2017-Moravcik

DeepStack: Expert-level artificial intelligence in heads-up no-limit poker

---

- **Date**: 2017-05-05
- Link
- **Authors:**
  - Moravcik, Matej
  - Schmid, Martin
  - Burch, Neil
  - Lisy, Viliam
  - Morrill, Dustin
  - Bard, Nolan
  - Davis, Trevor
  - Waugh, Kevin
  - Johanson, Michael
  - Bowling, Michael
- **Cites:**
- **Cited by:**
- **Keywords:** #poker
- **Collections:**
- **Status:** #in-progress

---

0. - Abstract

- Poker is a long standing benchmark for imperfect information games.

- DeepStack is an algorithm:
  - Recursive reasoning to handle information asymmetry
  - Decomposition to focus computation on the relevant decision
  - Intuition that is learned from self-play using deep learning.

- DeepStack outperformed professional poker players in heads-up no-limit poker.

- This is an improvement on prior work.

1 - Introduction

- *Information symmetry* - all players have identical information about the current state of the game.

- Many successes have been made in solving games with information symmetry (i.e. perfect information games).

- However, the solutions developed for these games, namely local search during play, do not translate to the imperfect information domain.

- Inherent challenge of imperfect information games:

  - The correct decision at a given moment depends on the probability distribution over private information that an opponent holds.

  - This distribution is formed by observing the opponent's prior actions.

  - The catch however is that the opponent's actions are informed by their own knowledge of our private information and how our actions reveal it.

  - This is like a recursive cat-and-mouse game in which each player is trying level the other.

- *Counterfactual Regret Minimization* (CFR) - common technique for solving imperfect information games in which the optimal strategy is computed entirely before play.

- This runs into problems in when the game size is too large to reasonably be computed in its entirety.

- One solution to this is to map game states onto a smaller abstract game state space.
  - E.g. transform HUNL 10^{160} states onto a 10^{14} abstract state space.
  - However this solution performs poorly in practice.

- DeepStack's approach:

  - Does not compute and store a complete strategy prior to play.

  - Considers each particular situation as it arises during play.

- Reasons by looking forward a fixed depth, then substitutes the leaf values with an approximate estimate.
- This estimate is computed using deep learning techniques.

1. - DeepStack

- *DeepStack* - general-purpose algorithm for a large class of sequential imperfect-information games.
  - For the sake of explanation, poker is used as an example for this paper.
- Definitions:
  - *Player's private information* - the cards they hold face down.
  - *Public state* - the cards laying face up and the sequence of betting actions made by the players.
  - *Public game tree* - extensive form game tree formed by public states.
  - *Decision point* - public game state and player's private information pair.
  - *Player's strategy* - probability distribution over valid actions for each of the player's decision points.
  - *Player's range* - probability distribution computed over the player's possible hands given that the public state is reached.
  - *Player's utility* (at a terminal state) - bilinear function of both players' ranges using a payoff matrix determined by the rules of the game.
  - *Player's expected utility* (at a non-terminal state) - expected utility over reachable terminal states given the players' fixed strategies.
  - *Nash Equilibirum* - solution to a two-player zero-sum game, such as HUNL, where each player maximizes their expected utility when playing against a best-response opponent strategy.
  - *Exploitability of a strategy* - the difference between a strategy and a Nash equilibrium with respect to expected utility against a best response agent.
- The DeepStack algorithm:
  - Only computes a strategy for the public states that actually arise during play.
    - i.e. no strategy is retained over the vast public game tree.
  - The strategy is stochastic and static, arising from a deterministic process.
- DeepStack algorithmic ingredients:
  1. Sound local strategy for the current public state.
  2. Depth limited look-ahead using a learned value function.
  3. A restricted set of look-ahead actions.
- *Heuristic search* - method of game solving in which no information is retained of how or why prior actions were taken. *Continual researching* is done each time an action must be taken using local search.
- DeepStack is the first algorithm to successfully implement heuristic search for imperfect information games, though heuristic search has produced many successful results for perfect-information games.

1.1 - Continual Re-Solving

- *Re-Solving* - process by which, at a given game state, the strategy that was used to arrive at that game state is reconstructed.
- Two sets of information needed for re-solving:
  - Our range at the public state.
  - Expected values achieved by the opponent under the previous solution for each possible opponent hand.
    - These are counter-factual values (i.e. the "what-if" expected value if the opponent managed to reach this state with a given hand.)
    - CFR produces these values so it is a natural solver for this case.
- *Continual Re-Solving* - re-solving done at every decision point such that a strategy is never maintained across decisions.
- Relaxation of our opponent counterfactual values for continual re-solving:
  - Must be an upper bound on the value the opponent can achieve with each hand in the current public state.
  - Must be smaller than or equal to the value the opponent *could have* achieved had they deviated from reaching the public state.
    - *Note to self: I need to understand this better.*
  - See proof of sufficiency in **Theorem 1**.

- Initial starting values:
  - Our range is uniform (we can be dealt any two cards at the start.)
  - The opponent counterfactual values are initialized to the value of being dealt each private hand.
- Value update after each game step:
  - After our own actions:
    - Replace the opponent counterfactual values with those computed in the re-solved strategy for our chosen action.
    - Update our own range using the computed strategy and Bayes' rule.
  - Chance action:
    - Replace the opponent counterfactual values with those computed for this chance action from the last re-solve. *(I'm not entirely sure how this works?)*
    - Update our own range by zeroing hands in the range that are impossible given new public cards. (We can't have the two-of-spades in our hand if it's dealt on the flop.)
  - Opponent action:
    - No change to our range or opponent values.
- This procedure guarantees the opponent counterfactual values meet the criteria above, and produces strategies that are arbitrarily close to the Nash equilibrium.
- Key differences between DeepStack and action abstraction methods:
  - The opponent range is never computed or tracked, saving time.
  - Does not require knowledge of opponent actions.
- Re-solving is impractical for large sub-game trees, limiting is required.

1.2 - Limited-depth look-ahead via intuition

- Challenge - opposed to heuristic search in perfect-information games, we can not simply replace sub-trees with precomputed values.
  - Our counterfactual values are dynamic, depending on how players play to reach a public state and change with each iteration of CFR.
- DeepStack solves this challenge by using a function to approximate the counterfactual values using the current iteration's public state and player ranges.
  - Input into the approximation function:
    - Ranges for both players.
    - Pot size.
    - Public cards.
  - This can be thought of as a description of a poker game - the ranges being the probability of being dealt given cards and the pot being the stakes.
  - Output from the approximation function:
    - Vector for each player containing the counterfactual values of holding each hand in that situation.
  - This can be thought of as estimates of how valuable holding certain cards would be in such a game.
- Depth limit of four actions reduces the game for re-solving from 10^{160} to 10^{17} decision points.
- DeepStack uses a deep neural network for the learned value function.

1.3 - Sound reasoning

- **Theorem 1** - If the values returned by the valuee function used when the depth limit is reached have error less than ϵ , and T iterations of CFR are used to re-solve, then the resulting strategy's exploitability is less than $k_1\epsilon + k_2/\sqrt{T}$ where k_1 and k_2 are game specific constants.

1.4 - Sparse look-ahead trees

- DeepStack restricts the number of actions considered to produce sparse look-ahead trees, only considering the actions fold, call, two or three bet, and all-in.
- Trade-off - removes the theoretical guarantee provided by **Theorem 1** allows for a speed-up to comparable human levels.
- This further reduces the re-solved game trees to 10^7 decision points.
- This is sufficient to solve in $5s$ on a NVIDIA GeForce GTX 1080 graphics card.
- Sparse look-ahead is also used at the start of the game to compute the initial opponent counterfactual values.

1.5 - Relationship to heuristic search in perfect-information games

- Three key challenges for heuristic search in perfect-information games:
  1. Re-solving of public states using the agent's range and opponent counterfactual values.
     - We need knowledge of how and why the players acted to reach the public state in order to reconstruct a strategy.
  2. Re-solving is an iterative process that traverses the look-ahead tree multiple times.
     - Each iteration queries the evaluation function with different ranges.

3. Counterfactual evaluation function is more complex than the perfect-information case.
    - A vector of values is returned for a given public state and player ranges, rather than a single fixed value.

1.6 - Relationship to abstraction-based approaches

- Similarities:
    - DeepStack restricts the action set in the look-ahead tree.
    - Hand clustering is used as inputs to the approximation function.
        - However this is done at the end of the look-ahead tree rather than limiting the information the players actually have.
- Differences:
    - Each re-solve starts from the actual public state.
    - The opponent's actual action is never needed, avoiding the translation of opponent bets.
- These differences result in drastically different observed behavior.

2. Deep counterfactual value networks

- An auxiliary network is used for preflop evaluations.
- Two separate networks are used for the flop and turn.
- Note - no evaluation network is needed for the river since we can look-ahead to the end of the game.

2.1 - Architecture

- Standard feedforward network.
- 7 fully connected hidden layers with 500 nodes each.
- Parametric rectified linear units for the output.
- Outer wrapper around the model enforces the zero-sum property.
    - Takes the two value estimations, one for each player, and performs a weighted sum according to the player's input ranges, resulting in two separate game values.
    - These two game values should sum to zero, though they're not guaranteed to.
    - Half of the total sum is subtracted from each player's value, resulting in a net zero sum.
    - This is a differentiable process suitable for gradient descent.
- The player ranges are encoded by clustering to 1000 buckets and input as a vector of probabilities over the buckets.
- Outputs vectors consisting of fractions of the pot size, estimating the counterfactual values.

2.2 - Training

- Turn network training - 10 million randomly generated poker turn games.
    - Using randomly generated ranges, public cards, and pot sizes.
    - Target counterfactual values for each training game were generated by solving the game with no card abstraction.
        - Players' actions were restricted to fold, call, pot-sized bet, and all-in.
- Flop network was trained similarly - 1 million randomly generated flops.
    - Difference from turn - target values were generated using the normal depth-limited search procedure and the trained turn network.

3. Evaluating DeepStack

- See the paper for impressive results against professional players.

3.1 - Exploitability

- DeepStack is shown to be unexploitable (using LBR).

4. - Discussion

- DeepStack outperforms professional poker players with little domain knowledge and no training from expert human games.
- DeepStack allows computation to be focused on specific situations that arise when making decisions and the use of automatically trained value functions.