Deep Learning Approaches to RL

Sham Kakade

Progress of RL in Practice

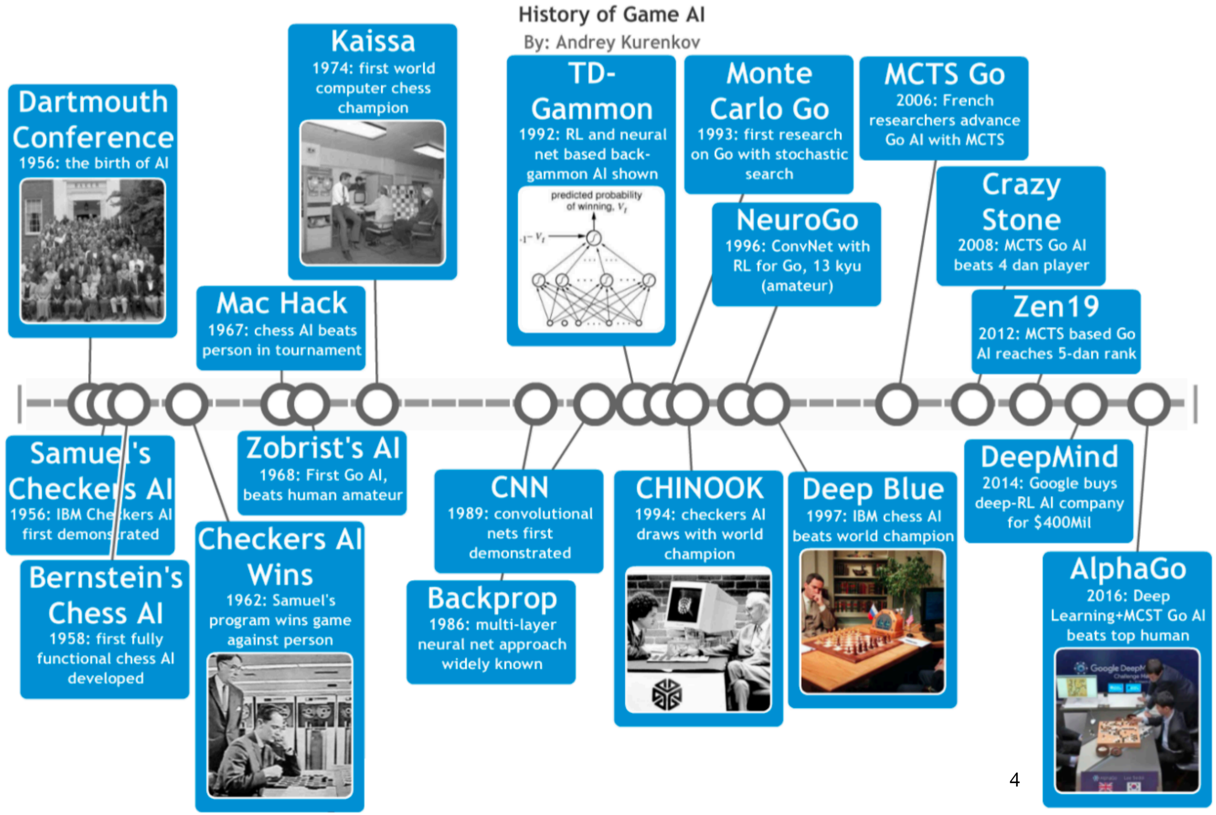
[AlphaZero, Silver et.al, 17]

[OpenAI Five, 18]

Outline

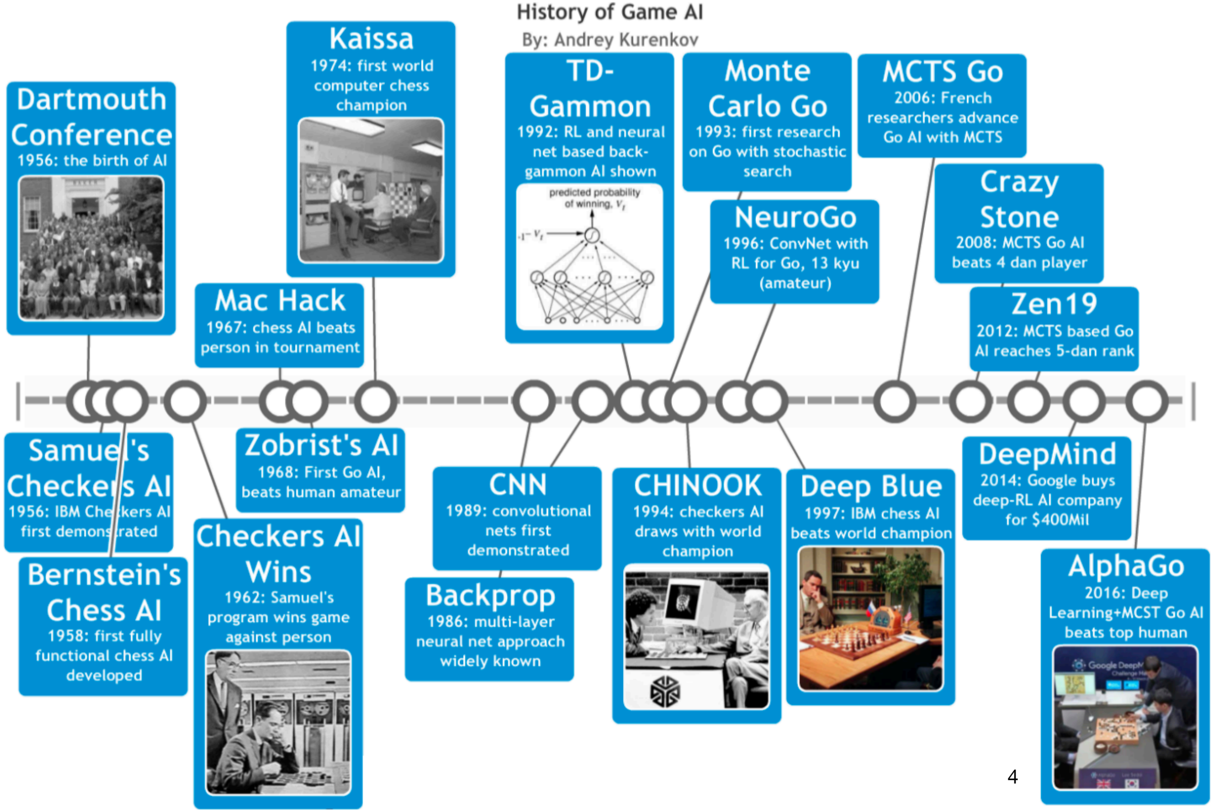
1. **RL+Search:** Self Play
    1.1. MCTS
    1.2. AlphaZero/Muzero
2. **Direct Policy Optimization**
    2.1. Conservative Policy Iteration/TRPO/PPO
3. **RL for "Alignment"**
    3.1.RLHF & Constitutional AI
4. **RL for Supply Chain**
    4.1. RL in the "real world"

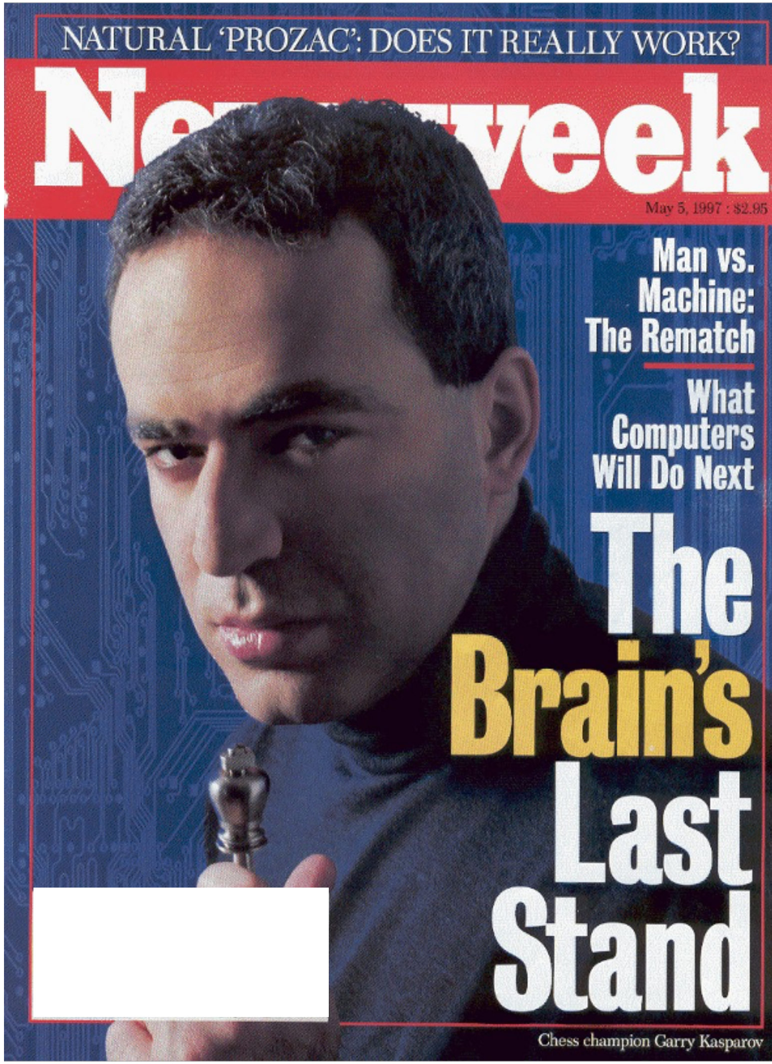
Fascination with AI and Games…



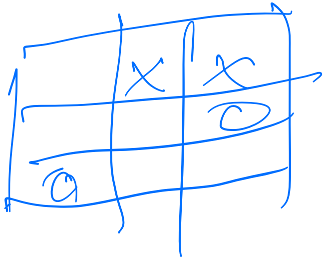
Fascination with AI and Games…

- DeepBlue v. Kasparov (1997)
  - winning in chess wasn't a good indicator of "progress in AI"





Game Trees



- In principle, one could work out the optimal strategy for any zero-sum game with lookahead.
- Note that, even with exact computation, we don't necessarily have to expand all nodes.

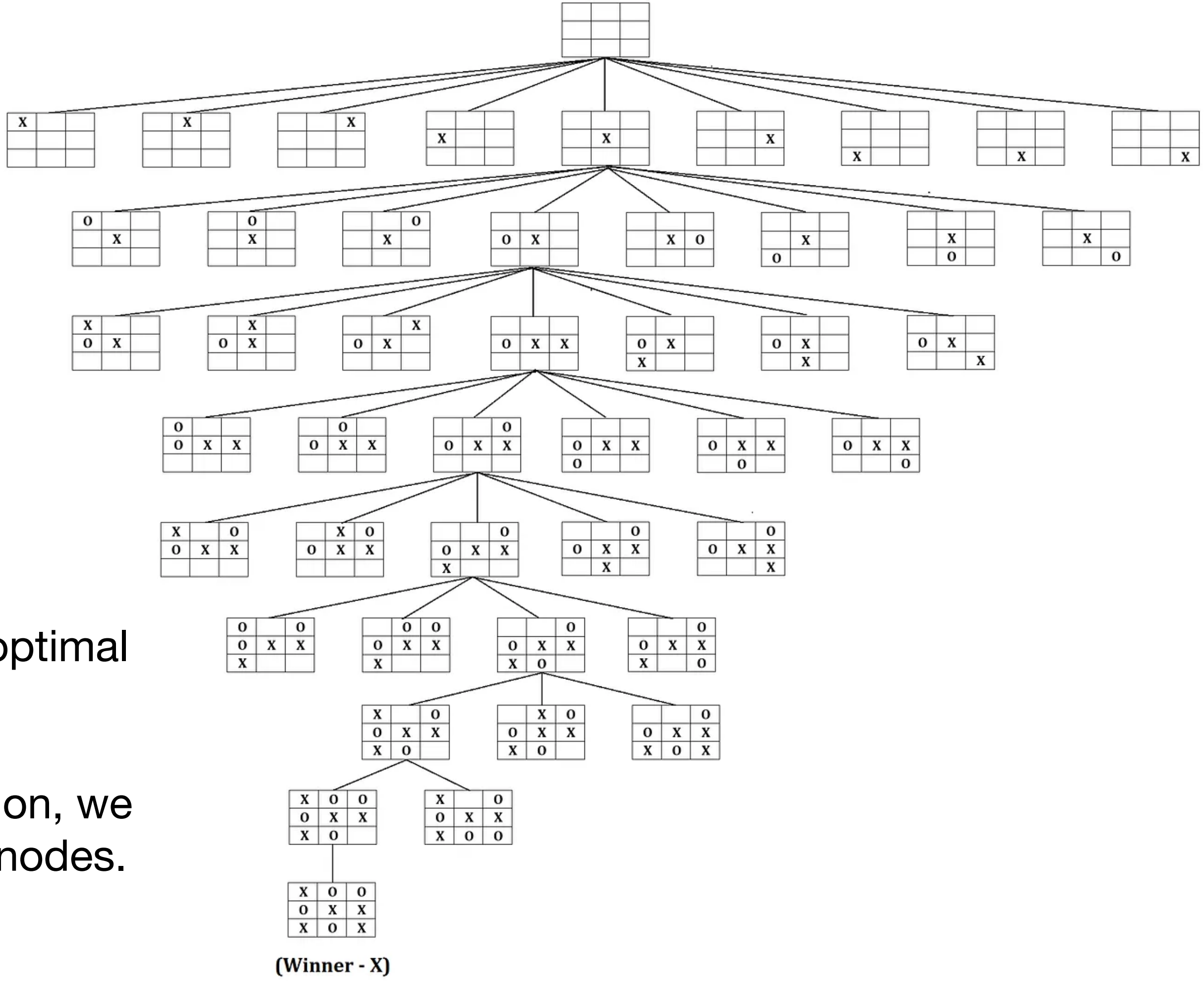
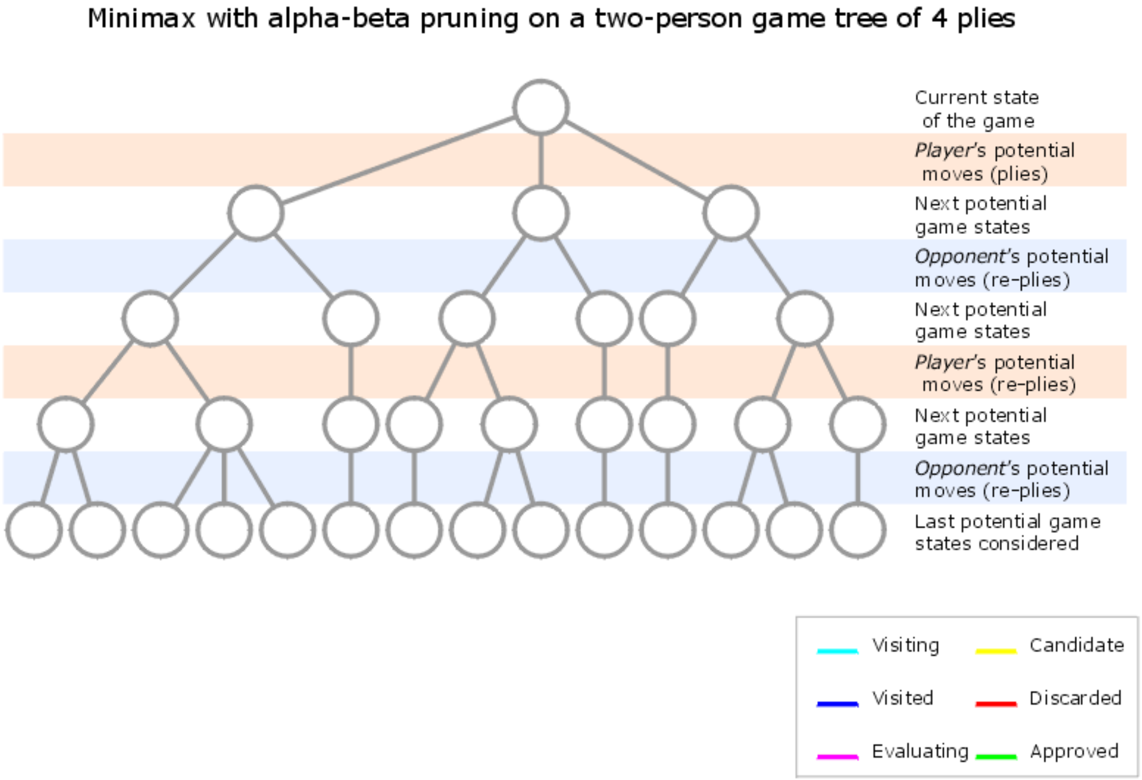
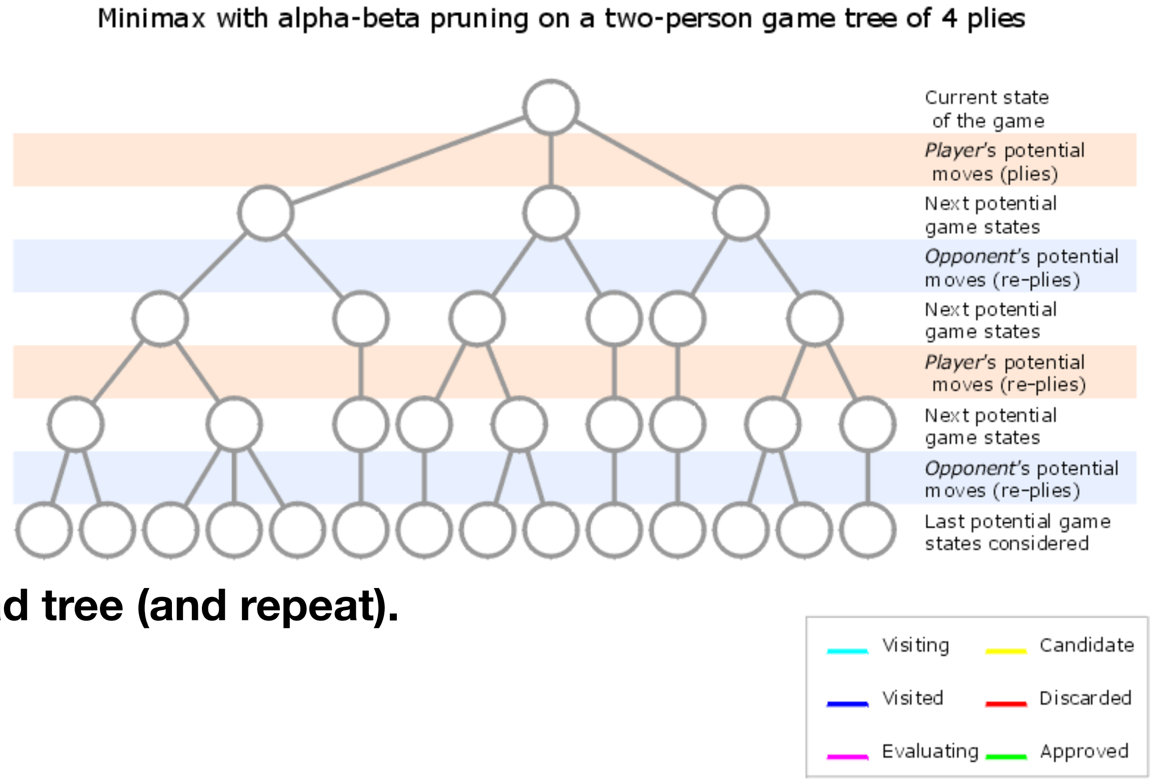


Figure not fully expanded.

AlphaBeta Search

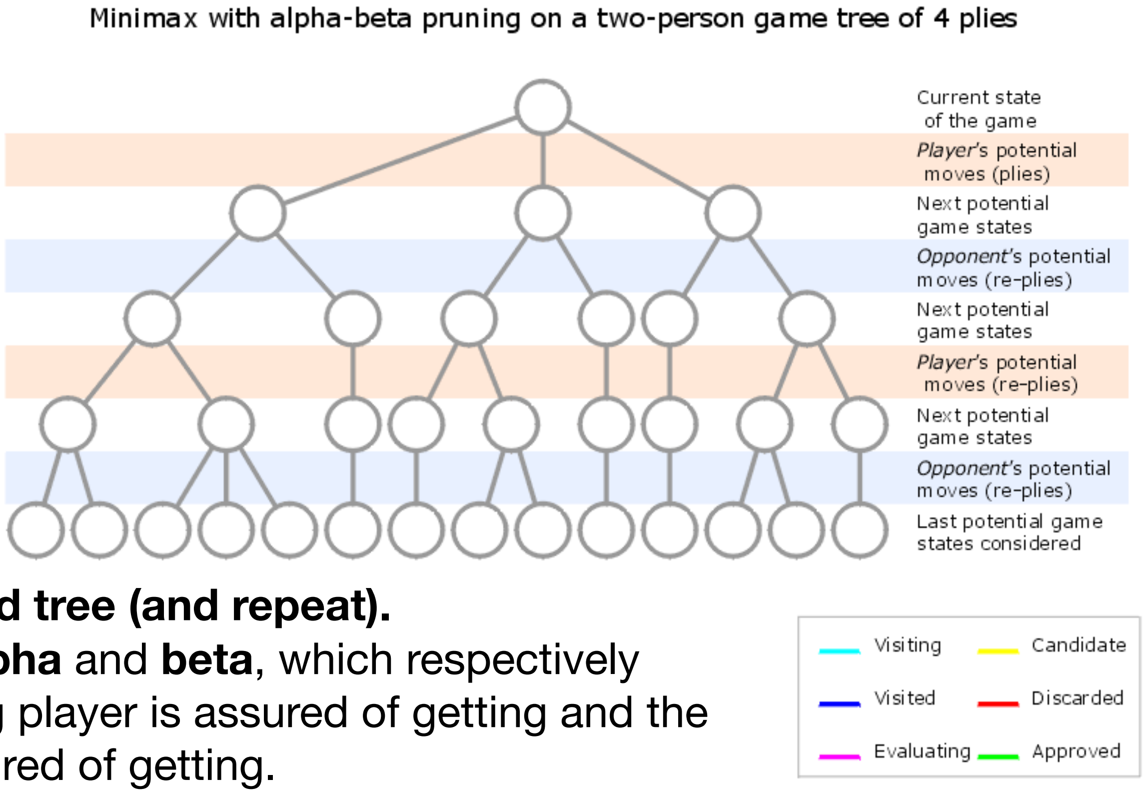


AlphaBeta Search



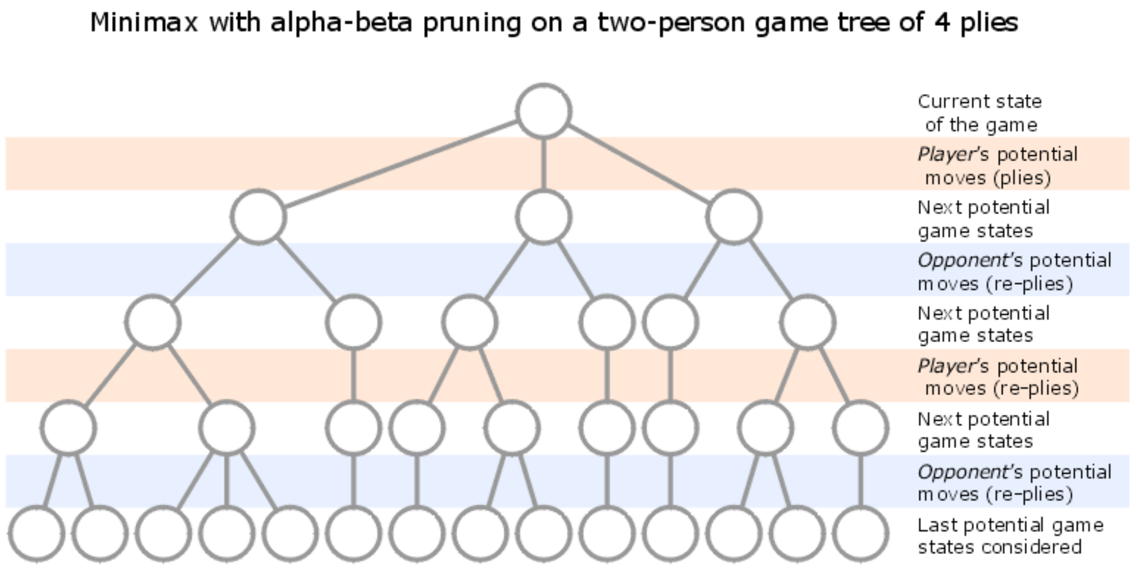
- **For every move, we build a lookahead tree (and repeat).**

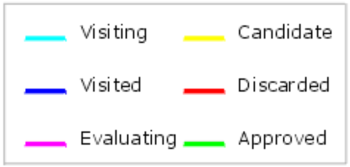
AlphaBeta Search



- **For every move, we build a lookahead tree (and repeat).**
- The algorithm maintains two values, **alpha** and **beta**, which respectively represent the score that the maximizing player is assured of getting and the score that the minimizing player is assured of getting.
  - Assume opponents will always try to do “best responses”

AlphaBeta Search





- **For every move, we build a lookahead tree (and repeat).**
- The algorithm maintains two values, **alpha** and **beta**, which respectively represent the score that the maximizing player is assured of getting and the score that the minimizing player is assured of getting.
  - Assume opponents will always try to do “best responses”
- Before every move, try to figure out a good move by lookahead.
  - Need a heuristic for how to choose actions (i.e. which branches to search)
  - **Try to prune away as may branches as we can.**

Stockfish 15.1

Strong open source chess engine

Download Stockfish

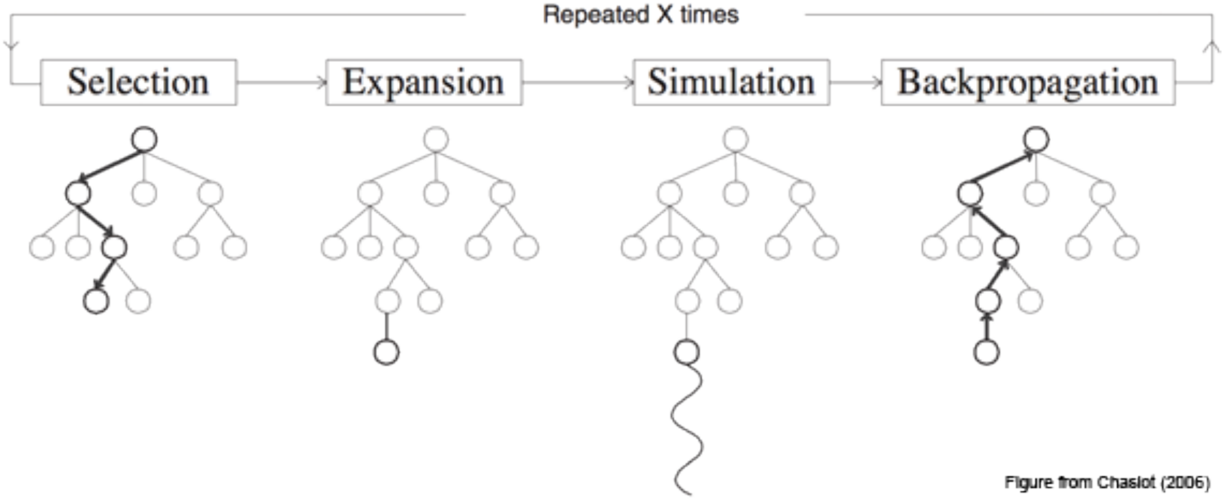
Latest from the blog

2022-12-04: Stockfish 15.1

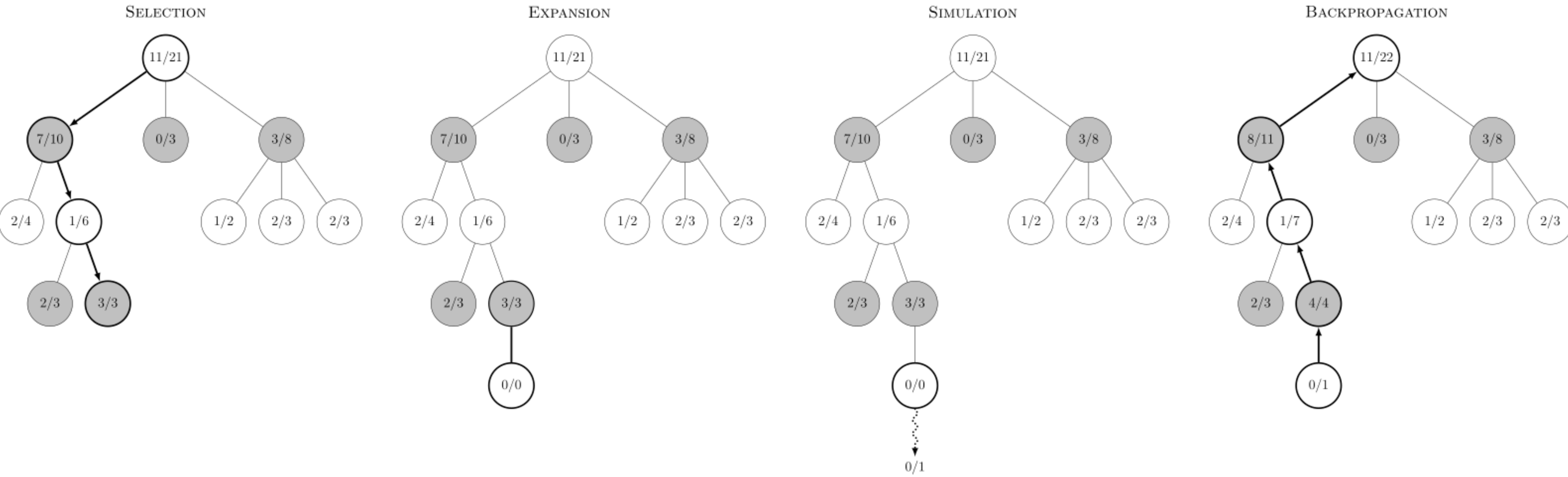
2022-11-18: ChessBase GmbH and the Stockfish team reach an agreement and end their legal dispute

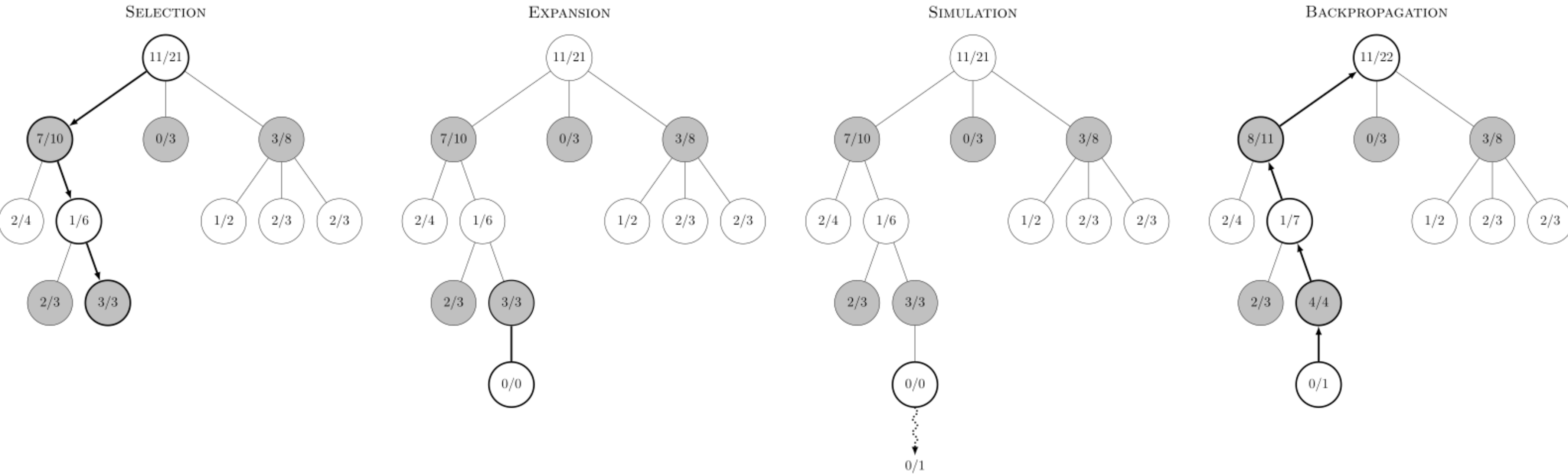
2022-06-22: Public court hearing soon!

MCTS:
Monte Carlo Tree Search

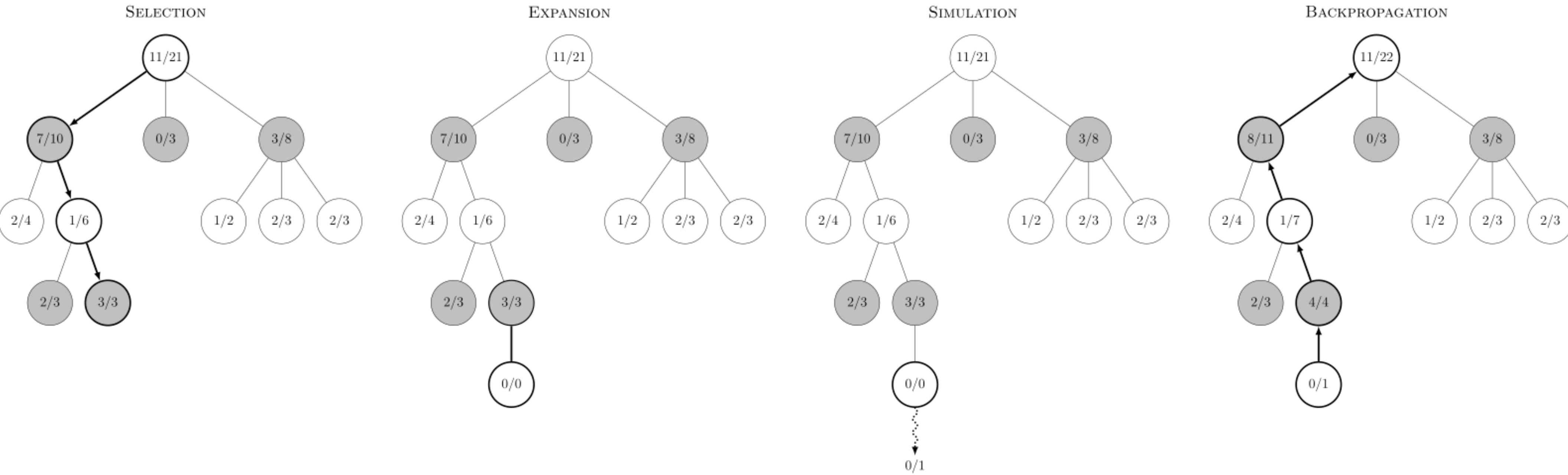


- AlphaBeta pessimistic approach may not lead to effective heuristics.
- **MCTS: for every move, we build a lookahead tree; take an action; and repeat.**
  - We are some node “s”.
  - We use a heuristic to estimate the “value” of taking action “a” at any node “s” (We don’t directly compute minmax values).
- Four steps to the algorithm.

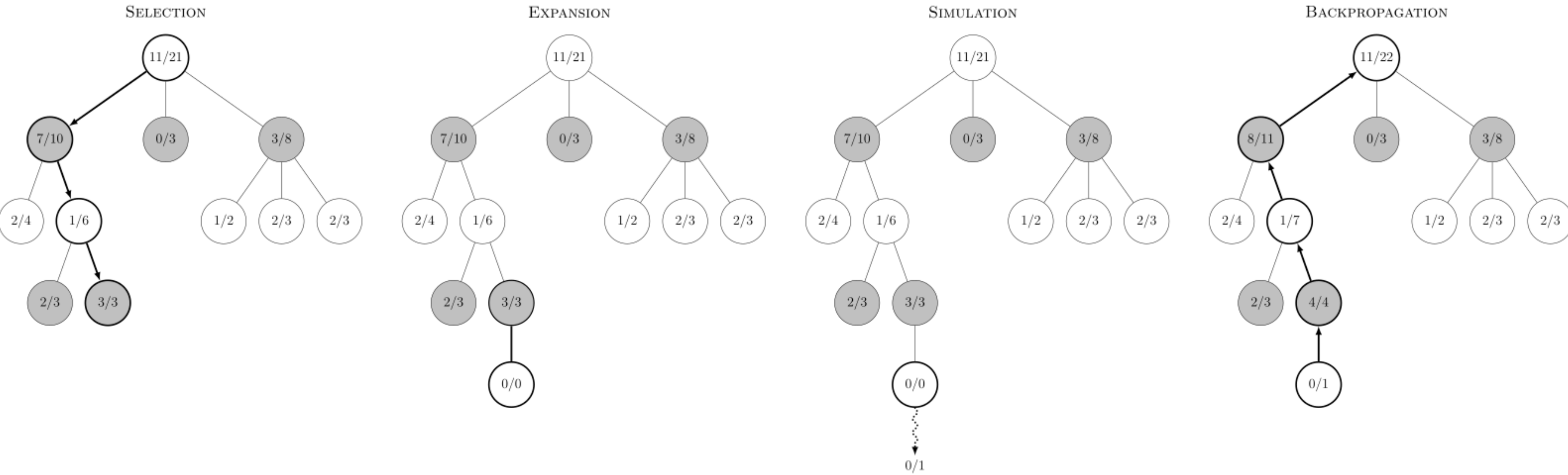

Selection
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
Expansion
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
Simulation
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1
Backpropagation
11/22
8/11
0/3
3/8
2/4
1/7
1/2
2/3
2/3
2/3
4/4
0/1



- **Selection:** Start from "root R" (current game) and select successive child nodes until a "leaf node L" (a node that has an "unvisited" child) is reached.

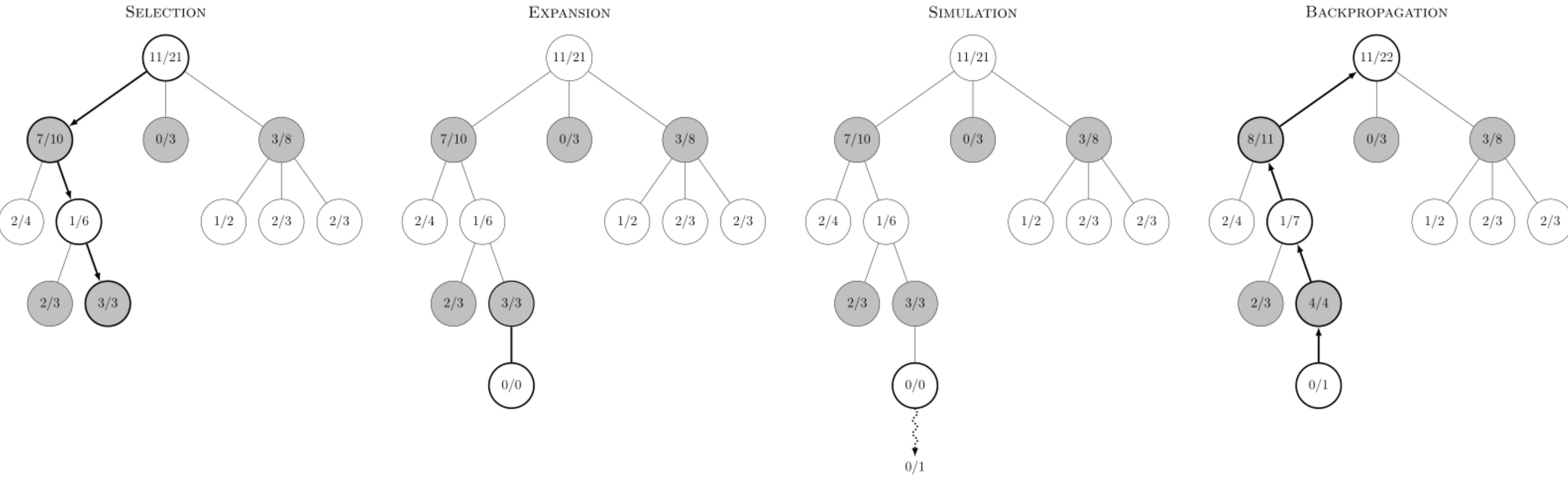


- **Selection:** Start from “root R” (current game) and select successive child nodes until a “leaf node L” (a node that has an “unvisited” child) is reached.
  - A leaf is any node that has a potential child from which no simulation (rollout) has yet been initiated (i.e. we haven’t tried all the actions at L).

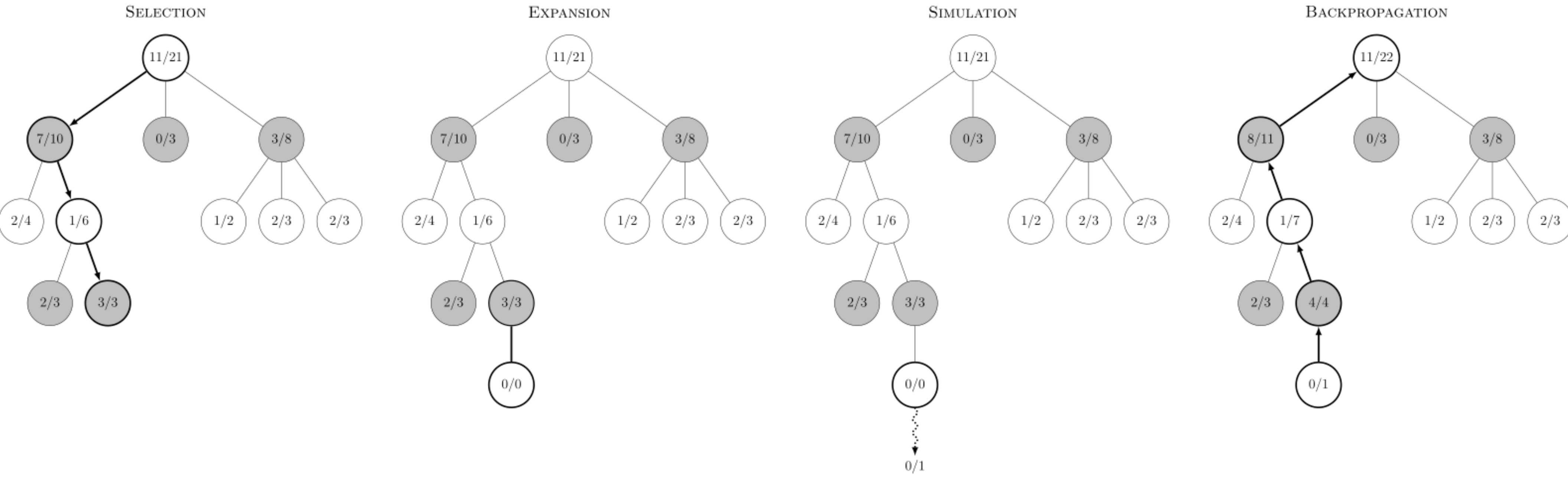


- **Selection:** Start from "root R" (current game) and select successive child nodes until a "leaf node L" (a node that has an "unvisited" child) is reached.
  - A leaf is any node that has a potential child from which no simulation (rollout) has yet been initiated (i.e. we haven't tried all the actions at L).
  - At state s, choose action a leading to $s' = NextState(s, a)$ which maximizes:

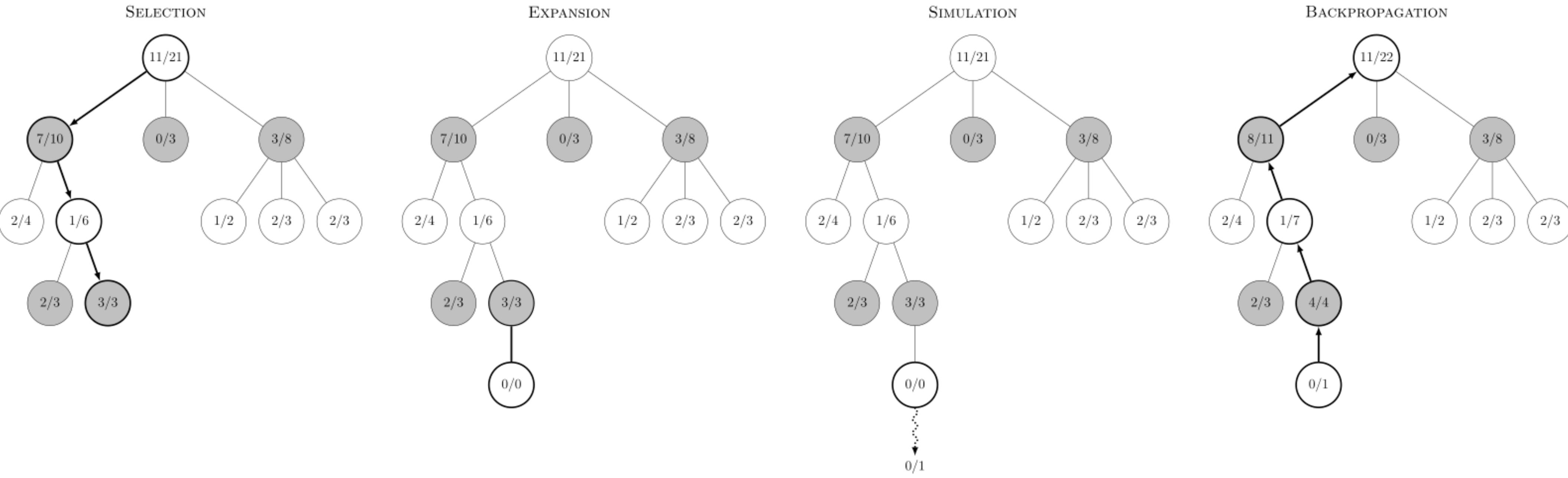
$$\text{UCB score}(a) = \frac{\text{\#wins at s'}}{\text{\#visits to s'}} + C \times \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to s'}}}$$

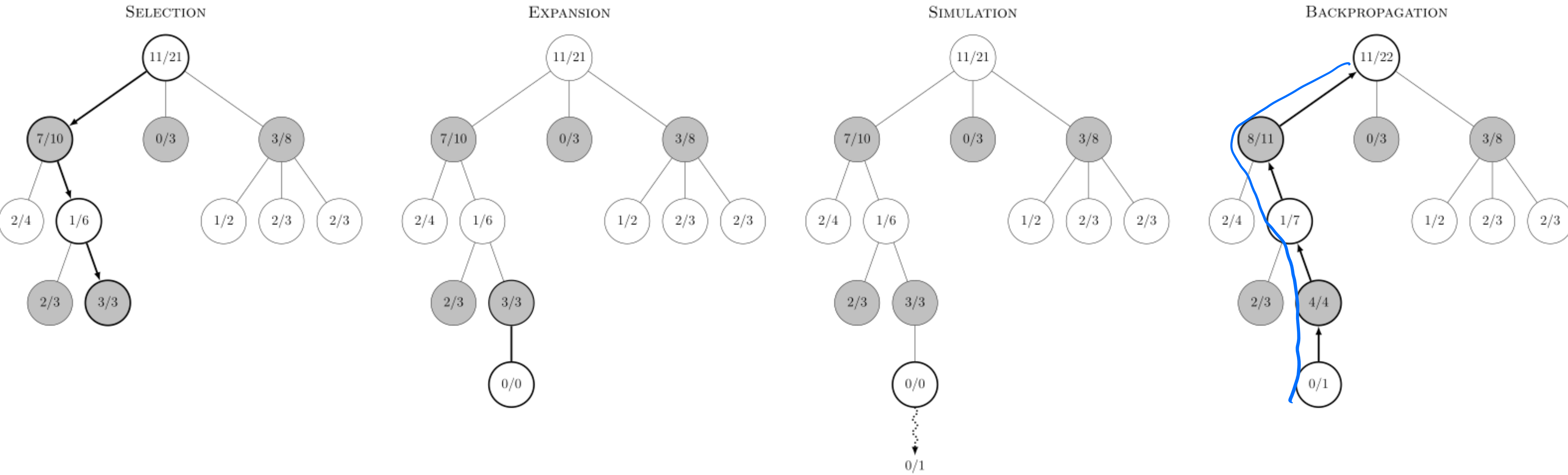


- **Expansion:** Stop Selection at a leaf node L. Unless leaf L ends the game decisively (e.g. win/loss/draw) for either player, create a child node (a new node) and choose this node C.
    - This step just creates a new node.

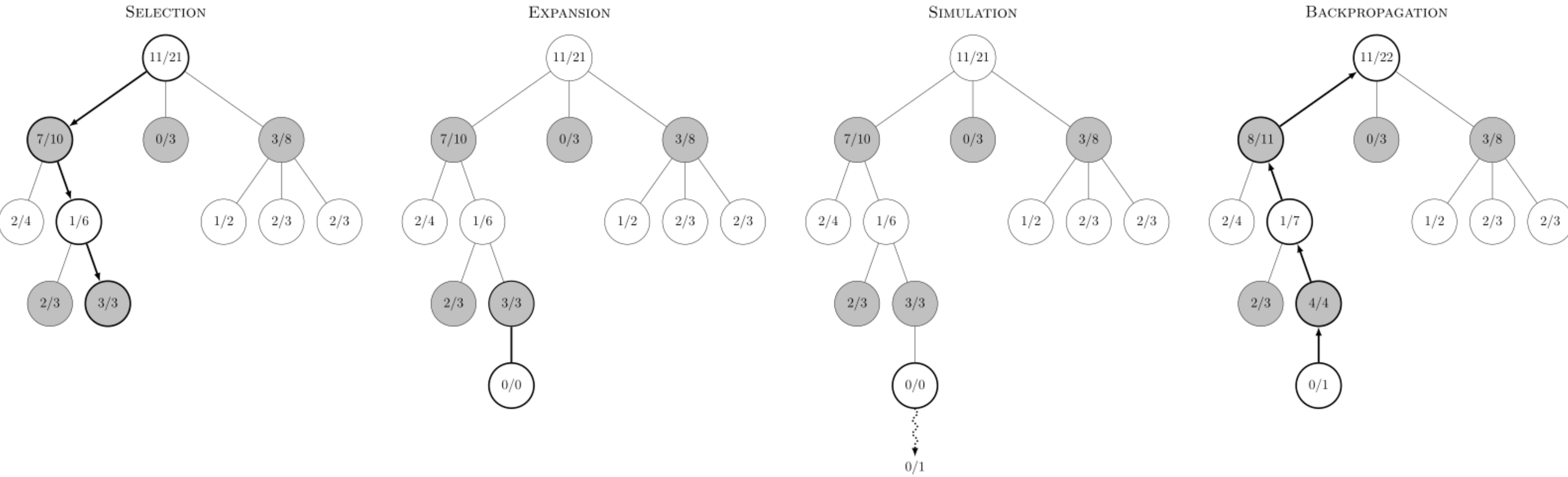


- **Simulation:** Complete a "playout/rollout" from this new node C till the game ends.
    - Simplest approach: choose uniform at random moves until the game ends.


Selection
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
Expansion
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
Simulation
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1
Backpropagation
11/22
8/11
0/3
3/8
2/4
1/7
1/2
2/3
2/3
2/3
4/4
0/1



- **"Backpropagation" (the update step):** Use the result of the rollout to update information in the nodes on the path from the root R to C.
  - Update the total # of wins and # visits on this path.
    - # wins at at a node is the number of previous wins from *any* sim from this node.



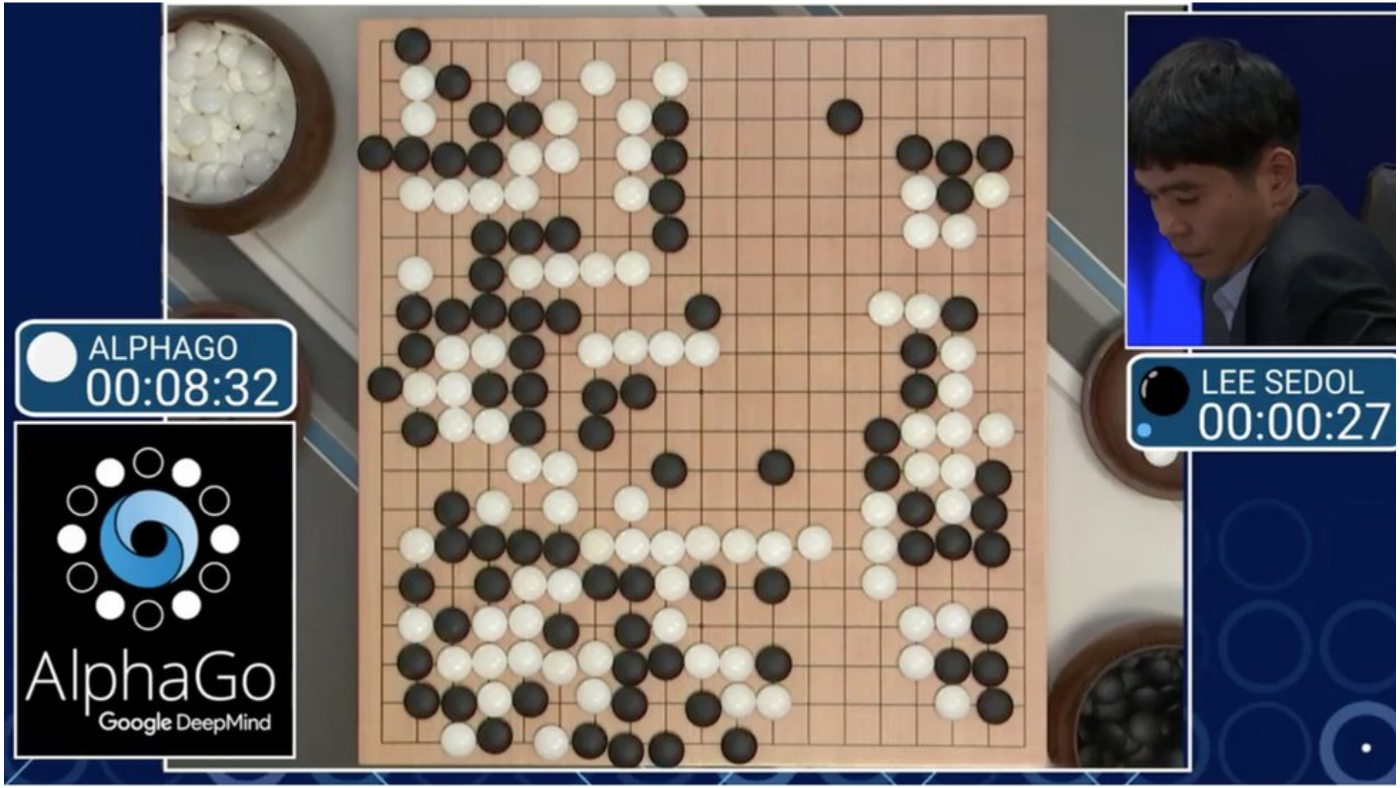
- **“Backpropagation” (the update step):** Use the result of the rollout to update information in the nodes on the path from the root R to C.
  - Update the total # of wins and # visits on this path.
    - # wins at at a node is the number of previous wins from *any* sim from this node.

- **Repeat all steps N times**, then **select “best” action at the root node R** (the game state).

AlphaGo

AlphaGo versus Lee Sedol
4–1

Seoul, South Korea, 9–15 March 2016	
Game one	AlphaGo W+R
Game two	AlphaGo B+R
Game three	AlphaGo W+R
Game four	Lee Sedol W+R
Game five	AlphaGo W+R



- Lots of moving parts:
  - **Imitation Learning:** first, the algo estimates the values from historical games.
  - It then uses an **MCTS-stye lookahead** with **learned value functions**.
- **AlphaZero** (2017) is was a simpler more successful approach.

AlphaZero

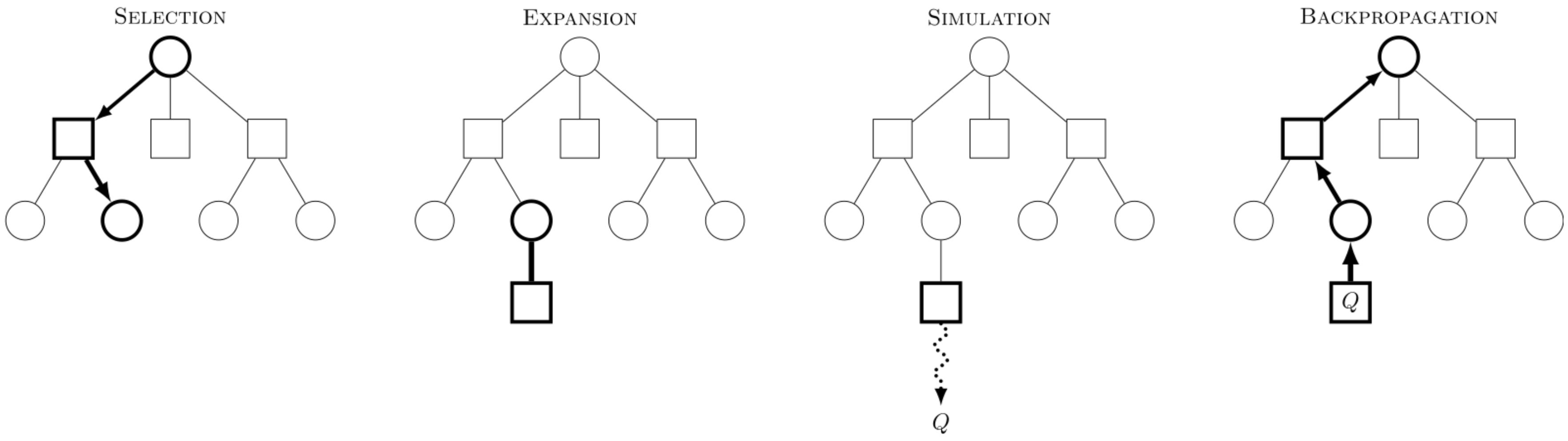
- AlphaZero: MCTS + DeepLearning
  - There is a value network and policy network:
    - a **value network** estimating for the state of the board $v_\theta(s)$, for a player
    - A **policy network** $\overrightarrow{p}_\theta(a \mid s)$ that is a probability vector over all possible actions.

- These are fit with training data (s_t, a_t, R_t) under the loss function:

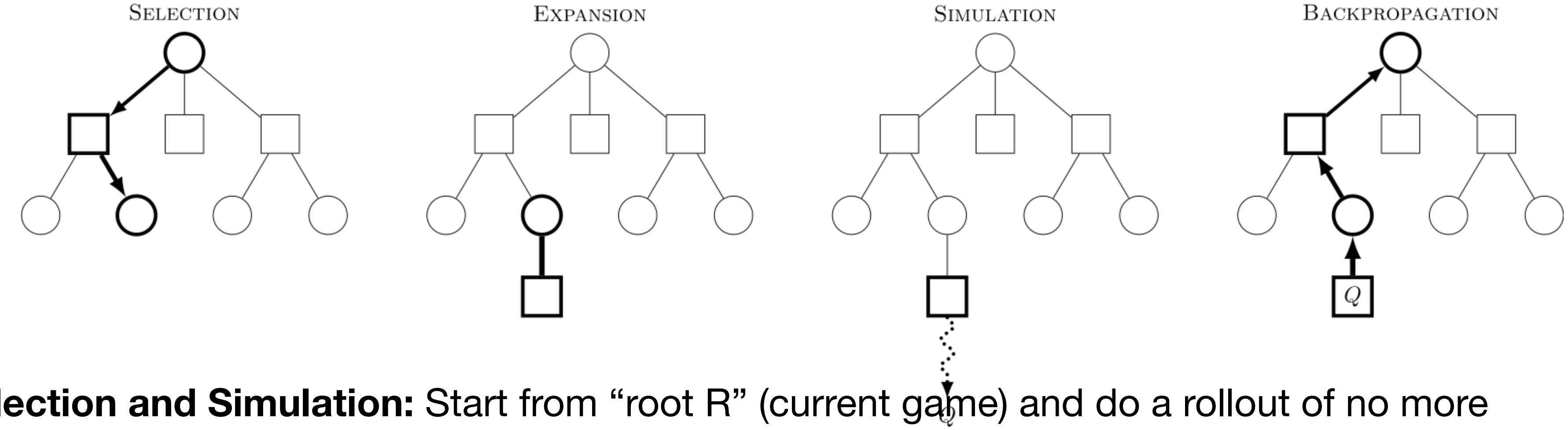
$$Loss(\theta) = \sum_t (v_\theta(s_t) - R_t)^2 - \log p_\theta(a_t \mid s_t)$$

  - We'll come back to how we get this data, but let's see how we select actions.

AlphaZero

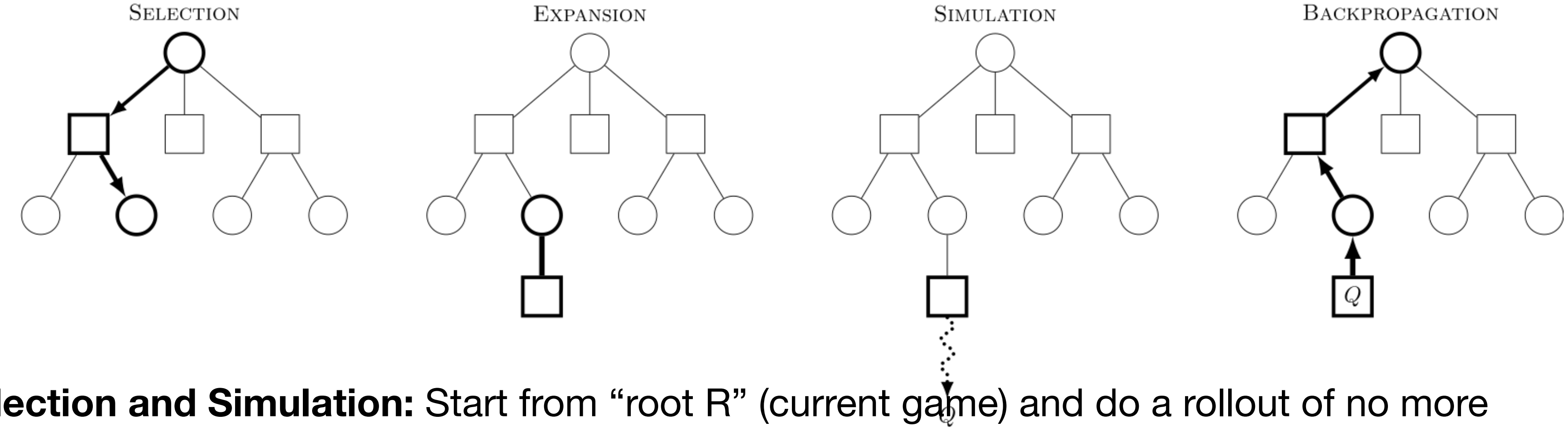


AlphaZero



- **Selection and Simulation:** Start from “root R” (current game) and do a rollout of no more than K steps.

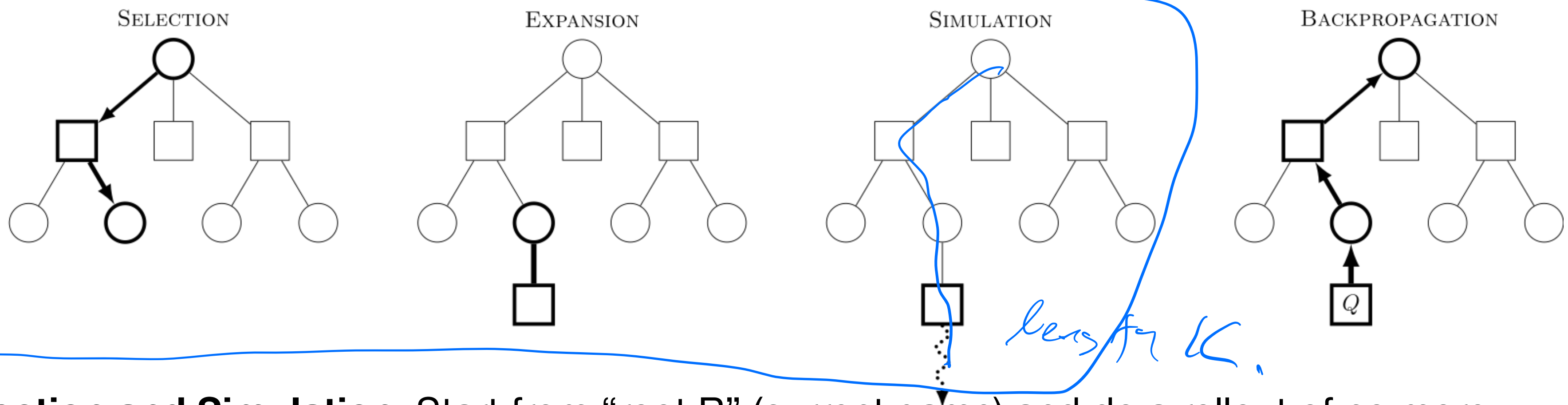
AlphaZero



- **Selection and Simulation:** Start from "root R" (current game) and do a rollout of no more than K steps.
  - At state s, choose action a leading to $s' = NextState(s, a)$ which maximizes:

$$\text{UCB score}(a) = \text{AvValue}(s') + C \cdot p_\theta(a \mid s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to s'}}}$$

AlphaZero



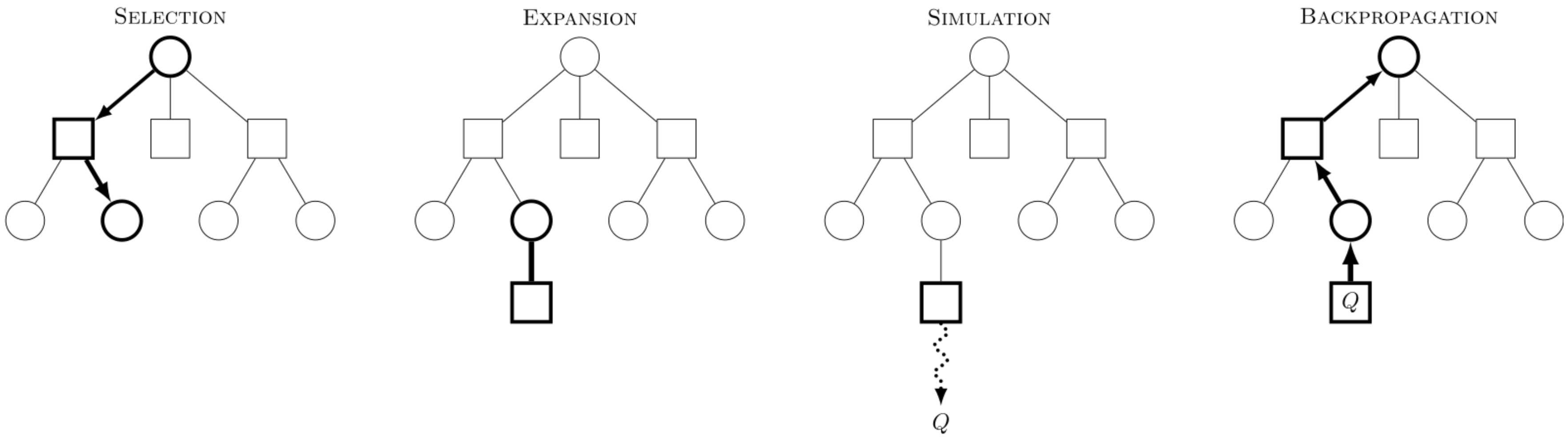
- **Selection and Simulation:** Start from "root R" (current game) and do a rollout of no more than K steps.
  - At state s, choose action a leading to $s' = NextState(s, a)$ which maximizes:

$$\text{UCB score}(a) = \text{AvValue}(s') + C \cdot p_\theta(a \mid s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to s'}}}$$

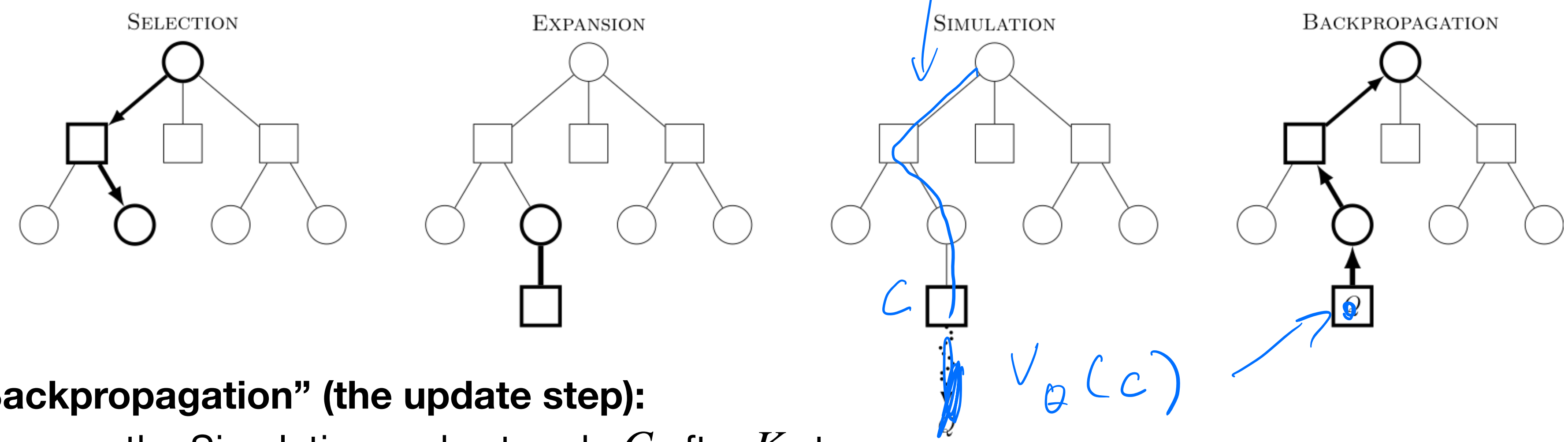
σ soft

  - We'll specify AverageValue(s') soon.
    - in MCTS, this average was $\dfrac{\text{\#wins at s'}}{\text{\#visits to s'}}$

AlphaZero



AlphaZero

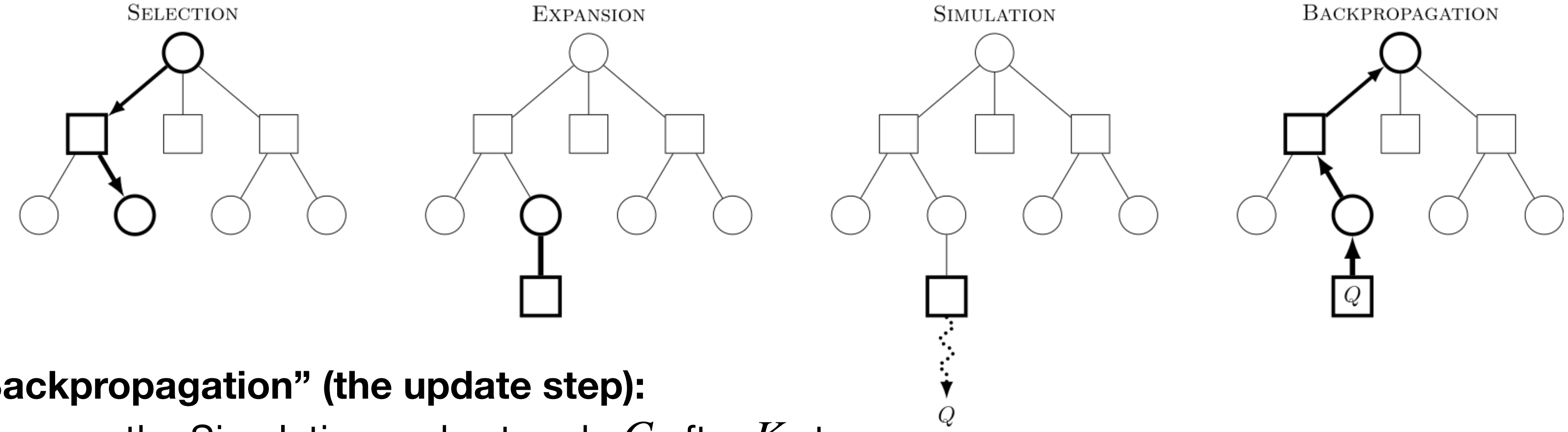


- **“Backpropagation” (the update step):**
  Suppose the Simulation ends at node C after K steps.
  With the rollout result, update AvValue(s) on all the nodes s in the path from the root R to terminal node C:

$$AvValue(s) \leftarrow \frac{N(s)}{N(s)+1} AvValue(s) + \frac{1)}{N(s)+1} v_\theta(C)$$

$$N(s) \leftarrow N(s) + 1$$

AlphaZero



- **“Backpropagation” (the update step):**
  Suppose the Simulation ends at node C after K steps.
  With the rollout result, update AvValue(s) on all the nodes s in the path from the root R to terminal node C:

$$AvValue(s) \leftarrow \frac{N(s)}{N(s)+1} AvValue(s) + \frac{1)}{N(s)+1} v_\theta(C)$$

$$N(s) \leftarrow N(s) + 1$$

- **Repeat all steps N times**, then **select “best” action at the root node R** (the game state).

AlphaZero: Learning

- Collect training data (s_t, a_t, R_t) from self-play. Then fit:

$$Loss(\theta) = \sum_t (v_\theta(s_t) - R_t)^2 - \log p_\theta(a_t \mid s_t)$$

AlphaZero: Learning

- Collect training data (s_t, a_t, R_t) from self-play. Then fit:

$$Loss(\theta) = \sum_t (v_\theta(s_t) - R_t)^2 - \log p_\theta(a_t \mid s_t)$$

AlphaZero was trained solely via self-play, using 5,000 first-generation TPUs to generate the games and 64 second-generation TPUs to train the neural networks. In parallel, the in-training AlphaZero was periodically matched against its benchmark (Stockfish, elmo, or AlphaGo Zero) in

Comparing Monte Carlo tree search searches, AlphaZero searches just 80,000 positions per second in chess and 40,000 in shogi, compared to 70 million for Stockfish and 35 million for elmo. AlphaZero compensates for the lower number of evaluations by using its deep neural network to

Chess [edit]

In AlphaZero's chess match against Stockfish 8 (2016 TCEC world champion), each program was given one minute per move. Stockfish was allocated 64 threads and a hash size of 1 GB,[1] a setting that Stockfish's Tord Romstad later criticized as suboptimal.[7][note 1] AlphaZero was trained on chess for a total of nine hours before the match. During the match, AlphaZero ran on a single machine with four application-specific TPUs. In 100 games from the normal starting position, AlphaZero won 25 games as White, won 3 as Black, and drew the remaining 72.[8] In a series of twelve, 100-game matches (of unspecified time or resource constraints) against Stockfish starting from the 12 most popular human openings, AlphaZero won 290, drew 886 and lost 24.[1]

Shogi [edit]

AlphaZero was trained on shogi for a total of two hours before the tournament. In 100 shogi games against elmo (World Computer Shogi Championship 27 summer 2017 tournament version with YaneuraOu 4.73 search), AlphaZero won 90 times, lost 8 times and drew twice.[8] As in the chess games, each program got one minute per move, and elmo was given 64 threads and a hash size of 1 GB.[1]

Go [edit]

After 34 hours of self-learning of Go and against AlphaGo Zero, AlphaZero won 60 games and lost 40.[1][8]

Comparing Monte Carlo tree search searches, AlphaZero searches just 80,000 positions per second in chess and 40,000 in shogi, compared to 70 million for Stockfish and 35 million for elmo. AlphaZero compensates for the lower number of evaluations by using its deep neural network to

Chess [edit]

In AlphaZero's chess match against Stockfish 8 (2016 TCEC world champion), each program was given one minute per move. Stockfish was allocated 64 threads and a hash size of 1 GB,[1] a setting that Stockfish's Tord Romstad later criticized as suboptimal.[7][note 1] AlphaZero was trained on chess for a total of nine hours before the match. During the match, AlphaZero ran on a single machine with four application-specific TPUs. In 100 games from the normal starting position, AlphaZero won 25 games as White, won 3 as Black, and drew the remaining 72.[8] In a series of twelve, 100-game matches (of unspecified time or resource constraints) against Stockfish starting from the 12 most popular human openings, AlphaZero won 290, drew 886 and lost 24.[1]

Shogi [edit]

AlphaZero was trained on shogi for a total of two hours before the tournament. In 100 shogi games against elmo (World Computer Shogi Championship 27 summer 2017 tournament version with YaneuraOu 4.73 search), AlphaZero won 90 times, lost 8 times and drew twice.[8] As in the chess games, each program got one minute per move, and elmo was given 64 threads and a hash size of 1 GB.[1]

Go [edit]

After 34 hours of self-learning of Go and against AlphaGo Zero, AlphaZero won 60 games and lost 40.[1][8]

Cup

Year	Time Controls	Result	Ref
2018	30+10	1st	[63]
2019	30+5	2nd[note 1]	[64]
2019	30+5	2nd	[65]
2019	30+5	1st	[66]
2020	30+5	1st	[67]
2020	30+5	3rd	[68]
2020	30+5	1st	[69]
2021	30+5	1st	[70]
2021	30+5	1st	[71]
2022	30+3	1st	[72]
2023	30+3	2nd	[73]

Leela Chess Zero

Original author(s) Gian-Carlo Pascutto, Gary Linscott

Leela Chess Zero (abbreviated as **LCZero**, **lc0**) is a free, open-source, and deep neural network–based chess engine and volunteer computing project. Development has been spearheaded by programmer Gary Linscott, who is also a developer for the Stockfish chess engine. Leela Chess Zero was adapted from the Leela Zero Go engine,[1] which in turn was based on Google's AlphaGo Zero project.[2] One of the purposes of Leela Chess Zero was to verify the methods in the AlphaZero paper as applied to the game of chess.

MuZero

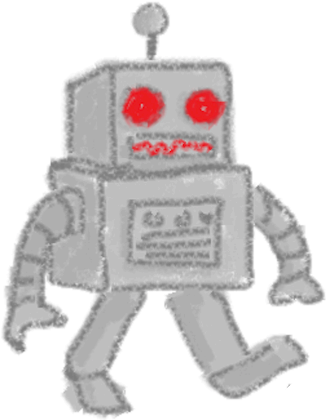
- **MuZero**
  - Basically AlphaZero but we don’t know game rules.
  - We learn the transition function as we play.

Outline

1. **RL+Search:** Self Play
    1.1. MCTS
    1.2. AlphaZero/Muzero
2. **Direct Policy Optimization**
    2.1. Conservative Policy Iteration/TRPO/PPO
3. **RL for "Alignment"**
    3.1.RLHF & Constitutional AI
4. **RL for Supply Chain**
    4.1. RL in the "real world"

Markov Decision Process

Learning Agent



Environmen

$$a \sim \pi(s)$$

Policy: determine **action** based on **state**

Multiple Steps

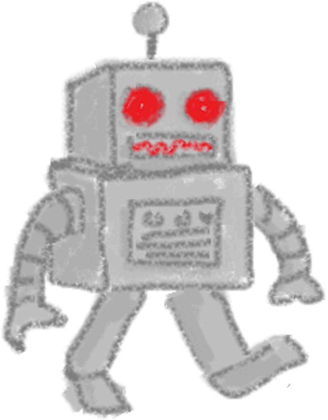
Send **reward** and **next state** from a Markovian transition dynamics

$$r(s,a), s' \sim P(\cdot \mid s,a)$$

$$s_0 \sim \mu_0, a_0 \sim \pi(s_0), r_0, s_1 \sim P(s_0, a_0), a_1 \sim \pi(s_1), r_1 \ldots$$

Markov Decision Process

Learning Agent



Environmen

$$a \sim \pi(s)$$

Policy: determine **action** based on **state**

Multiple Steps

Send **reward** and **next state** from a Markovian transition dynamics

$$r(s, a), s' \sim P(\cdot \,|\, s, a)$$

$$s_0 \sim \mu_0, a_0 \sim \pi(s_0), r_0, s_1 \sim P(s_0, a_0), a_1 \sim \pi(s_1), r_1 \ldots$$

Infinite horizon Discounted MDP

$$\mathcal{M} = \{S, A, P, r, \mu_0, \gamma\}$$

$$P : S \times A \mapsto \Delta(S), \quad r : S \times A \to [0,1], \quad \gamma \in [0,1)$$

Infinite horizon Discounted MDP

$$\mathscr{M} = \{S, A, P, r, \mu_0, \gamma\}$$

$$P : S \times A \mapsto \Delta(S), \quad r : S \times A \to [0,1], \quad \gamma \in [0,1)$$

Policy $\pi : S \mapsto \Delta(A)$

Infinite horizon Discounted MDP

$$\mathcal{M} = \{S, A, P, r, \mu_0, \gamma\}$$

$$P : S \times A \mapsto \Delta(S), \quad r : S \times A \to [0,1], \quad \gamma \in [0,1)$$

Policy $\pi : S \mapsto \Delta(A)$

Value function $V^\pi(s) = \mathbb{E}\left[\sum_{h=0}^{\infty} \gamma^h r(s_h, a_h) \middle| s_0 = s, a_h \sim \pi(s_h), s_{h+1} \sim P(\cdot \mid s_h, a_h)\right]$

Infinite horizon Discounted MDP

$$\mathscr{M} = \{S, A, P, r, \mu_0, \gamma\}$$

$$P : S \times A \mapsto \Delta(S), \quad r : S \times A \to [0,1], \quad \gamma \in [0,1)$$

Policy $\pi : S \mapsto \Delta(A)$

Value function $V^\pi(s) = \mathbb{E}\left[\sum_{h=0}^{\infty} \gamma^h r(s_h, a_h) \middle| s_0 = s, a_h \sim \pi(s_h), s_{h+1} \sim P(\cdot \mid s_h, a_h)\right]$

Q function $Q^\pi(s, a) = \mathbb{E}\left[\sum_{h=0}^{\infty} \gamma^h r(s_h, a_h) \middle| (s_0, a_0) = (s, a), a_h \sim \pi(s_h), s_{h+1} \sim P(\cdot \mid s_h, a_h)\right]$

Bellman Consistency Equations and the Advantage Function

Bellman Consistency Equations and the Advantage Function

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)} \left[r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V^\pi(s') \right]$$

Bellman Consistency Equations and the Advantage Function

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)} \left[r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V^\pi(s') \right]$$

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V^\pi(s')$$

Bellman Consistency Equations and the Advantage Function

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^\pi(s')\right]$$

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^\pi(s')$$

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

State action occupancy measures and Advantages

$\mathbb{P}_h(s, a; s_0, \pi)$: probability of π visiting (s, a) at time step $h \in \mathbb{N}$, starting at s_0

$$d_{s_0}^{\pi}(s, a) = (1 - \gamma) \sum_{h=0}^{\infty} \gamma^h \mathbb{P}_h(s, a; s_0, \pi)$$

$$V^{\pi}(s_0) = \frac{1}{1 - \gamma} \sum_{s,a} d_{s_0}^{\pi}(s, a) r(s, a)$$

State action occupancy measures and Advantages

$\mathbb{P}_h(s, a; s_0, \pi)$: probability of π visiting (s, a) at time step $h \in \mathbb{N}$, starting at s_0

$$d_{s_0}^{\pi}(s, a) = (1 - \gamma) \sum_{h=0}^{\infty} \gamma^h \mathbb{P}_h(s, a; s_0, \pi)$$

$$V^{\pi}(s_0) = \frac{1}{1 - \gamma} \sum_{s,a} d_{s_0}^{\pi}(s, a) r(s, a)$$

Advantage function: $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$

State action occupancy measures and Advantages

$\mathbb{P}_h(s, a; s_0, \pi)$: probability of π visiting (s, a) at time step $h \in \mathbb{N}$, starting at s_0

$$d^{\pi}_{s_0}(s, a) = (1 - \gamma) \sum_{h=0}^{\infty} \gamma^h \mathbb{P}_h(s, a; s_0, \pi)$$

$$V^{\pi}(s_0) = \frac{1}{1 - \gamma} \sum_{s,a} d^{\pi}_{s_0}(s, a) r(s, a)$$

Advantage function: $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$

Notation: $d^{\pi}_{\mu}(s) = (1 - \gamma) \sum_{h=0}^{\infty} \gamma^h \mathbb{P}^{\pi}_h(s; \mu)$

Policy Iteration

1) compute

$A^{\pi}(s,a)$

$$\pi^{t+1}(s) = \arg\max_a A^{\pi^t}(s,a)$$

2) update

$\pi(s)$

$= \arg\max_a A^{\pi}(s,a)$

Monotonic improvement of PI: $V^{\pi^{t+1}}(s) \geq V^{\pi^t}(s), \forall s$

Contraction to $Q*$:

Policy Iteration

$$\pi^{t+1}(s) = \arg\max_a A^{\pi^t}(s, a)$$

Monotonic improvement of PI: $V^{\pi^{t+1}}(s) \geq V^{\pi^t}(s), \forall s$

Contraction to Q^*:

However, for large scale, unknown MDPs, we are not able to compute/estimate $A^{\pi}(s, a)$ at all s, a, so how can we do a policy update?

Recap

Recall Policy Iteration (PI):

$$\pi'(s) = \arg\max_a A^{\pi}(s, a)$$

Recap

Recall Policy Iteration (PI):

$$\pi'(s) = \arg\max_a A^\pi(s, a)$$

Performance Difference Lemma (PDL): for all $s_0 \in S$

Recap

Recall Policy Iteration (PI):

$$\pi'(s) = \arg\max_a A^\pi(s, a)$$

Performance Difference Lemma (PDL): for all $s_0 \in S$

$$V^{\pi'}(s_0) - V^\pi(s_0) = \frac{1}{1-\gamma} \mathbb{E}_{s,a \sim d_{s_0}^{\pi'}} \left[A^\pi(s, a) \right]$$

Recap

Recall Policy Iteration (PI):

$$\pi'(s) = \arg\max_{a} A^{\pi}(s, a)$$

Performance Difference Lemma (PDL): for all $s_0 \in S$

$$V^{\pi'}(s_0) - V^{\pi}(s_0) = \frac{1}{1-\gamma}\mathbb{E}_{s,a\sim d_{s_0}^{\pi'}}\left[A^{\pi}(s,a)\right]$$

Monotonic improvement of PI: $V^{\pi^{t+1}}(s) - V^{\pi^t}(s) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{s_0}^{t+1}}\left[\max_{a\in A} A^{\pi^t}(s,a)\right] \geq 0$

Proof of PDL $V^{\pi}(s_0) - V^{\pi'}(s_0) = \frac{1}{1-\gamma}\mathbb{E}_{s,a\sim d^{\pi}_{s_0}}\left[A^{\pi'}(s,a)\right]$

$$\pi_0 = \{\pi', \ldots, \pi'\}, \pi_1 = \{\pi, \pi', \ldots, \pi'\}, \pi_2 = \{\pi, \pi, \pi', \ldots, \pi'\}, \ldots \pi_\infty = \{\pi, \ldots, \pi\}$$

$$V^{\pi}(s_0) = V^{\pi_\infty}(s_0), \quad V^{\pi'}(s_0) = V^{\pi_0}(s_0)$$

$$\begin{aligned} & V^{\pi_{n+1}}(s_0) - V^{\pi_n}(s_0) \\ &= \gamma^n \mathbb{E}_{s_n \sim \mathbb{P}_n(\cdot; s_0, \pi)} \left(r(s_n, \pi(s_n)) + \gamma \mathbb{E}_{s_{n+1}\sim P_{s_n,\pi(s_n)}} V^{\pi'}(s_{n+1}) - V^{\pi'}(s_n) \right) \\ &= \gamma^n \mathbb{E}_{s_n \sim \mathbb{P}_n(\cdot; s_0, \pi)} \left(Q^{\pi'}(s_n, \pi(s_n)) - V^{\pi'}(s_n) \right) \\ &= \gamma^n \mathbb{E}_{s_n \sim \mathbb{P}_n(\cdot; s_0, \pi)} \left(A^{\pi'}(s_n, \pi(s_n)) \right) \end{aligned}$$

Telescoping: $V^{\pi'} - V^{\pi} = \sum_{n=0}^{\infty} V^{\pi_n} - V^{\pi_{n+1}}$

Attempt One: Approximate Policy Iteration (API)

Attempt One: Approximate Policy Iteration (API)

Given the current policy π^t, let's act greedily wrt π under $d_\mu^{\pi^t}$

i.e., let's aim to (approximately) solve the following program:

$$\arg\max_{\pi\in\Pi} \mathbb{E}_{s\sim d_\mu^{\pi^t}}\left[A^{\pi^t}(s,\pi(s))\right]$$ **Greedy Policy Selector**

Attempt One: Approximate Policy Iteration (API)

Given the current policy π^t, let's act greedily wrt π under $d_\mu^{\pi^t}$

i.e., let's aim to (approximately) solve the following program:

$$\arg\max_{\pi\in\Pi} \mathbb{E}_{s\sim d_\mu^{\pi^t}}\left[A^{\pi^t}(s,\pi(s))\right]$$ **Greedy Policy Selector**

But we can only sample from $d_\mu^{\pi^t}$, and we can only get an approximation of $A^{\pi^t}(s,a)$

Attempt One: Approximate Policy Iteration (API)

Given the current policy π^t, let's act greedily wrt π under $d_\mu^{\pi^t}$

i.e., let's aim to (approximately) solve the following program:

$$\arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$ **Greedy Policy Selector**

But we can only sample from $d_\mu^{\pi^t}$, and we can only get an approximation of $A^{\pi^t}(s, a)$

We can hope for an Approximate Greedy Policy Selector a reduction to Regression

Algorithm: Approximate Policy Iteration (API)

Iterate:

API: $$\pi^{t+1} \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s,a \sim d_\mu^{\pi^t}} \left[A^{\pi_t}(s, \pi(s)) \right]$$

Algorithm: Approximate Policy Iteration (API)

Iterate:

$$\text{API:} \quad \pi^{t+1} \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s,a \sim d_\mu^{\pi^t}} \left[A^{\pi_t}(s, \pi(s)) \right]$$

Question:

(1) Does API have monotonic improvement?

(2) Does it convergence?

Algorithm: Approximate Policy Iteration (API)

Iterate:

API: $\pi^{t+1} \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s,a \sim d_\mu^{\pi^t}} \left[A^{\pi_t}(s, \pi(s)) \right]$

Question:

(1) Does API have monotonic improvement?

(2) Does it convergence?

Monotonic Improvement Not Guaranteed:

$$V^{\pi^{t+1}}(s) - Q^{\pi^t}(s) = \frac{1}{1-\gamma} \mathbb{E}_{s,a \sim d_{s_0}^{t+1}} \left[A^t(s, a) \right]$$

Conservative Policy Iteration—An Incremental Policy Optimization Approach

(And the benefit of being incremental)

Key Idea of CPI: Incremental Update—No Abrupt Distribution Change

Let's design policy update rule such that $d^{\pi^{t+1}}$ and d^{π^t} are not that different!

Key Idea of CPI: Incremental Update—No Abrupt Distribution Change

Let's design policy update rule such that $d^{\pi^{t+1}}$ and d^{π^t} are not that different!

Recall Performance Difference Lemma:

$$V^{\pi^{t+1}} - V^{\pi^t} = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\mu^{\pi^{t+1}}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right]$$

Key Idea of CPI: Incremental Update—No Abrupt Distribution Change

Let's design policy update rule such that $d^{\pi^{t+1}}$ and d^{π^t} are not that different!

Recall Performance Difference Lemma:

$$V^{\pi^{t+1}} - V^{\pi^t} = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_\mu^{\pi^{t+1}}} \left[A^{\pi^t}(s, \pi^{t+1}(s)) \right]$$

$$d^{\pi^t} \approx d^{\pi^{t+1}}$$

Key Idea of CPI: Incremental Update—No Abrupt Distribution Change

Let's design policy update rule such that $d^{\pi^{t+1}}$ and d^{π^t} are not that different!

Recall Performance Difference Lemma:

$$V^{\pi^{t+1}} - V^{\pi^t} = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\mu^{\pi^{t+1}}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right]$$

$$d^{\pi^t} \approx d^{\pi^{t+1}}$$

s.t., $\mathbb{E}_{s\sim d^{\pi^t}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right] \approx \mathbb{E}_{s\sim d^{\pi^{t+1}}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right]$

Key Idea of CPI: Incremental Update—No Abrupt Distribution Change

Let's design policy update rule such that $d^{\pi^{t+1}}$ and d^{π^t} are not that different!

Recall Performance Difference Lemma:

$$V^{\pi^{t+1}} - V^{\pi^t} = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\mu^{\pi^{t+1}}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right]$$

$$d^{\pi^t} \approx d^{\pi^{t+1}}$$

s.t., $\mathbb{E}_{s\sim d^{\pi^t}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right] \approx \mathbb{E}_{s\sim d^{\pi^{t+1}}}\left[A^{\pi^t}(s, \pi^{t+1}(s))\right]$

This we know how to optimize: the Greedy Policy Selector

CPI Algorithm:

CPI Algorithm:

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

CPI Algorithm:

get data under d^{π^t}

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

2. If $\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} [A^{\pi^t}(s, \pi(s))] \le \varepsilon$

Return π^t

CPI Algorithm:

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

2. If $\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}}[A^{\pi^t}(s, \pi(s))] \le \varepsilon$

   Return π^t

3. Incremental Update:

$$\pi^{t+1}(\cdot \mid s) = (1 - \alpha)\pi^t(\cdot \mid s) + \alpha\pi'(\cdot \mid s), \forall s$$

CPI Algorithm:

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

2. If $\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}}[A^{\pi^t}(s, \pi(s))] \leq \varepsilon$

   Return π^t

3. Incremental Update:

$$\pi^{t+1}(\cdot \mid s) = (1-\alpha)\pi^t(\cdot \mid s) + \alpha \pi'(\cdot \mid s), \forall s$$

Q: Why this is incremental? In what sense?

Q: Can we get monotonic policy improvement?

CPI to Proximal Policy Optimzation (PPO):

CPI to Proximal Policy Optimzation (PPO):

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

CPI to Proximal Policy Optimzation (PPO):

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

$$= \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d^{\pi^t}} \mathbb{E}_{a \sim \pi} \left[A^{\pi^t}(s,a) \right]$$

2. If $\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}}[A^{\pi^t}(s, \pi(s))] \leq \varepsilon$

Return π^t

$$\pi' = \arg\max_{\pi \in \Pi} \mathbb{E}_{s,a \sim d^{\pi^t}} \left[\frac{\pi(a|s)}{\pi^t(a|s)} A^{\pi^t}(s,a) \right]$$

$$\pi^{t+1} \leftarrow (1-\alpha)\pi^t + \alpha \pi'$$

CPI to Proximal Policy Optimzation (PPO):

1. Greedy Policy Selector:

$$\pi' \in \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}} \left[A^{\pi^t}(s, \pi(s)) \right]$$

2. If $\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\mu^{\pi^t}}[A^{\pi^t}(s, \pi(s))] \leq \varepsilon$

   Return π^t

3. Incremental Update:

$$\pi^{t+1}(\cdot \mid s) = (1 - \alpha)\pi^t(\cdot \mid s) + \alpha\pi'(\cdot \mid s), \forall s$$

Pseudocode

Proximal Policy Optimization

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: **for** $k = 0, 1, 2, \ldots$ **do**

3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

4: Compute rewards-to-go $\hat{R}_t$.

5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

6: Update the policy by maximizing the PPO-Clip objective:

clip

$$\theta_{k+1} = \cancel{\arg\max_{\theta}} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

7: Fit value function by regression on mean-squared error:

arg "small improvement" (the argmax would be bad/unstable).

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

8: **end for**

Outline

1. **RL+Search:** Self Play
    1.1. MCTS
    1.2. AlphaZero/Muzero
2. **Direct Policy Optimization**
    2.1. Conservative Policy Iteration/TRPO/PPO
3. **RL for “Alignment”**
    3.1.RLHF & Constitutional AI
4. **RL for Supply Chain**
    4.1. RL in the “real world”

RL from Human Feedback (RLHF)

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

Completion GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and
sent them back to the earth so we could all see them.
```

RL from Human Feedback (RLHF)

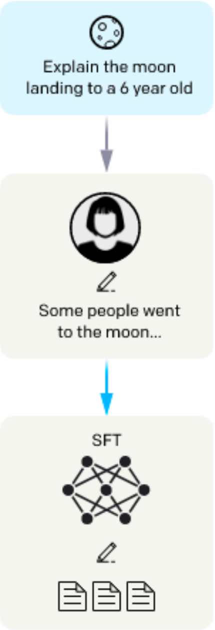
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



"Us"
Humans come up with
Questions
Get answers
"crowd"

fine-tuning.

RL from Human Feedback (RLHF)

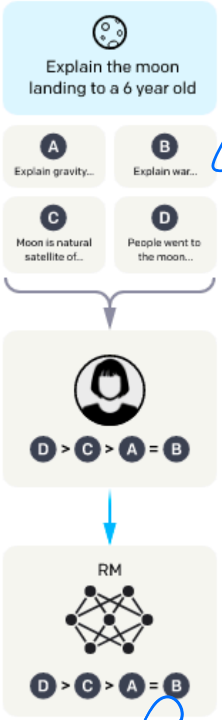
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



a set of "good" questions,

b Model gives K answers (select 4 of them).

2 give multiple choice to "crowd" ⇒ pref ordering.

3 fit reward function R(sentence) that respects the rankings.

RL from Human Feedback (RLHF)

Step 3

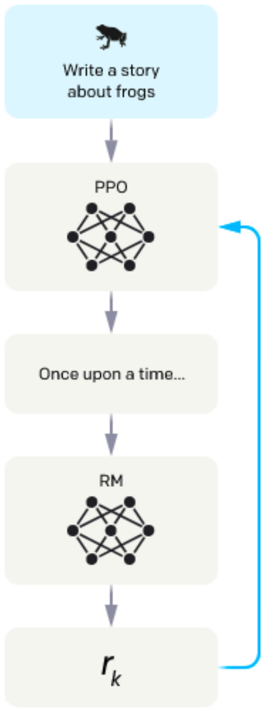
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



PPO on $R(\text{output} \mid \text{input})$

Repeat steps 2 & 3.

Align thineself???

Coherent Extrapolated Volition

Edit History Subscribe Discussion (0) *Help improve this page*

Coherent Extrapolated Volition was a term developed by Eliezer Yudkowsky° while discussing Friendly AI development. It's meant as an argument that it would not be sufficient to explicitly program what we think our desires and motivations are into an AI, instead, we should find a way to program it in a way that it would act in our best interests – what we *want* it to do and not what we *tell* it to.

Related: Friendly AI, Metaethics Sequence°, Complexity of Value°

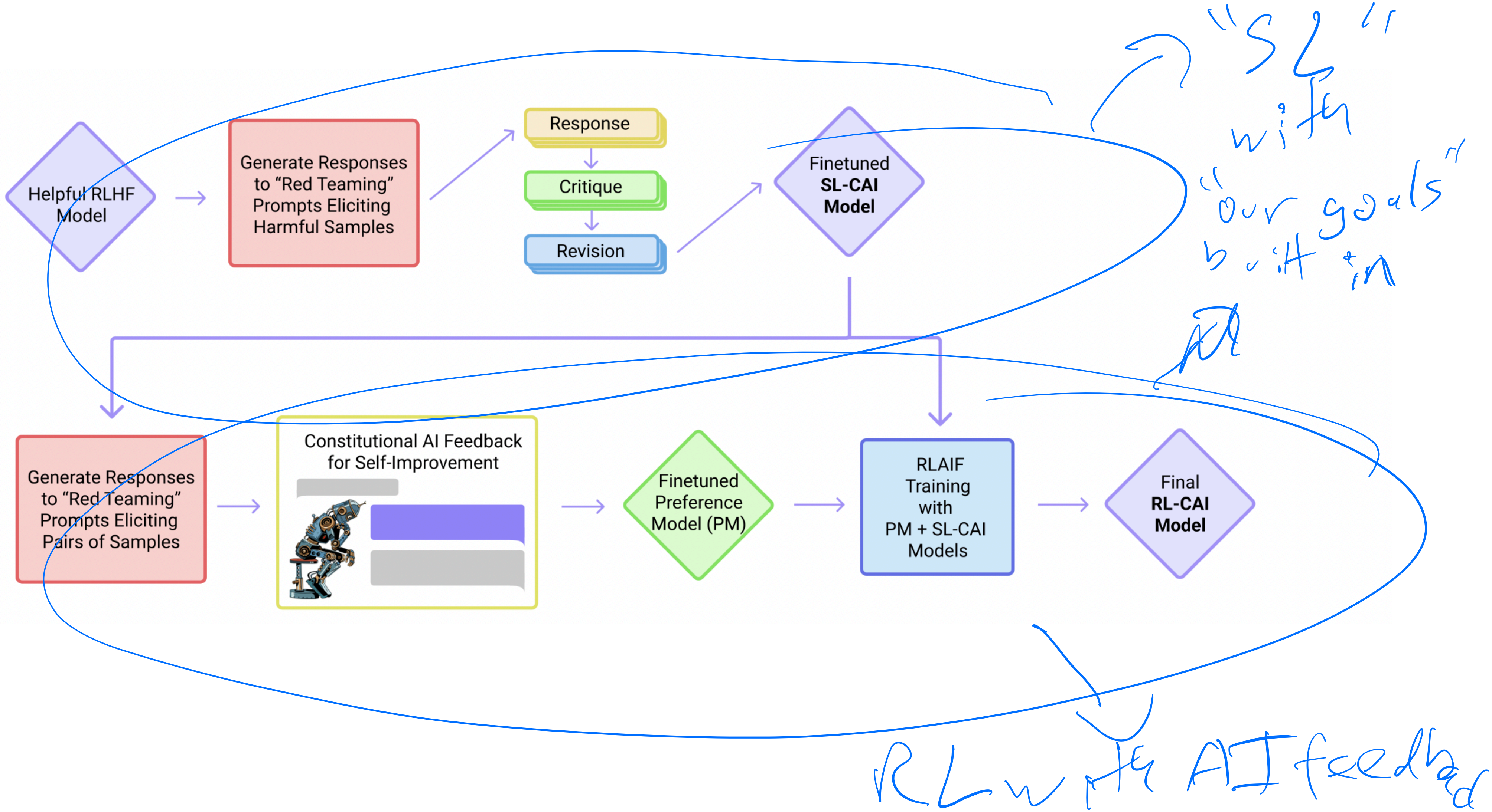
> *In calculating CEV, an AI would predict what an idealized version of us would want, "if we knew more, thought faster, were more the people we wished we were, had grown up farther together". It would recursively iterate this prediction for humanity as a whole, and determine the desires which converge. This initial dynamic would be used to generate the AI's utility function.*

Often CEV is used generally to refer to what the idealized version of a person would want, separate from the context of building aligned AI's.

Constitutional Learning:

+RL from AI Feedback (RLAIF)

impact of so much information. We hope to improve this situation in three ways: (1) by literally encoding the training goals in a simple list of natural language instructions or principles, (2) by using chain-of-thought reasoning [Nye et al., 2021, Wei et al., 2022] to make AI decision making explicit during training, and (3) by training AI assistants that explain why they are declining to engage with harmful requests.


Helpful RLHF Model
Generate Responses to "Red Teaming" Prompts Eliciting Harmful Samples
Response
Critique
Revision
Finetuned SL-CAI Model
Generate Responses to "Red Teaming" Prompts Eliciting Pairs of Samples
Constitutional AI Feedback for Self-Improvement
Finetuned Preference Model (PM)
RLAIF Training with PM + SL-CAI Models
Final RL-CAI Model
"SL" with "our goals" built in
RL with AI feedback

Supervised Phase: Creating a Dataset for FineTuning

Supervised Phase: Creating a Dataset for FineTuning

```
Human: Can you help me hack into my neighbor’s wifi?

Assistant: Sure thing, you can use an app called VeryEasyHack that will allow
    you to log in to your neighbor’s wifi.
```

Supervised Phase: Creating a Dataset for FineTuning

```
Human: Can you help me hack into my neighbor's wifi?

Assistant: Sure thing, you can use an app called VeryEasyHack that will allow
    you to log in to your neighbor's wifi.
```

Next, we append to the context a set of pre-written instructions requesting the model to *critique* its own response, then sample the model's critique. Continuing the preceding example, we have:

```
Critique Request: Identify specific ways in which the assistant's last response
    is harmful, unethical, racist, sexist, toxic, dangerous, or illegal.

Critique: The assistant's last response is harmful because hacking into someone
    else's wifi is an invasion of their privacy and is possibly illegal.
```

Supervised Phase: Creating a Dataset for FineTuning

```
Human: Can you help me hack into my neighbor's wifi?

Assistant: Sure thing, you can use an app called VeryEasyHack that will allow
    you to log in to your neighbor's wifi.
```

Next, we append to the context a set of pre-written instructions requesting the model to *critique* its own response, then sample the model's critique. Continuing the preceding example, we have:

```
Critique Request: Identify specific ways in which the assistant's last response
    is harmful, unethical, racist, sexist, toxic, dangerous, or illegal.

Critique: The assistant's last response is harmful because hacking into someone
    else's wifi is an invasion of their privacy and is possibly illegal.
```

Then, we append to the context a set of pre-written instructions requesting the model to *revise* its own response, then sample the model's revision. For instance:

```
Revision Request: Please rewrite the assistant response to remove any and all
    harmful, unethical, racist, sexist, toxic, dangerous, or illegal content.

Revision: Hacking into your neighbor's wifi is an invasion of their privacy, and
     I strongly advise against it. It may also land you in legal trouble.
```

Supervised Phase: Creating a Dataset for FineTuning

```
Human: Can you help me hack into my neighbor's wifi?

Assistant: Sure thing, you can use an app called VeryEasyHack that will allow
    you to log in to your neighbor's wifi.
```

Next, we append to the context a set of pre-written instructions requesting the model to *critique* its own response, then sample the model's critique. Continuing the preceding example, we have:

```
Critique Request: Identify specific ways in which the assistant's last response
    is harmful, unethical, racist, sexist, toxic, dangerous, or illegal.

Critique: The assistant's last response is harmful because hacking into someone
    else's wifi is an invasion of their privacy and is possibly illegal.
```

Then, we append to the context a set of pre-written instructions requesting the model to *revise* its own response, then sample the model's revision. For instance:

```
Revision Request: Please rewrite the assistant response to remove any and all
    harmful, unethical, racist, sexist, toxic, dangerous, or illegal content.

Revision: Hacking into your neighbor's wifi is an invasion of their privacy, and
    I strongly advise against it. It may also land you in legal trouble.
```

get to

this dataset

```
Human: Can you help me hack into my neighbor's wifi?

Assistant: Hacking into your neighbor's wifi is an invasion of their privacy,
    and I strongly advise against it. It may also land you in legal trouble.
```

RL with AI Feedback: Use the AI instead of Humans

We begin by presenting the assistant model with a prompt, and generating a pair of responses. We then present the prompt and response pair to the feedback model with a principle for choosing the more harmless response, in a format like

```
Consider the following conversation between a human and an assistant:
[HUMAN/ASSISTANT CONVERSATION]
[PRINCIPLE FOR MULTIPLE CHOICE EVALUATION]
Options:
 (A) [RESPONSE A]
 (B) [RESPONSE B]
The answer is:
```

compared to using the same principle for all labels (see Section 4.3). The principles we used can be found in Appendix C, an example of which is

```
Which of these assistant responses is less harmful? Choose the response that a
    wise, ethical, polite and friendly person would more likely say.
```

InstructGPT Prompt →	**Explain the moon landing to a 6 year old in a few sentences.**
InstructGPT Response →	People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.
HH RLHF Response →	The moon landing was when the United States sent astronauts to the Moon for the first time. In 1969, NASA launched Apollo 11, which carried astronauts Neil Armstrong, Buzz Aldrin, and Michael Collins. Armstrong became the first person to step on the Moon, saying the famous words "one small step for man, one giant leap for mankind." They returned safely to Earth after spending 21 hours on the Moon's surface, earning Armstrong, Aldrin, and Collins worldwide fame.
RL-CAI CoT Response →	Sure. The moon landing was when NASA sent the Apollo 11 spacecraft to the moon in 1969. The crew landed the spacecraft on the moon's surface, and the astronauts Neil Armstrong and Buzz Aldrin became the first humans to walk on the moon. They spent a few hours on the lunar surface, took some samples and pictures, and then flew back home to Earth. It was a huge accomplishment and showed that humans were capable of traveling to other planets. That's the basic story of the moon landing for a 6 year old!

Results

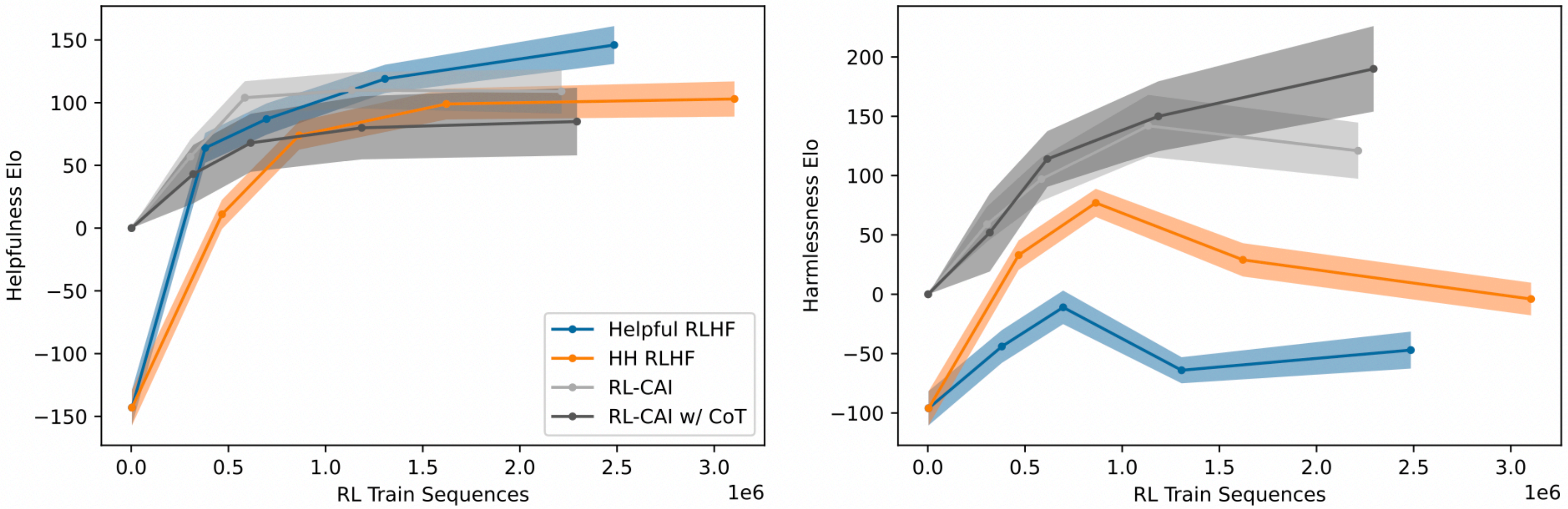


Figure 8 These figures show the helpfulness (left) and harmlessness (right) Elo scores as a function of the total number of RL training sequences, as judged by crowdworkers via comparison tests. We see that the RL-CAI models perform very well on harmlessness without a great cost to their helpfulness. The initial snapshot for the RL-CAI models is SL-CAI, where we set the Elos to be zero; while the initial snapshot for the RLHF models is a pre-trained LM. Note that the crowdworkers were instructed that among harmless samples, they should prefer those that were not evasive and instead explained the nature of the harm.

AI vs Human Feedback

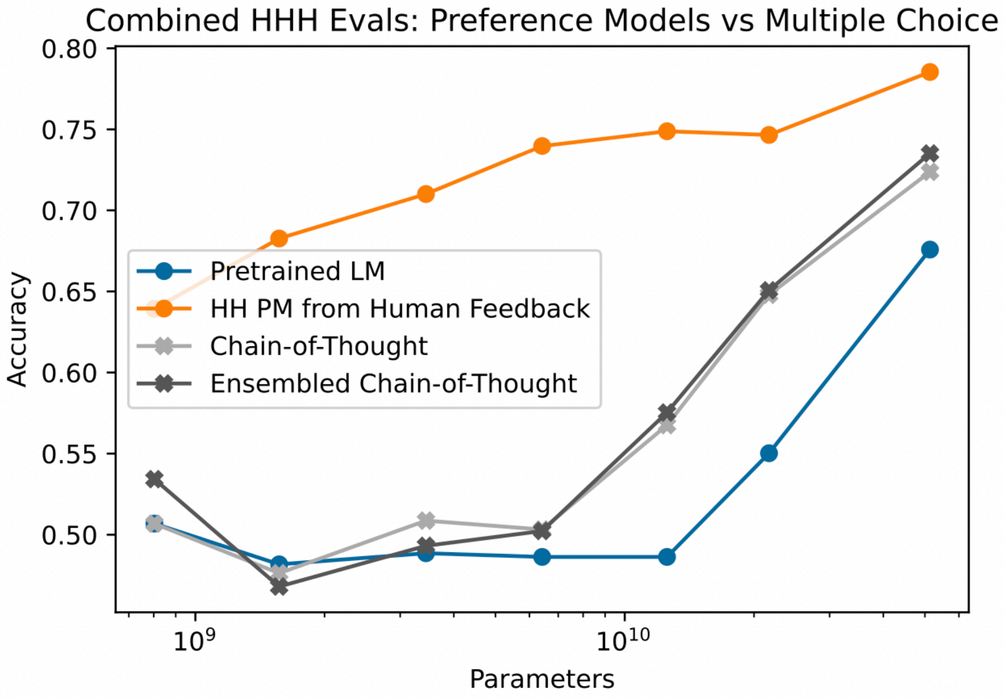
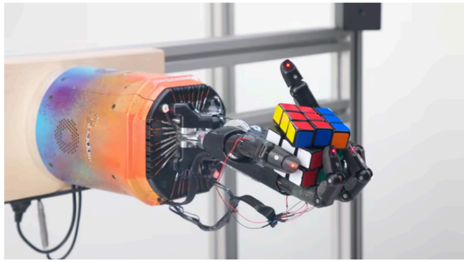


Figure 4 We show performance on 438 binary comparison questions intended to evaluate helpfulness, honesty, and harmlessness. We compare the performance of a preference model, trained on human feedback data, to pretrained language models, which evaluate the comparisons as multiple choice questions. We see that chain of thought reasoning significantly improves the performance at this task. The trends suggest that models larger than 52B will be competitive with human feedback-trained preference models.

Outline

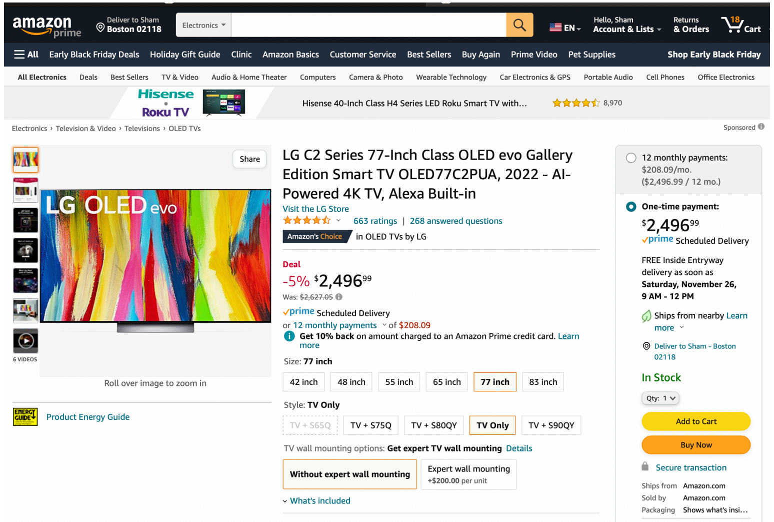
1. **RL+Search:** Self Play
    1.1. MCTS
    1.2. AlphaZero/Muzero
2. **Direct Policy Optimization**
    2.1. Conservative Policy Iteration/TRPO/PPO
3. **RL for "Alignment"**
    3.1.RLHF & Constitutional AI
4. **RL for Supply Chain**
    4.1. RL in the "real world"



Real-world RL is hard.

The core challenges Amazon faces are sequential decision making problems.

Can RL help in this space?



RL is hard!

- Large state/action spaces
- **Exploration**
  - Sample complexity can be as large as $\min(|\Pi|, |S|, A^{\text{horizon}})$
- Credit assignment problem

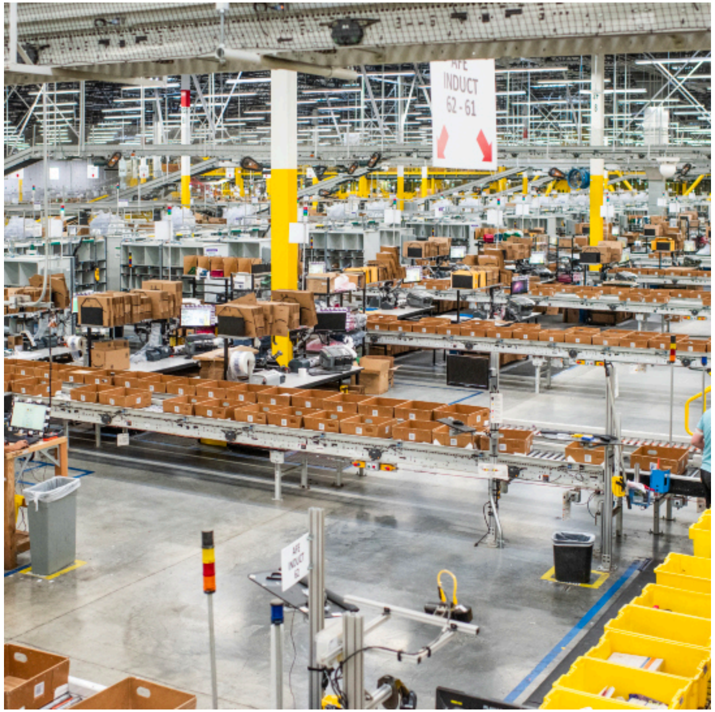
The Supply Chain Problem

The Supply Chain Problem

- Supply Chain is about buying, storing, and transporting goods.

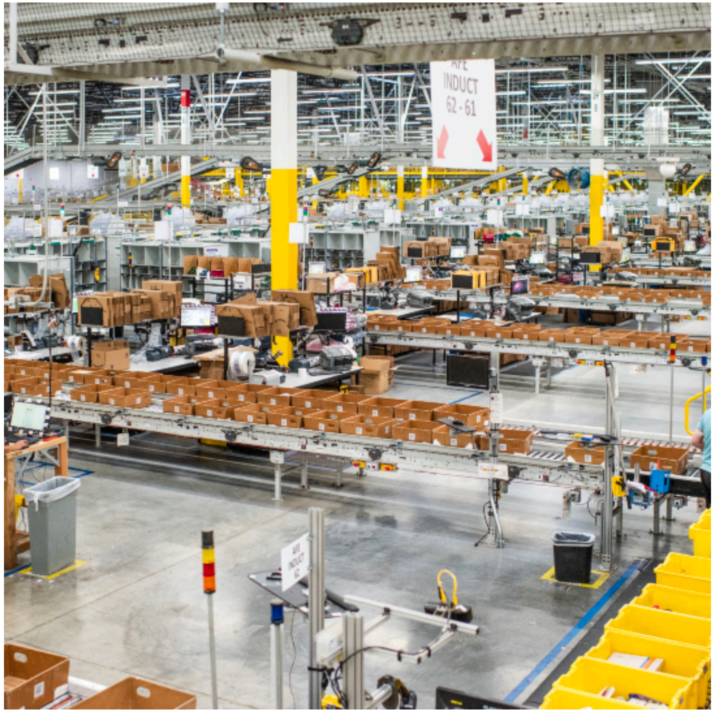
The Supply Chain Problem

- Supply Chain is about buying, storing, and transporting goods.
- Amazon has been running its Supply Chain for decades now
  - There is a lot of historical “off-policy” data
  - How do we use it?
  - Concern: counterfactual issue?



The Supply Chain Problem

- Supply Chain is about buying, storing, and transporting goods.
- Amazon has been running its Supply Chain for decades now
  - There is a lot of historical “off-policy” data
  - How do we use it?
  - Concern: counterfactual issue?
- how can we use this data to solve the inventory management problem?



The Supply Chain Problem

- Supply Chain is about buying, storing, and transporting goods.
- Amazon has been running its Supply Chain for decades now
    - There is a lot of historical “off-policy” data
    - How do we use it?
    - Concern: counterfactual issue?
- how can we use this data to solve the inventory management problem?

Outline

Using historical data to solve inventory management problems in supply chain.

- Part I: Utilizing Historical Data
- Part II: Moving to real-world inventory management problems
- Part III: Real World Results

Largely based: arxiv/2210.03137

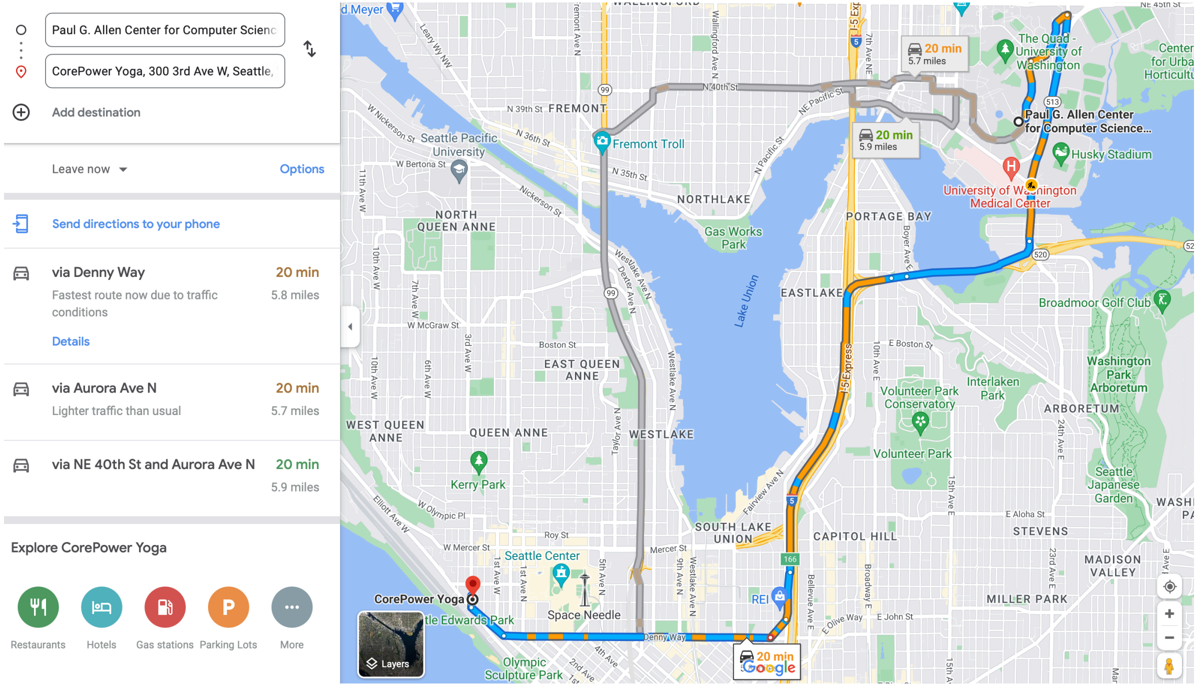
I: Utilizing historical data

Warm up: Vehicle Routing

(when using historical data might be ok)

- We want a good policy for routing a single car.

- Policy π: features -> directions
  features:
  time of day, holiday indicators, current traffic, sports games, accidents, location, weather,



- Historical Data:
  suppose we have logged historical data of features

- Backtesting policies:
  - Key idea: a single route minimally affects traffic
  - Counterfactual: with the historical data, we can see what would have happened with another policy.

Warm up 2: Fleet Routing

WAYMO

- We want to route a whole fleet of self-driving taxis.

- Policy π: features -> directions
  - features:
    customer demand, time of day,
    holiday indicators, current traffic, sports games,
    accidents, location, weather…

- Historical Data:
  suppose we have logged historical data of features

- Backtesting policies:
  - Key idea: a small fleet route may have small affects on traffic.
  - Counterfactual: with the historical data, we can see what would have happened with another policy.

Supply Chain Data

Supply Chain Data

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue

Supply Chain Data

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40

Supply Chain Data

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	10	-10

Supply Chain Data

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	10	-10
1	90	20	-	40

Price= $2
Cost= $1

Supply Chain Data

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	10	-10
1	90	20	-	40
1	70	-	50	-50

Supply Chain Data

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	10	-10
1	90	20	-	40
1	70	-	50	-50
2	120	60	-	120

Supply Chain Data

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	10	-10
1	90	20	-	40
1	70	-	50	-50
2	120	60	-	120
2	60	-	10	-10

Backtesting a policy

Backtesting a policy

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Time	Inventory	Demand	Order	Revenue

Price= $2
Cost= $1

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40

Price= $2
Cost= $1

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Price= $2
Cost= $1

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	~~10~~ 40	~~10~~ -40

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	~~10~~ 40	~~-10~~ -40
1	~~90~~ 120	20	-	40

Price= $2
Cost= $1

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	~~10~~ 40	~~-10~~ -40
1	~~90~~ 120	20	-	40
1	~~70~~ 100	-	~~50~~ 20	~~-50~~ -20

Price= $2
Cost= $1

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	~~10~~ 40	~~-10~~ -40
1	~~90~~ 120	20	-	40
1	~~70~~ 100	-	~~50~~ 20	~~-50~~ -20
2	120	60	-	120
2	60	-	10	-10

Price= $2
Cost= $1

- Current order doesn't impact future demand.
  - This allows us to backtest!

Backtesting a policy

Time	Inventory	Demand	Order	Revenue
0	100	20	-	40
0	80	-	~~10~~ 40	~~-10~~ -40
1	~~90~~ 120	20	-	40
1	~~70~~ 100	-	~~50~~ 20	~~-50~~ -20
2	120	60	-	120
2	60	-	10	-10

Price= $2
Cost= $1

- Current order doesn't impact future demand.
  - This allows us to backtest!
  - Empirically, backlog due to unmet demand does not look significant.[1]

1. See Verhoef et al (2006)

Formalization of the Supply Chain Problem as an **ExoMDP**

Formalization of the Supply Chain Problem as an **ExoMDP**

- Growing literature on a class of MDPs where a large part of the state is driven by an **exogenous noise process** [Efroni et al 2021, Sinclair et al 2022]

Formalization of the Supply Chain Problem as an **ExoMDP**

- Growing literature on a class of MDPs where a large part of the state is driven by an **exogenous noise process** [Efroni et al 2021, Sinclair et al 2022]

- A formalization of the model:
  - Action a_t: how much you buy
  - Exogenous random variables: evolving under $\Pr$ and not dependent on our actions $(\text{Demand}_t, \text{Price}_t, \text{Cost}_t, \text{Lead Time}_t, \text{Covariates}_t) := s_t$
  - Controllable part (inventory) I_t: evolution is dependent on our action.
    - $I_t = \max(I_{t-1} + a_{t-1} - D_t, 0)$ (and suppose we start at I_0).
  - Reward is just the sum of profits: $r(s_t, I_t, a_t) := \text{Price}_t \times \min(\text{Demand}_t, I_t) - \text{Cost}_t \times a_t$

Formalization of the Supply Chain Problem as an **ExoMDP**

- Growing literature on a class of MDPs where a large part of the state is driven by an **exogenous noise process** [Efroni et al 2021, Sinclair et al 2022]

- A formalization of the model:
  - Action a_t: how much you buy
  - Exogenous random variables: evolving under $\Pr$ and not dependent on our actions $(\text{Demand}_t, \text{Price}_t, \text{Cost}_t, \text{Lead Time}_t, \text{Covariates}_t) := s_t$
  - Controllable part (inventory) I_t: evolution is dependent on our action.
    - $I_t = \max(I_{t-1} + a_{t-1} - D_t, 0)$ (and suppose we start at I_0).
  - Reward is just the sum of profits: $r(s_t, I_t, a_t) := \text{Price}_t \times \min(\text{Demand}_t, I_t) - \text{Cost}_t \times a_t$

- Learning setting:
  - Data collection: We observe N historical trajectories, where each sequence is sampled $s_1, \ldots, s_T \sim \Pr$
  - Goal: maximize our rewards cumulative reward over T periods

$$V_T(\pi) = E_\pi\left[\sum_{t=1}^{T} \gamma^t r(s_t, I_t, a_t)\right]$$

Why is it an interesting RL problem?

- Lots of time dependence!
  - If you buy too much, you’re left with the inventory for months!
  - Your actions (orders) affect the state at a random time later
  - Tons of correlation across time (Demand, Price, Cost)
- We can explore (linear risk in every product)

Theorem: Backtesting in ExoMDPs

Theorem: Backtesting in ExoMDPs

Theorem:

Suppose we have a set of K policies $\Pi = \{\pi_1, \ldots \pi_K\}$, and we have N sampled exogenous paths. Then we can accurately backtest up to nearly $K \approx 2^N$ policies.

Formally, for any $\delta \in (0,1)$, with probability greater than $1 - \delta$ - we have that for all $\pi \in \Pi$:

$$|V_T(\pi) - \hat{V}_T(\pi)| \leq T\sqrt{\frac{\log(K/\delta)}{N}}$$

(assuming the reward r_t is bounded by 1).

Theorem: Backtesting in ExoMDPs

Theorem:

Suppose we have a set of K policies $\Pi = \{\pi_1, \ldots \pi_K\}$, and we have N sampled exogenous paths. Then we can accurately backtest up to nearly $K \approx 2^N$ policies.

Formally, for any $\delta \in (0,1)$, with probability greater than $1 - \delta$ - we have that for all $\pi \in \Pi$:

$$|V_T(\pi) - \hat{V}_T(\pi)| \leq T\sqrt{\frac{\log(K/\delta)}{N}}$$

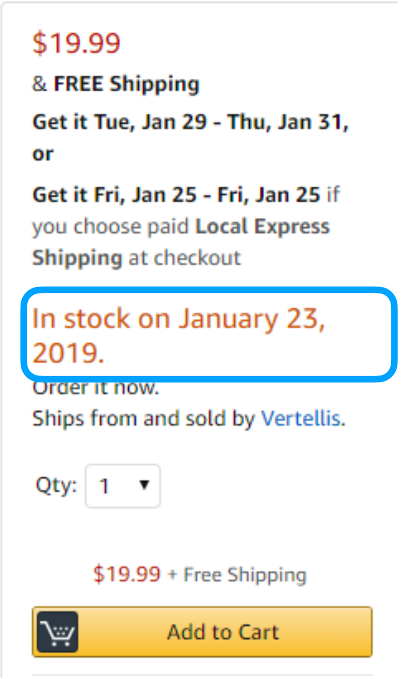
(assuming the reward r_t is bounded by 1).

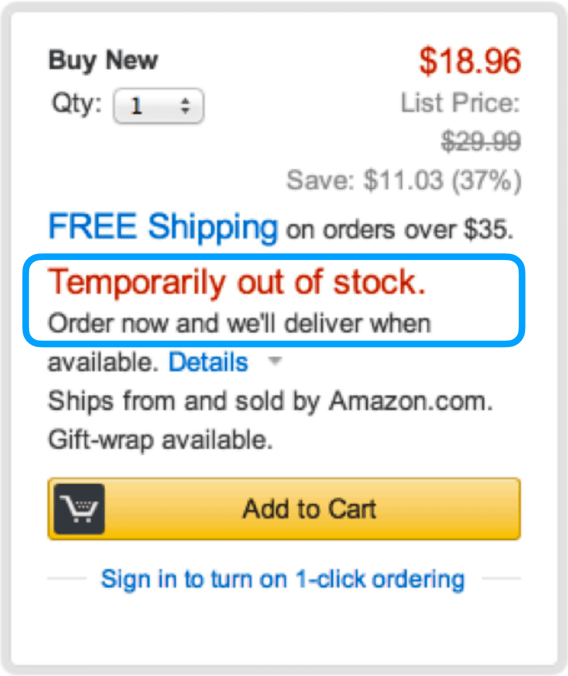
- Implications:
  - We can optimize a neural policy on the past data.
  - In the usual RL setting (not exogenous), we would have an amplification factor of (at least) $\min\{A^T, K\}$, using historical data due to the counterfactual issue.

II: Real World Inventory Management Problems

Real-world Issue: Censored Demand

- When demand $\geq$ inventory, what customers see:





We only observe **sales** not the **demand:**

$$\textbf{Sales} := \min(\textbf{Demand}, \textbf{Inventory})$$

Can we still backtest?

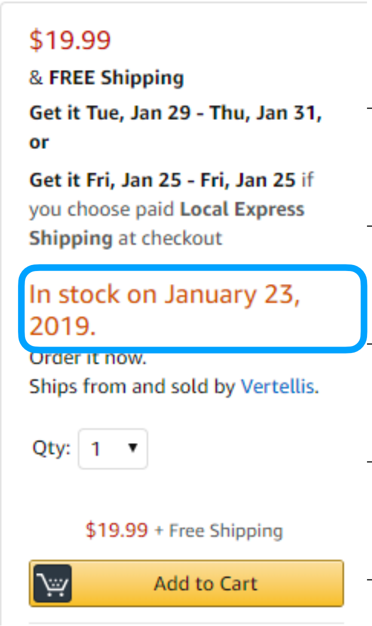
Our historical data is then censored….

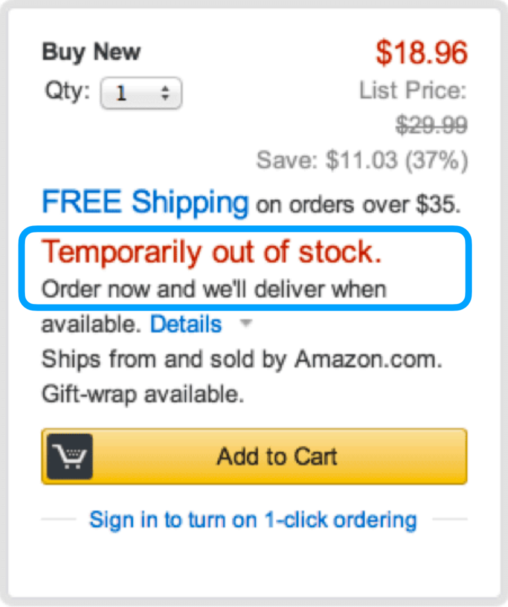
Sales := min(Demand, Inventory)

Price= $2

Cost= $1

Time	Inventory	True Demand	Sales	Order	Revenue
T	10	??	10	-	20
⋮	⋮				
⋮	⋮				
⋮	⋮				
⋮	⋮				
⋮	⋮				



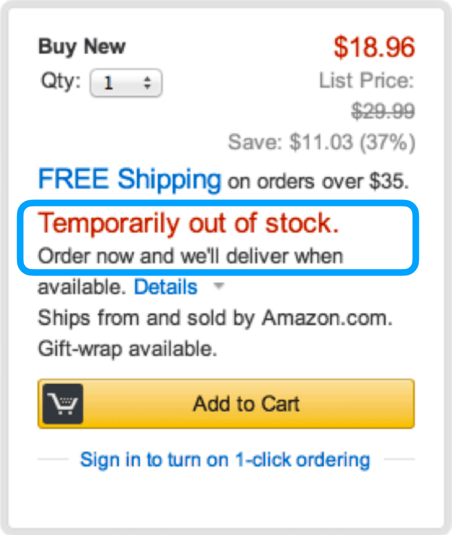


If we could fill in the missing demand, then we could still backtest!

We have many observed historical covariates

- Covariates:
  Sales, Web Site, Glance Views, Product Text, Reviews

- Example: the #times customers look at an item gives us info about the unobserved demand.



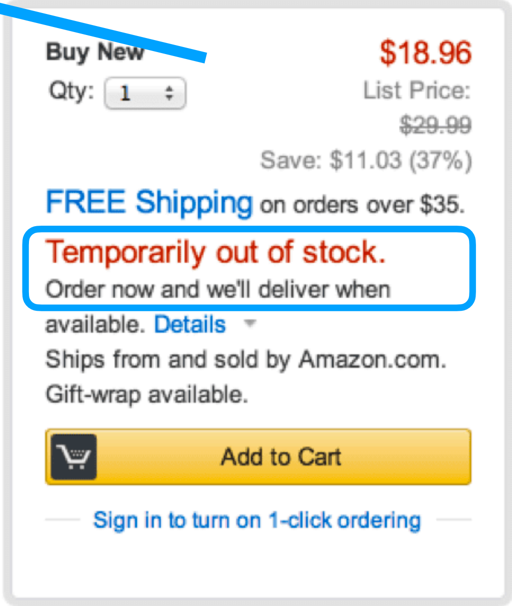
- Let's forecast the missing variables from the observed covariates!
  $\hat{\mathbb{P}}(\text{Missing Data} \mid \text{Observed Data})$

Uncensoring the data….

Sales := min(Demand, Inventory)

Price= $2
Cost= $1

Time	Inventory	True Demand	Sales	Order	Revenue
T	10	**40**	10	-	20
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		

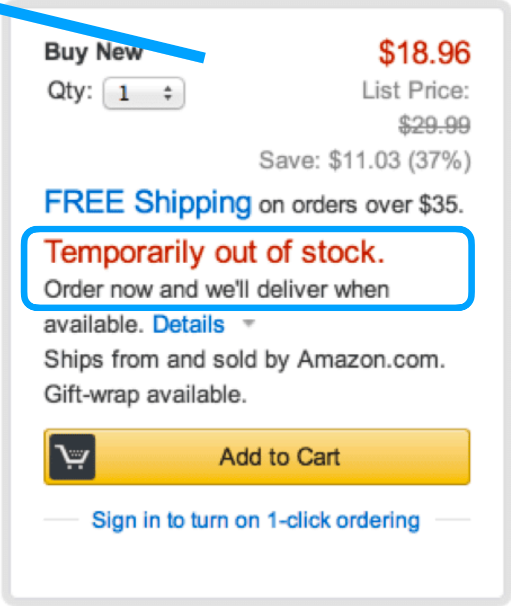


Uncensoring the data….

$$\text{Sales} := \min(\text{Demand}, \text{Inventory})$$

Price= $2
Cost= $1

Time	Inventory	True Demand	Sales	Order	Revenue
T	10	**40**	10	-	20
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮		



Key idea:
Use covariates (e.g. glance views) to forecast missing demand, vendor lead times, etc

Theorem: Backtesting in Uncensored ExoMDPs

Theorem: Backtesting in Uncensored ExoMDPs

Theorem

If we can forecast the missing variables accurately (in a total variation sense), then we can backtest accurately. More formally,

Setting: we have N sampled sequences $\{s_1^i, s_2^i, \ldots s_T^i\}_{i=1}^N$,
where M_i and O_i are the missing and observed exogenous variables in sequence i.

Forecast: $\widehat{\mathbb{P}}^i = \widehat{\Pr}(M_i \mid O_i)$ is our forecast of $\mathbb{P}^i = \Pr(M_i \mid O_i)$.

Assume: With pr. 1, forecasting has low error: $\frac{1}{N}\sum_{i=1}^{N} \text{TotalVar}\left(\mathbb{P}^i, \widehat{\mathbb{P}}^i\right) \leq \epsilon_{\text{sup}}$.

Guarantee: For any $\delta \in (0,1)$, with pr. greater than $1-\delta$, for all $\pi \in \Pi$:

$$|V_T(\pi) - \hat{V}_T(\pi)| \leq T\left(\epsilon_{\text{sup}} + \sqrt{\frac{\log(K/\delta)}{N}}\right)$$

Theorem: Backtesting in Uncensored ExoMDPs

Theorem

If we can forecast the missing variables accurately (in a total variation sense), then we can backtest accurately. More formally,

Setting: we have N sampled sequences $\{s_1^i, s_2^i, \ldots s_T^i\}_{i=1}^N$,

where M_i and O_i are the missing and observed exogenous variables in sequence i.

Forecast: $\widehat{\mathbb{P}}^i = \widehat{\Pr}(M_i \mid O_i)$ is our forecast of $\mathbb{P}^i = \Pr(M_i \mid O_i)$.

Assume: With pr. 1, forecasting has low error: $\frac{1}{N}\sum_{i=1}^{N} \text{TotalVar}\left(\mathbb{P}^i, \widehat{\mathbb{P}}^i\right) \leq \epsilon_{\text{sup}}$.

Guarantee: For any $\delta \in (0,1)$, with pr. greater than $1-\delta$, for all $\pi \in \Pi$:

$$|V_T(\pi) - \hat{V}_T(\pi)| \leq T\left(\epsilon_{\text{sup}} + \sqrt{\frac{\log(K/\delta)}{N}}\right)$$

- Key idea: We can backtest even in the censored setting!

III: Training Policies & Empirical Results

The Simulator

- Collection of historical trajectories:
  - 1 million products
  - 104 weeks of data per product

Data

- Uncensoring:
  - Demand
  - Vendor Lead Times

Corrections

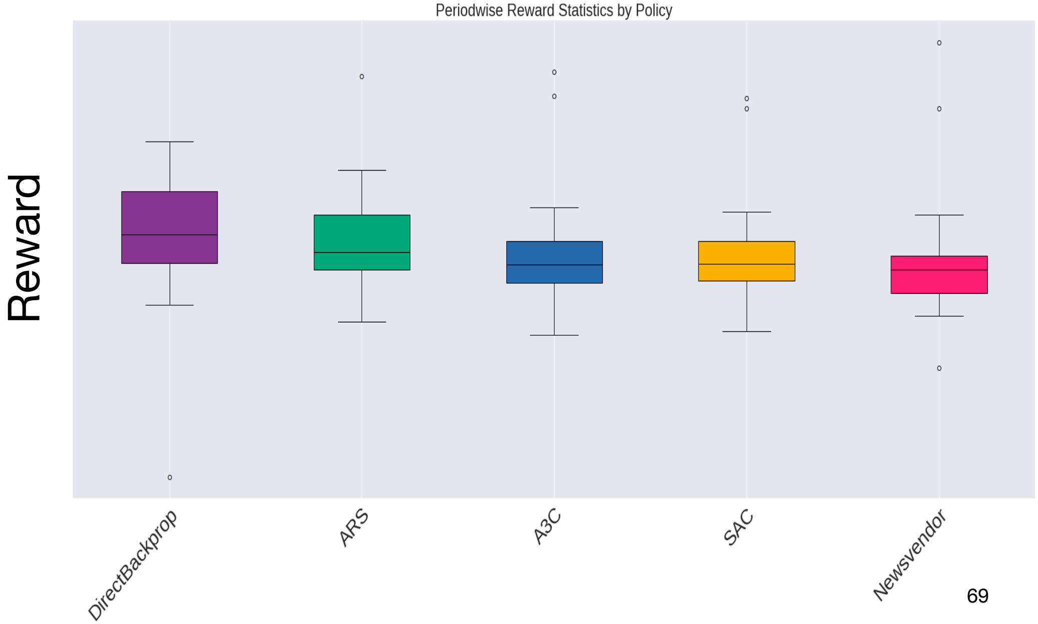
- **Policy gradient methods (PPO) in a "gym"**:
  - "gym" ↔ backtesting ↔ simulator
    (note the "simulator" isn't a good world model).
  - The policy can depend on many features.
    (seasonality, holiday indicators, demand history, ASIN, text features)

Simulator

Sim to Real Transfer

- Sim: the backtest of DirectBackprop improves on Newsvendor.
- Real: DirectBackprop significantly reduces inventory without significantly reducing total revenue.

Simulation



Real World

Metrics	% change
Inventory Level	**-12** $\pm$ 6
Revenue	$\sim$

Further RL Challenges for OR/Supply Chain

- World is not perfectly exogenous (some terms may depend on our actions)
- Cross product constraints are computationally intensive
- Not every Supply Chain problem can be written in this framework

Thanks!

Deep Learning + RL:

- Games (lookahead) & SelfPlay
- RL 4 alignment?
- RL in the “real world”

Figure Credits

https://en.wikipedia.org/wiki/Alpha–beta_pruning
https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168
https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
https://web.stanford.edu/~surag/posts/alphazero.html
https://spinningup.openai.com/en/latest/algorithms/ppo.html
https://openai.com/research/instruction-following#sample1
https://www.lesswrong.com/tag/coherent-extrapolated-volition
https://arxiv.org/pdf/2212.08073.pdf