, irrationality as well as psychological tactics during play (i.e. bluffing) may undermine the opponents ability to discern the true value of the players hand. This leads to suboptimal decision-making as the opponent attempts to maximize the payoff based on the assumption that both players have aligned incentives to converge on the Nash equilibrium. Irrationality in play enables a player to influence and mislead their opponent.

3.2. Leduc Hold'em

To explore semi-rational and psychologically manipulative AI playing agents, a simplified Poker game will be used known as Leduc Hold'em. When compared to HUNL, Leduc Hold'em has a smaller game tree and reduces playing time, drastically reducing time to learn.

Leduc Holdem is a two-player poker game with a six-card deck consisting of Jacks, Queens and Kings (two of each). The deck is shuffled prior to each hand and the game begins with a one-chip ante and a private card dealt to each player. An initial betting round transpires followed by the flop, where a single public card is shown. A second betting round transpires before the showdown where players reveal their private cards. The player with a pair wins the pot and in the case where neither does, the highest private card wins. If both players have a pair, then they split the pot.

3.3. Limitations of Rationality in Poker

In traditional poker, understanding your opponent's risk-tolerance and playing strategy is critical to winning. Common strategies in poker include bluffing, evaluating risk, and limiting exposure to exploitation. These are all computationally-simple attempts at a similar resolving process to what DeepStack simulates.

In this work, we explore how a player should behave when faced with an open-loop DeepStack opponent. An average player, with no access to a computer, is certainly unable to play Nash equilibrium strategy. We also explore how an assumption of rationality in poker can lead to ineffective training when incentive incompatibility among players is present.

4. Methodology

Our approach will explore rules-based, probabilistic and intelligent, semi-rational AI agents. Each agent will compete with a pre-trained, open-loop DeepStack. There are four types of personalities in poker categorized based on their level of risk-tolerance and their level of aggression (i.e. Passive vs. Aggressive). Within these broader definitions exist two playing profiles known as Rocks

Player Profile	Type	Description
Rocks	Rules-based	Risk-averse player with strict folding logic.
Passive-Rocks	Rules-based	Risk-averse player with nuanced folding logic.
Mild Adaptive Rocks	Intelligent	Player that adjusts behavior based on success variance for state
Strong Adaptive Rocks	Intelligent	Player that adjusts behavior based on success variance for state
Randomized Bluffer	Random	Risk-tolerant player that randomly bluffs in play.
Semi Bluffer	Probabilistic	Risk-tolerant player that raises based on the probability of the opponent folding.
Aggressive Bluffer	Probabilistic	Player that liberally raises based on the probability of the opponent folding.

Table 1. List of the irrational playing profiles explored.

and Passive Rocks that will be modeled in this approach. Simultaneously, we also explore three broadly defined bluffing behaviors in poker, differing on bluffing logic.

4.1. Player Profiles Explored

Table 1 outlines an initial list of player profiles explored in this project which may be expanded further, with a focus on modeling risk-aversion and bluffing behaviors typically found in poker.

Rocks represents a very risk-averse (i.e. "tight") personality type that folds in most cases when a high value card or potential pair does not exist. In the strategy space of Leduc Hold'em, this translates to playing only when the private card is a King. An example scenario of this player is found in Figure 1. Passive Rocks will call or bet in some cases beyond the the highest value card. In the strategy space of Leduc Hold'em, this translates to playing hands that consist of a Queen or King. Both of these traditional risk-averse playing profiles can be easily modeled using a rules-based approach. Extending this rules-based approach, the Randomized Bluffer will simulate a coin toss to determine whether to bluff or logically. Further profiles will be explored in this category of playing tactics.

4.2. Modeling the Bluffer Player Profiles

In order for bluffing players to know when a raise is a "bluff" (versus a rational play) they must have some idea of rational play. Thus, the logic for bluffing is built on top of DeepStack. The randomly-bluffing player uses re-solving to determine the probabilities of each action. If "raise" is a valid action, then, with probability .3, the player ignores the rationally-chosen action and does a large raise instead (and otherwise behaves rationally). The semi-bluffer uses probabilities of the opponent folding, based on the resolving of the game (the value estimates for the opponent when resolved). The semi-bluffer only irrationally raises if the estimated probability of the opponent folding is high.

4.3. Modeling the Semi-rational Player Profiles

Semi-rational player profiles maximize a utility function incorporating risk aversion on top of DeepStack as opposed to maximizing risk-neutral expected payoff. This incorporates traditional notions of risk-aversion from decision theory into strategy evaluation. To do this, instead of simply approximating the expected value of a given action in a given state, we also will calculate the variance of outcomes and vary risk-aversion level to gauge performance impact.

4.4. Modifying DeepStack to have an Open-Loop

Taking inspiration from Libratus' self-improver module, we will modify DeepStack's architecture to implement an open-loop system which will enable DeepStack to learn from prior playing history to optimize to an opponent's playing profile. In doing so, this will enable us to test incentive incompatibility during training when DeepStack attempts to learn on the set of modeled playing profiles that fare best.

4.5. Evaluation Criteria

Exploitability is the typical primary metric used to assess performance in this type of study. Exploitability of a strategy is defined as the measure of performance against a worst-case opponent, or distance from the Nash equilibrium strategy. Exploitability is quantified as the difference in value of the game to a player and the opponents payoff if the opponent plays best response. In addition to exploitability, given our agents deliberately do not optimize to Nash Equilibrium, we will also evaluate the average bankroll per hand and the average win rate per hands played.

5. Code/Preliminary Results

We expand upon two starter Github-repos, with code located at https://github.com/rawls238/deep_learning_project. The rational DeepStack player is cloned from the Leduc-DeepStack github [7]. The structure to create custom rule-based players in python, as well as evaluation criteria, is cloned from a final project from last year's class which pitted DeepStack against Libratus [3]. We alter code from DeepStack's re-solver to create the players built on top of DeepStack and implement our rule-based players in python from scratch. Instructions on how to play with these new players is included in the readme for our project. Thus far, we implemented three player profiles, which are listed in the readme along with some basic ones for testing purposes.

The preliminary results of these matches (against DeepStack) are located in the outputs folder of the github repo. Each opponent matching played 1000 games of Leduc against each other. The numbers in the output file are their total ending win/loss – so the higher the number, the better. These preliminary results show that, in order from best to worst matched against a closed-loop Deepstack, are:

1. Rocks (-6.1 chips per game average)
2. PassiveRocks (-47)
3. RandomizedBluffer (-67.8)
4. AlwaysCall (-89.6)
5. AllIn (-124)
6. RandomAction (-137.6)

References

[1] G. Brown, Lerer and Sandholm. Deep counterfactual regret minimization. *eprint arXiv:1811.00164*, 2018. 1

[2] N. Brown, T. Sandholm, and B. Amos. Depth-limited solving for imperfect-information games. *arXiv preprint arXiv:1805.08195*, 2018. 1, 2

[3] Echlin, Porubin and Dunn. Coms 4995 Final Project - Fall 2017. https://github.com/mp3242/coms4995-finalproj, 2018. Online; accessed 25 October. 3

[4] D. Eil and J. W. Lien. Staying ahead and getting even: Risk attitudes of experienced poker players. *Games and Economic Behavior*, 87, 2014. 1

[5] J. K. Goeree and C. A. Holt. Ten little treasures of game theory and ten intuitive contradictions. *American Economic Review*, 91(5):1402–1422, 2001. 1

[6] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017. 1

[7] M. Schmid. Deepstack for leduc hold-em. https://github.com/lifrordi/DeepStack-Leduc, 2017. 3

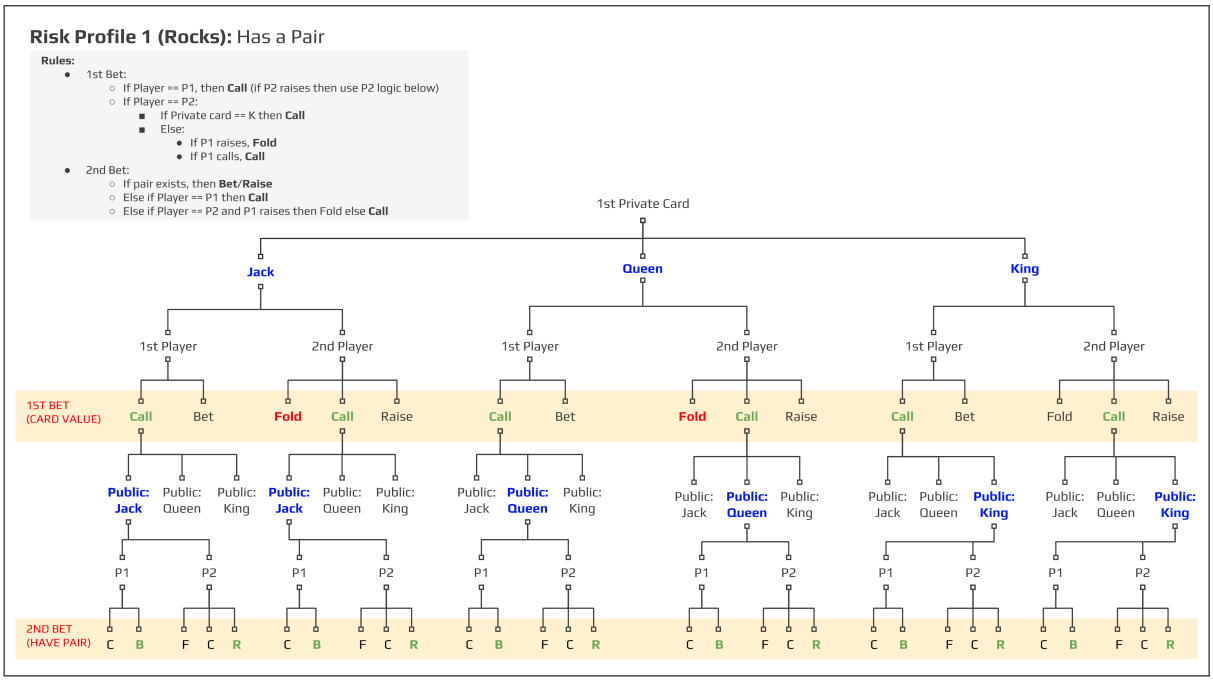

Figure 1. Example of a Rocks Player Game tree in the scenario where a pair exists.