# Lecture 14: Monte Carlo Tree Search

Emma Brunskill

CS234 Reinforcement Learning.

Spring 2024

- With many slides from or derived from David Silver

# Refresh Your Understanding

Select all that are true:

- Upper confidence bounds are used to balance exploration and leveraging the acquired information to achieve high reward
- These algorithms can be used in bandits and Markov decision processes
- If the reward model is known, there is no benefit to using an upper confidence bound algorithm

# Refresh Your Understanding

Select all that are true:

- Upper confidence bounds are used to balance exploration and leveraging the acquired information to achieve high reward
- These algorithms can be used in bandits and Markov decision processes
- If the reward model is known, there is no benefit to using an upper confidence bound algorithm

# Class Structure

- Last time: Fast / sample efficient Reinforcement Learning
- **This Time: MCTS**
- Next time: Rewards in RL

# AlphaZero and Monte Carlo Tree Search

- Responsible in part for one of the greatest achievements in AI in the last decade– becoming a better Go player than any human
- Incorporates a number of interesting ideas

# Table of Contents

# Computing Action for Current State Only

- So far in class, compute a policy for whole state space
- Key idea: can prioritize some additional local computation to make a better decision for right now

# Simple Monte-Carlo Search

- Given a model $\mathcal{M}_v$ and a simulation policy $\pi$
- For each action $a \in \mathcal{A}$
  - Simulate $K$ episodes from current (real) state $s_t$

$$\{s_t, a, R_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_v, \pi$$

  - Evaluate actions by mean return (Monte-Carlo evaluation)

$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} q_\pi(s_t, a) \tag{1}$$

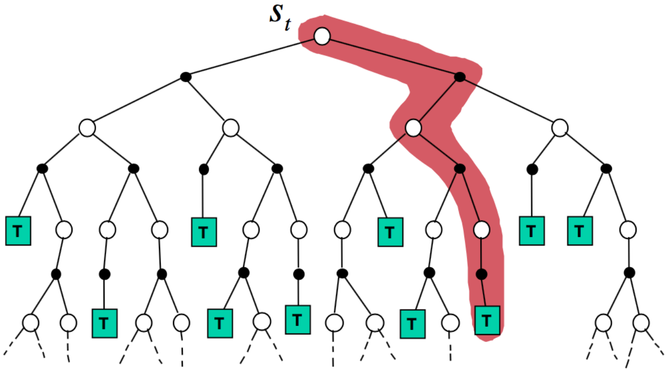
- Select current (real) action with maximum value

$$a_t = \underset{a \in A}{\operatorname{argmax}}\, Q(s_t, a)$$

- This is essentially doing 1 step of policy improvement

# Simulation-Based Search

- Simulate episodes of experience from now with the model
- Apply model-free RL to simulated episodes

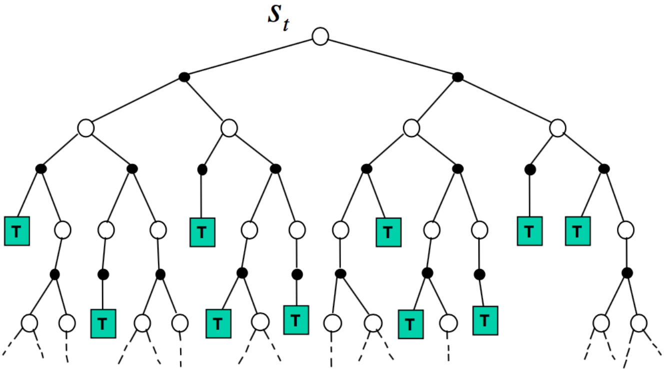


# Expectimax Tree

- Can we do better than 1 step of policy improvement?
- If have a MDP model $\mathcal{M}_v$
- Can compute optimal $Q(s, a)$ values for current state by constructing an **expectimax** tree

# Forward Search Expectimax Tree

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state st at the root
- Using a model of the MDP to look ahead




- No need to solve whole MDP, just sub-MDP starting from now

# Expectimax Tree

- Can we do better than 1 step of policy improvement?
- If have a MDP model $\mathcal{M}_v$
- Can compute optimal $q(s, a)$ values for current state by constructing an expectimax tree
- Limitations: Size of tree scales as $(|S||A|)^H$

# Monte-Carlo **Tree** Search (MCTS)

- Given a model $\mathcal{M}_v$
- Build a **search tree** rooted at the current state $s_t$
- Samples actions and next states
- Iteratively construct and update tree by performing $K$ simulation episodes starting from the root state
- After search is finished, select current (real) action with maximum value in search tree

$$a_t = \arg\max_{a \in A} Q(s_t, a)$$

- Check your understanding: How does this differ from Monte Carlo Simulated Search?

# Check Your Understanding: MCTS

- MCTS involves deciding on an action to take by doing tree search where it picks actions to maximize $Q(S, A)$ and samples states. Select all
  1. Given a MDP, MCTS may be a good choice for short horizon problems with a small number of states and actions.
  2. Given a MDP, MCTS may be a good choice for long horizon problems with a large action space and a small state space
  3. Given a MDP, MCTS may be a good choice for long horizon problems with a large state space and small action space
  4. Not sure

# Upper Confidence Tree (UCT) Search

- How to select what action to take during a simulated episode?

# Upper Confidence Tree (UCT) Search

- How to select what action to take during a simulated episode?
- UCT: borrow idea from bandit literature and treat each node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm

# Upper Confidence Tree (UCT) Search

- How to select what action to take during a simulated episode?
- UCT: borrow idea from bandit literature and treat each **node** where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm at a node

$$Q(s, a, i) = \frac{1}{N(i,a)} \sum_{k=1}^{N(i,a)} G_k(i, a) + c\sqrt{\frac{O(\log N(i))}{N(i,a)}}$$

- where $N(i, a)$ is the number of times selected arm $a$ at node $i$, $G_k(i, a)$ is the $k$-th return (discounted sum of rewards) from node $i$ following action $a$, and
- For simulated episode $k$ at node $i$, select action/arm with highest upper bound to simulate and expand (or evaluate) in the tree

$$a_{ik} = \arg\max Q(s, a, i)$$

- This implies that the policy used to simulate episodes with (and expand/update the tree) can change across each episode

# Advantages of MC Tree Search

- Highly selective best-first search
- Evaluates states dynamically (unlike e.g. DP)
- Uses sampling to break curse of dimensionality
- Works for "black-box" models (only requires samples)
- Computationally efficient, anytime, parallelisable

# Table of Contents

# AlphaGo

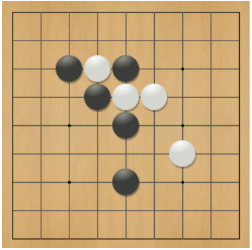
AlphaGo trailer link

# Case Study: the Game of Go

- Go is 2500 years old
- Hardest classic board game
- Grand challenge task (John McCarthy)
- Traditional game-tree search has failed in Go
- Check your understanding: does playing Go involve learning to make decisions in a world where dynamics and reward model are unknown?

# Rules of Go

- Usually played on 19x19, also 13x13 or 9x9 board
- Simple rules, complex strategy
- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game



# AlphaGo and AlphaZero

- Self Play
- Strategic Computation
- Highly selective best-first search
- Power of Averaging
- Local Computation
- Learn and Update Heuristics

# Self Play for Go

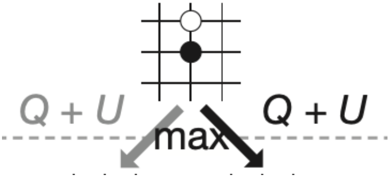
- Key idea: have agent play itself
- Game operates by computing best move at current state, then, for opponent move, doing the same
- Bottleneck is only computation, no humans needed
- Self-play also provides a well-matched player
- Check your understanding: how does this help with policy training? What is the reward density?

# Self Play for Go: Solution

- Key idea: have agent play itself
- Game operates by computing best move at current state, then, for opponent move, doing the same
- Bottleneck is only computation, no humans needed
- Self-play also provides a well-matched player
- Check your understanding: how does this help with policy training? What is the reward density?

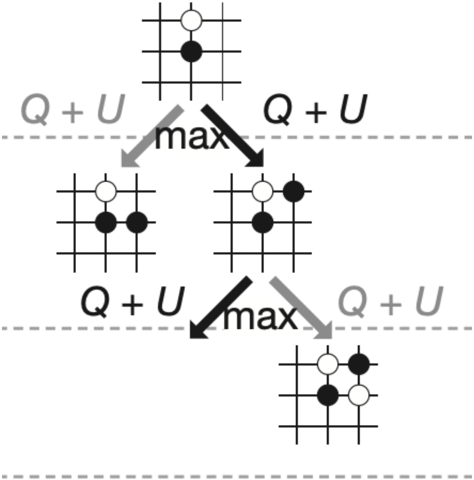
# Selecting a Move in a Single Game: Start at Root[1]

- Inspired by Upper Confidence Tree Search but many changes

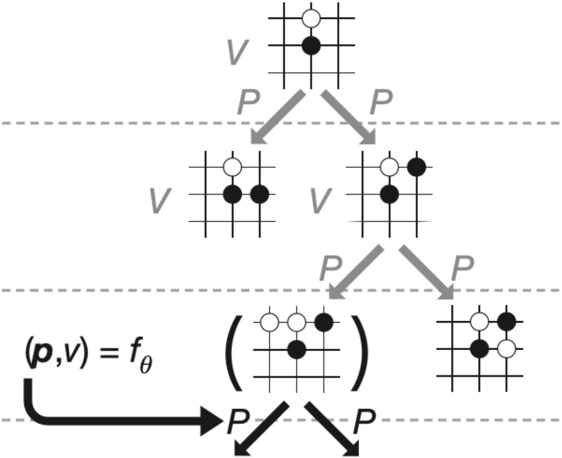



[1] Images from Silver et al. Nature 2017

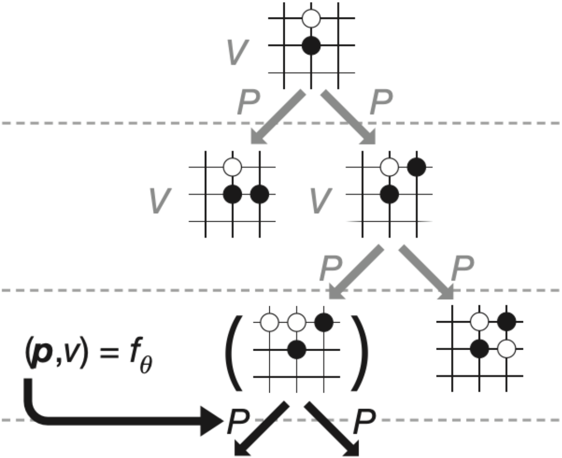
# Selecting a Move in a Single Game: Repeatedly Expand[2]



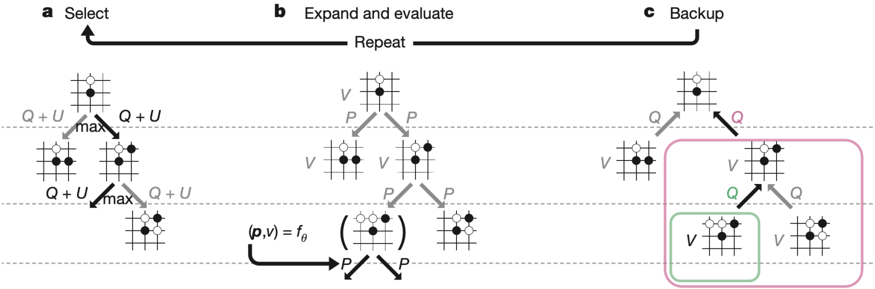
# Selecting a Move in a Single Game: Note Using Network Predictions for Action Probabilities[3]



# Selecting a Move in a Single Game: At Leaf, Plug in Network Predictions for Value[4]

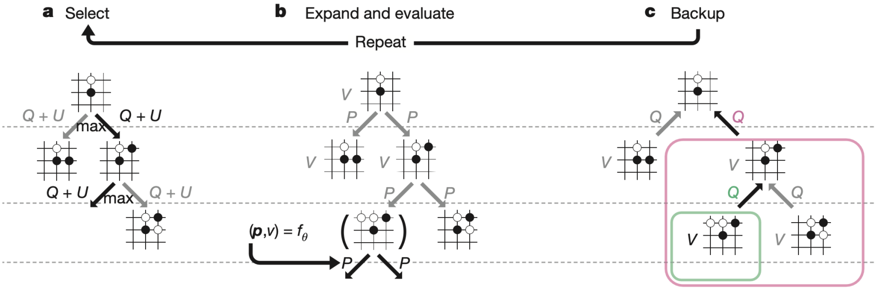


# Selecting a Move in a Single Game: Update Ancestors[5]




[5]Images from Silver et al. Nature 2017

# Selecting a Move in a Single Game: Repeat Many Times[6]

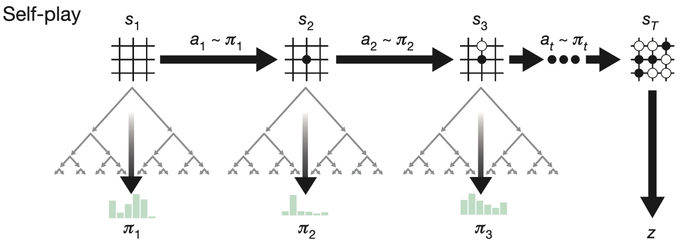



- Repeat roll out and backup process many times
- Note: inside the network alternating whether opponent or agent is "maximizing" its value. Therefore tree is mimicking a min-max tree
- At end, compute a policy for root node by

$$\pi(s) \propto N(s,a)^{\frac{1}{\tau}} \tag{2}$$

[6]Images from Silver et al. Nature 2017

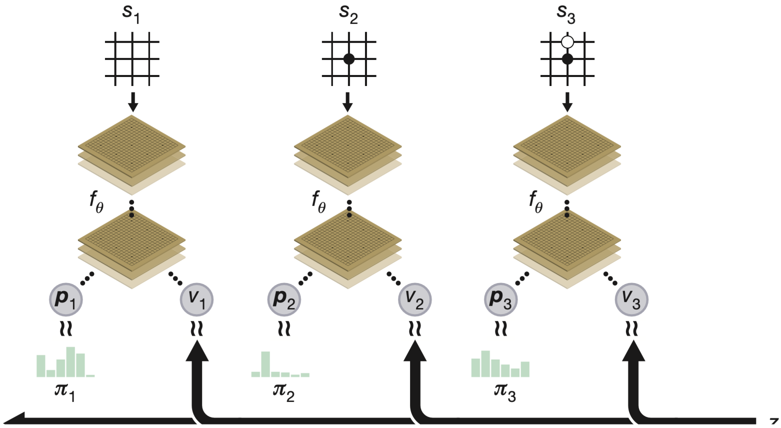
# Self Play a Game[7]




- Select an action according to root policy, take action, and repeat whole process
- Repeat until game ends* and observe a win or loss

[7] Images from Silver et al. Nature 2017

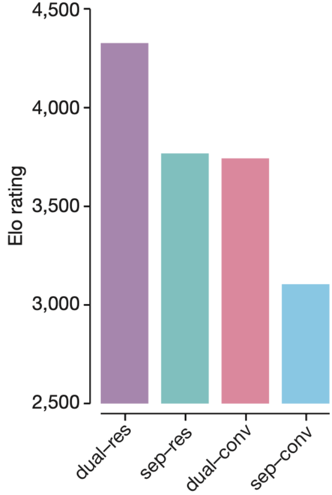
# Train Neural Network to Predict Policies and Values footnoteImages from Silver et al. Nature 2017



# AlphaGo and AlphaZero: Recap and Evaluation

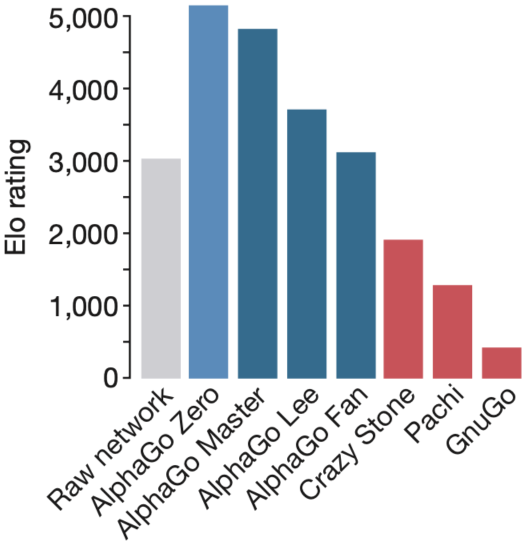
- Features:
  - Self Play
  - Strategic Computation
  - Highly selective best-first search
  - Power of Averaging
  - Local Computation
  - Learn and Update Heuristics
- Evaluation Questions
  - What is the influence of architecture?
  - What is the impact of using MCTS (on top of learning a policy / value function)?
  - How does it compare to human play or using human play?

# Impact of Architecture[8]



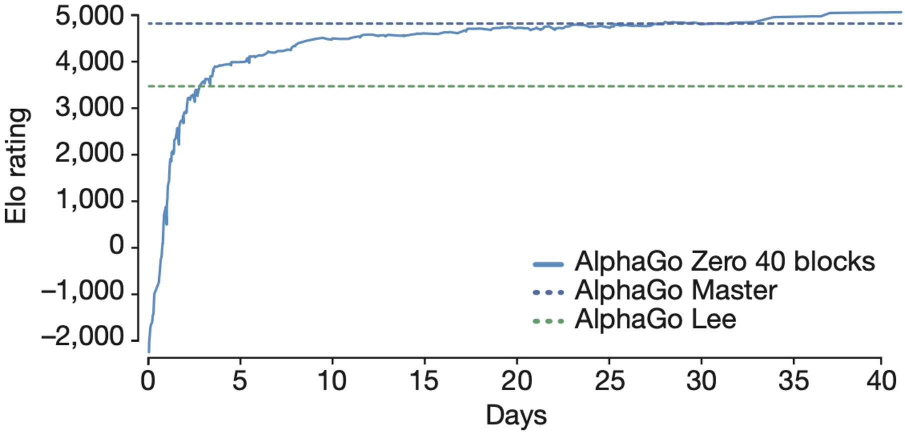

[8] Images from Silver et al. Nature 2017

# Impact of MCTS[9]



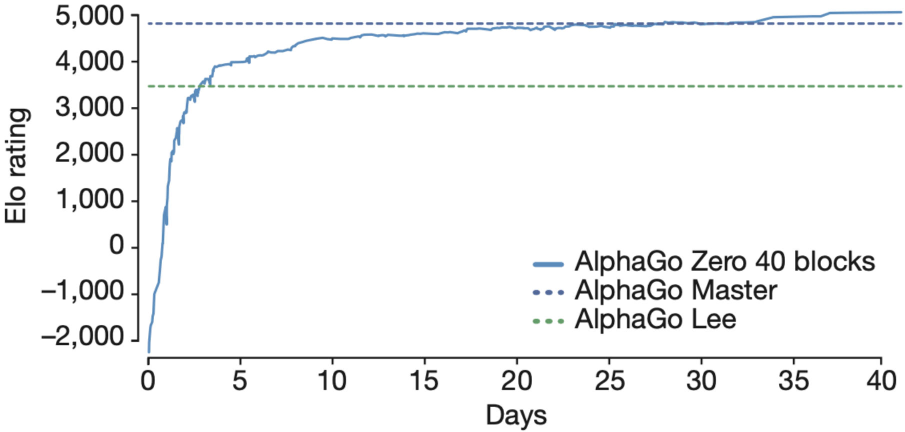

[9] Images from Silver et al. Nature 2017

# Overall performance[10]




[10] Images from Silver et al. Nature 2017

# Need for Human Data?[11]




[11] Images from Silver et al. Nature 2017

# In more depth: Upper Confidence Tree (UCT) Search

- UCT: borrow idea from bandit literature and treat each tree node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm and select the best arm
- Check your understanding: Why is this slightly strange? Hint: why were upper confidence bounds a good idea for exploration/ exploitation? Is there an exploration/ exploitation problem during simulated episodes?[12]

[12] Relates to metalevel reasoning (for an example related to Go see "Selecting Computations: Theory and Applications", Hay, Russell, Tolpin and Shimony 2012)

# Check Your Understanding: UCT Search

- In Upper Confidence Tree (UCT) search we treat each tree node as a multi-armed bandit (MAB) problem, and use an upper confidence bound over the future value of each action to help select actions for later rollouts. Select all that are true
  1. This may be useful since it will prioritize actions that lead to later good rewards
  2. UCB minimizes regret. UCT is minimizing regret within rollouts of the tree. (If this is true, think about if this a good idea?)
  3. Not sure

# In more depth: Upper Confidence Tree (UCT) Search

- UCT: borrow idea from bandit literature and treat each tree node where can select actions as a multi-armed bandit (MAB) problem
- Maintain an upper confidence bound over reward of each arm and select the best arm
- Hint: why were upper confidence bounds a good idea for exploration/ exploitation? Is there an exploration/ exploitation problem during simulated episodes?[13]

[13] Relates to metalevel reasoning (for an example related to Go see "Selecting Computations: Theory and Applications", Hay, Russell, Tolpin and Shimony 2012)

# Class Structure

- Last time: Fast Learning
- **This Time: MCTS**
- Next time: Rewards in RL