Comparing Performance between Q-Learning and Fictitious Self-Play Trained on Neural Networks for No-Limit Texas Hold'em Poker

Stanford CS230 Project

Aaron Han
Department of Computer Science
Stanford University
than21@stanford.edu

Abstract

A combination of traditional self-play algorithms and neural networks to "solve" No-Limit Texas Hold'em Poker proved to become very potent against even the top human players. However, these agents are extremely computationally expensive and the resources created from these agents are also very expensive to access. This study attempts to build self-playing poker agents through reinforcement learning algorithms including Q-learning and Fictitious Self-Play with limited computing power that can be run on personal computers given sufficient time. Despite previous studies showing that Fictitous Self-Play is the superior algorithm than Q-learning for imperfect information games like No-Limit Texas Hold'em, Q-learning based agents have shown to outperform in a heads-up match. However, that is likely a result of the greediness of Q-learning, which could make it very exploitable for human players.

1 Introduction

Texas Hold'em Poker is becoming closer and closer to being solved. For humans playing the game, game theory optimal (GTO) has risen as the most ideal strategy to play poker. GTO essentially entails a strategy where a player has a ratio of actions (between folding/calling/raising/checking) at any stage of a poker game given the hand they have. [1]

Computers are aiding the progress on "solving" Texas Hold'em as well. In just 2017, two incredibly high-performing poker agents were announced for no-limit heads-up Texas Hold'em [2] [3]. In 2019, another agent capable of beating poker pro's in a multiplayer table was announced. [4] It is interesting to note that all of these agents were developed a learning method called counterfactual regret minimization (CFR) instead of reinforcement learning (RL), which is more frequently used to develop game-playing agents.

However, these types of information are incredibly difficult to access. GTO charts that humans can use in cash games or tournaments are very expensive. General GTO range charts not tuned for other players' tendencies could cost as much as $75 per year. The bots also have the power of generating its own range charts as well, but they are computationally expensive as they all have used academic grants (DeepStack, Liberatus, Pluribus) and/or have been backed by corporations (Pluribus).

As such, the goal of this study is to develop No-Limit Texas Hold'em agents that can be trained with significantly reduced costs and still produce adequate results.

2 Related Work

There is plenty of literature on how agents with different self-learning mechanisms have performed in various settings of poker. An agent named Libratus, backed by CMU, was trained using a new variant of counterfactual regret minimization to beat top human players in Heads-Up No-Limit Texas Hold'em. [2] DeepStack also utilized counterfactual regret minimization for Heads-Up No-Limit Texas Hold'em, but estimated the counterfactual regret values through a neural network instead of exploring every possible state space. [3] Another paper studied the performance between Deep Q-Learning and Non-Fictitious Self-play in a simplified version of Texas Hold'em and explained Q-Learning strategies are relatively exploitable due to the lack of stochastic processing involved to perform well in imperfect information games. [5]

3 Dataset (Gym)

Given that the project's scope is within reinforcement learning/self-learning, training a model based on a dataset is not completely necessary. While it is beneficial to have a set of data to train a supervised model to base the self-improving reinforcement learning agent – AlphaGo being the first to show the potency of this method – it is not an absolute requirement to have said data. [6] The data for poker hands in particular are very expensive as well as studying it is directly correlated to performance and monetary gain. As such, for the purposes of building a lower-cost poker agent, I have elected to not use any pre-existing data for this project.

The counterpart to the data necessary for this project is the "gym" environment. To build a heads-up no-limit poker environment, I had three options:

- Build my own gym
- Use RLCard (link)'s No Limit Hold'em implementation as a starting point for the gym
- Use neuron_poker (link)'s OpenAI-based implementation for the gym

Out of these options, I elected to utilize RLCard. Building my own gym that not only had the rules of No-Limit Hold'em fully implemented but also had the auxiliary functions necessary for a self-learning agent would be extremely time-consuming. I explored basing the gym on the neuron_poker library but quickly realized the limitations of OpenAI specifically for poker. OpenAI's custom gym requires a pre-set action and observation spaces. While having a specificed action space is fine – actions for No-Limit Texas Hold'em can be limited to check, fold, call, and raise x/y/z amount – the need for preset observation space is quite limiting for no-limit poker in particular. An absolutely necessary observation for any poker agent is the action history (history of check/call/raises until a particular point of a hand), and given the nature of No-Limit Hold'em, the length of the history can vary greatly hand-by-hand.

That being said, the base RLCard library still required significant adjustments to more closely imitate how No-Limit Hold'em is played. The logic for determining what legal actions a player can take given the stage of the hand (pre-flop vs. post-flop) and previous actions. For instance, you cannot check when the other player has raised, but the base RLCard library would allow this to happen. Furthermore, when an all-in and a call happens, the library originally evaluated the winner at the moment where the all-in happened. That is, if pocket 5's went all-in vs. Ace-King before any community cards were shown, the library would automatically evaluate the pocket 5's to win even though Ace-King statistically wins about 50% of the time with the runout of community cards. These two were the most egregious issues of the RLCard library, and they have been addressed along with some smaller issues.

4 Methods

4.1 Q-Learning

Q-learning is one of the more basic forms of reinforcement learning. It takes the state space, a set of actions, and the rewards associated with the actions. The goal of Q-learning is to maximize the sum of rewards given all the observations of the state space and the rewards resulting from the series of actions. Training a neural network is necessary to apply Q-learning to a game like No-Limit Heads-up

Texas Hold'em which has a state space up to 10^{17} possible states. [3] A 64x64 layer was used to build the neural network to estimate the reward values.

4.2 Fictitious Self-Play

Fictitious Self-Play (FSP) is a algorithm learning from self-play specifically designed for two-player zero-sum games. When used with a neural network, it trains two separate neural networks. The first is $Q(s, a|\theta^Q)$, which predicts action values from off-policy reinforcement learning memory which results in a greedy Q-learning algorithm. The second neural network is a supervised training network $\Pi(s, a|\theta^\Pi)$, which imitates the best response from training the reinforcement learning network. The agent trained using FSP mixes the strategy from both of these neural network during play. The strategy devised by these agents should have lower exploitability – a metric that measures how close a strategy is to nash equilibrium (GTO). [5] The architecture for both neural networks was 64x64 layers that takes in the state and the possible actions the agent could take.

4.3 Baseline & Improvements

Since the agents are being trained with self-play mechanisms, they must have a baseline to train against. However, there is no baseline to train against to begin with; as such, basic agents were trained using the two aforementioned algorithms against a completely random agent. After having these agents, the two algorithms were used again to train agents against the two basic agents, resulting in a total of 6 agents: two basic agents trained against random agents and four agents trained against the two basic agents.

5 Results & Discussion

6 different types of agents were trained using the aforementioned training methods and were compared against each other as well as a random agent. The following table is the average number of chips won per round through 100,000 hands. As the small blind and the big blind for these hands were 1 chip and 2 chips respectively, dividing the below figures by 2 would result in the number of big blinds won:

Agent type	vs. Random	vs. Q (random)	vs. FSP (random)	vs. Q (Q)
Q (random)	4.48292		0.91039	-1.33912
FSP (random)	5.15881	-0.91039		-2.96436
Q (Q)	5.08819	1.33912	2.96463	
Q (FSP)	2.81241	-6.2255	-1.78796	-2.73417
FSP (Q)	1.23403	-8.16672	-4.68493	-5.20085
FSP (FSP)	1.72537	-2.05518	-1.36847	-2.43409

Table 1: Performance of agents against each other

Agent type	vs. Q (FSP)	vs. FSP (Q)	vs. FSP (FSP)
Q (random)	6.2255	8.16672	2.05518
FSP (random)	1.78796	4.68493	1.36847
Q (Q)	2.73417	5.20085	2.43409
Q (FSP)		-2.19402	3.51143
FSP (Q)	2.19402		-1.12741
FSP (FSP)	3.51143	1.12741	

Table 2: Performance of agents against each other

Note that the notation of each agent is *Algorithm used (agent trained against)*. For instance, *Q (FSP)* is an agent trained using Q-learning against the basic FSP agent trained against a random agent.

Based on the payoffs, all agents have shown to at least outperform the random agents, which is promising given that. A more interesting observation is that Q-learning agents, especially the one trained with Q-learning against the base Q-learning agent, outperformed FSP in most cases. This is

surprising given that a previous paper has showcased that FSP eventually learns strategies that are less exploitable and closer to GTO than Q-learning. [5] This is most likely a result from Q-learning's greedy properties, which may have been trained to be extremely aggressive and cause the FSP algorithm to fold more often.

In order to explore the playing styles of the agents, I also played 20 30 hands against each and observed the results. The most noticeable trait shared across all agents was the frequency at which the agents went all-in. This most likely was a result of the positive and negative rewards being directly fed into the training models instead of being weighted. As winning all-ins have extremely high payoffs in poker, these agents learned to frequently move all-in. This would classify the agents as loose-aggressive in the poker playing style matrix:

Figure 1: Poker playing style matrix

While being aggressive is important for poker – being passive would mean that you are guaranteed to lose every time you don't have a good hand yourself – the frequency at which the agents went all-in was quite alarming. I personally prefer and play with a tighter and less aggressive style than the agents, meaning that I would only elect to raise before seeing the community cards only if I have a serviceable hand, after which I would bet aggressively. In contrast, the agents would frequently move all-in with hands like 42o pre-flop, which is a very poor hand as it is unsuited (less flush potential) and loses to almost every other combination of hole cards. A potential solution to help agents play tighter would be to penalize negative rewards more.

That being said, I did observe that the Q (Q) agent made the most sensible all-in's out of all of the agents that were trained. The following hand was an instance where the agent moved all-in where I would have as well:

I had the King and Queen of hearts while the agent had 6 and 7 of clubs. The agent raised first, after which I raised knowing I have a very good hand. The agent called, and the flop was 5 of clubs, 8 of clubs, and ace of hearts. I checked, the agent went all-in, and I folded given that I lose to any pair at the moment and do not have any strong draws (flush/straight). Even if I had an ace and called the all-in, the agent would still have close to 50% probability of winning the hand as it has an up-and-down (can land either a 4 or 9) straight flush draw, the strongest hand in poker. While it was pleasing to see the agent go all-in in a spot where it recognizes that it has a very strong draw and can force opponents with technically stronger opponents away, this was unfortunately one of the few instances where an agent made a good play.

Figure 2: Example of a hand vs. Q (Q) agent

6 Future work

6.1 Counterfactual Regret Minimization Agent

CFR is a self-learning algorithm in which "regret" for all actions in a possible set of actions is calculated for every state in a stochastic process. [7] Counterfactual regret is defined by the loss in reward by not taking certain actions. In a simple game of rock paper scissors where the rewards are -1, 0, 1 respectively for losing, drawing, and winning, the counterfactual regrets after playing scissor against a rock would be 1 for rock and 2 for paper. [7] The authors of DeepStack integrated this approach with neural networks to estimate these counterfactual regret values as No-Limit Heads-Up Poker's state space is way too large to store every counterfactual regret value for the range of actions given a state. [3] DeepStack was one of the first self-trained agents to heavily outperform top human players in No-Limit Texas Hold'em.

This project originally was centered around how an agent trained using the CFR algorithm, used to train DeepStack, would perform against agents trained using more tradition reinforcement learning algorithms for self-play such as Q-learning and FSP. However, I was unable to implement such an agent within the time constraints for this project. It would be possible and beneficial to explore the difference in performance between a CFR agent and the agents developed in this project provided more time.

6.2 Improved Data Logging & Metrics

Improving the way data is logged will allow for more metrics to be built to analyze the performance of the agents. Currently, the evaluation function does log actions made by an agent and only keeps track of the payoffs of each hand. Keeping track of these actions would allow for calculation of exploitability (how close an agent is performing to GTO) as well as range charts, although it would be rather computationally expensive.

7 Team member contributions

I have not been collaborating with anyone else.

References

[1] Game theory optimal (gto) texas holdem poker theory. In *Course Blog for INFO 2040/CS 2850/ECON 2040/SOC 2090*, 2021.

[2] Noam Brown and Toumas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. In *Science Vol. 359, No. 6374*, 2017.

[3] Matej Moravcik, Martin Schmid, Neil Burch, Villiam Lisy, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. In *Science Vol. 356, No. 6337*, 2017.

[4] Noam Brown and Toumas Sandholm. Superhuman ai for multiplayer poker. In *Science Vol. 365, No. 6456*, 2019.

[5] Jonathan Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. 2016.

[6] Google DeepMind. Mastering the game of go with deep neural networks and tree search. In *Nature 529*, 2016.

[7] Todd Neller and Marc Lanctot. An introduction to counterfactual regret minimization. 2013.