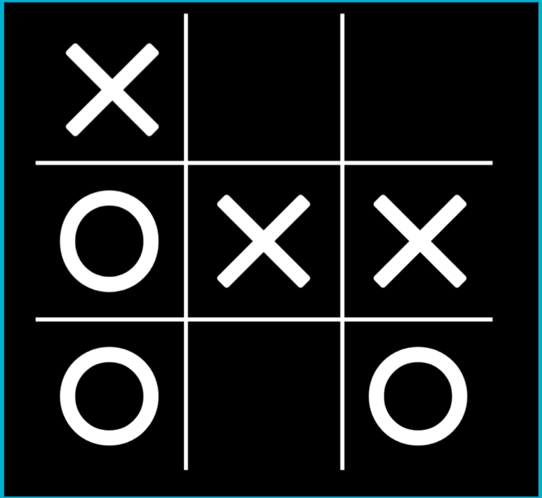
Modern Monte Carlo Tree Search

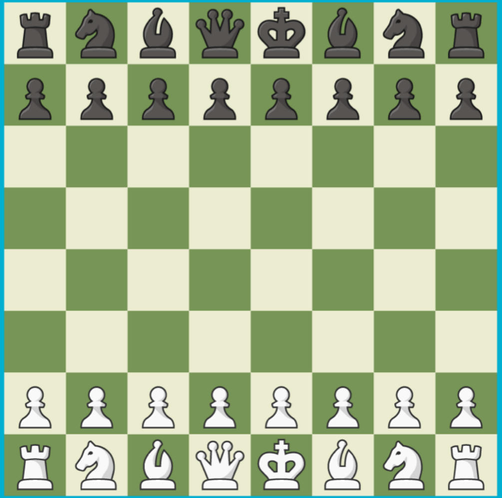
Andrew Li, John Chen, Keiran Paster

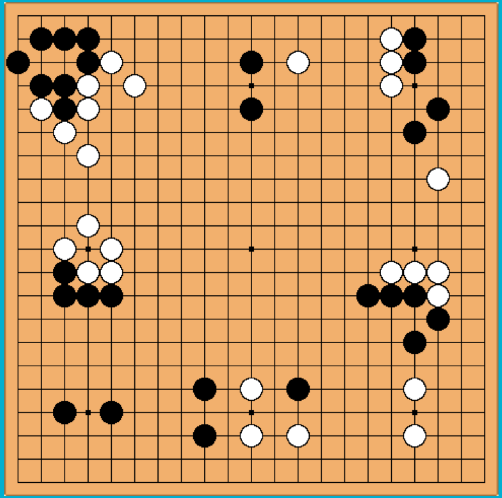
Outline

- Motivation
- Optimistic Exploration and Bandits
- Monte Carlo Tree Search (MCTS)
- Learning to Search in MCTS
  - Thinking Fast and Slow with Deep Learning and Tree Search (Anthony, et al. 2017) **[Expert Iteration]**
  - Mastering the Game of Go without Human Knowledge (Silver, et al. 2017) **[AlphaGo Zero]**
  - Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm (Silver, et al. 2017) **[AlphaZero]**

Motivation



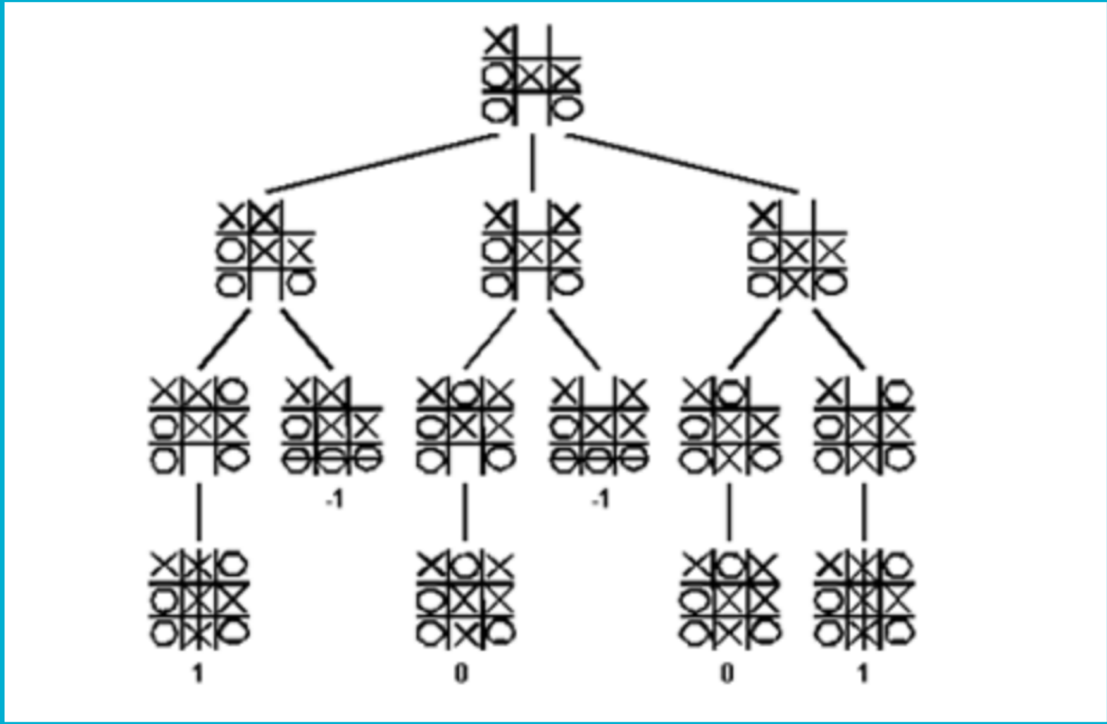




Motivating Problem: Two Player Turn-Based Games

Game Tree Search

- Enumerate all possible moves to minimize your opponent's best possible score (**minimax algorithm**).
- Exact optimal solution can be found with enough resources.
- Useful for finite-length sequential decision-making task where the number of actions is reasonably small.



https://www.cs.cmu.edu/~adamchik/15-121/lectures/Game%20Trees/Game%20Trees.html

Why this doesn't scale

Exponential growth of the game tree!

$$O(b^d)$$

b: branching factor (number of actions)
d: depth

Go: $\sim 10^{170}$ legal positions
Chess: over 10^{40} legal positions

No hope of solving this exactly through brute force!

Ways to speed it up

<u>Action Pruning:</u> Only look at a subset of the available actions from any state.

<u>Depth-Limited Search:</u> Only look at the tree up to a certain depth and use an evaluation function to estimate the value.

Application: Stockfish

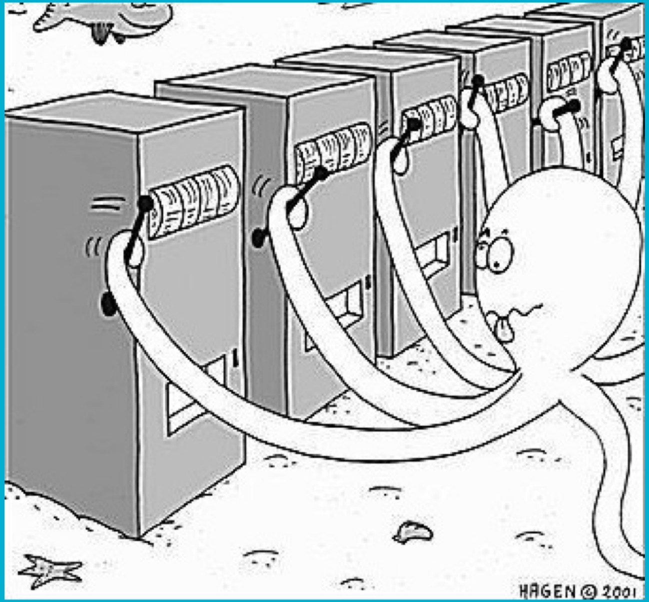
- One of the best chess engines
- Estimates the value of a position using heuristics:
  - Material difference
  - Piece activity
  - Pawn structure
- Uses aggressive action pruning techniques

How to efficiently search without relying on expert knowledge?

- **Exploration:** Learn the values of actions we are uncertain about
- **Exploitation:** Focus the search on the most promising parts of the tree

Multi-Armed Bandits

- *k* slot machines payout according to their own distributions.
- **Goal:** maximize total expected reward earned over time by choosing which arm to pull.
- Need to balance exploration (learning the effects of different actions) vs exploitation (using the best known action).

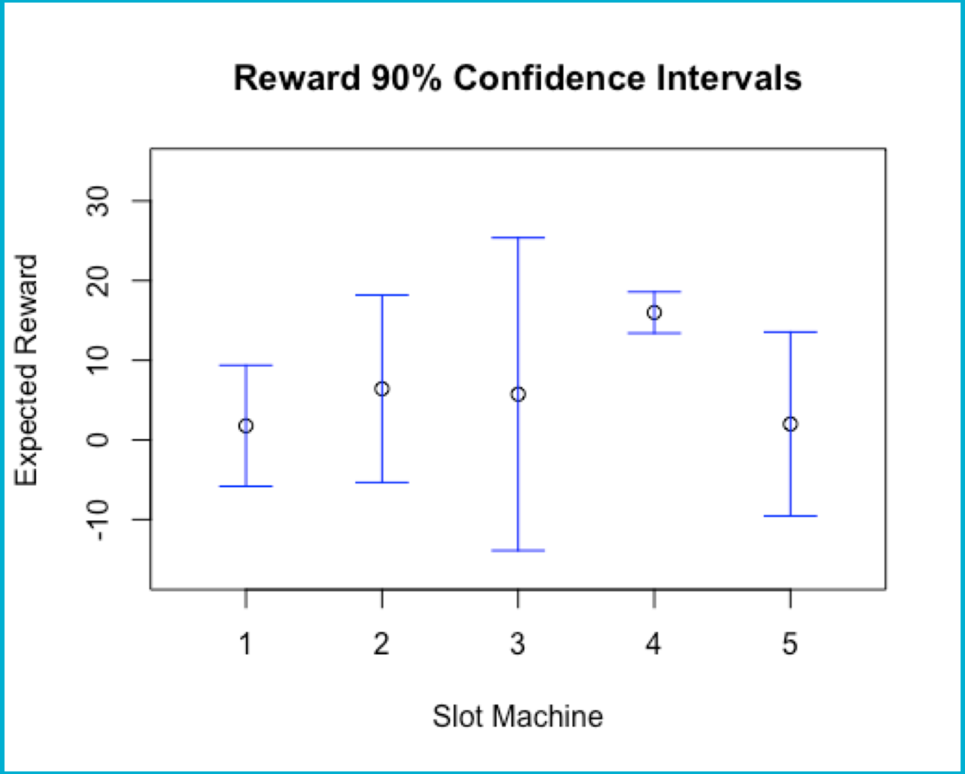


Multi-Armed Bandits Solutions

- **Information State Search:** Exploration provides information which can increase expected reward in future iterations.
- Optimal solution can be found by solving an infinite-state Markov Decision Process over information states. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/XX.pdf
- Computing this solution is often intractable. Heuristics are needed!

Upper Confidence Bound Algorithm

- Record the mean reward for each arm.
- Construct a confidence interval for each expected reward
- Optimistically select the arm with the highest upper confidence bound.
  - Increase the required confidence over time.



Original Image

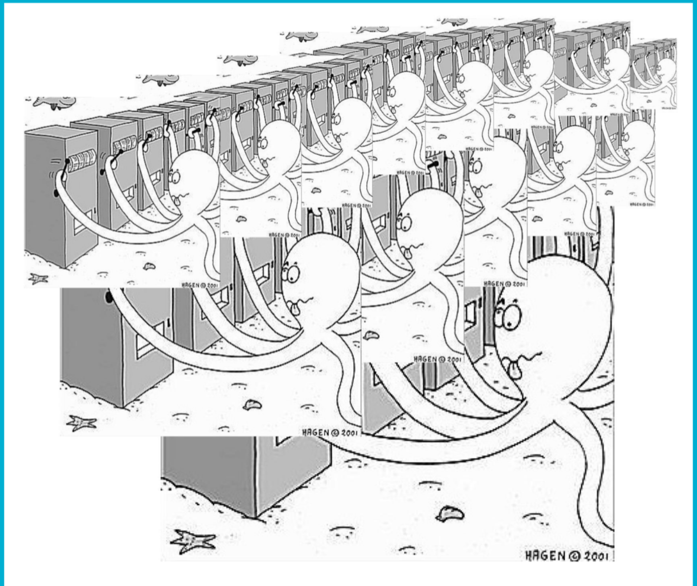
Finite time analysis of the multiarmed bandit problem (P. Auer, et al. 2002)

Monte Carlo Tree Search

Upper Confidence Bounds applied to Trees (UCT)

Bandit Based Monte-Carlo Planning (L. Kocsis and C. Szepesvári)

Treat selecting a node to traverse in our search as a bandit problem.

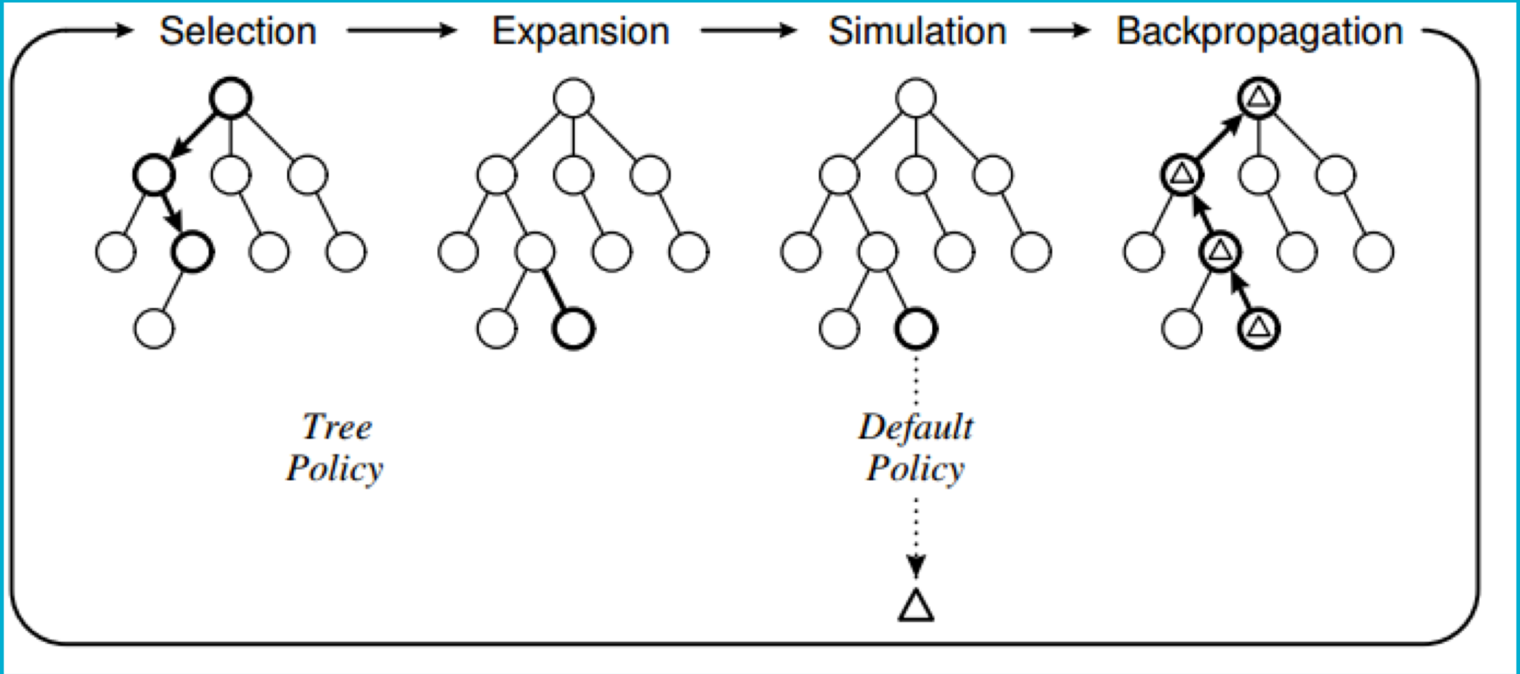


Original Image (adapted)

Monte Carlo Tree Search (MCTS)

- Term coined in 2006 (Couloum et al.) but idea goes back to at least 1987
- Maintain a tree of game states you've seen
- Record the average reward and number of visits to each state
- **Key idea**: instead of a hand-crafted heuristic to estimate the value of a game state, let's just repeatedly **randomly simulate** a game trajectory from that state
    - combined with UCB gives us a good approximation of how good a game state is

An Iteration of MCTS



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

Selection

Tree Policy: choose the child that maximizes the UCB:

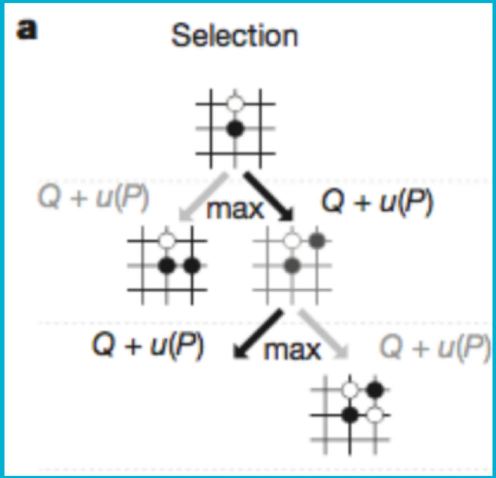
$$\frac{1}{n_i}\sum_{t=1}^{n_i} r_t + c\sqrt{\frac{\ln N}{n_i}}$$

N = number of times the parent node has been visited

n_i = number of times the child has been visited

r_t = reward from t-th visit to the child

c = exploration hyperparameter

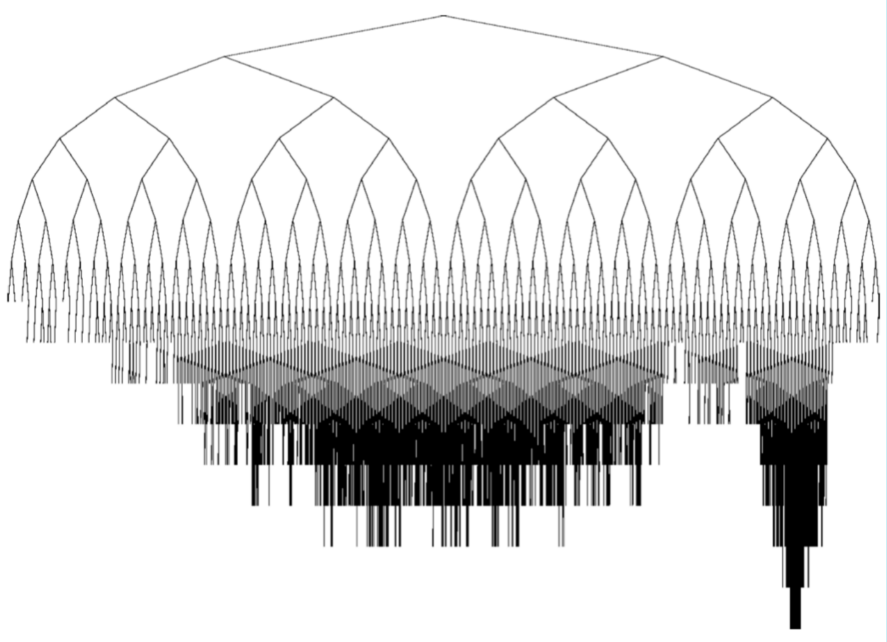


Expansion / Simulation / Backpropagation

What to do when you reach a node without data?

- Always **expand** children nodes that are unvisited by adding it to the tree.
- Estimate the value of the new node by randomly **simulating** until the end of the game (roll-out).
- **Backpropagate** the value to the ancestors of the node. (Unrelated to backpropagation of gradients in neural networks!)

Example: MCTS Tree



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

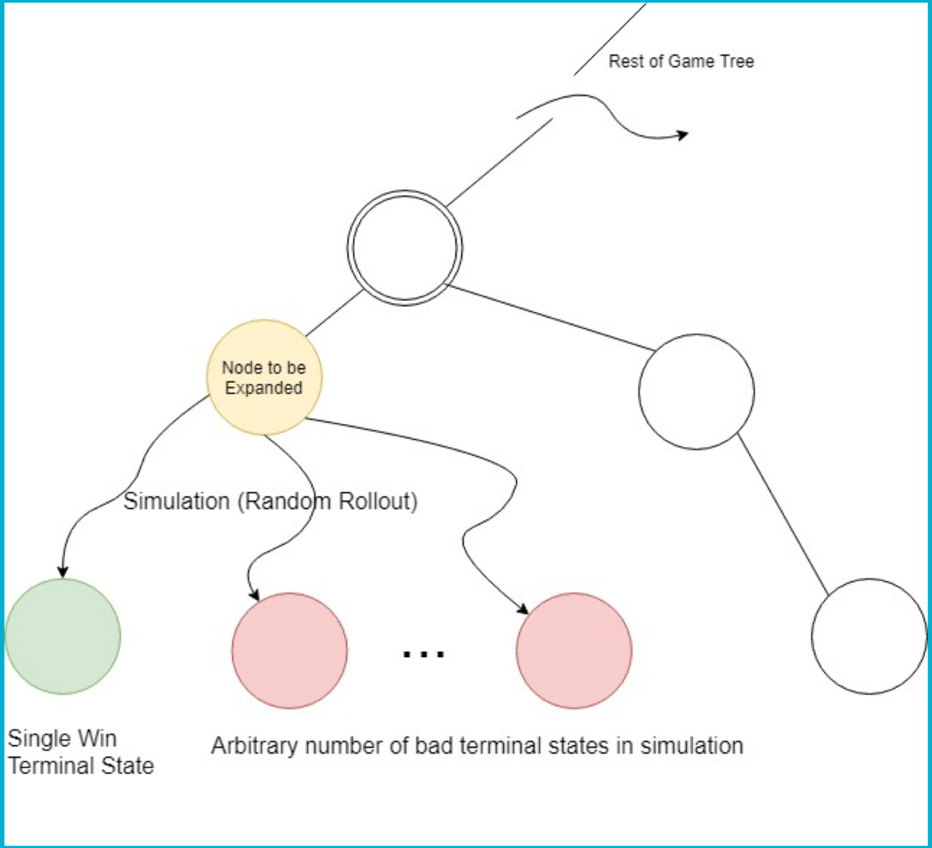
Using MCTS in Practice

- Works well without expert knowledge
- MCTS is anytime: accuracy improves with more computation
- Easy to parallelize
  - Ex. do rollouts for the same node in parallel to get a better estimate

Learning to Search in MCTS

Limitations

- Often a random rollout is not a great estimator for the value of a state
  - **Learn to estimate the value of states**
  - **Learn a smarter policy for rollouts**



Original Content: Mismatch between true value and random Monte Carlo Estimation

Limitations

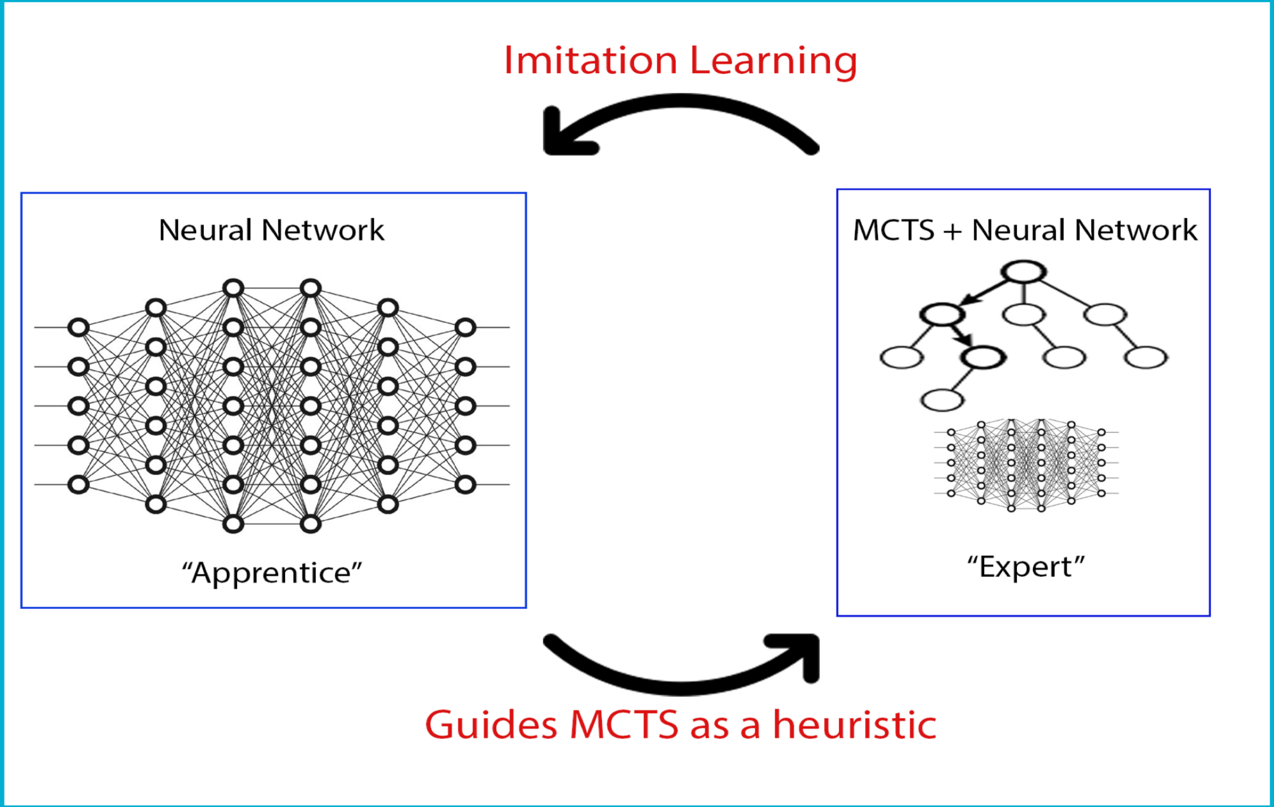
- UCT expands every child of a state before going deeper
  - Learn which states are promising enough to expand
- UCT does not use prior knowledge at test time
  - Remember the results of simulations during training to speed up decision making at test time

Modern Approaches

These three papers (**Expert Iteration, AlphaGo Zero, AlphaZero**) are very related and came out in 2017.

We will point out any important differences!

Expert Iteration, AlphaGo Zero, AlphaZero Main Idea



Original image.

What they learn

- Policy Network - $\pi(a|s)$
  - Probability distribution over the moves
  - Used to focus the search towards good moves
  - Can replace the random policy during rollouts
- Value Network - $V(s)$
  - Predicts the value of any given game state
  - An alternative to rollout simulation in MCTS

- Data is collected from self-play games
- Policy and Value networks are either trained after each iteration (AlphaGo Zero, Expert Iteration) or continuously (AlphaZero)

Learning the Policy Network

- Run MCTS for *n* iterations on a state *s*
- Define the target policy: $\pi_{\text{MCTS}}(a|s) = \dfrac{n(s,a)}{n(s)}$
- Why not train the policy to pick just the optimal (MCTS) action instead?
    - Some states have several good actions.

$$\mathcal{L}_\pi = -\sum_a \pi_{\text{MCTS}}(a|s) \log[\pi(a|s)]$$

Learning the Value Network

- Gather state / value pairs either by rolling out directly with the policy network (ExIt) or via MCTS rollouts (AlphaZero).
- Treat the target value as the probability of winning
    - Cross entropy loss (ExIt)
- Or as some arbitrary reward (win = +1, tie = 0, loss = -1)
    - Squared error loss (AlphaGo Zero, AlphaZero)

Improving MCTS with the Learned Policy

UCB: $$\frac{1}{n_i}\sum_{t=1}^{n_i} r_t + c_{\mathrm{UCT}}\sqrt{\frac{\ln N}{n_i}}$$

ExIt: $$\frac{1}{n_i}\sum_{t=1}^{n_i} r_t + c_{\mathrm{UCT}}\sqrt{\frac{\ln N}{n_i}} + c_\pi \frac{\pi(a_i|s)}{n_i+1}$$

(a bonus for exploration and for choosing likely optimal actions)

Note: in ExIt unexplored actions are always taken.

Improving MCTS with the Learned Policy

UCB: $$\frac{1}{n_i}\sum_{t=1}^{n_i} r_t + c\sqrt{\frac{\ln N}{n_i}}$$

AlphaZero: $$\frac{1}{n_i}\sum_{t=1}^{n_i} r_t + c\pi(a_i|s)\frac{\sqrt{N}}{n_i+1}$$

(Mask out bad states from exploration)

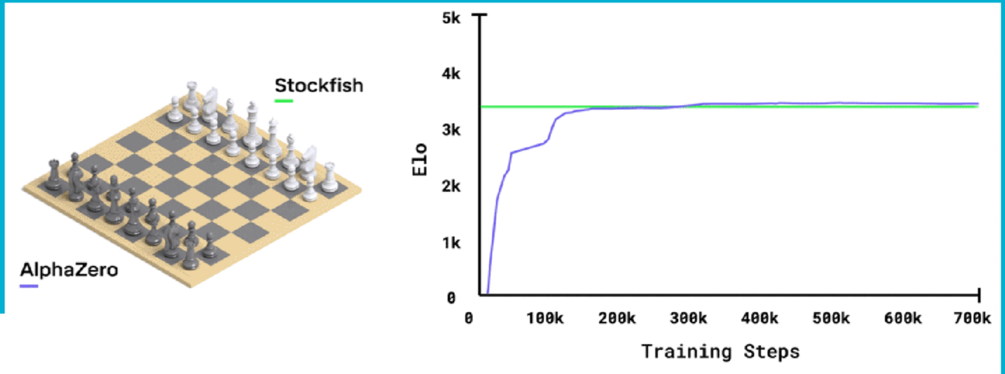
Improving MCTS with the Learned Value

- Evaluate positions with the value network instead of rollouts.

- Some variants (ExIt, AlphaGo) use a combination of a rollout (using the policy network) and the value network.
    - Rollouts are usually more expensive than value network computations.

Performance

https://www.theverge.com/2017/5/27/15704088/alphago-ke-jie-game-3-result-retires-future



https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

Related Work

- AlphaGo Fan
  - Train a neural network to imitate professional moves
  - Use REINFORCE during self play to improve the policies
  - Train a value network to predict the winner of these self play games
  - At test time, combine these networks with MCTS
- AlphaGo Lee
  - Train the value network with the AlphaGo MCTS + NN games rather than just the NN games
  - Iterate several times
- AlphaGo Master
  - Uses the AlphaGo Zero algorithm but is pre trained to imitate a professional.

Limitations/Future Work

- AlphaGo Zero and AlphaZero required an *ungodly* amount of computation for training (over 5000 TPUs, $25 million in hardware for AlphaGo Zero)
- Requires a fast simulator / true model of the environment.
- Doesn't apply to (multiplayer) games with simultaneous moves / imperfect information
- Heuristic is restricted to a specific class of functions: those structured like UCT
    - MCTS-nets: use a neural net to learn an *arbitrary* function (neural nets are universal function approximators)

Thanks for listening!

https://en.chessbase.com/post/the-future-is-here-alphazero-learns-chess