

ECE Reflections | Projections



The Reflections|Projections conference is bringing a host of distinguished speakers in the AI/ML field to UIUC this weekend. The list includes Christine Hung: Head of Spotify Data Science, Amar Das: Director of Healthcare Research at IBM Watson, and Travis Oliphant: creator of NumPy. More information and registration can be found at acmrp.org.

Announcements

- MP1 is due Monday at 23:59:59
- If you think you need an extension (e.g., because you have an exam or MP due in another class on Monday), you need to send me e-mail by tonight (9/28/2017 23:59:59).

CS440/ECE448 Lecture 10: Stochastic Games, Stochastic Search, and Learned Evaluation Functions

Slides by Svetlana Lazebnik, 9/2016

Modified by Mark Hasegawa-Johnson, 9/2017

Types of game environments

	Deterministic	Stochastic
Perfect information (fully observable)	Chess, checkers, go	Backgammon, monopoly
Imperfect information (partially observable)	Battleship	Scrabble, poker, bridge

Content of today's lecture

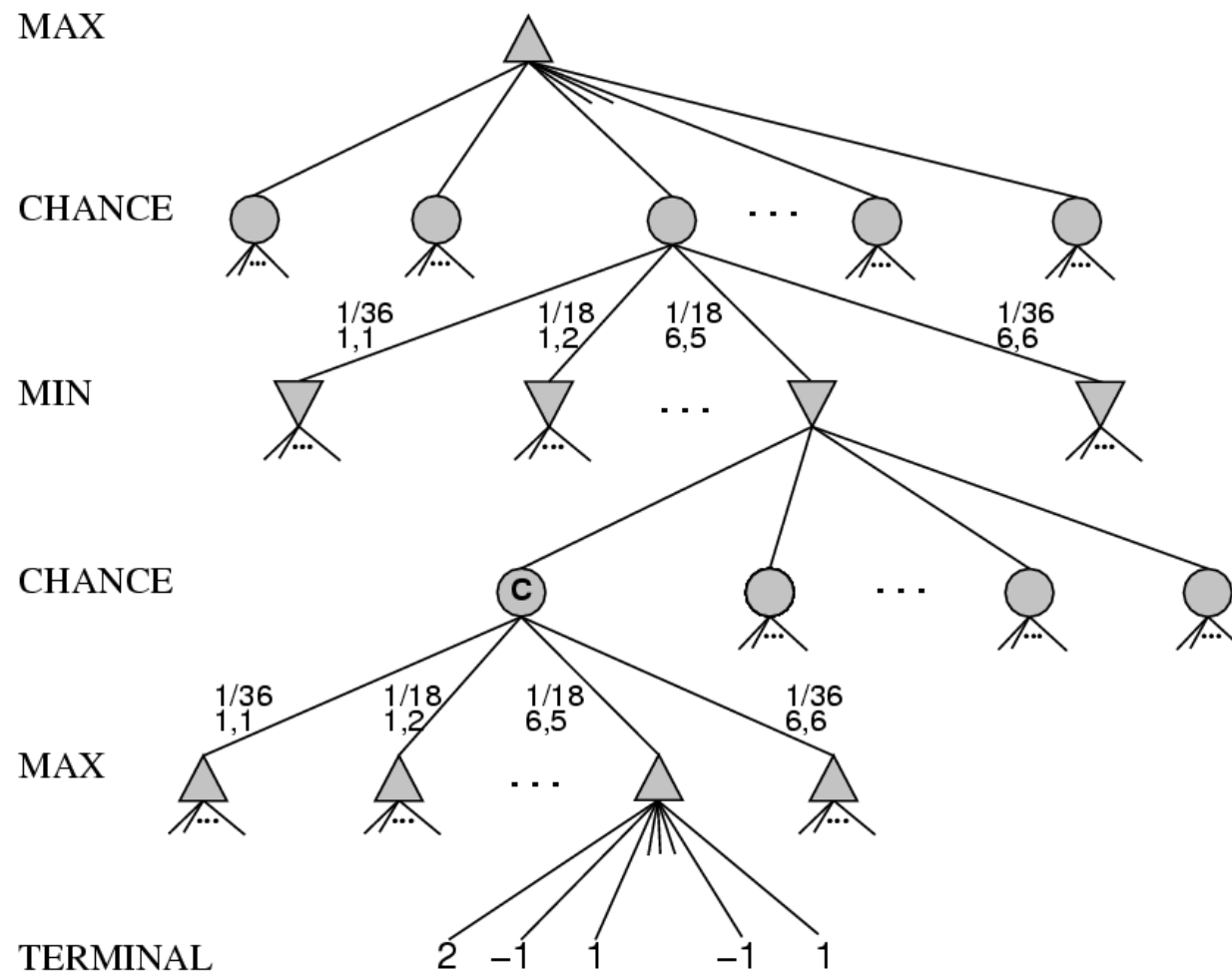
- Stochastic games: the Expectiminimax algorithm
- Imperfect information
 - Minimax formulation
 - Expectiminimax formulation
- Stochastic search, even for deterministic games
- Learned evaluation functions
- Case study: Alpha-Go

Stochastic games

How can we incorporate dice throwing into the game tree?



Stochastic games



Minimax vs. Expectiminimax

- **Minimax:**

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the
- **Reward**

$$Value(node) = \max_{my\ moves} \left(\min_{Min's\ moves} (Reward) \right)$$

- **Expectiminimax:**

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the
- **Expected** reward

$$Value(node) = \max_{my\ moves} \left(\min_{Min's\ moves} (\mathbb{E}[Reward]) \right)$$

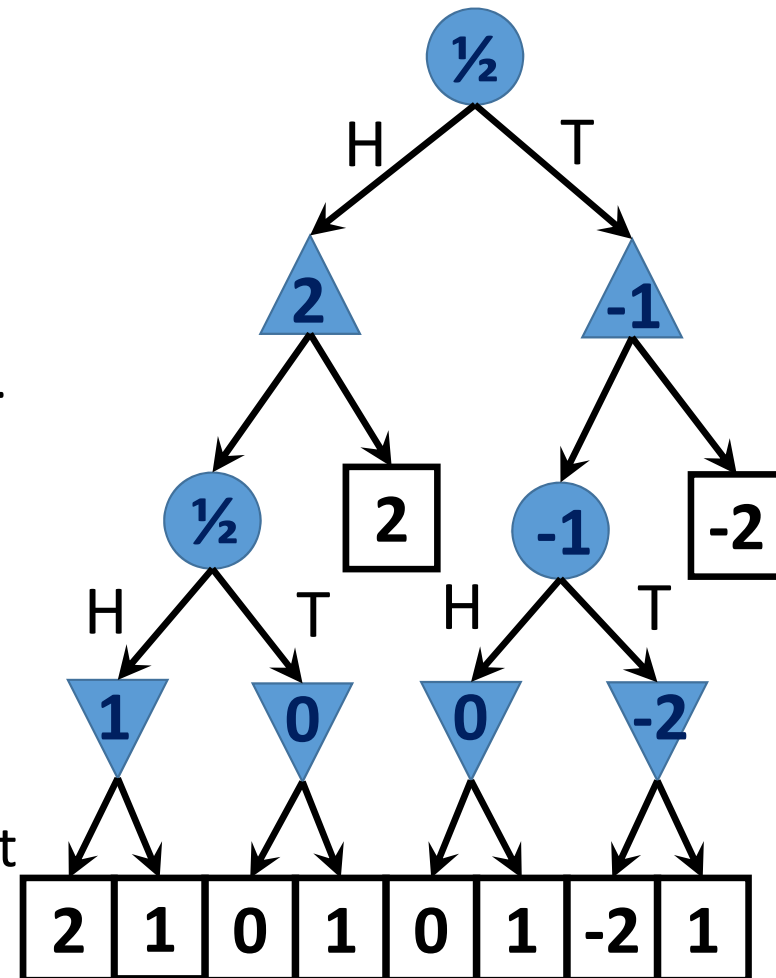
$$\mathbb{E}[Reward] = \sum_{outcomes} Probability(outcome) \times Reward(outcome)$$

Stochastic games

- **Expectiminimax:** for chance nodes, sum values of successor states weighted by the probability of each successor
- **Value**(*node*) =
 - $Utility(node)$ if *node* is terminal
 - $\max_{action} \text{Value}(Succ(node, action))$ if *type* = MAX
 - $\min_{action} \text{Value}(Succ(node, action))$ if *type* = MIN
 - $\sum_{action} P(Succ(node, action)) * \text{Value}(Succ(node, action))$ if *type* = CHANCE

Expectiminimax example

- RANDOM: Max flips a coin. It's heads or tails.
- MAX: Max either stops, or continues.
 - Stop on heads: Game ends, Max wins (value = \$2).
 - Stop on tails: Game ends, Max loses (value = -\$2).
 - Continue: Game continues.
- RANDOM: Min flips a coin.
 - HH: value = \$2
 - TT: value = -\$2
 - HT or TH: value = 0
- MIN: Min decides whether to keep the current outcome (value as above), or pay a penalty (value=\$1).



Expectiminimax summary

- All of the same methods are useful:
 - Alpha-Beta pruning
 - Evaluation function
 - Quiescence search, Singular move
- Computational complexity is pretty bad
 - Branching factor of the random choice can be high
 - Twice as many “levels” in the tree

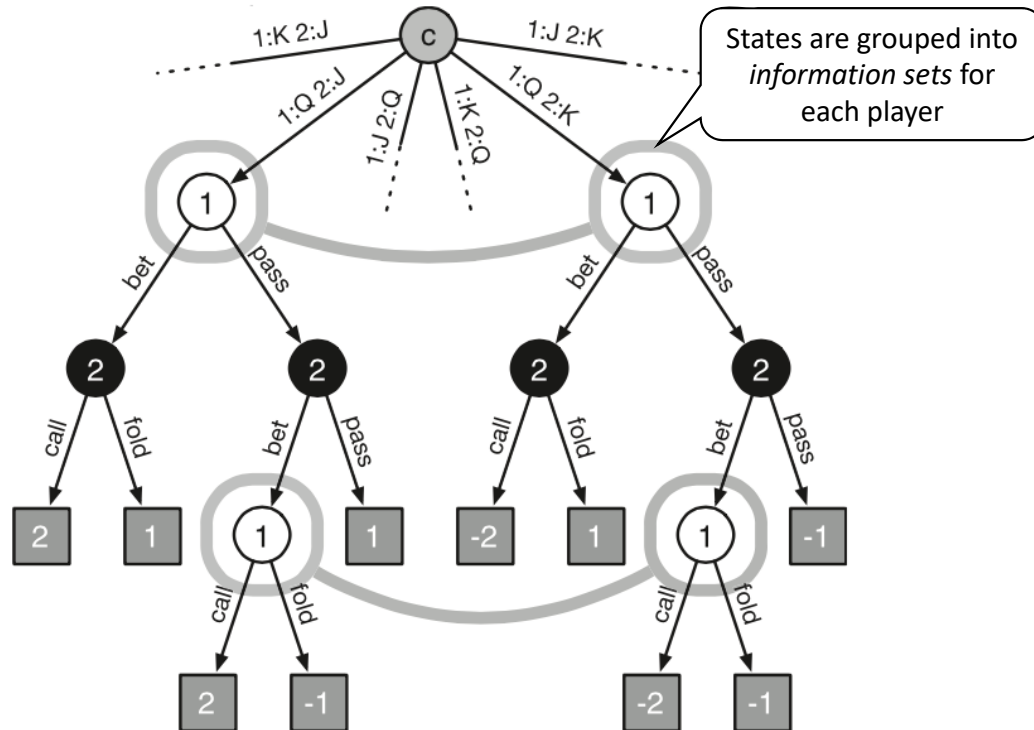
Games of Imperfect Information



Stochastic games of imperfect information

Fig. 1. Portion of the extensive-form game representation of three-card Kuhn poker (16).

Player 1 is dealt a queen (Q), and the opponent is given either the jack (J) or king (K). Game states are circles labeled by the player acting at each state ("c" refers to chance, which randomly chooses the initial deal). The arrows show the events the acting player can choose from, labeled with their in-game meaning. The leaves are square vertices labeled with the associated utility for player 1 (player 2's utility is the negation of player 1's). The states connected by thick gray lines are part of the same information set; that is, player 1 cannot distinguish between the states in each pair because they each represent a different unobserved card being dealt to the opponent. Player 2's states are also in information sets, containing other states not pictured in this diagram.



[Source](#)

Miniminimax with imperfect information

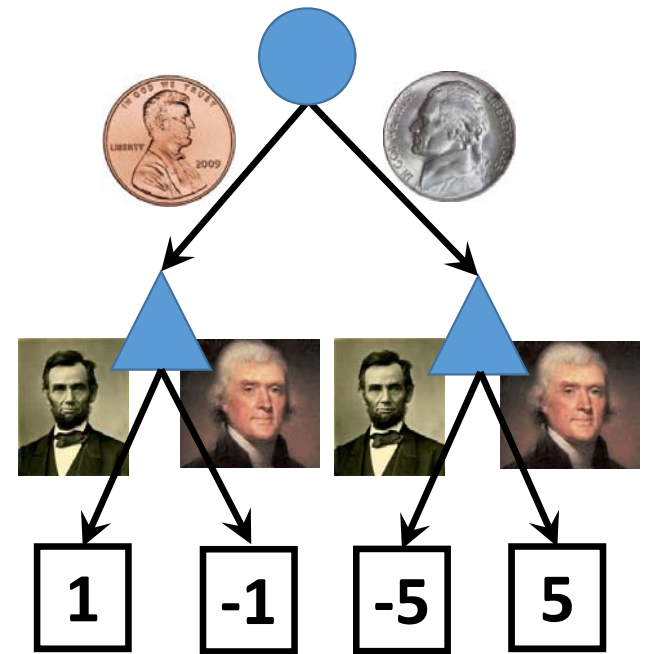
- **Minimax:**

- **Maximize** (over all possible moves I can make) the
- **Minimum**
 - (over all possible states of the information I don't know,
 - ... over all possible moves Min can make) the
- Reward.

$$Value(node) = \max_{my\ moves} \left(\min_{\substack{missing\ info, \\ Min's\ moves}} (Reward) \right)$$

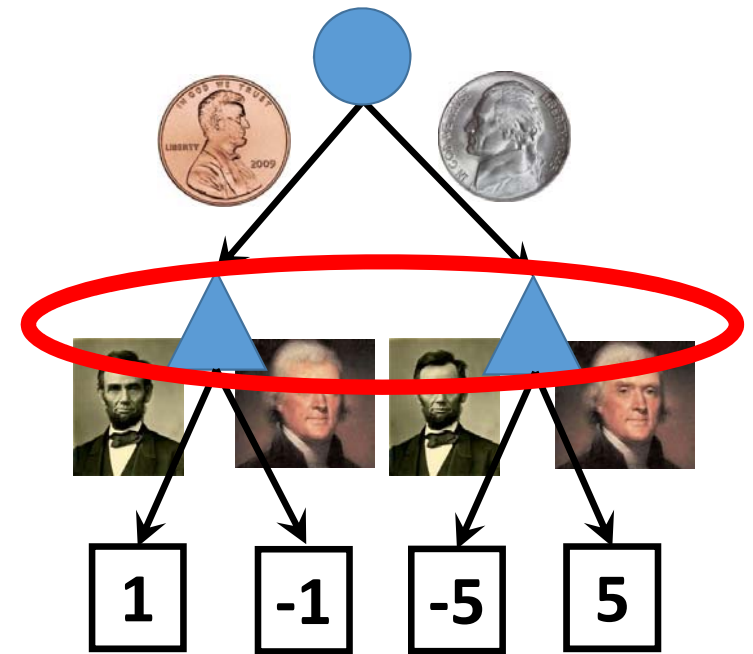
Imperfect information example

- Min chooses a coin.
- I say the name of a U.S. President.
 - If I guessed right, she gives me the coin.
 - If I guessed wrong, I have to give her a coin to match the one she has.





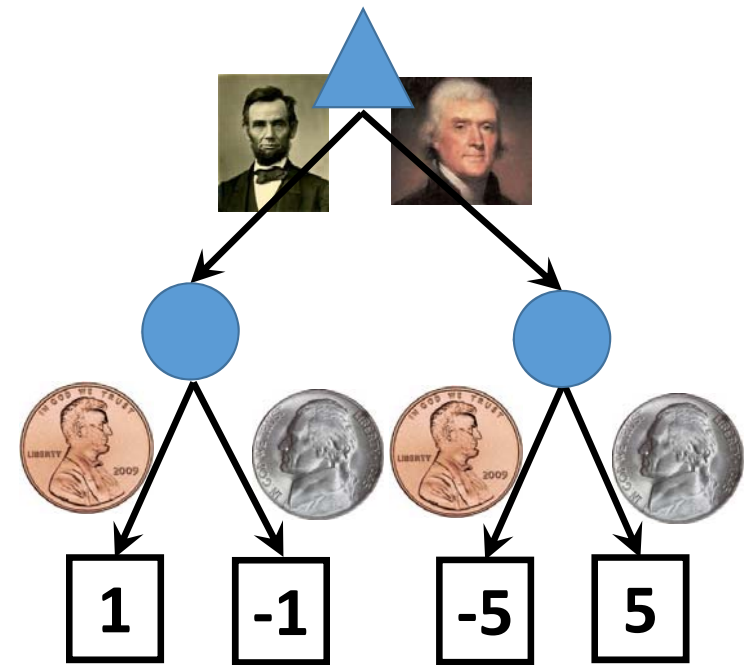
Method #1: Treat “unknown” as “unknown”

- The problem: I don't know which state I'm in. I only know it's one of these two.
- The solution: choose the policy that maximizes my minimum reward.
 - “Lincoln”: minimum reward is -5.
 - “Jefferson”: minimum reward is -1.
- Minimax policy: say “Jefferson”.



Method #2: Treat “unknown” as “random”

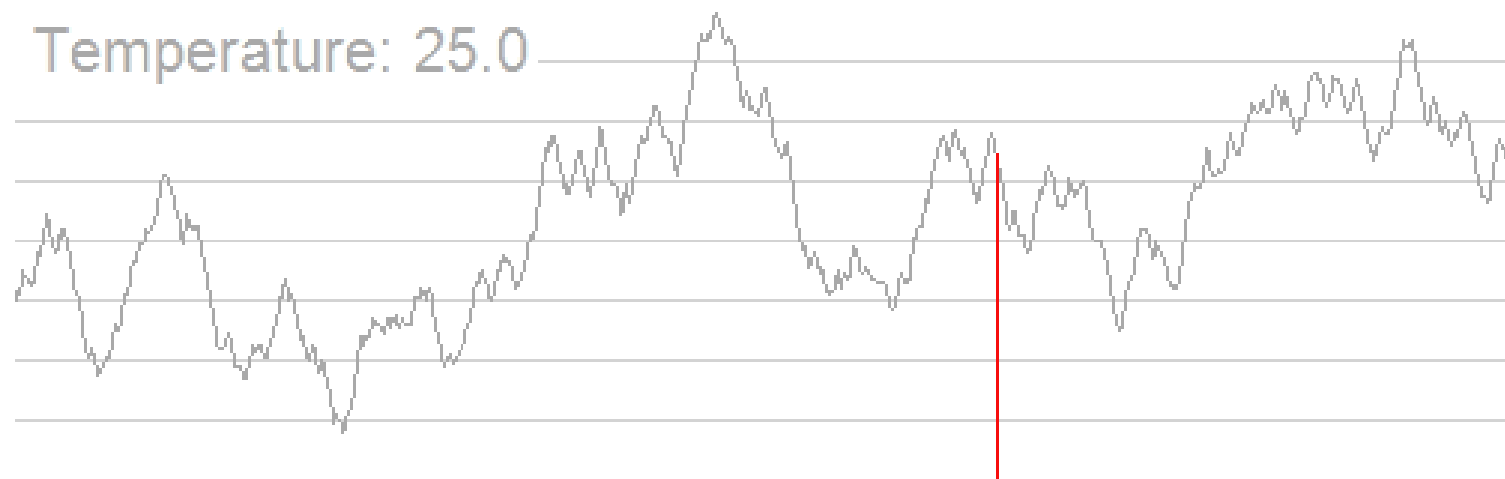
- Expectiminimax: treat the unknown information as random.
- Choose the policy that maximizes my expected reward.
 - “Lincoln”: $\frac{1}{2} \times 1 + \frac{1}{2} \times (-5) = -2$
 - “Jefferson”: $\frac{1}{2} \times (-1) + \frac{1}{2} \times 5 = 2$
- Expectiminimax policy: say “Jefferson”.
- BUT WHAT IF:  and  are not equally likely?



How to deal with imperfect information

- If you think you know the probabilities of different settings, and if you want to maximize your average winnings (for example, you can afford to play the game many times): **expectiminimax**
- If you have no idea of the probabilities of different settings; or, if you can only afford to play once, and you can't afford to lose: **miniminimax**
- If the unknown information has been selected intentionally by your opponent: use **game theory**

Stochastic search



Stochastic search for stochastic games

- The problem with expectiminimax: huge branching factor (many possible outcomes)

$$\mathbb{E}[Reward] = \sum_{outcomes} Probability(outcome) \times Reward(outcome)$$

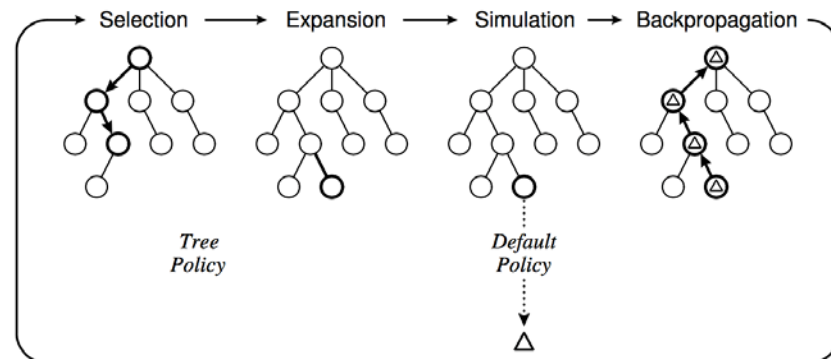
- An approximate solution: Monte Carlo search

$$\mathbb{E}[Reward] \approx \frac{1}{n} \sum_{i=1}^n Reward(i'th\ random\ game)$$

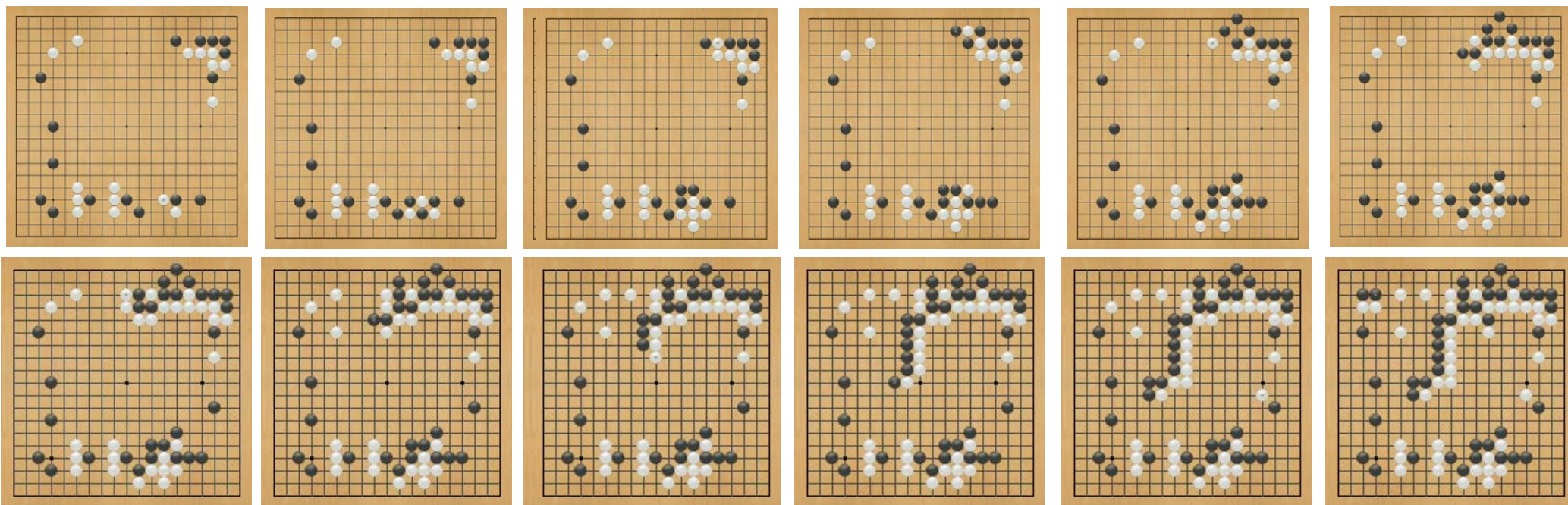
- Asymptotically optimal: as $n \rightarrow \infty$, the approximation gets better.
- Controlled computational complexity: choose n to match the amount of computation you can afford.

Monte Carlo Tree Search

- What about ***deterministic*** games with deep trees, large branching factor, and no good heuristics – like Go?
- Instead of depth-limited search with an evaluation function, use randomized simulations
- Starting at the current state (root of search tree), iterate:
 - Select a leaf node for expansion using a *tree policy* (trading off *exploration* and *exploitation*)
 - Run a simulation using a *default policy* (e.g., random moves) until a terminal state is reached
 - Back-propagate the outcome to update the value estimates of internal tree nodes



Learned evaluation functions



Stochastic search off-line

Training phase:

- Spend a few weeks allowing your computer to play billions of random games from every possible starting state
- Value of the starting state = average value of the ending states achieved during those billion random games

Testing phase:



- During the alpha-beta search, search until you reach a state whose value you have stored in your value lookup table
- Oops.... Why doesn't this work?

Evaluation as a pattern recognition problem

Training phase:

- Spend a few weeks allowing your computer to play billions of random games from billions of possible starting states.
- Value of the starting state = average value of the ending states achieved during those billion random games

Generalization:

- Featurize (e.g., x_1 =number of  patterns, x_2 = number of  patterns, etc.)
- Linear regression: find a_1 , a_2 , etc. so that $\text{Value}(\text{state}) \approx a_1 * x_1 + x_2 * x_2 + \dots$

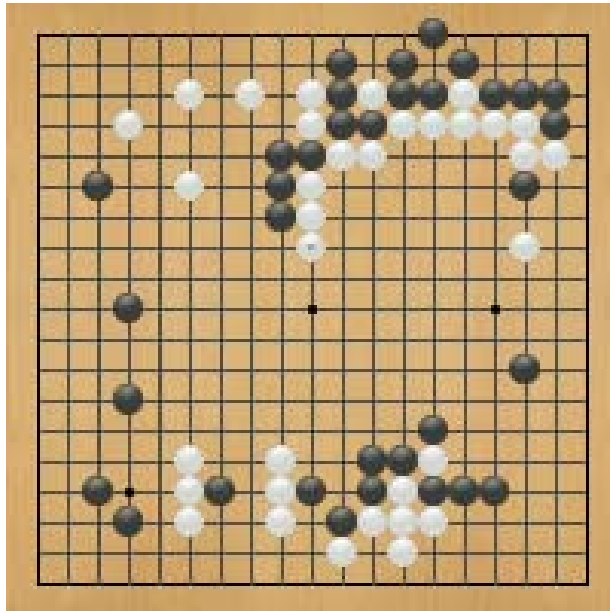
Testing phase:

- During the alpha-beta search, search as deep as you can, then estimate the value of each state at your horizon using $\text{Value}(\text{state}) \approx a_1 * x_1 + x_2 * x_2 + \dots$

Pros and Cons

- Learned evaluation function
 - Pro: off-line search permits lots of compute time, therefore lots of training data
 - Con: there's no way you can evaluate every starting state that might be achieved during actual game play. Some starting states will be missed, so generalized evaluation function is necessary
- On-line stochastic search
 - Con: limited compute time
 - Pro: it's possible to estimate the value of the state you've reached during actual game play

Case study: AlphaGo



- “Gentlemen should not waste their time on trivial games -- they should play go.”
- -- *Confucius*,
- *The Analects*
- *ca. 500 B. C. E.*

Anton Ninno
antonninno@yahoo.com
roylaird@gmail.com

Roy Laird, Ph.D.

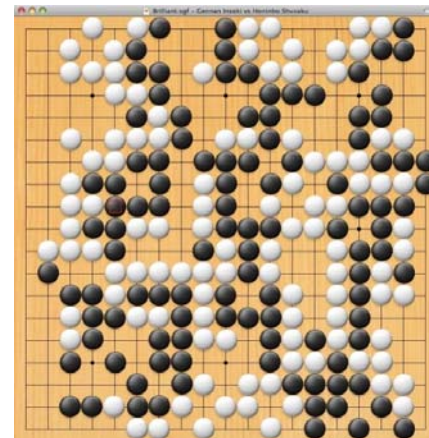
special thanks to Kiseido Publications

Game AI: State of the art

- Computers are better than humans:
 - **Checkers:** [solved in 2007](#)
 - **Chess:**
 - State-of-the-art search-based systems now better than humans
 - [Deep learning machine teaches itself chess in 72 hours, plays at International Master Level](#) (arXiv, September 2015)
- Computers are competitive with top human players:
 - **Backgammon:** [TD-Gammon system](#) (1992) used *reinforcement learning* to learn a good evaluation function
 - **Bridge:** top systems use Monte Carlo simulation and alpha-beta search

Game AI: State of the art

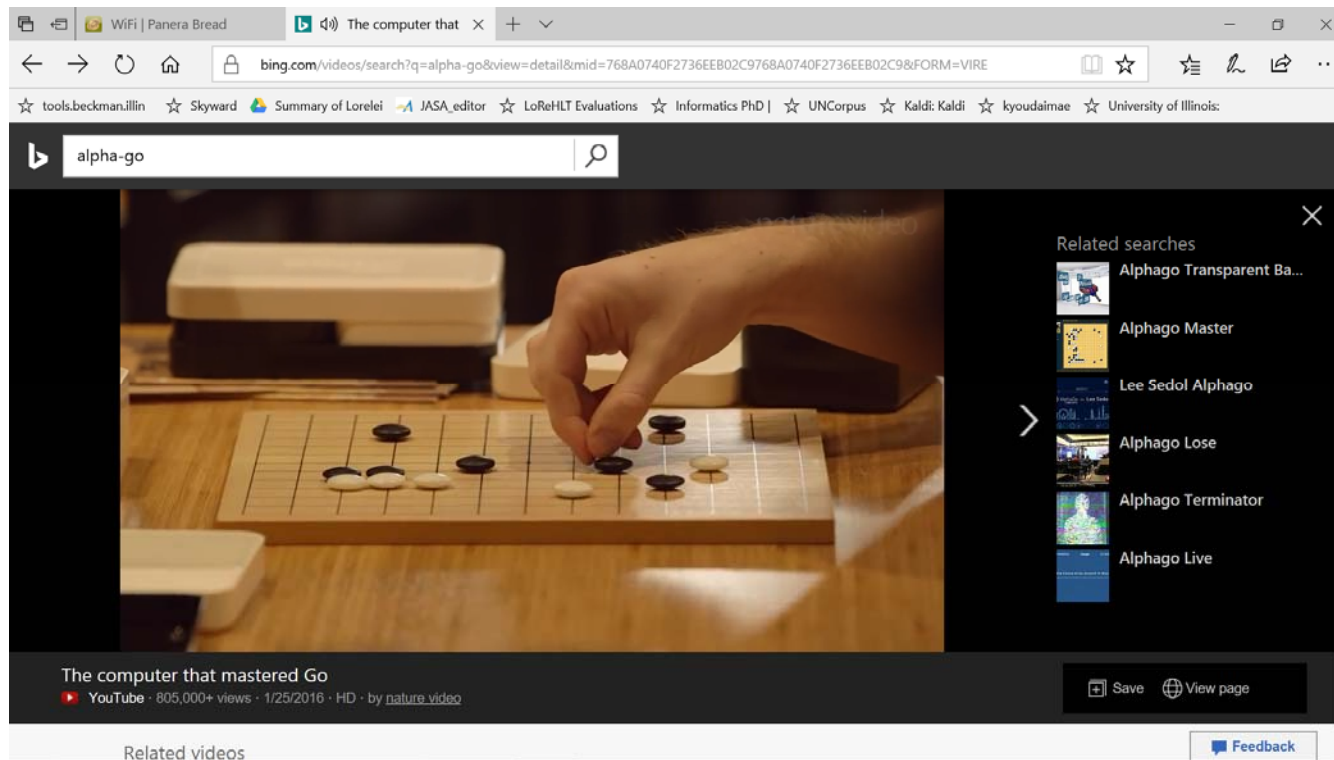
- Computers are not competitive with top human players:
 - **Poker**
 - [Heads-up limit hold'em poker has been solved](#) (Science, Jan. 2015)
 - Simplest variant played competitively by humans
 - Smaller number of states than checkers, but partial observability makes it difficult
 - *Essentially weakly solved* = cannot be beaten with statistical significance in a lifetime of playing
 - Huge increase in difficulty from limit to no-limit poker, but [AI has made progress](#)
 - **Go**
 - Branching factor 361, no good evaluation functions have been found
 - Best existing systems use Monte Carlo Tree Search and pattern databases
 - New approaches: [deep learning](#) (44% accuracy for move prediction, can win against other strong Go AI)

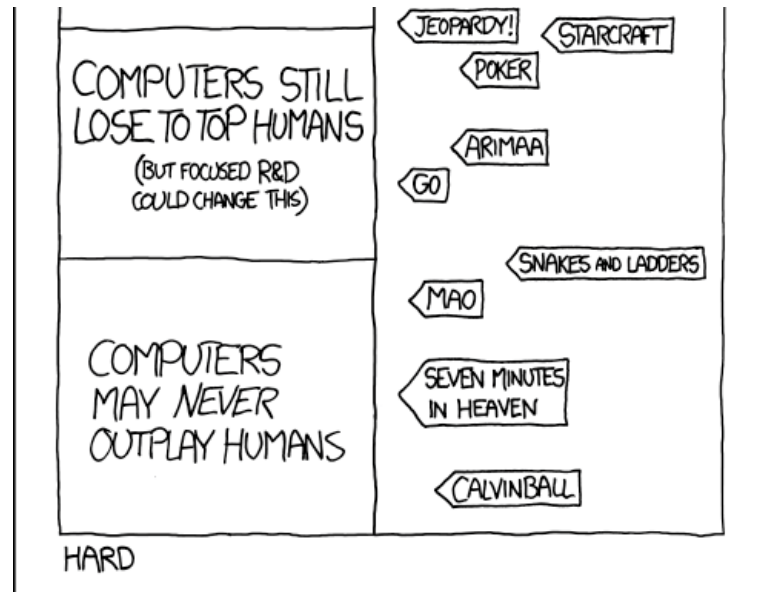
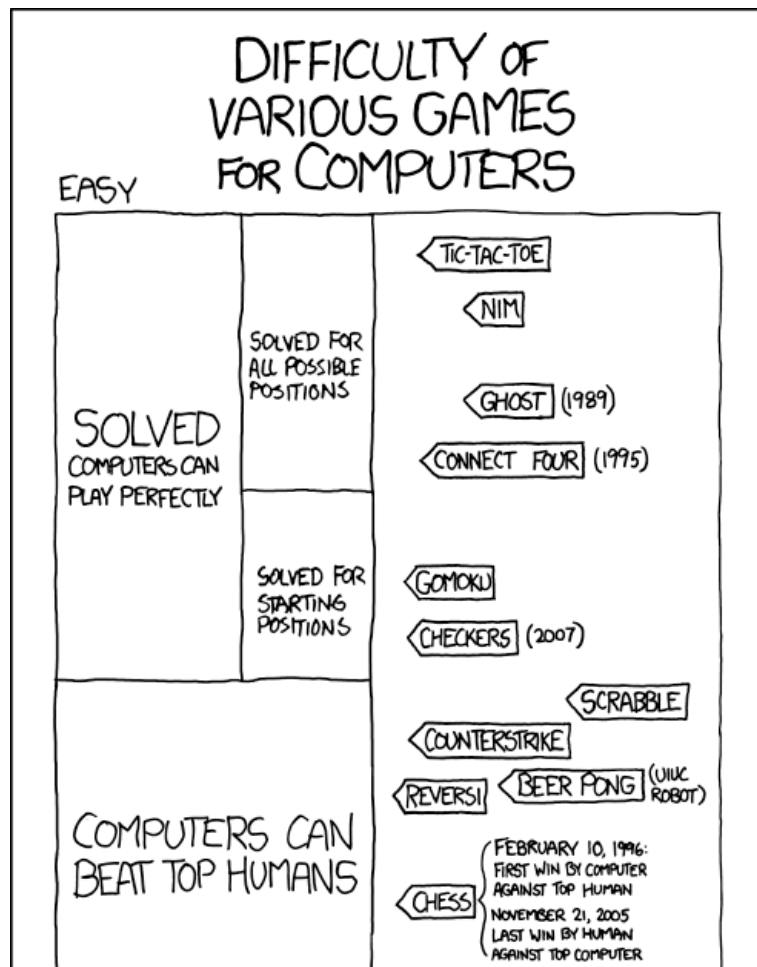


Alpha-Go Innovations

- Learned policy function
 - From training data, learn a 361-dimensional regression $[\pi_1, \dots, \pi_{361}] = \text{function}(\text{features})$.
 - During each round of game play: compute features from the current board position, then plug them into the function to compute $[\pi_1, \dots, \pi_{361}]$. If $\pi_i > 0$, then search the i^{th} move. If $\pi_i < 0$, then don't search it.
 - Branching factor = # positive elements in $[\pi_1, \dots, \pi_{361}]$, which is much less than 361.
- Learned value function
 - From training data, learn a real-valued $\text{Value} = \text{function}(\text{features})$.
 - Search down to a pre-determined search depth, then estimate the value of the horizon state using your value function.
- On-line stochastic search
 - Search to a small depth using exhaustive search, as described above
 - ... then search much deeper, using stochastic search.
 - $\text{Value}(\text{node}) = \text{average}(\text{evaluation function, stochastic search result})$

Alpha-Go video





<http://xkcd.com/1002/>

See also: <http://xkcd.com/1263/>



calvinball



Calvinball:

- [Play it online](#)
- [Watch an instructional video](#)