Math

Derivative Rules

Chain rule: $\frac{d}{dx}f(u) = \frac{d}{du}f(u) \cdot \frac{d}{dx}u(x)$
Sum/difference rule: $(f \pm g)' = f' \pm g'$
Product rule: $(f \cdot g)' = f' \cdot g + f \cdot g'$
Quotient rule: $(\frac{f}{g})' = \frac{f' \cdot g - g' \cdot f}{g^2}$

Common Derivatives

$\frac{d}{dx}a^x = a^x \ln(a)$
$\frac{d}{dx}e^x = e^x$
$\frac{d}{dx}\log_a(x) = \frac{1}{x \ln(a)}$
$\frac{d}{dx}\ln(x) = \frac{1}{x}$

Games

Game tree - describes the possibilities of a game and models opponents & randomness. Each node is a decision point for a player; each root-to-leaf path is a possible outcome of the game. Legend: △ - maximizing node, ▽ - minimizing node, and ◯ - chance node

Two-player zero-sum game - Each state is fully observed and such that players take turns; utility of the agent is negative the utility of the opponent (so the sum of the two utilities is zero).

Players: $= \{\text{agent}, \text{opp}\}$
s_{start}: start state
Actions(s): possible actions from state s
Succ(s, a): resulting state if choose action a in state s
IsEnd(s): whether s is an end state
Utility(s): agent's utility for end state s
Player(s): player who controls the state s

Types of policies -

Stochastic policies: $\pi_p(s, a) \in [0, 1]$ probability of player p taking action a in state s.
Deterministic policies: $\pi_p(s) \in \text{Actions}(s)$ action that player p takes in state s. A (special) instance of Stochastic policies.

Game evaluation - analogous to recurrence for policy evaluation in MDPs. $V_{\text{eval}}(s) =$

$$\begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \sum_{a \in \text{A}(s)} \pi_{\text{ag}}(s, a) V_{\text{eval}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{ag} \\ \sum_{a \in \text{A}(s)} \pi_{\text{op}}(s, a) V_{\text{eval}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{op} \end{cases}$$

As the agent, we want to solve $\pi_{\text{agent}}(s, a)$: the best thing we should do.

Expectimax - $V_{\text{exptmax}}(s)$ is the max expected utility of any agent policy when playing w.r.t. a *fixed and known* π_{opp}. $V_{\text{exptmax}}(s) =$

$$\begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{A}(s)} V_{\text{e-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{ag} \\ \sum_{a \in \text{A}(s)} \pi_{\text{op}}(s, a) V_{\text{e-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{op} \end{cases}$$

$\Rightarrow \pi_{\text{exptmax}(7)}, \pi_7$ (assuming the fixed opponent policy π_{opp} is π_7, then the the best policy computed by expectimax recurrence for agent is denoted as $\pi_{\text{exptmax}(7)}$).

Minimax - Find an optimal agent policy against an adversary by assuming the worst case: the opponent does everything to minimize the agent's utility. $V_{\text{minimax}}(s) =$

$$\begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{A}(s)} V_{\text{m-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{ag} \\ \min_{a \in \text{A}(s)} V_{\text{m-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{op} \end{cases}$$

$\Rightarrow \pi_{\text{max}}, \pi_{\text{min}}$:
$\pi_{\text{max}}(s) = \arg\max_{a \in \text{A}(s)} V_{\text{minimax}}(\text{Suc}(s, a))$
$\pi_{\text{min}}(s) = \arg\min_{a \in \text{A}(s)} V_{\text{minimax}}(\text{Suc}(s, a))$

Minimax properties - we can play an agent policy π_{agent} against an opponent policy π_{opp}, which produces an expected utility via game evaluation, denoted as $V(\pi_{\text{agent}}, \pi_{\text{opp}})$

1. if the agent were to change its policy from π_{max} to any π_{agent}, then the agent wouldn't be better off (and in general, worse off).

$$\boxed{\forall \pi_{\text{agent}}, V(\pi_{\text{max}}, \pi_{\text{min}}) \geq V(\pi_{\text{agent}}, \pi_{\text{min}})}$$

2. if the opponent were to change its policy from π_{min} to any π_{opp}, then the opponent wouldn't be better off (the value of the game can only increase, which is favorable to the agent).

$$\boxed{\forall \pi_{\text{opp}}, V(\pi_{\text{max}}, \pi_{\text{min}}) \leq V(\pi_{\text{max}}, \pi_{\text{opp}})}$$

From the agent's point of view, this can be interpreted as guarding against the worst case $\Rightarrow$ If $V_{\text{minimax}}(s) = 1$, the agent is guaranteed at least a value of 1 no matter what the opponent does.

3. if the opponent is known to be not adversarial, then the minimax policy might not be optimal for the agent.

$$\boxed{\text{For } \pi_7,\ V(\pi_{\text{max}}, \pi_7) \leq V(\pi_{\text{exptmax}(7)}, \pi_7)}$$

$$\begin{aligned} V(\pi_{\text{exptmax}(7)}, \pi_{\text{min}}) &\leq V(\pi_{\text{max}}, \pi_{\text{min}}) \\ &\leq V(\pi_{\text{max}}, \pi_{\text{opp}}) \\ &\leq V(\pi_{\text{exptmax}(7)}, \pi_7) \end{aligned}$$

Expectiminimax - **Players**: $= \{\text{agent}, \text{opp}, \text{coin}\}$: a third player representing any sort of natural randomness (metaphorically "coin") is introduced which always follows a known stochastic policy. $V_{\text{exptminmax}}(s) =$

$$\begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{A}(s)} V_{\text{e-m-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{ag} \\ \min_{a \in \text{A}(s)} V_{\text{e-m-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{op} \\ \sum_{a \in \text{A}(s)} \pi_{\text{co}(s,a)} V_{\text{e-m-m}}(\text{Suc}(s, a)) & \text{Player}(s) = \text{co} \end{cases}$$

Speeding up minimax

Depth-limited tree search - Stop at maximum depth d_{max}. Use: at state s, call $V_{\text{minmax}}(s, d_{\text{max}})$. Convention: decrement depth at last player's turn. $V_{\text{minmax}}(s, d) =$

$$\begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \text{Eval}(s) & d = 0 \\ \max_{a \in \text{A}(s)} V_{\text{e-m-m}}(\text{Suc}(s, a), d) & \text{Player}(s) = \text{ag} \\ \min_{a \in \text{A}(s)} V_{\text{e-m-m}}(\text{Suc}(s, a), d - 1) & \text{Player}(s) = \text{op} \end{cases}$$

Evaluation function - a domain-specific and possibly very weak estimate of the value $V_{\text{minmax}}(s)$, analogous to $A^\star$'s FutureCost(s) but unlike $A^\star$ no guarantees on the error from approximation.

Depth-limited exhaustive search - $O(b^{2d})$ time. Still not ideal.

Optimal path - path that minimax policies take. Values of all the nodes on path are the same.

Alpha-beta pruning - a domain-general exact method optimizing the minimax algorithm by avoiding the unnecessary exploration of parts of the game tree. To do so, each player keeps track of the best value they can hope for (stored in α for the maximizing player and in β for the minimizing player). At a given step, $\beta < \alpha \Rightarrow$ the optimal path is not going to be in the current branch as the earlier player had a better option at their disposal.

Order matters:

- Worst ordering: $O(b^{2d})$ time
- Best ordering: $O(b^{2 \cdot 0.5d})$ time
- Random ordering: $O(b^{2 \cdot 0.75d})$ time when $b = 2$

In practice, can use Eval(s):

- on a max node, order successorts by decreasing Eval(s')
- on a min node, order successorts by increasing Eval(s')