Search

First Try: Reflex Agent (only based on memory, prediction X)
↳ can be rational if needing quick decisions

Second Try: Planning Agent (decision based on possible consequences)
↳ completeness (gives an answer?), optimality (best answer?)
↳ a "replanning" agent solves the problem on-the-fly.

Search Problems consist of:
- state space
- successor function (actions & costs)
- start state & goal test

→ a solution is a sequence of actions that transforms the start state into an end state

World state VS Search state (abstraction)
→ things that don't change or don't matter for the solution don't need to be in the search state

State Space Graph: mathematical representation of search problem

↳ nodes are world state, arrows are successors.

Search Tree: encodes possible decisions as a chonological tree

Tree Search: expand on tree nodes, order matters!

↳ uses fringe, expansion, and exploration strategy

↳ main question: which fringe nodes to explore?

Search Strategies:

Depth-First: expand the deepest node first.

↳ expands some left prefix, $O(b^m)$ time for finite tree, only stores siblings from path to root → space $O(bm)$

↳ not complete if infinite tree, not optimal and only finds the "left most" solution

Breadth-First: expand the shallowest node first

↳ expands all nodes above the shallowest solution

↳ time $O(b^s)$, space $O(b^s)$

↳ Complete, optimal iff all costs are 1.

Iterative Deepening: Combine DFS & BFS
↳ Run DFS with depth limit increasing iteratively.
↳ ordering is BFS-like, but saves memory
↳ the "last layer" out costs the previous iterations, so asymptotics isn't that bad.

Uniform Cost: explore "cheap" paths first

Informed Search

Heuristic: a function that estimates how close a state is to a goal
ex) Manhattan Distance, Euclidean Distance

Greedy Search: Only look at the lowest heuristic
↳ Similar to DFS, but only considers future costs.

A* Search: Combines UCS and Greedy ($f(n) = g(n) + h(n)$)
Is A* optimal? → can fail if too pessimistic (trapped)

Admissible heuristics: always underestimates cost to goal
↳ heuristic h is admissible if $0 \leq h(n) \leq h^*(n)$ where h^* is the true cost function.

"If A is an optimal goal, B is a suboptimal goal, h is admissible, A will exit the fringe before B."

→ Proof: Imagine B is on the fringe.

Some ancestor of A (n) is in the fringe, too.

Then, n will be expanded before B

↳ $f(n) \leq f(A)$ by admissibility

$f(A) \leq f(B) \rightarrow \underline{f(n) \leq f(B)}$ //

With the same argument, all ancestors of A are expanded before B! //

How to design admissible heuristics?

↳ often relaxing constraints work (ex) Manhattan)

Heuristics should be informative, but not too costly to compute

* maximum of admissible heuristic is still admissible.

Graph Search: don't expand the same state twice.

↳ however, if the newly computed cost is better than the previously stored cost, expand it again.

→ still optimal

* If a heuristic is consistent, the first expansion ensures optimality for that node. (not covered)

CSP

"What is the best assignment to variables?"

Standard Search: state is a black box, goal & successors can be anything

CSP: State is defined by variables X with values from domain D.

↳ the goal test is a set of constraints of allowable assignments

ex) map coloring without adjacent states sharing colors

↳ variables: regions $\{R_1, R_2, \dots, R_n\}$. domain: colors {red, green, blue}

constraint: $(R_1 \neq R_2)$ (implicit), $(R_1, R_2) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), \dots\}$ (explicit)

solution: assignment satisfying all constraints

Binary CSP: all constraints take at most two variables

ex) N-Queens: variables $\{X_{ij} \mid i \in \#\text{rows}, j \in \#\text{columns}\}$

domain $\{0, 1\}$. constraints? $\forall i,j,k\ (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$ column

row $\forall i,j,k\ (X_{ij}, X_{kj}) \in \{(0,0), (1,0), (0,1)\}$, diagonal constraints ...,

also need to include $\sum_{i,j} X_{ij} = N$ to prevent trivial solution of all zeroes.

ex2) Different N-Queens formulation: Variable Q, domain {1 ... N}

↳ assign a queen in each row and assign column #s.

Varieties of CSPs:

Discrete/Continuous, Finite/Infinte domains, Unary/Binary/Higher Order Constraints, Soft Constraints (Preferences)

How to Solve CSPs: Standard Search Formulation?

Start with empty assignment, successor to assign a single variable

↳ BFS would be ineffective since the solution lives in the deepest layer!

↳ DFS works, but naively checking solutions doesn't check for early fails

Backtracking: $\underbrace{\text{One variable at a time}}_{\text{(variable ordering)}}$, $\underbrace{\text{check constraints on the fly}}_{\text{(fail-on-violation)}}$.

↳ strategies: ① filtering (detecting failures early) ② ordering (advantageous order?)

Filtering: Forward Checking - cross off violations when adding a variable to an existing assignment → exit on impossible variable

↳ however, it doesn't fail until the actual impossible assignment.

also, it only enforces constraints on the variable just assigned.

→ Contraint Propagation: reason from constraint to constraint

Arc consistency: An arc $X \rightarrow Y$ is consistent iff for every x in the tail, there is some y in the head which could be assigned without violating a constraint.

If an arc is inconsistent, remove an assignment from the tail such that the arc is now consistent.
If a tail is removed, check all arcs that had it as head need to be updated.
Detect early failure if a variable has no possible assignments
↳ Runtime: $O(n^2 d^3)$, can be reduced to $O(n^2 d^2)$.
However, detecting all future problems is NP-Hard.

Arc consistency only enforces constraints on pairs → needs backtracking

→ k=2 is arc consistency
K-consistency: for any k nodes, any assignments to (k-1) of the nodes can be "extended" to the last node
↳ "extended": there exists a valid assignment given other assignments
Strong n-consistency ensures a solution to a CSP.
↳ all of 1~(n-1) are consistent

Ordering: How to pick the variable/assignment to try next?

Variable ordering: Minimum Remaining Values (MRV)

↳ try variables with the fewest elements left in its domain

"fail fast" ordering, tackle the hardest subproblems first

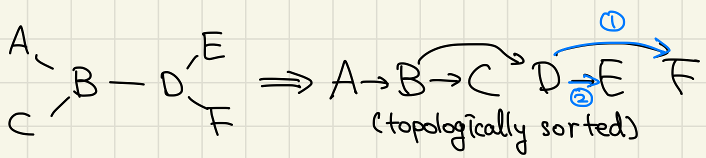
Value ordering: Least Constraining Value (LCV)

↳ given a choice of variable, choose the value that rules out the fewest values in remaining variables.

↳ being optimistic that the easiest path is correct

Reducing Structures: Disconnected graphs → independent subproblems

↳ If the constraint graph is a tree, CSP is solved in $O(nd^2)$ time

ex) 

Starting from the sink node, enforce arc consistency backwards checking

edges with it as head are unchecked yet

↳ there are no multiple checks following a removal → $O(n \cdot d^2)$ nodes

then, just pick variables from the source. it is ensured to give a solution (arc consistency ensures a valid assignment for all edges)

↳ no backtracking required! (infact, this is why NNs are DAGs)

If the CSP is not a tree, how about enforcing into one?

↳ Identify a cutset s.t. the remaining variables form a tree

⇒ improvement in runtime from exponential to (kind of) poly.

↳ try to cut out as little as possible when forming a tree

Iterative Improvement: start with some assignment, and improve inconsistent variables <u>locally</u> and greedily, such that the reassignment minimizes the # of remaining inconsistencies.

Difficulty of CSP: $R = \frac{\text{\# of constraints}}{\text{\# of variables}}$ → hard when not extreme

↳ R is big → almost trivial, R is small → large solution space

Local Search: no fringe, faster & effecient but incomplete & suboptimal

Adversarial Search

"How to choose actions in the presence of other agents?"

Types of games: Zero-sum (agents have opposite utilities), General games (independent utilities) → cooperation? indifference?

Deterministic/Stochastic? # of players? Perfect information?

⇒ build a <u>strategy</u> to recommend an action based on current state

Adversarial Games: Deterministic, 2-player, zero sum, perfect information

- States: S (starts at S_0)
- Players: $P = \{MAX, MIN\}$
- Actions: A (depends on player/state)
- Transition Function: $S \times A \to S$
- Terminal Test: $S \to \{T, F\}$
- Terminal Utilities: $S \to R$ (reward = score)

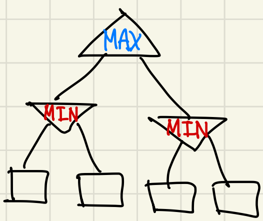
A <u>value</u> of a state := best achievable outcome (=utility) from that state

↳ for non-terminal states: $V(s') = \max_{s \in successor(s')} V(s)$

A state is terminal when its value is (presumed to be) known

Minimax: when the adversary chooses, they try to <u>minimize</u>

↳ under opponent's control: $V(s') = \min_{s \in successor(s')} V(s)$




```
def max/min-value(V):
    V ← -(+) ∞
    for each successor of V:
        V = max/min (V, min/max-value(successor))
    return V
```

min and max are counterparts

def value(V):

call min-value or max-value depending whose turn it is

Minimax will be optimal against a perfect opponent. Otherwise?

↳ imperfect opponent → different modeling (Expectimax)

Efficiency: $\simeq O(DFS) \rightarrow$ time $= O(b^m)$, space $= O(bm)$

↳ not realistic in most game scenarios

Game Tree Pruning: Can we not traverse every single subtree?

↳ Intuition: once we see a value less than the current max value, stop for that branch since the minimizer will return it or something even worse (the maximizer never chooses that branch)

→ pass the current rolling $\underbrace{\text{"maximum min value"}}_{\alpha}$ to min-value.

min-value stops exploring when $\underbrace{\text{its value}}_{n}$ drops below it $(n < \alpha)$.

(max-value version is symmetric) ⇒ Alpha-Beta Pruning

def $\frac{\text{max}}{\text{min}}$-value($V, \alpha, \beta$):

↳ no effect on minimax value for the root

however, intermediate values have different values

$V \leftarrow -(+)\infty$

for each successor of V:

$V = \frac{\max}{\min}(V, \text{value}(\text{successor}, \alpha, \beta))$

if $V \geq \beta \ (\leq \alpha)$, return V

$\frac{\alpha}{\beta} = \frac{\max}{\min}(\frac{\alpha}{\beta}, V)$

Good child ordering improves pruning efficiency!

With "perfect ordering", time drops to $O(b^{m/2})$.

↳ this doubles solvable depth!

Depth-Limited Search: Just stop and appoximate after some depth

↳ an evaluation function guesses the utility of a state
→ similar to heuristics in A* search!

Not guaranteed optimal play anymore, but use iterative deepening for flexibility when computing

Eval(s) is usually a linear combination of game features

A bad evaluation function can cause an infinite loop...

Markov Decision Processes

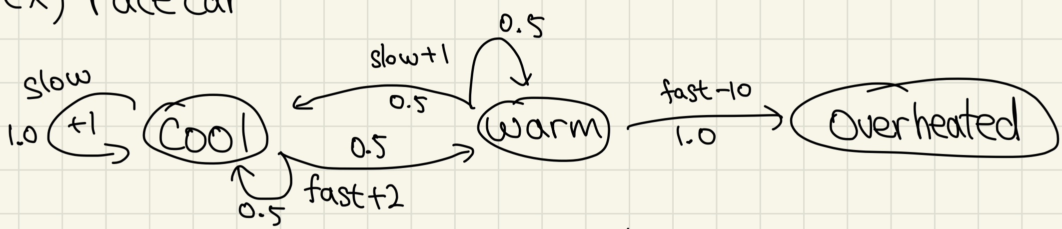
- A set of states $s \in S$, A set of actions $a \in A$
- Transition function $T(s,a,s') := P(s'|s,a)$
- Reward function $R(s,a,s')$ (sometimes just $R(s)$ or $R(s')$)
- A start state, maybe a terminal state

"Markov"ness: action outcomes only depend on <u>current</u> state

For an MDP, we want a policy $\pi^*: S \to A$

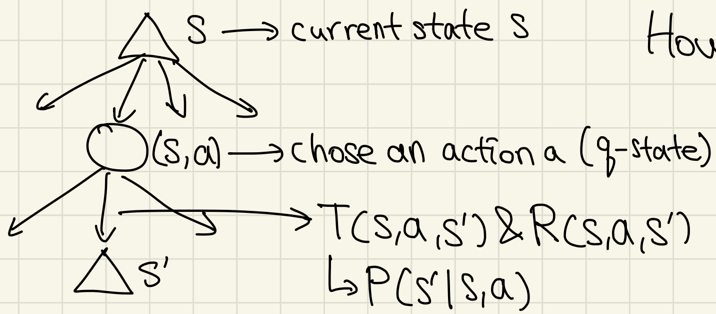
↳ the optimal policy maximizes the expected utility

ex) race car



optimal policy: $\pi^*(\text{cool}) = \text{fast}$, $\pi^*(\text{warm}) = \text{slow}$, $\pi^*(\text{over}) = \text{end}$

MDPs can be formulated as a search tree (expectimax)



How to model rewards? now or later?

↳ decay rewards exponentially

$(1, \gamma, \gamma^2, \cdots | \gamma \in [0,1])$

Each round, the reward will be multiplied by discount factor γ.

↳ Sooner rewards have higher rewards than later ones

↳ It also helps rewards converge rather than approach infinity

$$U([r_0,\dots,r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \le R_{max}/(1-\gamma) \text{ (bounded)}$$

How to solve MDPs? → think like expectimax, kind of

↳ states are repeated in subtrees → cache them!

↳ do depth-limited computation until changes are small

$V^*(s)$:= expected utility of starting in s & acting optimally

$Q^*(s,a)$:= expected utility of the q-state (s,a) & acting optimally

$\pi^*(s)$:= the optimal action from state s

Bellman Equations (similar to expectimax):

$$V^*(s) = \max_a Q^*(s,a), \quad Q^*(s,a) = \sum_{s'} T(s,a,s')[\underbrace{R(s,a,s')}_{\text{immediate rewards}} + \underbrace{\gamma V^*(s')}_{\text{discounted expected utility}}]$$

$$\Rightarrow V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$ → how to solve this?

Time Limited Values: $V_k(s)$:= optimal value of s if the game ends in k more steps (depth-k expectimax for s)

$$V_0(s) \leftarrow 0, \quad V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \underbrace{V_k(s')}_{\text{given all } V_k \text{ values for all } s'}]$$

↳ repeat until convergence, which yields V^* ($O(S^2A)$ each step)

Bellman Equation for Q^*: $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma(\max_{a'} Q^*(s',a'))]$

↳ Leads to Q-value iteration algorithm for RL

But how do we get information about actions (policies)?

↳ Imagine we have the optimal values $V^*(s)$. How should we act?

Do a mini-expectimax: $\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

(argmax returns the "key value" of the largest value in a dict)

⇒ Policy extraction, since it gets optimal policies by values.

If we have optimal Q-values, $\pi^*(s) = \arg\max_a Q^*(s,a)$

↳ extracting policies are a lot easier with q-values!

Issues with value iteration: ① slow, ② "max" rarely changes

③ policies converge much faster than values

⇒ Policy-based methods can be more efficient!

Policy Evaluation: what are the consequences of a policy?

↳ rather than computing maximizer nodes, just do what policy tells

⇒ S will take $\pi(S)$ and land in q-state $(S, \pi(S))$

↳ $V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$ → no more max, always choose $\pi(s)$

Turn $V^\pi(s)$ into iterations: $V_0^\pi(s) = 0$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma V_k^\pi(s')\right]$$

↳ efficiency is $O(s^2)$, no more factor of a when maxing

↳ without $\max_a$, this is just a set of linear equations!

Policy Iteration: Alternate between Policy $\overbrace{\text{evaluation}}^{\pi \to V^\pi}$ & $\overbrace{\text{extraction}}^{V \to \pi^V}$

① calculate utilities for some fixed policy until convergence

② update policy using one-step lookahead with calculated utilities

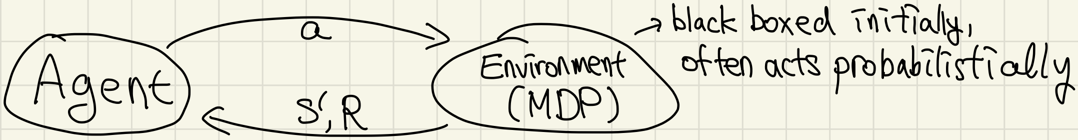
⇒ still optimal, could converge faster than value iteration

Reinforcement Learning

Still assume MDP, looking for a policy $\pi(s)$

↳ What if we don't know T or R? (no measure of "good")

→ Must try out actions to learn from them!



Offline (MDP) vs. Online (RL)

Passive RL vs. Active RL

↳ Model-Based RL ↳ Exploration vs. Exploitation

↳ Model-Free RL

Model-Based Idea: Learn an approximate model, and solve for values assuming it is correct

① Learn the distribution $\hat{T}(s,a,s')$ & $\hat{R}(s,a,s')$

② Solve the model with iteration

③ Run the learned policy, repeat if unsatisfactory

Model-Free: Don't know T and R, first learn $V(s)$.

Direct Evaluation: Just average <u>all experiences afterwards</u> when that state was visited (no state dependency)

↳ Bellman updates don't work b/c they depend on T & R.

⇒ How do we take the weighted average without knowing them?

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma V_{k+1}^{\pi}(s')\right]$$

↳ take samples of outcomes s' and average them

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n}\sum_{i}\left[\underline{R(s,\pi(s),s_i') + \gamma V_k^{\pi}(s_i')}\right]$$

↳ samples will already be weighted by frequency

Temporal Difference Learning: learn from <u>every</u> experience!

keep a running average of $V(s)$ until s is visited again

$\rightarrow V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \cdot \text{sample}$

$\equiv V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(\text{sample} - V^{\pi}(s))$

Exponential Moving Average: $\overline{x}_n = (1-\alpha)\overline{x}_{n-1} + \alpha \cdot x_n$

↳ recent samples are emphasized, past estimates are "forgotten"

⇒ still only does evaluation, we want new, better policies

Q-Learning: sample-based Q-value iteration

$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample} \quad \text{sample} \rightarrow R(s,a,s') + \gamma \max_{a'} Q(s',a')$

⇒ converges to optimal policy (off-policy learning)

↳ as long as Q-value can converge (# of trials, lr decay, etc)

how we choose to collect samples does not matter!

Active RL: how to act to collect data?

↳ The learner can choose what it wants to explore!

Simplest scheme: ε-greedy (act randomly with probability ε)

↳ not really deliberate in exploring other states

⇒ somehow represent "novelty" to promote exploration!

Exploration Function: $f(u,n) = u + k/n$ (n: visit count, u: utility)

→ high when N is low!

$$Q(s,a) \xleftarrow[\alpha]{\text{(weighted update)}} R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a))$$

Regret: how effectively did we learn? (optimally learn the optimal)

↳ less regret means faster learning

Feature Representation Formulas:

$$Q(s,a) = \vec{w} \cdot \vec{f}(s,a)$$

$$\text{diff} = \left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

$$w_i \leftarrow w_i + \alpha \, \text{diff} \cdot f_i(s,a)$$

Probability

Observed variables (evidence): what the agent knows
Unobserved variables: agent needs to reason about these
Model: agent knows how to relate observed to unobserved

Random Variables: Aspect of the world we might have uncertainty
↳ each RV has a domain, discrete, boolean, continuous, tuples, etc.
Probability Distribution: Assigns each value of a RV a probability
↳ $P(X=v)$ denotes the probability X takes on value v.
⇒ $\forall x, P(X=x) \geq 0, \sum_x P(X=x) = 1$ (basic rules for PD)
Joint Distribution: Probability of set of RVs, $P(x_1, x_2, \dots, x_n)$
↳ the size of JD grows exponentially as variables increase
Events: Set of possible outcomes, $P(E) = \sum_{(x_1,\dots,x_n)\in E} P(x_1, \dots, x_n)$
↳ E acts like a filter for which JD we are interested in
Marginal Distribution: Collapsed rows by eliminating RVs in JD
↳ Acts as if we have no knowledge of the eliminated RV
↳ $P(X_1 = x_1) = \sum_{x_2} P(X_1 = x_1, X_2 = x_2)$ (sum up all possible X_2 over X_1)

Conditionals: $P(a|b) = \frac{P(a,b)}{P(b)}$ ($P(a)$ given that b already holds)

↳ simple relation between joint and conditional probability

⇒ $P(b)$ can generally be found by marginalization over b

Conditional Distribution: PD over some variables when others are fixed

↳ Acts like taking a subset of the JD then renormalizing probabilities

Probabilistic Inference: compute a desired probability based on others

↳ generally compute conditionals, new evidence cause beliefs to be updated

Inference by Enumeration: $\overbrace{E_1, \ldots, E_k}^{\text{evidence}} = e_1, \ldots, e_k, \overbrace{Q}^{\text{query}}, \overbrace{H_1, \ldots, H_r}^{\text{hidden}} \in X_{(1-n)}$

⇒ $P(Q|e_1, \ldots, e_k)$ (observed $e_1, \ldots, e_k$, then what is $P(Q)$?)

1) Select entries consistent with evidence

2) Sum out H to get JD of E and Q, 3) Normalize

↳ Leads to runtime & space complexity $O(d^n)$ (Ineffecient!)

Product Rule: $P(x,y) = P(y)P(x|y)$ (derived from $P(x|y) = \frac{P(x,y)}{P(y)}$)

⇓

Chain Rule: $P(x_1, x_2, \ldots, x_n) = \prod_i P(x_i | x_1, \ldots, x_{i-1})$

↳ $P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1) = \cancel{P(x_1)} \frac{\cancel{P(x_2, x_1)}}{\cancel{P(x_1)}} \cdot \frac{P(x_3, x_2, x_1)}{\cancel{P(x_2, x_1)}}$

Bayes Rule: $P(x,y) = P(x)P(y|x) = P(y)(x|y) \rightarrow \underline{\underline{P(x|y) = \frac{P(x)P(y|x)}{P(y)}}}$

↳ useful for "flipping" probabilities when finding $\underset{\swarrow}{\underline{\text{one}}}$ is easier than $\underset{\downarrow}{\underline{\text{other}}}$

P(effect|cause) P(cause|effect)

ex) M: meningitis, S: stiff neck. $P(+m) = 0.0001$, $P(+s|+m) = 0.8$, $P(+s|-m) = 0.01$. What is $P(+m|+s)$?

$$\hookrightarrow P(+m|+s) = \frac{P(+s|+m)P(+m)}{P(+s) \rightarrow ?} = \frac{P(+s|+m)P(+m)}{P(+s,+m) + P(+s,-m)} \text{ (marginals)}$$

$$= \frac{P(+s|+m)P(+m)}{P(+s|+m)P(+m) + P(+s|-m)P(-m)} \text{ (product rule)} = \frac{0.8 \cdot 0.0001}{0.8 \cdot 0.0001 + 0.01 \cdot 0.9999} \simeq \underline{0.008}$$

Bayes Nets

Independence: X, Y are independent if $\forall (x,y) \in (X,Y)$, $\underline{\underline{P(x,y) = P(x)P(y)}}$

↳ also implies $\forall x,y$, $\underline{\underline{P(x|y) = P(x)}}$ (y reveals nothing about x)

⇒ Independence is a <u>modeling assumption</u>! Emperical JD are "close".

Conditional Independence: Independent when a third variable is observed

X is cond. ind. of Y given Z iff:

$$\forall x,y,z: P(x,y|z) = P(x|z)P(y|z), \quad P(x|z,y) = P(x|z)$$

Decomposition of Chain Rule: $P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_2,x_1)$

can be reduced to simpler structures $\rightarrow P(x_3|x_2,x_1) = P(x_3|x_1)$

Bayes' Nets: describing complex JD using local conditionals

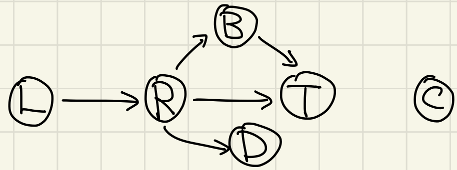
Graphical Models: nodes → variables, arcs → interactions

ex) N independent coin flips: X_1 X_2 . . . X_N

ex) Traffic: R(raining), T(traffic) R ⟶ T

ex) Traffic II: R, T, L(low pressure),

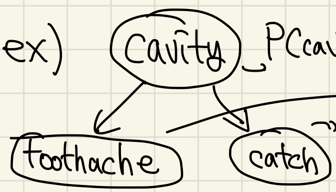
D(roof drips), B (ballgame), C (cavity)



Semantics: DAG topology + $\underbrace{\text{conditional } P(X \mid a_1, \dots, a_n)}$ where ↳ this is a table (CPT)

$a_i \to X$ is an edge in the DAG. $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{Parents}(x_i))$

ex)



P(cavity)

→ P(toothache | cavity)

P(catch | cavity)

$P(+\text{cav}, +\text{catch}, -\text{tooth})$

$= P(+\text{cav}) \cdot P(+\text{catch} \mid +\text{cav})$

$\cdot P(-\text{tooth} \mid +\text{cav})$

Why is this true? $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid x_1, \dots, x_{i-1}) \to P(x_i \mid \text{Parents}(x_i))$

↳ this is a core assumption of the world modeling

* Not all JD can be represented from BN!

ex) N independent coin flips: $P(h,h,t,h) = \prod_{i=1}^{n} P(x_i \mid \cancel{\text{Parents}(x_i)}) = 0.5^4$

ex) Traffic: R ⟶ T $P(+r, -t) = P(+r \mid \varnothing) \cdot P(-t \mid +r) = P(+r) P(-t \mid +r)$

Reverse Causality? T → R still possible to reconstruct the JD!

↳ Direction of the edges do not mean direction of causality!

* Topology really encodes <u>conditional probabilities</u>

Size of the BN: N boolean variables $\rightarrow 2^N$ entries of JD

↳ N node BN with $\leq k$ parents $\rightarrow N \cdot 2^{(k+1)}$ entries of JD

$\Rightarrow$ if $k \ll N$, $N \cdot 2^{(k+1)} < 2^N \rightarrow$ faster local CPTs & queries!

Bayes' Nets: Independence

$F \rightarrow S \rightarrow A$

ex) Alarm $\perp\!\!\!\perp$ Fire | Smoke $\rightarrow$ Alarm doesn't care about source of smoke

BN often give rise to additional conditional independence

ex) $X \rightarrow Y \rightarrow Z \rightarrow W$: $Z \perp\!\!\!\perp X \mid Y$, $W \perp\!\!\!\perp X, Y \mid Z \rightarrow$ <u>$W \perp\!\!\!\perp X \mid Y$</u> ?? how

D-Seperation: Algorithm for determining conditional independence from graphs

↳ study properties of <u>triples</u>, then compose them into complex paths

1) Causual Chains: $X \rightarrow Y \rightarrow Z$ $\quad P(x,y,z) = P(x)P(y|x)P(z|y)$

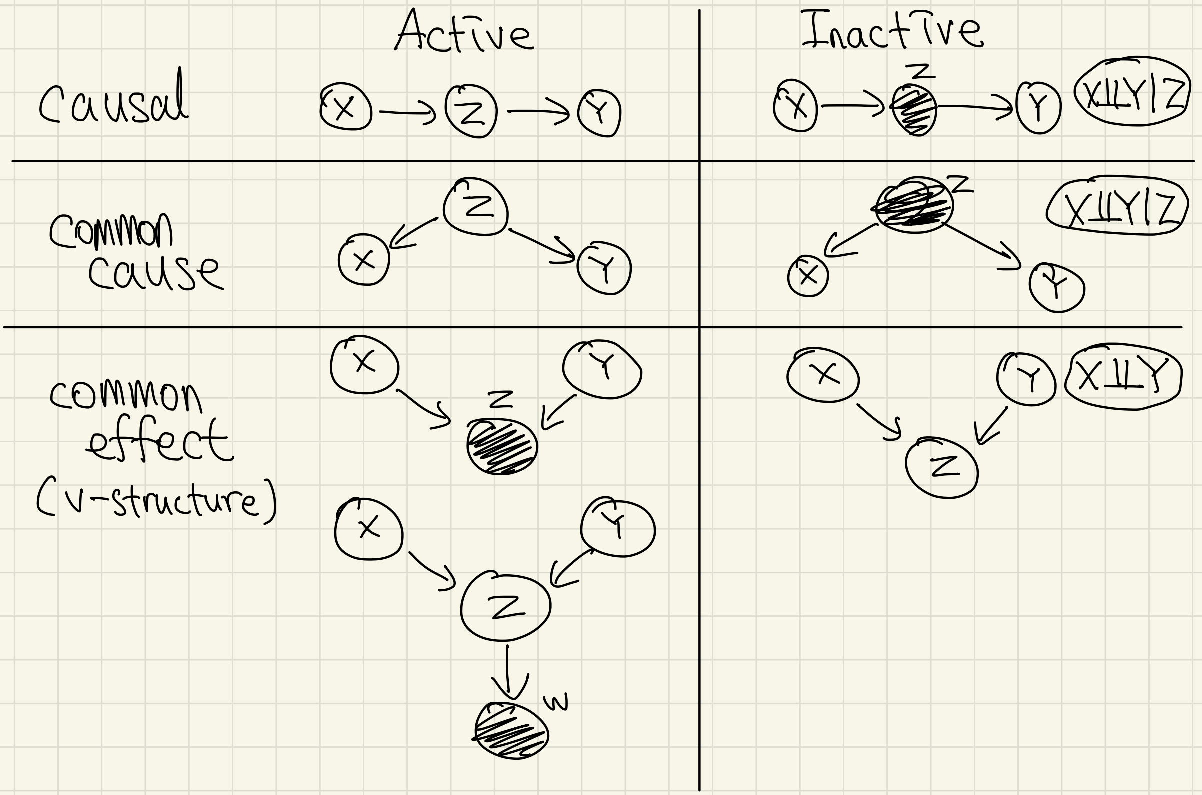
↳ $Z \perp\!\!\!\perp X$ is false (not guaranteed), however $Z \perp\!\!\!\perp X \mid Y$ is true! $(P(z|x,y) = P(z|y))$

2) Common Cause: $X \leftarrow Y \rightarrow Z$ $\quad P(x,y,z) = P(y)P(x|y)P(z|y)$

↳ $Z \perp\!\!\!\perp X$ is false (not guaranteed), however $Z \perp\!\!\!\perp X \mid Y$ is true! $(P(z|x,y) = P(z|y))$

3) Common Effect: $X \rightarrow Z \leftarrow Y$ $\quad P(x,y,z) = P(x)P(y)P(z|x,y)$

↳ $X \perp\!\!\!\perp Y$ is true (guaranteed), however $X \perp\!\!\!\perp Y \mid Z$ is false!! (it is likely that one is actually contributing to Z, which decreases likelihood of the other)



General Case: entire graph is just repetition of the three canonical cases!

↳ All it takes to block a path is <u>a single active segment</u>

ex) A − B − C − D − E → A-B-C, B-C-D, C-D-E

If there are <u>any</u> path $X \rightsquigarrow Y$ that is active, <u>not D-seperated</u>.

<u>* $X \perp\!\!\!\perp Y \mid \{Z\}$ is guaranteed iff. X and Y are D-separated given $\{Z\}$.</u>

↳ This does not imply anything about $X \perp\!\!\!\perp Y \mid Z$ when X and Y aren't D-seperated, only that it's <u>not</u> guaranteed!

Bayes' Nets: Inference

Inference: Calculating some useful quantities from a JD

ex) Posterior: $P(Q \mid E_1=e_1, \ldots, E_k=e_k)$

Most likely: $\underset{q}{\operatorname{argmax}}\, P(Q=q \mid E_1=e_1, \ldots, E_k=e_k)$

Inference by Enumeration is slow b/c it expands to a full JD

Variable Elimination can take shortcuts when marginalizing.

Factors: ① $P(X,Y) \rightarrow$ sums to 1 ② $P(x,Y) \rightarrow$ sums to $P(x)$

③ $P(Y \mid x) \rightarrow$ sums to 1 ④ $P(Y \mid X) \rightarrow$ sums to $|X|$

⑤ $P(y \mid X) \rightarrow$ sums to... unknown! In general, $P(Y_1 \ldots Y_N \mid X_1 \ldots X_M)$

has a dimension equal to # of unassigned variables

Enumeration: $\sum_t \sum_r P(L \mid t) P(r) P(t \mid r)$ vs Elimination: $\sum_t P(L \mid t) \sum_r P(r) P(t \mid r)$

If we have evidence, start with consistent entries only.

General VE procedure: $P(Q \mid E_1=e_1, \cdots, E_k=e_k)$

while $\exists H_i$: Join all factors mentioning H_i, then eliminate H_i.

Finally, normalize the JD to match the original query.

↳ basically reordering to lessen redundant multiplications, worst case exponential runtime w.r.t. size of the BN.

Bayes' Nets: Sampling

Sampling is like repeated simulation.

Basic Idea: Draw N samples from sampling distribution S.

Compute an approximate posterior probability. Show that this converges to the true probability P as N grows.

Step 1) $u \leftarrow$ uniform$(0,1)$ (kind of given)

Step 2) Convert u into an outcome based on subintervals in $[0,1)$

Prior Sampling: Naively repeat sampling from start to finish

for $i = 1 \ldots n$: Sample x_i from $P(x_i \mid \text{Parents}(x_i))$

Rejection Sampling: Only sample those that are absolutely needed

for $i = 1 \ldots n$: Sample x_i from $P(x_i \mid \text{Parents}(x_i))$

if x_i not consistent with evidence: reject & return early

↳ Rejects a LOT of samples, and evidence is not utilized.

Likelihood Weighting: what if we just force the evidence?

↳ just do it, but keep track of the likelihood that it ACTUALLY happens with a weight factor

$w \leftarrow 1.0$. for $i = 1 \dots n$:

if X_i is an evidence variable:

$X_i \leftarrow$ observation x_i for X_i

Set $w \leftarrow w \times P(x_i \mid Parents(X_i))$

↳ basically means "this is equivalent to w # of samples, where $w \in [0, 1)$"

else:

Sample x_i from $P(X_i \mid Parents(X_i))$

↳ Pretty good, just that it ignores evidence that comes later

Gibbs Sampling: Kind of like local search, perturb one observation

1) Fix evidence 2) Initialize all other variables

3) Repeat: Choose a non-evidence variable X.

Resample X from $P(X \mid \text{all other variables})$

⇒ $P(X \mid \text{all other variables})$ is very efficient due to cancellation with BN assumptions

Decision Network

Bayes' Nets, but with additional types of nodes!

- Action Node (some domain, agent's choice) □
- Utility Node (based on its parents' outcomes) ◇

Goal: Maximize expected utility, given the evidence!

Action Selection: 1) Instantiate all evidence
2) Set action in every way 3) Calculate posteriors
4) Calculate expected utility 5) Choose maximizing action

Almost looks like expectimax / MDP, but with BN distribution

* MEU <u>can</u> decrease with additional information, but it doesn't mean that we are less happy, it just means that the initial assumptions were inaccurate descriptions of reality.

$\underline{MEU(E=e) = \max_a \sum_s P(s|e)\,U(s,a)}$. (same for multiple evid.)

Value of Information: compute the value of acquiring evidence

↳ value := expected gain in MEU with new evidence

☆ $\underline{MEU(E') = \sum_{e'} P(E'=e')\,MEU(E'=e')}$, $\underline{VPI(E') = MEU(E') - MEU(\emptyset)}$

VPI Properties:

1) Nonnegativity: $\forall E', e$, $VPI(E'|e) \geq 0$

2) Nonadditivity: $VPI(E_j, E_k|e) \neq VPI(E_j|e) + VPI(E_k|e)$

3) Order-independent: $VPI(E_j, E_k|e) = VPI(E_j|e) + VPI(E_k|E_j, e)$

$$= VPI(E_k|e) + VPI(E_j|E_k, e)$$

*If Parents$(U) \perp\!\!\!\perp Z$ | Current Evidence, then $VPI(Z|\text{Curr.Evi}) = 0$

POMDP: MDP, but states update their probabilities over time

↳ solve using truncated expectimax to appoximate utilities

Hidden Markov Models

"What if the state of the world evolves over time?"

Markov Model: $P(x_1)$, $P(X_t|X_{t-1})$ Same for all (stationary)

↳ past & future independent of present, only dependent on previous

$$P(x_t) = \sum_{x_{t-1}} P(x_{t-1}, x_t) = \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}) \rightarrow \text{converges as } t \to \infty!$$

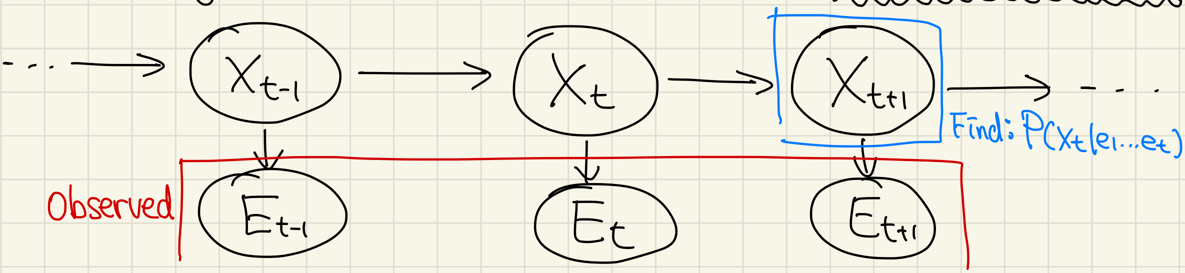
Stationary Distribution: $\underline{\underline{P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x) P_\infty(x)}}$

↳ This can be solved as a system of linear equations!

However, Markov models are generally not good modeling of reality

Hidden Markov Models (HMM): observe outputs at every time step!

↳ defined by: Initial $P(X_1)$, Transitions $P(X_t|X_{t-1})$, Emissions $P(E_t|X_t)$



Independence Properties: 1) X_{t+1} is only dependent on X_t.

2) Current observation is independent of all else given the current state.

* It is not the case that evidences are always independent!

Filtering: Tracking and updating $B_t(X) = P_t(X_t|e_1 \dots e_t)$ over time

↳ idea: start at $P(X_1)$ and derive $B_t(X)$ using $B_{t-1}(X)$

Two steps: Passage of Time & Observation (Incomplete & Complete)

↳ Passage of Time: $B_t(X) = P_t(X|e_{1:t}) \Rightarrow P_t(X_{t+1}|e_{1:t}) = \sum_{x_t} P(X_{t+1}|\underbrace{x_t, e_{1:t}}_{}) P(x_t|e_{1:t})$ actually irrelevant!

$= \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t}) \Rightarrow \underline{\underline{B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)}}$

↳ Observation: $B'(X_{t+1}) = P(X_{t+1}|e_{1:t+1}) \Rightarrow P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, \underbrace{e_{t+1}|e_{1:t}) / P(e_{t+1}}_{}|e_{1:t})$ constant

$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1}|e_{1:t}) = P(e_{t+1}|e_{1:t}, X_{t+1}) P(X_{t+1}|e_{1:t}) = P(e_{t+1}|X_{t+1}) P(X_{t+1}|e_{1:t})$ irrelevant!

$\Rightarrow \underline{\underline{B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})}}$ ("reweighting" beliefs after observing)

↳ need renormalization after derivation!

Forward Algorithm: $\underline{\underline{P(x_t|e_{1:t}) \propto_{X_t} P(e_t|x_t) \sum_{X_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})}}$

→ How do we deal with large state spaces?

Particle Filtering: Approximate Inference for Markov models

↳ Representation of $P(X)$ is a list of N samples (particles)

Passage of Time: $x' = \text{sample}(P(X'|x))$ (generate the next step)

Observation: $w(x) = P(e|x)$, $B(X) \propto \overbrace{P(e|X)}^{w(X)} B'(X)$ (downweight w.r.t. likelihood)

Resample: Choose new samples $\underline{\underline{\text{based on } B(X)\text{'s distribution}}}$ (≃ renormalizing)

Dynamic Bayes' Nets*: Multiple Markov / Observation nodes in BN!

Machine Learning: Naïve Bayes

"How to acquire a model from data/experience"

Types of Problems: Supervised (→ labels), Reinforcement (→ reward func.), Unsupervised (→ no labels, just features)

Supervised Learning
→ Classification: Discrete domains
→ Regression: Real-valued domains

Classification: Dataset $(x, y) \xrightarrow{\text{extraction}}$ Features $\xrightarrow{\text{ML}}$ Predict y

↳ ML learns patterns between features and labels from data!

ex) Spam Filter: Dataset(Email, {spam, ham}) → predict spams!

↳ What features do we want to look at? words (FREE), symbols ($), ...

ex) Digit Recognition: Dataset(Pixel grid, {0, ..., 9}) → predict digits!

↳ Features: Pixel(x, y) = On/Off, shape patterns (components, loops, ...)

Model-Based Classification: Build a BN where both label and features are RVs. Instantiate any observed variables, and find distribution of y.

Naive Bayes: All features (F_i) are independent effects of the label (Y).

⇒ $P(Y)$: Prior. $P(F_i|Y)$: Probability of feature, given the label.

Naive Bayes for Digits: One feature $F_{i,j}$ for every pixel grid position (i,j)

→ $P(Y)$ (likelihood of every digit), $P(F_{i,j}|Y)$ (on/off when the label is y)

Naive Bayes for Text: W_i is the word at position i ($w_i \in$ {Dictionary})

Moreover, each $P(W_i|Y)$ is assumed to be the same. ⇒ identically distributed

↳ This assumption reduces the # of parameters, also generalizes better!

↳ However, it will be insensitive to word ordering! (design choice)

→ $P(Y)$ (spam/ham), $P(W|\{spam, ham\})$ (likelihood of word given the type of email)

In general, the joint probability will be $P(Y, F_1, \dots, F_n) = P(Y)\prod_{i=1} P(F_i|Y)$.

↳ total # of parameters is linear w.r.t. n!

⇒ Computing $P(Y|F_1, \dots, F_n)$ is just inference in BN.

↳ Inference by Enumeration: $P(Y|F_1, \dots, F_n) \propto P(Y, F_1, \dots, F_n) = P(Y)\prod_{i=1} P(F_i|Y)$.

⇒ $P(y_j)\prod_{i=1} P(F_i|y_j)$, then normalize to get $P(y_j|f_1, \dots, f_n)$.

We also need to estimate the CPTs → Let θ denote all (trainable) parameters!

Parameter Estimation: Empirically learn using training data

↳ Maximum Likelihood: choose θ that maximizes the probability of data! $P(\text{Data}|\theta)$

→ solve $\underset{\theta}{\text{argmax}}(f(\theta))$ where $f(\theta)$ is the probability of data happening

Useful fact: $\underset{\theta}{\text{argmax}} f(\theta) = \underset{\theta}{\text{argmax}} \ln(f(\theta))$ (easier analytic solution with differentiation)

↳ For Naive Bayes, $P(y) = \frac{\#\text{ of } y}{\text{total}}$, $P(f|y) = \frac{\#\text{ of } f \text{ AND } y}{\#\text{ of } y}$

Empirical Risk Minimization: we want models to perform well on unseen data

↳ More training data, or regularize model complexity

However, training data could misrepresent the true distribution!

In general, we don't want to assign 0 probabilities for uncertain θs.

↳ need smoothing or regularization

Laplace Smoothing: $P_{LAP_k}(X) = \frac{C(x)+k}{\sum_x[C(x)+k]} = \frac{C(x)+k}{N+k|X|}$

↳ intuitively, act as if we observed k more events of each outcome

Tuning: find the optimal smoothing value k via held-out dataset

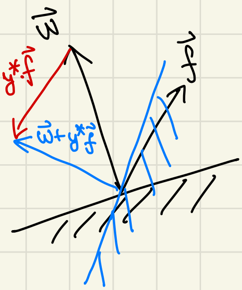
Perceptrons

Binary Classifier: $\text{activation}_w(X) = \text{sgn}(\sum_i w_i \cdot f_i(x)) = \text{sgn}(\vec{w} \cdot \vec{f}(x))$

↳ dot product signifies the correlation between weight & feature

In the feature vector space, data are points, and weight vectors are hyper-planes.

⇒ we need to learn the weight vector from data!



Weight Updates: $y = \begin{cases} +1 \text{ if } \vec{w} \cdot \vec{f}(x) \geq 0 \\ -1 \text{ if } \vec{w} \cdot \vec{f}(x) < 0. \end{cases}$

update if y is wrong $(y \neq y^*)$, $w \leftarrow w + \overbrace{y^*}^{\text{correct label}} \cdot f$

↳ Intuitively, we are shifting the hyperplane to reflect observed data

Multiclass Decision: $\vec{w}_y$ for each class, $y = \underset{y}{\text{argmax}}\ \vec{w}_y \cdot \vec{f}(x)$

↳ update $w_y = w_y - f(x)$ for wrong answer, $w_{y^*} = w_{y^*} + f(x)$ for correct answer
(if $y \neq y^*$)

If the data are perfectly seperable, the perceptron will converge

↳ However, it might have problems if not. (thrashing, suboptimal)

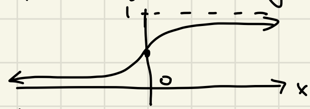
Logistic Regression

Non-Seperable Data: Any linear boundary will make at least one mistake

↳ interpret the line as a probabilistic decision (50:50)

Perceptron scoring: $z = \vec{w} \cdot \vec{f}(x) \rightarrow$ want $Pr \rightarrow 1$ if positive, $\rightarrow \emptyset$ if negative

↳ Sigmoid: $\phi(z) = \frac{1}{1+e^{-z}}$ achieves this behavior!



$\Rightarrow Pr(y=1 \mid x, w) = \frac{1}{1+e^{-(w \cdot f(x))}}, \; Pr(y=-1 \mid x, w) = 1 - \frac{1}{1+e^{-(w \cdot f(x))}}$

↳ increasing w will make the boundary sharper (best w?)

MLE of Log. Reg.: Log likelihood $= \sum_i \log Pr(y^{(i)} \mid x^{(i)}, w)$

$\Rightarrow$ a probilistic intepretation can also improve the seperable case!

Multiclass Log. Reg: $\forall z_i \in \{z_1, \ldots, z_n\}, \; \text{softmax}(z_i) := \frac{e^{z_i}}{\sum_j e^{z_j}}$

↳ transforming original activations into "softmax" activations

$\Rightarrow P(y \mid x, w) = \frac{e^{w_y \cdot f(x)}}{\sum_{y'} e^{w_{y'} \cdot f(x)}}$ for perceptron interpretation

Deep Neural Network: Cascading logistic regression of multiple layers

↳ a hidden layer $h_i^{(l)} := \phi(\vec{w}_i^{(l)} \cdot \vec{h}^{(l-1)}) = \phi\left(\sum_j w_{ji}^{(l)} \cdot h_j^{(l-1)}\right)$

$\rightarrow$ in matrix form, $\vec{h}^{(l)} = \phi(W^{(l)} \times \vec{h}^{(l-1)})$ where $W^{(l)}$ is the matrix $\begin{bmatrix} - \vec{w}_1^{(l)} - \\ \vdots \\ - \vec{w}_n^{(l)} - \end{bmatrix}$

↳ still uses MLE, but now it is iterative