# Search (state space: how to rep. / state)

A*: frontier rep. as priority Q: sorted by lowest est. cost
: f(n) = g(n) + h(n) | g(n): dist traveled already | h(n): est cost to goal
: h(n) informed about knowledge on world

Admissability: $\forall n,\ 0 \le h(n) \le h^*(n)$; $h^*(n)$ is True cost to goal
↳ under estimates cost to goal. "optimistic"

Consistency $\forall A, C\ \ h(A) - h(C) \le cost(A, C)$
↳ under estimate cost to next node. triangle ineq.
↳ Stronger than Admissability: Consistency ⇒ Admissability

Guaranteed → optimal path
DFS: ✗
BFS: if cost = 1
A*: if Consistent Heuristic
G equal
if Cost = 1, BFS = UCS

Monte Carlo tree Search
↳ randomly
↳ uses random decisions
↳ low action factor
↳ selection, first position, Simulation, Back prop.

Tree search V.S. Graph Search
↳ Admissible h(n) optimal for tree ≠ graph

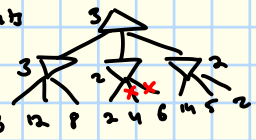
completeness
BFS, A* (bad heuristic)

tree search = graph search + [reached set]

# GAMES

Zero-sum: Nodes take turns (Max & Min if optimal)
(Minimax): $b^m = O(n)$

Alpha beta pruning: $O(n) = b^{m/2}$ → we can prune expectimax if there is No chance it meets criterion
if node is evaluated, we don't need to look at other parents




A is Necessary & Sufficient: for B
⇒ ⇐ ⇔

Expectimax: random node care cases & optimal
: Be careful about pruning.
Ex →

# LOGIC: use Knowledge base (from observations) to deduce things

Logical Eq
$\alpha \wedge \beta \equiv \beta \wedge \alpha$
$\alpha \vee \beta \equiv \beta \vee \alpha$
$(\alpha \wedge \beta) \wedge \gamma \equiv \alpha \wedge (\beta \wedge \gamma)$
$(\alpha \vee \beta) \vee \gamma \equiv \alpha \vee \beta \vee \gamma$
$\neg(\neg\alpha) \equiv \alpha$
$\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$
$\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta \wedge \beta \Rightarrow \alpha)$
$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
$\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Model: T, F assignment for any given problem

Valid: True in all worlds / models

Satisfiable: True in one world

Unsatisfiable: Never true. $(\neg A \wedge A)$

entailment (⊨): $A \models B$: iff in All worlds A is true, B is true
↳ Direct Proof $A \models B$: iff $A \Rightarrow B$ is Valid
↳ Contradiction $A \models B$ iff $A \wedge \neg B$ is unsatisfiable

CNF: Conjuctive normal form: only in: Ands & ors
$\wedge, \neg, \vee, (, )$

Model Checking: $KB \models q$
↳ Enumeration / Truth table
↳ table one truth table and find out

SAT solver ↳ DPLL: Checks Satisfiability (find 1 world it is true)
todo: find a world (variable assignment to make statement true)

Theorem Proving
↳ if $A \Rightarrow B$, if we know A, we know B
↳ if $A \wedge B$, if we infer A, we infer B
↳ if we have $A \wedge B$, we infer $A \wedge B$
↳ Forward chaining: implication chart

Inference of proportionality ($\propto$)
→ Ex: find $P(+x \mid b)$, $P(-x \mid b)$
→ $\frac{P(b \mid +x)P(+x)}{P(b)}$ $\quad$ $\frac{P(b \mid -x)P(-x)}{P(b)}$
$\propto P(+x, b) = P(+x)P(b \mid +x)$ $\quad$ $\propto P(-x, b) = P(-x)P(b \mid +x)$
→ find answer then normalize:
$\frac{P(+x, b)}{P(+x, b) + P(-x, b)}$ ⇐ Linearization step → $\frac{(-x, b)}{P(+x, b) + P(-x, b)}$

Successor State Axioms:
$F^{T+1} \Leftrightarrow$ Action caused $(F^T) \vee (F^+ \wedge \neg$ Action causes not $(F^+))$

Model: {}
Symbols: [L, M, N, P, Q, R, S]
clauses: $(\neg L \vee \neg S) \wedge (M \vee Q \vee N) \wedge (L \vee \neg M) \wedge (L \vee \neg Q) \wedge (\neg L \vee \neg P) \wedge (R \vee P \vee N) \wedge (\neg R \vee \neg L) \wedge S$
Model: {S: T}
Symbols: [L, M, N, P, Q, R]
clauses: $(\neg N \vee \neg T) \wedge (M \vee Q \vee N) \wedge (L \vee \neg M) \wedge (L \vee \neg Q) \wedge (\neg L \vee \neg P) \wedge (R \vee P \vee N) \wedge (\neg R \vee \neg L)$
Model: {S: T, N: F}
clauses: $(M \vee Q \vee F) \wedge (L \vee \neg M) \wedge (L \vee \neg Q) \wedge (\neg L \vee \neg P) \wedge (R \vee P \vee F) \wedge (\neg R \vee \neg L)$
* Choose M = True, then M = False
if you get $(R) \wedge (-R)$ or some unsolvable case, Unsatisfiable, else: Satisfiable

A→B, A→C, B∧C→D, D∧E→Q, A∧D→Q | Q?

Inferred: [] | Answer: [A] | Cost: [1, 1, 2, 2, 2, 0]

A→B, A→C, B∧C→D, D∧E→Q, A∧D→Q

Infer: [A] | Answer: [B, C] | Cost: [0, 0, 2, 2, 1, 0]

B∧C→D, D∧E→Q, A∧D→Q

Infer: [A, B, C] | Answer: [C] | Cost: [1, 0, 1, 2, 1, 0]

D∧E→Q, A∧D→Q  Q is true

Infer [A, B, C, D] ...

## Factored state rep

↳ Can be rep. with binary state. / symbols

## Stochastic transition

↳ Bayes & MDP

## Known physics

↳ search, CSP, Bayes, MDP

## Probability

$P(a|b) = \frac{P(b|a)\,P(a)}{P(b)}$ ← Bayes

Indep: $P(x|y) = P(x)$; $P(y|z) = P(y)$

Conditional indep: X is cond. indep of Y given Z if (⫫ : symbol)

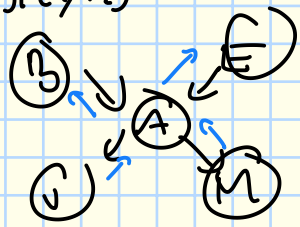
$\forall x,y,z \quad P(x|y,z) = P(x,z)$

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

Unitary Constraints uses 1 var...
binary Constraint uses 2 var ...

$A \perp\!\!\!\perp B | C \Rightarrow P(A,B|C) = P(A|C)\cdot P(B|C)$

## BAYES NETS




CPT: P(B), P(E)
P(A|B,E) P(J|A)
P(M|A)


$P(C=c|B=b) = P(+a)P(C|a,b) + P(\neg a)P(c|\neg a,B)$

A → B: $A \perp\!\!\!\perp B$ if $P(B|A) = P(B)$ can still be inv

**Variable elim:** join all factors of X → sum out X.
factor: any unnormalized prob

**Inference by Enumeration**
list all out: CPT: P(W), P(D), P(I|W,D), P(A|W,D), P(G,I|A)

$P(G) = \sum_{w\in W}\sum_{d\in D}\sum_{i\in I}\sum_{a\in A} P(W=w)P(D=d)P(i=I_{i,w,d})P(a=A_{a,w,d})P(G, A=a, I=i)$

**Prior Sampling (biggest)**
gen samples,
→ discard samples inconsistent w evidence at end
→ est. prob.

**Rejection Sampling**
gen samples,
discard samples inconsistent w evidence as we generate
→ est. prob.

**Likelyhood weighting**
fix evidence var. to a value (weight our by prob)
Multiply out distributions

**Gibbs Sampling**
→ fix ev. variables. (all other vars are free)
→ gen new states by looping through non evidence variables given states.

**Independence**
→ given parents, node is indep. to all other variables nodes
→ given Markov blanket, Each node is cond. indep (Parents, Children, Children's other parents)

**Maximising P(α) given constraints**
→ Make polynomial of new probability given Lagrange
→ solve derivate of P(α) = local max when $\frac{dP(\alpha)}{d\alpha} = 0$

Let $q = P(+n)$ represent the probability, and then $P(+d) = (1-q)$. We then know that $P(-n) = (1-q)$ and $P(-d) = 1-(1-q) = q$. Then we have:

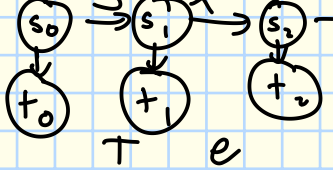
$$P(+u) = 0.8q(1-q) + 0.3q^2 + 0.6(1-q)^2 + 0.7(1-q)q$$
$$P(+u) = -0.6q^2 + 0.3q + 0.6$$

To optimize this expression, we take the first derivative and set it to zero. Since it is a quadratic with negative leading coefficient, this point would be a maximum.

**Markov Blanket:** each node is indep to all nodes given its Markov Blanket: → Parents, children, children's other parents

## HMMS

hidden(S); observed(T) → **Viterbi: Shortest path**




↳ most likely sequence $S_0, S_1, S_2, \ldots$ to produce observe
↳ if (−log(P)), go with smallest probs: like transition
↳ if (log(P)), we could maximize our states

Bayes classifiers

$\text{class} = \arg\max_y P(y) \prod_{i} P(\hat{x}_i, y)$ (new terms)

$T_1 \perp\!\!\!\perp S_0 | S_1$

$P(T_1|S_1) = P(T_1|S_0,S_1)$

Belief distribution

$B(S_n) = P(S_n | t_0, t_1, \ldots t_n)$

$B'(S_n) = P(S_n | t_0, t_1, \ldots t_{n-1})$

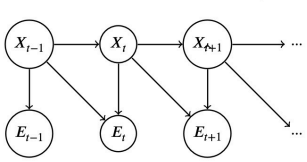
Forward:

$B'(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i)B(w_i)$

$B(W_{i+1}) \propto P(t_{i+1}|W_{i+1})\,B'(W_{i+1})$

$B(W_{i+1}) \propto P(t_{i+1}|W_{i+1}) \sum_{w_i} P(W_{i+1}|w_i)B(w_i)$

(c) [4 pts] After switching to a low-cost sensor, the observations are noisier. One way to address the increased uncertainty in the observations is to condition the observation on more than one state. Complete the forward algorithm updates for an HMM in which the observation $E_t$ is conditioned on both the current state $X_t$ and the previous state $X_{t-1}$.




$$P(x_t|e_{1:t}) = \sum_{x_{t-1}} P(e_t|x_{t-1})P(x_t|x_{t-1})P(x_{t-1}|e_{1:t-1})$$

Expected utilities:

(M)ax (E)xpected (U)tility = Max-util given choice

(V)alue of (P)erfect (I)nfo.

$VPI(E', e) = MEU(e, E') - MEU(e)$ ~ can be negative

Bellman eq.

$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$

$U^*(s) = \max_a Q^*(s,a)$

R = reward

$Q^*$ = Optimal Q state.

$U^*$ = optimal value @ S.

Value Iteration ~ DP alg. to compute MDP until converge

1) $\forall s \in S$, set $U_0(s) = 0$

2) Repeat til converge. $O(S^2 \cdot A)$

→ if $\gamma = 1$, doesn't converge. if $\gamma < 1$ converge

$\forall s \in S, U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$

Policy Iteration ~ More efficient method to converge to opt. policy

1) def initial policy

2) use policy eval. for curr. policy $O(S^3)$

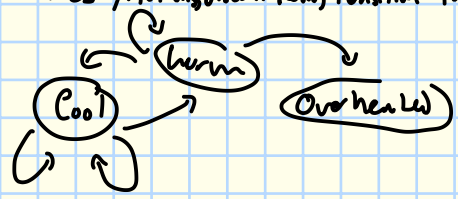
3) use policy improvement

$U^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma U^\pi(s')]$

$\pi_{i+1}(s) = \text{argmax}_{a'} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^{\pi_i}(s')]$

- MDP

States, Actions, discount factor, transition function, reward funct




transition funct.

$T(s,a,s') = P(s' \mid s,a)$ = prob of going to $s'$ state, given action, new state

reward

$R(s,a,s') = \text{reward}$

$\gamma, \gamma^2, \gamma^3 \Rightarrow$ future states matter less: $\frac{R_{max}}{1-\gamma}$

$T(s,a,s') = P(s' \mid s,a)$

Policy $\pi: S \to A$, for each state → action

State optimal value $U^*(s)$: Best expected Utility of optimal agent @ s

Optimal Q-value $Q^*(s,a)$: expected utility of optimal agent @ s taking a

- State space S
- actions A
- Transition function $T(s,a,s')$: Probability of taking action a @ s resulting in $s'$
- Reward funct $R(s,a,s')$: pos/neg reward for each state
- terminal state $C(s)$
- star + state $S_0$

ML: learning parameters given data.

Decision trees ~ nonlinear filter classifier but can overfit

Entropy: uncertainty answer variable; $P_{zero} = \uparrow$ uncertainty

→ $H(v) = -\sum_k P(v_k) \log(P_k)$

Info gain: info gained by splitting on that feature

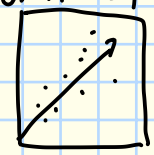
→ $Gain(type) = B(\frac{p}{p+n}) - \sum_{k=1}^{v} \frac{p_k+n_k}{p+n} B(\frac{p_k}{p_k+n_k})$

↳ entropy of binary var

$B(q) = -q \log_2 q - (1-q) \log(1-q)$

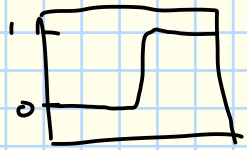
Linear Regression ~ loss of our funct. square diff. $(y - h(x))^2$

Least squares: $\hat{w} = (x^T x)^{-1} x^T y$



Log regression ~ classifier for categorical vars.

→ $h_w(x) = \frac{1}{1 + e^{-wx}}$




→ gradients

p(class | evidence)

Perception:

Linear classifier: classification using comb. of features

activation$_w(x) = h_w(x) = w \cdot f(x)$

classify(x) → $\begin{cases} + & w \cdot f(x) > 0 \\ - & w \cdot f(x) < 0 \end{cases}$

Decision bound: $w^T f(x) = 0$

perception alg:

1) $w = 0$

2) classify y & compare to true label $y^*$

if $y = y^*$ do this

if $y \neq y^*$, update $w \leftarrow w + y^* f(w)$

Naive Bayes ~ Model features as Bayes, assume each feature indep from eachother

↳ prediction(F) = $\text{argmax}_{y_i} P(y = y_i) \prod_j P(F_j = f_j \mid Y = y_i)$, normalize

$F_j$ = features, $Y_i$ = prediction, features are cond. indep (same pred down indep)

Maximum Likelihood Estimation (MLE): given i.i.d., method to learn probability, count

→ Likelihood: $L(\theta) = \prod_{i=1} P_\theta(x_i)$ take gradient

$\frac{d}{d\theta} L(\theta) = 0$ to get max, log likelihood

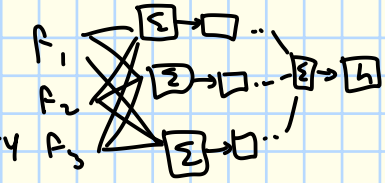
$\log L(\theta)$

→ Laplace Smoothing: mitigate overfitting, assume seen k extra each of each outcome

$P_{Lap,k}(x) = \frac{Count(x) + k}{N + k|X|}$ $|X|$ = set of poss values

N = total

# Neural Networks

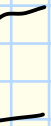


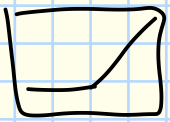

Multilayered perception: maps data to higher dimention then classify

Universal func approximator: two layer NN w/sufficient nevons can approx
any cont. func can use indicator function of sgn threshhold, softmax fnct to classify

log likeley hood: $\log L(w) = \hat{\prod}_{i=1} \log P(y_i \mid f(x_i), w)$

Activation functs

Sigmoid: $\sigma(x) = \left\{ \frac{1}{1+e^{x}} \right.$ 

ReLU: $f(x) = \begin{cases} 0 \text{ if } x<0 \\ x \text{ if } x \geq 0 \end{cases}$ 

Back prop. determine gradient of output w/ respect to each of input

1) Forward pass: Compute Values through Computation graphs
2) Backwards Pass: Compute gradients taking advantage of chain rule

## Reinforcement Learning

Episode: Collection of Samples which are $(s, A, s', r)$ tuples

Model Based Learning: estimates transitions, reward functions w/ samples before using policy /value ite

→ Count times arrived in State and normalized counts, Law of large Numbers will converge on opti
→ Model-free learning: estimate Q values directly w/o constructing model
↳ Direct Evaluation: fix policy $\pi$ and have agent experience, Can compute by averages
↳ temporal diff learning: Learning from every experience

$$\text{Sample} = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$
$$V_k^{\pi}(s) \leftarrow (1-\alpha) V_{k-1}^{\pi}(s) + \alpha \cdot \text{sample}$$

→ Active Reinforcement learning: learning agent agent can use feedback to iteratively update policy
↳ Q-learning: Bypass model by directly learning Q values

Q-Value Iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q(s', a')\right]$$

Q value Samples:

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

Feature Based Representation: allows model to generalize learning experiences, store as linear func

Weight update: $w_i \leftarrow w_i - \alpha \cdot f_i(s,a) \cdot \frac{dQ_w}{dw_i}$ } negative for grad. decent

## Exploitation vs Exploration

$\frac{d(Q_w)}{dw_i} = \left[R(s,a,s') + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$

$\epsilon$-greedy: $0 \leq \epsilon \leq 1$, act randomly w/ prob $\epsilon$, should be lowered over time

Exploration functions:

modification update

$$Q(s,a) = (1-\alpha) Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} R(s', a')\right]$$
$$R(s,a) = Q(s,a) + \frac{k}{N(s,a)}$$

← predetermined
← # of times Q is visited

Regret: metric to measure model, differentiate between total reward for optimal and reward in our model