Monte Carlo Tree Search

Noah Syrkis

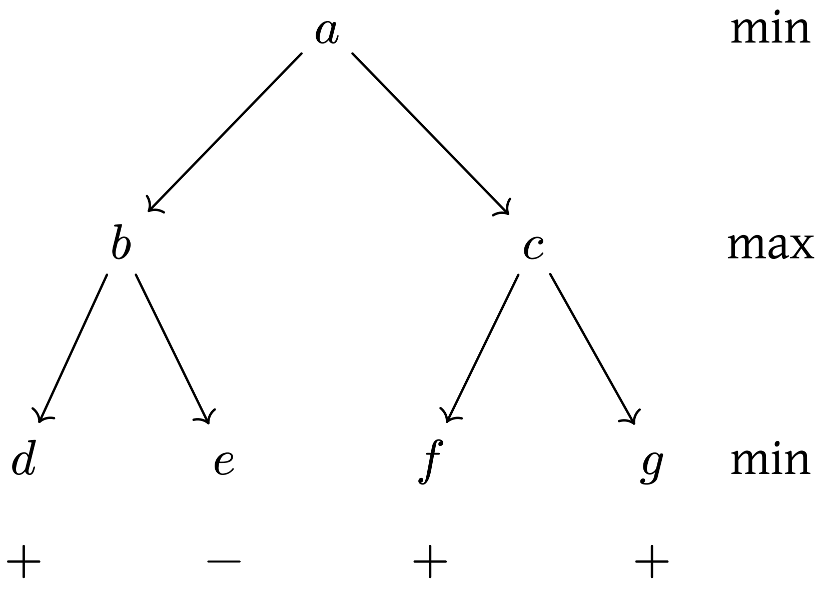
September 8, 2025

1 | Introduction

The future is a garden of forking paths [1]. Action a at state s_t yields a new state s_{t+1}. A different action a', however, might have yielded some different state s'_{t+1}.

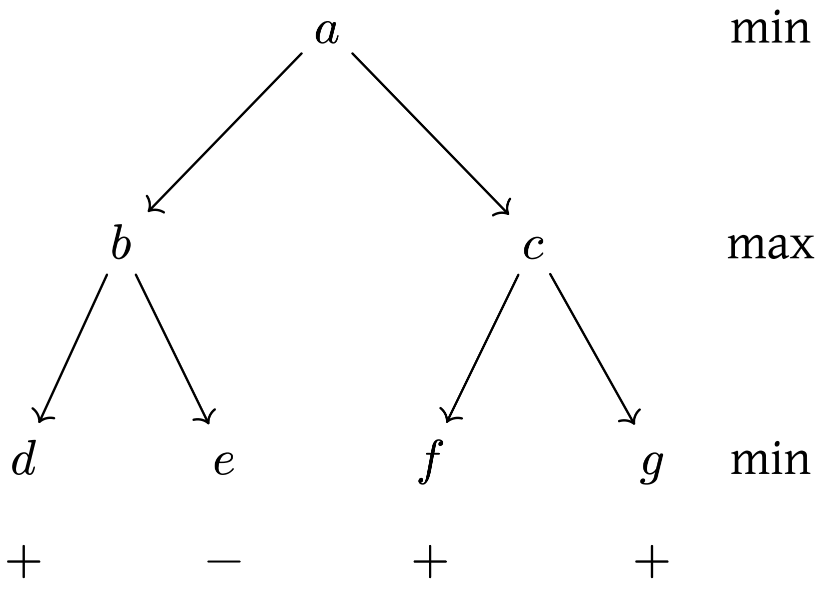
2 | Minimax

- Suppose we have a function that:
- given a state and an action returns a new state,
- and another that given a state returns who won
- What can we do?



2 | Minimax

- Suppose we have a function that:
- given a state and an action returns a new state,
- and another that given a state returns who won
- What can we do? Play perfectly and never loose



2 | Minimax

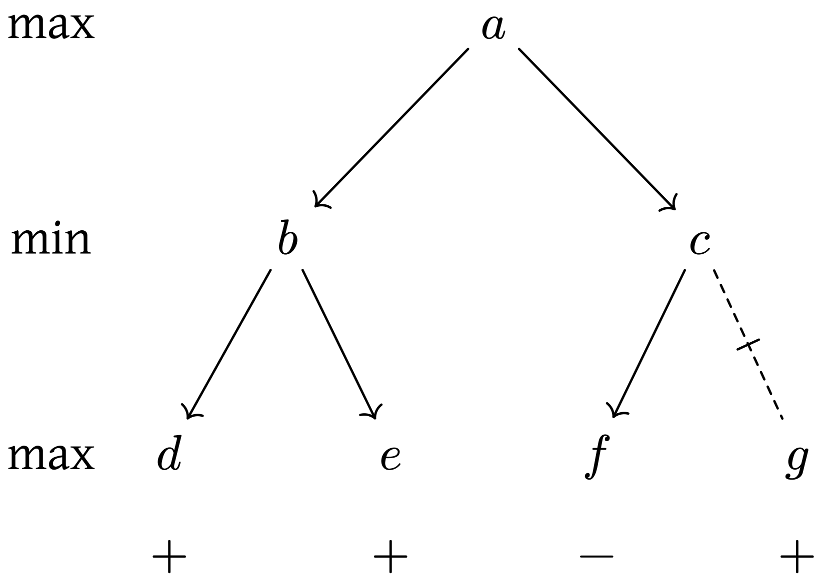
- We can win (or at least not loose) any game[1] by:
  1. Calling the minimax function for all actions
  2. Storing the values of each action in a list
  3. Taking the action with the highest value
- How can we do better? What are the issues?

Algorithm 1: minimax(state, maxim) $\to$ value

if node is terminal
  | **return** the value of node
temp = $-\infty$ if maxim else ∞
for each child of state
  | value = minimax(child, not maxim)
  | temp = (max if maxim else min)(temp, value)
return temp

[1] that is two player, winnable, deterministic, etc.

3 | $\alpha - \beta$ pruning



- Skip branches worse than current floor
- α and β refer to those precisely floors

3 | $\alpha - \beta$ pruning

- Algorithm 2 looks daunting but the idea is:
- Stop exploring paths you already know are bad

Algorithm 2: $\alpha - \beta$ pruning(node, maxim, α, β)

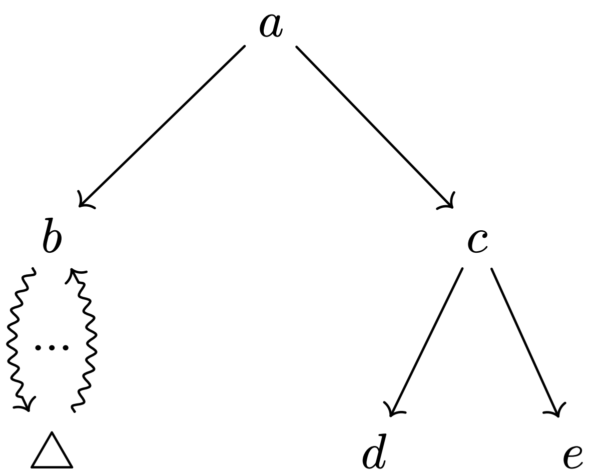
```
if node is terminal
    return the value of node
bestValue = −∞ if maxim else ∞
condition = max if maxim else min
for each child of node
    value = minimax(child, not maxim, α, β)
    bestValue = condition(bestValue, value)
    α = (condition(α, value) if maxim else α)
    β = (condition(β, value) if not maxim else β)
    if α >= β; break
return bestValue
```

4 | MCTS

- We haven't actually looked at the board
- Humans don't mentally finish n games

4 | MCTS

- Monte Carlo (random) tree search [2]
- Core idea: sample from bottom of each branch
- How much to sample from each branch?
- How should we reach the bottom?



4.1 | Explore / exploit

- When do we exploit the best tool we have?
- When should we explore for a new tool?
- There is a good entropy based solution [3]

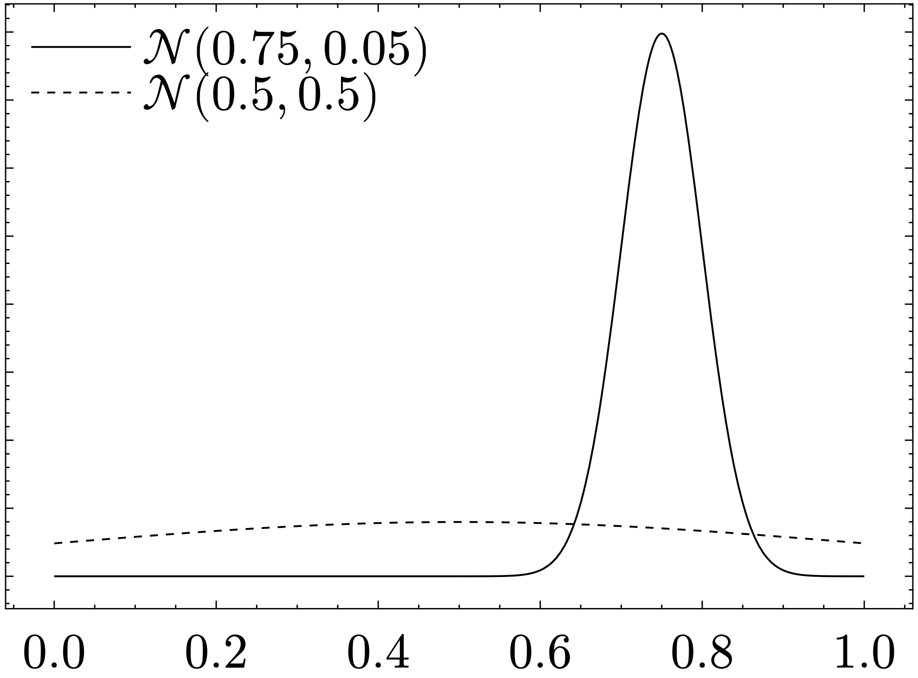


Figure 4: Which distribution would you sample from? Which is more likely to reach 1?

5 | Python

- You will see code that looks like Script 1
- In some games $s \neq o$, so we need seperate `obs`
- Multi player setup will have inner player loop

```
import gymnasium as gym
env = gym.game("tic_tac_toe")
state, done = env.init()

while not done:
  action = action_fn(state)
  state, done = env.step(state, action)
```

Script 1: Playing games in Python usually look something like this

5 | Python

- Some useful packages
- Understanding gymnasium is a must
- Get comfy with `.reset` and `.step`
- Sometimes `state` has a valid action mask!

`aigs`	package for our course
`gymnasium` [4]	Basic env package
`petting-zoo` [5]	gym for multiplayer

References

[1] J. L. Borges, "The Garden of Forking Paths," Ficciones. Grove Press, New York, 1962.

[2] C. B. Browne et al., "A Survey of Monte Carlo Tree Search Methods," IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1–43, Mar. 2012, doi: 10.1109/TCIAIG.2012.2186810.

[3] H. Robbins, "SOME ASPECTS OF THE SEQUENTIAL DESIGN OF EXPERIMENTS," 1952.

[4] M. Towers et al., "Gymnasium: A Standard Interface for Reinforcement Learning Environments." Mar. 2025.

[5] J. Terry et al., "Pettingzoo: Gym for Multi-Agent Reinforcement Learning," Advances in Neural Information Processing Systems, vol. 34, pp. 15032–15043, 2021.