NN-based Poker Hand Classification and Game Playing

Gautam Bhat
Department of Computer Science
Boston University
Boston, MA 02135
gautam@bu.edu

Kaviarasan Selvam
Department of Physics
Boston University
Boston, MA 02135
kavi21@bu.edu

Veena Dali
Department of Neuroscience
Boston University
Boston, MA 02135
vdali@bu.edu

Abstract

Poker is a family of complex card games, each with a different set of rules. In our project, we utilize a simplified version of Texas Hold'em poker to build and train a poker bot that learns to classify hands and devises a playing strategy in order to be competitive player. We developed a system that uses a fully-connected neural network that can be trained to understand the patterns between the cards and the hands that can be formed from those cards. Thus, the trained neural network can be the foundation of a poker bot. Once it can accurately classify hands, we devise a playing strategy based on a number of parameters, and record our results. We name our poker bot, 'AlphaPo'.

1 Introduction

Poker consists of a variety of complex card games, each with a different set of rules. It is widely regarded as the standard benchmark for AI in the space of incomplete-information games, where the players do not have common knowledge of the state of the game at all times. Adding to the difficulty of the challenge poker poses as compared to complete-information games like chess or Go, it also has an element of betting and bluffing that makes it even more complicated. Moreover, the class of incomplete information games consists of most games with practical significance, of which poker exhibits rich features such as a broad range of human expertise to test against, and the aforementioned betting and bluffing scenarios. Therefore, it makes creating a poker bot to play against human players very difficult and an exciting problem to solve.

We focus on Texas Hold'em as the base variant of poker on which to model the poker bot as well as the playing strategy. In this game, each player is dealt two cards, one by one, followed by the dealer placing five cards face-down on the table. After the first round, the first three cards are turned face-side-up, followed by a card each in the subsequent rounds, making it a maximum of four possible rounds. Between rounds, all players (unless a player folded in a previous round) are given the opportunity to place bets (fold, call, or raise). The winning pot goes to the player still in the game with the strongest hand. The order of strength of hands (from weakest to strongest) is as follows: Nothing in Hand, One Pair, Two Pair, Three of a Kind, Straight, Flush, Full House, Four of a Kind, Straight Flush, Royal Flush.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

The goal of our project was to understand learning, playing, and complexity involved in modelling a poker bot using machine learning techniques. Our approach was to start creating the poker bot from the ground up, and adding layers of complexity as the project would progress. The playing strategy does not implement all rules of poker yet, such as betting and playing blinds. The approach is instead boolean, with the poker bot deciding whether to continue playing or to fold, instead of betting and raising certain amounts of money. We shall refer to this version of the game as experimental Texas Hold'em poker.

2 Related work

As mentioned before, solving poker and other incomplete-information games represents a problem in computer science that has lots of attention from the research point of view due to its practical importance in fields outside of AI, such as economics. Over the years, there have been many efforts to design poker bots, using techniques ranging from traditional neural networks and probability-based Bayesian networks to deep learning, game theory and evolutionary algorithms.

Yakovenko et al (2016) created a system, Poker-CNN, using Convolutional Neural Networks and rank three tensor-based representation to play three distinct games of poker, using the same machinery to learn patterns for three different poker games.

Just as recently, Heinrich and Silver (2016) employed a game-theoretic approach that involves deep reinforcement learning methods for learning approximate Nash Equilibria of an imperfect-information version of poker, Leduc poker. Their goal was to devise a scalable end-to-end approach for learning approximate Nash equilibria without prior knowledge. As extraordinary as their findings were, the software and source code required hardware and resources which was deemed as beyond the scope of this project for the time being, due to limited hardware and time. Many systems could potentially take several days to train even with GPUs. Therefore, instead of trying to improving on these, we decided aim to create our own poker bot that could be implemented with limited hardware and resources while not compromising on the accuracy and performance.

3 Data

A poker hand data set was obtained from UCI's Machine Learning Repository to train the neural network. The data set consisted of a poker hand of five cards, and the class the hand belonged to. As is the case with real-world probabilities of hands occurring, the data set of 25000 instances was largely biased towards 'High Card' and 'One Pair', having more than 20000 instances of those two alone (only 5 'Royal Flush' in comparison). In order for the neural network to successfully learn and classify hands, other hands' instances were added to remove this bias,increasing the data size to about 70000 instances. The format of the data was changed to be in line with the architecture of the neural network.

4 Methods

The first step in creating the poker bot is to train it to accurately identify hands, given a set of cards. The network used for this classification problem is a two-layer feedforward network, with a sigmoid transfer function in the hidden layer, and a softmax transfer function in the output layer. There are 85 input neurons, 50 neurons in the hidden layer, and 10 output neurons (each representing a type of poker hand). We use Scaled Conjugate Gradient algorithm to train this network. Once trained, weights of the network are taken and used to recognize hands during the playing strategy. The idea behind formulating this strategy is to write an algorithm to estimate the winning likelihood based on the cards that are dealt to the bot. This algorithm includes parameters that add weights to each hand, define a risk factor and a compare a threshold value, which together influence the bot's decision making. These are the important parameters and their assigned values.

Occurrence of each hand P contains the probability of each hand occurring in a real-world game of poker, ranging from 0.000154% for 'Royal Flush' to 50.1177% for 'High Card'. The probabilities are stored as a 10-element, rank one tensor.

Weight vector W is a 10-element, rank one tensor with weights assigned to each class, in order to calculate the value of the decision function.

Risk factor Ideally between 0 and 1, this dictates the overall value of the decision function, and symbolizes the amount of risk the bot is willing to take. Essentially, a higher risk factor will result in the bot choosing to continue playing with a weaker hand.

Estimated frequency distribution Another 10-element, rank one tensor, that estimates the likelihood of a hand coming up, given the current cards dealt to the bot. For the first turn, there are 2 revealed cards and 5 hidden cards, resulting in 2118760 possible card combinations in subsequent turns. During the second turn, 5 cards are revealed, and 2 are unknown, resulting in 1081 possible card combinations. Similarly, the possible number of card combinations in the third turn given 6 cards and 1 unknown, are 46. F is an estimated frequency distribution of each class of hands, which can be normalized to represent an analogous posterior probability.

Threshold value Threshold at each turn determines the numerical value the decision function needs to exceed for the bot to continue playing. If $D_f < threshold$, the bot shall fold.

These parameters combined are used to model the playing strategy.

$$Decision(Turn, Cards) = F(P, W, R_f, Threshold, F, NN)$$

The value of the decision function is computed as:

$$D_f = \frac{R_f}{C} \sum_{i=1}^{10} \frac{F_i W_i}{P_i}$$

where C is a constant term, predefined to normalize values of Df to a smaller range.

4.1 Playing strategy

The following pseudo code gives a brief overview of the algorithm and playing strategy:

```
{Generate/Deal Cards}
{Reveal two cards to bot}
turn_0
...Wait for opponent...
If decision(opponent) == 0
  Return                    %Opponent Folded
If decision(turn_0) == 0
  Return                    %Agent Folded
{Reveal next 3 cards}
turn_1
...Wait for opponent...
If decision(opponent) == 0
  Return                    %Opponent Folded
If decision(turn_1) == 0
  Return                    %Agent Folded
{Reveal next card}
turn_2
...Wait for opponent...
If decision(opponent) == 0
  Return                    %Opponent Folded
If decision(turn_2) == 0
  Return                    %Agent folded
{Reveal final card}
turn_3
```

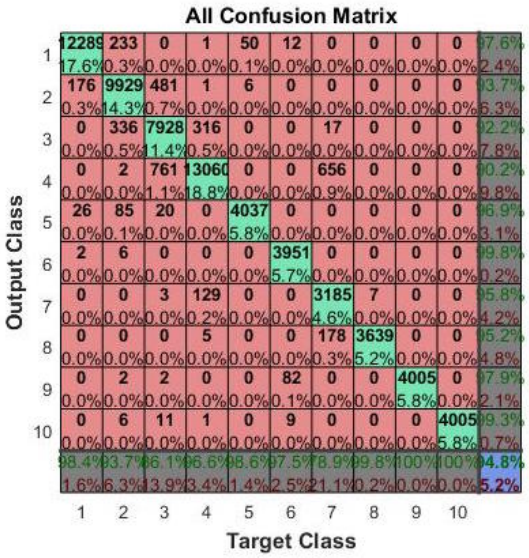

Figure 1: Confusion Matrix

Table 1: Outcome Summary

R_f	Wins	Total
0.1	3	15
0.5	5	15
0.8	10	15
1.0	8	15

```
...Wait for opponent...
If decision(opponent) == 0
  Return                    %Opponent Folded
If decision(turn_3) == 0
  Return                    %Agent Folded
{Compare Hands}
```

5 Results

After fixing the bias prior to training, the neural network was able to accurately classify 94.8% of the hands in the testing data. As can be seen from the confusion matrix in Figure 1, error in classifying several 'Three of a Kind' and 'Two Pair' was accountable for the loss in accuracy. The curve of receiver operating characteristic[1] in Figure 2 shows the network performs very well in classifying the hands correctly.

Outcomes of games were recorded to understand how the poker bot makes decisions as the risk factor is varied when playing against opposition. The results are summarized in Table 1.

It is, however, important to note that, winning at poker depends as much on the poker bot's cards and strategy as it does on the opponents. Also, No wins can mean a draw, a loss, or an early fold. Similarly, a win can also signify that the opponent folded first.

[1]ROC is a plot of the true positive rate (sensitivity) versus the false positive rate (1 - specificity) as the threshold is varied

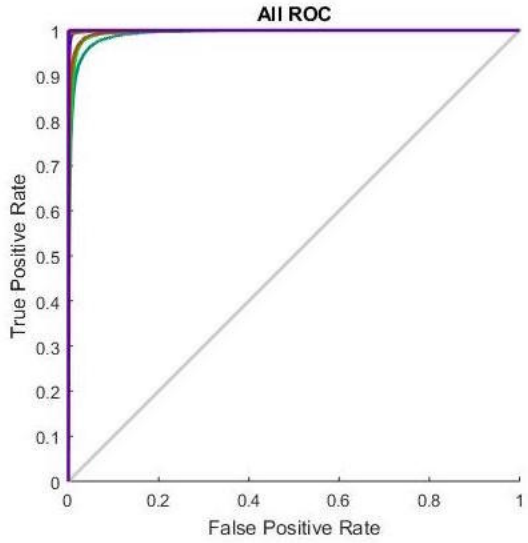

Figure 2: ROC

6 Discussion

From Figure 1, we see that the network has noticeable error when trying to classify 'Two Pair', 'Three of a Kind', and 'Full House'. This is due to the similarity of the hands in that many card combinations are similar, which tends to confuse the neural network. Regardless, the neural network does well in classifying most hands, despite a comparatively small data set. A more vast data set can help train the neural network to be more accurate, while removing the error of incorrect classification among hands, since that can be a major deterrent in making better decisions.

The poker bot took very little amount of risk when $R_f = 0.1$, as expected, and ended up folding too early even if the cards would have eventually won it the game. On the other hand, when $R_f = 1.0$, the player took too many risks, eventually losing out more games than when $R_f = 0.8$. This is essentially modelling different kinds of poker players based the risk they take, and also gives an option to adjust R_f in order to optimize the strategy. The same is the case with the other parameters, each yielding different kind of win-draw-loss ratios. The end goal would be to allow the bot to adjust these parameters on its own, learning from every game.

7 Future Work

There is a lot of scope for further development on this project, with more time and capable hardware required. As mentioned earlier, incorporating betting into the playing strategy should be the next major step in building a more competent poker bot, more aligned to the rules of a mainstream variant of poker. Another major aspect for improvement would be the playing strategy itself; a more dynamic strategy would be needed to automatically adjust parameters as the game progresses, as compared to having the same value of parameters throughout the game. There potential to incorporate Game Theory with reinforcement learning in the construction of a more refined playing strategy. The poker bot can be trained to play against itself, but it should also be configured to implement elements of Game Theory throughout the entire "learning process". This can be attained by recording the outcome of games as a tensor and creating a data set out of it, and feeding this information to a reinforcement learning strategy. Game-theoretic approaches can be coupled with the same, with the poker bot playing against multiple iterations of itself where each agent is playing its best move, leading to a Nash equilibrium.

As for the classification problems, it remains to be known if the neural network will be able to differentiate between 'Three of a Kind' and 'Full House' if the network is fed more data pertaining to those hand rankings.

Acknowledgments

We would like to thank Dr. Sang "Peter" Chin and Kun He for their dedicated support during the progress of this project.

References

[1] Cattral R., Oppacher F., & Deugo, D. Evolutionary Data Mining with Automatic Rule Generalization. Recent Advances in Computers, Computing and Communications, pp.296-300, WSEAS Press, 2002.

[2] Heinrich J. & Silver D. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. University College London, UK, 2016.

[3] Lichman, M. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2013.

[4] Yakovenko, N., Cao, L., Raffel, C., & Fan, J. Poker-cnn: A pattern learning strategy for making draws and bets in poker games using convolutional networks. In 30th AAAI Conference on Artificial Intelligence, 2016.