Monte Carlo Tree Search and Go

Cathy Wu

6.7920 Reinforcement Learning: Foundations and Methods

Readings

1. *Sutton & Barto (SB) §16.6*

Outline

1. Online planning
2. Monte Carlo Tree Search (MCTS)
3. AlphaGo: Learning-guided MCTS

Outline

1. **Online planning**
   a. The game of Go
   b. Online planning vs offline planning
   c. Exhaustive search
   d. Lookahead + rollout as policy iteration
   e. Reducing depth with offline value function
   f. TD-Gammon
2. Monte Carlo Tree Search (MCTS)
3. AlphaGo: Learning-guided MCTS

Why the fascination in AI for the game of Go?

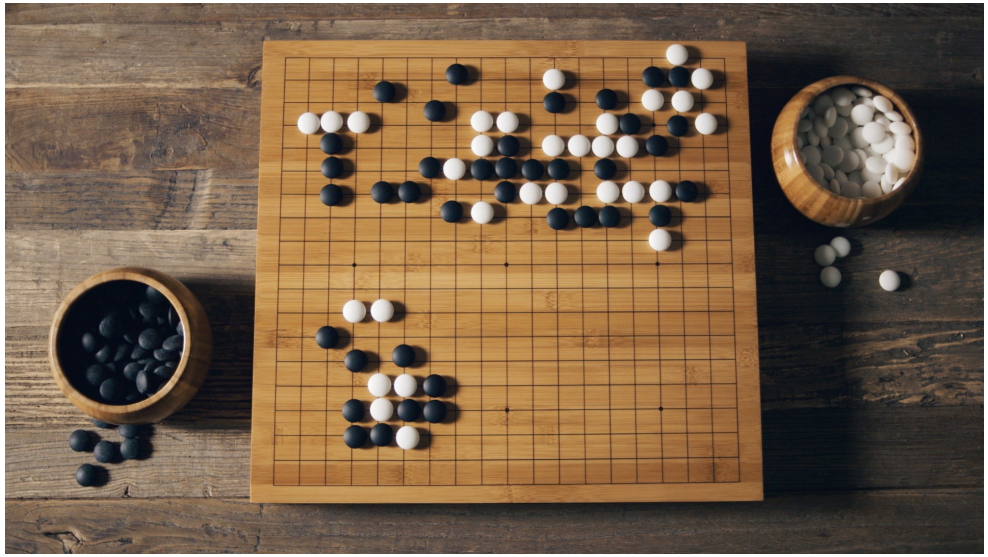
Brute force search intractable:

- **Search space is huge**

Game tree complexity = b^d

- b = Branching factor
- d = Depth

3000 year old game



19 x 19 Board 361 Actions

Average length of game: 211 moves

Why the fascination in AI for the game of Go?

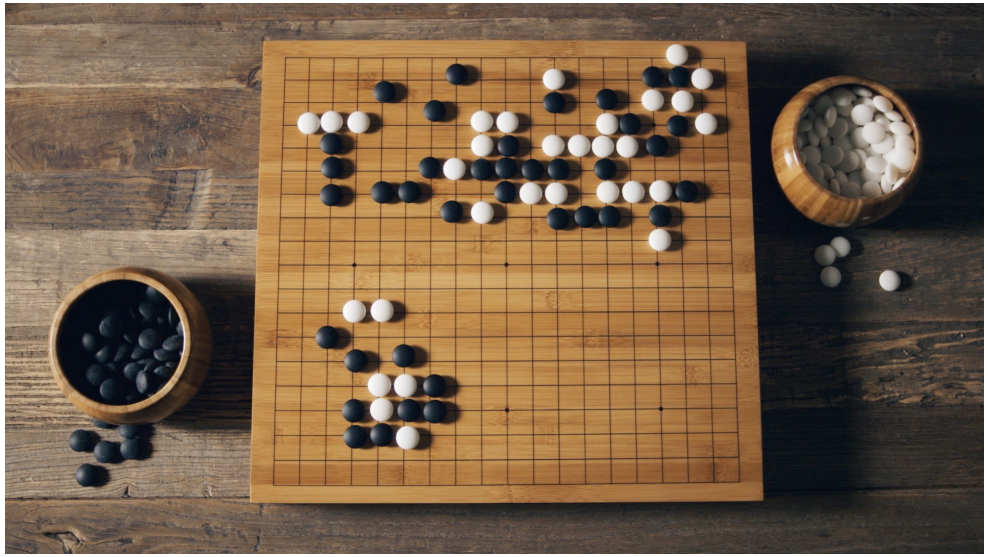
Brute force search intractable:

- Search space is huge
- **Impossible to evaluate who is winning (position evaluation)**

Approaches taken:

- Designed by hand / experts
- Supervised learning
- Search
- Reinforcement learning
- AlphaGo: all of the above

3000 year old game



10^{170} board positions

10^{80} atoms in universe

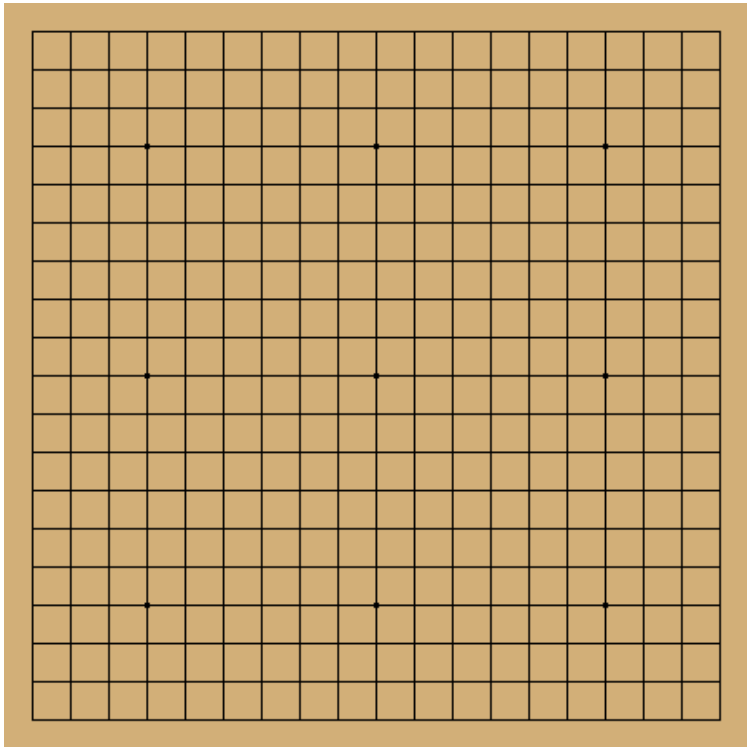
AlphaGo

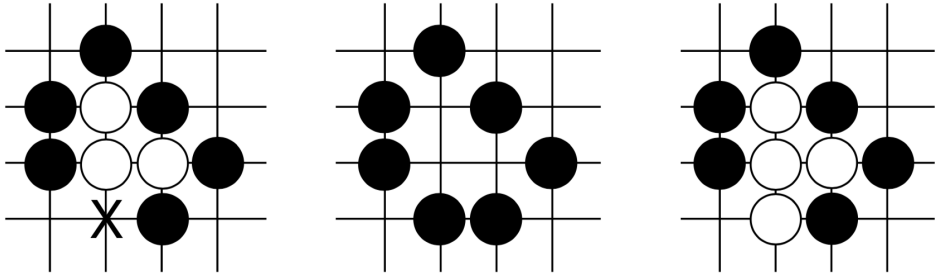
First computer program to defeat a world champion (2016)

AlphaGo vs Lee Sedol

- Lee Sedol (9p): winner of 18 world titles
- Match was played in Seoul, March 2016
- AlphaGo won the match 4-1

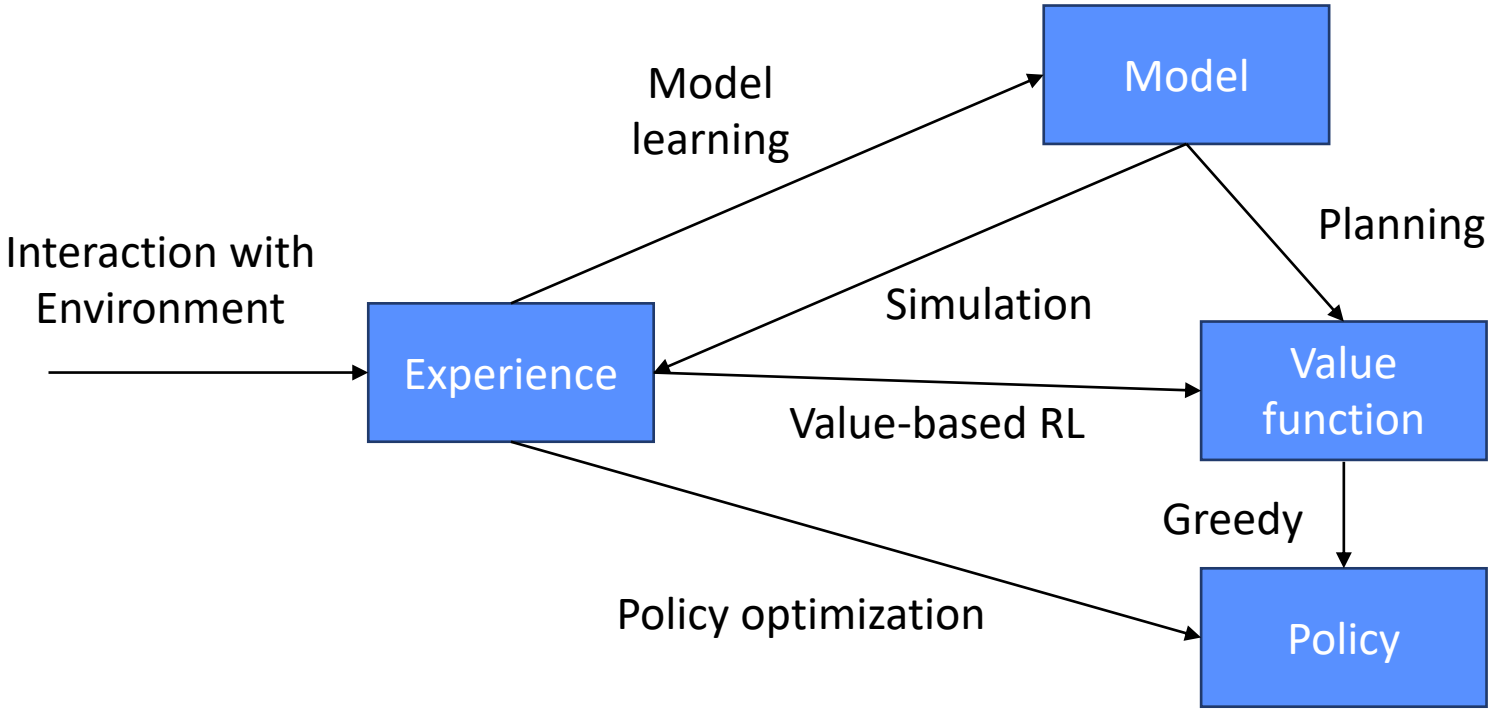
The game of Go





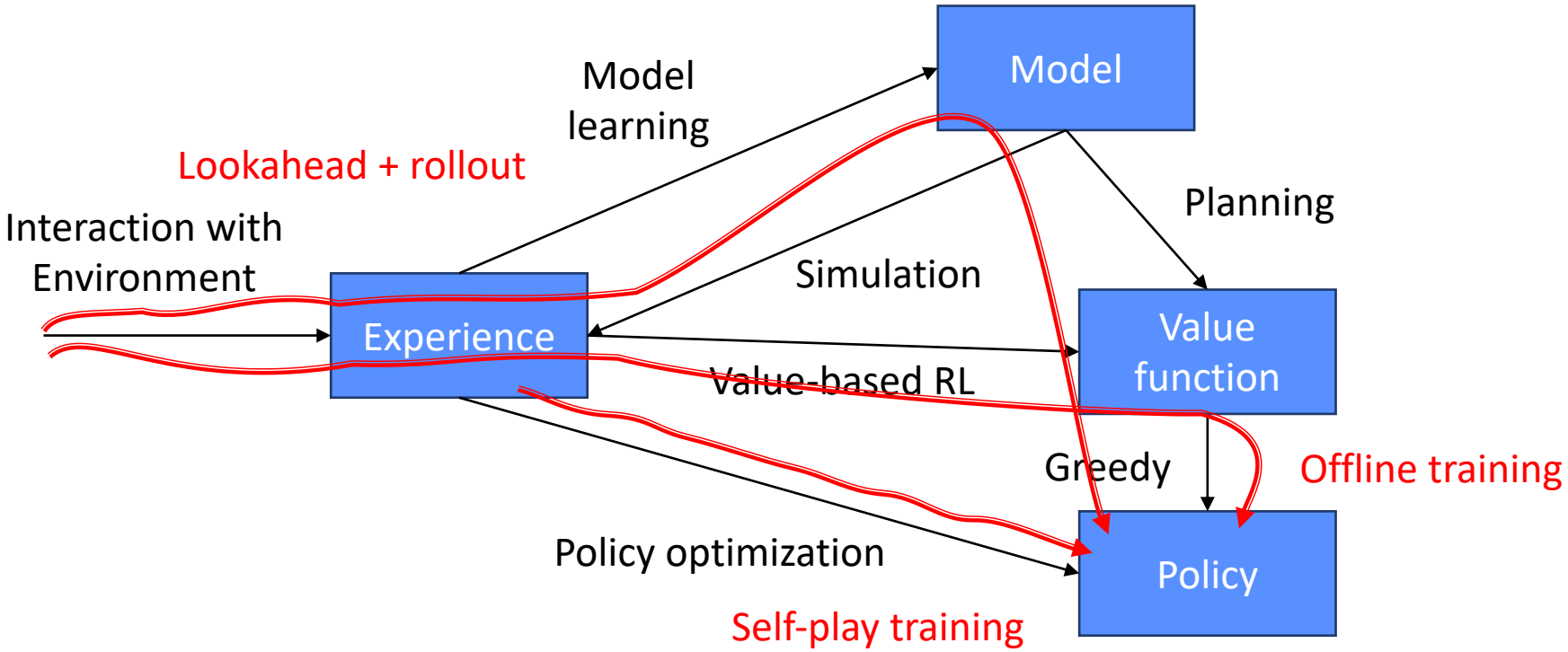
Sutton & Barto. Reinforcement Learning: An Introduction, 2018.

Sequential decision making



Adapted from Katerina Fragkiadaki (CMU)

Today



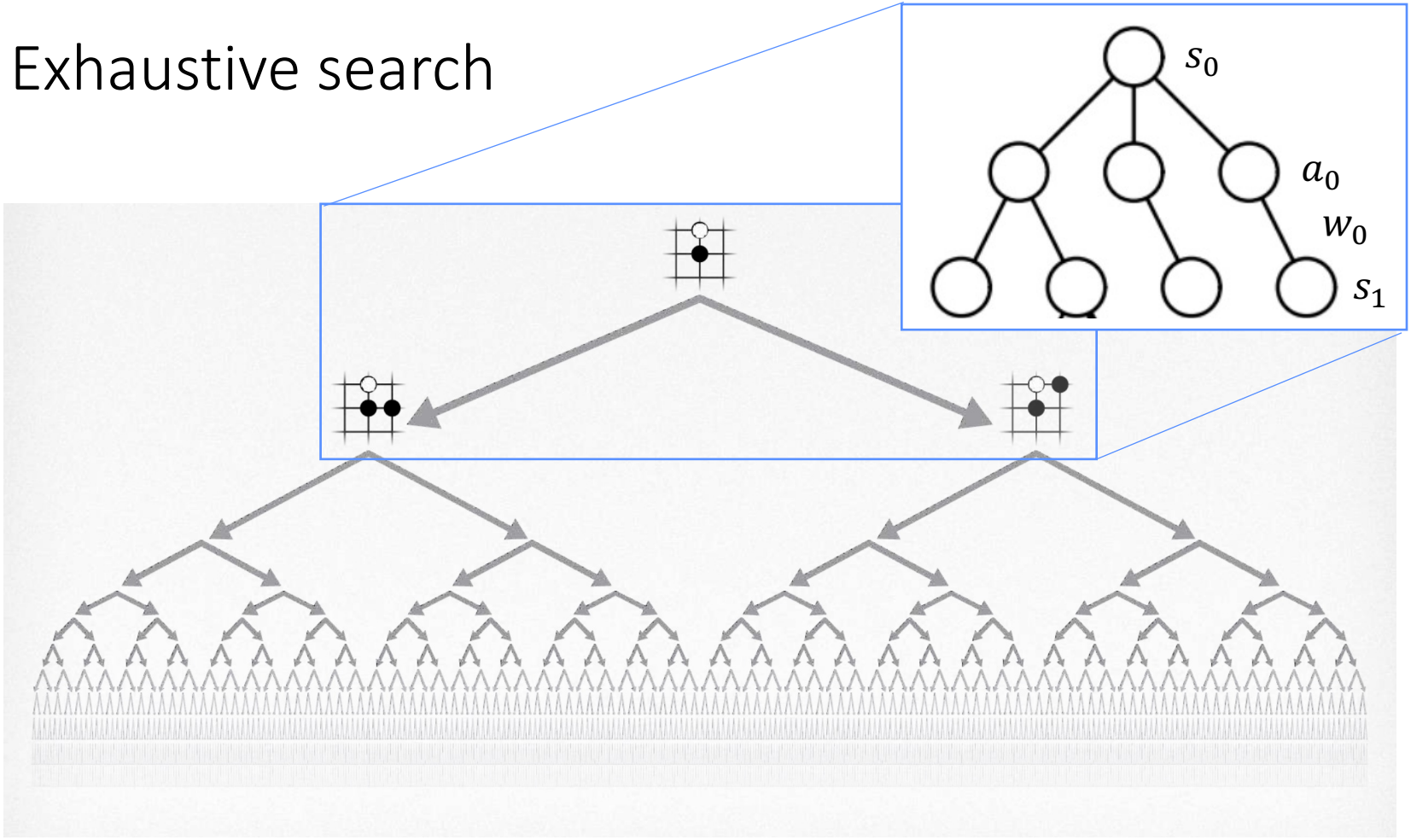
Adapted from Katerina Fragkiadaki (CMU)

Setting

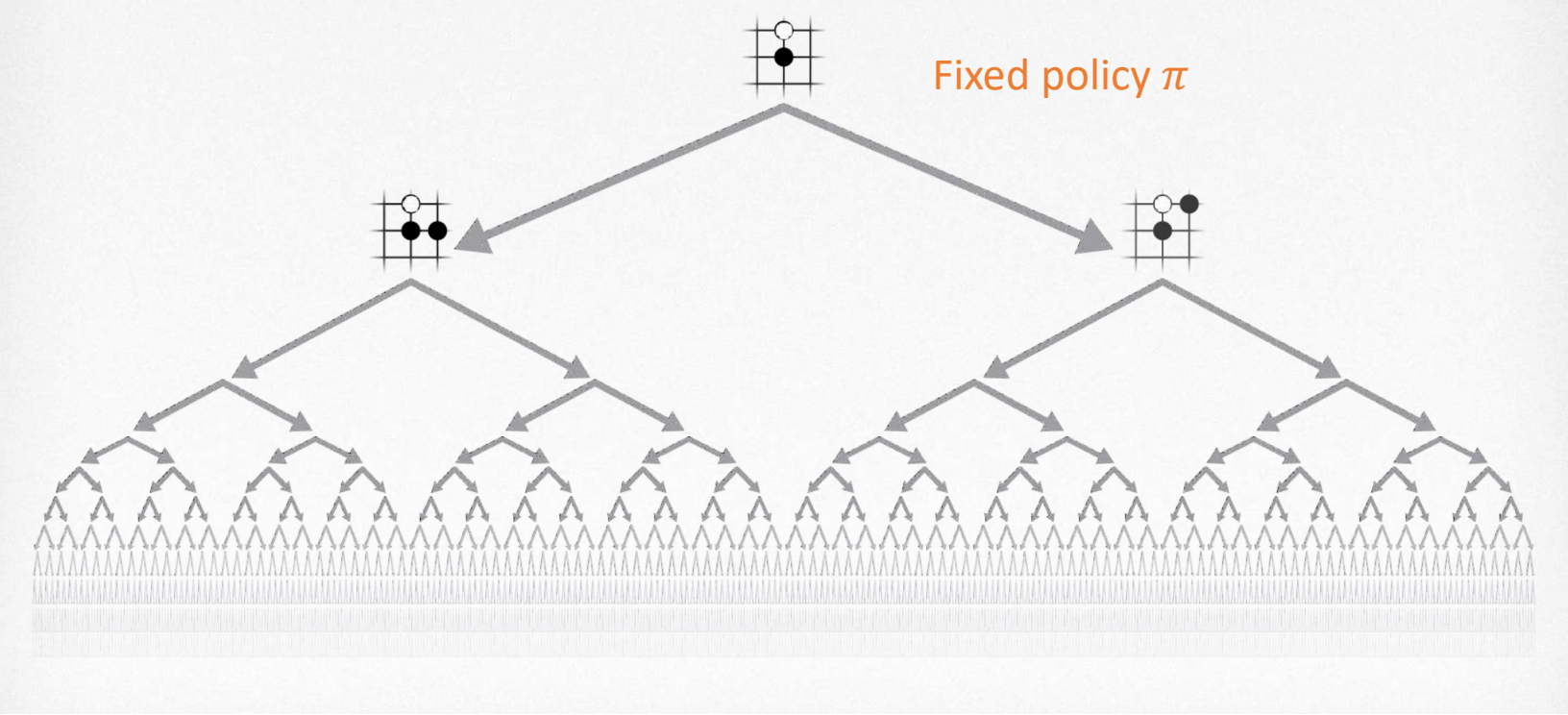
- An MDP with known $s' \sim P(s, a)$ – model-based method possible
- Terminal States (Finite Horizon Problem).
- Discrete State, Action spaces with fully observed (perfect information) states
- Challenges:
  - High branching factor
  - Inability to evaluate a state
  - Long term time dependencies
  - Long horizon

Adapted from Kendall Lowrey (UW)

Exhaustive search

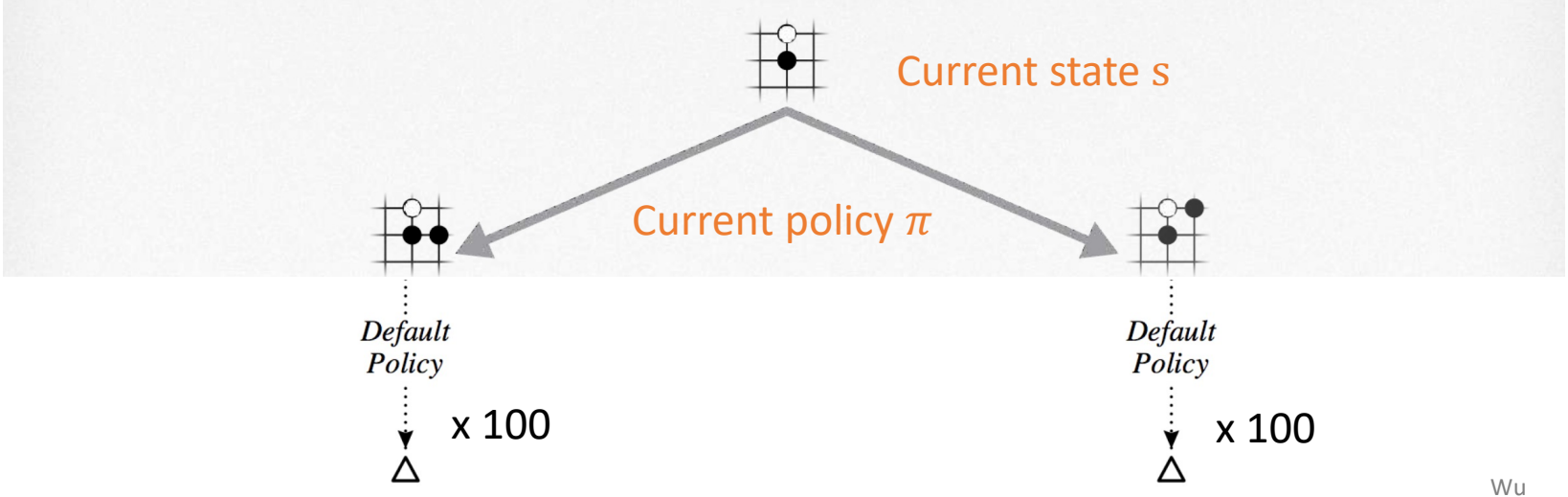


Exhaustive ~~search~~ policy evaluation / improvement



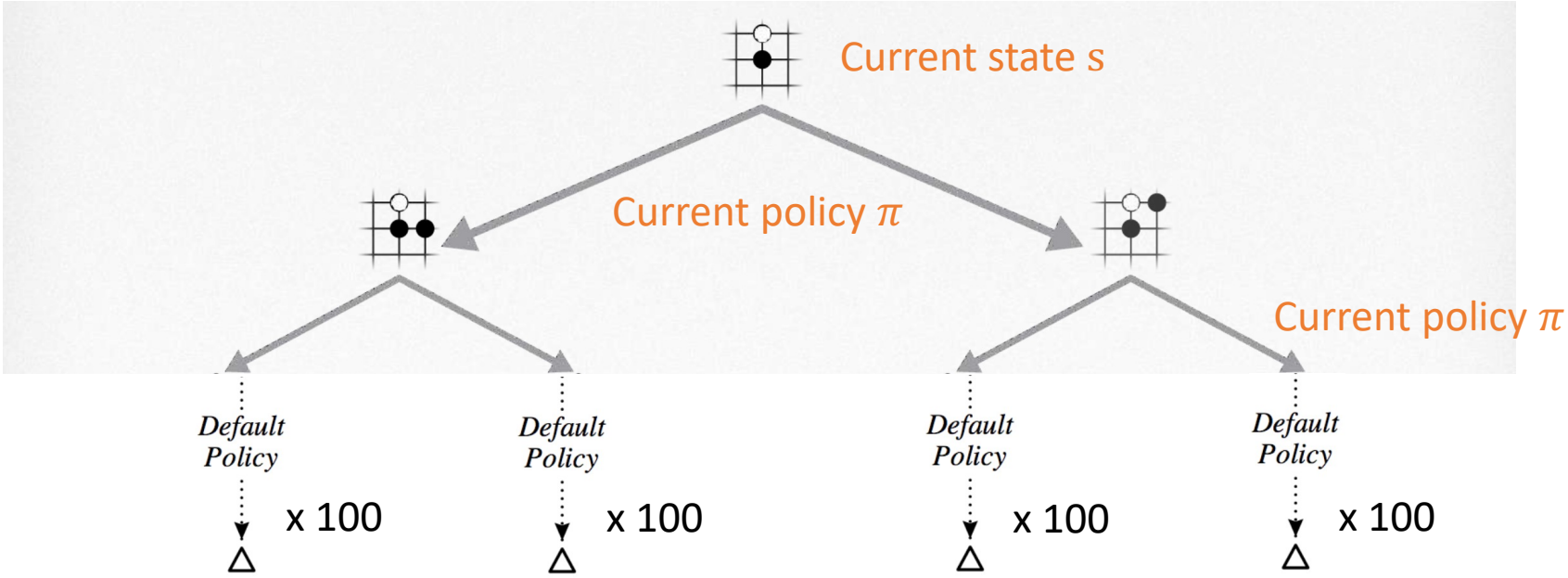
Online vs offline planning

- Offline planning: solve the full problem with DP
  - A full policy evaluation (and improvement) is intractable
- Online planning: only need local policy evaluation at current state s
- 1-step lookahead: Policy evaluation $\tilde{V}_\pi$ with MC rollouts
  - Policy improvement with greedy action selection, $T\tilde{V} =: T_{\pi_1}\tilde{V}$



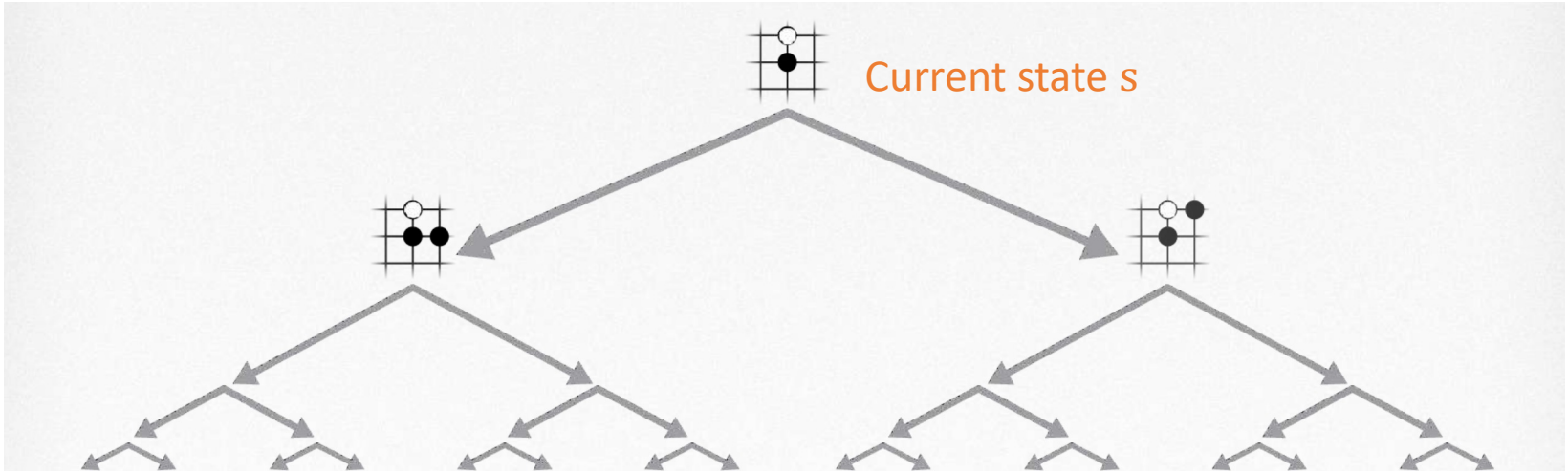
Online vs offline planning

- 2-step lookahead: Policy evaluation $\tilde{V}_\pi$ with MC rollouts
  - Policy improvement with greedy action selection
  - $T^2\tilde{V} =: T_{\pi_1}T_{\pi_2}\tilde{V}$



Lookahead

- k-step lookahead: Policy evaluation $\tilde{V}_\pi$ with MC rollouts
- If k is very large, we solve the problem to optimality
- If $\left\|\tilde{V} - V^*\right\|_\infty \leq \epsilon$, then $\left\|T^{k-1}\tilde{V} - V^*\right\|_\infty \leq \gamma^{k-1}\epsilon$
  - So, we are taking greedy wrt a better $\tilde{V}$



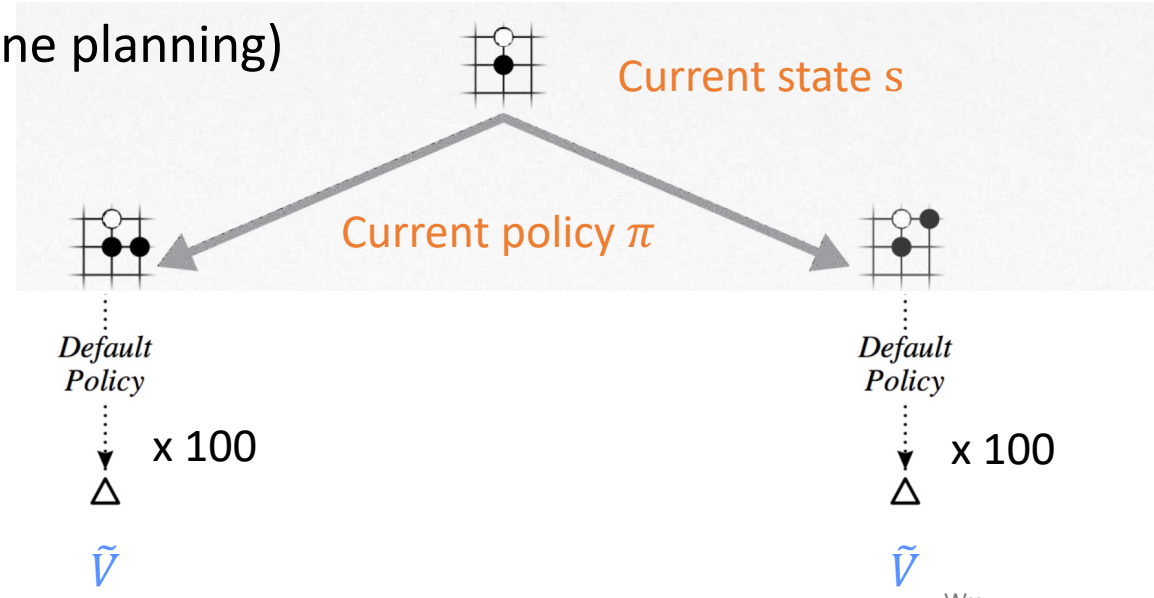
Lookahead

- Intuition: If $\tilde{V}$ is decent, then a few steps of policy iteration can give big improvements
  - Base policy $\pi \rightarrow$ local value function $\tilde{V}^{\pi} \rightarrow$ greedy improvement π'
  - This is essentially one step of policy iteration!
  - Base policy can be from offline training

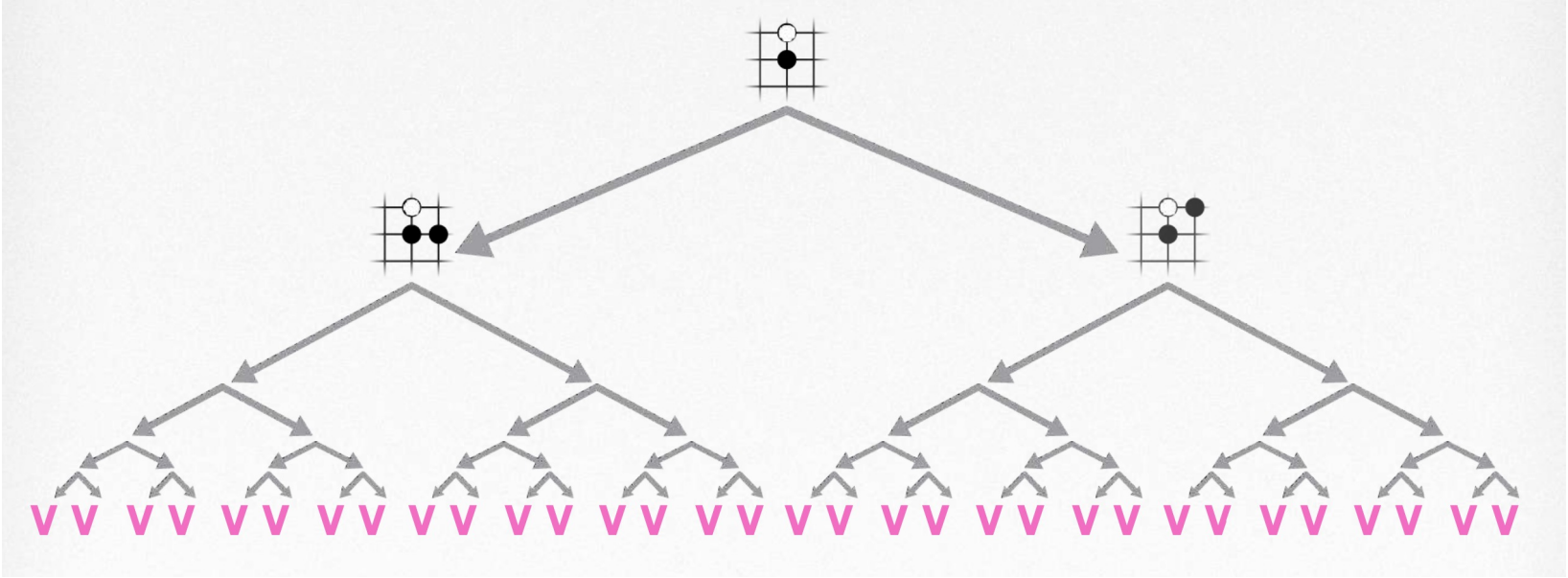
- Issues: large branching factor, deep tree
  - Exact if small k
  - Approximately if large k
    - Approx. solution to exact problem, e.g., value function approximation. Too general.
    - **Exact solution to easier version: Focused learning**

General structure

- k-step lookahead: Policy evaluation $\tilde{V}_\pi$ with MC rollouts
  - Policy improvement with greedy action selection
- If horizon long (or infinite), can truncate MC rollout with offline trained $\tilde{V}$
- Application matters (online planning)
  - Inventory: overnight
  - Chess: 5 seconds
  - Car: 0.1 seconds



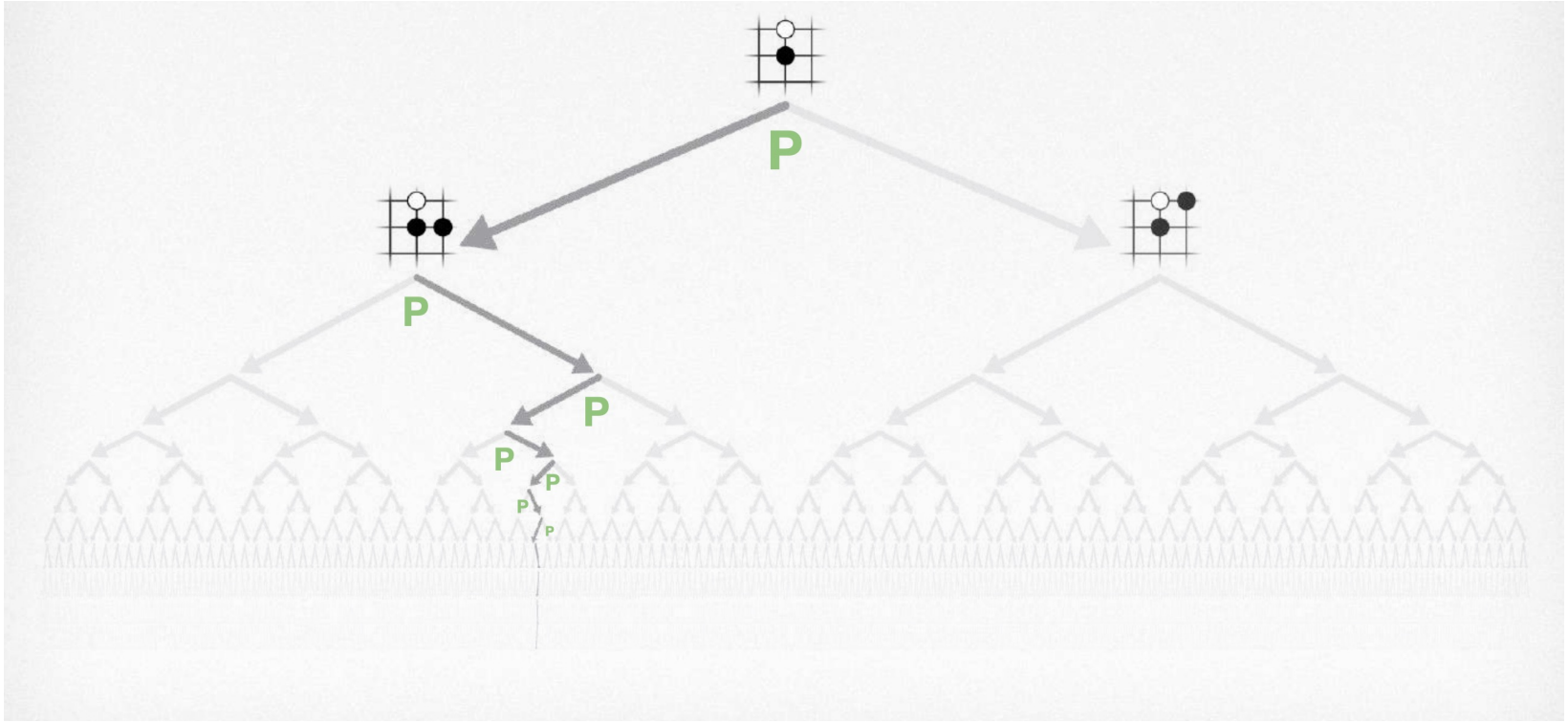
Reducing depth with value network



TD-Gammon

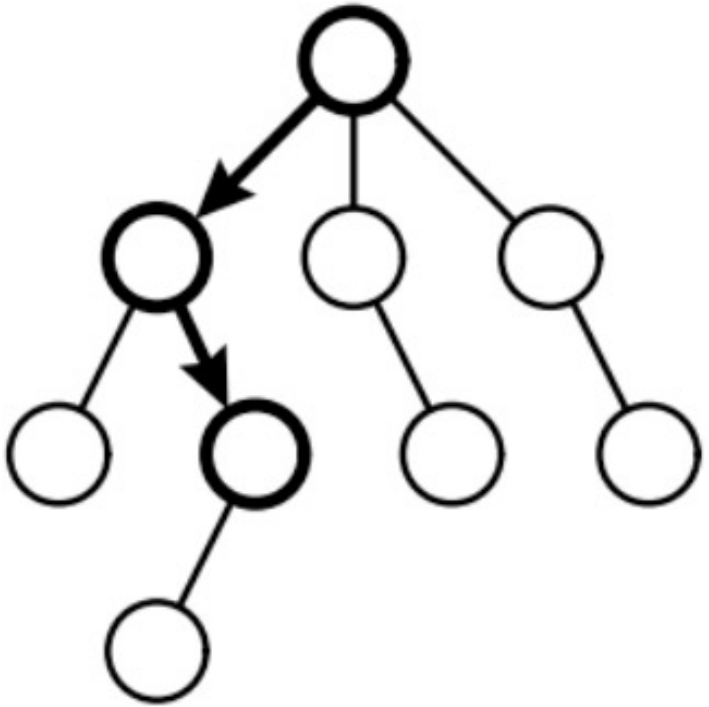
- Game Backgammon
- $\tilde{V}$: Offline training using $TD(\lambda)$
- World-class player
  - Use greedy + rollouts
  - 2-step lookahead
- Similarly for Tetris

Reducing breadth with policy network – How?



Intuition

- So far: Knowing $s' \sim P(s, a)$ can let us build a tree structure for mapping how states branch into next states through actions.
- If the **branching factor** is too high, we can never fully build out the tree.
- If we do not build out the whole tree, we cannot follow the branches that lead us to high reward (aka **winning**).
- How do we know which branches to take?

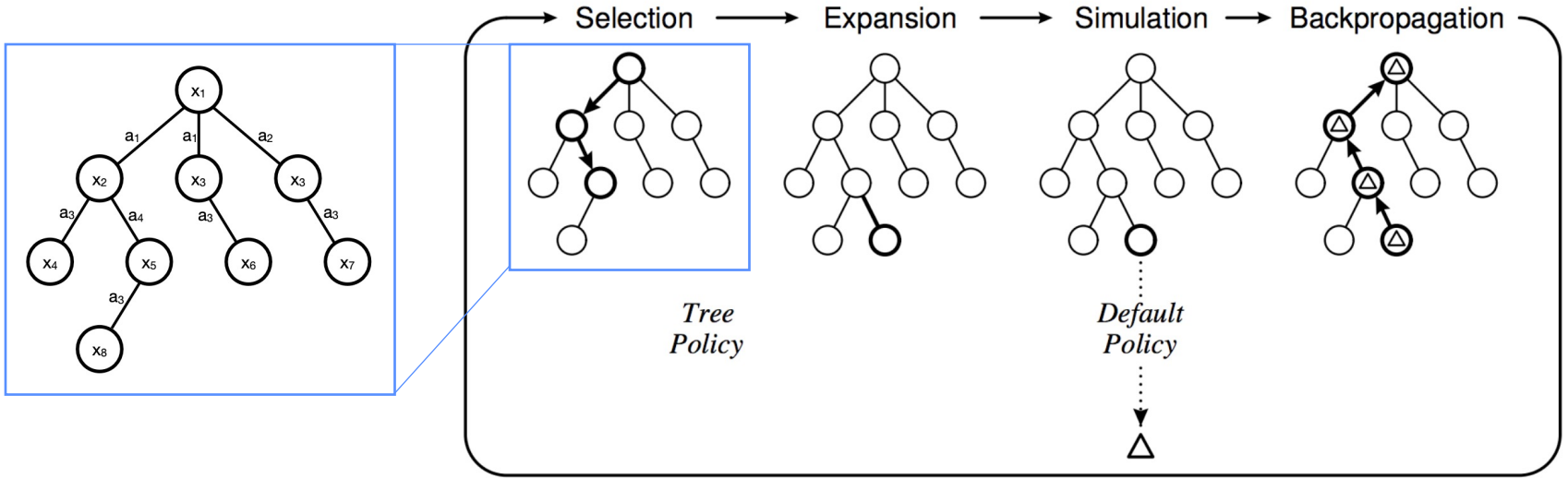


Adapted from Kendall Lowrey (UW)

Outline

1. Online planning
2. **Monte Carlo Tree Search (MCTS)**
   a. Selection, Expansion, Simulation, Backpropagation
   b. Upper Confidence Bound (UCB)
   c. Upper Confidence Tree (UCT)
3. AlphaGo: Learning-guided MCTS

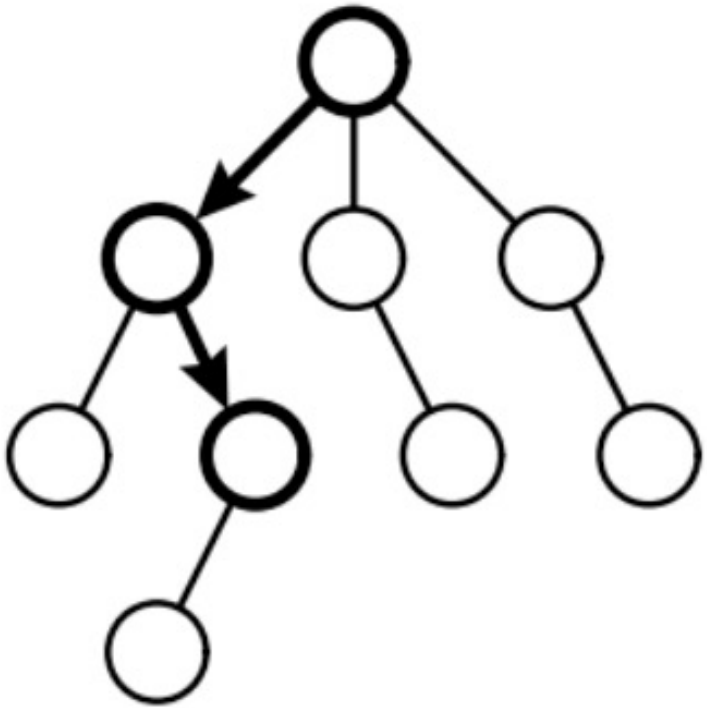
Monte-Carlo Tree Search



Kocsis and Szepesvari, “Bandit based Monte-Carlo Planning,” 2006.
Browne, et al. “A survey of Monte Carlo Tree Search Methods.” IEEE Transactions on Computational Intelligence and AI in Games, 2012.

Selection step: Tree traversal

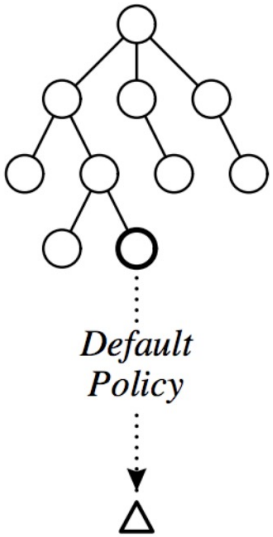
- In contrast to lookahead (exhaustive action selection), we will now be deliberate about selecting actions and thus traversing (and expanding) the tree.
- We will discuss how.

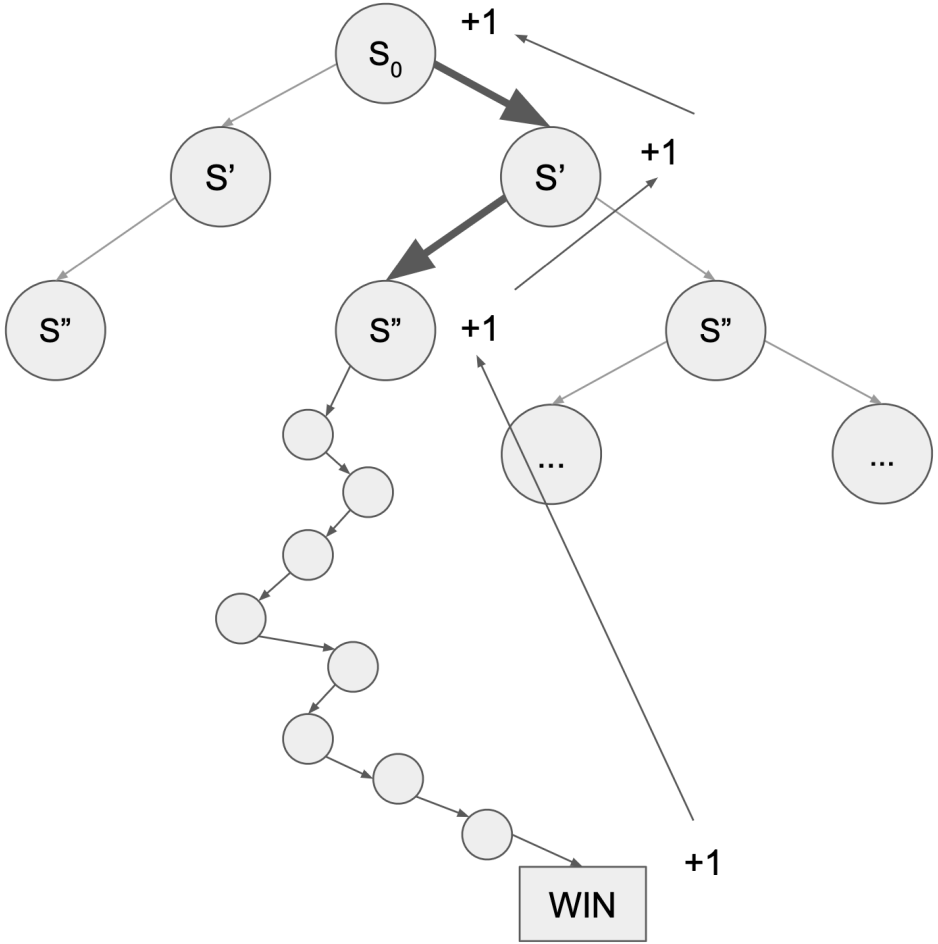


Adapted from Kendall Lowrey (UW)

Simulation step: rollout

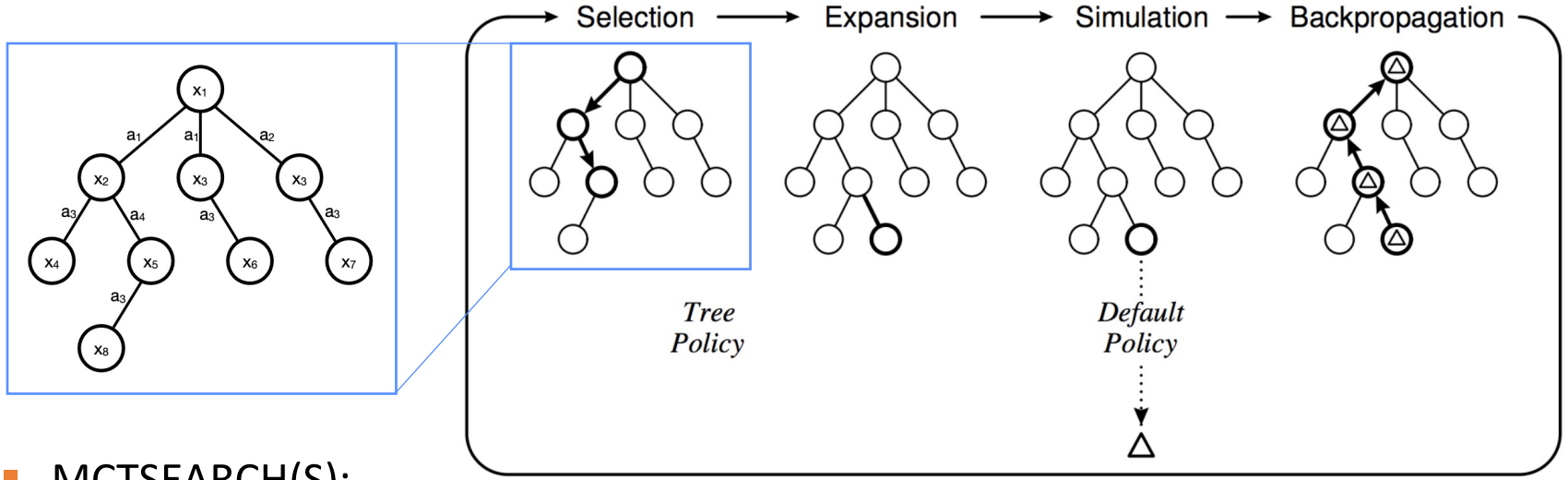
- Similarly to lookahead approach, we will use MC rollouts to estimate the local policy evaluation





Adapted from Kendall Lowrey (UW)

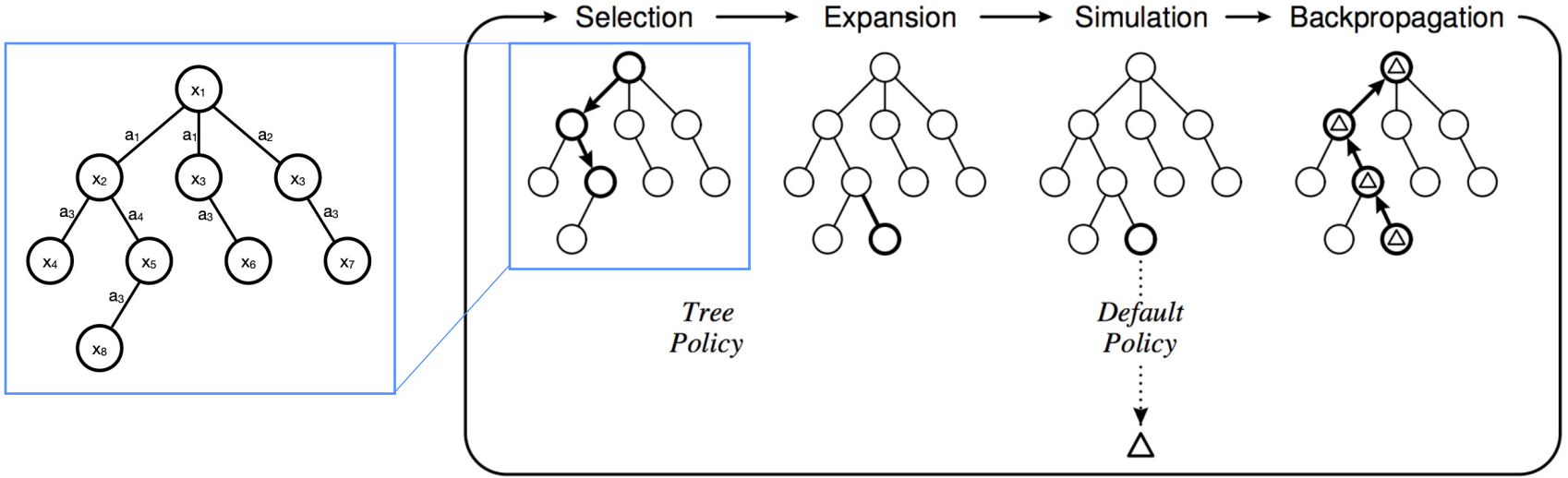
Monte-Carlo Tree Search Algorithm



- MCTSEARCH(S):
  - create new tree T with state S as root
  - LOOP:
    - L ← TRAVERSE T until leaf node
    - V ← SIMULATE MDP from L
    - BACKUP value V through T
  - return BESTCHILD(S)

Kocsis and Szepesvari, “Bandit based Monte-Carlo Planning,” 2006.
Browne, et al. “A survey of Monte Carlo Tree Search Methods.” IEEE Transactions on Computational Intelligence and AI in Games, 2012.

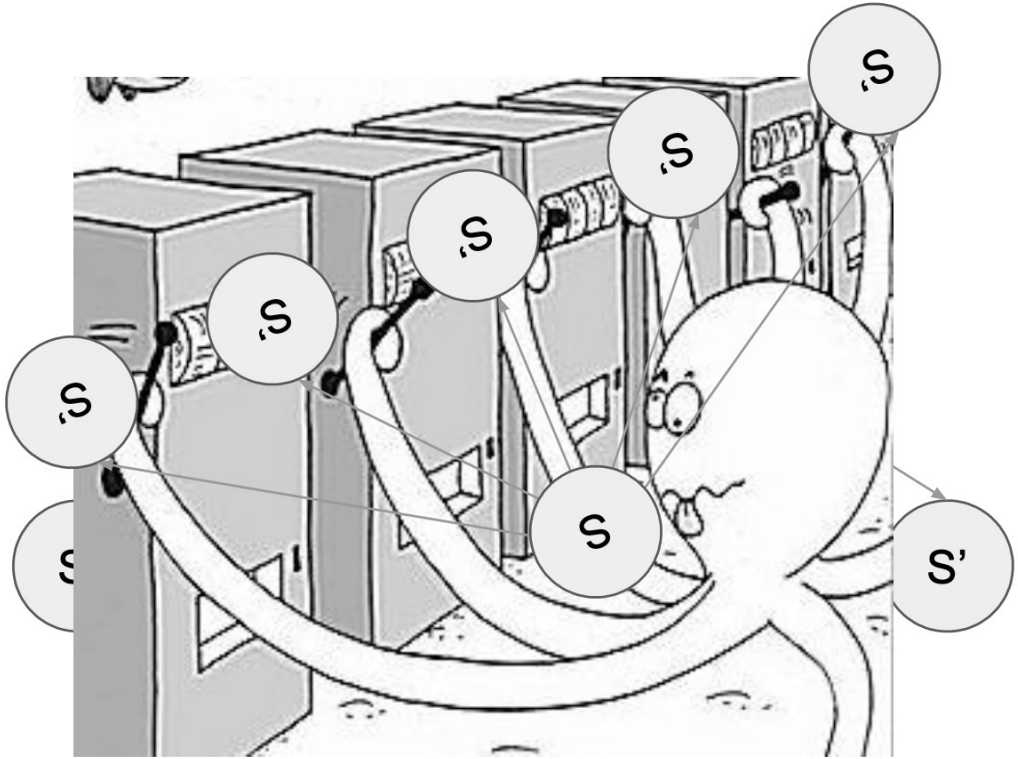
Monte-Carlo Tree Search Algorithm



- No heuristics needed: we can default to random search
- No minimum computation: we can query for action at anytime
- No symmetries: the tree growth has no rules, and depends on how you traverse

Kocsis and Szepesvari, “Bandit based Monte-Carlo Planning,” 2006.
Browne, et al. “A survey of Monte Carlo Tree Search Methods.” IEEE Transactions on Computational Intelligence and AI in Games, 2012.

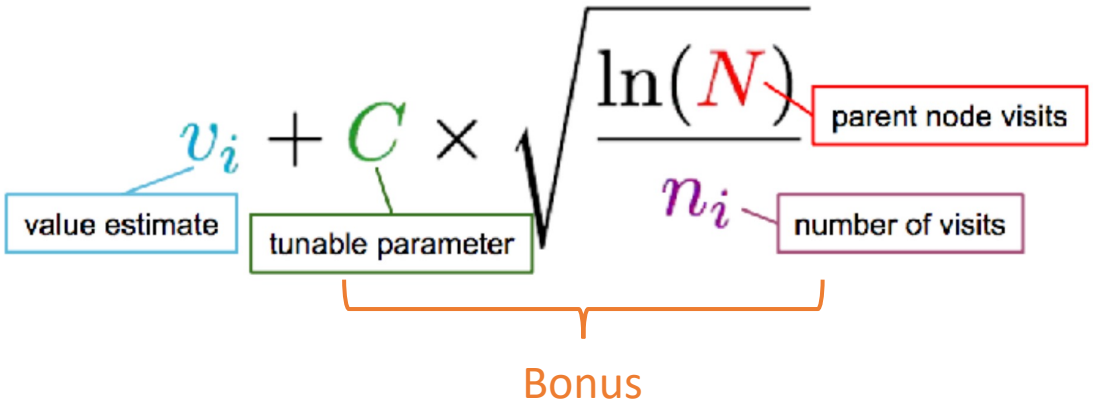
How do we traverse our tree?



- Each node of the tree is like a multi-armed bandits (MAB) problem....

Upper-Confidence Bound (UCB)

- Principle: optimism in the face of uncertainty
- Sample actions according to the following score:

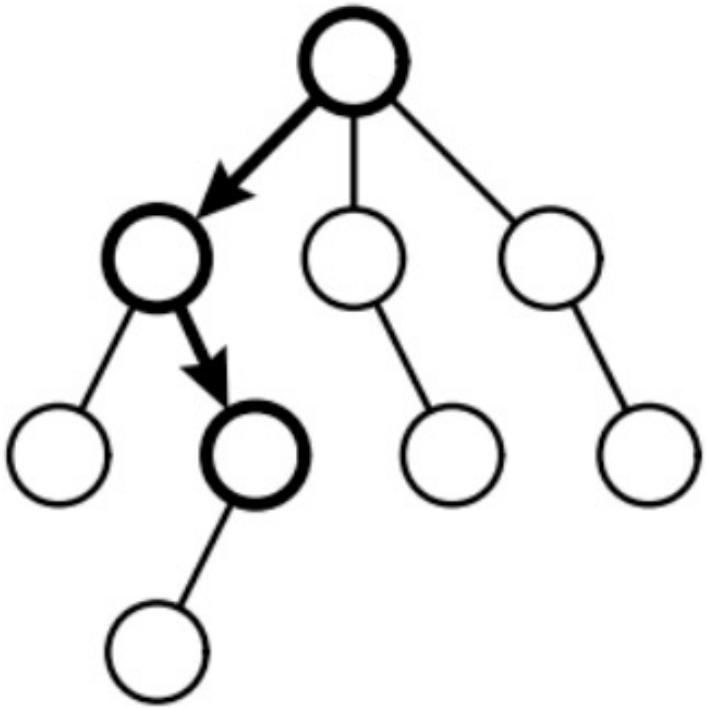


- score is decreasing in the number of visits (explore)
- score is increasing in a node's value (exploit)
- always tries every option once

Finite-time Analysis of the Multiarmed Bandit Problem, Auer, Cesa-Bianchi, Fischer, 2002

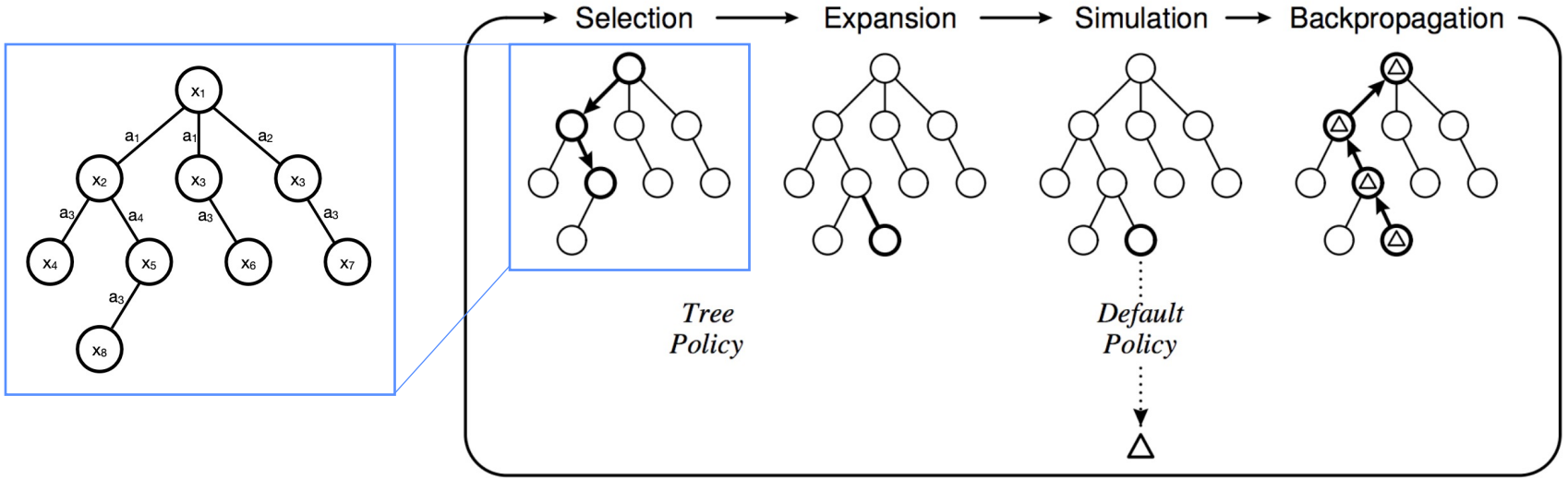
Upper confidence tree (UCT)

- UCT = MCTS + UCB
  - Treat each internal node in MCTS as a K-armed bandit
  - Use UCB to select action



Levente Kocsis, Csaba Szepesvari, and Jan Willemson. Improved Monte-Carlo Search., 2006

Monte-Carlo Tree Search



1. Selection (traverse)
   - Used for nodes we have seen before
   - Pick according to UCB (i.e., UCT)
2. Expansion
   - Used when we reach the frontier
   - Add one node per playout
3. Simulation (rollout)
   - Used beyond the search frontier
   - Don't bother with UCB, just play randomly
4. Backpropagation (backup)
   - After reaching a terminal node
   - Update value and visits for states expanded in selection and expansion

Kocsis and Szepesvari, "Bandit based Monte-Carlo Planning," 2006.
Browne, et al. "A survey of Monte Carlo Tree Search Methods." IEEE Transactions on Computational Intelligence and AI in Games, 2012.

UCT MCTS Algorithm

- MCTSEARCH(S):
  - create new tree T with state S as root
  - LOOP:
    - L ← TRAVERSE T until leaf node w/ UCT
    - V ← SIMULATE MDP from L
    - BACKUP value V through T
  - return BESTCHILD(S)

- BESTCHILD(S):
  - return argmax UCT(S_j,S)
- BACKUP(S,V):
  - N(S) += 1
  - Q(S) += V
  - BACKUP(parent(S),V)

- If we run this for some set time / computational limit, the hope is that the most explored branches are the same as the highest reward branches.

Adapted from Kendall Lowrey (UW)

Can we do better?

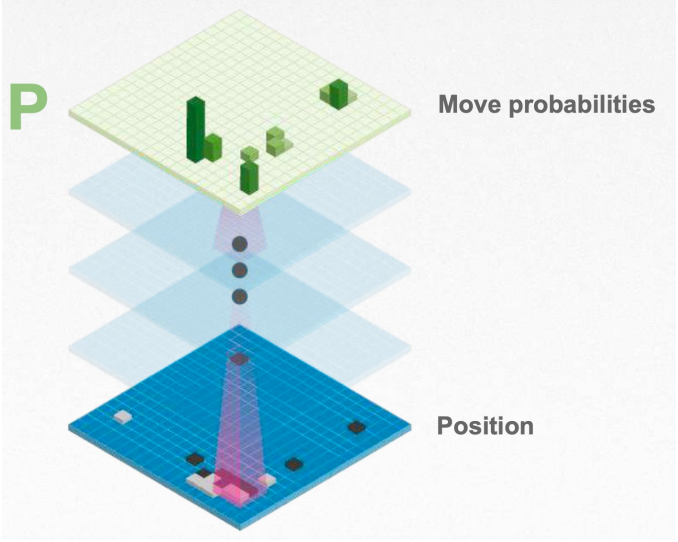
- Can we inject prior knowledge into value functions to be estimated and actions to be tried, instead of initializing uniformly?

Outline

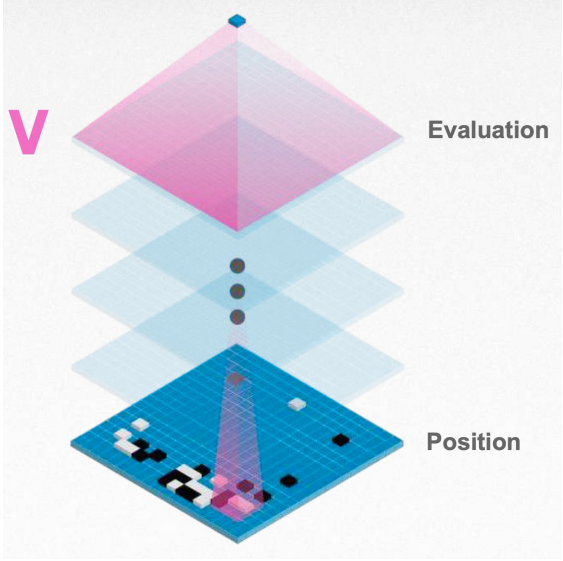
1. Online planning
2. Monte Carlo Tree Search (MCTS)
3. **AlphaGo: Learning-guided MCTS**
   a. AlphaGo
   b. AlphaGoZero
   c. AlphaZero, MuZero
   d. Learning-guided optimization

AlphaGo: Learning-guided MCTS

- Value neural net to evaluate board positions
- Policy neural net to select moves
- Combine those networks with MCTS

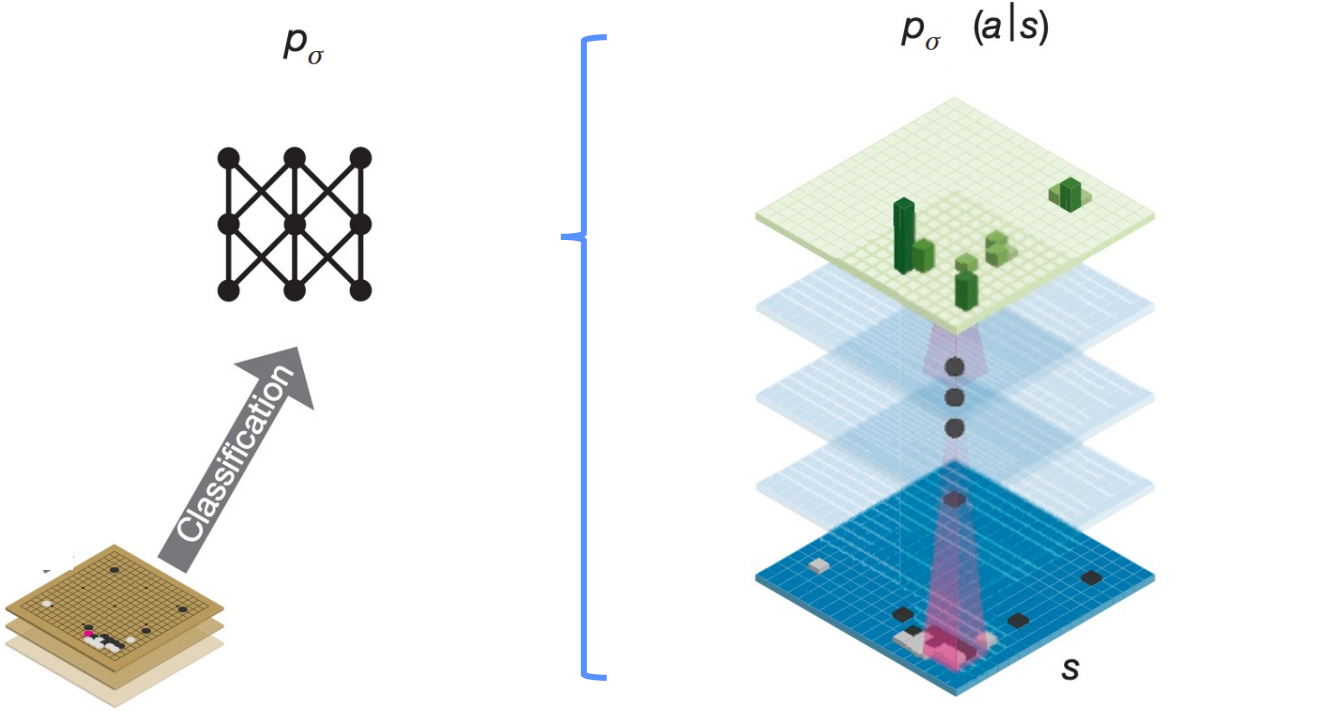


Policy network



Value network

Training the policy network



Human expert positions

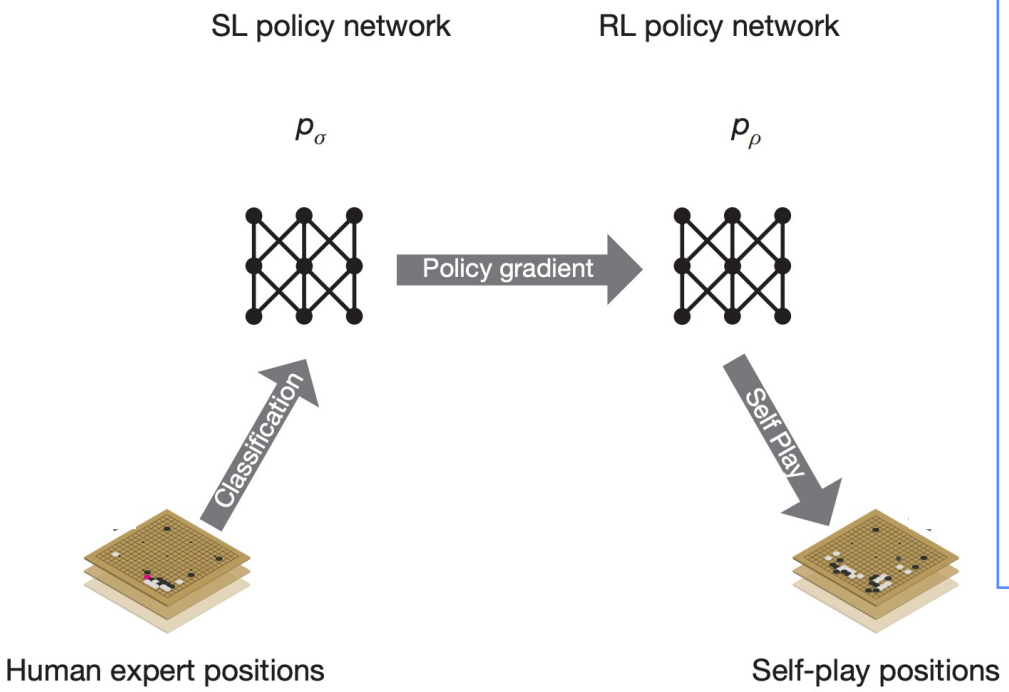
30M Game States

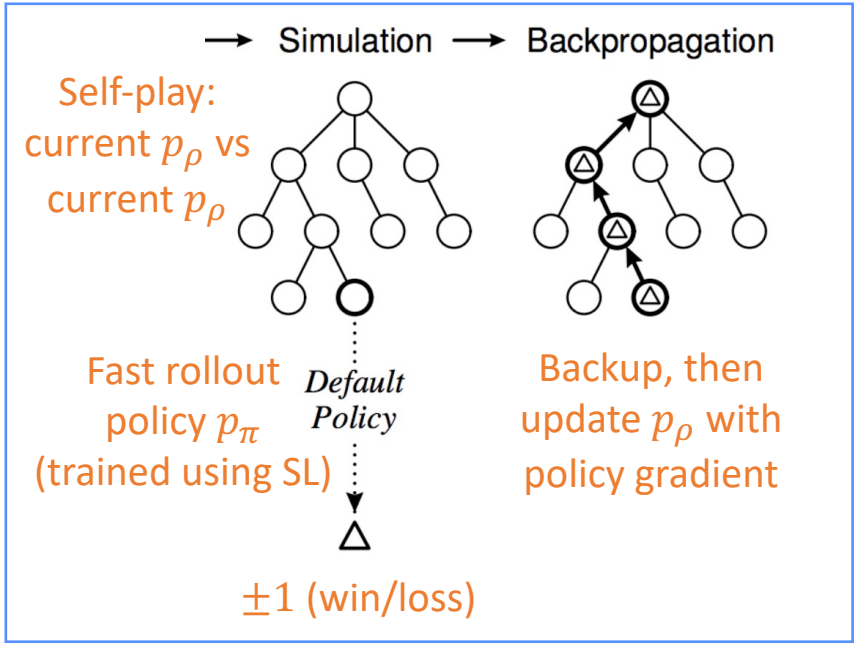
13-layer CNN

3 weeks of training, 50 processors

~57% accuracy

Improving the policy network with policy gradient

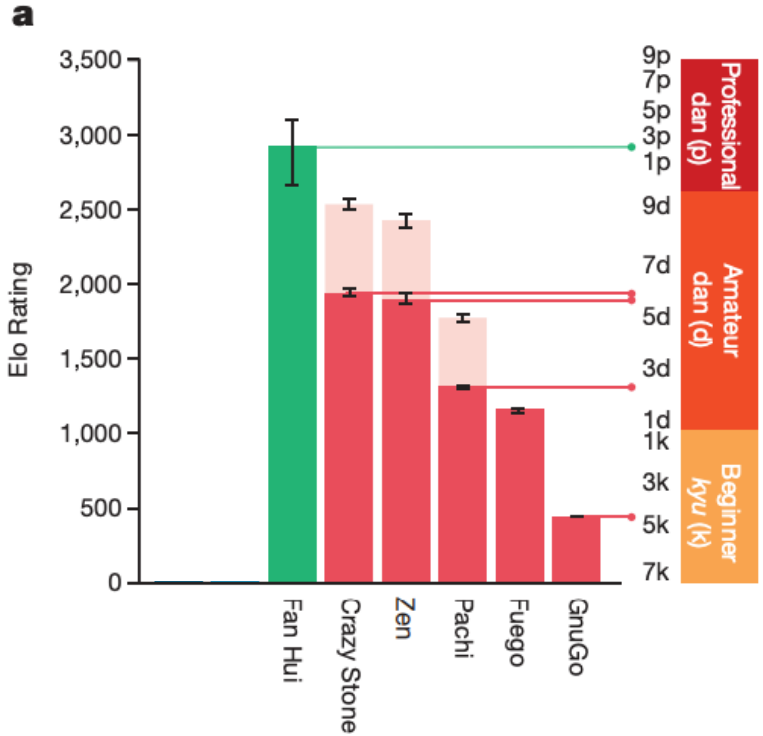




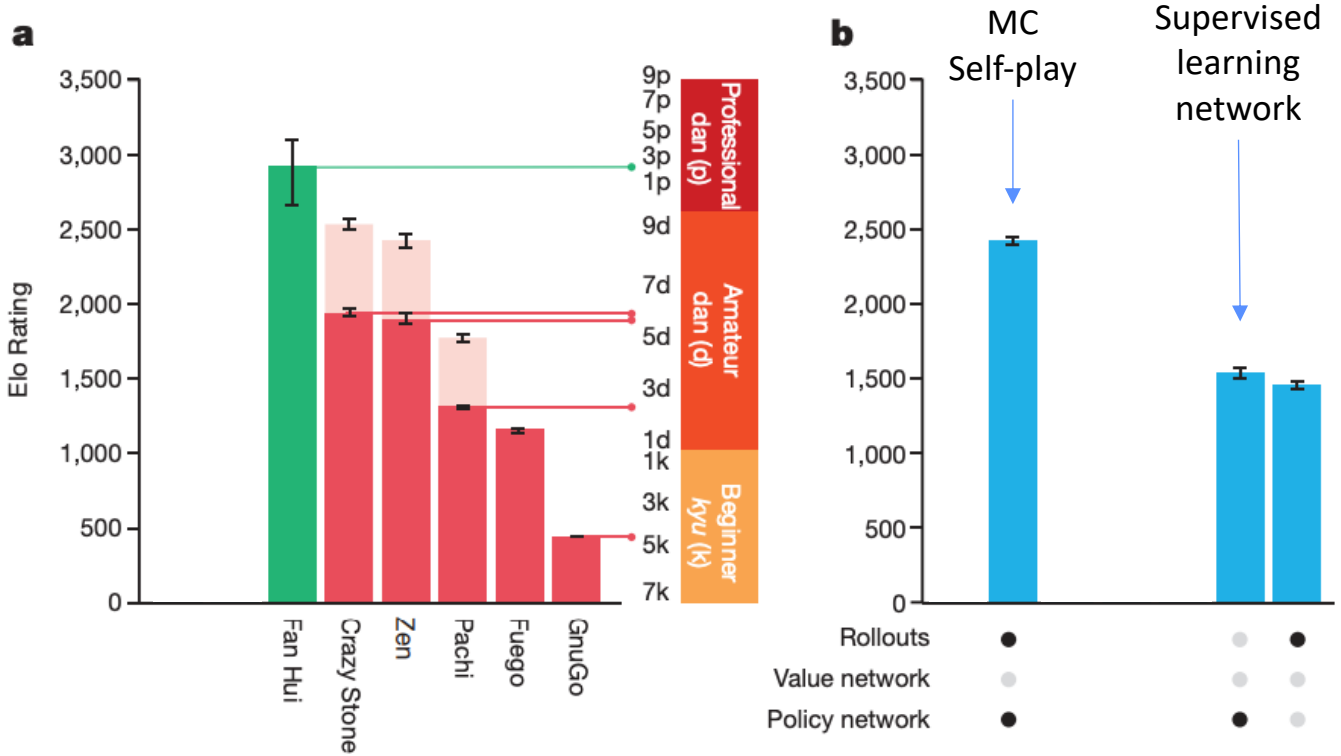
RL policy wins 80% games against SL policy

1 day of training, 50 GPUs

Performance

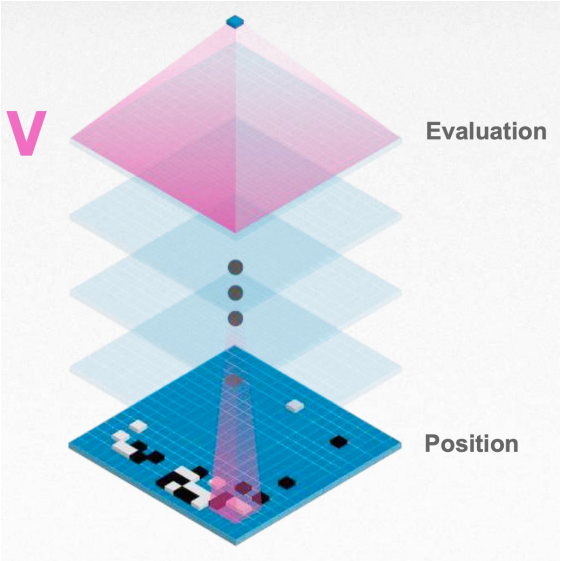


Performance

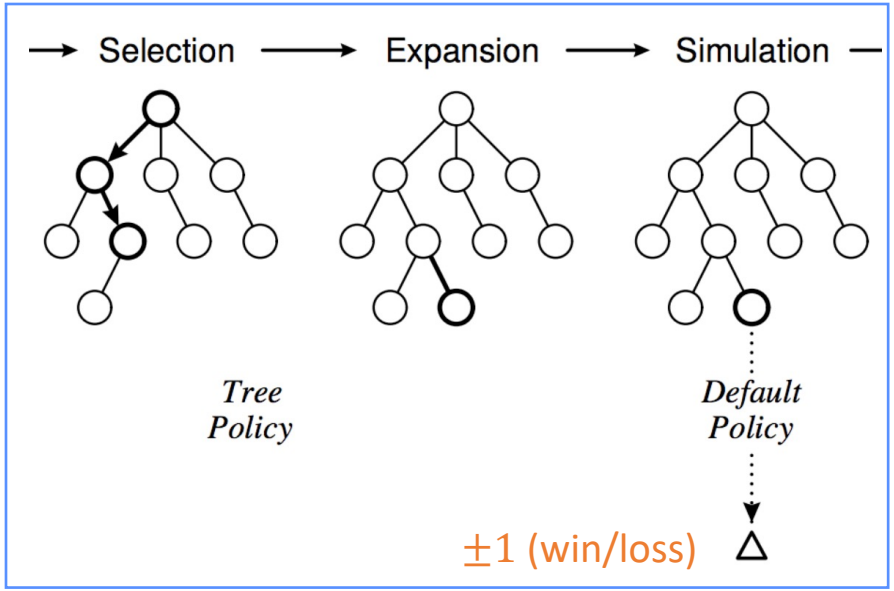


Improving selection with value function estimation

- As p_ρ improves, can use win/loss to inform selection & expansion
- Train value network v_θ via regression on outcome z of playing with p_ρ.



Value network

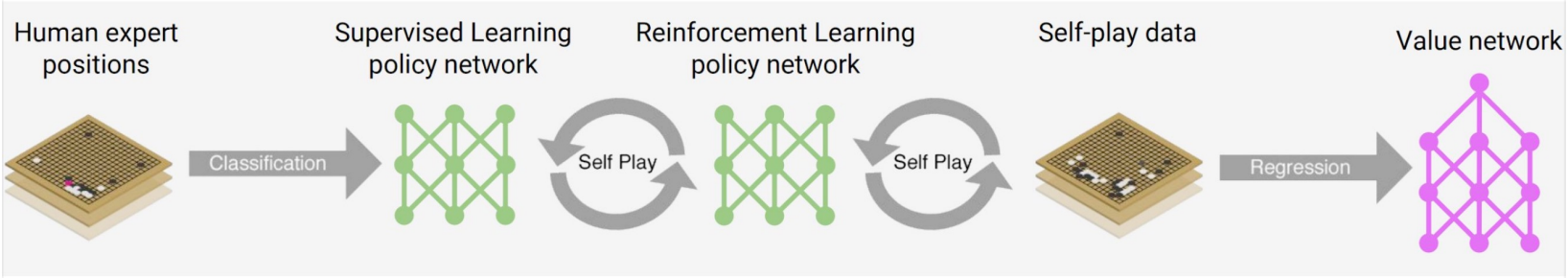


13-layer CNN

1 weeks of training, 50 GPUs

50M mini-batches (of size 32) from 30M game positions from unique self-play games

AlphaGo Lee: Overall MCTS pipeline



- Policy Network p_σ augments UCT during **Selection** (traverse) step

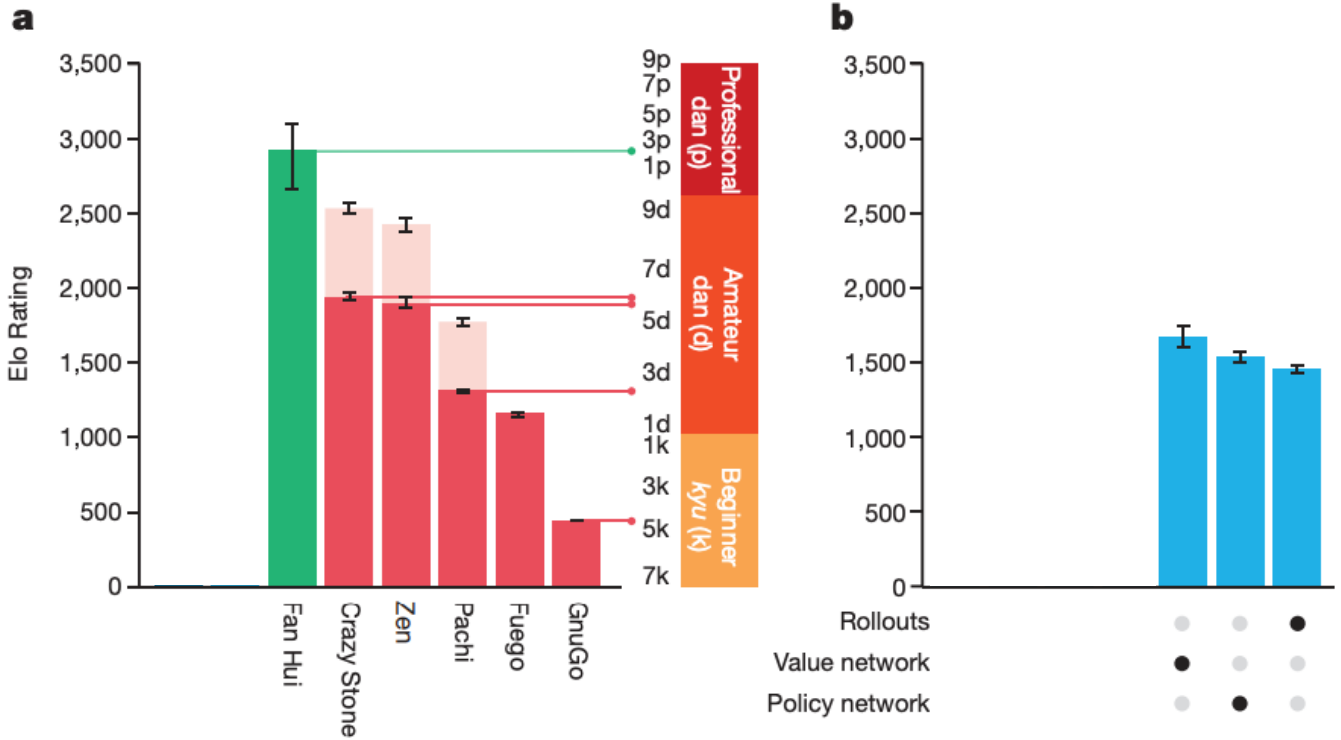
$$a_t = \text{argmax}_\text{a}[Q(s_t, a) + u(s_t, a)]$$

$$u(s, a) \propto \frac{p_\sigma(a|s)}{1 + N(s, a)}$$

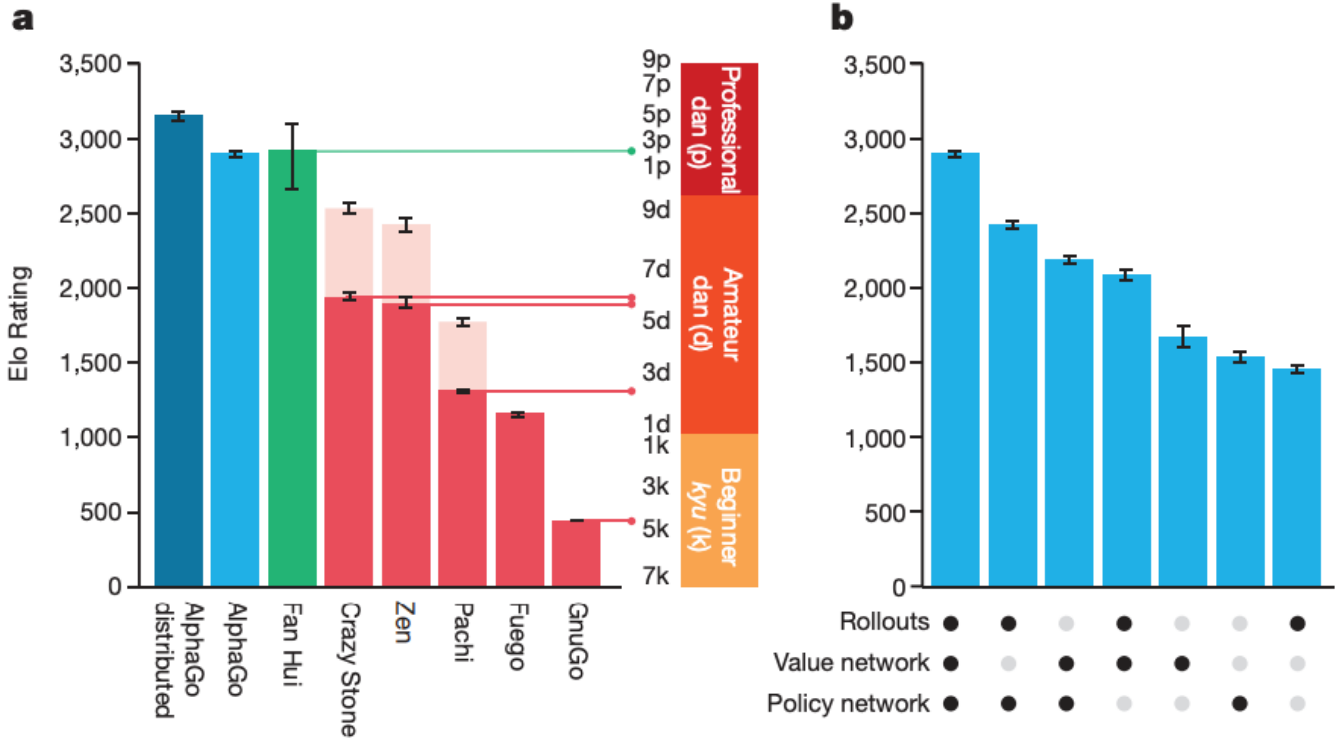
- Value Network v_θ is combined with fast rollouts (z, using p_π) for **Simulation** step

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$$

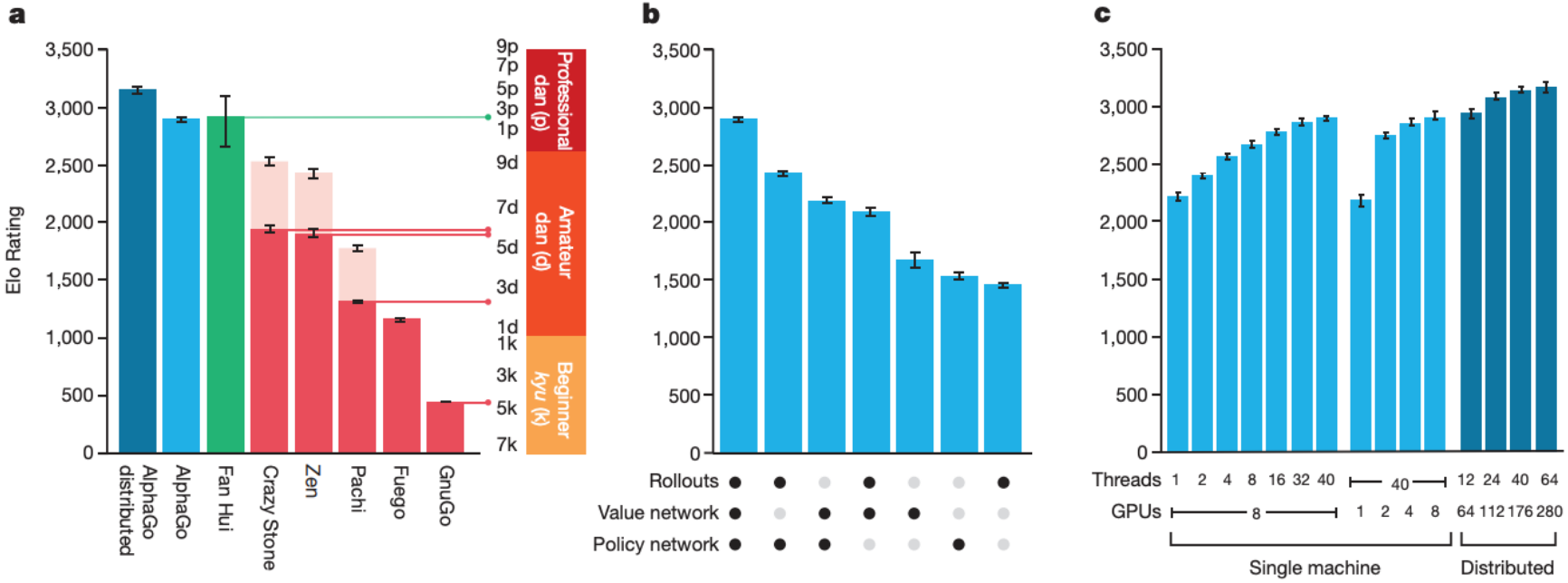
Performance



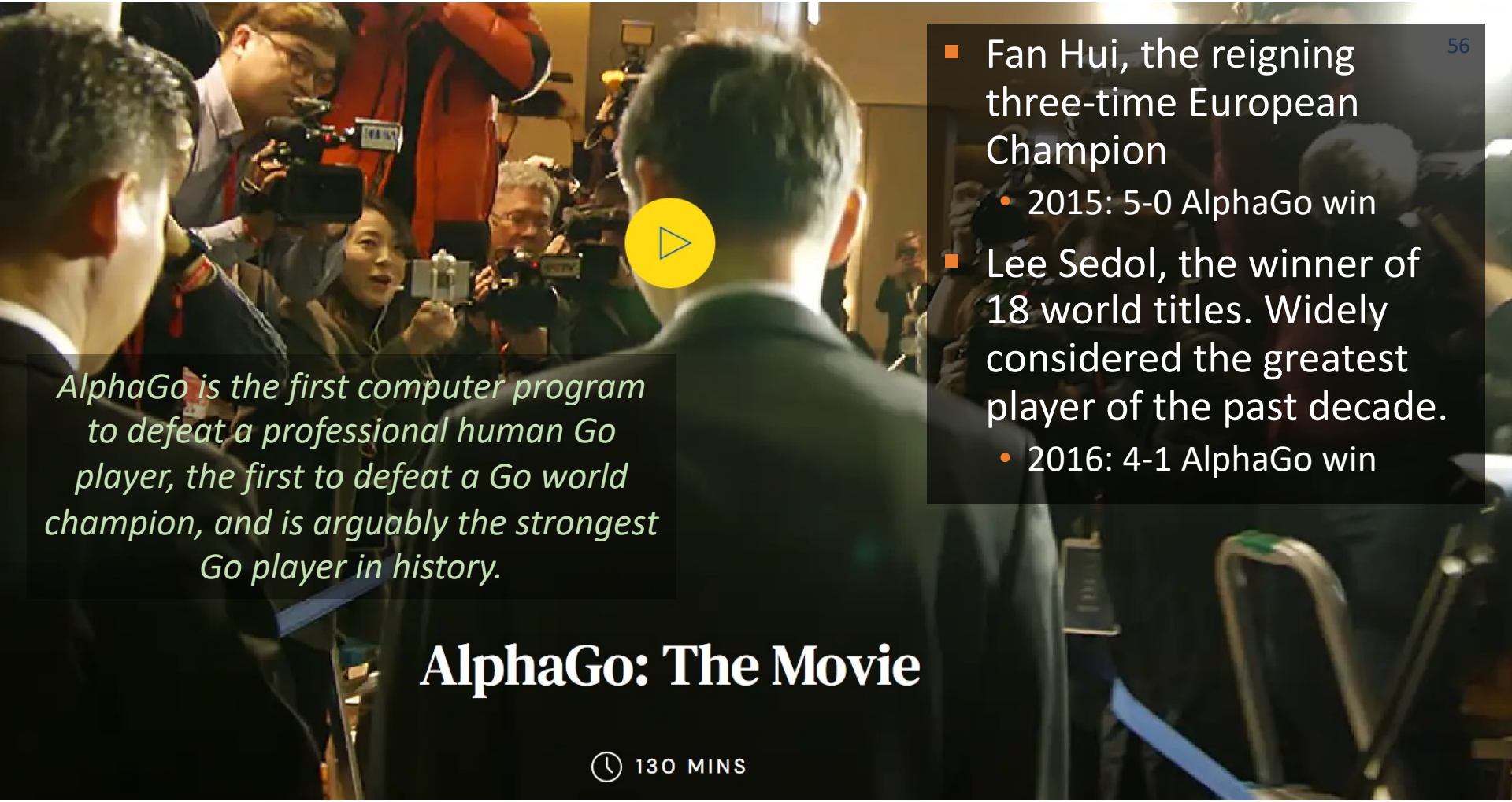
Performance



Performance



Wu



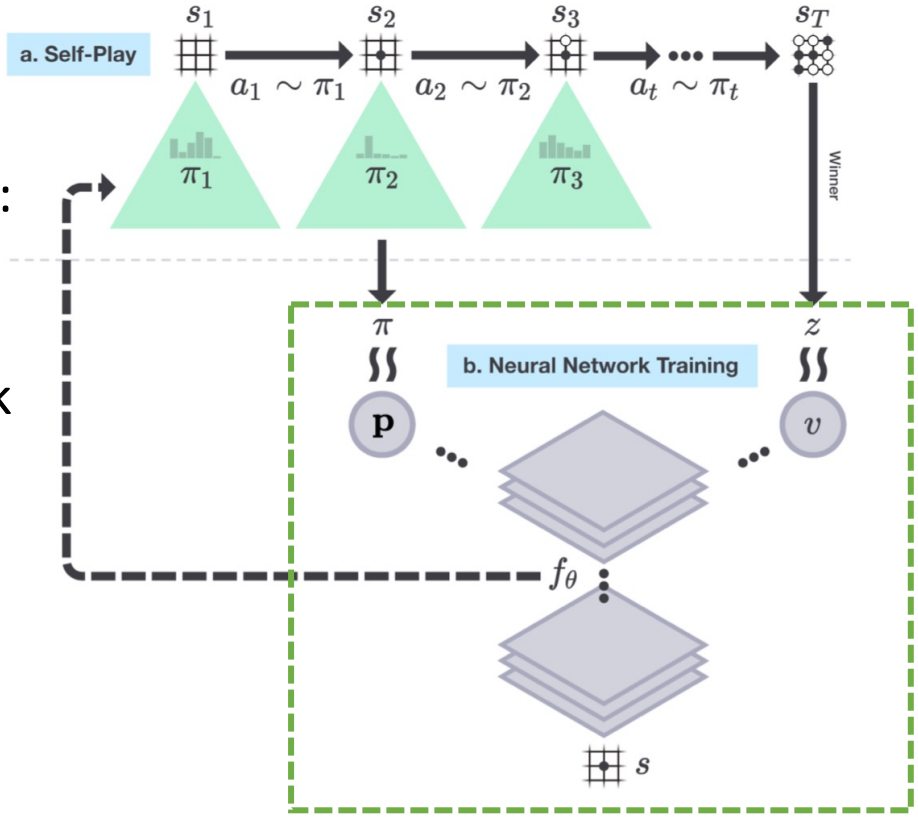
- Fan Hui, the reigning three-time European Champion
  - 2015: 5-0 AlphaGo win
- Lee Sedol, the winner of 18 world titles. Widely considered the greatest player of the past decade.
  - 2016: 4-1 AlphaGo win

AlphaGo is the first computer program to defeat a professional human Go player, the first to defeat a Go world champion, and is arguably the strongest Go player in history.

https://www.youtube.com/watch?v=WXuK6gekU1Y

AlphaGo Zero (2017)

- Simpler method!
- No more supervised learning step: straight to RL Policy network starting from random play.
- Policy Network and Value network are combined: more efficient training.
  - A single 44-layer DNN (ConvNet)
  - 2 heads: Value, Action probabilities (19x19 + 1)
- No ‘Simulation’ step: direct board evaluation with value function.

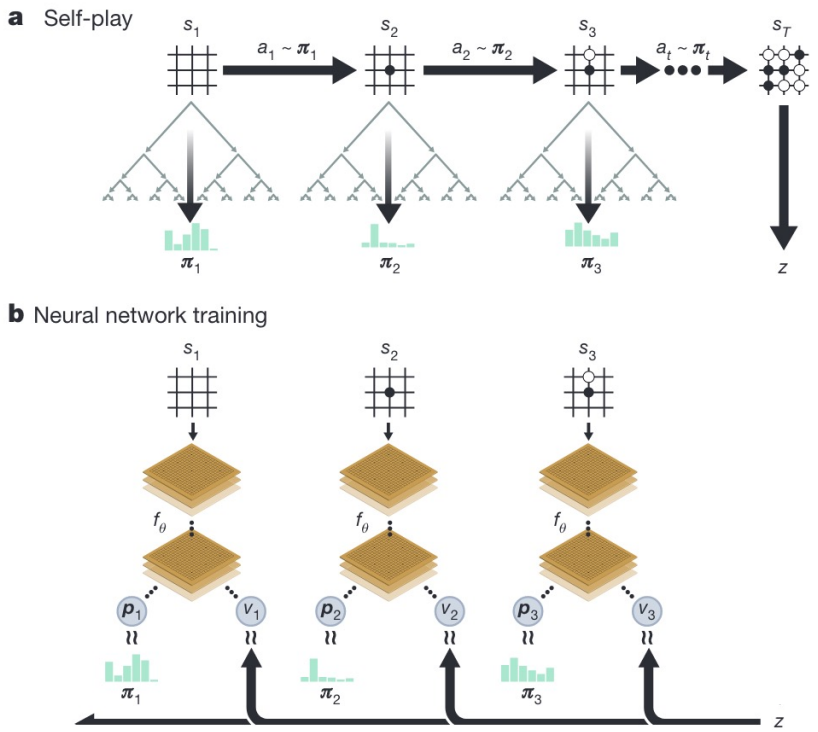


4.9-29M self-play games

3-40 days of training

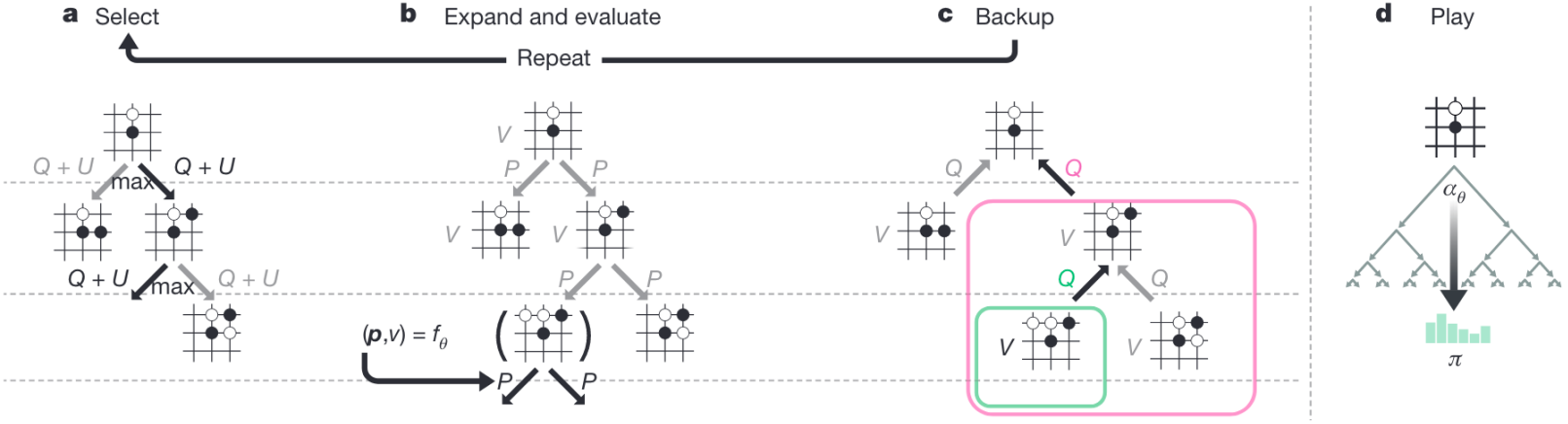
AlphaGoZero: Lookahead search during training!

- Given any policy, a MCTS guided by this policy will produce an improved policy (policy improvement operator)
- Train so that the policy network mimics this improved policy
  - Maximum likelihood
- Train so that the position evaluation network output matches the outcome (same as in AlphaGo)



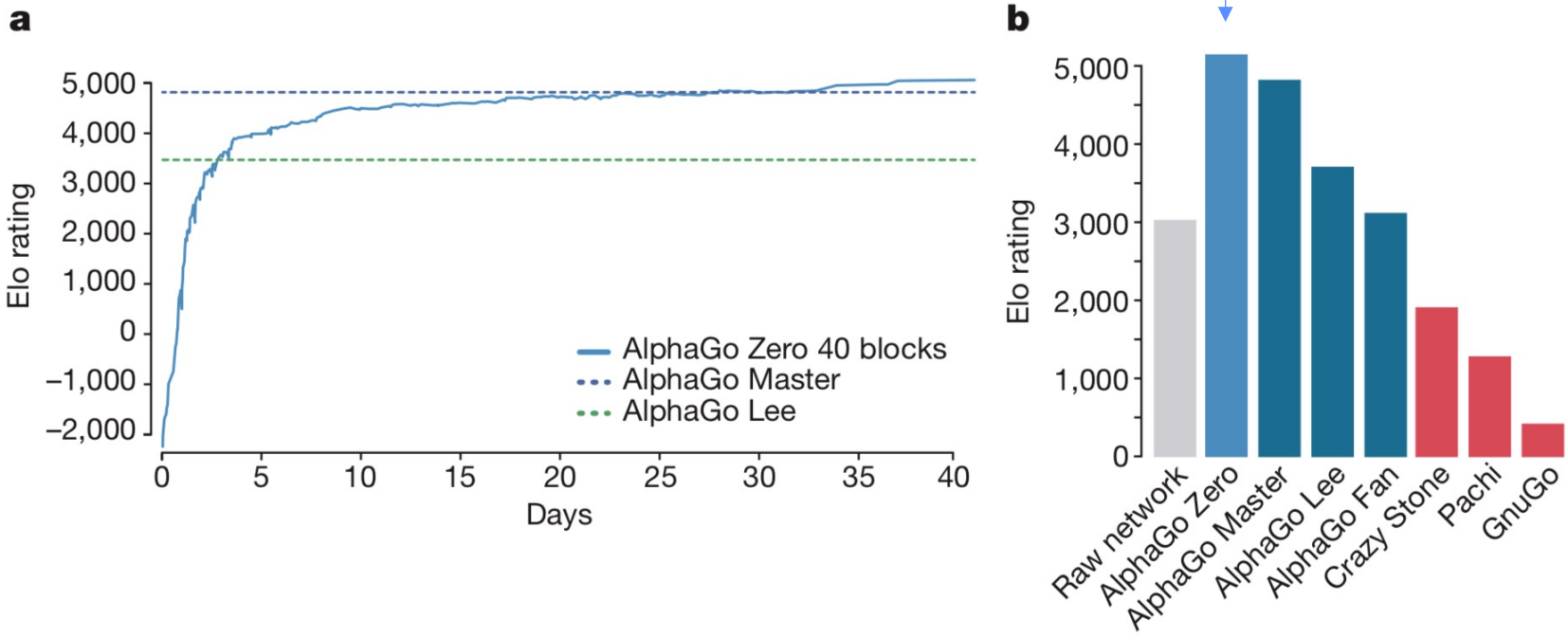
AlphaGoZero: no MC rollouts till termination

- MCTS uses always value net evaluations of leaf nodes, no rollouts!

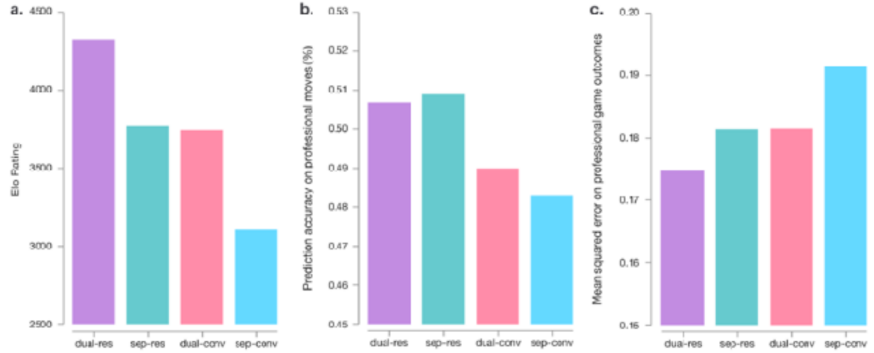


AlphaGo Zero (2017)

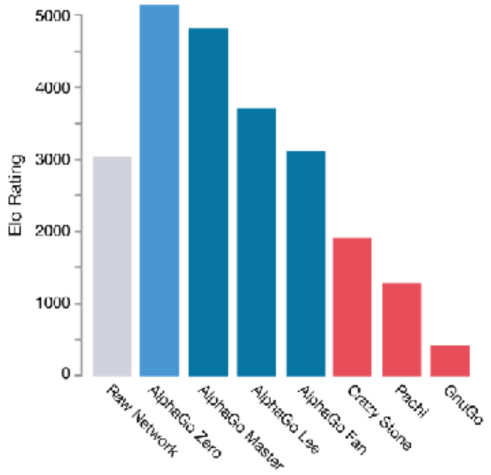
No supervised learning!



Architectures

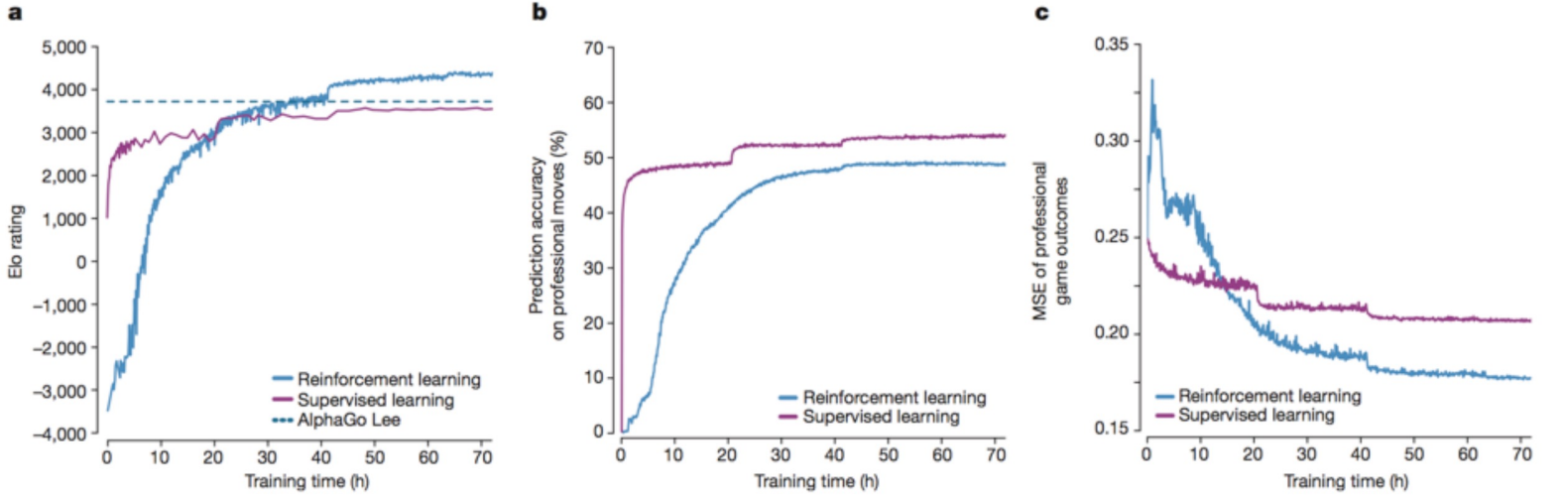


- Resnets help
- Jointly training the policy and value function using the same main feature extractor helps



- Lookahead tremendously improves the basic policy

RL vs SL


a
Elo rating
5,000
4,000
3,000
2,000
1,000
0
−1,000
−2,000
−3,000
−4,000
0 10 20 30 40 50 60 70
Training time (h)
Reinforcement learning
Supervised learning
AlphaGo Lee
b
Prediction accuracy on professional moves (%)
70
60
50
40
30
20
10
0
0 10 20 30 40 50 60 70
Training time (h)
Reinforcement learning
Supervised learning
c
MSE of professional game outcomes
0.35
0.30
0.25
0.20
0.15
0 10 20 30 40 50 60 70
Training time (h)
Reinforcement learning
Supervised learning

AlphaZero (Science, 2018)

- Removes some Go-specific features and heuristics

RESEARCH

COMPUTER SCIENCE

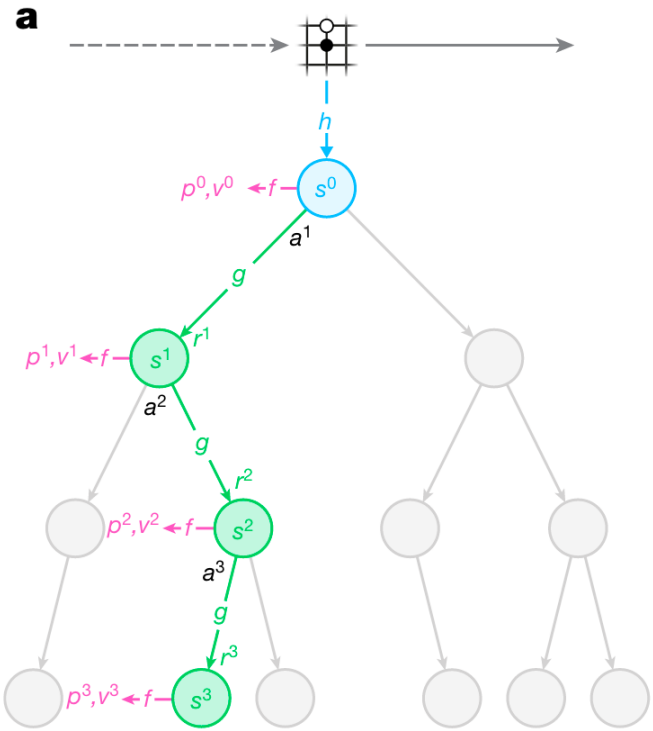
A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2*†], Thomas Hubert[1*], Julian Schrittwieser[1*], Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1†]

The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. By contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go by reinforcement learning from self-play. In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess), as well as Go.

MuZero (2020)

- Tree search on a learned model
  - Learned model g = compressed environment → faster rollouts
  - Learned model predicts next state & reward



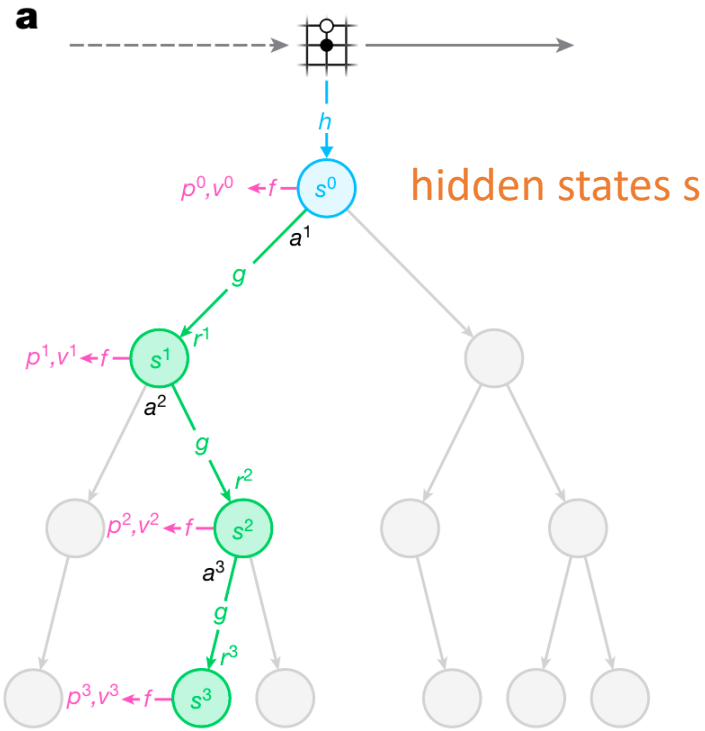
MCTS on learned model

J. Schrittwieser et al., "Mastering atari, go, chess and shogi by planning with a learned model," Nature, vol. 588, no. 7839, pp. 604–609, 2020.

MuZero

- State embedding h enables tree search beyond board games (e.g., Atari)
- Prediction function f predicts policy and value function

- Details:
  - g,h,f are jointly trained

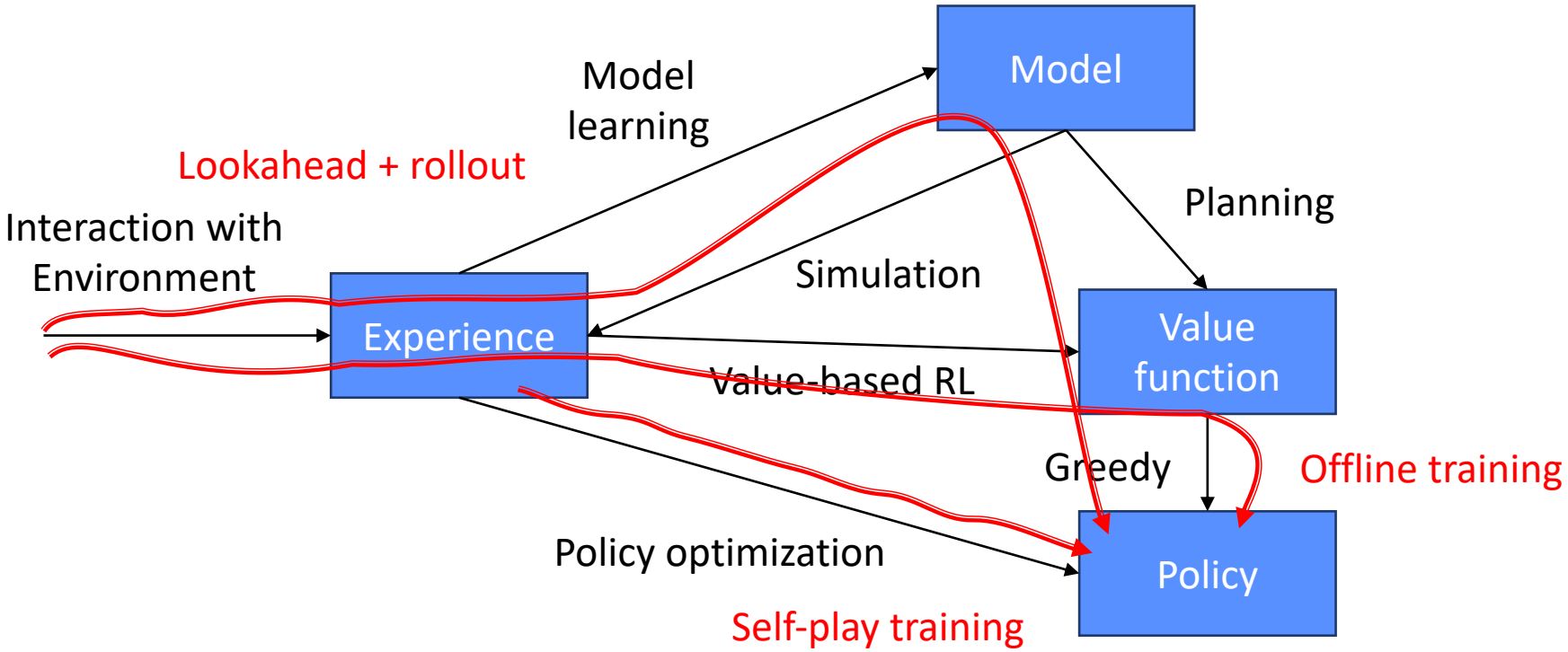


J. Schrittwieser et al., "Mastering atari, go, chess and shogi by planning with a learned model," Nature, vol. 588, no. 7839, pp. 604–609, 2020.

A note on computation cost

- 1 TPU ≈ 5-30 GPUs
- AlphaGo computation for real-time game play
  - 50 TPUs on Google Cloud
  - Searches ~50 moves deep
  - ~100,000 positions per second
- MuZero training cost: ≈220 GPU-years
  - (3.8 GPU-years per Atari game) x (57 games)
  - “For each board game, we used 16 TPUs for training and 1,000 TPUs for self-play. For each game in Atari, in the 20 billion frame setting we used 8 TPUs for training and 32 TPUs for self-play.”

Learning-guided MCTS



Adapted from Katerina Fragkiadaki (CMU)

Upcoming lecture: learning-guided optimization

Intuition: leverage **known structure** for aspects of problems that are well-modeled, and leverage **learning** to guide aspects that are not.

Examples:

Vehicle routing problems (VRP) [1]

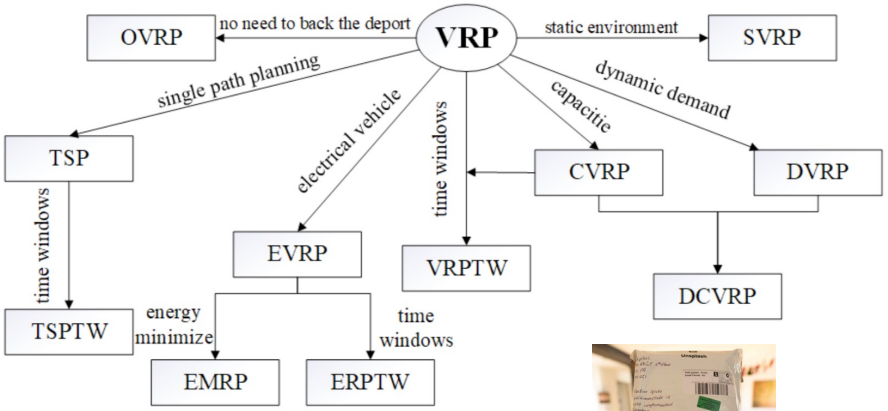
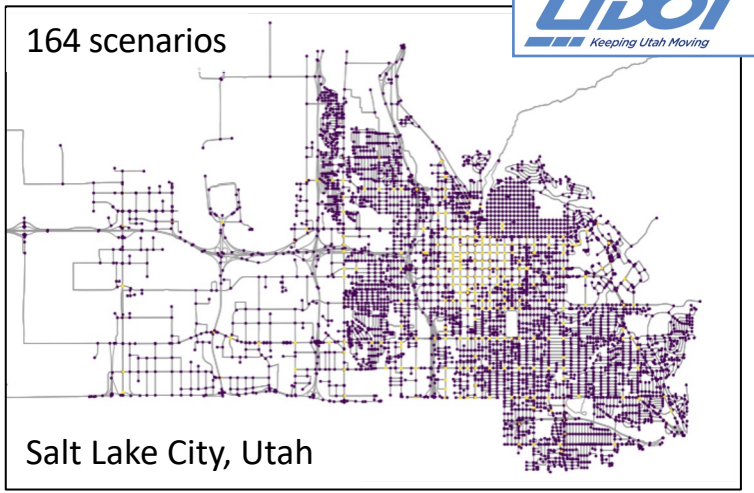


Figure courtesy Li, et al. arXiv 2107.07076, 2021.

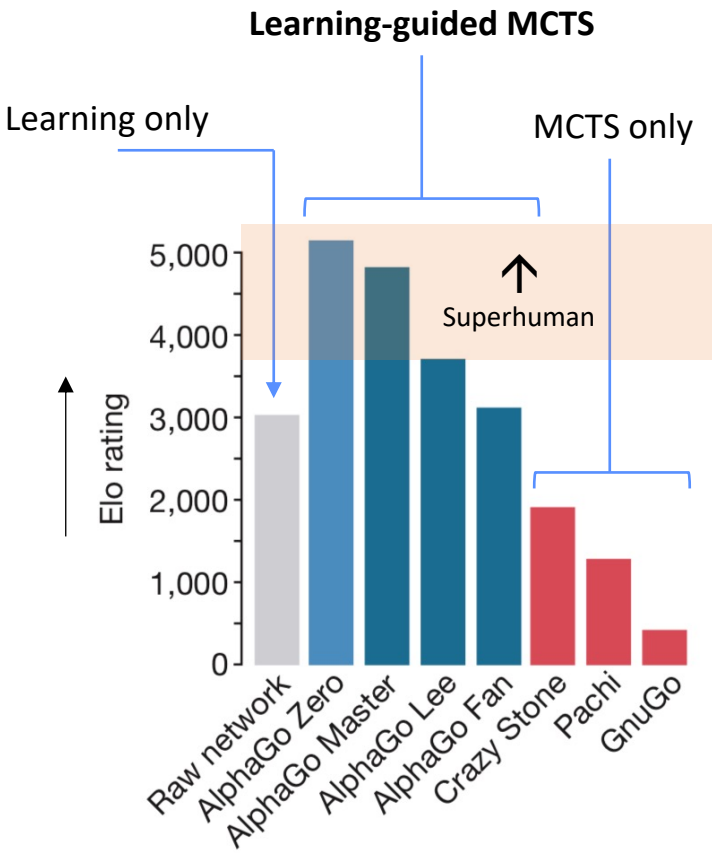
60 years of study! [2]

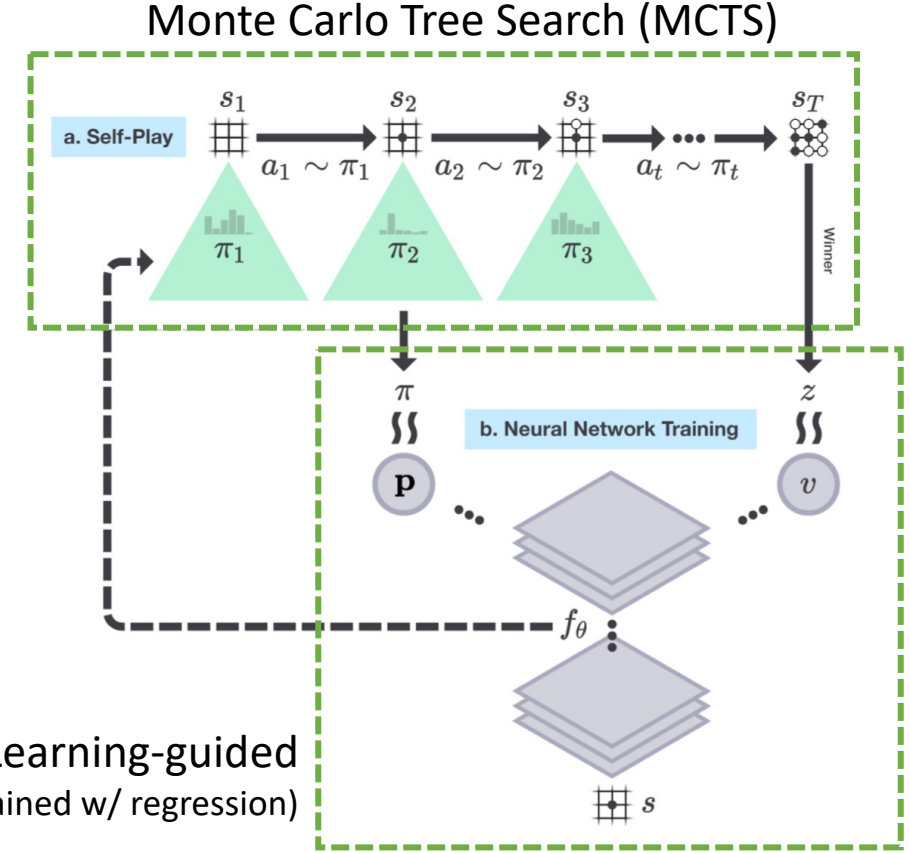
Signalized intersections [3]



[1] Li, et al. **An overview and experimental study of learning-based optimization algorithms for the vehicle routing problem**. IEEE/CAA Journal of Automatica Sinica, 2022.
[2] Laporte. **Fifty Years of Vehicle Routing**. Transportation Science, 2009.
[3] Qu*, Valiveru*, Tang, Jayawardana, Freydt, Wu. **What is a Typical Signalized Intersection in a City? A Pipeline for Intersection Data Imputation from OpenStreetMap**. TRB, 2023.

Example: AlphaGo Zero = Learning-guided MCTS



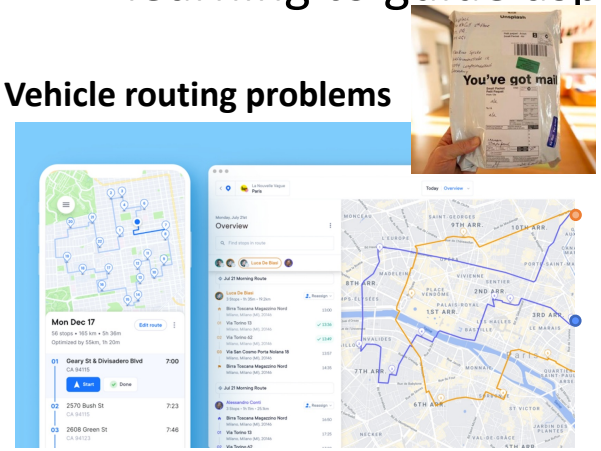


D. Silver *et al.*, **Mastering the game of go without human knowledge**, *Nature*, 2017.

Learning-guided optimization

- Tailor the optimization method to YOUR problem.
  - Example: Learning-guided MCTS (AlphaGo)
- *Intuition:* leverage known solvers for aspects of problems that are well-modeled, and leverage learning to guide aspects that are not.

Vehicle routing problems

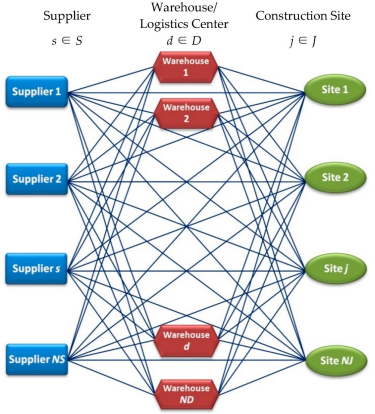


NeurIPS 2021 Spotlight

Warehouse automation

In review

Mixed integer programming (MIP)



NeurIPS 2023