Math

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: $(\frac{f}{g})' = \frac{f' \cdot g - g' \cdot f}{g^2}$

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

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

$\boxed{\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1} = \sigma(z)(1-\sigma(z))}$

$\boxed{\frac{d}{dx}\tanh(x) = \frac{d}{dx}\frac{e^x - e^{-x}}{e^x + e^{-x}} = 1 - \tanh^2(x)}$

Search

Definition - by doing action a from state s, we deterministically arrive in state $\text{Succ}(s,a)$. Goal: determine a sequence of actions between s_{start} and an end state on a minumum cost path.

- S_{start}
- $\text{IsEnd}(s)$: whether an end state was reached
- $\text{Actions}(s)$: possible actions from state s
- $\text{Cost}(s,a)$: action cost
- $\text{Succ}(s,a)$ of state s after action a

State - a summary of all past actions sufficient to choose future actions optimally.

- **Explored**: states for which the optimal path has been found
- **Frontier**: states seen for which we are still figuring out how to get there with the cheapest cost
- **Unexplored**: states not yet seen

Tree search

Algorithms - b: # actions per state; d: solution depth, and D: maximum depth.

Algo	Cost	Space	Time
Backtracting	any	$O(D)$	$O(b^D)$
BFS	$c \geq 0$	$O(b^d)$	$O(b^d)$
DFS	0	$O(D)$	$O(b^D)$
DFS-ID	$c \geq 0$	$O(d)$	$O(b^d)$

Graph search

Dynamic programming (DP) - a Backtracting search algorithm that only works for acyclic graphs. $\text{FutureCost}(s) = \min_{a \in A(s)} [\text{Cost}(s,a) + \text{FutureCost}(\text{Succ}(s,a))]$ or 0 if at end state.

Uniform cost search (UCS) - explores states s in increasing order of $\text{PastCost}(s)$, assuming **all action costs are non-negative**. Adding a positive constant to all costs would make a *different* problem.

Algorithms - N: # total states; n # states explored before s_{end}.

Algo	Cycle?	Cost	Time/space
DP	No!	any	$O(N)$
UCS	Ok	$c \geq 0$	$O(n \log(n))$

Markov decision processes

Find the maximum value policy by using MDPs that help us cope with randomness and uncertainty, in order to find our way between an initial state and an end state.

Definition -:

- s_{start}, $\text{Actions}(s)$, $\text{IsEnd}(s)$
- $\text{Reward}(s,a,s')$
- $T(s,a,s')$: $\forall s, a, \sum_{s' \in S} T(s,a,s') \equiv 1$
- Discount: *(living in the moment "greedy algo")* $0 \leq \gamma \leq 1$ *(save for the future)*

Search is a special case of MDP where

$$T(s,a,s') = \begin{cases} 1 & s' = \text{Succ}(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Policy - π is a function that maps each state s to an action $a \in \text{Actions}(s)$. Following a policy yields a random path.

Utility (of a policy π / path) - the (discounted) sum of the rewards on the path, making it a random variable. $\text{u}(\pi) = \sum_{i=0}^{\infty} \gamma^i \text{Reward}(s_i, \pi(s_i), s_i + 1)$

Value (of a policy π at state s_0) - the expected utility received by following policy π from state s over random paths; randomness comes from $T(s,a,s')$ and possibly π. $V_\pi(s) = Q_\pi(s, \pi(s))$

Q-value - the expected utility of a "chance node" (taking action a from state s and then following π). $Q_\pi(s,a) =$

$$\underbrace{\sum_{s'} T(s,a,s')\text{Reward}(s,a,s')}_{\text{exp'd rwd of taking (s, a)}} + \underbrace{\sum_{s'} T(s,a,s')\gamma V_\pi(s')}_{\text{(discntd) exp'd future rwd}}$$

$V_\pi(s) = 0$ if $\text{IsEnd}(s)$; otherwise a recurrence:

$$\boxed{\begin{aligned} &V_\pi(s) = Q_\pi(s, \pi(s)) \\ &= \sum_{s'} T(s, \pi(s), s') \left[\text{Reward}(s, \pi(s), s') + \gamma V_\pi(s')\right] \end{aligned}}$$

Algorithms

Policy evaluation - iteratively computes V_π (given a specific policy π).

- **Initialization**: $\boxed{\forall s, V_\pi^{(0)}(s) \leftarrow 0}$
- **Iteration** $t = 1, \ldots, T_{PE}$:

  $\forall s$ (for each state), $\boxed{V_\pi^{(t)}(s) \leftarrow Q_\pi^{(t-1)}(s, \pi(s))}$ with

$$\boxed{\begin{aligned} Q_\pi^{(t-1)}(s, \pi(s)) &= \sum_{s'} T(s, \pi(s), s') \\ &\left[\text{Reward}(s, \pi(s), s') + \gamma V_\pi^{(t-1)}(s')\right] \end{aligned}}$$

- until values don't change much: $\boxed{\max_{s \in S} |V_\pi^{(t)}(s) - V_\pi^{(t-1)}(s)| \leq \epsilon}$ (error tolerance)

Time complexity $O(T_{PE}|S||S'|)$ ($|S|$: # of states; $|S'|$: # of s' with $T(s,a,s') > 0$)

Optimal Q-value (of state s with action a) - the maximum Q-value attained by any policy taken after taking action a from state s.

$$\boxed{\begin{aligned} Q_{opt}(s,a) &= \sum_{s'} T(s,a,s') \\ &\left[\text{Reward}(s,a,s') + \gamma V_{opt}(s')\right] \end{aligned}}$$

Optimal value (of state s) - the maximum value attained by any policy. $V_{opt}(s) = 0$ if $\text{IsEnd}(s)$; otherwise

$$\boxed{V_{opt}(s) = \max_{a \in A(s)} Q_{opt}(s,a)}$$

Optimal policy (of state s) - ("opt" is still a policy!) the policy that leads to the optimal values. $\forall s$ $\boxed{\pi_{opt}(s) = \arg\max_{a \in A(s)} Q_{opt}(s,a)}$

Value iteration (off-policy) - finds $V_{opt}(s)$ and therefore $\pi_{opt}(s)$.

- **Initialization**: $\boxed{\forall s, V_{opt}^{(0)}(s) \leftarrow 0}$
- **Iteration** $t = 1, \ldots, T_{VI}$:

  $\forall s$ (for each state),

  $\boxed{V_{opt}^{(t)}(s) \leftarrow \max_{a \in A(s)} Q_{opt}^{t-1}(s,a)}$ with

$$\boxed{\begin{aligned} Q_{opt}^{t-1}(s,a) &= \sum_{s'} T(s,a,s') \\ &\left[\text{Reward}(s,a,s') + \gamma V_{opt}^{(t-1)}(s')\right] \end{aligned}}$$

- VI convergence guaranteed by **either** $\gamma < 1$ **or the MDP graph being acyclic**.

Time complexity $O(T_{VI}|S||A||S'|)$ ($|S|$: # of states; $|A|$: # of actions per state; $|S'|$: # of s' with $T(s,a,s') > 0$)

Reinforcement Learning

Dealing with unknown T and Rewards.

MDPs (offline) - have a mental model of the world; find π_{opt} to maximize rewards collected.

RL (online) - don't know how the world works; perform actions to find out and collect rewards.

On-policy - estimate the value of a data-generating (exploration) policy (usually to get Q_π).

Off-policy - estimate the value of another policy (usually to get Q_{opt}).

Model-based Monte Carlo (off-policy) - estimates $T(s,a,s')$ and $\text{Reward}(s,a,s')$ using findings from *randomly* (i.e. no policy-following) traversing the space:

$\boxed{\hat{T}(s,a,s') = \frac{\text{\# times } (s,a,s') \text{ occurs}}{\text{\# times } (s,a) \text{ occurs}}}$ and

$\boxed{\hat{\text{Reward}}(s,a,s') = r \text{ in } (s,a,r,s')}$. $\hat{T}\hat{R}$ can then be used to deduce Q-values ($\hat{Q}_\pi$ and $\hat{Q}_{opt}$).

Model-free Monte Carlo (on-policy) - directly estimates Q_π.

$\boxed{\hat{Q}_\pi(s,a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a}$

(u_t: the utility starting at step t of a given episode). The estimated value is dependent on the policy π used to generate the data.

Convex combination formula - for each (s,a,u) of the training set and by introducing $\boxed{\eta = \frac{1}{1 + (\text{\# updates to } (s,a))}}$:

$$\boxed{\hat{Q}_\pi(s,a) \leftarrow \hat{Q}_\pi(s,a) - \eta \left[\underbrace{\hat{Q}_\pi(s,a)}_{\text{prediction}} - \underbrace{u}_{\text{target}}\right]}$$

Issue: reaching state-action pair (s_{t-1}, a_t) required the sum until termination ($u_t = \sum_{i=0}^{\infty} \gamma^i r_{t+1}$) just for a signle update.

SARSA (on-policy) - estimate Q_π by using both raw data (observed r) and prediction (estimate of $\hat{Q}_\pi$) as part of the update rule. For each tuple (s,a,r,s',a') in the sequence of exploration via π

$$\boxed{\begin{aligned} &\hat{Q}_\pi(s,a) \leftarrow \hat{Q}_\pi(s,a) \\ &- \eta \left[\underbrace{\hat{Q}_\pi(s,a)}_{\text{prediction}} - \underbrace{r}_{\text{data}} + \gamma \underbrace{\hat{Q}_\pi(s',a')}_{\text{estimate}}\right] \end{aligned}}$$

SARSA estimate is updated on the fly as opposed to the model-free MC estimate where the estimate can only be updated at the end of the episode. However if rewards only exist at the terminal state, SARSA updates would be slower than Model-free Monte Carlo.

Q-learning (off-policy) - produces an estimate for Q_{opt}. For each (s,a,r,s',a'):

$$\boxed{\begin{aligned} &\hat{Q}_{opt}(s,a) \leftarrow \hat{Q}_{opt}(s,a) \\ &-\eta \left[\underbrace{\hat{Q}_{opt}(s,a)}_{\text{prediction}} - \underbrace{(r + \gamma \max_{a' \in A(s')} \hat{Q}_{opt}(s',a'))}_{\text{target}}\right] \end{aligned}}$$

Exploration vs Exploitation -

- Too greedy (always picking the best action): won't explore everywhere
- Too much exploring: learn too slowly

Epsilon-greedy policy - an algorithm that balances exploration with probability ϵ and exploration with probability $1 - \epsilon$. For a given state s, the policy is computed as:

$$\boxed{\pi_{act}(s) = \begin{cases} \arg\max_{a \in A(s)} \hat{Q}_{opt}(s,a) & 1 - \epsilon \\ \text{random action from } A(s) & \epsilon \end{cases}}$$

Q-learning with function approximation - avoid memorizing every state as $\hat{Q}_{opt}(s,a)$ just maps (s,a) to a value estimate so define $\hat{Q}_{opt}(s,a;\mathbf{w}) = \mathbf{w} \cdot \phi(s,a)$ where the learned $\mathbf{w}$

can replace the mapping table. For each (s,a,r,s',a'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left[\underbrace{\hat{Q}_{opt}(s,a;\mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \max_{a' \in A(s')} \hat{Q}_{opt}(s',a';\mathbf{w}))}_{\text{target}}\right] \phi(s,a)$$

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: $\triangle$ - maximizing node, $\bigtriangledown$ - minimizing node, and $\bigcirc$ - 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}, Actions(s), Succ(s,a), IsEnd(s)
- 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. $V_{\text{eval}}(s) =$

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

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}. $V_{\text{exptmax}}(s) =$

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

$\Rightarrow \pi_{\text{exptmax}(7)}, \pi_7$ (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. $V_{\text{minimax}}(s) =$

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

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

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 $\pi_{\max}$ to any π_{agent}, then the agent wouldn't be better off (and in general, worse off).
   $\boxed{\forall \pi_{\text{agent}}, V(\pi_{\max}, \pi_{\min}) \geq V(\pi_{\text{agent}}, \pi_{\min})}$
2. if the opponent were to change its policy from $\pi_{\min}$ 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).
   $\boxed{\forall \pi_{\text{opp}}, V(\pi_{\max}, \pi_{\min}) \leq V(\pi_{\max}, \pi_{\text{opp}})}$
   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.
   $\boxed{\text{For } \pi_7,\ V(\pi_{\max}, \pi_7) \leq V(\pi_{\text{exptmax}(7)}, \pi_7)}$
   $$\boxed{\begin{aligned} V(\pi_{\text{exptmax}(7)}, \pi_{\min}) &\leq V(\pi_{\max}, \pi_{\min}) \\ &\leq V(\pi_{\max}, \pi_{\text{opp}}) \leq V(\pi_{\text{exptmax}(7)}, \pi_7) \end{aligned}}$$

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. $V_{\text{exptminmax}}(s) =$

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

Speeding up minimax

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

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

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

Temporal difference (TD) learning - picks a piece of experience (s,a,r,s') and updates $\mathbf{w}$. Used when we don't know the transitions / rewards. The value is based on exploration policy.
Evaluation function could be hand-crafted but also learned from data:
Eval$(s) = V(s;\mathbf{w}) = \mathbf{w} \cdot \phi(s)$ (linear).

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left[\underbrace{\hat{V_\pi}(s;\mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V_\pi}(s';\mathbf{w}))}_{\text{target}}\right] \nabla_{\mathbf{w}} \hat{V_\pi}(s;\mathbf{w})$$

For linear functions: $V(s;\mathbf{w}) = \mathbf{w} \cdot \phi(s)$; $\nabla_{\mathbf{w}} V(s;\mathbf{w}) = \phi(s)$.

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} - \eta\, [\mathbf{w} \cdot \phi(s) - (r + \gamma \mathbf{w} \cdot \phi(s'))]\, \phi(s)}$$

Feature selection: how good my "board" is.

TD learning vs Q-learning - Q-learning operates on $\hat{Q}_{\text{opt}}(s,a;\mathbf{w})$, off-policy (value based on estimate of optimal policy), and doesn't need to know MDP transitions $T(s,a,s')$. **TD learning**: operates on $\hat{V_\pi}(s;\mathbf{w})$, on-policy (value is based on exploration policy), and **needs to know rules of the game Succ**(s,a).

Simultaneous games

On the contrary of turn-based games, no ordering on the player's moves in simultaneous games.

Single-move simultaneous game - Players $= \{A, B\}$ with given possible actions. $V(a,b)$: **A's utility** if A chooses action a and B chooses action b. **Payoff matrix**: $V \in |\text{Actions}|^2$.

Pure strategy - just a single action: $\boxed{a \in \text{Actions}}$. **Mixed strategy -** a probability distribution over actions: $\boxed{\forall a \in \text{Actions}, 0 \leq \pi(a) \leq 1}$. Examples:

- Fixed, always show "1": $\pi = [1, 0]$
- Fixed, always show "2": $\pi = [0, 1]$
- Mixed, uniformly random: $\pi = [\frac{1}{2}, \frac{1}{2}]$

Game evaluation - the value of the game if player A follows π_A and player B follows π_B:
$\boxed{V(\pi_A, \pi_B) = \sum_{a,b} \pi_A(a) \pi_B(b) V(a,b)}$

von Neumann minimax theorem - for every simultaneous two-player zero-sum game with a finite number of actions:
$\boxed{\max_{\pi_A} \min_{\pi_B} V(\pi_A, \pi_B) = \min_{\pi_B} \max_{\pi_A} V(\pi_A, \pi_B)}$
where π_A, π_B range over **mixed strategies**.

Non-zero games

Payoff matrix - utility for player p: $V_p(\pi_A, \pi_B)$.
Nash equilibrium - $(\pi_A^\star, \pi_B^\star)$ such that no player has an incentive to change their strategy:
$\boxed{\forall \pi_A, V_A(\pi_A^\star, \pi_B^\star) \geq V_A(\pi_A, \pi_B^\star)}$ and $\boxed{\forall \pi_B, V_B(\pi_A^\star, \pi_B^\star) \geq V_B(\pi_A^\star, \pi_B)}$. **Nash's existence theorem -** in any finite-player game with finite number of actions, there exists **at least one** Nash equilibrium.