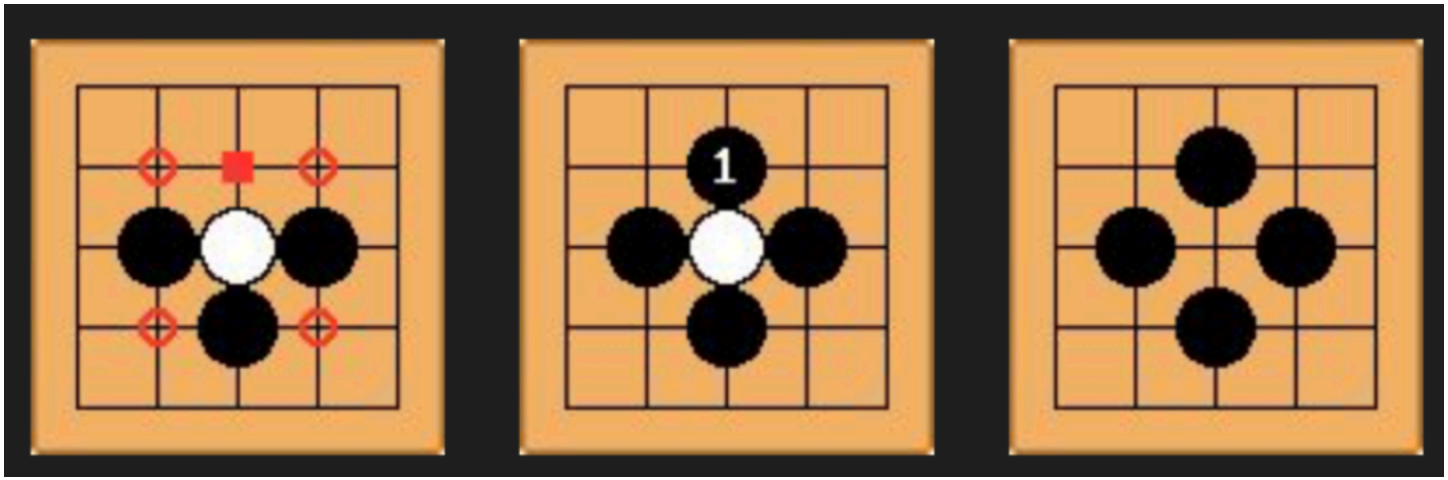
AlphaGo

Go in numbers

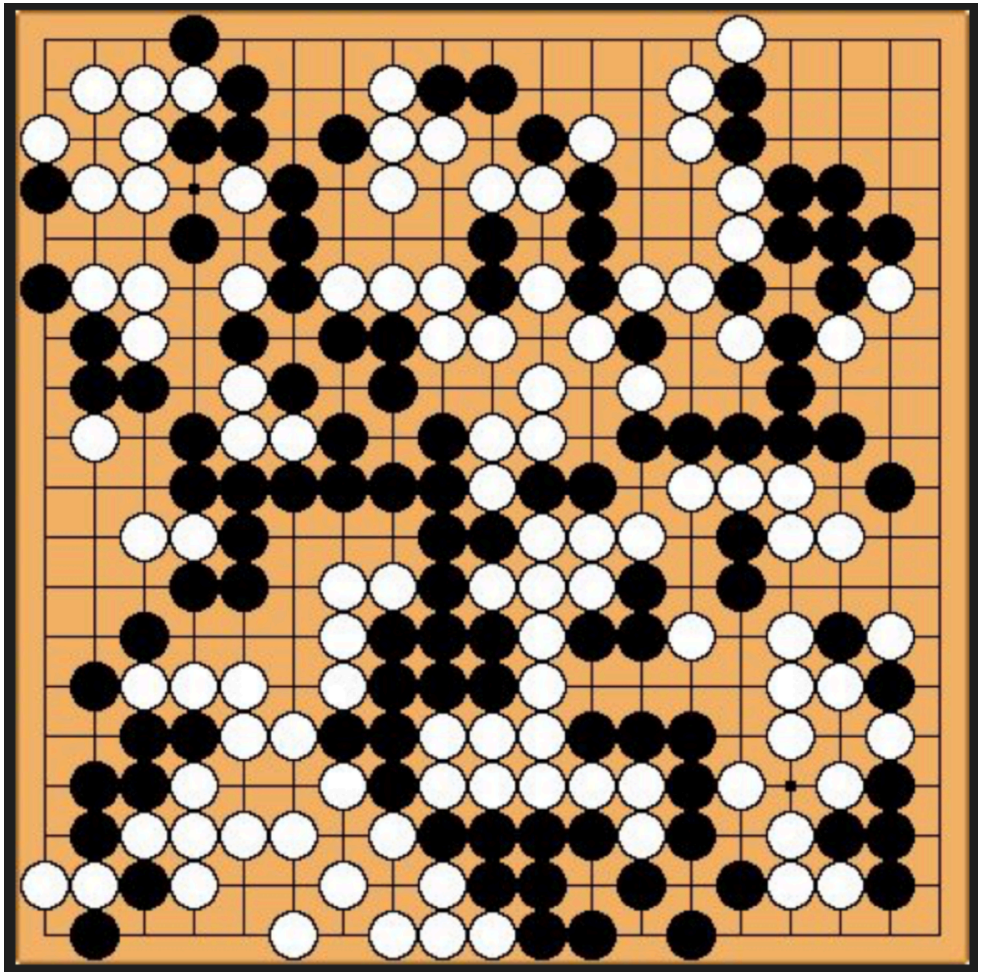
Rules of Go

- Played on a 19x19 board
- Two players, black and white, each place one stone per turn
- Capture the opponent's stones by surrounding them



Rules of Go

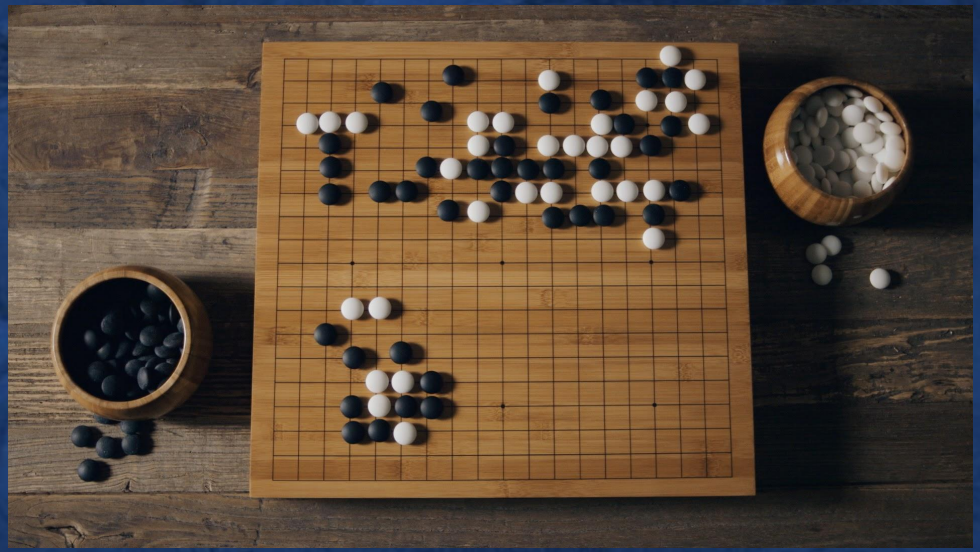
- Goal is to control as much territory as possible.



Why is Go hard for computers to play?

Brute force search intractable:

1. Search space is huge
2. “Impossible” for computers to evaluate who is winning



Game tree complexity = b^d

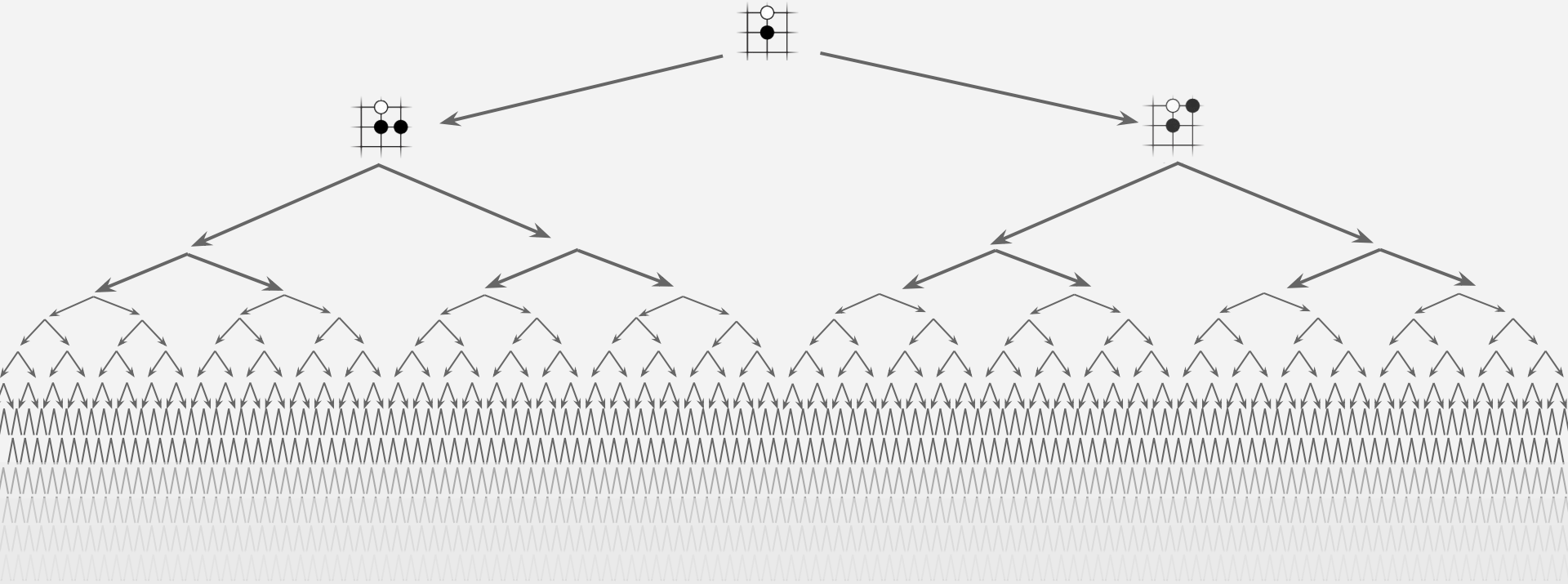
Techniques behind AlphaGo

- Deep learning + Monte Carlo Tree Search + High Performance Computing
- Learn from 30 million human expert moves and 128,000+ self play games

Image: AP/ Lee Jin-man

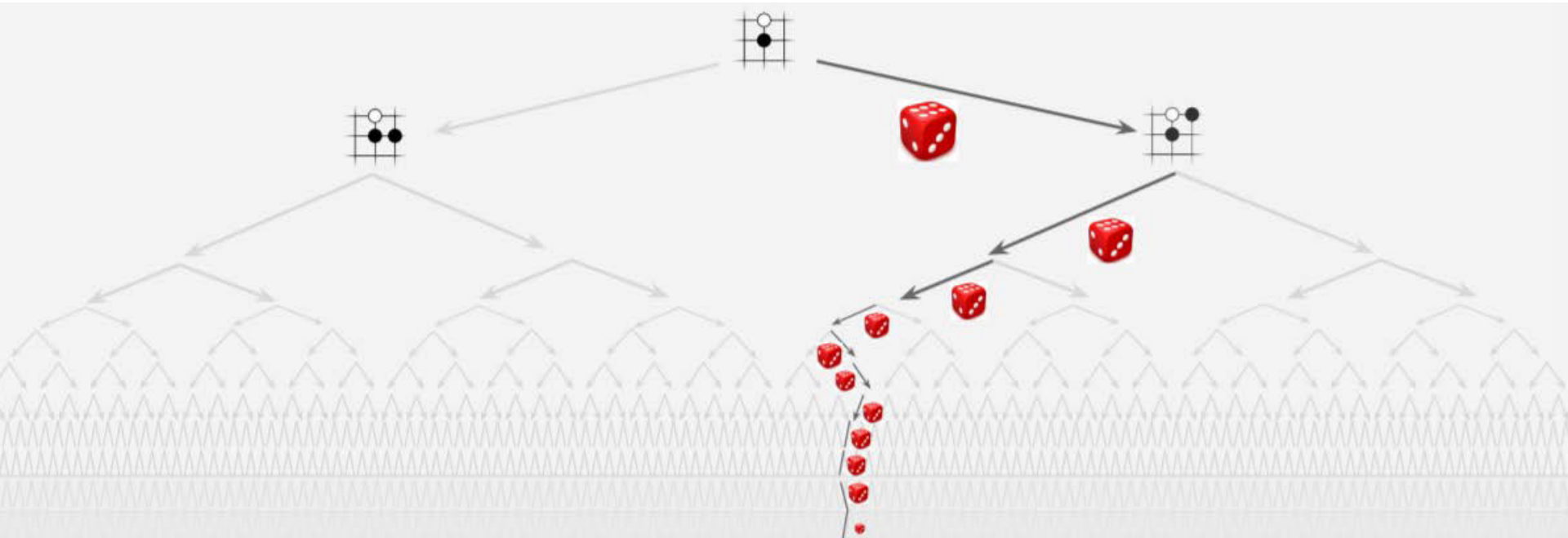
March 2016:
AlphaGo beats Lee Sedol 4-1

Exhaustive search



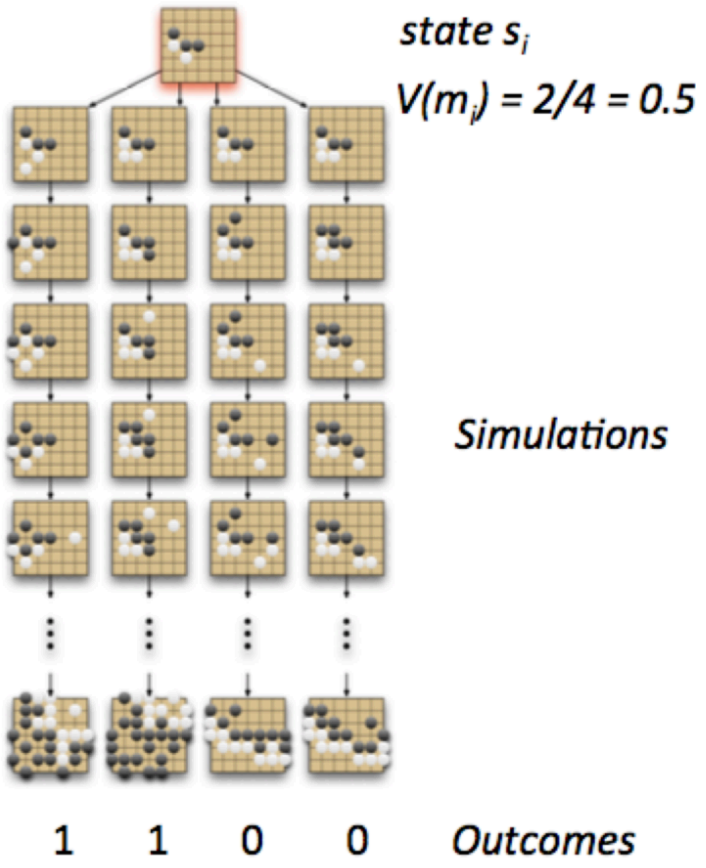
Monte-Carlo Tree Search

- Heuristic search algorithm for decision trees
- Application to deterministic game pretty recent (less than 10 years)



Basic Idea

- No evaluation function?
  - Simulate game using random moves
  - Score game at the end, keep winning statistics
  - Play move with best winning percentage
  - Repeat

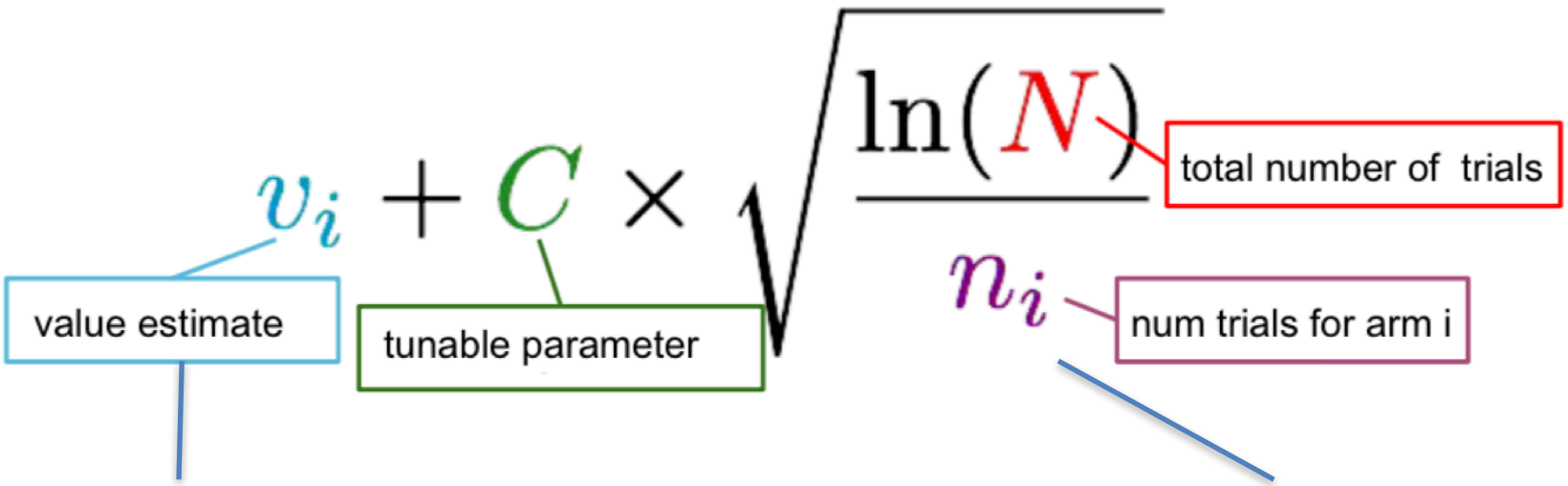


Monte Carlo Tree Search

- Use results of simulation to guide the growth of the game tree
- What moves are interesting to us?
  - Promising moves (simulated and won most)
  - Moves where uncertainty about evaluation are high (less simulated)
- Seems two contradictory goals
  - Theory of bandits can help

Upper Confidence Bound

- Policy
  - First, try each arm once
  - Then, at each time step
    - Choose the arm that maximizes formula:

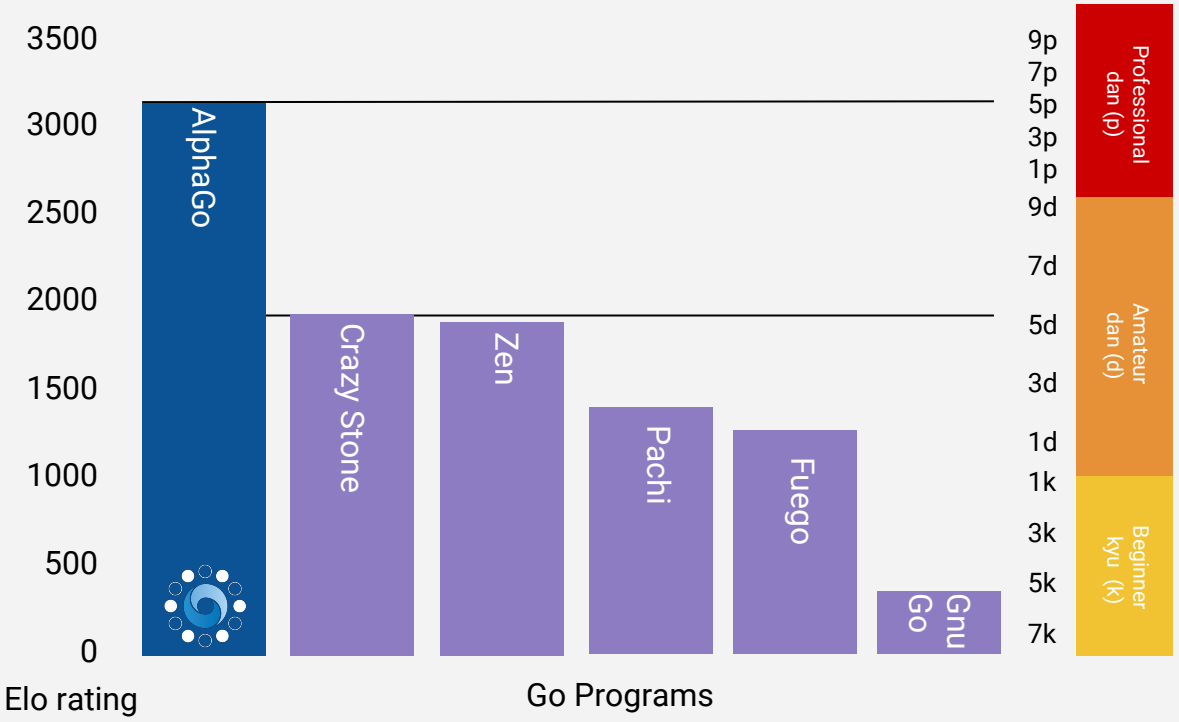


Evaluating Nature AlphaGo against computers

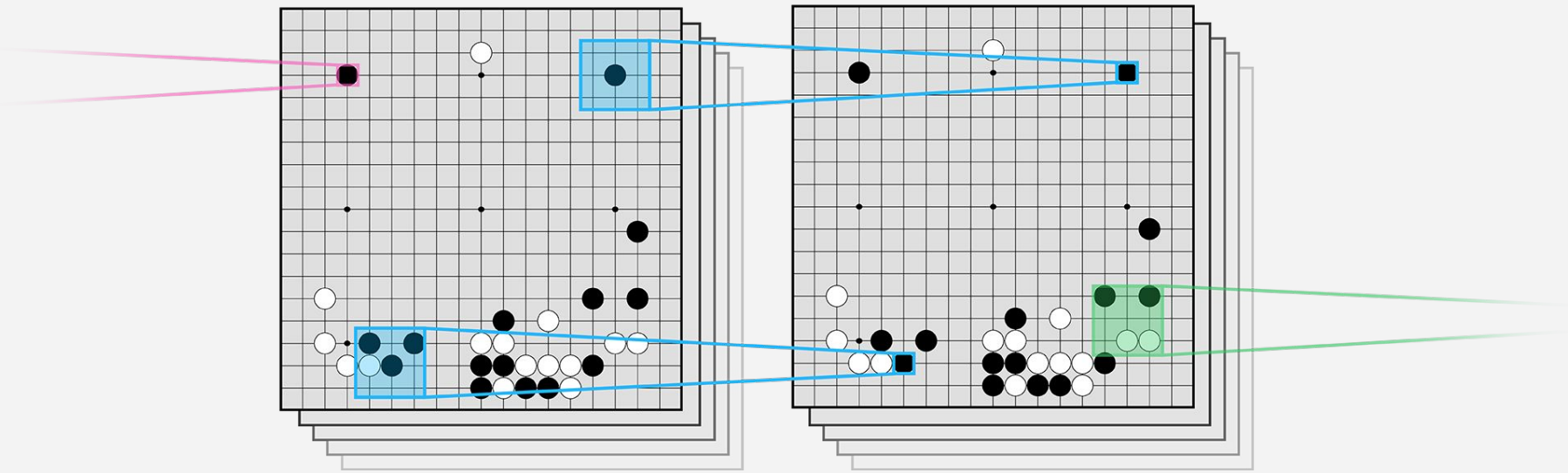
494/495 against computer opponents

>75% winning rate with 4 stone handicap

Even stronger using distributed machines



Convolutional neural network

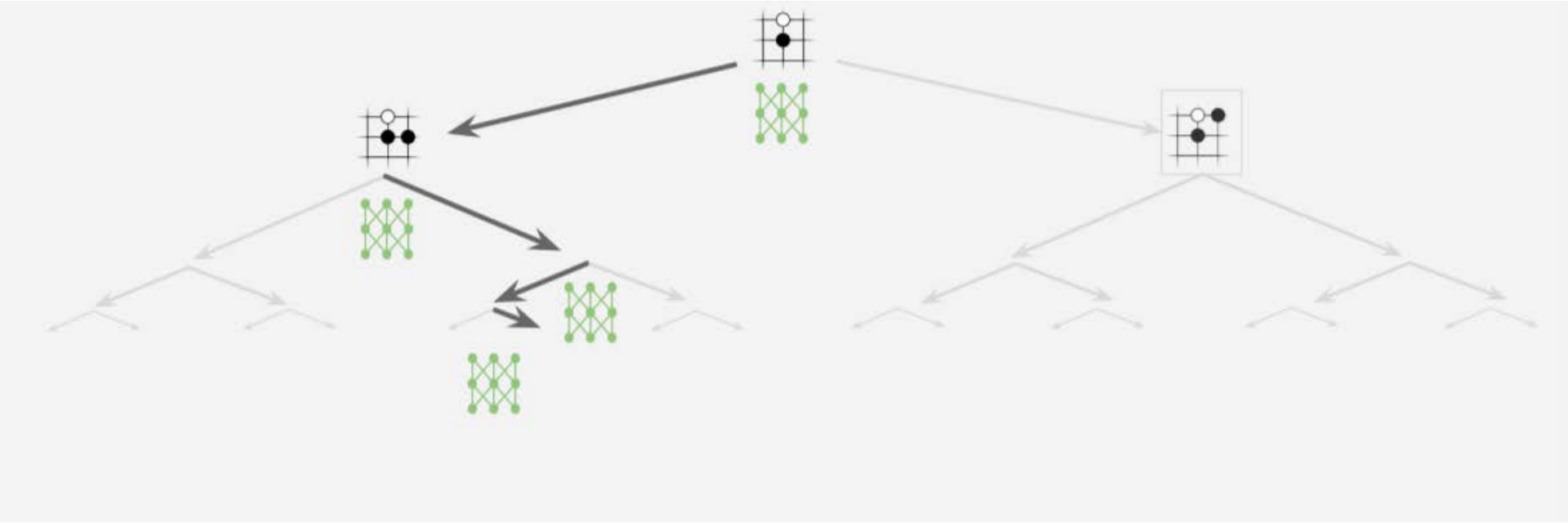


Policy and Value Networks

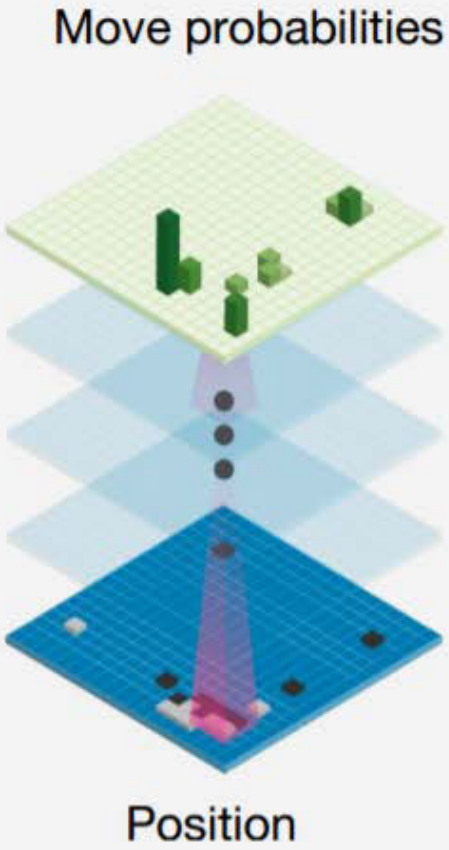
- Goal: Reduce both branching factor and depth of search tree
- How?
  - Use policy network to explore better (and fewer) moves
    - How?
  - Use value network to estimate lower branches of tree (rather than simulating to the end)
    - How?

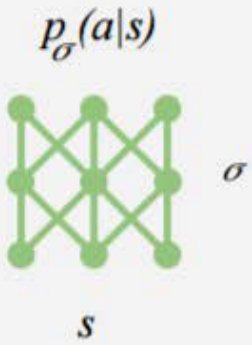
Policy and Value Networks

- Reducing branching factor: Policy Network



Policy and Value Networks

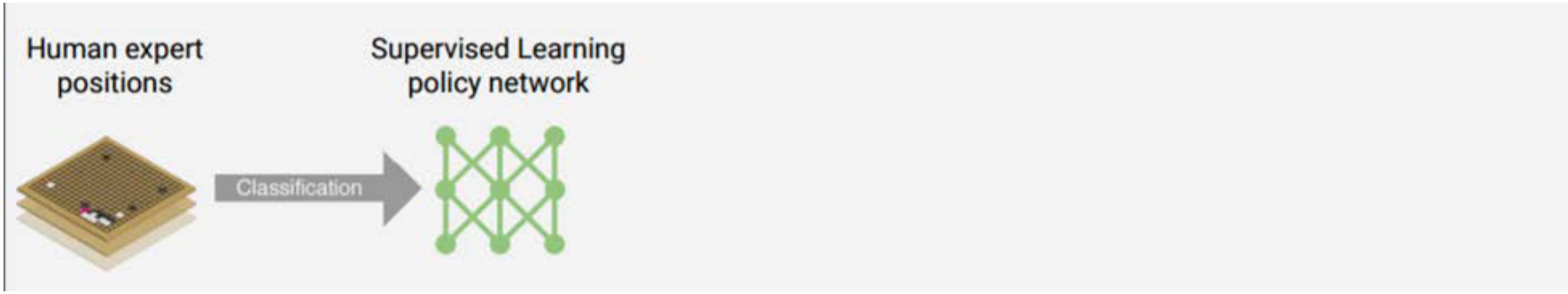




Predicts the probability of a move being best move

Policy and Value Networks

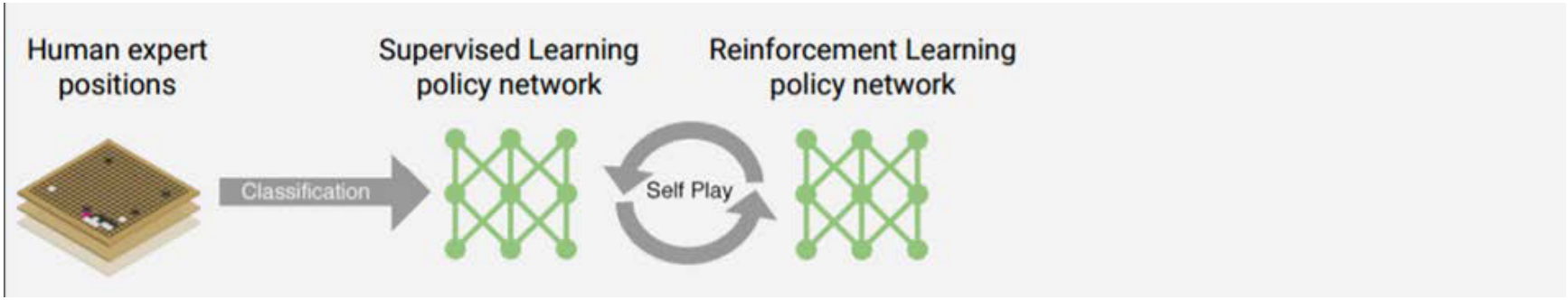
- Supervised learning



- Training data: 30 million positions from human expert games
- Likelihood of a human move selected at a state *s*
- Training time: 4 weeks
- Results: predicted human expert moves with 57% accuracy

Policy and Value Networks

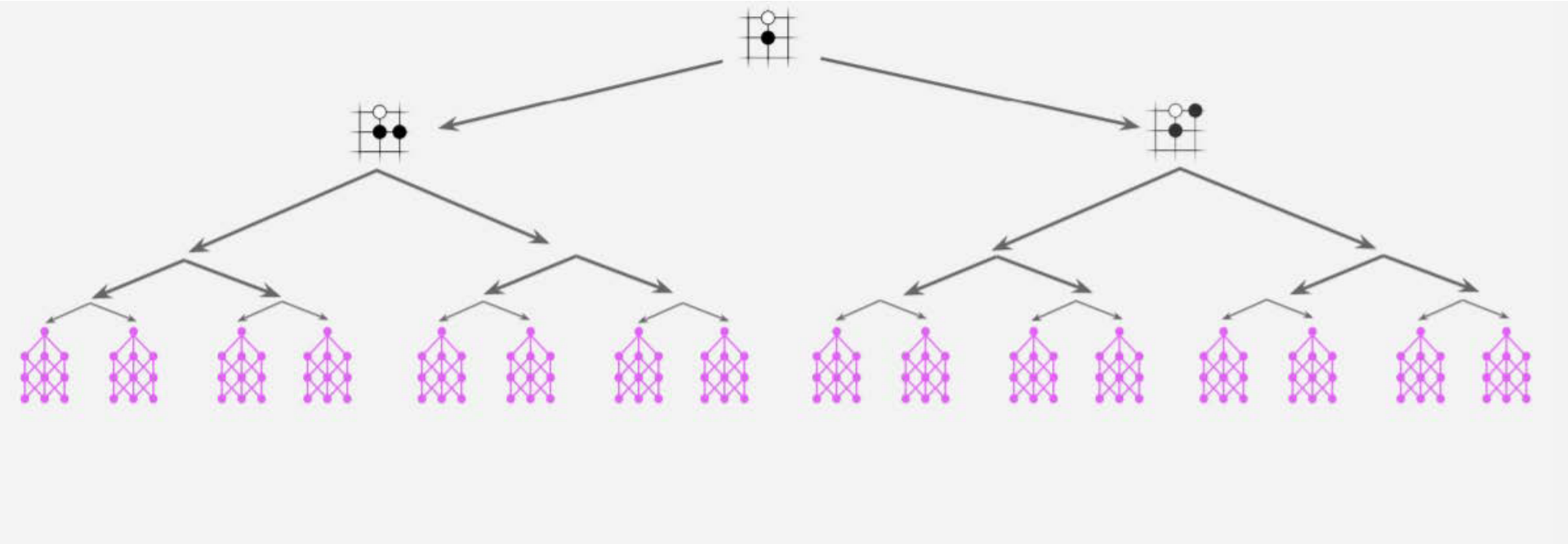
- Reinforcement learning



- Training data: 128,000+ games of self-play using policy network in 2 stages
- Training algorithm: maximize wins of the action $\Delta\boldsymbol{\sigma}$
- Training time: 1 week
- Results: won more than 80% games vs. supervised learning

Policy and Value Networks

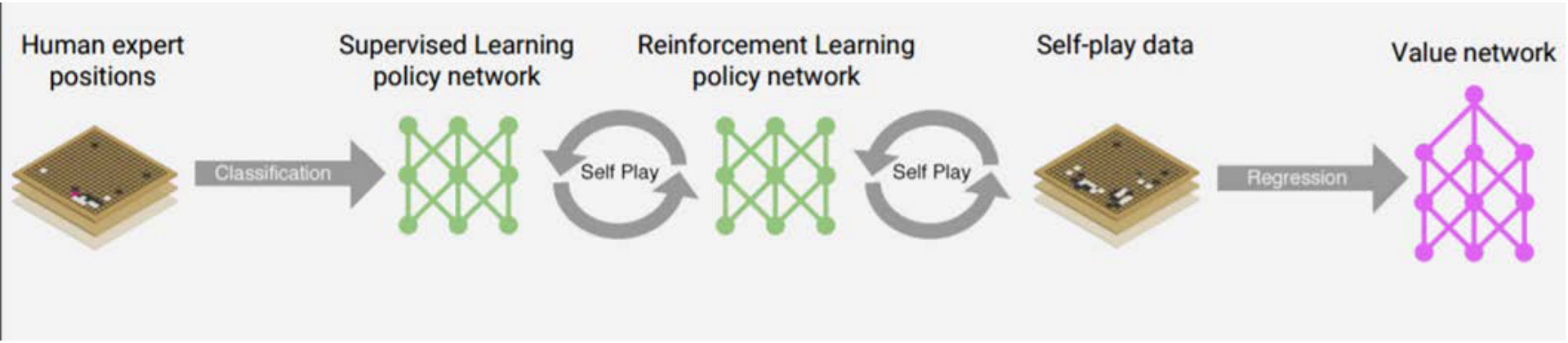
- Reducing depth: Value Network



- Given board states, estimate probability of victory
- No need to simulate to the end of the game

Policy and Value Network

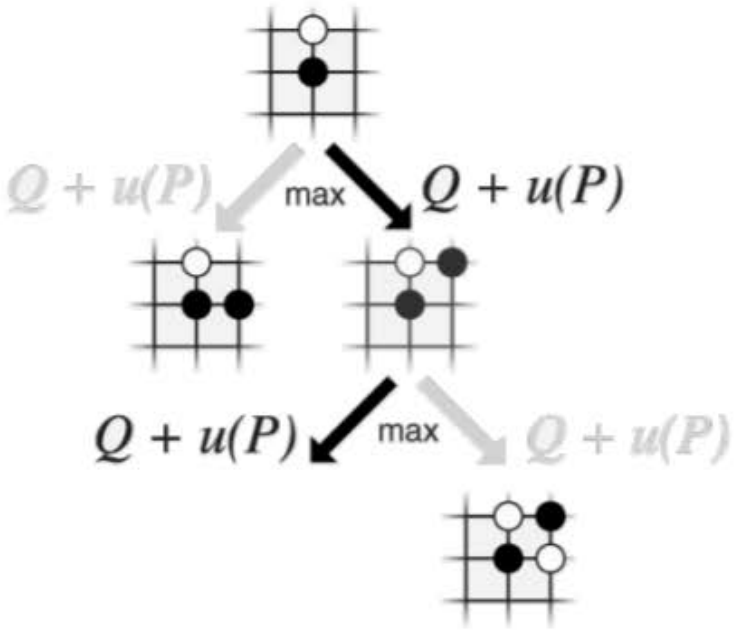
- Reinforced learning



- Training data: 30 million games of self-play
- Training algorithm: minimize mean-squared error by stochastic gradient descent
- Training time: 1 week
- Results: AlphaGo ready for playing against pros

MCTS + Policy / Value Networks

- Selection



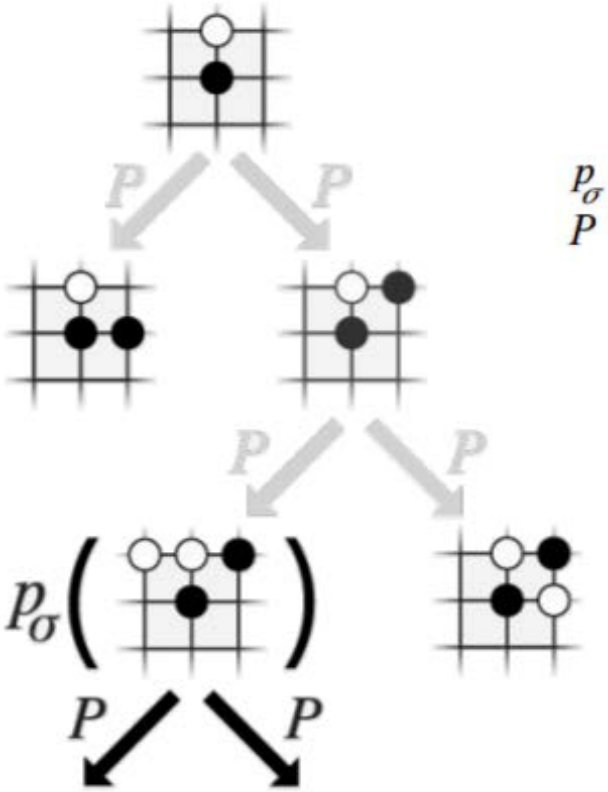
P prior probability
Q action value

Q+u(P)

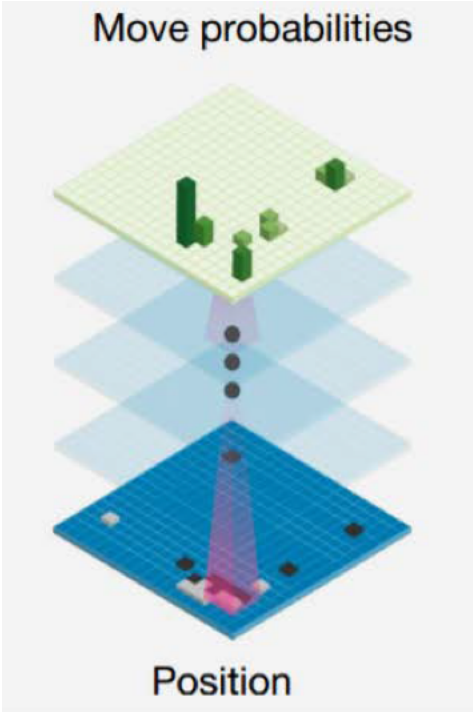
- Initially no simulation yet, so action value = 0, prefers high prior probability and low visits count
- Asymptotically, prefers actions with high action value.

MCTS + Policy / Value Networks

- Expansion

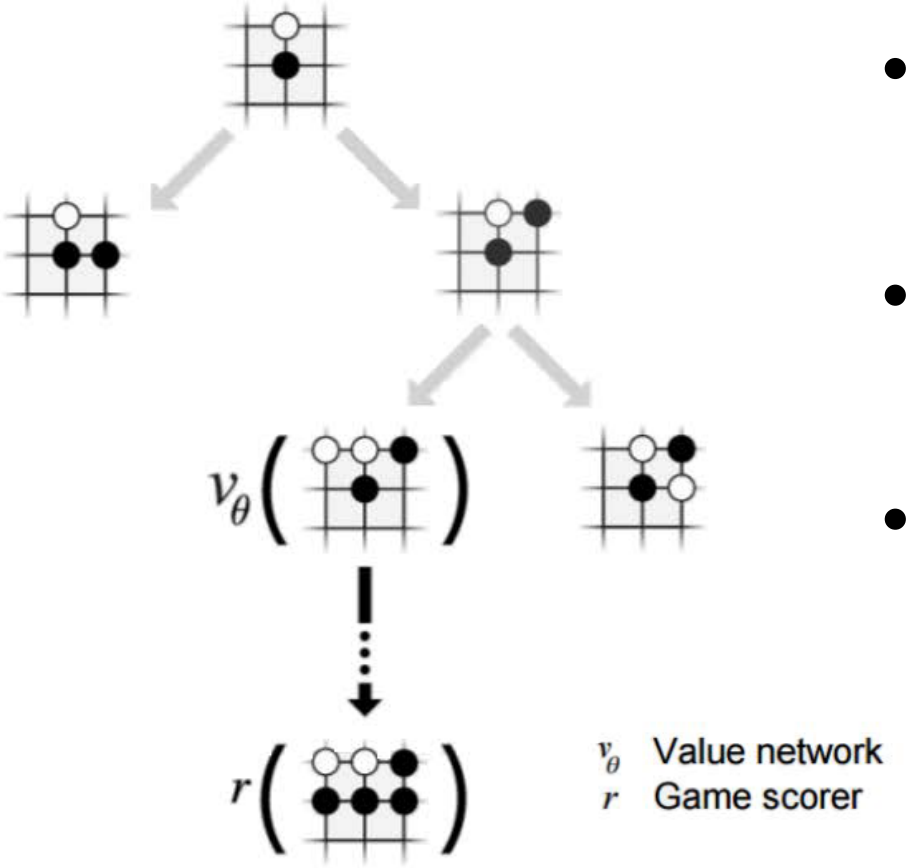


p_σ Policy network
P prior probability



MCTS + Policy / Value Networks

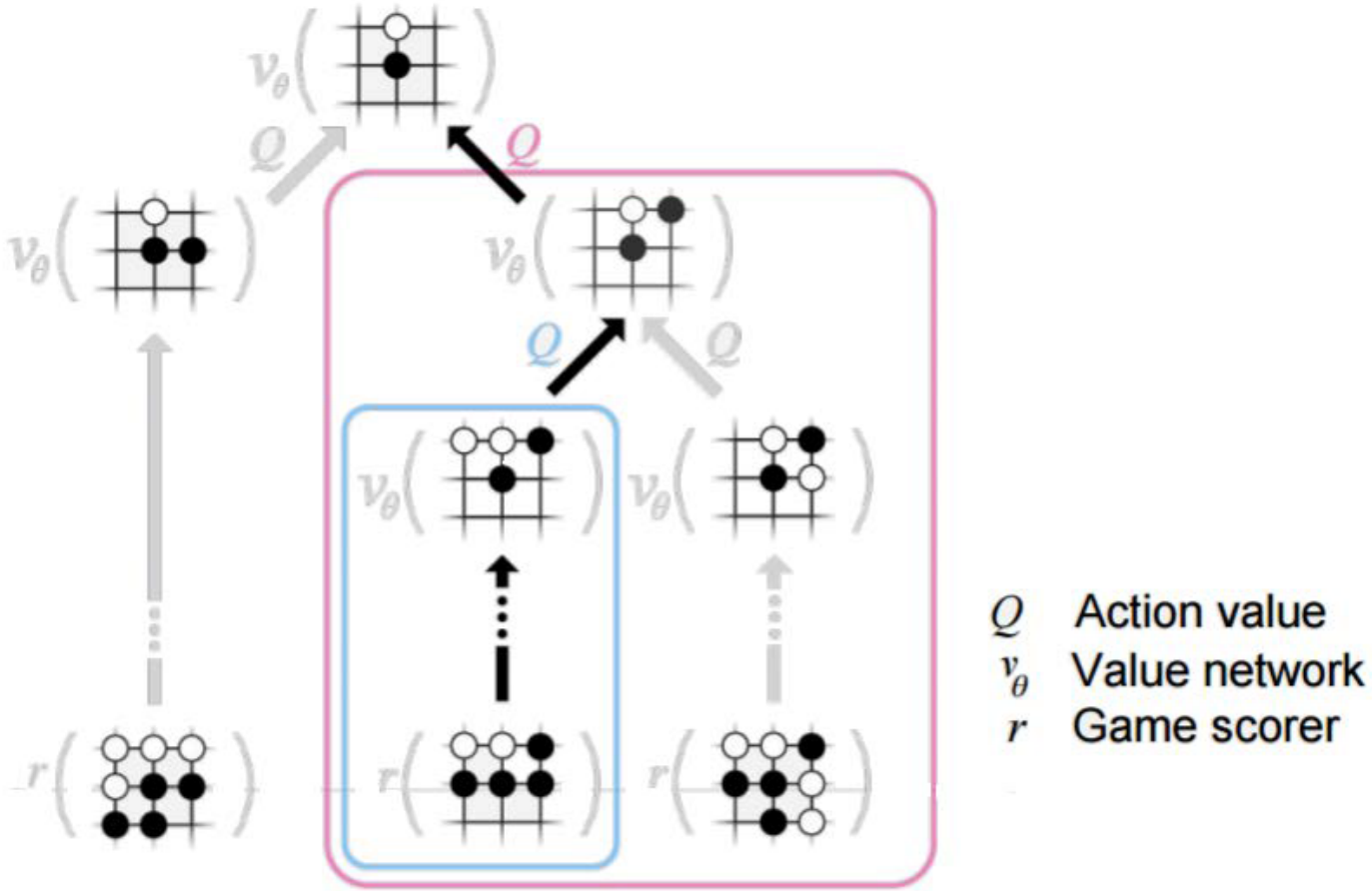
- Simulation



- Run multiple simulations in parallel
- Some with value network
- Some with rollout to the end of the game

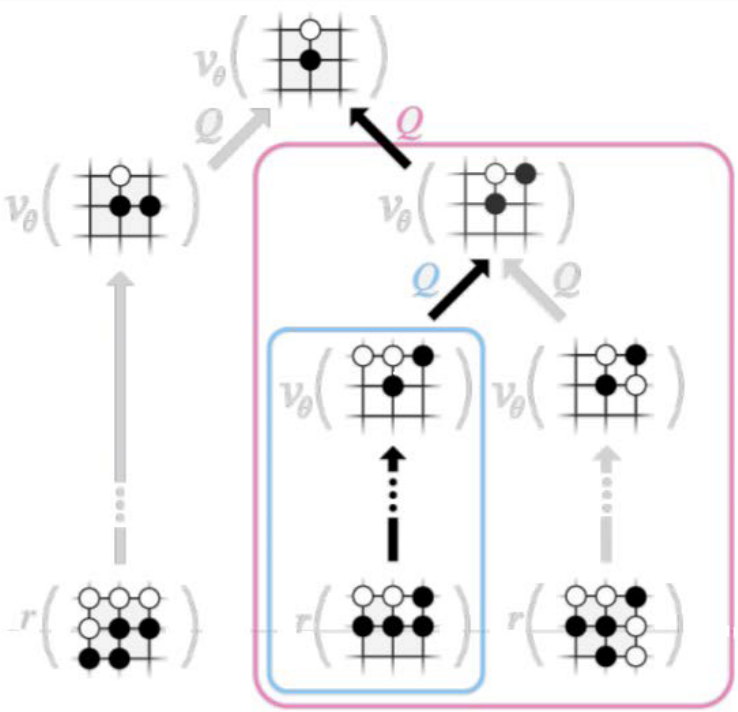
MCTS + Policy / Value Networks

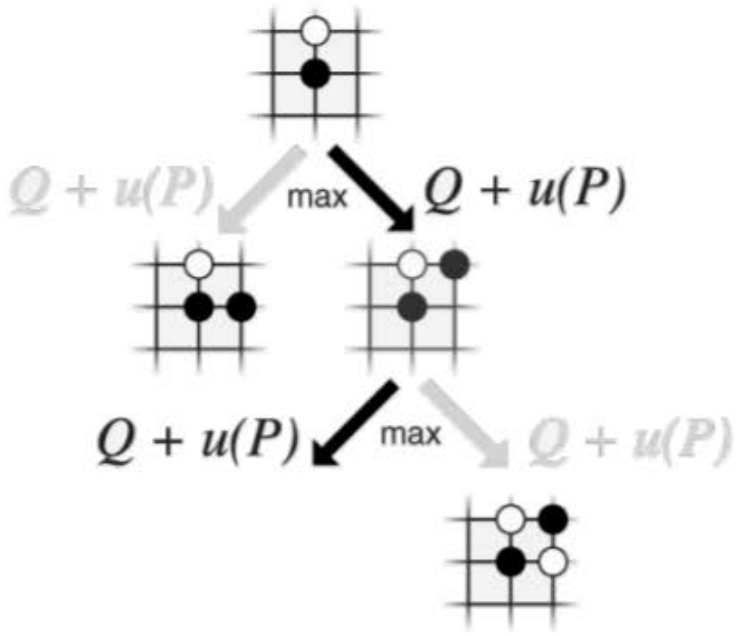
- Propagate values back to root



MCTS + Policy / Value Networks

- Repeat





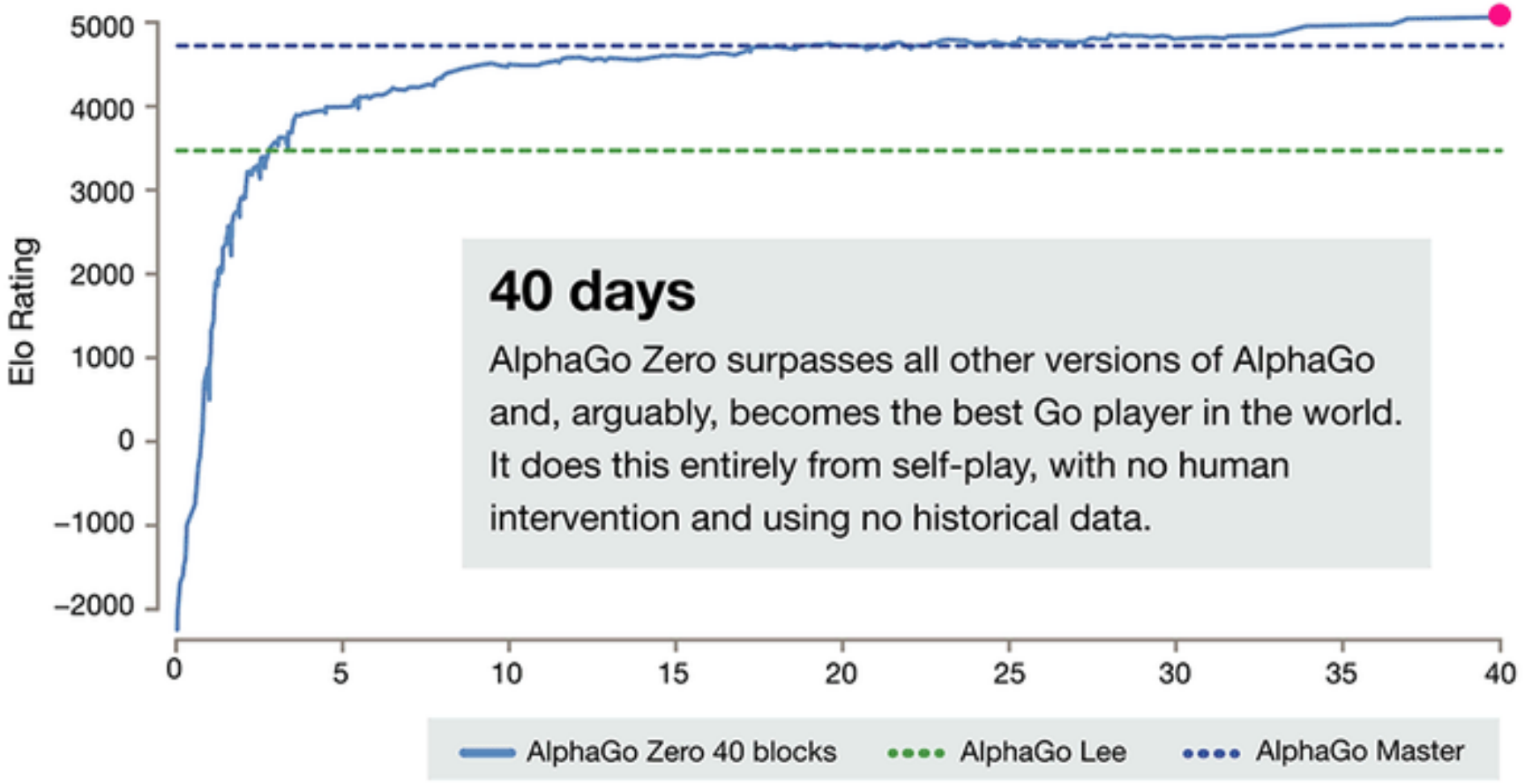
Selection

AlphaGo Zero

- AlphaGo
  - Supervised learning from human expert moves
  - Reinforcement learning from self-play
- AlphaGo Zero
  - Solely reinforcement learning from self-play

AlphaGo Zero

- Beats AlphaGo by 100:0



AlphaGo Takeaway

- Go is still an easy AI problem
  - Fully observable vs. partially observable
  - single agent vs. two agents vs. multiple agents
  - deterministic vs. stochastic
  - discrete vs. continuous
- Combining look-ahead search with learning is very general and is widely applicable

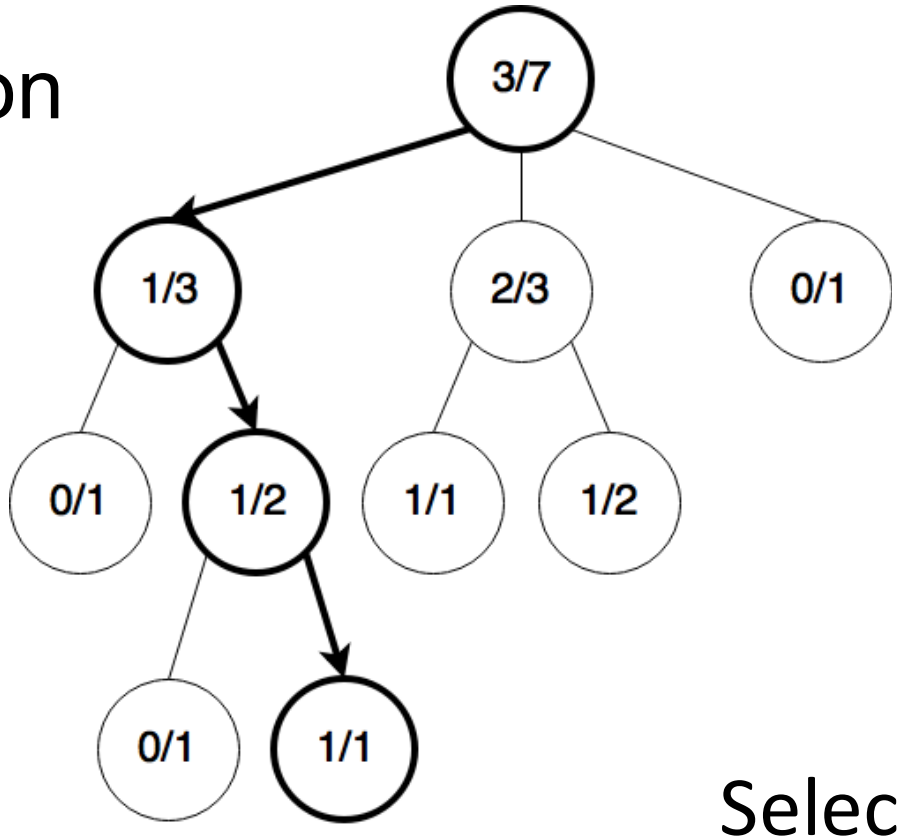
Thanks

References

- All slides are from the following materials.
  - https://icml.cc/2016/tutorials/AlphaGo-tutorial-slides.pdf
  - https://www.coursehero.com/file/28385717/jia-mcts-alphagopdf/

Monte Carlo Tree Search

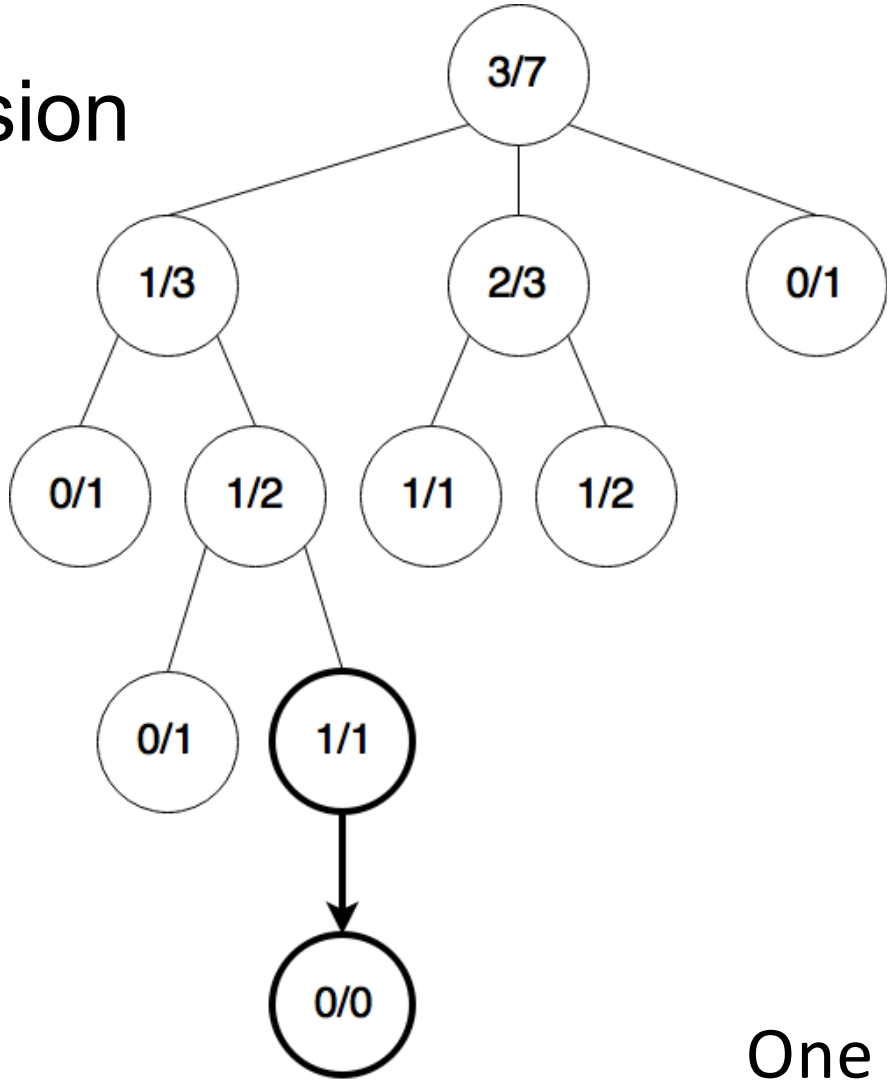
(1) Selection



Selection policy is applied recursively until a leaf node is reached

Monte Carlo Tree Search

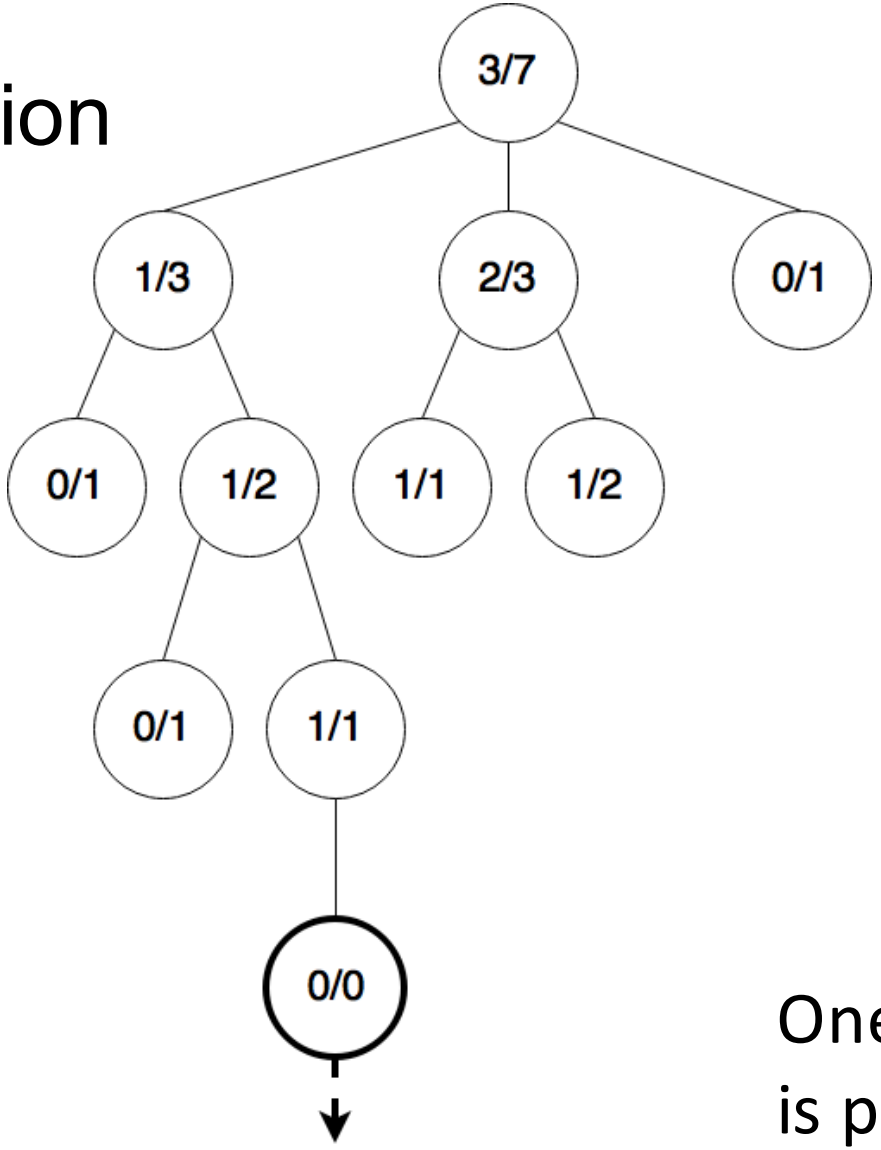
(2) Expansion



One or more nodes are created.

Monte Carlo Tree Search

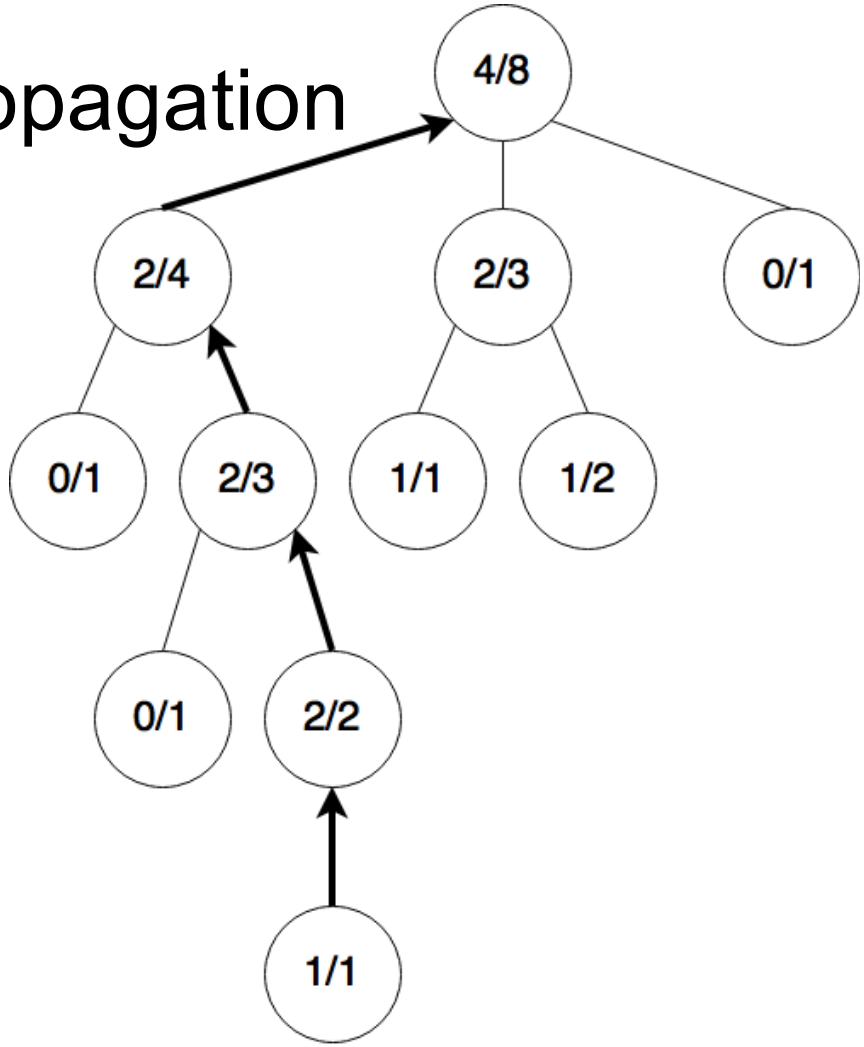
(3) Simulation



One simulated game is played.

Monte Carlo Tree Search

(4) Backpropagation



Multi-Armed Bandit Problem

- Assumptions
  - Choice of several arms
  - Each arm pull is independent of other pulls
  - Each arm has fixed, unknown average payoff
- Which arm has the best average payoff?

Exploration strategy

- Want to explore all arms
  - We don't want to miss any potentially good arm
  - But, if we explore too much, may sacrifice the reward we could have gotten
- Want to exploit promising arms more often
  - Good arms worth further investigation
  - But, if we exploit too much, may get stuck with sub-optimal values