An Adaptive Intelligence For Heads-Up No-Limit Texas Hold'em

Etan Green

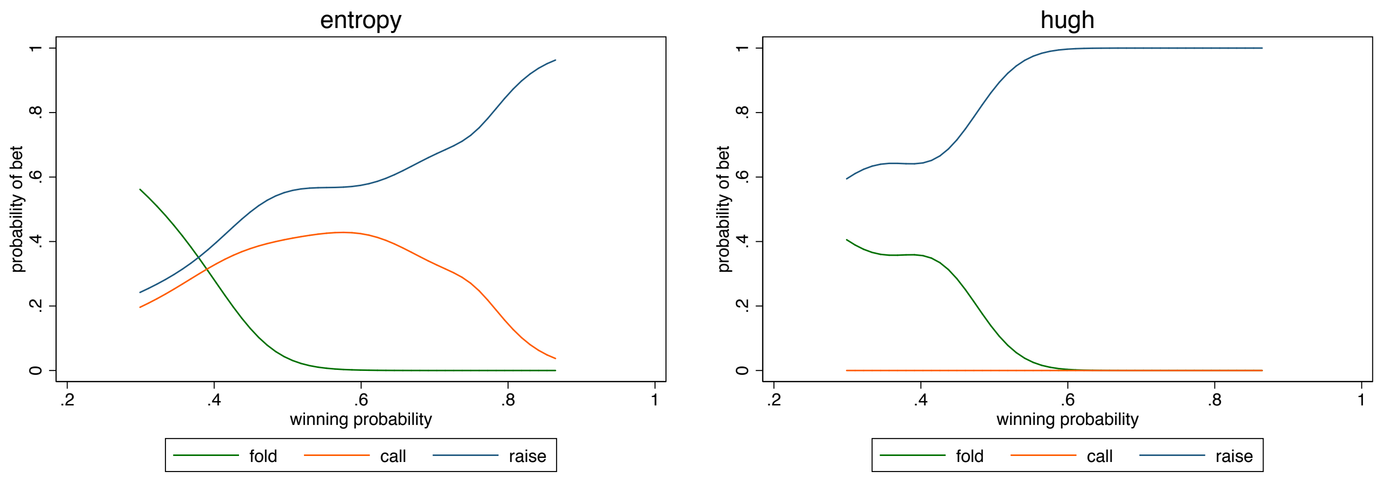
December 13, 2013

Skill in poker requires aptitude at a single task: placing an optimal bet conditional on the game state and the opponent. The best poker artificial intelligences place bets that are optimal with respect to the game state, but not the opponent. These pokerbots train complex betting functions from tens of millions of hand histories or billions of simulated games, and they tend to work well against opponents that resemble the bot's collective experience. But an optimal strategy against one opponent may be a poor approach against another. Games of heads-up poker last for dozens, if not hundreds, of hands, and each hand provides information about the opponent's strategy. Existing bots cannot use this information to adapt their betting functions—their parameters, trained on billions of hands, are too numerous to update from relatively sparse experience with a given opponent. By contrast, my bot relies on a parsimonious betting function whose parameters are updated when it observes its opponent's bets. This function will likely be inferior to existing pokerbots when it has no information about its opponent. But after a number of hands, it should outperform generic betting functions. For my CS229 final project, I trained the initial parameter vector using hand histories from a 2013 computer poker tournament.

Strategies vary considerably, even among top players. Consider two of the world's best pokerbots, 'Entropy' and 'Hugh'. Figure 1 depicts how each bets when it faces the first bet of a hand. Here, the blinds are 50 and 100, so the first bettor can either fold, call with 50, or raise by at least 100. On the x-axis is the winning probability associated with the bot's hole cards. At this point, the shared cards are unknown, and the bot knows nothing about what its opponent might hold. I calculate these winning probabilities by running a Monte Carlo simulation over all of the unseen cards. Pocket aces win close to 90% of simulated hands; low, unsuited, non-pairs win about 30% of simulated hands. Both bots fold less and raise more as their cards improve, but similarities end there. Entropy predominantly folds when its cards are weak; Hugh is most likely to raise even on the weakest cards. Entropy calls on as much as 40% of hands; Hugh never calls.

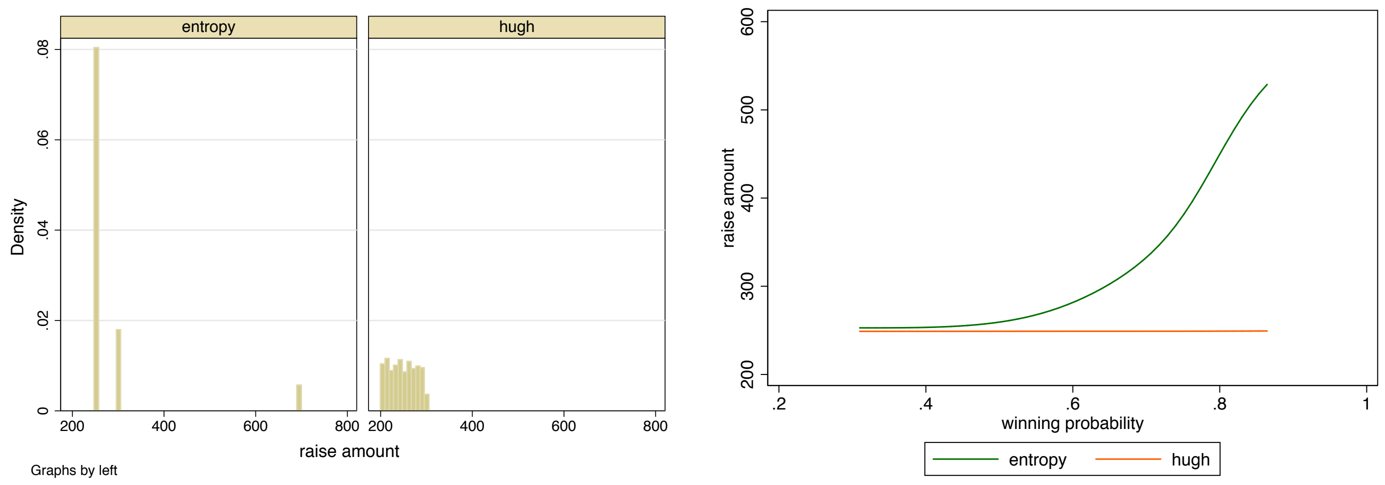
The strategies of these bots also diverge when they raise. Figure 2 shows a histogram of their raises on the left and the relationship between their raises and the quality of their cards on the right. Entropy chooses

Figure 1: Probability of folding, calling, or raising by winning probability for 2 entrants in a computer poker tournament. Sample restricted to the first bet of the hand with blinds of 50/100. Winning probabilities calculated for player's hole cards via a Monte Carlo simulation over unseen cards. Estimates via kernel regression.



among three values when it raises initially: 250, 300, and 700; Hugh chooses uniformly between 200 and 300.[1] The more Entropy bets, the better its cards tend to be; for Hugh, there is no relation between the size of its initial raise and the quality its cards.

Figure 2: Histogram of raise amounts on first bet of hand (left), and kernel regression of raise amount on winning probability (right).



The proclivities of one's opponent matter when making a bet. An opponent's bets give information both about the cards it holds and about how it plays in particular game states. A generic betting algorithm might rightly suppose that an initially high bet signals good cards, but if its opponent were Hugh, that supposition would be false. It might also make a bet under the belief that its opponent would call with high probability,

[1]The minimum value of 200 represents the amount the player has put in the pot (50), the amount needed to call (50), and the minimum raise (100).

which will be more true for some opponents than others.

My bot keeps track of two functions that predict an opponent's actions from the game state. The first function predicts whether the opponent will fold, call, or raise (or check or raise) in a particular game state. The second function predicts how much the opponent will raise, conditional on choosing to raise. Let $\phi(x)$ summarize the game state. Then for $Bet \in \{\text{fold}, \text{call}, \text{raise}\}$ or $Bet \in \{\text{check}, \text{raise}\}$,

$$P(Bet = i|\phi(x);\theta) = \frac{\exp(\phi(x)\theta_i)}{\sum_j \exp(\phi(x)\theta_j)},$$

where $\theta_i = 0$ for some i. I assume that each raise, R, is a random drawn from a log-normal distribution. For some realization $r \in (0, \infty)$ of R,

$$P(R = \log(r)|\phi(x);\theta) \sim \mathcal{N}(\mu(\phi(x);\theta), \sigma^2),$$

where $\mu = \phi(x)\theta_\mu$ and σ^2 is assumed to be known.

With $\hat{\theta}$, the bot can identify its opponent's expected action in a particular game state x and calculate the value of its bets through an expectimax routine. Let a indicate who's turn it is and b be an object that summarizes a betting round. Then the value function for a betting round is:

$$V_{opt}(a,b) = \begin{cases} b.\text{scores[bot]} + P(\text{bot wins}) \times b.\text{pot} & b.\text{isOver} \\ \max_{bet \in b.\text{bets}} V_{opt}(\neg a, b.\text{makeBet}(bet)) & a = \text{bot} \\ \sum_{bet \in b.\text{bets}} P(Bet = bet|\phi(x)) \times V_{opt}(\neg a, b.\text{makeBet}(bet)) & a = \text{opp} \end{cases}$$

When a betting round is over, the bot gets the negative amount it has put in the pot (b.scores[bot]) plus its share of the pot in expectation (P(bot wins) $\times$ b.pot).[2]

Two quantities remain undefined, the attributes of the game state, $\phi(x)$, and the probability that the bot will win the hand. The game state is partly defined by attributes of the betting round: the round number (blinds, flop, turn, river), the size of the pot, and the amount to call. But it is also defined by the cards held by the bot, the shared cards that have been revealed, the unseen shared cards, and the hole cards the opponent might hold. Because the combinatorics of these cards is immense, I define the game state in terms of winning probabilities: the bot's beliefs about how likely it is to win given what it knows about the cards, and what it thinks its opponent believes about its own likelihood of winning.

[2] V_{opt} is infinitely recursive if the bot's best response is always to raise, and the opponent responds by raising with some probability. I amend V_{opt} to consider only d successive raises by the opponent. After $2d$ recursive calls, the opponent calls or folds with probability 1, ending the betting round.

The bot's beliefs encapsulate three quantities:

1. The probability that the opponent holds a particular *pair*: p_{pair}.
2. The probability that the bot will win conditional on its opponent holding *pair*: $p_{win|pair}$.
3. The opponent's beliefs conditional on holding *pair*:
   (a) The probability that the bot holds a particular *pair'*: $q_{pair,pair'}$
   (b) The probability that the opponent will win conditional on the bot holding *pair'*: $q_{win|pair,pair'}$.

At the beginning of a hand, the bot knows its own cards and that the opponent holds one of $\binom{50}{2}$ pairs with equal probability. For each pair, $p_{win|pair}$ and $q_{win|pair'}$ are deterministic and can be calculated via a Monte Carlo simulation over the $\binom{48}{5}$ combinations of shared cards. The bot believes its probability of winning, p_{win}, to be the dot product of $\vec{p}_{pair}$ and $\vec{p}_{win|pair}$. It also believes that its opponent believes its own winning probability, q_{win}, to be $\vec{p}_{pair} \cdot \vec{q}_{win|pair}$, where $q_{win|pair} = \vec{q}_{pair,pair'} \cdot \vec{q}_{win|pair,pair'}$. The attributes of a game state, x, include both observables of the betting round, r, and p_{win} and $\vec{q}_{win|pair}$.

Beliefs are updated when shared cards are revealed, or when a player makes a bet. When shared cards are revealed, pairs that contain any of the shared cards are eliminated from beliefs, and the vectors of winning probabilities $\vec{p}_{win|pair}$ and $\vec{p}_{win|pair;opp}$ are recomputed by iterating over the possible unseen shared cards. When the opponent makes a bet, the bot performs a Bayesian update on $\vec{p}_{pair}$ by weighting pairs that are rationalized by the bet: $p^{(1)}_{pair} \propto p^{(0)}_{pair} P(Bet = bet|\phi(r, q_{win|pair}))$. If the opponent typically raises when it believes its winning probability to be high, then observing the opponent raise tells the bot that it likely has cards associated with high subjective winning probabilities. The bot represents $\vec{p}^{(0)}_{pair}$ as a Dirichlet distribution. Since the likelihood follows a multinomial distribution with non-integer counts, and the Dirichlet and multinomial are conjugate distributions, $\vec{p}^{(1)}_{pair}$ also follows a Dirichlet. When the bot makes a bet, the bot updates $\vec{q}_{pair|pair'}$ for each *pair* and *pair'*: $q^{(1)}_{pair|pair'} \propto q^{(0)}_{pair|pair'} P(Bet = bet|\phi(r, 1 - q_{win|pair,pair'}))$.[3] Bets by the opponent inform the bot's beliefs about the pair the opponent holds. Bets by the bot inform the bot's beliefs about the opponent's beliefs about the pair held by the bot.

Since p and q are both functions of θ and factor into the likelihood of a bet, I estimate $\hat{\theta}$ from hand histories using a two-step estimator. The data come from a 2013 computer poker tournament in which 14 bots played $\sim$ 20M hands.[4] I estimate $\hat{\theta}$ on 10,000 hands played by Entropy and Hugh. I loop over the data 5 times. In each iteration, I loop through the hands in a random order. For each hand, I progress through the bets

[3]Note that $q_{win|pair,pair'}$, or what the bot suspects the opponent to believe about its chances of winning if the opponent holds *pair* and the bot holds *pair'*, is a function only of the cards, not the bets. Were I to presume common knowledge, beliefs would be infinitely recursive. I specify only one level of recursion.

[4]Unlike hand histories from poker websites, these data show the hole cards of each player for each hand, even when the hand does not end in a showdown.

sequentially, updating the parameters once for each bet. Estimation at each bet occurs in two stages: first, I update the agent's beliefs from the previous bet, holding $\hat{\theta}$ fixed. Then I perform one iteration of gradient ascent on $\hat{\theta}$, holding p and q fixed: $\theta^{(1)} = \theta^{(0)} + \alpha \frac{\partial P(Bet|\phi(x))}{\partial \theta}$. I settled on $\alpha = 0.001$, which appears to produce some measure of convergence after 5 iterations. I define the ϕ transformation to include the linear, squared, and cross terms of x.

A principal obstacle in estimation is computational. When shared cards are revealed, the bot updates $\vec{p}_{win|pair}$ and $\vec{q}_{win|pair,pair'}$ for each $pair$. After the flop, each vector is $\binom{47}{2}$ pairs long, and there are $\binom{47}{2}+1$ vectors to update.[5] Updating each pair requires iteration over $\binom{45}{2}$ combinations of unseen shared cards. This update requires over a billion iterations for each flop, for each agent. This is an infeasible chore, so I estimate $\hat{\theta}$ only on betting during the blinds, before any shared cards are revealed. The estimates are directionally sensical: the higher an agent believes its likelihood to be, the more likely it is to raise and to raise larger amounts; the larger the pot, the less likely the agent is to fold; and the higher the amount required to call, the more likely the agent is to fold.

The problem is that pots in the data tend to stay small during the blinds round, and parameters estimated on these bets do not perform well in simulations when the pots become large. For instance, when my bot makes the first bet of the hand, it uses V_{opt} to evaluate the expected payoff of a call, a fold, and the raise r that maximizes the payoff heuristic:[6]

$$\text{Payoff}(r) = P_{opp}(Bet = fold|\phi(x|r)) \times 200 + \big(1 - P_{opp}(Bet = fold|\phi(x|r))\big) \times \big[p_{win}(200 + 2r) - (1 - p_{win})r\big]$$

Here, the bot gets a pot of 200 if r induces a fold; if the opponent does not fold, the bot gets a pot of $200 + 2r$ with probability p_{win} or $-r$ with probability $1 - p_{win}$. Since folds are not often observed in the blinds stage, $P_{opp}(Bet = fold|\phi(x|r))$ does not reach 1 at any r, and the bot bets high (~ 3000) when $p_{win} > \frac{1}{2}$. At these raises, the likelihood is dictated by the size of the pot and the amount to call, and is uniform across the belief variables p and q. This means that each element of $\vec{p}_{pair}$ and $\vec{q}_{pair,pair'}$ is given the same weight after a bet, yielding no update to p_{win} or $\vec{q}_{win|pair}$.

In addition to estimating $\hat{\theta}$ on later betting rounds, I plan to put more structure on $\phi(x)$. $\phi(x)$ should correspond to the payoff an agent expects from a bet; the additive specification of squared and cross terms does not. I suspect parameterizing $\phi(x)$ as in the payoff function above will inspire intelligent play across a range of game states.

[5] One for each $pair$ in $\vec{q}_{win|pair,pair'}$ plus one for $\vec{p}_{win|pair}$.

[6] I maximize this quantity using the Golden Section algorithm.