Project Report

Subject: Poker playing agent (specifically heads-up no-limit Texas Hold'em, Doyle's game)

Description of problem:
Poker is increasingly becoming an area of interest in AI Research.
This is due to certain interesting qualities present in Poker that are absent from more oldfashioned parlor games, such as chess. The imperfect information and stochastic nature of poker are of particular interest to AI, additionally handling the opponents attempts at disinformation is also a crucial aspect of the game. All of these make the search space for poker prohibitively large for classic AI techniques.

As discussed in a recent poker research review[1], almost all efforts so far have been concentrated on the Texas Hold'em style of poker. This is the variant this project deals with as well.

Texas Hold'em overview:
The game of Texas Hold'em is played in 4 stages – preflop, flop, turn and river. During the preflop all players at the table are dealt two hole cards, which only they can see. Before any betting takes place, two forced bets are contributed to the pot, i.e. the small blind and the big blind. The big blind is typically double that of the small blind. The player to the left of the big blind, known as under the gun, then begins the betting by either folding, calling or raising. The possible betting actions common to all variations of poker are described as follows:

- *Fold*: When a player contributes no further chips to the pot and abandons their hand and any right to contest the chips that have been added to the pot.
- *Check/Call*: When a player commits the minimum amount of chips possible in order to stay in the hand and continue to contest the pot. A check requires a commitment of zero further chips, whereas a call requires an amount greater than zero.
- *Bet/Raise*: When a player commits greater than the minimum amount of chips necessary to stay in the hand. When the player could have checked, but decides to invest further chips in the pot, this is known as a bet. When the player could have called a bet, but decides to invest further chips in the pot, this is known as a raise.

In a limit game all bets are in increments of a certain amount. In a no limit game players can wager up to the total amount of chips they possess in front of them. Once the betting is complete, as long as at least two players still remain in the hand, play continues on to the next stage. Each further stage involves the drawing of community cards from the shuffled deck of cards as follows:

- flop – 3 community cards
- turn – 1 community card
- river – 1 community card.

During each stage players combine their hole cards with the public community cards to form their best 5 card poker hand. Each stage also involves its own round of betting and play continues as long as there are players left who have not folded their hands.
A showdown occurs after the river where the remaining players reveal their hole cards and the player with the best hand wins all
If there are only 2 players left in the game, the situation is called heads-up.
Lastly Doyle's game simply describes s simplifying assumption that eliminates the need for bankroll management. Namely it is assumed that every player starts every hand with the same amount of chips, regardless of past events.

Related Approaches:
A variety of methods have been used for poker bots, but the field seems to still be in a relative infancy (especially the no-limit variety). The most popular techniques, thoroughly covered in [1], are mostly based on either custom built expert systems or approximate Nash equilibrium techniques. Both of these have severe limitations though.

The expert systems can only be as good as the experts who designed them and frequently are significantly worse than that as it is quite tricky to teach a computer to perform and detect bluffs.
The Nash equilibrium approach on the other hand suffers from its desire to always play “perfectly” which therefore makes it incapable of fully exploiting weak opponents, which is an integral part of poker. Indeed, in the 2011 AAAI poker competition, despite the fact that a Nash equilibrium bot beat every opponent, it lost the overall bankroll challenge due to this weakness.

Some other efforts exist that would in theory be able to avoid the above mentioned pitfalls. Specifically imperfect information game tree search[4] and agents that try to directly model their opponents and play accordingly[5]

Finally, there have been some attempts at neural network and evolutionary approaches with mixed results[6][7]. These approaches show promise as they manage to discover complex strategies such as bluffing without any prior knowledge, however, at the moment they don't seem to be performing as well overall as some of the other methods.
It's also worth noting that an interesting upset occurred in the AAAI competition of 2009 when one of the entries for the limit heads-up tournament was a basic ANN bot using 2 neural networks for pre-flop and post-flop play that had been trained on the data from the previous year. The bot managed to secure one 3rd place and one 1st place finish, edging out the significantly more complex competition.

Data:
As mentioned before, there's is an annual poker AI competition[2] held as part of the AAAI Conference on Artificial Intelligence. The competition logs are made public. Only in the past three years there are about 36,000,000 hands of the specific variety needed for this project.
Additionally this dataset presents both players hole cards, even if the hand was folded. This is information that wouldn't be available from a conventional poker site dataset.
The downside, however, is that the poker hands weren't actually played by humans, but by other poker agents. This turns out not to be a problem for the purposes of this project since the winning bots in this competition play at a very high level, even when compared to human experts.[3]

Lastly, since the dataset is computer generated it comes in a format that is relatively easy to process. I was able to write a script that extracted each decision of a certain bot on a line and then used this to obtain a list of decisions taken by the champion agent in the 2011 competition. Due to the nature of the game, several “decisions” must be taken in each hand, so even though I only extracted data from one bot in one year of the competition, I still ended up with some 10 million decisions.
This proved to be too much to analyze given the time and computing constraints, so I had to heavily filter the dataset and bring it to around 500,000 decisions. While doing this I attempted to preserve as much of the variance as possible and made sure to include hands played against every opponent as well as hands played both early and late in the game.

Methods:
I looked at several different ways to approach this problem from a machine learning perspective.
First the parameters had to be defined. I decided that I should output a single number, the expected value of my hand in the current game situation. Based on this expected value the decision about whether to fold, call or bet can be made. Deciding which inputs best represent the complex game space is more difficult and error-prone however. In total, I looked at three different input sets:

- position, amount needed to stay in hand and estimated value of my hand
- as above + overall aggressiveness of opponent
- as above + both overall and recent aggressiveness of opponent

The estimated value of my hand was calculated using the PokerEval library[8] which itself uses Monte Carlo simulation to calculate the odds of a player winning the showdown given the currently known cards. The opponents aggressiveness score uses a simple heuristic which was shown to work well in a previous paper [6].
The recent aggressiveness is the same as the overall one, except that it only considers the last 10 hands.
Lastly, each input was scaled to fit roughly into a [0,1] interval. This is meant so that no one input dominates the result of the computation.
This tuning turned out to have quite a significant effect on the final accuracy.

Based on this framework I explored a few different variations of Neural Networks and compared them against some naïve baseline methods as well as against one another. All of these are presented in detail in *Table 1*.

From the above results it became clear to me that the performance of the networks can vary wildly with a small change in the parameters, so I tried to formalize the search for the best network by using the NEAT algorithm[9]. However this turned out to just transform one optimization problem into another, since now I had the task of figuring out appropriate values for the 25 or so arguments taken by NEAT.
I didn't obtain any concrete results, but, on average, the generated networks seemed to confirm my hypothesis that a relatively small number of nodes should be sufficient. Two examples of generated topologies are presented in *Pictures 1 and 2*.

At this point I attempted to use a Support Vector Machine method instead of the neural network one.
For this purpose I used the 5 input representation mentioned above, however, instead of outputting a single value I had to break the cases up into a handful of categories instead, one each for folding and calling and several for betting, depending on amount.
I used the LIBSVM library [10] to generate the actual SVM model as well as to detect the optimal parameters of the kernel function (Picture 3). Even though the model achieved a ~75% accuracy on test data, it had a lot of difficulty detecting bets and thus predicted too large a proportion of folds and calls.
This lead to poor performance in the tests with the other methods(Table 1). I believe this may be due to the fact that bets were fragmented into more than one category, according to bet size, thus there may have been too few instances of each of the betting categories. The training data could be filtered to ensure sufficient bets are present, but that would skew the prior probabilities and would probably lead to too much aggressiveness in betting. This is a problem that needs further work.

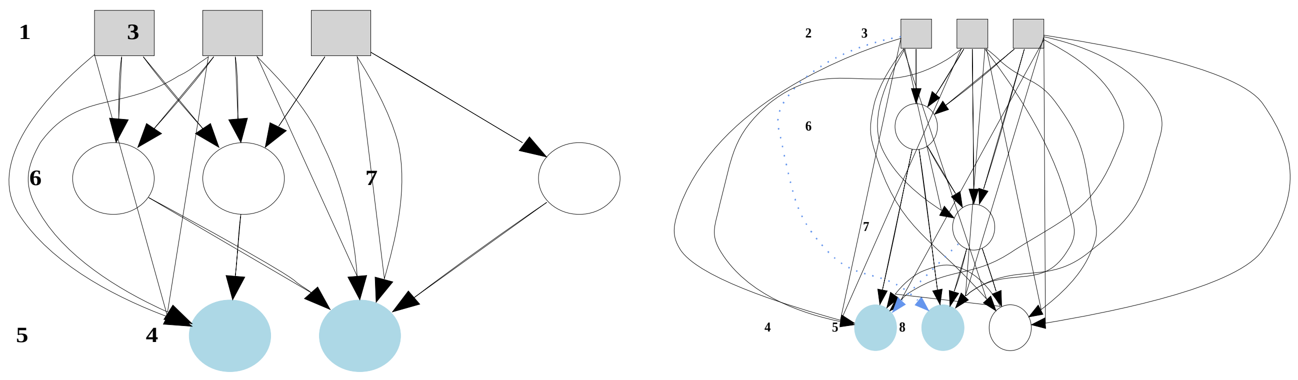
	SimpleNet	DoubleNet	AggroNet	FourNet	SVMBot	RaiseBot	CallBot
SimpleNet		-3184201	17430349	7590137	1558090	12552263	10909086
DoubleNet	3184201		-1406034	843386	2109114	1148421	3221558
AggroNet	-17430349	1406034		8715307	4039286	-494207	6521785
FourNet	-7590137	-843386	-8715307		-4497580	818372	3831863
SVMBot	-1558090	-2109114	-4039286	4497580		263200	0
RaiseBot	-12552263	-1148421	494207	-818372	-263200		-7800
CallBot	-10909086	-3221558	-6521785	-3831863	0	7800	

Table 1: These are the results of 10,000 hands played between each pair of bots. In order to reduce variance the same 5000 hands are used twice with the bots switching places.

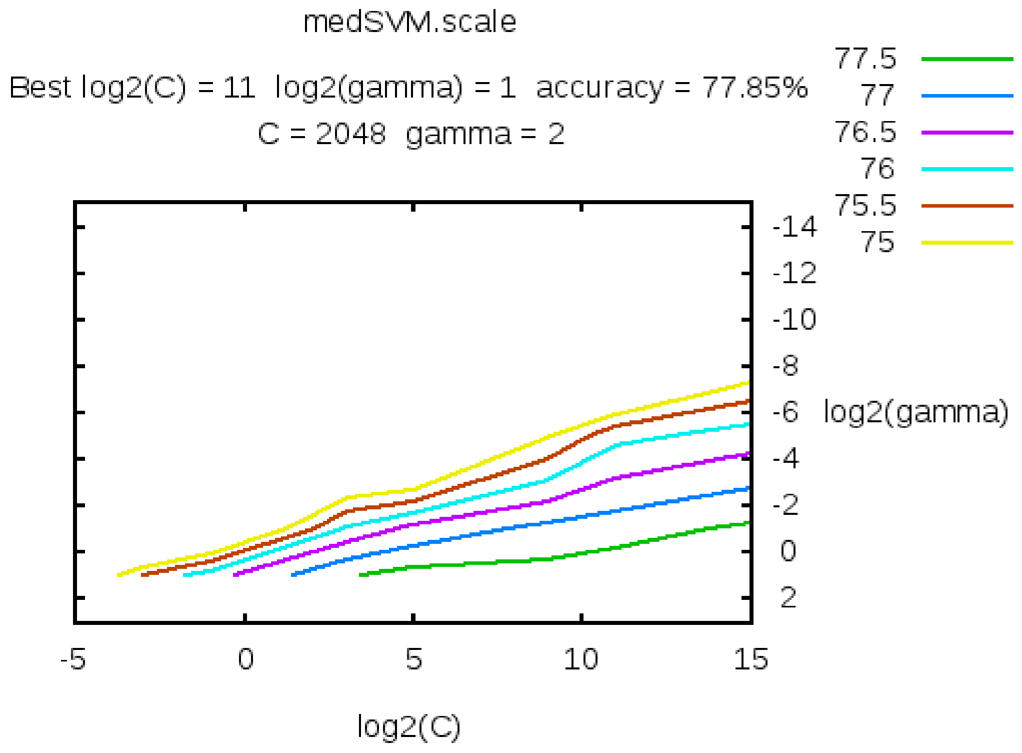
SimpleNet: 47098038 = 3 input, 3 hidden, 1 output neural net
DoubleNet: 9343473 = 2 SimpleNets, one for pre-flop and one for post-flop
AggroNet: 3002099 = 4 input ,3 hidden, 1 output neural net
SVMBot: -2691949 = SVM model, 5 features, multiple classes
RaiseBot: -14048448 = Baseline bot, always raises
FourNet: -16754685 = 4 SimpleNets, one for each stage of play
CallBot: -24215141 = Baseline bot, always calls

It's interesting to note that there is a cycle between the best 3 methods. SimpleNet loses to DoubleNet which loses to AggroNet which loses to SimpleNet (AND to RaiseBot oddly). This shows the difficulty in building a robust bot as well as the intransitive nature of the game

It's also worth noting that AggroBot's aggressiveness modeling does seem to lead to better exploitation of the weak bots as he gets the most chips out of FourNet and SVMBot and also a significant amount from CallBot. The fact that he loses so badly to SimpleNet is another interesting event.
Lastly, the abysmal performance of FourNet is also worth noticing. This bot likely suffers from similar problems to SVMBot regarding the lack of sufficient test-cases but also from the fact that there is no information being passed from one network to the other. Therefore FourNet cannot capitalize on any information gained about the opponent's hand in previous betting rounds.



Pictures 2 & 3: Topologies generated by NEAT



Picture 3: Determining the optimal parameters for the SVM kernel

References:

[1] J. Rubin, I. Watson, Computer poker: A review, Artificial Intelligence (2011), doi:10.1016/j.artint.2010.12.005
[2] http://www.computerpokercompetition.org/
[3] http://webdocs.cs.ualberta.ca/~games/poker/man-machine/Results/
[4] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M.H. Bowling, R.C. Holte, J. Schaeffer, D. Szafron, Game-tree search with adaptation in stochastic imperfect-information games, in: Computers and Games, 4th International Conference, CG 2004, 2004, pp. 21–34.
[5]M.B. Johanson, Robust strategies and counter-strategies: Building a champion level computer poker player, Master's thesis, University of Alberta, 2007.
[6] G. Nicolai, R. Hilderman, No-limit Texas hold'em poker agents created with evolutionary neural networks, in: CIG-2009, IEEE Symposium on Computational Intelligence and Games, 2009, pp. 125–131.
[7] J. Noble, Finding robust Texas hold'em poker strategies using Pareto coevolution and deterministic crowding, in: Proceedings of the 2002 International
Conference on Machine Learning and Applications – ICMLA 2002, 2002, pp. 233–239.
[8] http://freecode.com/projects/poker-eval
[9]Evolving neural network through augmenting topologies - Kenneth O. Stanley and Risto Miikkulainen Department of Computer Sciences, The University of Texas at Austin *Evolutionary Computation* 10(2):99-127, 2002.
[10] http://www.csie.ntu.edu.tw/~cjlin/libsvm/