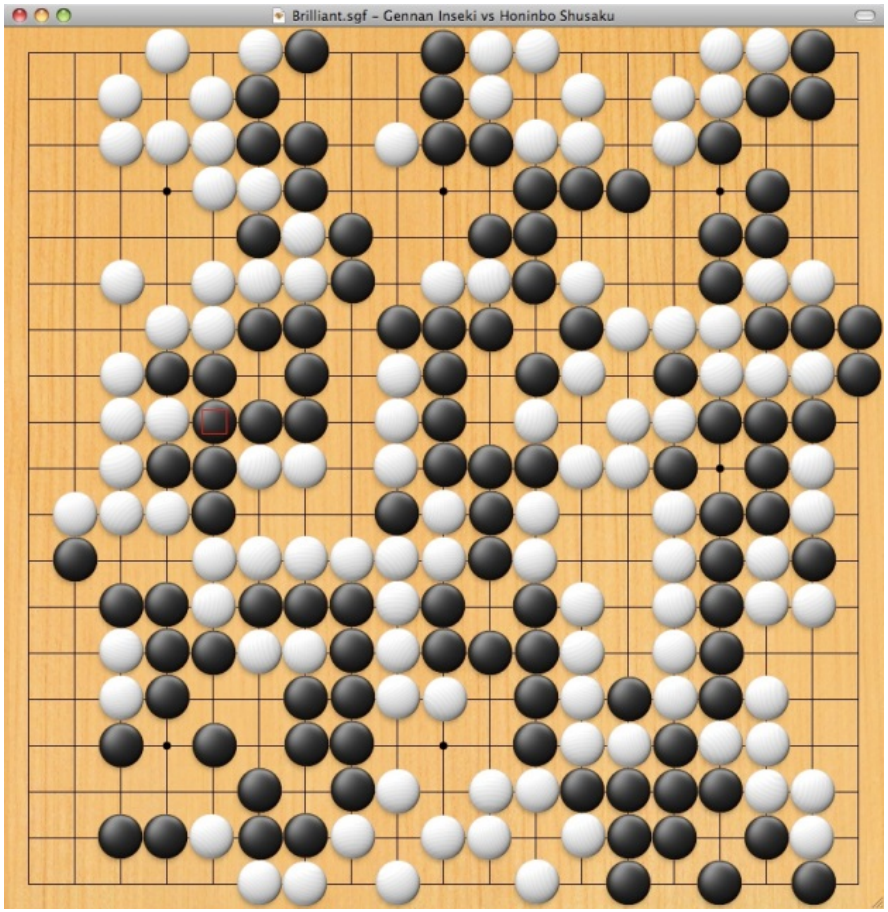
Stochastic tree search and stochastic games

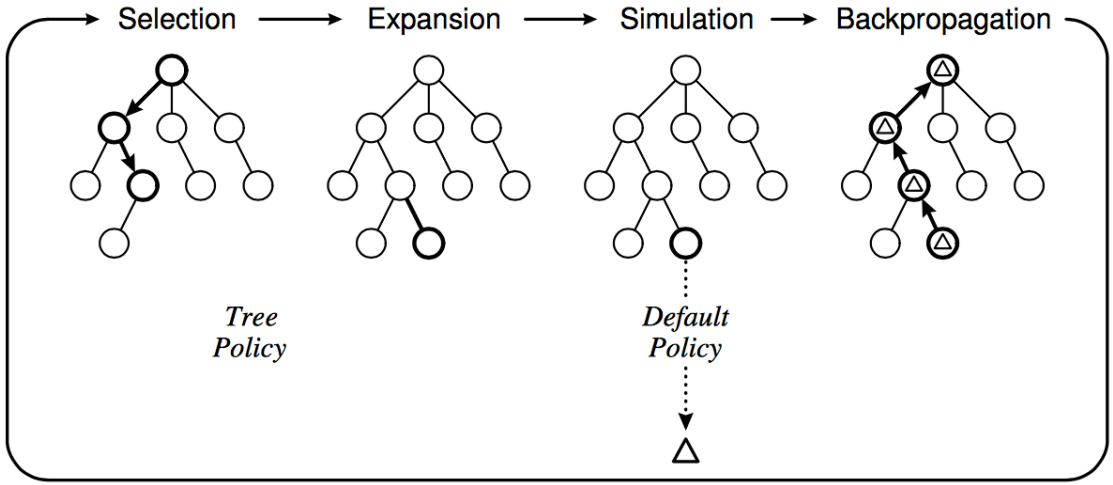
Monte Carlo Tree Search

- Minimax search fails for games with deep trees, large branching factor, and no simple heuristics
  - Go: branching factor 361 (19x19 board)



Monte Carlo Tree Search

- Instead of depth-limited search with an evaluation function, use randomized simulations
- For each simulation:
  - Traverse the tree using a *tree policy* (trading off *exploration* and *exploitation*) until a leaf node is reached
  - Run a *rollout* using a *default policy* (e.g., random moves) until a terminal state is reached
  - Update the value estimates of nodes along the path to reflect the win percentage of simulations passing through that node

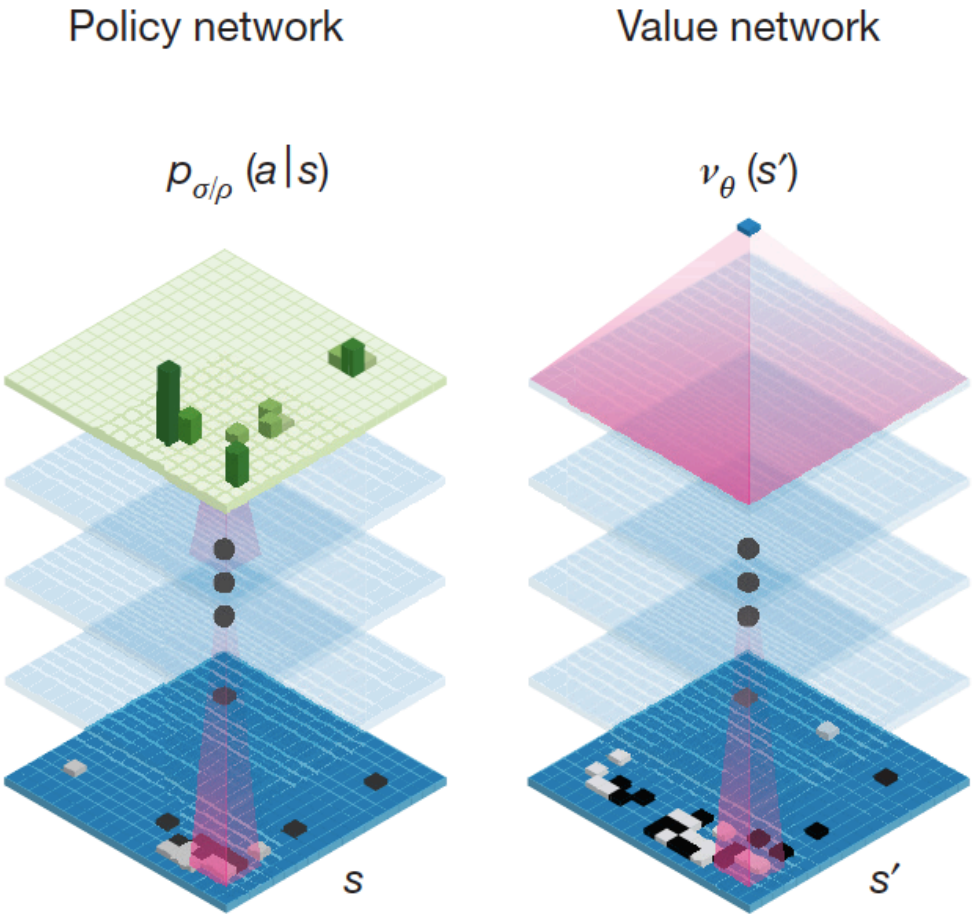


C. Browne et al., A survey of Monte Carlo Tree Search Methods, 2012

AlphaGo

https://deepmind.com/research/alphago/

AlphaGo



- Deep convolutional neural networks
  - Treat the Go board as an image
  - Powerful function approximation machinery
  - Can be trained to predict distribution over possible moves (*policy*) or expected *value* of position

D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

AlphaGo

- SL policy network
  - Idea: perform *supervised learning* (SL) to predict human moves
  - Given state s, predict probability distribution over moves a, P(a|s)
  - Trained on 30M positions, 57% accuracy on predicting human moves
  - Also train a smaller, faster *rollout policy* network (24% accurate)
- RL policy network
  - Idea: fine-tune policy network using *reinforcement learning* (RL)
  - Initialize RL network to SL network
  - Play two snapshots of the network against each other, update parameters to maximize expected final outcome
  - RL network wins against SL network 80% of the time, wins against open-source Pachi Go program 85% of the time

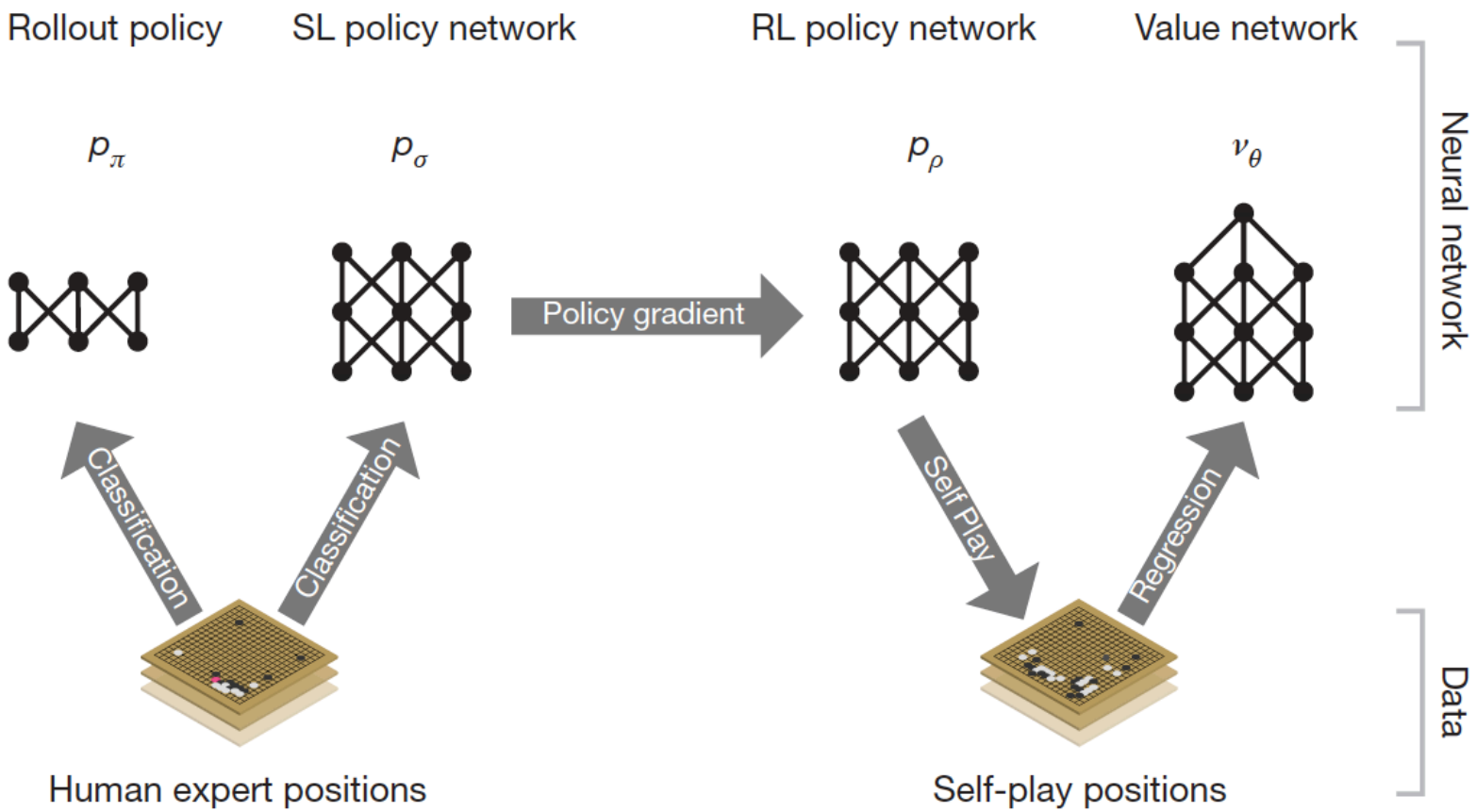
D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

AlphaGo

- SL policy network
- RL policy network
- Value network
  - Idea: train network for position evaluation
  - Given state s, estimate v(s), expected outcome of play starting with position s and following the learned policy for both players
  - Train network by minimizing mean squared error between actual and predicted outcome
  - Trained on 30M positions sampled from different self-play games

D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

AlphaGo



D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

AlphaGo

- Monte Carlo Tree Search
  - Each edge in the search tree maintains *prior probabilities* P(s,a), *counts* N(s,a), *action values* Q(s,a)
  - P(s,a) comes from SL policy network
  - Tree traversal policy selects actions that maximize Q value plus exploration bonus (proportional to P but inversely proportional to N)
  - An expanded leaf node gets a value estimate that is a combination of value network estimate and outcome of simulated game using rollout network
  - At the end of each simulation, Q values are updated to the average of values of all simulations passing through that edge

D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

AlphaGo

- Monte Carlo Tree Search

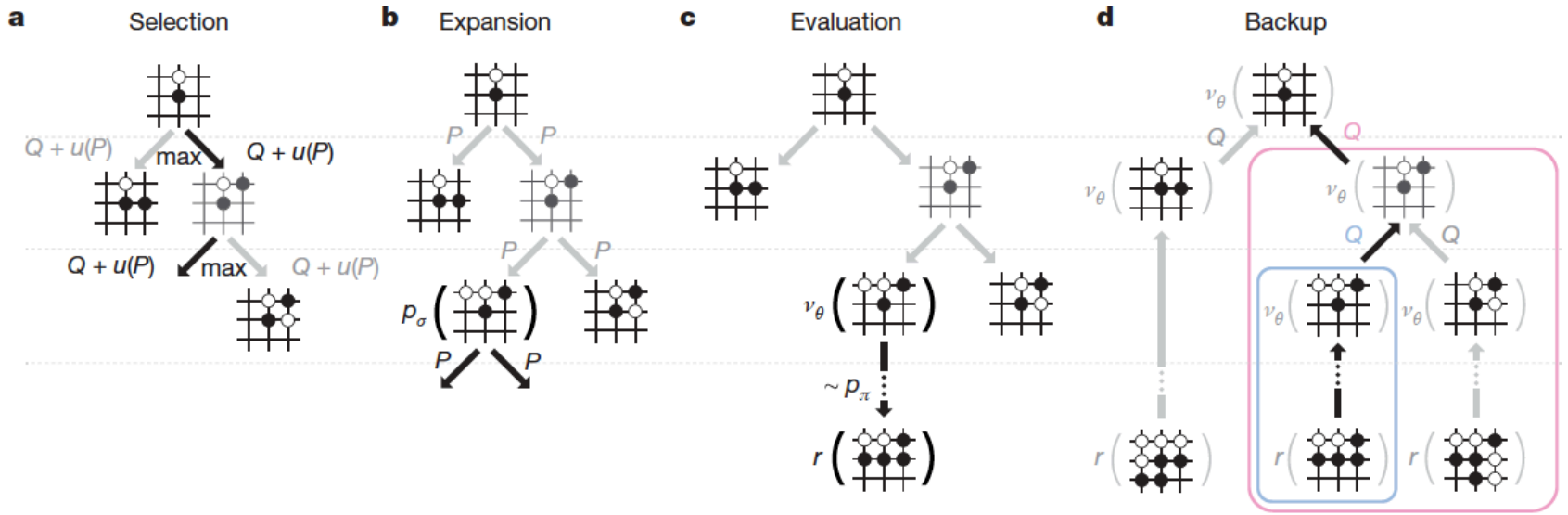


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

AlphaGo

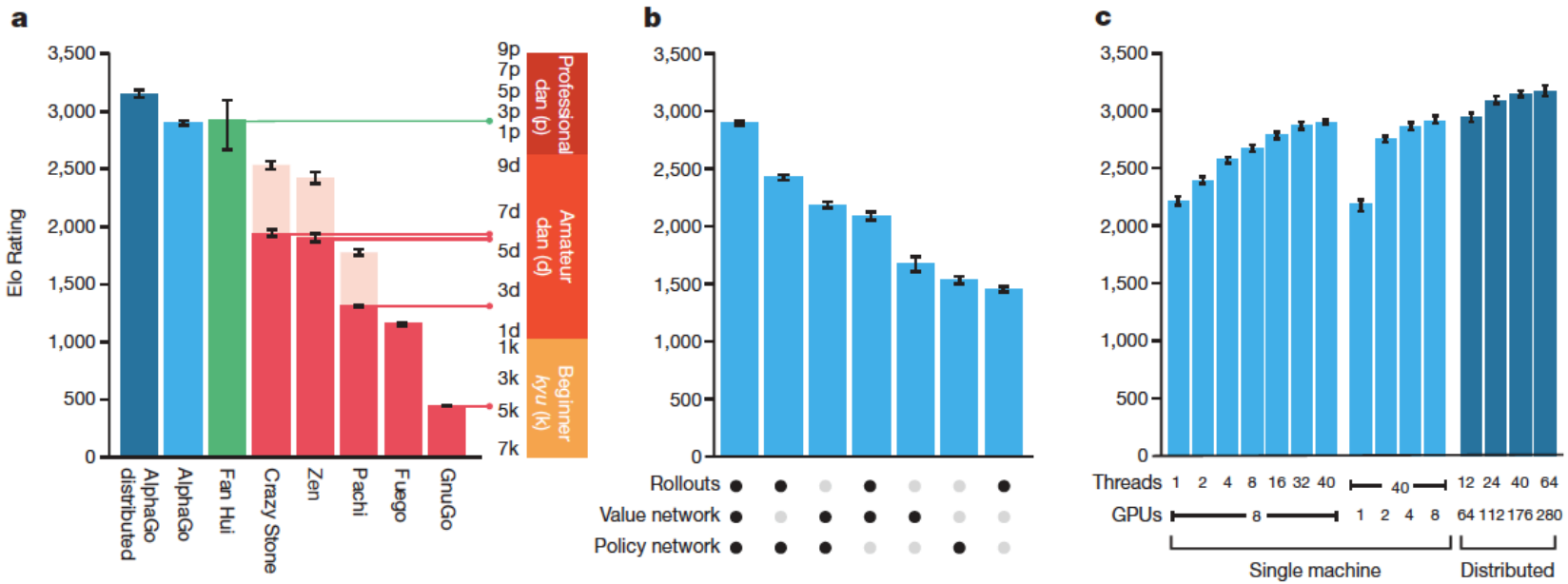


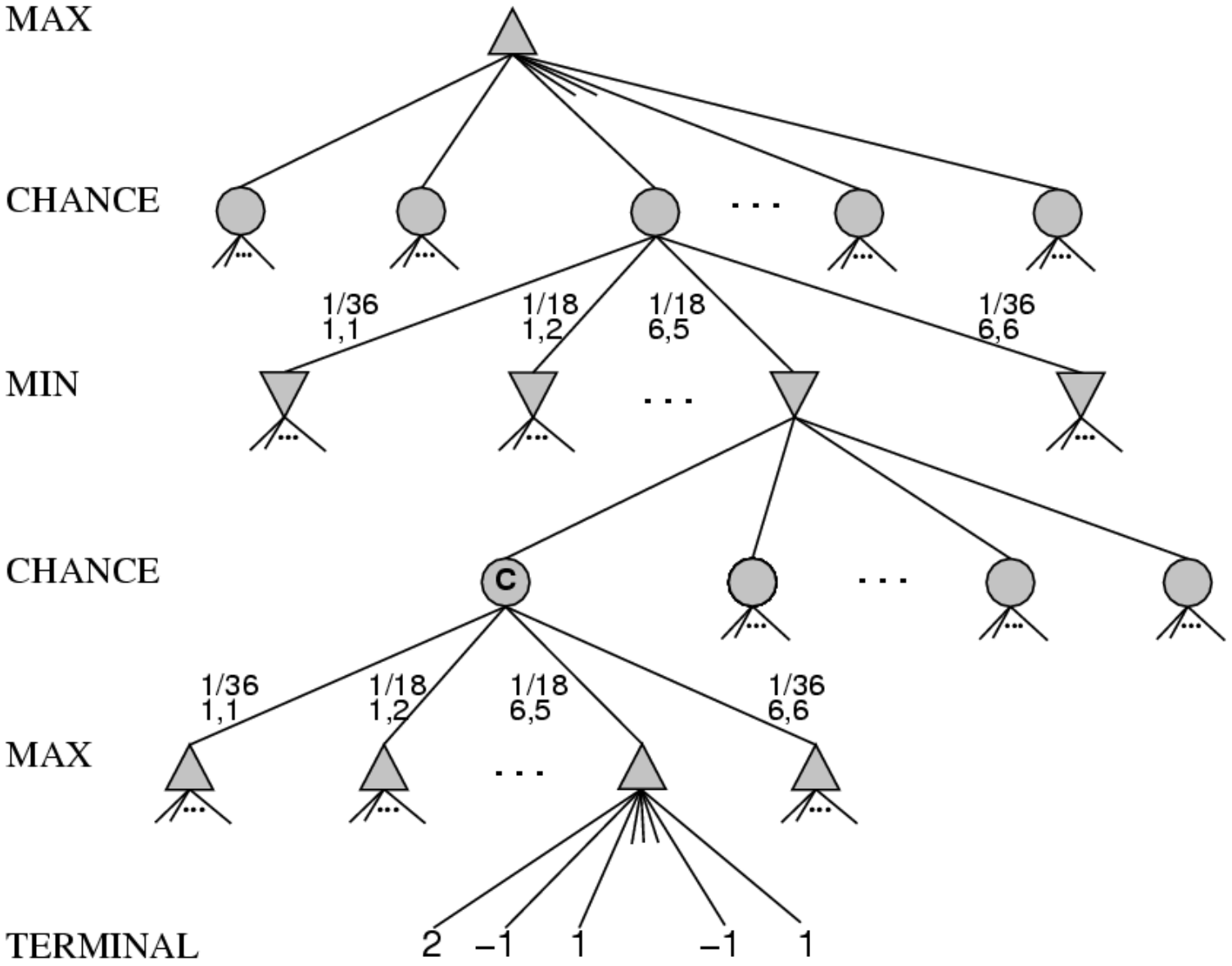
Figure 4 | Tournament evaluation of AlphaGo. a, Results of a tournament between different Go programs (see Extended Data Tables 6–11). Each program used approximately 5 s computation time per move. To provide a greater challenge to AlphaGo, some programs (pale upper bars) were given four handicap stones (that is, free moves at the start of every game) against all opponents. Programs were evaluated on an Elo scale[37]: a 230 point gap corresponds to a 79% probability of winning, which roughly corresponds to one amateur *dan* rank advantage on KGS[38]; an approximate correspondence to human ranks is also shown, horizontal lines show KGS ranks achieved online by that program. Games against the human European champion Fan Hui were also included; these games used longer time controls. 95% confidence intervals are shown. **b**, Performance of AlphaGo, on a single machine, for different combinations of components. The version solely using the policy network does not perform any search. **c**, Scalability study of MCTS in AlphaGo with search threads and GPUs, using asynchronous search (light blue) or distributed search (dark blue), for 2 s per move.

D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016

Stochastic games

- How to incorporate dice throwing into the game tree?

Stochastic games



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
  - $\max_{action}$ **Value**(Succ(*node, action*)) if *type* = MAX
  - $\min_{action}$ **Value**(Succ(*node, action*)) if *type* = MIN
  - sum_{action} P(Succ(*node, action*)) * **Value**(Succ(*node, action*)) if *type* = CHANCE

Stochastic games

- **Expectiminimax:** for chance nodes, sum values of successor states weighted by the probability of each successor
  - Nasty branching factor, defining evaluation functions and pruning algorithms more difficult
- **Monte Carlo simulation:** when you get to a chance node, simulate a large number of games with random dice rolls and use win percentage as evaluation function
  - Can work well for games like Backgammon

Stochastic games of imperfect information

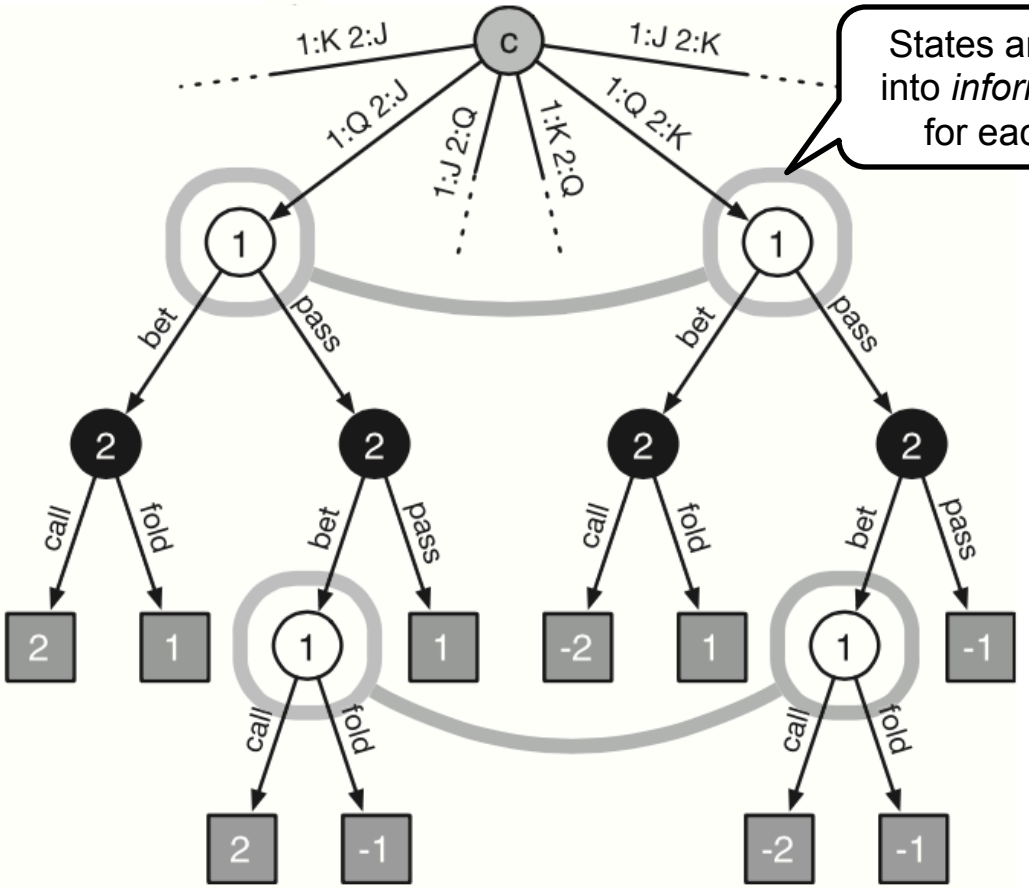


Fig. 1. Portion of the extensive-form game representation of three-card Kuhn poker (*16*). Player 1 is dealt a queen (Q), and the opponent is given either the jack (J) or king (K). Game states are circles labeled by the player acting at each state ("c" refers to chance, which randomly chooses the initial 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

Stochastic games of imperfect information

- Simple Monte Carlo approach: run multiple simulations with random cards pretending the game is fully observable
  - "Averaging over clairvoyance"
  - Problem: this strategy does not account for bluffing, information gathering, etc.

Game AI: Origins

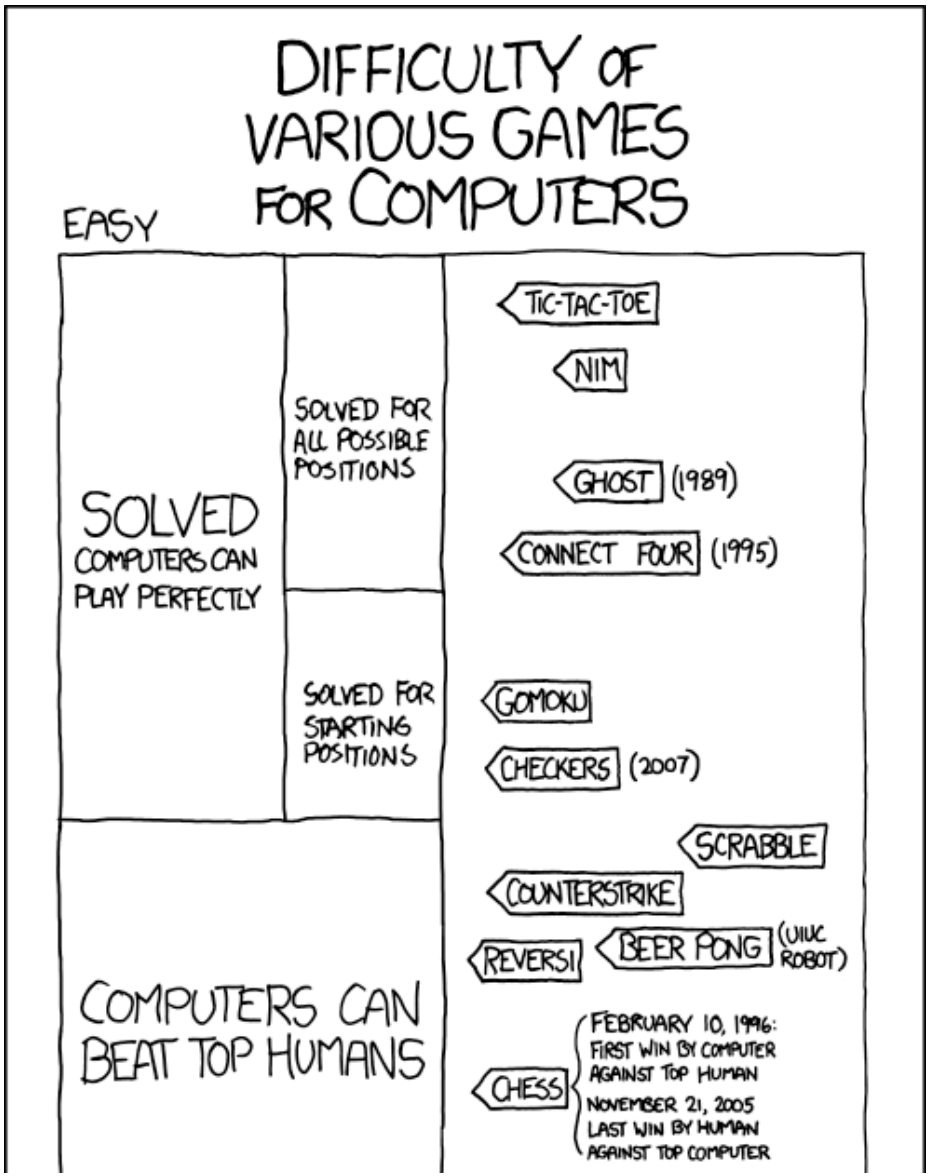
- Minimax algorithm: Ernst Zermelo, 1912
- Chess playing with evaluation function, quiescence search, selective search: Claude Shannon, 1949 (paper)
- Alpha-beta search: John McCarthy, 1956
- Checkers program that learns its own evaluation function by playing against itself: Arthur Samuel, 1956

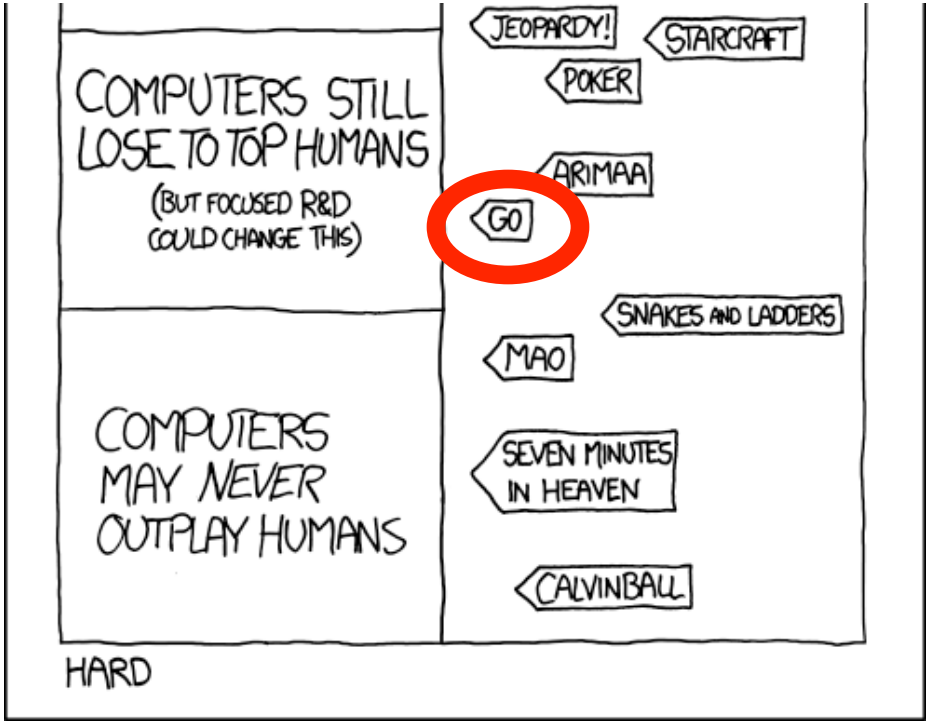
Game AI: State of the art

- 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
  - **Go:** computers were not considered competitive until AlphaGo in 2016

Game AI: State of the art

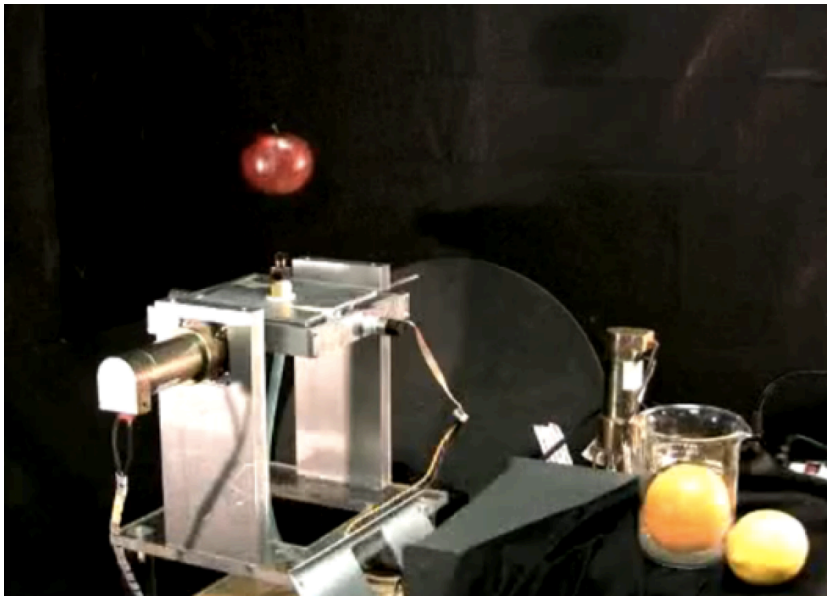
- Computers are not competitive with top human players:
  - **Poker**
    - Heads-up limit hold'em poker has been solved (Science, Jan. 2015)
      - Simplest variant played competitively by humans
      - Smaller number of states than checkers, but partial observability makes it difficult
      - *Essentially 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





http://xkcd.com/1002/

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



UIUC robot (2009)